ΕΘΝΙΚΟ ΜΕΤΣΟΒΕΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

Ανάπτυξη εκπαιδευτικών μικροεφαρμογών φυσικής σε Java

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Κούγγελης Βασίλειος

Τομέας Φυσικής
ΣΧΟΛΗ ΕΦΑΡΜΟΣΜΕΝΩΝ ΜΑΘΗΜΑΤΙΚΩΝ ΚΑΙ ΦΥΣΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

Επιβλέπων Καθηγητής: Κωνσταντίνος Αναγνωστόπουλος

4 Νοεμβρίου 2009

ΕΘΝΙΚΟ ΜΕΤΣΟΒΕΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

Ανάπτυξη εκπαιδευτικών μικροεφαρμογών φυσικής σε Java

Κούγγελης Βασίλειος

Διπλωματική Εργασία που υποβλήθηκε για την απόκτηση του
Προπτυχιακού Διπλώματος στην Εφαρμοσμένη Φυσική στον
Τομέα Φυσικής
ΣΧΟΛΗ ΕΦΑΡΜΟΣΜΕΝΩΝ ΜΑΘΗΜΑΤΙΚΩΝ ΚΑΙ ΦΥΣΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

Εγκρίθηκε από την τριμελή επιτροπή την . . . Οκτωβρίου 2009.

Κ.Αναγνωστόπουλος Αναπληρωτής Καθ. ΕΜΠ	Κ.Φαράκος Αναπληρωτής Καθ. ΕΜΠ	Κ. Παρασκευαΐδης Αναπληρωτής Καθ. ΕΜΠ

Βασίλειος Θ. Κούγγελης
ΔΙΠΛΩΜΑΤΟΥΧΟΣ ΦΥΣΙΚΟΣ
Ειδικότητες: *Οπτοηλεκτρονική και Lasers*
Φυσική των Υλικών

'Never give a sword to a man who can't dance.'

Confucius

ΕΘΝΙΚΟ ΜΕΤΣΟΒΕΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

Σύνοψη

Τομέας Φυσικής
Σχολή Εφαρμοσμένων Μαθηματικών και Φυσικών Επιστημών

του Κούγγελη Βασίλειου

Ανταποκρινόμενοι στις σύγχρονες ανάγκες της Υπολογιστικής Φυσικής μια διεθνής διαπανεπιστημιακή ομάδα επιστημόνων, με συντονιστή τον δρ. Wolfgang Christian του Davidson college, Νέα Καρολίνα ΗΠΑ, ανέπτυξε ένα σύνολο βιβλιοθηκών λογισμικού ανοιχτού κώδικα σε γλώσσα JAVA με το όνομα Open Source Physics, προσφέροντας με αυτό τον τρόπο μια ισχυρή και ταυτόχρονα εξαιρετικά εύχρηστη πλατφόρμα ανάπτυξης εκπαιδευτικών και ερευνητικών εφαρμογών. Χρησιμοποιώντας προγραμματιστικά εργαλεία και κλάσεις από το συγκεκριμένο project αναπτύσονται δυο εφαρμογές εξομοίωσης (simulations) φυσικών φαινομένων, συγκεκριμένα η κίνηση φορτισμένου σωματίου στον τρισδιάστατο χώρο υπό την επίδραση ηλεκτρικού και μαγνητικού πεδίου (Νόμος της δύναμης Lorentz) και η καταγραφή του ηλεκτρικού πεδίου και του δυναμικού μεγάλου αριθμού φορτισμένων σωματίων κινούμενων σε δισδιάστατο χώρο. Δίνεται επίσης η ευκαιρία να παρουσιαστούν δύο γνωστές μέθοδοι αριθμητικής επίλυσης διαφορικών εξισώσεων καθώς και τα βασικά χαρακτηριστικά του αντικειμενοστραφούς προγραμματισμού και τα πλεονεκτήματα που αυτός προσφέρει στην ανάπτυξη παραμετροποιήσιμων επιστημονικών εφαρμογών εφοδιασμένων με αλληλεπιδραστικά γραφικά περιβάλλοντα.

Παραπομπές

Η παρούσα διπλωματική δε θα μπορούσε να πραγματοποιηθεί αν δεν υπήρχαν η εκπαιδευτική και οργανωτική υποδομή του Εθνικού Μετσοβείου Πολυτεχνείου (Τομέας Φυσικής, Τμήμα Βιβλιοθήκης, Γραμματεία). Είμαι υπόχρεος στον επιβλέποντα καθηγητή μου Κωνσταντίνο Αναγνωστόπουλο για την επιστημονική καθοδήγηση, την υπομονή και την αμέριστη προθυμία του να με βοηθήσει σε οποιοδήποτε ζήτημα προέκυψε κατά τη διάρκεια της ανάπτυξης/συγγραφής της εργασίας. Επίσης οφείλω να ευχαριστήσω την οικογένειά μου για την ατελείωτη υπομονή, υλική και ηθική υποστήριξη που μου προσέφερε. Τέλος πρέπει να αναφέρω τη συμφοιτήτριά μου Ζωγλωπίτη Βασιλική για το σημαντικό ρόλο που διαδραμάτισε στην ολοκλήρωση της διπλωματικής.

Περιεχόμενα

Στην οικογένειά μου

Πρόλογος

Καθώς οι σπουδαστές μεταβαίνουν από το σχολείο στην ανώτατη εκπαιδευτική βαθμίδα, μια από τις αλλαγές που παρατηρούν εύκολα στην εκπαιδευτική διαδικασία είναι ότι οι ίδιοι αρχίζουν να συμμετέχουν ενεργά στην παραγωγή και μετάδοση της γνώσης, κάτι που στο σχολείο ήταν κυρίως ευθύνη των εκπαιδευτών τους.

Στη Φυσική, η οποία είναι μια κατ' εξοχήν πειραματική επιστήμη, η συμμετοχή των σπουδαστών στην εκπαιδευτική διαδικασία γίνεται συν τοις άλλοις με την εργαστηριακή εξάσκηση και τις εξομοιώσεις σε υπολογιστή (computer simulations). Με το πείραμα και την εξομοίωση οι σπουδαστές εξοικειώνονται με τις διατάξεις που χρησιμοποίησαν οι μεγάλοι *φυσικοί* στο παρελθόν για να ανακαλύψουν τη *φυσική* που διδάσκεται σήμερα και αποκτούν ζωντανές παραστάσεις των φυσικών φαινομένων.

Η διπλωματική αυτή αναφέρεται σε εξομοιώσεις φυσικής σε υπολογιστή γραμμένες σε γλώσσα προγραμματισμού Java και πιο συγκεκριμένα προτείνει έναν τρόπο κατασκευής τέτοιων εξομοιώσεων αρκετά εύληπτο ώστε ο αναγνώστης μετά το πέρας των πρώτων κεφαλαίων να είναι σε θέση δημιουργήσει τη δικιά του εξομοίωση ή να χρησιμοποιήσει μια από αυτές που ακολουθούν. Ως εκ τούτου το κείμενο αυτό αφορά είτε καθηγητές που θα ήθελαν να προσφέρουν ένα ακόμη μέσο "αφομοίωσης της φυσικής" στους φοιτητές τους, χτίζοντας τις δικές τους εξομοιώσεις με βάση το περιεχόμενο του μαθήματος είτε σπουδαστές προχωρημένου επιπέδου που κάνουν τα πρώτα τους βήματα στο συναρπαστικό κόσμο της μοντελοποίησης. Τέλος δεν πρέπει να ξεχνάμε ότι οι συγκεκριμένες εφαρμογές βασίζονται σε Java, μια σύγχρονη και ευέλικτη γλώσσα προγραμματισμού με αναρίθμητες δυνατότητες. Διαθέτοντας ένα επαρκές προγραμματιστικό και υπολογιστικό background οι ενδιαφερόμενοι θα μπορούσαν να γράψουν ισχυρά εργαλεία υπολογιστικής φυσικής αμιγώς ερευνητικού χαρακτήρα.

Το κείμενο λοιπόν που ακολουθεί είναι δομημένο έτσι ώστε να εισάγει τον αναγνώστη όσο το δυνατόν πιο ομαλά στο χώρο των Java-based computer simulations in physics, ο οποίος κερδίζει όλο και περισσότερους οπαδούς τα τελευταία χρόνια.

Το πρώτο κεφάλαιο έχει εισαγωγικό χαρακτήρα και αναφέρεται γενικά στα θετικά των εξομοιώσεων σε υπολογιστή, στη Java και στο Open Source Physics project (OSP project). Το project αυτό είναι η προσπάθεια μιας διεθνούς ομάδας φυσικών και προγραμματιστών να κατασκευάσουν μια βιβλιοθήκη κλάσεων ανοιχτού κώδικα (οι κλάσεις είναι το δομικά συστατικά των εφαρμογών γραμμένων σε Java) εξειδικευμένων στο χτίσιμο εξομοιώσεων φυσικής.

Το δεύτερο κεφάλαιο παραθέτει τις ουσιώδεις προγραμματιστικές γνώσεις που πρέπει να έχει κανείς για να ξεκινήσει τη σύνθεση εφαρμογών φυσικής γραμμένων σε Java μέσα από το γραφικό περιβάλλον του EJS. Το EJS (Easy Java Simulations) αναλύεται στο τρίτο κεφάλαιο και αποτελεί ένα εξειδικευμένο προγραμματιστικό εργαλείο ανάπτυξης εφαρμογών το οποίο στηρίζεται στις βιβλιοθήκες Java του OSP διευκολύνοντας σε μεγάλο βαθμό το έργο κάποιου που θέλει να εστιάσει στο φυσικό μοντέλο της εξομοίωσης και να μην πελαγοδρομήσει στον κώδικα για την εμφάνιση και τον έλεγχο της εφαρμογής που κατασκευάζει.

Τέλος στο τέταρτο κεφάλαιο παρουσιάζονται τα δύο φυσικά συστήματα που μοντελοποιούνται στο πλαίσιο της παρούσας εργασίας και η εσωτερική δομή των δυο εξομοιώσεων που αναπτύχθηκαν για αυτό το σκοπό στο EJS. Και οι δύο εξομοιώσεις αφορούν φαινόμενα από τον Ηλεκτρομαγνητισμό:

- η κίνηση ενός φορτισμένου σωματιδίου στον τρισδιάστατο χώρο υπό την συνδυασμένη επίδραση ηλεκτρικού και μαγνητικού πεδίου όπως περιγράφεται από το νόμο της δύναμης *Lorentz*
- η ηλεκτροστατική αλληλεπίδραση ενός τυχαίου αριθμού φορτισμένων σωματίων που μπορούν να κινούνται ελεύθερα σε δισδιάστατο χώρο και η απεικόνιση σε πραγματικό χρόνο του δυναμικού και του ηλεκτρικού πεδίου που δημιουργούν.

Κεφάλαιο 1

Προσομοιώσεις Φυσικής σε Υπολογιστή

1.1 Ο Τρίτος Δρόμος

Οι πυλώνες πάνω στους οποίους στηρίχτηκε και εξακολουθεί να στηρίζεται και να αναπτύσσεται η επιστήμη της Φυσικής είναι δύο: η θεωρητική ανάλυση και το πείραμα. Με την έλευση όμως και εξάπλωση της χρήσης των ηλεκτρονικών υπολογιστών [1] προστέθηκε ακόμα ένας, αυτός της υπολογιστικής ανάλυσης. Προκειμένου να θεμελιώσουν τις θεωρίες τους οι επιστήμονες της επονομαζόμενης σύγχρονης φυσικής κατέφευγαν σε όλο και πιο πολύπλοκα μαθηματικά μοντέλα, των οποίων η επίλυση με αναλυτικές μεθόδους ήταν αδύνατη. Οι ηλεκτρονικοί υπολογιστές, και οι μαθηματικοί κλάδοι της στατιστικής και της αριθμητικής ανάλυσης έδωσαν τη δυνατότητα της προσεγγιστικής επίλυσης των εξισώσεων που περιγράφουν τα παραπάνω μοντέλα.

Πιο συγκεκριμένα, ένα φυσικό σύστημα ή ένας φυσικός νόμος περιγράφεται από ένα σύνολο εξισώσεων (συνήθως διαφορικών) που περιγράφουν το συσχετισμό των φυσικών μεγεθών που εμπλέκονται σε αυτό. Όσο περισσότερες και πιο υψηλόβαθμες εξισώσεις απαιτούνται για να περιγράψουν το σύστημα, τόσο δυσκολότερο είναι να βρεθούν αναλυτικές λύσεις για αυτό. Ακόμη και σήμερα παρά τις τεράστιες προόδους στον τομέα των μαθηματικών, ελάχιστα είδη διαφορικών εξισώσεων επιλύονται αλγεβρικά. Μπροστά στο αδιέξοδο αυτό, οι φυσικοί (και οι μαθηματικοί) επιστήμονες είχαν δύο επιλογές: είτε να αναζητήσουν πιο εξεζητημένες μαθηματικές θεωρίες που θα επέτρεπαν την αναλυτική επίλυση είτε να ακολουθήσουν την οδό της αριθμητικής ανάλυσης.

[1] Η πρώτη χρήση ηλεκτρονικού υπολογιστή στη φυσική χρονολογείται το 1945, στο διαβόητο Manhatan Project για την κατασκευή της πρώτης πυρηνικής βόμβας

Η αριθμητική ανάλυση είναι τομέας των μαθηματικών που έγκειται στη μελέτη και κατασκευή *αλγορίθμων* [2] οι οποίοι μετά από έναν πεπερασμένο αριθμό αριθμητικών υπολογισμών *προσεγγίζουν* τις λύσεις ενός μαθηματικού προβλήματος με κάποιο *αποδεκτό* για την εκάστοτε εφαρμογή σφάλμα. Η φιλοσοφία της αριθμητικής προσέγγισης υπήρχε από την εποχή των Βαβυλωνίων μαθηματικών [3] και στις μέρες μας, ως ανεξάρτητη πια μαθηματική επιστήμη, έχει απειράριθμες εφαρμογές: σε όλους τους τομείς της μηχανικής, στη βελτιστοποίηση συστημάτων, στις στοχαστικές διαφορικές εξισώσεις, στα Μαρκοβιανά συστήματα ακόμα και στην κοινωνική ψυχολογία.

1.2 Εργαστήριο vs Προσομοίωση: Ομοιότητες και Διαφορές

Στην παρούσα εργασία βέβαια δε σκοπεύουμε να ασχοληθούμε με δυσεπίλυτες μορφές εξισώσεων: τα φυσικά συστήματα που εξομοιώνονται επιλύονται και αναλυτικά. Στόχος μας είναι να δείξουμε πώς γίνεται η μοντελοποίησή τους στον υπολογιστή προκειμένου να κατασκευαστεί μια προσομοίωση, ένα *εικονικό* πείραμα. Στον παρακάτω πίνακα γίνεται μια προσπάθεια να αντιστοιχίσουμε τα κυριότερα μέρη ενός εργαστηριακού πειράματος με αυτά μιας προσομοίωσης σε υπολογιστή.

- Το μαθηματικό *μοντέλο* που αναπτύχθηκε για να περιγράψει το φυσικό σύστημα και με αλγοριθμικό τρόπο ενσωματώθηκε στο πρόγραμμά μας παίζει το ρόλο του προς μέτρηση *δείγματος* μιας εργαστηριακής διάταξης. Οι αλγόριθμοι, όντας κοινός τόπος μεταξύ των μαθηματικών και της πληροφορικής, είναι τα βασικότερα εργαλεία στη διαδικασία ενσωμάτωσης του φυσικού μοντέλου σε ένα πρόγραμμα όπως επίσης και στην ανάλυση και επεξεργασία των δεδομένων που θα προκύψουν. Καθώς οι αλγόριθμοι είναι σύνολα από διαδοχικές, αρκετές φορές επαναλαμβανόμενες, αριθμητικές και λογικές πράξεις, χρησιμοποιούμε το συντακτικό της εκάστοτε γλώσσας προγραμματισμού για να τις μετατρέψουμε σε προγραμματιστικές εντολές και να τις ενσωματώσουμε στον κώδικα της εφαρμογής μας.

- Η *πειραματική διάταξη*, που μας επιτρέπει να παίρνουμε μετρήσεις από το δείγμα μας σε ένα εργαστήριο αντικαθίσταται από το *πρόγραμμα* της προσομοίωσης, την εφαρμογή δηλαδή που τρέχει ο χρήστης. Εδώ διαφαίνεται ένα σημαντικό, πρακτικό πλεονέκτημα των εξομοιώσεων φυσικής σε υπολογιστές: στο πλαίσιο της εκπαίδευσής τους όλοι οι

[2] Σαν αλγόριθμο ορίζουμε κάθε διαδικασία επίλυσης ενός προβλήματος με ένα πεπερασμένο αριθμό διακριτών βημάτων[1]. Η μετάβαση από το ένα βήμα στο επόμενο μπορεί να είναι ντετερμινιστική ή στοχαστική

[3] Οι Βαβυλώνιοι προκειμένου να προσεγγίσουν την τιμή μιας οποιασδήποτε τετραγωνικής ρίζας ανέπτυξαν έναν αλγόριθμο γνωστό ως η *βαβυλωνιακή μέθοδος*

σπουδαστές φυσικών επιστημών εξασκούνται σε εργαστηριακές διατάξεις, μια διαδικασία που απαιτεί αρκετές ώρες δουλειάς από την πλευρά του διδακτικού προσωπικού καθώς ο αριθμός των διατάξεων (και ο ελεύθερος χρόνος των επιβλεπόντων) είναι συνήθως περιορισμένος. Για "τρέξει" ένα εικονικό πείραμα, το μόνο που χρειάζεται είναι ένας υπολογιστής ή έστω ένα τερματικό, συνθήκη που ικανοποιείται πολύ εύκολα για τους περισσότερους σπουδαστές. Το γεγονός ότι οι εφαρμογές που περιγράφονται στην εργασία αυτή είναι γραμμένες σε Java, μια γλώσσα βελτιστοποιημένη για *διαδικτυακές εφαρμογές* τους προσδίδει ακόμα μεγαλύτερη ευελιξία, επιτρέποντας *on-line* εξομοιώσεις μέσω Διαδικτύου.

- Πριν ξεκινήσει οποιαδήποτε πειραματική μέτρηση γίνεται πάντα *βαθμονόμηση* και *ρύθμιση* των εργαστηριακών οργάνων για να εξασφαλιστεί η ορθότητα της μετρητικής τους ικανότητας. Τέτοιου είδους απαιτήσεις δεν υπάρχουν σε μια προσομοίωση. Παρόλα αυτά ο προγραμματιστής πριν διαθέσει την εφαρμογή του για χρήση (κατά τη διάρκεια του διαβόητου *debugging*) έχει εξετάσει επιμελώς τα αποτελέσματα που παράγονται ώστε να συμβαδίζουν με το θεωρητικό μοντέλο.

- Στις δύο τελευταίες φάσεις η μεθοδολογία του εργαστηριακού με το εικονικό πείραμα τείνουν να συμπέσουν. Αυτό συμβαίνει γιατί στη φάση της παρατήρησης και των *μετρήσεων* η χρήση ηλεκτρονικού υπολογιστή για *καταγραφή* των δεδομένων και περαιτέρω επεξεργασία τους είναι πολύ διαδεδομένη σήμερα. Τα δεδομένα μεταφέρονται σε υπολογιστή είτε "χειρωνακτικά" από τους εκτελούντες το πείραμα, είτε αυτόματα, όταν ο υπολογιστής είναι συνδεδεμένος μέσω κάποιας διεπαφής (interface) με την πειραματική διάταξη[4]. Σε κάθε περίπτωση τα δεδομένα των μετρήσεων καταλήγουν σε ψηφιακή μορφή για να επακολουθήσει το τελευταίο στάδιο, αυτό της *επεξεργασίας* και εξαγωγής συμπερασμάτων για το φυσικό σύστημα που μελετάται.

  Σε μια προσομοίωση δεν υπάρχει *δείγμα* του οποίου μπορούμε να μετρήσουμε κάποιο φυσικό μέγεθος. Αντί για μετρήσεις πραγματοποιούνται σειρές *αριθμητικών υπολογισμών* από τους αλγόριθμους του προγράμματος προκειμένου να προσεγγιστούν με μια ορισμένη ακρίβεια οι προαναφερθείσες φυσικές παράμετροι, αυτή τη φορά όμως μέσω των εξισώσεων που περιγράφουν το σύστημά μας. Είναι σύνηθες φαινόμενο η επεξεργασία και γραφική παρουσίαση των δεδομένων να γίνονται από το ίδιο το πρόγραμμα της προσομοίωσης χάριν ευκολίας. Σε περίπτωση που αυτό δεν είναι δυνατό δίνεται η δυνατότητα στο χρήστη να σώσει τα δεδομένα σε αρχείο κατάλληλου φορμάτ για να τα διαχειριστεί με το πρόγραμμα της προτίμησής του.

[4]Πολύ συχνά ο έλεγχος ολόκληρης της εργαστηριακής διάταξης γίνεται μέσω υπολογιστή.

1.3 Πλεονεκτήματα και Μειονεκτήματα των Προσομοιώσεων

Υπάρχουν και αυτοί που ισχυρίζονται ότι με τη χρήση προσομοιώσεων δεν αναπτύσσονται οι αναλυτικές δεξιότητες που οφείλει να έχει κάθε φυσικός επιστήμονας. Με απλά λόγια: ο επεξεργαστής κάνει τη δουλειά που έπρεπε να κάνει το μυαλό του φυσικού. Το επιχείρημα αυτό καταρρίπτεται αν αναλογιστούμε ότι για να "στηθεί" μια λειτουργική προσομοίωση απαιτείται εις βάθος γνώση της θεωρίας από την πλευρά του φυσικού/προγραμματιστή. Δεν είναι τυχαίο το γεγονός ότι οι "βαρύτερες" προσομοιώσεις χρησιμοποιούνται από τους θεωρητικούς φυσικούς. Όσο για τους εκπαιδευόμενους, οφείλουμε να διευκρινίσουμε εδώ ότι το παρόν κείμενο δεν προωθεί τις προσομοιώσεις σε υπολογιστή σαν υποκατάστατο της πειραματικής μεθόδου. Η φυσική στηρίχτηκε και θα συνεχίσει να στηρίζεται στην πειραματική διαδικασία, το βασικότερο εργαλείο που διαθέτει για την απόδειξη των επιστημονικών υποθέσεών της. Απλά οι προσομοιώσεις έρχονται να καλύψουν σύγχρονες εκπαιδευτικές ανάγκες και να δώσουν επιπλέον δυνατότητες τεκμηρίωσης των φυσικών θεωριών.

Ο ρόλος τους στην έρευνα και τις εφαρμοσμένες επιστήμες είναι σημαντικότατος, προσφέροντας τη δυνατότητα να προσεγγιστούν *υπολογιστικά*, "δύστροπα" φυσικά συστήματα[5] τα οποία θα απαιτούσαν διαφορετικά πολυσύνθετες εργαστηριακές διατάξεις για να μελετηθούν. Βέβαια, όσο περισσότεροι μαθηματικοί υπολογισμοί χρειάζονται για να επιτευχθεί η ζητούμενη ακρίβεια στις τιμές των παραμέτρων του συστήματος τόσο "βαρύτερη" είναι η προσομοίωση, απαιτώντας υπολογιστή (ή ακόμα και συστοιχίες υπολογιστών) μεγάλης επεξεργαστικής ισχύος. Παρόλα αυτά, συνήθως το *κόστος* για το στήσιμο μιας τέτοιας "φάρμας υπολογιστών" είναι υποπολλαπλάσιο αυτού που θα ξοδευόταν αν επιλέγονταν η εργαστηριακή μέθοδος [6] Εξάλλου πολλά θεωρητικά μοντέλα δεν έχουν καν φυσικό αντίστοιχο, καταδεικνύοντας την υπολογιστική οδό ως μοναδική διέξοδο.

Στην εργασία αυτή θα ασχοληθούμε βέβαια με μικρότερης κλίμακας εφαρμογές, προσανατολισμένες κυρίως στην επεξήγηση της φυσικής, οι οποίες κάλλιστα μπορούν να διευκολύνουν την είσοδο εκπαιδευόμενων στο χώρο της υπολογιστικής φυσικής. Η φιλοσοφία της προσομοίωσης είναι *εκμάθηση μέσα από το παράδειγμα*. Μέσω από αλληλεπιδραστικά (interactive) γραφικά περιβάλλοντα ενθαρρύνει τους σπουδαστές να *εξερευνήσουν* μόνοι τους τις ιδέες πίσω από τους φυσικούς νόμους, παίζοντας με τις παραμέτρους ενός εικονικού πειράματος χωρίς να τίθεται θέμα σοβαρών συνεπειών για τους ίδιους ή τον εξοπλισμό του

[5] Τα περισσότερα συστήματα στη φύση θεωρούνται *μη γραμμικά*, γιατί περιγράφονται από μη γραμμικές εξισώσεις τις οποίες δε μπορούμε να λύσουμε με την *αρχή της επαλληλίας*. Το πιο γνωστό μη γραμμικό φυσικό σύστημα είναι ο *καιρός* και ένα παράδειγμα μη γραμμικών διαφορικών εξισώσεων είναι οι Navier-Stokes της ρευστοδυναμικής.

[6] Είναι χαρακτηριστικό το παράδειγμα της BMW το ερευνητικό τμήμα της οποίας προκειμένου να αναπτύξει το επόμενο αγωνιστικό μοντέλο Φόρμουλας1 προτίμησε αντί της κατασκευής μιας νέας αεροσήραγγας, να επενδύσει σε μια επιπλέον φάρμα υπολογιστών στην οποία μοντελοποίησε και προσομοίωσε τα αεροδυναμικά τεστ.

εργαστηρίου. Επιπλέον παρέχοντας τις βασικές γνώσεις του αντικειμενοστραφούς προγραμματισμού για μια ευρέως διαδεδομένη γλώσσα όπως η Java, προτρέπει στην εφαρμογή τους για τη δημιουργία προσομοιώσεων "κομμένων και ραμμένων" στις ανάγκες του κάθε χρήστη. Το έργο αυτό διευκολύνεται από τις βιβλιοθήκες εξειδικευμένων open-source κλάσεων που διαθέτει το OSP/CSM project για το οποίο γίνεται λόγος στην επόμενη παράγραφο.

1.4 Νέα Εργαλεία

Σε μια προσπάθεια να διαδώσει τη χρήση προσομοιώσεων φυσικής για εκπαιδευτική χρήση, ο Wolfgang Christian με μερική χρηματοδότηση του NSF[7] και την υποστήριξη του Davidson College της Βόρειας Καρολίνας των ΗΠΑ ανέπτυξε ένα εκτενές σύνολο κλάσεων της Java υπό την άδεια χρήσης GPL[link][8] του λογισμικού ανοιχτού κώδικα GNU[9]. Το επίτευγμά του έχει το όνομα project **O**pen **S**ource **P**hysics[link]. Το λογισμικό που παρήγαγε, όντας ανοιχτού κώδικα μπορεί να χρησιμοποιηθεί ή να τροποποιηθεί κατά βούληση από οποιοδήποτε χρήστη και οι εφαρμογές που θα παραχθούν να εκδοθούν δημόσια, με την προϋπόθεση να φέρουν την άδεια χρήσης GPL. Οι δύο εφαρμογές που αναπτύχθηκαν για τη διπλωματική αυτή βασίστηκαν σε κλάσεις και προγραμματιστικά εργαλεία από το OSP project, ως εκ τούτου υπόκεινται στη GPL η οποία εμπεριέχεται στον πηγαίο τους κώδικα.

1.4.1 Φυσική Ανοιχτού Κώδικα

Η ανάπτυξη εφαρμογών για χρήση στη φυσική δεν είναι καινούριο φαινόμενο. Έως σήμερα όμως οι φυσικοί προτιμούσαν "γρήγορες" διαδικαστικές (procedural) γλώσσες όπως η FORTRAN, η True Basic και φυσικά η C/C++. Δε θα εμπλακούμε εδώ στα πλεονεκτήματα και μειονεκτήματα της κάθε γλώσσας (αυτό θα γίνει στο επόμενο κεφάλαιο, αφιερωμένο στη Java). Αρκεί να αναφέρουμε απλά ότι θέλοντας να εμπλουτίσουν τις προσομοιώσεις τους με εύχρηστα γραφικά περιβάλλοντα και εύληπτες απεικονίσεις των αποτελεσμάτων οι επιστήμονες άρχισαν να αναζητούν πιο σύγχρονες λύσεις. Το OSP project στα πρώτα του βήματα δεν ήταν παρά μια προσπάθεια να αναπτυχθούν σε True Basic γραφικά περιβάλλοντα που θα "έντυναν" υπάρχουσες προσομοιώσεις γραμμένες σε άλλες γλώσσες. Καθώς το project εξελισσόταν, ο δημιουργός του συνειδητοποίησε ότι η εκπαιδευτική απόδοση μιας προσομοίωσης πολλαπλασιάζεται αν ο εκπαιδευόμενος συμμετάσχει στην ανάπτυξή της. Ο προγραμματισμός σε ανοιχτό κώδικα κρίθηκε ιδανικός για το ρόλο αυτό. Η σύλληψη ολοκληρώθηκε με την υιοθέτηση μιας σύγχρονης και καθαρά αντικειμενοστραφούς γλώσσας η

[7] National Science Foundation
[8] General Public Licence
[9] Gnu is Not Unix!

οποία παρείχε επαρκές συντακτικό τόσο για τη μοντελοποίηση όσο και για το γραφικό περιβάλλον των προσομοιώσεων. Η *Java* παρείχε επιπλέον εγγενή υποστήριξη για διαδικτυακή χρήση και ήταν platform-independent, δηλαδή οι εφαρμογές της "τρέχουν" σε οποιοδήποτε λειτουργικό σύστημα (Windows, Unix, Mac κτλ).

Έτσι ο αρχικός σκοπός του OSP project επαναπροσδιορίστηκε και συνοψίζεται στα εξής σημεία:

- Να χρησιμοποιήσει τις προσομοιώσεις σε υπολογιστή σαν εξελιγμένο παιδαγωγικό μέσο, ένα *εικονικό εργαστήριο* που δίνει τη δυνατότητα στους εκπαιδευόμενους να ανακαλύψουν τη "φυσική" με τον τρόπο που οι "φυσικοί" την ανακάλυψαν.
- Να τους εντάξει στη χρήση μεθόδων αριθμητικής ανάλυσης και τη μοντελοποίηση φυσικών συστημάτων.
- Να προάγει τις προγραμματιστικές δεξιότητες προτρέποντας τους σπουδαστές να τροποποιήσουν υπάρχουσες προσομοιώσεις, να μεταγλωττίσουν κώδικα και να εμπλακούν στις διαδικασίες του app-testing και του debugging.
- Να τους προσφέρει εξειδικευμένα προγραμματιστικά εργαλεία για να αντιμετωπίσουν δύσκολα φυσικά προβλήματα ή για να αναπτύξουν γραφικές απεικονίσεις σε ήδη υπάρχοντα.
- Να τους δώσει τη δυνατότητα να ανταλλάξουν μέσω διαδικτύου ιδέες, γνώσεις ακόμα και κώδικα ή ολόκληρες εφαρμογές, με άτομα με τα οποία μοιράζονται τα ίδια ενδιαφέροντα.

Για να πετύχουν τα παραπάνω αναπτύχθηκε και διαρκώς εξελίσσεται μια συλλογή από κλάσεις, διαπροσωπείες και βοηθητικές εφαρμογές σε γλώσσα Java[10], οργανωμένες σε βιβλιοθήκες και πακέτα και αποθηκευμένα στο web server του Davidson College στη διάθεση οποιουδήποτε χρήστη του Διαδικτύου.

Τα κυριότερα πακέτα κλάσεων (packages) είναι:

- Το πακέτο **controls** (`org.opensourcephysics.controls`[11].) το οποίο υλοποιεί γραφικά στοιχεία ελέγχου των προσομοιώσεων και δίνει τη δυνατότητα οργάνωσης μέσω .xml αρχείων.
- Το πακέτο **display** (`org.opensourcephysics.display`) που δίνει τη δυνατότητα σχεδιασμού δισδιάστατων αντικειμένων και γραφικών παραστάσεων στην κλάση `DrawingPanel` μέσω της διαπροσωπείας `Drawable`.

[10] Το OSP project βασίζεται στην έκδοση 1.4 ή νεότερη.
[11] Προφανώς δεν πρόκειται για url αλλά για όνομα java package

- Το πακέτο **display2d** (`org.opensourcephysics.display2d`) που εκτυπώνει στην οθόνη δισδιάστατα δεδομένα υπό μορφή ισοδυναμικών γραμμών ή επιφανειών.
- Το πακέτο **display3d** (`org.opensourcephysics.display3d`) που ορίζει και ενσωματώνει ένα API[12] για το σχεδιασμό τρισδιάστατων αντικειμένων.
- Το πακέτο **EJS** (`org.opensourcephysics.ejs`) με τις κλάσεις του οποίου ο χρήστης μπορεί να δημιουργήσει ολοκληρωμένες προσομοιώσεις και γραφικά περιβάλλοντα για τον έλεγχό τους (GUIs[13]).
- Το πακέτο **numerics** (`org.opensourcephysics.numerics`) το οποίο περιλαμβάνει εργαλεία αριθμητικής ανάλυσης όπως αλγόριθμους για την επίλυση διαφορικών εξισώσεων.
- Το πακέτο **tools** (`org.opensourcephysics.tools`) το οποίο περιέχει ολόκληρα προγράμματα όπως ο Launcher-Builder για τη διαχείριση των υπαρχόντων προσομοιώσεων.

Οι βιβλιοθήκες κώδικα του OSP έχουν αξιοποιηθεί από διάφορες ομάδες επιστημόνων με αποτέλεσμα να προκύψουν θυγατρικά projects που επεκτείνουν ή εμπλουτίζουν τις δυνατότητές του και τα οποία εκδίδονται παράλληλα με αυτό:

- Το βιβλίο *Open Source Physics: A user's guide with examples*[2] του **Wolfgang Christian** πρακτικά αναλύει τα περιεχόμενα του OSP.
- Το βιβλίο *An Introduction to Computer Simulation Methods*[3] από την ομάδα των **Harvey Gould, Jan Tobochnik** και **Wolfgang Christian**.
- Το βιβλίο *Statistical and Thermal Physics* των **Harvey Gould** και **Jan Tobochnik**.
- Το εργαλείο ανάπτυξης εφαρμογών *EJS*[4] από τον **Francisco Esquembre**[14].
- Το πρόγραμμα ανάλυσης video και καταγραφής πειραματικών δεδομένων *Tracker*[5] του **Doug Brown**.

1.4.2 Μια εικόνα, χίλιες λέξεις.

Όποιος έχει προσπαθήσει να αναπτύξει μια ολοκληρωμένη εφαρμογή γνωρίζει πόσο χρήσιμο είναι ένα γραφικό περιβάλλον για τον έλεγχό της αλλά και πόσο επίπονη είναι η φάση όπου

[12] Application Programming Interface
[13] Graphical User Interfaces
[14] Το EJS είναι ένα Integrated Development Environment (IDE)

γράφεται ο αντίστοιχος κώδικας. Ακόμα και σε γλώσσες που έχουν εξειδικευμένο συντακτικό για τη δημιουργία παραθύρων, διακοπτών, μενού, πεδίων συμπλήρωσης τιμών κτλ, η διαδικασία σύνδεσης του κώδικα της γραφικής αναπαράστασης με τον κώδικα του υπολογιστικού πυρήνα της εφαρμογής μπορεί να αποβεί εξαιρετικά χρονοβόρα. Το γεγονός αυτό θα απέτρεπε έναν μαθητή ή έναν σπουδαστή προπτυχιακού επιπέδου να προγραμματίσει μια δικιά του προσομοίωση άρα καθιστά τους στόχους του OSP project αδύνατους. Γι΄ αυτό οι φυσικοί/προγραμματιστές που μετέχουν στο project εκμεταλλεύτηκαν στο έπακρο την αντικειμενοστρέφεια της Java ώστε να προσφέρουν πηγαίο κώδικα και προγραμματιστικά εργαλεία που επιτρέπουν στους εκπαιδευόμενους να εστιάζουν στον κώδικα που μοντελοποιεί τη φυσική της προσομοίωσης, "συναρμολογώντας" με ευκολία το γραφικό της περιβάλλον.

Η υποστήριξη ενός εύχρηστου περιβάλλοντος ελέγχου από την προσομοίωση και η δυνατότητα να παράγει οπτικές αναπαραστάσεις των δεδομένων που προκύπτουν κρίθηκε "εκ των ουκ άνευ" για τις εφαρμογές που αναπτύσσονται στο πλαίσιο του OSP. Σε όλους τους γνωστικούς τομείς που εμπλέκονται μαθηματικά, οι γραφικές παραστάσεις μεγεθών ή συναρτήσεων αποκαλύπτουν με αμεσότητα ιδιότητες και μηχανισμούς που είναι δύσκολο να παρατηρήσει κανείς μέσα από τους τύπους και τα αριθμητικά δεδομένα. Για τους ίδιους λόγους, σε μια προσομοίωση φυσικής τα δεδομένα των αλγορίθμων, τα αποτελέσματα της επεξεργασίας τους και το ίδιο το εικονικό σύστημα που μελετάται οφείλουν να απεικονίζονται γραφικά. Πέρα από τις γραφικές παραστάσεις μια προσομοίωση οφείλει να είναι διαδραστική (interactive), να επιτρέπει δηλαδή στο χρήστη να επεμβαίνει στην εξέλιξη του εικονικού πειράματος (ακόμα και "on-the-fly", καθώς η προσομοίωση "τρέχει") τροποποιώντας τιμές παραμέτρων και γενικά μεταβάλλοντας τις συνθήκες μέσω γραφικών στοιχείων ελέγχου όπως κουμπιά, διακόπτες, sliders ή ακόμα και τον κέρσορα του ποντικιού.

Υπενθυμίζουμε ότι μέρος της εκπαιδευτικής φιλοσοφίας του OSP είναι η συμμετοχή στην ανάπτυξη της προσομοίωσης. Το στάδιο του debugging, της εύρεσης δηλαδή των λαθών στον πηγαίο κώδικα που εμποδίζουν την εφαρμογή να λειτουργήσει φυσιολογικά διευκολύνεται σημαντικά για το σπουδαστή, αν τα δεδομένα αναπαρίστανται γραφικά. Είναι πιο εύκολο να αντιληφθείς το λάθος ενός αλγορίθμου αν τοποθετήσεις τα αποτελέσματά του σε μια γραφική παράσταση από το να παρατηρείς στήλες αριθμητικών τιμών[15]. Για τους παραπάνω λόγους, ένα από τα βασικά μελήματα των προγραμματιστών του OSP ήταν η ανάπτυξη ενός πλούσιου συντακτικού κλάσεων και διαπροσωπειών της Java για "στήσιμο" κάθε είδους γραφικής αναπαράστασης και διαδραστικών στοιχείων ελέγχου. Η Java διαθέτει επαρκείς βιβλιοθήκες για τέτοιου είδους χρήση. Ο Wolfgang Christian όμως τις επέκτεινε και τις εξειδίκευσε για εφαρμογές φυσικής και μαθηματικών. Επιπλέον αναπτύχθηκαν ενδεικτικά αρκετές ολοκληρωμένες προσομοιώσεις για διάφορους τομείς της φυσικής προς άμεση χρήση καθώς και high-level εφαρμογές που προορίζονται για τη δημιουργία, διάθεση και οργάνωση όλων των υπαρχόντων προσομοιώσεων που γράφτηκαν με τη βοήθεια του OSP. Μια από

[15] Αν και τελικά, στα δύσκολα, όλοι καταφεύγουμε στην ταπεινή κονσόλα

αυτές, το *EJS*[16] αξιοποιήθηκε για την ανάπτυξη των δυο προσομοιώσεων Ηλεκτρομαγνητισμού που περιλαμβάνει η διπλωματική αυτή. Ο τρόπος χρήσης του EJS αναλύεται διεξοδικά στο τρίτο κεφάλαιο.

Όμως ένα εντυπωσιακό γραφικό περιβάλλον μπορεί να έχει και μειονεκτήματα. Για να απεικονιστούν το περιβάλλον ελέγχου της προσομοίωσης και οι γραφικές αναπαραστάσεις των διαφόρων φυσικών παραμέτρων καταναλώνονται πόροι του υπολογιστή όπως η μνήμη και η επεξεργαστική ισχύς. Σε μια "βαριά" ερευνητική προσομοίωση όπου οι αλγόριθμοι πρέπει να τρέξουν εκατοντάδες χιλιάδες φορές για να προσεγγιστούν ικανοποιητικά οι ζητούμενες λύσεις, οι παραπάνω πόροι καθίστανται πολύτιμοι και επηρεάζουν σε *εκθετικό βαθμό* το χρόνο ολοκλήρωσης της συνολικής διεργασίας. Στις περιπτώσεις αυτές οι ερευνητές αναγκάζονται να ολοκληρώσουν πρώτα τους υπολογισμούς και να απεικονίσουν μετά την εξέλιξη του φαινομένου. Ακόμα και στο σχεδιασμό μικρών εκπαιδευτικών προσομοιώσεων πρέπει η χρήση γραφικών διευκολύνσεων να γίνεται με σύνεση καθώς δεν είναι δεδομένο ότι το υπολογιστικό κέντρο όπου γίνεται το μάθημα ή οι συμμετέχοντες εκπαιδευόμενοι διαθέτουν σύγχρονους και ισχυρούς υπολογιστές.

[16] Easy Java Simulations

Κεφάλαιο 2

Η γλώσσα JAVA

2.1 Μια δημοφιλής γλώσσα.

Όλοι έχουμε ακούσει κάπου ή και χρησιμοποιήσει τη Java σήμερα. Είτε καθώς σερφάρουμε στο Διαδίκτυο συναντώντας ιστοσελίδες με περίτεχνα μενού και λειτουργίες αναζήτησης· είτε στα κινητά μας τηλέφωνα όπου εγκαθιστούμε και χειριζόμαστε μικροεφαρμογές και παιχνίδια· είτε τέλος έχουμε στον υπολογιστή μας κάποια δημοφιλή εφαρμογή που βασίζεται σε Java όπως η σουίτα επεξεργασίας κειμένου *OpenOffice.org*, ο bittorent client *Azureus (Vuze)* ή το ολοκληρωμένο περιβάλλον ανάπτυξης εφαρμογών *Eclipse*· με τον έναν ή με τον άλλο τρόπο έχουμε έρθει σε επαφή με τη δημοφιλή αυτή γλώσσα προγραμματισμού.

Η μεγάλη εξάπλωση που γνωρίζει σήμερα η Java οφείλεται σε τρία σημαντικά χαρακτηριστικά που κατέχει: τη *φορητότητα*, την *ταχύτητά* της και την *ασφάλεια*. Η ταχύτητα είναι ο μοναδικός τομέας στον οποίο υστερούσε σε σχέση με την με την έταιρη διάσημη γλώσσα προγραμματισμού, τη C/C++. Καθώς η Java είναι interpreted γλώσσα, ο κώδικάς της δεν εκτελείται αμέσως μόλις μεταγλωττιστεί από τον compiler αλλά μετατρέπεται σε bytecode τον οποίο αναλαμβάνει να εκτελέσει η *JVM*[1], ένα μίνι λειτουργικό σύστημα που πρέπει να είναι εγκατεστημένο σε κάθε σύστημα (υπολογιστή, κινητό, palmtop) που θέλει να τρέξει μια εφαρμογή γραμμένη σε Java. Η JVM είναι ο εκτελεστικός πυρήνας της γλώσσας και σε αυτόν οφείλονται τα άλλα δυο πλεονεκτήματα που αναφέραμε πιο πάνω:η μεγάλη φορητότητα γιατί ο κωδικάς της εκτελείται από τη JVM και **όχι** από το εκάστοτε λειτουργικό σύστημα στο οποίο είναι εγκατεστημένη (Windows, Linux, Macintosh) κτλ. Επίσης, δεδομένου ότι το μέλλον της πληροφορικής διαφαίνεται όλο και πιο διαδικτυακό, η Java προσφέρει εγγενή υποστήριξη για χρήση εφαρμογών μέσω διαδικτύου, εγκατεστημένων σε Web servers

[1] Java Virtual Machine

ή ενσωματωμένων στον ίδιο τον πλοηγό (Web browser embedded applets)·όσο για την ασφάλεια, η JVM εκτελεί τον bytecode αφού πρώτα ελέγξει για την ορθότητά του (βάλε κι άλλα) αποφεύγοντας σφάλματα στα πιο κάτω στρώματα.

2.1.1 Η Java kai oi 'alles.

Θα άξιζε να αναφέρουμε εδώ λίγα για το παρελθόν της Java και τη σχέση της με τις άλλες γνωστές γλώσσες προγραμματισμού. Οι δημιουργοί της Java είναι οι *Bill Joy* και *James Gosling*, στελέχη της εταιρείας *SUN Microsystems*, οι οποίοι τη δεκαετία του '90 είχαν ξεκίνησαν μια προσπάθεια ανάπτυξης μιας απλής, ευέλικτης γλώσσας για χρήση σε κινητά, palmtops και ITVs. Η πρώτη προσπάθεια ονομαζόταν *Oak* και θεωρείται ο πρόγονος της σημερινής Java. Καθώς η Oak εξελίχθηκε σε Java, τα πλεονεκτήματα της νέας γλώσσας έγιναν γρήγορα φανερά. Έχοντας την υποστήριξη μεγάλων ονομάτων της πληροφορικής όπως η IBM, η Intel, η Microsoft[2], και ανοίγοντας τα ενδότερά της στους χρήστες ως ανοιχτό λογισμικό η Java διαδόθηκε ευρύτατα και εξελίχθηκε με νέες εκδόσεις και περισσότερες δυνατότητες. Σήμερα μετράει αισίως την 6η έκδοσή της (version 1.6 ή αλλιώς Java 6).

Η Java ανήκει στις αντικειμενοστραφείς γλώσσες προγραμματισμού (Object Oriented languages) οι οποίες διαφέρουν σημαντικά από τις λεγόμενες procedural (διαδικαστικές;) γλώσσες. Παραδείγματα διαδικαστικών γλωσσών είναι οι πρώτες εκδόσεις των C, Fortran, Basic. Ο αντικειμενοστραφής προγραμματισμός *(**O**bject **O**riented **P**rogramming **OOP**)* είναι ουσιαστικά μια νέα θεώρηση στο σχεδιασμό και οργάνωση του πηγαίου κώδικα. Ενώ στις μη αντικειμενοστραφείς γλώσσες τα δεδομένα με τις δομές ελέγχου συνδυάζονται μόνο μέσα στον τελικό κώδικα, στις αντικειμενοστραφείς, δεδομένα και δομές ελέγχου συνθέτουν αφηρημένες, πλήν διακριτές οντότητες, τα *"αντικείμενα"*: το κάθε αντικείμενο είναι ένα κομμάτι λογισμικού (software), ορίζει μια κατάσταση (state) ή μια συμπεριφορά και μπορεί να χρησιμοποιηθεί ως έχει ή να συνδυαστεί με άλλα προκειμένου να αναπτυχθεί μια εφαρμογή. Η αντικειμενοστρέφεια κερδίζει στις μέρες μας όλο και περισσότερο έδαφος στις εφαρμογές πληροφορικής. Για το λόγο αυτό πολλές από τις παλιές, procedural γλώσσες προγραμματισμού έχουν προσθέσει στοιχεία OOP στις νέες εκδόσεις τους υπό μορφή επεκτάσεων, όπως οι *C++, Fortran 2005, PHP, Perl*, ενώ αρκετές νέες καθαρά αντικειμενοστραφείς έχουν εμφανιστεί *(Python, Ruby, C#)*.

Οι βασικές αρχές του αντικειμενοστραφούς προγραμματισμού είναι:

- Η **data abstraction** είναι η τάση να θεωρούμε ετερόκλητα σύνολα δεδομένων, εντολών και λειτουργιών σαν αυτοτελείς οντότητες.

[2]Η διένεξη που επήλθε μεταξύ Sun και Microsoft για την ενσωμάτωση της Java στον Internet Explorer οδήγησετη δεύτερη στην ανάπτυξη μιας νέας αντικειμενοστραφούς γλώσσας, της C#.

- Η **απόκρυψη πληροφοριών** (information hiding) αναφέρεται στο γεγονός ότι χρησιμοποιούμε τις παραπάνω οντότητες, τα αντικείμενα, χωρίς να μας ενδιαφέρει ο εσωτερικός τρόπος λειτουργίας τους.

- Η **ενθυλάκωση δεδομένων** (data encapsulation) είναι η προσπάθεια απόκρυψης του τρόπου λειτουργίας του ενός αντικειμένου από το άλλο, αφήνοντας φανερές μόνο τις παραμέτρους και τις διεργασίες που χρειάζονται για την επικοινωνία τους.

- Η **modularity**, έννοια που συνεπάγεται από τις παραπάνω, καθώς οποιοδήποτε αντικείμενο, από απλή κλάση ως ολοκληρωμένη εφαρμογή εφόσον αποτελείται από έναν αριθμό διακριτών τμημάτων μπορεί να τροποποιηθεί κατά βούληση, προσθαφαιρώντας κάποια από αυτά ή αλλάζοντας τον τρόπο λειτουργίας τους.

- Ο **πολυμορφισμός** (polymorphism) περιγράφει τη δυνατότητα που υπάρχει στον OOP να δημιουργώ νέες "εκφάνσεις" υπαρχόντων αντικειμένων όταν τα χρειάζομαι, κλώνους που φέρουν όλα τα χαρακτηριστικά των αρχικών, χωρίς να πρέπει να τα επαναπροσδιορίζω εξαρχής κάθε φορά. Μπορώ μάλιστα να προσθέσω στα αντίγραφα αυτά επιπλέον λειτουργίες και ιδιότητες, μιλώντας πια όχι για κλώνους αλλά για **απογόνους** των αρχικών κλάσεων.

- Η **κληρονομικότητα** (inheritance) καθώς κάθε νέο αντικείμενο 'γεννιέται' από κάποιο ήδη ορισμένο, μπορούμε να θεωρήσουμε μια *γενεαλογική σχέση* μεταξύ όλων των αντικειμένων-κλάσεων της Java. Η κλάση προπάτορας όλων είναι η `Object class`. Η έννοια της κληρονομικότητας δεν είναι απλά μια επιτυχημένη παρομοίωση: κάθε κλάση-απόγονος κληρονομεί αυτούσιες όλες τις μεθόδους, παραμέτρους και ιδιότητες της κλάσης/γονιού καθώς και **όλων** των προγόνων της. Αν ο χρήστης το επιθυμεί μπορεί κατά τη διάρκεια της κλήσης/γέννησης της νέας κλάσης να την εφοδιάσει με επιπλέον χαρακτηριστικά τα οποία προστίθενται στο υπάρχον 'γενετικό υλικό' της. Πέρα από τις κλάσεις η κληρονομικότητα υφίσταται και μεταξύ διαπροσωπειών, των βασικών δομών ελέγχου δηλαδή, ανοίγοντας νέες δυνατότητες στο σχεδιασμό και την ανάπτυξη του πηγαίου κώδικα μιας εφαρμογής.

Οι παραπάνω αρχές αφενός προσφέρουν απεριόριστες δυνατότητες παραμετροποίησης των βασικών βιβλιοθηκών της Java, οι οποίες περιλαμβάνουν χιλιάδες κλάσεις, αφετέρου ορίζουν μια αυστηρώς ιεραρχική δομή για την οργάνωση και ταξινόμησή τους σε πακέτα.

2.1.2 Οι προσομοιώσεις και η Java.

Είναι πια φανερό γιατί το OSP project αναπτύχθηκε σε Java. Ο Wolfgang Christian εκμεταλλεύτηκε τις προαναφερθείσες αρετές της γλώσσας παίρνοντας βασικές κλάσεις/αντικείμενα και τροποποιώντας τες ή συνδυάζοντάς τες παρήγαγε καινούριες, εξειδικευμένες

για ανάπτυξη προσομοιώσεων φυσικής. Καθώς οι κλάσεις που ανέπτυξε αποτελούν λογισμικό ανοιχτού κώδικα, οι χρήστες είναι ελεύθεροι να επεκτείνουν κι άλλο τις δυνατότητες και τη συμπεριφορά τους για να δημιουργήσουν το πρόγραμμα της αρεσκείας τους[3]. Για να το καταφέρουν όμως αυτό απαιτούνται κάποιες βασικές ικανότητες προγραμματισμού και μια εξοικείωση με τα ενδότερα της Java.

2.2 Χρήσιμα εργαλεία

Ακόμα και μια "συνοπτική" περιγραφή μιας τόσο εξελιγμένης γλώσσας προγραμματισμού θα απαιτούσε τουλάχιστον 100 σελίδες κειμένου, θα έβγαζε την παρούσα εργασία εκτός θέματος και στην τελική, θα την έκανε δυο φορές βαρύτερη απ'οτι είναι. Έτσι λοιπόν θα παρουσιάσουμε στη συνέχεια μόνο τα απολύτως απαραίτητα χαρακτηριστικά της Java για να ξεκινήσει κάποιος να αναπτύσσει τις εφαρμογές του με τα εργαλεία του OSP project. Το περισσότερο υλικό αυτού του κεφαλαίου προήλθε από το εξαιρετικό βιβλίο της O'Reilly, Learning Java[6], καθώς και από την αστείρευτη πηγή πληροφοριών του διαδικτύου[7]. Ειδικά για μια τόσο δημοφιλή, διαδικτυακά προσανατολισμένη γλώσσα υπάρχουν αναρίθμητοι ιστότοποι, forums και wikis αφιερωμένα, χωρίς να συμπεριλάβουμε τα (συνήθως δωρεάν) ψηφιακά συγγράμματα. Όσοι αναγνώστες νοιώσουν την ανάγκη να εμβαθύνουν σε επιμέρους τομείς ή να επεκταθούν σε καινούριους μπορούν να συμβουλευτούν τη βιβλιογραφία και τους υπερσυνδέσμους που παρατίθενται εκεί. Από την άλλη πλευρά, αν κατέχετε τη γλώσσα ή επείγεστε να αποκτήσετε μια γενική άποψη της εργασίας, μπορείτε κάλλιστα να παρακάμψετε το παρόν κεφάλαιο.

2.2.1 Java Virtual Machine

Για να τρέξει ένα λειτουργικό σύστημα εφαρμογές της Java θα πρέπει να έχει εγκατεστημένο το μεταγλωττιστικό και εκτελεστικό πυρήνα της γλώσσας, την **J**ava **V**irtual **M**achine. Πρόκειται για λογισμικό, προσαρμοσμένο κατάλληλα για το εκάστοτε λειτουργικό σύστημα, που εμπεριέχει το επονομαζόμενο *runtime system* της γλώσσας και "τρέχει" όποια εφαρμογή γραμμένη σε Java του υποδείξουμε. Η JVM μπορεί να υφίσταται είτε υπό μορφή αυτόνομου προγράμματος είτε να είναι ενσωματωμένη σαν plugin σε κάποιον Internet browser **μέσα** στον οποίο τρέχουμε κάποιο Java applet[4].

[3] Ένα απτό παράδειγμα είναι αυτό της κλάσης `PlotFrame`: Ο W.Christian πήρε μια βασική κλάση της Java, την `JFrame`, που υλοποιεί ένα παραθυρικό πλαίσιο όπου μπορούν να τυπωθούν γραφικά, και την επέκτεινε, δημιουργώντας την κλάση-απόγονο `PlotFrame`, η οποία περιλαμβάνεται στις βιβλιοθήκες του OSP. Η `PlotFrame` είναι ένα παραθυρικό πλαίσιο **εξειδικευμένο** για προβολή μαθηματικών γραφικών παραστάσεων. Με τη σειρά του, ο γραφών επέκτεινε την κλάση `PlotFrame` για τη δημιουργία των κλάσεων που υλοποιούν τις **συγκεκριμένες** γραφικές παραστάσεις που εμφανίζονται στις προσομοιώσεις της παρούσας εργασίας.

[4] Τα Java applets είναι μικροεφαρμογές της Java που τρέχουν μόνο με τη βοήθεια κάποιου browser ή ενός εξειδικευμένου εργαλείου που λέγεται `appletviewer`

Η JVM είναι υπεύθυνη συν τοις άλλοις για τις παρακάτω λειτουργίες:

- Φορτώνει τα μεταγλωτισμένα `.class` αρχεία που υλοποιούν τα προγράμματα σε Java.
- Επαληθεύει τις κλάσεις που προέρχονται από άγνωστους προορισμούς.
- Εκτελεί τον μεταγλωττισμένο ψευδοκώδικα (bytecode).
- Διαχειρίζεται τους πόρους και τη μνήμη του συστήματος ως ένα άλλος, ιδεατός υπολογιστής.
- Υλοποιεί το λεγόμενο *dynamic compilation* μετατρέποντας τον ψευδοκώδικα σε γλώσσα μηχανής.

Πέρα από τη βασική εγκατάσταση της Java σε ένα σύστημα που θα περιλαμβάνει οποσδήποτε την JVM και τις βασικές βιβλιοθήκες, μπορούμε να εγκαταστήσουμε επιπλέον αρκετά προγραμματιστικά εργαλεία που βοηθούν στην ανάπτυξη εφαρμογών. Το πιο γνωστό είναι το JDK[5] που μας δίνει τη δυνατότητα να μεταγλωττίσουμε (`javac`), να τρέξουμε (`java`) και να οργανώσουμε τις κλάσεις που δημιουργούμε σε πακέτα. Για παράδειγμα, εφόσον έχουμε εγκατεστημένο το JDK, για να εκκινήσουμε από την κονσόλα[6] την εφαρμογή `myApplication` πληκτρολογούμε:

```
% java myApplication
```

Τέλος αξίζει να σημειώσουμε ότι υπάρχουν αρκετά ολοκληρωμένα περιβάλλοντα για ανάπτυξη εφαρμογών (IDEs) σε Java όπως το NetBeans της Sun Microsystems, το Eclipse της IBM ή το JBuilder από τη Borland. Τα δύο πρώτα είναι μάλιστα λογισμικά ανοιχτού κώδικα, ως εκ τούτου δωρεάν.

2.2.2 Ξεκινώντας

Για να υπάρξει μια εφαρμογή της Java, θα πρέπει να περιέχει τουλάχιστον μια κλάση που να φέρει μέσα της τη μέθοδο `main()`. Οι *μέθοδοι*, όπως θα δούμε παρακάτω, είναι για τη Java ότι είναι οι ρουτίνες για τη Fortran, ή οι functions στη γλώσσα C. Η κλάση αυτή, περιέχει κώδικα ο οποίος θα εκτελεστεί πρώτος κατα την εκκίνηση και ανήκει στις λεγόμενες, domain classes του προγράμματός μας. Για να τρέξουμε μια εφαρμογή, εκκινούμε την JVM, δίνοντάς της σαν όρισμα το όνομα του αρχείου .class που την υλοποιεί και αν χρειαστεί, παράμετρους για τον compiler ή και τα ορίσματα που απαιτούνται.

π.χ. `% java myApplication`

[5] **J**ava **D**evelopment **K**it
[6] γραμμή εντολών /command line

Ο μεταγλωττιστής αναζητά τότε στο *classpath* όσες άλλες κλάσεις αναφέρονται στη βασική μας κλάση και της είναι απαραίτητες. Η αναζήτηση αυτή γίνεται ακόμα και σε συμπιεσμένα αρχεία καθώς η Sun έχει εισάγει ένα δικό της είδος συμπιεσμένου αρχείου, το *.jar*[7] το οποίο δύναται να εκτελεστεί. Για την ακρίβεια, αν θέλαμε να τρέξουμε μια εφαρμογή που βρίσκεται υπο μορφή .jar αρχείου, θα δίναμε την εντολή σε περιβάλλον κονσόλας:

```
% java -jar myApplication.jar
```

Μόλις οι αναγκαίες κλάσεις βρεθούν και επαληθευτούν, ενεργοποιούνται οι δομές ελέγχου και οι αλγόριθμοι του πηγαίου κώδικα της εφαρμογής, καλούνται μέθοδοι και εκκινούνται νήματα εργασιών (threads).

2.2.3 Το classpath.

Η έννοια του *PATH* (μονοπάτι,διαδρομή) είναι γνωστή τόσο σε Unix, όσο και σε Windows συστήματα. Πρόκειται για μια μεταβλητή που προσδιορίζει στο λειτουργικό σύστημα τη θέση κάποιων απαραίτητων για αυτό αρχείων, υπό μορφή λίστας τοποθεσιών/φακέλων. Καθώς η JVM αποτελεί ένα ανεξάρτητο, ιδεατό σύστημα, ενσωματωμένο στο λειτουργικό μας, έχει τη δικιά του PATH μεταβλητή, επονομαζόμενη *java CLASSPATH*, η οποία με παρόμοιο τρόπο καταδεικνύει κλάσεις απαραίτητες για την ομαλή λειτουργία της. Σε Unix based συστήματα, το CLASSPATH ορίζεται από την κονσόλα ως εξής:

```
% CLASSPATH=/home/Bill/Java/classes:/home/Bill/lib/myApp.jar:.
```

Στο παραπάνω παράδειγμα, με τις άνω-κάτω τελείες (που επίτηδες χρωματίζονται κόκκινες) διαχωρίζουμε το κάθε στοιχείο της λίστας με τις τοποθεσίες. Το παραπάνω CLASSPATH υποδεικνύει λοιπόν στην JVM τη θέση του υποφακέλου `classes`, τη θέση της εφαρμογής `myApp.jar` και τον παρόν υποκατάλογο στον οποίο βρίσκεται ο χρήστης, μέσω της τελείας (.) στο τέλος. Οποιαδήποτε στιγμή θελήσουμε να εξετάσουμε τι περιέχει το CLASSPATH χρησιμοποιούμε την εντολή:

```
% export CLASSPATH
```

Αν βρισκόμαστε σε λειτουργικό σύστημα Windows, ο καθορισμός του java CLASSPATH δίνεται παρακάτω - παρατηρούμε τη χρήση του ελληνικού ερωτηματικού (semicolon)σαν διαχωριστικό αντί για τις άνω-κάτω τελείες.

```
C:\>set CLASSPATH=C:\Users\Bill\Java\classes; C:\Users\Bill\myPackage;
```

Ενα χρήσιμο εργαλείο για το CLASSPATH είναι το `javap` που αναλύει ποιές κλάσεις συσχετίζονται με αυτή που του θέτουμε σαν όρισμα:

[7] JAR = **J**ava **AR**chive

```
% javap myApp
```

Εδώ για παράδειγμα θα μας δώσει μια λίστα με τις τοποθεσίες (πακέτα) όπου περιέχονται οι κλάσεις που αναφέρει στον κώδικά της η κλάση `myApp`. Το εργαλείο `javap` περιέχεται στο JDK.

2.2.4 Ο μεταφραστής (compiler) ths java.

Ο μεταφραστής είναι το λογισμικό που μετατρέπει τον πηγαίο κώδικα που γράφει ο χρήστης, σε ψευδοκώδικα (bytecode) που με τη σειρά του μετατρέπεται σε γλώσσα μηχανής από την JVM. Είναι γραμμένος και ο ίδιος σε γλώσσα Java και περιλαμβάνεται στο JDK. Τα αρχεία πηγαίου κώδικα της Java έχουν από σύμβαση την κατάληξη `.java` ενώ τα μεταγλωττισμένα, την κατάληξη `.class`. Κάθε αρχείο `.java` μεταγλωττίζεται χωριστά και ενδέχεται να περιέχει παραπάνω από μια κλάσεις/αντικείμενα που συνήθως διαθέτουν κοινά `import` και `package` statements.

Ενα σημείο που πρέπει να τονιστεί εδώ είναι ότι παρόλο που περιέχονται πολλές κλάσεις μέσα σε ένα `.java` file, μόνο μια από αυτές επιτρέπεται να δηλωθεί `public`, να είναι δηλαδή ορατή προς τα έξω. Αν υπάρχουν παραπάνω από μια `public` κλάσεις, τότε ο μεταφραστής θα δώσει compilation error.

Σε αντίθεση με τον μεταγλωττιστή (java interpreter), ο μεταφραστής δέχεται σαν όρισμα ολόκληρο το όνομα του αρχείου. Για παράδειγμα, για να μετατρέψω τον πηγαίο κώδικα της εφαρμογής `myApp`, θα πρέπει να δώσω στον compiler:

```
% javac myApp.java
```

Αν η διαδικασία εξελιχθεί ομαλά, θα προκύψει ένα αρχείο `myApp.class` που μπορώ να εκτελέσω καλώντας τον interpreter (JVM):

```
% java myApp
```

Σε περίπτωση που το προς μετάφραση αρχείο βρίσκεται σε άλλο φάκελο από τον τρέχον, χρησιμοποιώ τον compiler με την παράμετρο `-d` και τη διαδρομή ως αυτόν το φάκελο:

```
% javac -d /home/Bill/Java/classes myApp.java
```

Ο java compiler είναι αρκετά "έξυπνος" και ευέλικτος διευκολύνοντας τη δύσκολη ζωή των προγραμματιστών: κάθε φορά που μεταφράζει μια κλάση πηγαίου κώδικα, εξετάζει αν οι κλάσεις από τις οποίες εξαρτάται αυτή (reference classes) είναι πρόσφατα μεταφρασμένες, ξαναμεταφράζοντάς τες αν χρειάζεται. Επίσης, μπορεί να ολοκληρώσει το compilation, ακόμα και όταν μερικές από τις (reference classes) βρίσκονται υπό μορφή `.class` αρχείων.

2.2.5 Ασφάλεια και αρχεία αδειών (policy files).

Όπως προαναφέραμε στην αντίστοιχη παράγραφο, η JVM κατά την εκτέλεση της κάθε εφαρμογής, εξετάζει παράλληλα την προέλευση των κλάσεων, αποτρέποντας πιθανές κακόβουλες ενέργειες που θα έβλαπταν το σύστημα του χρήστη. Ειδικά οι java applets οι οποίες συνήθως τρέχουν από το Διαδίκτυο μέσα στον εκάστοτε internet browser, θα μπορούσαν να επιτρέψουν σε τρίτους την πρόσβαση στο λειτουργικό σύστημα, εκμεταλλευόμενες ανοιχτές θύρες. Ο διαχειριστής ασφαλείας (security manager) της JVM δεν είναι ενεργοποιημένος by default, τον καλούμε εμείς από την κονσόλα ως εξής:

```
C:\> java -Djava.security.manager myApp
```

Εδώ καλούμε την εφαρμογή `myApp` αλλά υπό την εποπτεία του security manager ο οποίος δε θα την αφήσει να συνδεθεί με το Δίκτυο, ούτε να γράψει και να διαβάσει αρχεία εκτός του δικού της φακέλου.

Τα *policy files* ορίζουν την ελευθερία κάθε κλάσης, τις ενέργειες που επιτρέπεται ή απαγορεύεται να κάνει, παρέχοντας στην JVM τα απαραίτητα αποδεικτικά στοιχεία. Λογικό ήταν λοιπόν το JDK να παρέχει ένα εργαλείο, την `policytool` *utility* η οποία κατασκευάζει άδειες για οποιαδήποτε κλάση επιθυμεί ο χρήστης, μέσω ενός εύχρηστου παραθυρικού περιβάλλοντος. Εφόσον διαθέτουν τις αντίστοιχες άδειες, ο διαχειριστής ασφαλείας κάνει εξαιρέσεις, επιτρέποντας δέσμες ενεργειών στις κλάσεις/εφαρμογές.

2.3 Δομή και βασικές λειτουργίες.

Προχωράμε τώρα στο βασικές λειτουργίες της γλώσσας, όπως αυτές εκφράζονται μέσα από το συντακτικό της. Θα εξετάσουμε τα κυριότερα είδη αντικειμένων (κλάσεις, μεθόδους, διαπροσωπείες[8]), τους κύριους τύπους δεδομένων και εντολών και το ουσιαστικότερο, το πώς συσχετίζονται όλα αυτά.

2.3.1 Μια ιεραρχική δομή.

Αν διαβάσουμε τον πηγαίο κώδικα μιας κλάσης, ή ξεκινήσουμε να γράψουμε τον δικό μας, θα συναντήσουμε στην αρχή του τα λεγόμενα `import` και `package` statements. Πρόκειται για εντολές που από τη μια καθορίζουν σε πιο πακέτο (`package`) ανήκει η κλάση μας, από την άλλη **συσχετίζουν** την κλάση μας με άλλα αντικείμενα, που αξιοποιεί στον κώδικά της. Παραδείγματα τέτοιων δηλώσεων είναι:

[8] Αναφερόμαστε στις διαπροσωπείες σε προγραμματιστικό επίπεδο (Java Interfaces), και όχι στις διεπαφές που επιτρέπουν στο χρήστη τον χειρισμό ενός προγράμματος με γραφικό περιβάλλον (GUIs = Graphical User Interfaces). Κάποιοι τις αποδίδουν στα ελληνικά και ως *διασυνδέσεις*.

```
package osp.myproject;
import javax.swing.*;

class MyApp {...
}
```

Εδώ δηλώνουμε ότι η κλάση `MyApp` ανήκει στο πακέτο `osp.myproject` και στην οποία μπορούμε να εισάγουμε (`import`) και να χρησιμοποιήσουμε ονομαστικά, οποιαδήποτε κλάση ανήκει στο πακέτο `javax.swing`.

Τα πακέτα είναι ένα είδος βιβλιοθηκών, με τις κλάσεις και τις διαπροσωπείες να παίζουν το ρόλο των βιβλίων. Οργανώνουν ιεραρχικά τα ονόματα των κλάσεων προκειμένου ο compiler και η JVM να οδηγηθούν στα απαραίτητα κάθε φορά αρχεία, για τη μεταγλώττιση και την εκτέλεση της εφαρμογής αντίστοιχα. Η κατηγοριοποίηση γίνεται βάσει των λειτουργιών κάθε κλάσης και στα ονόματα των πακέτων χρησιμοποιείται η τελεία (.) για να διαχωρίσει τις υποκατηγορίες. Η κατανομή των κλάσεων σε ομάδες έχει μια άμεση συνέπεια: όλες οι κλάσεις ενός πακέτου μπορούν να "καλούν" η μια την άλλη χωρίς τη χρήση του `import` statement. Αντιθέτως, αν χρειάζεται να χρησιμοποιηθεί μια κλάση από άλλο πακέτο πρέπει ή να την καλέσουμε με το ολοκληρωμένο όνομα-διαδρομή[9] ή να την κάνουμε `import` στην αρχή (μόνη της ή όλοκληρο το πακέτο στο οποίο ανήκει με χρήση του αστερίσκου (*), βλέπε πιο πάνω παράδειγμα). Προκειμένου να δηλώσουμε σε ποιό πακέτο ανήκει μια κλάση χρησιμοποιούμε το `package` statement.

Εδώ πρέπει να τονιστούν κάποιες σημαντικές λεπτομέρειες: Μόνο οι δηλωμένες ως `public` κλάσεις είναι ορατές εκτός ενός πακέτου. Οπότε με ένα `import` statement της μορφής `import javax.swing.*` θα έχουμε πρόσβαση σε όσες public κλάσεις έχει το πακέτο `javax.swing`, οι υπόλοιπες είναι προσβάσιμες μόνο εντός του πακέτου. Η τελευταία έκδοση της Java μας δίνει τη δυνατότητα του `import static`. Με την εντολή αυτή εισάγουμε όλα τα static members ενός πακέτου (κλάσεις, μεταβλητές, διαπροσωπείες, μεθόδους) και τα χρησιμοποιούμε με το όνομά τους. Για παράδειγμα οι κλάσεις που υλοποιούν μαθηματικές συναρτήσεις είναι στατικά μέλη του πακέτου `java.lang.Math`. Γράφοντας:

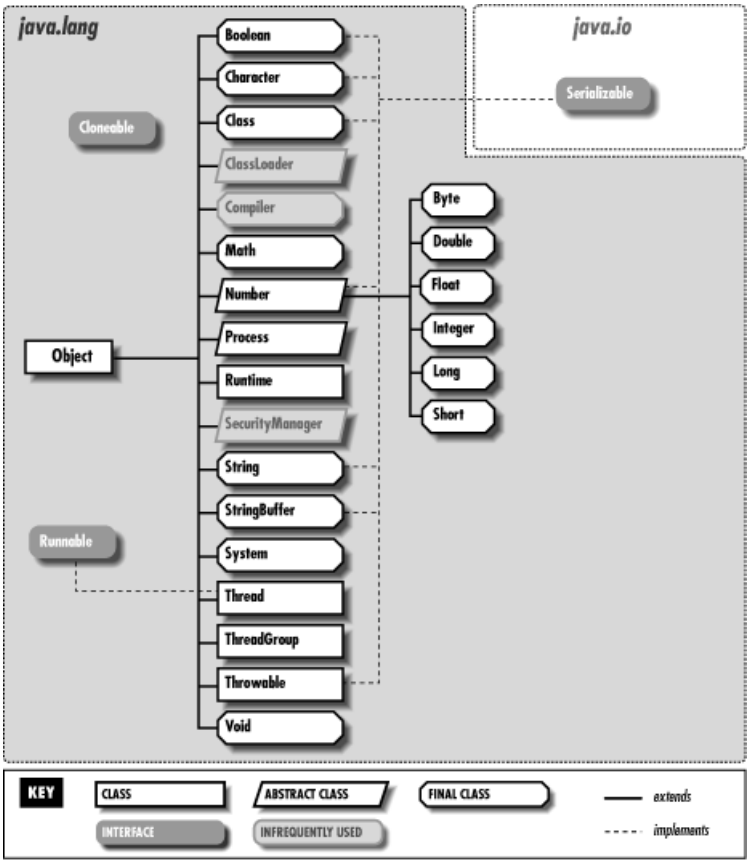
```
import static java.lang.Math.*;
```

μπορούμε στο σώμα της κλάσης μας να γράφουμε εντολές σαν κι αυτή:

```
double diameter = sin(omega);
```

Καλούμε δηλαδή τη στατική μέθοδο `java.lang.Math.sin()` με το "μικρό" της.

[9] FQN=**F**ull **Q**ualified **N**ame Για παράδειγμα για την κλάση `JFrame` που υλοποιεί ένα παραθυρικό πλαίσιο, το FQN είναι `javax.swing.JFrame`



Σχημα 2.1: Παράδειγμα ιεραρχικής οργάνωσης της Java - Η βασική βιβλιοθήκη `java.lang` με τα πακέτα που περιέχει και τους συσχετισμούς της με άλλες βιβλιοθήκες.

Τέλος, πέρα από την "ορατότητα" των κλάσεων, η ιεραρχική ονοματολογία των πακέτων δεν έχει αντικειμενική έκφανση: π.χ. το γεγονός ότι τα πακέτα `java.lang.String` και `java.lang.Thread` είναι και τα δύο υποκατηγορίες του java.lang δε σημαίνει ότι υπάρχει στο σύστημα φάκελος `lang` με υποφακέλους τους `String` και `Thread`. Πρόκειται για έναν ιδεατό διαχωρισμό με καθαρά πρακτικούς σκοπούς.

2.3.2 Οι κλάσεις

Οι κλάσεις, μαζί με τις μεθόδους, αποτελούν τα βασικά αντικείμενα κάθε αντικειμενοστραφούς γλώσσας. Είναι οι θεμελιώδεις λίθοι των προγραμμάτων και μπορεί να υλοποιούν οτιδήποτε, από κάτι εντελώς συγκεκριμένο, όπως ένα πλήκτρο ελέγχου στο παράθυρο μιας εφαρμογής, μέχρι κάτι εντελώς αφηρημένο, όπως οι εκφράσεις ανίας ενός φανταστικού πρωταγωνιστή βιντεοπαιχνιδιού. Κάθε κλάση περιέχει παραμέτρους, μεθόδους, αλγορίθμους και άλλους τύπους πληροφοριών και σκοπό έχει να τελέσει μια εργασία, από επεξεργασία δεδομένων μέχρι ορισμό νέων κλάσεων. Ορίζουμε μια κλάση με τις εντολές:

```
class Particle {
int mass = 2;
double charge = 3.43;

double getSpeed() { ...}

}
```

Η κλάση `Particle` ορίζει και αρχικοποιεί στο σώμα της μια ακέραια μεταβλητή με το όνομα `mass`, μια δεκαδική διπλής ακριβείας με το όνομα `charge` και μια μέθοδο με το όνομα `getSpeed()` που παράγει αριθμητικό αποτέλεσμα δεκαδικό διπλής ακριβείας `(double)`. Παρατηρούμε ότι το όνομα της κλάσης αρχίζει πάντα με κεφαλαίο γράμμα, ενώ των μεθόδων και των παραμέτρων με μικρό.

Το παραπάνω παράδειγμα είναι αρκετά απλοϊκό. Σε μια ολοκληρωμένη εφαρμογή, η κύρια κλάση (domain class), που εκκινείται πρώτη από την JVM περιέχει, όπως προαναφέραμε, οπωσδήποτε τη μέθοδο `main()`, τα `import` και `package` statements καθώς και μια σειρά από modifiers που καθορίζουν την εμβέλεια δράσης της. Ένα πιο αντιπροσωπευτικό παράδειγμα λοιπόν, είναι:

```
package osp.myproject
import javax.lang.String.*

public class MyApp {
  public static void main( String[] args) {
  ...
  }
}
```

Η κλάση μου εδώ λέγεται `MyApp`, ανήκει στο πακέτο `osp.myproject` και έχει πρόσβαση σε όσες κλάσεις χρειάζεται από το πακέτο `java.lang.String`. Παρατηρούμε ότι είναι δηλωμένη ως `public`, άρα παρόλο που ανήκει στο `osp.myproject` είναι ορατή και εκτός αυτού. Η κλάση μου περιέχει μια `main()` μέθοδο, άρα είναι είναι η κύρια κλάση της εφαρμογής μου. Εδώ η `main()` method φέρει τους modifiers `public`, `static` και `void`. Μια μέθοδος δηλωμένη `public`, παρόμοια με τις κλάσεις, μπορεί να κληθεί από οποιαδήποτε άλλη κλάση εκτός του πακέτου, αρκεί η τελευταία να έχει κάνει `import` το πακέτο στο οποίο ανήκει η `MyApp`. Ο χαρακτηρισμός `void` υποδηλώνει ότι η μέθοδος `main()` δε θα επιστρέψει κανένα αποτέλεσμα μετά την κλήση της[10]. Για το modifier `static` θα μιλήσουμε αργότερα. Η μέθοδος `main()` παίρνει εδώ ένα αλφαβαριθμητικό όρισμα (string argument) το οποίο δεν είναι τίποτα άλλο από τις παραμέτρους που παραθέτουμε μαζί με το όνομα στη γραμμή εντολών, κατά την εκτέλεση της εφαρμογής μας.

[10] Αυτό δε σημαίνει ότι δεν παράγει έργο, απλά μπορεί να αλλάζει τις τιμές κάποιων παραμέτρων, ή να επεξεργάζεται δεδομένα τα οποία αποθηκεύονται κάπου κτλ.

2.3.3 Η κληρονομικότητα στις κλάσεις - οι τελεστές *new* και *extends*.

Από μια άλλη οπτική γωνία, οι κλάσεις μπορούν να θεωρηθούν, εκτός από διακριτά σύνολα κώδικα με συγκεκριμένη λειτουργία, και ως προσχέδια, έτοιμες φόρμες τις οποίες χρησιμοποιούμε για να δημιουργήσουμε κλώνους ή απογόνους της μητρικής κλάσης για να συνθέσουμε την εφαρμογή μας. Ένα κατανοητό παράδειγμα είναι η κλάση `JButton` που ανήκει στο πακέτο `javax.swing` και υλοποιεί ένα interactive πλήκτρο στο παραθυρικό περιβάλλον της εφαρμογής μας. Κατά την ανάπτυξη μιας εφαρμογής, μπορεί να χρειαστώ να τοποθετήσω αρκετά πλήκτρα στο παράθυρό της, το καθένα επιφορτισμένο με άλλη λειτουργία. Κάθε πλήκτρο λοιπόν θα έχει το δικό του τίτλο, ή σύμβολο και θα ενεργοποιεί έναν ξεχωριστό αλγόριθμο στα ενδότερα του κώδικά μου. Όπως καταλαβαίνουμε, όλα τα πλήκτρα που θα χρησιμοποιήσω στην εφαρμογή μου, θα αποτελούν ξεχωριστές κλάσεις/απογόνους της μητρικής κλάσης `JButton` κατασκευασμένες με τη βοήθεια του τελεστή (operator) **`new`**.

Ο τελεστής **`new`** μας δίνει στην πραγματικότητα ένα νέο δείκτη στο μητρικό αντικείμενο (reference to object). Ο κώδικας που "ορίζει" ένα νέο αντικείμενο, είτε απ΄το μηδέν, είτε σαν απόγονο άλλου, ονομάζεται constructor (κατασκευαστής). Συνεχίζοντας το παράδειγμα με τους απογόνους του `JButton` ας δούμε τη χρήση του **`new`**:

```
JButton mybutton = new JButton( 25, 50);
```

Παρόλο που είναι μια γραμμή, ο συγκεκριμένος constructor "λέει πολλά". Δηλώνει τη γέννηση ενός νέου αντικειμένου, με το όνομα mybutton το οποίο είναι **τύπου** `JButton`. Από την κλάση `JButton` λοιπόν που υλοποιεί ένα γενικό γραφικό πλήκτρο γεννήθηκε ένα **συγκεκριμένο** γραφικό πλήκτρο εφοδιασμένο με όνομα και καθορισμένες διαστάσεις σε pixels. Εδώ αρχίζουν να διαφαίνονται τα πλεονεκτήματα της αντικειμενοστρέφειας: δε χρειάζεται να γράψω από την αρχή τον κώδικα για να ζωγραφίσω ένα πλήκτρο στο παράθυρό μου, απλά καλώ την κλάση που το κάνει και την παραμετροποιώ όπως απαιτείται. Επίσης αντιλαμβανόμαστε ότι κάθε κλάση της Java πρακτικά αποτελεί και από ένα διαφορετικό τύπο δεδομένων[11]. Άρα σε μια αντικειμενοστραφή γλώσσα, υπάρχουν τόσοι διαφορετικοί τύποι δεδομένων, όσα αντικείμενα.

Στην περίπτωση που θέλουμε ο απόγονος μιας κλάσης να έχει διαφορετικά χαρακτηριστικά από τη μητρική, θα χρησιμοποιήσουμε τον τελεστή **`extends`**. Ορίζοντας μια κλάση μέσω αυτού έχουμε τη δυνατότητα να προσθέσουμε επιπλέον ικανότητες στην κλάση μας ορίζοντας νέες μεθόδους και μεταβλητές ή τροποποιώντας τη λειτουργία όσων έχει κλήρονομήσει από τη μητρική. Το παράδειγμα παρακάτω είναι πιο διαφωτιστικό:

[11] Όπως οι `int, double` κτλ.

```
class Animal {
  double weight;
  ...
  void eat() { ...}
}
```

Η μητρική κλάση `Animal` περιέχει μια `double` μεταβλητή με το όνομα `weight` και μια μέθοδο με το όνομα `eat()`. Χρησιμοποιώντας τον τελεστή `extends` δημιουργώ την κλάση `Bird`:

```
class Bird extends Animal {
  int legs = 2;
  int wings = 2;
  fly() { ...}
}
```

Η κλάση `Bird` διαθέτει δύο νέες, ακέραιες μεταβλητές, τις `legs` και `wings` και μια νέα μέθοδο, την `fly()`. Παρόλο που δεν το αναφέρει, διαθέτει επίσης και τις μεταβλητές και μεθόδους της `Animal` γιατί αποτελεί απόγονο/*υποκλάση* της. Αν θέλαμε θα μπορούσαμε να επαναπροσδιορίζαμε κάποια από τα στοιχεία της `Animal`. Ως εξής:

```
class Bird extends Animal {
  int legs = 2;
  int wings = 2;
  void eat() { allos algorithmos }
  fly() { ...}
}
```

Εδώ η κλάση `Bird` επαναπροσδιορίζει τη μέθοδο `eat` που όρισε η κλάση `Animal`. Η τεχνική αυτή λέγεται *method overriding* και αφού εφαρμοστεί, ορατή θα είναι πια η δεύτερη εκδοχή της `eat`. Το ίδιο ισχύει και για τον προσδιορισμό ομώνυμων μεταβλητών από την υποκλάση. Αξιοποιώντας λοιπόν τη μαγεία της κληρονομικότητας μπορούμε να ορίζουμε νέα αντικείμενα (κλάσεις, μεθόδους, διαπροσωπείες) από ήδη υπαρχοντα χωρίς να επαναλαμβάνουμε άσκοπα κώδικα.

2.3.4 Μια παντοδύναμη τελεία - ο τελεστής dot.

Έστω ότι η τυχαία εφαρμογή `MyApp` διαθέτει ένα γραφικό παραθυρικό περιβάλλον, και ο πηγαίος κώδικάς της περιέχει τις πιο κάτω εντολές:

```
JFrame mywindow = new JFrame();
mywindow.setSize(200, 300);
```

Προφανώς στην πρώτη γραμμή, ορίζουμε ένα νέο αντικείμενο, ένα γραφικό παράθυρο με το όνομα `mywindow`, τέκνο της κλάσης `JFrame`. Η κλάση αυτή, ανήκει στίς βασικές βιβλιοθήκες της Java , στο πακέτο `javax.swing`, που περιέχει συστατικά για γραφικές υλοποιήσεις. Στη δεύτερη γραμμή, η κλάση που δημιουργήσαμε φαίνεται ενωμένη με μια μέθοδο, με το όνομα `setSize()`. Ο συνδετικός κρίκος τους είναι μια τελεία. Η τελεία είναι ο τελεστής dot. Ουσιαστικά είναι ένας ορατός δείκτης (reference pointer) ο οποίος μας βοηθά να αναφερόμαστε σε μια μέθοδο ή μια μεταβλητή που ανήκει σε άλλο αντικείμενο. Οφείλουμε όμως να σεβαστούμε την κληρονομικότητα. Στη παραπάνω περίπτωση η διασύνδεση γίνεται ως εξής: Η μητρική κλάση `JFrame` ορίζει στον πηγαίο κώδικά της[12] μια μέθοδο `setSize()` η οποία παίρνοντας δύο αριθμητικά ορίσματα καθορίζει το ύψος και το πλάτος του παραθύρου που υλοποιείται. Το αντικείμενο `mywindow` που παράγεται στην πρώτη γραμμή, όντας απόγονος/υποκλάση της `JFrame` κληρονομεί από αυτή όλα τα στοιχεία της άρα **και** τη μέθοδο `setSize()`. Θέλοντας λοιπόν να προσδιορίσουμε τις διαστάσεις του νεο-αποκτηθέντος αντικειμένου αναφερόμαστε σε μια μέθοδο που γνωρίζουμε ότι κατέχει από το γονιό του, με τη βοήθεια του τελεστή dot.

Ας δούμε ένα ακόμη παράδειγμα χρήσης του dot operator. Αυτή τη φορά θα τον χρησιμοποιήσουμε μαζί με τον τελεστή `new`. Έστω ότι ορίζουμε ένα αντικείμενο τύπου `JFrame` (ένα παραθυρικό πλαίσιο) με το όνομα `mywindow` και θέλουμε να προσθέσουμε μέσα σε αυτό ένα πλήκτρο, δηλαδή ένα αντικείμενο τύπου `JButton`:

```
JFrame mywindow = new JFrame();
JButton mybutton = new JButton();
mywindow.add(mybutton);
```

Ορίζουμε στην αρχή τα δύο αντικείμενα, το πλαίσιο και το πλήκτρο καλώντας τους αντίστοιχους constructors και στη συνέχεια με τη βοήθεια της μεθόδου `add()` και του τελεστή dot τοποθετώ το πλήκτρο μέσα στο πλαίσιό μου. Η μέθοδος `add()` ανήκει στην κλάση `JFrame` και ότι παίρνει σαν όρισμα προστίθεται στο εσωτερικό του παραθυρικού πλαισίου. Εδώ το όρισμα της `add()` είναι ένα ολόκληρο αντικείμενο, μια έκφανση της κλάσης `JButton` που υλοποιεί το πλήκτρο `mybutton`. Καθώς το όρισμα της `add()` προστίθεται στο `mywindow`, το πλήκτρο θα σχεδιαστεί μέσα στο παράθυρό μου. Σημειώνουμε ότι το αντικείμενο `mywindow` όντας *έκφανση (instance)* της `JFrame` έχει κληρονομήσει τη μέθοδο `add()` από αυτή, γι' αυτό και επιτρέπεται να την καλεί.

Ένα τελευταίο παράδειγμα με τον τελεστή dot:

```
double [] arrayOfnumbers = new double[5];
arrayOfnumbers[0] = 3.14;
```

[12] Το θεωρούμε γνωστό, η `JFrame` ανήκει στις βασικές βιβλιοθήκες της Java .

```
arrayOfnumbers[1] = particle.getX();
arrayOfnumbers[2] = particle.getY();
```

Εδώ ανακατεύονται στην υπόθεση για πρώτη φορά και οι πίνακες. Στην πρώτη γραμμή ορίζουμε έναν πίνακα δεκαδικών αριθμών διπλής ακριβείας, με το όνομα `arrayOfnumbers`. Ο πίνακας αυτός δεσμεύει στη μνήμη του υπολογιστή πέντε "κελιά", στα οποία μπορώ να αποθηκευτούν δεκαδικοί. Στη δεύτερη γραμμή αποθηκεύουμε τον αριθμό 3.14 στο πρώτο κελί του πίνακα[13]. Στη δεύτερη και την τρίτη γραμμή, καλούμε τις μεθόδους `getX()`, `getY()` που ανήκουν σε ένα υποθετικό αντικείμενο/κλάση με το όνομα `particle` που θα μπορούσε να υλοποιεί τη θέση ενός σωματίου στο χώρο. Οι μέθοδοι αυτοί όταν κληθούν, δίνουν αντίστοιχα την x και y συντεταγμένη του σωματιδίου στο χώρο τις οποίες, με τη βοήθεια του dot, αποθηκεύουμε στο δεύτερο και τρίτο κελί του πίνακα.

2.3.5 Οι μέθοδοι - Τοπικές και μη τοπικές μεταβλητές.

Για τις μεθόδους έχουμε μιλήσει λίγο στις προηγούμενες παραγράφους και τις είδαμε σε χρήση σε κάποιες κλάσεις, αλλά δεν εμβαθύναμε αρκετά. Οι μέθοδοι όπως προείπαμε είναι το αντίστοιχο των ρουτινών στη Fortran ή των functions στη C/C++. Δέχονται ορίσματα, ορίζουν μεταβλητές και αλγορίθμους που εκτελούνται κατά την κλήση της μεθόδου από κάποια κλάση και μπορεί να επιστρέφουν ή όχι αποτέλεσμα. Ένα παράδειγμα μεθόδου είναι το παρακάτω:

```
class Bird {
  int xPos, yPos;

    double fly ( int x, int y ) {
    double distance = Math.sqrt(x*x + y*y);
    xPos = x;
    yPos = y;
    return distance;
  }
}
```

Αρχικά ορίζουμε μια κλάση/αντικείμενο με το όνομα `Bird`. Η κλάση μας ορίζει κάποιες μεταβλητές και μια μέθοδο, τη `fly()`. Οι μεταβλητές `xPos, yPos` που ορίζονται στο κυρίως σώμα της κλάσης ονομάζονται *μεταβλητές στιγμιότυπου* (instance variables) και είναι ορατές παντού μέσα στην συγκεκριμένη κλάση, υπό ορισμένες συνθήκες και έξω από αυτή. Αντίθετα οι μεταβλητές που ορίζονται **μέσα** στη μέθοδο `fly()` ονομάζονται *τοπικές* (local) και έχουν εμβέλεια μόνο εντός της `fly()`. Η `fly()` τις χρησιμοποιεί για να υπολογίσει την

[13] Όπως θα δούμε πιο κάτω, οι πίνακες στη Java αριθμούν τα κελιά τους ξεκινώντας από το μηδέν, άρα το πρώτο κελί είναι το υπάριθμόν [0].

απόσταση τρέχοντας μια συνάρτηση και "σώζει" τις τιμές τους **εκτός** της μεθόδου, στις μεταβλητές στιγμιοτύπου `xPos, yPos`. Οι τιμές των τοπικών μεταβλητών **δεν** επιζούν μετά το πέρας της εκάστοτε μεθόδου που τις περιέχει.

Συνεχίζοντας στις διαφορές των τοπικών μεταβλητών με τις στιγμιοτύπου, οι πρώτες οφείλουν οπωσδήποτε να αρχικοποιηθούν πριν τις χρησιμοποιήσουμε κάπου αλλιώς ο compiler θα δώσει σφάλμα μεταγλώττισης *(compilation error)*. Από την άλλη πλευρά, οι μεταβλητές στιγμιοτύπου δεν είναι απαραίτητο να αρχικοποιηθούν. Αν δεν τους δώσουμε κάποια τιμή όταν τις ορίζουμε ο compiler τους δίνει αυτόματα την τιμή null, false ή μηδέν (0), ανάλογα για το αν πρόκειται για μεταβλητές τύπου κλάσης, λογικού τύπου (boolean) ή αριθμητικού τύπου (numerical) αντίστοιχα. Στο πιο πάνω παράδειγμα οι δύο μεταβλητές στιγμιότυπου `xPos, yPos` είναι αριθμητικού τύπου `(double)` και καθώς δεν τους αναθέτουμε κάποια αρχική τιμή, αυτόματα παίρνουν την τιμή μηδέν.

Τι γίνεται όμως στην περίπτωση που μέσα σε μια μέθοδο ορίζονται τοπικές μεταβλητές **ίδιου** ονόματος με κάποιες μεταβλητές στιγμιοτύπου εκτός της μεθόδου· Στην περίπτωση αυτή συμβαίνει το λεγόμενο *shadowing*, δηλαδή οι μεταβλητές εντός *επισκιάζουν* τις μεταβλητές εκτός, των οποίων οι τιμές δεν λαμβάνονται υπόψη κατά την εκτέλεση της μεθόδου.

Στο σημείο αυτό αξίζει να αναφερθούμε στο `static` modifier. Όταν μια μέθοδος, ή μια μεταβλητή χαρακτηριστεί ως `static` τότε μπορούμε να την καλέσουμε από οποιαδήποτε άλλη κλάση, χωρίς να χρειάζεται να κάνουμε import την κλάση στην οποία ανήκει, αρκεί να χρησιμοποιήσουμε το πλήρες ονοματεπώνυμο της μεταβλητής. Το πιο πάνω παράδειγμα της κλάσης `Bird` βοηθά να κατανοήσουμε αυτό το λεπτό σημείο. Η κλάση `Math` που ανήκει στις βασικές βιβλιοθήκες της `Java`[14] ορίζει μέσα της αρκετές μαθηματικές συναρτήσεις υπό μορφή `static` μεθόδων. Παρατηρούμε λοιπόν ότι μέσα στην κλάση `Bird` μπορώ να καλώ και να χρησιμοποιώ τη συνάρτηση/μέθοδο `sqrt()` που υπολογίζει την τετραγωνική ρίζα χωρίς να έχω κάνει `import` την κλάση `Math` στην κορυφή. Παρόλα αυτά η συνάρτηση/μέθοδος `sqrt()` καλείται ως `Math.sqrt()` υποδηλώνοντας ποια κλάση περιέχει τον ορισμό της.

Κλείνουμε την παράγραφο με ένα χαρακτηριστικό των μεθόδων που πολλαπλασιάζει την ευελιξία των προγραμματιστών στη σχεδίαση του πηγαίου κώδικα. Μπορούμε να ορίσουμε σε μια κλάση παραπάνω από μια μεθόδους με το **ίδιο** όνομα, αρκεί να διαφέρουν στα ορίσματα που δέχεται η καθεμιά. Η τακτική αυτή ονομάζεται method overloading και δίνει τη δυνατότητα καλώντας το όνομα μιας μόνο μεθόδου, να πραγματοποιούμε κάθε φορά διαφορετικές λειτουργίες[15]. Ο compiler κατά τη μεταγλώττιση θα ξεχωρίσει τις δυο μεθόδους λόγω του διαφορετικού αριθμού των ορισμάτων τους.

[14] `java.lang.Math`

[15] Ενώ στην πραγματικότητα κάθε λειτουργία εκτελείται από διαφορετική μέθοδο

2.3.6 Οι διαπροσωπείες στη Java.

Οι διαπροσωπείες (interfaces) αποτελούν ίσως τη σημαντικότερη αλλά και πιο δυσνόητη έννοια στη Java λόγω του υψηλού βαθμού αφαίρεσης που τις χαρακτηρίζει. Μια διαπροσωπεία αποτελεί ουσιαστικά *δήλωση* ότι μια κλάση διαθέτει μια συγκεκριμένη ικανότητα. Επειδή όμως στη Java σχεδόν οτιδήποτε μπορεί να θεωρηθεί αυτόνομη οντότητα (αντικείμενο), ακόμα και η δήλωση ικανότητας μιας κλάσης είναι ένα αφηρημένο αντικείμενο που μπορεί να κληρονομηθεί, να τροποιηθεί ή να ανατεθεί σε μια μεταβλητή. Είναι σαν να θεωρούμε οτί η "αυτοκίνηση" είναι άλλο ένα εξάρτημα ενός αυτοκινήτου, ένα εξάρτημα το οποίο αν θέλουμε το αφαιρούμε και το κολλάμε σε ένα άλλο αντικείμενο π.χ. σε ένα καρότσι, με τις κατάλληλες επεμβάσεις φυσικά. Στην πράξη, μια διαπροσωπεία είναι μια λίστα μεθόδων: όταν μια κλάση δηλώνει ότι υλοποιεί **(implements)** μια διαπροσωπεία, δηλώνει ότι στο σώμα της περιέχει **όλες** τις μεθόδους της διαπροσωπείας, οι οποίες της προσδίδουν τη συγκεκριμένη ικανότητα, όπως και το αυτοκίνητο-καρότσι οφείλει να διαθέτει κινητήρα, διαφορικό, τιμόνι, ότι χρειάζεται για να καταφέρει να αποκτήσει την ικανότητα της αυτοκίνησης.

Ας χρησιμοποιήσουμε όμως λίγο κώδικα με έννοιες από την καθημερινή ζωή για να κατασκευάσουμε ένα εύληπτο παράδειγμα. Έστω ότι ορίζουμε τη διαπροσωπεία `Driveable` στο σώμα κάποιας τυχαίας κλάσης. Από το όνομά της καταλαβαίνουμε ότι δηλώνει την ιδιότητα ενός αντικειμένου ότι "μπορεί να οδηγηθεί"[16]. Ο ορισμός της διαπροσωπείας γίνεται ως εξής:

```
interface Driveable {
  boolean startEngine();
  void stopEngine();
  double accelerate();
  double decelerate();
  double turn();
}
```

Όπως προλέχθηκε, η διαπροσωπεία απλά δηλώνει μια ιδιότητα και μια λίστα από μεθόδους. Παρατηρούμε κάτι σημαντικό: η διαπροσωπεία **δεν** προσδιορίζει τι κάνει η κάθε μέθοδος, αποτελεί απλά μια συλλογή ονομάτων. Το κυρίως σώμα της κάθε μεθόδου θα οριστεί από την κλάση που θα υλοποιήσει τη διαπροσωπεία. Η κλάση αυτή είναι η ακόλουθη:

```
class Automobile implements Driveable {

  public boolean startEngine() { ...  }
  public void stopEngine() { ...}
  public double accelerate() { ...  }
```

[16] Κατά λεξής: οδηγίσιμος

```
  public double decelerate() { ...  }
  public double turn() { ...  }
  ...
  boolean runningOnStreet() { ...  }
}
```

Η κλάση/αντικείμενο `Automobile` δηλώνει με τον τελεστή `implements` ότι υλοποιεί τη διαπροσωπεία `Driveable` και ως εκ τούτου "δύναται να οδηγηθεί" και το αποδεικνύει αυτό εμπεριέχοντας **όλες** τις μεθόδους που η διαπροσωπεία `Driveable` καθορίζει. Επίσης, προσδιορίζει και το τι κάνει η κάθε μέθοδος, δίνοντας έτσι ουσία στα αφηρημένα ονόματα που περιέχει η διαπροσωπεία (οι αλγόριθμοι στο σώμα κάθε μεθόδου παραλείπονται χάριν απλότητος). Εκτός από τις μεθόδους της διαπροσωπείας, η κλάση `Automobile` ορίζει και μια δικιά της μέθοδο, τη `runningOnStreets`.

Η ιδιότητα όμως "δύναμαι να οδηγηθώ" μπορεί να χαριστεί σε οποιοδήποτε άλλο αντικείμενο/κλάση θελήσουμε εμείς:

```
class Ship implements Driveable {

  public boolean startEngine() { ...  }
  public void stopEngine() { ...}
  public double accelerate() { ...  }
  public double decelerate() { ...  }
  public double turn() { ...  }
  ...
  boolean shippingOnTheSeas() { ...  }
}
```

Εδώ η κλάση `Ship` υλοποιεί και αυτή την interface `Driveable` ενσωματώνοντας τις αντίστοιχες μεθόδους. Σε αντίθεση όμως με το αυτοκίνητο που δύναται να οδηγηθεί στους δρόμους (μέθοδος `runningOnStreets()`) το πλοίο μας οδηγείται μόνο στις θάλασσες (περιέχει τη μέθοδο `shippingOnTheSeas()`).

Με τα πιο πάνω παραδείγματα καταλαβαίνουμε ότι οι κλάσεις της Java γίνεται να διαθέτουν κοινά χαρακτηριστικά και με τη βοήθεια των διαπροσωπειών, ο προγραμματιστής διαχειρίζεται τα χαρακτηριστικά αυτά σαν αφηρημένες μεν, αλλά απόλυτα διακριτές οντότητες. Επιπλέον, λόγω της αντικειμενοστρέφειας οι διαπροσωπείες μπορούν να κληρονομούνται από μια κλάση-γονιό σε μια κλάση-απόγονο ή να αποκτούν οι ίδιες, *διαπροσωπείες-απογόνους*, οι οποίες εξελίσσουν τα χαρακτηριστικά που φέρει η μητρική διαπροσωπεία προσφέροντας αναρίθμητες δυνατότητες προγραμματιστικής σχεδίασης.

Πριν κλείσουμε αξίζει να παραθέσουμε ένα παράδειγμα χρήσης μιας πραγματικής διαπροσωπείας της Java καθώς οι παραπάνω είχαν επεξηγηματικό ρόλο, με απλουστευμένα περιγραφικά ονόματα. Μια πολύ συνηθισμένη περίπτωση χρήσης τους, είναι σε γραφικά περιβάλλοντα χρηστών (GUIs) όπου τα διάφορα πλήκτρα και παράθυρα γίνονται αλληλεπιδραστικά (interactive) αντιδρώντας στις ενέργειες του χρήστη από πληκτρολόγιο ή το ποντίκι. Παρακάτω παρουσιάζουμε την Java interface `MouseMotionListener` η οποία προσδίδει σε όποια γραφική κλάση/αντικείμενο υλοποιηθεί, την ικανότητα να "νοιώθει" τις κινήσεις του κέρσορα του ποντικιού.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;


class WindowComponent extends JComponent
implements MouseMotionListener
{
  public WindowComponent (String message) {
    myOwnMessage = message;
    addMouseMotionListener(this);
  }

  public void mouseDragged (MouseEvent e) { ...  }

  public void mouseMoved (MouseEvent e) { ...  }

}
```

Η κλάση `WindowComponent`, απόγονος της `JComponent`, αφορά το περιεχόμενο ενός παραθύρου (απ'οπου και το όνομά της) και η οποία δηλώνει ότι ενσωματώνει και υλοποιεί τη διαπροσωπεία `MouseMotionListener`. Λόγω της διαπροσωπείας αυτής η κλάση οφείλει να περιέχει τις μεθόδους `mouseDragged()` και `mouseMoved()` που δέχονται σαν ορίσματα "γεγονότα"[17] που παράγει ο χρήστης με το ποντίκι του, όπως κίνηση του κέρσορα ή κλίκ και σύρσιμο μέσα στο παράθυρο. Έτσι το γραφικό περιεχόμενο του παραθύρου μας γίνεται interactive μπορεί και αντιδρά στις κινήσεις του ποντικιού. Η μέθοδος `addMouseMotionListener()` φανερώνει ότι έχουμε προσθέσει μια διαπροσωπεία τύπου Listener στην κλάση μας, μια διαπροσωπεία που δέχεται δηλαδή γεγονότα, γι' αυτό και το όρισμά της είναι η ίδια η κλάση `WindowComponent`[18].

[17] Οι μέθοδοι της Java δέχονται σαν ορίσματα ακόμα και αντικείμενα όπως τα events.
[18] Γίνεται χρήση του αυτοαναφορικού τελεστή `this`

Κεφάλαιο 3

Το περιβάλλον ανάπτυξης εφαρμογών Easy Java Simulations (EJS)

Στο κεφάλαιο αυτό θα αναλύσουμε ένα εξειδικευμένο πρόγραμμα ανάπτυξης εφαρμογών, βασισμένο στο λογισμικό ανοιχτού κώδικα του OSP project. Το εργαλείο *Easy Java Simulations* είναι ένα IDE σχεδιασμένο από τον *Fransisco Esquembre* του Πανεπιστημίου της Murcia της Ισπανίας[8], και ενσωματώνει βιβλιοθήκες κλάσεων της Java κατάλληλες για εφαρμογές φυσικής ή μαθηματικών με υποστήριξη προηγμένων γραφικών και εύκολης δικτυακής διάθεσης. Οι δυο εφαρμογές που αναπτύχθηκαν στα πλαίσια της παρούσας διπλωματικής, υλοποιήθηκαν με τη βοήθεια του συγκεκριμένου προγραμματιστικού εργαλείου.

Μια γνωστή προγραμματιστική στρατηγική για την ανάπτυξη επιστημονικών εφαρμογών, είναι η λεγόμενη **MVC** (Model - View - Control). Σύμφωνα με αυτή, ο κώδικας για το φυσικο-μαθηματικό μοντέλο, ο κώδικας για τις γραφικές αναπαραστάσεις με το γραφικό περιβάλλον χρήστη, και ο κώδικας που συνδέει τα δύο προηγούμενα, αναπτύσσονται ανεξάρτητα, έτσι ώστε σε περίπτωση που θέλουμε να τροποποιήσουμε ή να βελτιώσουμε αργότερα κάποιο τμήμα του προγράμματός μας, να μη χρειάστεί να τον ξαναγράψουμε όλο από την αρχή. Η εγγενής αντικειμενοστρέφεια της Java , και πιο συγκεκριμένα η αρχή της modularity, προάγει τη στρατηγική αυτή. Το EJS ακολουθεί την τακτική MVC στο έπακρο. Το περιβάλλον εργασίας του είναι χωρισμένο σε καρτέλες *(tabs)* και υπο-καρτέλες που επιτρέπουν στο χρήστη να αναπτύξει από την πρώτη στιγμή χωριστά τον κώδικα για το κάθε τμήμα της εφαρμογής του.

Όπως προλέχθηκε το EJS βασίζεται στη Java . Είναι γραμμένο σε Java και χρησιμοποιεί κλάσεις του OSP (οι οποίες είναι τροποποιημένες κλάσεις της Java) για να βοηθήσει το χρήστη να σχεδιάσει την εφαρμογή του. Ως εκ τούτου για να λειτουργήσει θα πρέπει στο εκάστοτε λειτουργικό σύστημα να είναι εγκατεστημένη, όχι μόνο μια πρόσφατη έκδοση της JVM, αλλά και το πακέτο βιβλιοθηκών JDK[1] που έχει χρήσιμα εργαλεία όπως ο *java compiler*. Αφού εγκατασταθεί[2], το EJS πρέπει να ρυθμιστεί ώστε να "βλέπει" το directory του JDK καθώς να οριστουν οι φάκελοι εργασίας (workspace) όπου θα αποθηκεύονται ο πηγαίος κώδικας, τα αναγκαία αρχεία για να τρέξει αναπτυσσόμενη εφαρμογή και τα εξαγόμενα .jar αρχεία που θα περιέχουν την τελική μορφή της. Αφού ολοκληρωθεί η εγκατάστασή του, η εκκίνηση γίνεται μέσω του εκτελέσιμου αρχείου `Ejs.jar`. Είναι αξιοσημείωτο ότι το περιβάλλον χρήσης του EJS μπορεί να ρυθμιστεί και στα ελληνικά.

Ακολουθούν συνοπτικά, ο τρόπος λειτουργίας και τα κυριότερα χαρακτηριστικά του EJS. Ο πλήρης οδηγός χρήσης του προγράμματος μπορεί να βρεθεί στον ιστοτόπο του EJS project υπό μορφή αρχείου .pdf [4].

3.1 Χτίζοντας το μοντέλο.

Για να μοντελοποιήσουμε ένα φυσικό φαινόμενο βρίσκουμε πρώτα ποιά μεγέθη εμπλέκονται σε αυτό και αντιστοιχίζουμε στο κάθε μέγεθος μια ή περισσότερες μεταβλητές ανάλογα με τις διαστάσεις του. Στη συνέχεια καταγράφουμε τις σχέσεις που συνδέουν τα μεγέθη/μεταβλητές μεταξύ τους και τις διαφορικές εξισώσεις που περιγράφουν την εξέλιξη (αν υφίσταται) από μια αρχική κατάσταση σε μια τελική. Η εξέλιξη μιας διαφορικής εξίσωσης σε προγραμματιστικό επίπεδο προσεγγίζεται από κάποιον αλγόριθμο, προιόν αριθμητικής ανάλυσης του εκάστοτε φαινομένου. Σε γενικές γραμμές, ο τρόπος λειτουργίας του αλγορίθμου είναι:

- Αρχικοποίηση των μεταβλητών
- Συσχετισμός τους και υπολογισμός όσων μεγεθών απορρέουν από τις αρχικές τιμές
- Βηματική αλλαγή των αρχικών τιμών των μεγεθών που μεταβάλλονται
- Υπολογισμός των συσχετιζόμενων μεγεθών
- Επανάληψη των βημάτων 3 και 4 μέχρι το σύστημα να φτάσει στην κατάσταση που έχουμε ορίσει ώς τελική

Τα παραπάνω ισχύουν βέβαια στις περιπτώσεις που το φαινόμενο εξελίσσεται μόνο του. Σε μια interactive εξομοίωση όμως ο χρήστης ενδέχεται να αλληλεπιδράσει με το σύστημα

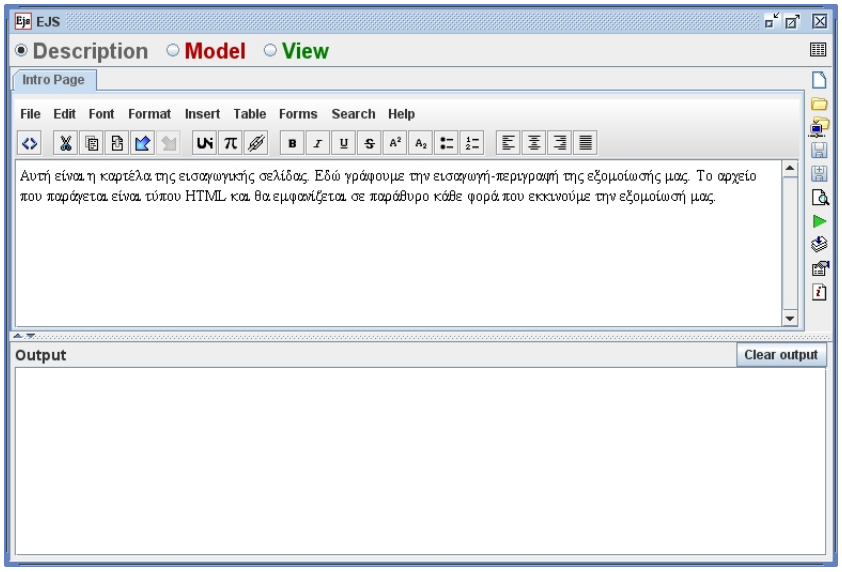
[1] Και τα δύο μπορούν να βρεθούν από τον ιστοτόπο της Sun Microsystems
[2] Πρακτικά το μόνο που χρειάζεται είναι extract σε κάποιο directory.

μεταβάλλοντας κάποιες μεταβλητές κατά τη διάρκεια κάποιου τυχαίου βήματος του αλγορίθμου. Για το λόγο αυτό η εξομοίωση πρέπει να είναι σχεδιασμένη κατάλληλα ώστε να "παγώσει" και να επανεκκινήσει τον αλγόριθμο μετά από τυχαία ενέργεια του χρήστη.

Δεδομένων των παραπάνω και προκειμένου να διευκολύνει όσο περισσότερο μπορεί το χρήστη, το περιβάλλον χρήσης του EJS είναι από τη στιγμή που εκκινείται χωρισμένο σε τρείς κύριες καρτέλες: *Description, Model* και *View*. Η πρώτη αφορά μια εισαγωγική περιγραφή που πιθανόν θέλουμε να εμφανίζει η εφαρμογή μας όταν ξεκινα. Η δεύτερη είναι χωρισμένη σε υπο-καρτέλες για τον ορισμό των μεταβλητών, των διαφορικών εξισώσεων, των σχέσεων που διέπουν το σύστημα και διαθέτει ακόμα και ελεύθερα πεδία για ανάπτυξη ελεύθερου κώδικα Java . Η τρίτη αφορά το γραφικό περιβάλλον χρήσης που θα κοσμήσει την εφαρμογή μας, και όσες γραφικές αναπαραστάσεις, animations, ή κείμενα εμφανίζονται στην οθόνη όσο αυτή λειτουργεί.

3.1.1 Εισαγωγική σελίδα, δήλωση μεταβλητών και αρχικοποίηση.

Ξεκινώντας από την καρτέλα *Description* (*'Εισαγωγή'* αν έχουμε ρυθμίσει σε ελληνικά), εκεί μπορούμε χρησιμοποιώντας τον **ενσωματωμένο** HTML editor που το EJS διαθέτει, να γράψουμε μια ιστοσελίδα με περιγραφή ή εισαγωγή για την εξομοίωση που κατασκευάζουμε. Αυτή θα ενσωματωθεί αυτόματα στην εφαρμογή μας και θα εμφανίζεται κάθε φορά που θα την εκκινούμε.



Σχημα 3.1: Η καρτέλα Description.

Αν δε μας βολεύει ο ενσωματωμένος HTML editor μπορούμε να χρησιμοποιήσουμε κάποιον της αρεσκείας μας, αρκεί μετά να μεταφέρουμε τα .html αρχεία που γράψαμε στο φάκελο-βιβλιοθήκη που θα περιέχει όλα τα απαραίτητα για την εφαρμογή που αναπτύσσουμε. Ο

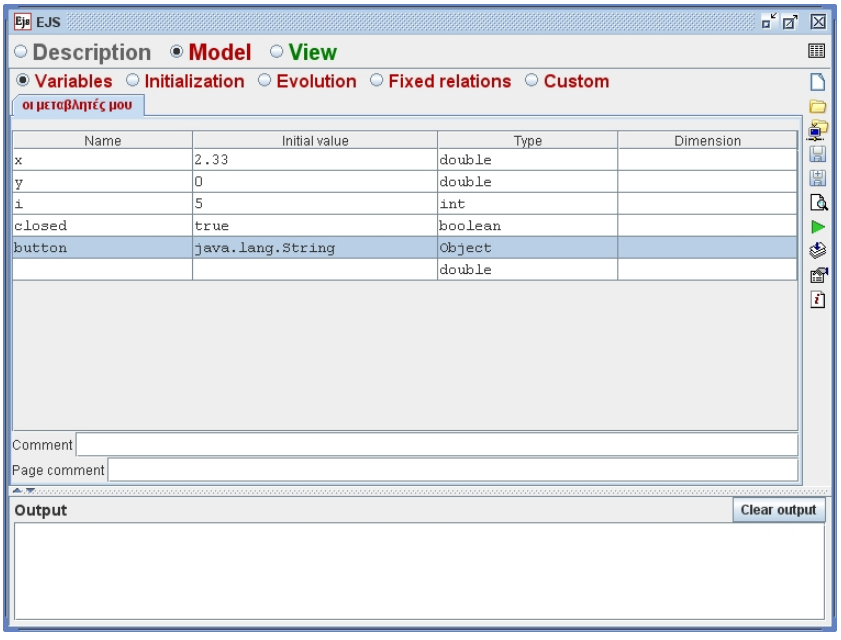
φάκελος αυτός είναι

`C:\...\EJS4.1\workspace\output\users\toOnomaTouXrhsth`

όταν βρισκόμαστε σε λειτουργικό σύστημα Windows. Όπου τελείες, το όνομα του φακέλου που περιέχει την εγκατάσταση του EJS.

Η καρτέλα **Model** αποτελεί ουσιαστικά τον προγραμματιστικό πυρήνα της εξομοίωσης, αφού εκεί έχουμε δήλωση όλων των μεταβλητών και των αλγορίθμων που της επιτρέπουν να λειτουργεί. Περιλαμβάνει τις υπο-καρτέλες Variables, Initialization, Evolution, Fixed Relations και Custom.

Στην καρτέλα **Variables** πρέπει να δηλωθούν όλες οι μεταβλητές που θα χρησιμοποιήσει το πρόγραμμά μας (είτε πρόκειται για local variables είτε για instance variables). Περιλαμβάνεται ένας πίνακας με ελεύθερα πεδία για το όνομα, τον τύπο, την αρχική τιμή τους και τις διαστάσεις τους αν πρόκειται για Java arrays. Ο τύπος της κάθε μεταβλητής επιλέγεται από το drop-down menu που ανοίγει μόλις κάνουμε κλικ στο αντίστοιχο πεδίο. Υποστηρίζονται οι τύποι boolean, int, double, string και Object. Ο τύπος Object αφορά τη δήλωση μεταβλητών-αντικειμένων, καθώς στη Java οι μέθοδοι μπορούν να παίρνουν σαν όρισμα ακόμα και αντικείμενα/κλάσεις. Για παράδειγμα μπορούμε να δηλώσουμε τη μεταβλητή color, τύπου Object με "τιμή" `java.awt.Color.RED` που είναι η κλάση που υλοποίησης του κόκκινου χρώματος. Παρατηρούμε ότι παρόλο που το πρόγραμμά μας αναπτύσσεται μέσα από φόρμες,το EJS παρέχει την ελευθερία για προχωρημένες προγραμματιστικές τεχνικές σε όποιον τις γνωρίζει.



Σχημα 3.2: Η σελίδα δήλωσης μεταβλητών στην καρτέλα του Μοντέλου.

Ένα σημείο που χρειάζεται προσοχή είναι τα ονόματα που δίνουμε στις μεταβλητές μας. Δεν επιτρέπεται να δηλωθούν μεταβλητές με ίδια ονόματα, με πρώτο γράμμα κεφαλαίο ή πρώτο χαρακτήρα το "_" Κάνοντας δεξί κλίκ στον πίνακα δήλωσης μεταβλητών εμφανίζονται επιλογές που αφορούν τη διάταξη και την επεξεργασία των πεδίων του. Τέλος, με δεξί κλίκ στην επικεφαλίδα της καρτέλας, μπορούμε να εμφανίσουμε επιπλέον πίνακες για περιπτώσεις που θέλουμε να οργανώσουμε σε κατηγορίες μεγάλο αριθμό μεταβλητών. Το EJS εμφανίζει pop-up hints (συμβουλές) όταν αφήσουμε τον κέρσορα πάνω από κάθε γραφικό στοιχείο, καθοδηγώντας μας στο τι λειτουργίες κρύβονται από πίσω.

Η αρχικοποίηση των μεταβλητών μπορεί να γίνει συμπληρώνοντας το αντίστοιχο πεδίο στον πίνακα ορισμού μεταβλητών, υπάρχει όμως η δυνατότητα να αρχικοποιηθούν και από την επόμενη καρτέλα Initialization, στην περίπτωση αυτή γράφοντας κώδικα Java . Μπορουμε μάλιστα με δεξί κλίκ να ενεργοποιήσουμε ένα βοηθητικό εργαλείο (code wizard) από το οποίο μπορούμε να διαλέξουμε μερικές από τις πιο διαδεδομένες εντολές. Γενικά καταφεύγουμε στην καρτέλα Initialization για πολύπλοκες αρχικοποιήσεις μοντέλων ή απλά όταν προτιμάμε τον παλιό καλό τρόπο. Για παράδειγμα αν χρησιμοποιούμε στο μοντέλο μας έναν πολυδιάστατο πίνακα και θέλουμε να αρχικοποιήσουμε τα όλα τα κελιά του με συγκεκριμένες τιμές θα πρέπει να γράψουμε στην καρτέλα Initialization (αφού ανοίξουμε μια καινούρια σελίδα) κάτι σαν το παρακάτω:

```
d=20;

for (int i=0; i<19; i++) {
  for (int j=0; j<19; j++) {
  grid[i][j] = -200.0 + i*d; //x-diastash -200ews300
  grid[i][j] = -200.0 + j*d; //y-diastash -200ews200
  }
}
```

Όπου αρχικοποιούμε έναν πίνακα (java array) δύο διαστάσεων με το όνομα `grid[][]`[3], δίνοντας τιμές από το -200 εως το 200 ανα 20 στα κελιά, με τέτοιο τρόπο ώστε να δημιουργηθεί ένα σύνολο σημείων διατεταγμένων στο δισδιάστατο χώρο με το κελί (1,1) να περιέχει την τιμή (-200,200), το (1,2) την τιμή (-180,200) κ.ο.κ Ένα τέτοιο πλέγμα θα μπορούσε να χρησιμοποιηθεί σαν βάση για να διατάξουμε στο χώρο γραφικά στοιχεία, π.χ. διανύσματα, που παίρνοντας τιμές από το μοντέλο μας θα περιέγραφαν ένα υποθετικό διανυσματικό πεδίο.

[3]Εννοείται πως πιο πρίν, στην καρτέλα Variables έχουμε δηλώσει την ύπαρξη και το είδος του πίνακα `grid[][]`.

3.1.2 Η ανάπτυξη και εξέλιξη του μοντέλου.

Η καρτέλα εξέλιξης είναι το πεδίο στο οποίο αναπτύσσουμε και παραμετροποιούμε τους αλγορίθμους που "κινούν" την εξομοίωσή μας. Ανοίγοντας, εμφανίζονται δυο ξεχωριστά πεδία, ένα για σύνθεση κώδικα και ένα για τους τύπους των διαφορικών εξισώσεων που πρόκειται να επιλυθούν.

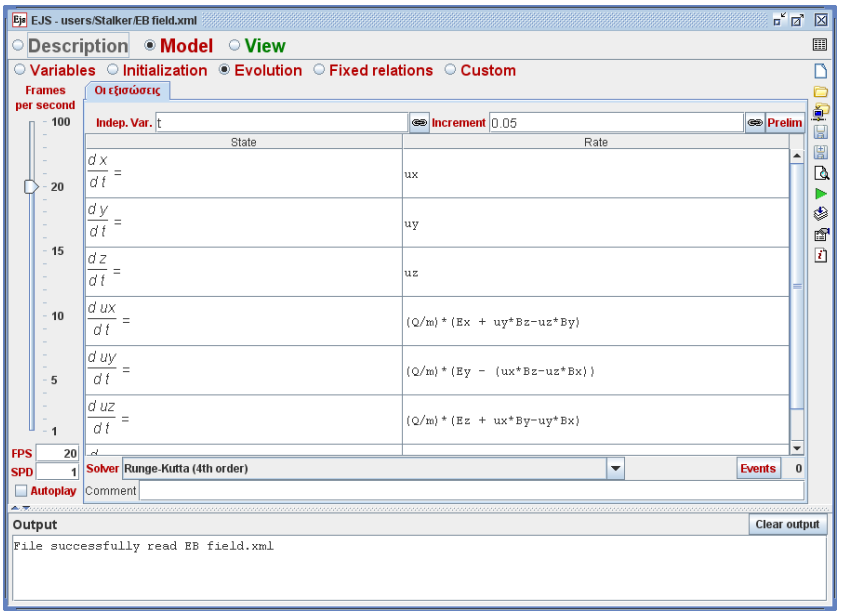
Όπως προλεχθηκε για να εξομοιωθεί ένα φυσικό σύστημα πρέπει να μοντελοποιηθεί πρώτα από έναν αριθμό εξισώσεων, συνήθως διαφορικών, οι οποίες επιλύονται υπολογιστικά με τη βοήθεια αριθμητικών μεθόδων. Οι αλγόριθμοι των αριθμητικών μεθόδων υλοποιούνται από τον πηγαίο κώδικα της εφαρμογής και οι ηλεκτρονικοί υπολογιστές αναλαμβάνουν τον όγκο των αριθμητικών πράξεων. Για τις αριθμητικές μεθόδους επίλυσης διαφορικών εξισώσεων γίνεται λόγος στο Μαθηματικό Παράρτημα στο τέλος του βιβλίου.

Στα πλαίσια του OSP project αναπτύχθηκαν αρκετές κλάσεις και διαπροσωπείες για την υλοποίηση αλγορίθμων αριθμητικής επίλυσης ΣΔΕ[4] όπως οι *ODE, ODEsolver, AbstractODE* κ.α. Εξαιτίας της εγγενούς αντικειμενοστρέφειας της γλώσσας, τα παραπάνω αντικείμενα[5] αποτελούν συστατικά που μπορούν να συνδυαστούν με πολλούς διαφορετικούς τρόπους για να προσεγγιστεί το εκάστοτε φυσικό μοντέλο. Σαν παράδειγμα από την καθημερινή ζωή μπορούμε να δώσουμε ένα σύνθετο μηχάνημα, π.χ. ένα αυτοκίνητο, στο οποίο αφού μας δωθεί ο σκελετός του (σασί) μπορούμε να προσαρμόσουμε σε αυτόν τα εξαρτήματα (μηχανή, ρόδες κτλ) που θέλουμε ανάλογα με τη χρήση του. Για να το καταφέρουμε αυτό όμως χρειαζόμαστε ένα μηχανουργείο και τις απαραίτητες μηχανολογικές γνώσεις. Το μηχανουργείο στην περίπτωσή μας είναι το περιβάλλον ανάπτυξης EJS που αυτοματοποιεί τη διαδικασία κατασκευής και συναρμολόγησης των εξαρτημάτων/κλάσεων/μεθόδων/διαπροσωπειών που θα συνθέσουν την εφαρμογή/όχημά μας.

Πιο συγκεκριμένα, και όσον αφορά τις ΣΔΕ που περιγράφουν το σύστημά μας, αυτές μπορούν να δηλωθούν είτε στη μορφή $\frac{d}{dx}(\vec{y}) = \vec{f}(x, \vec{y}(x))$ στο πανελ δήλωσης ΣΔΕ είτε να περιγραφούν από πηγαίο κώδικα σε Java στο αντίστοιχο πάνελ.

Υπάρχει η δυνατότητα να χρησιμοποιηθούν και τα δύο, ή να ανοιχτούν πολλές σελίδες δήλωσης ΣΔΕ για καλύτερη οργάνωση των εξισώσεων, σε κάθε περίπτωση πάντως στην καρτέλα της *Εξέλιξης* δηλώνουμε το μαθηματικό μοντέλο που πρόκειται να προσεγγιστεί. Ο χρήστης οφείλει να ορίσει την εξαρτημένη μεταβλητή, καθώς και το βήμα εξέλιξης της αριθμητικής μεθόδου. Επίσης ορίζεται ο ρυθμός των βημάτων ανα δευτερόλεπτο που θα εκτελεί η εξομοίωσή μας, δηλαδή πόσες φορές ο αλγόριθμός μας θα ολοκληρώνει έναν πλήρη υπολογιστικό κύκλο κάθε δευτερόλεπτο. Ο ρυθμός αυτός έχει ιδιαίτερη σημασία αν η εξομοίωσή μας περιλαμβάνει κινούμενα γραφικά που τροφοδοτούνται από τις τιμές της

[4] Συνήθεις Διαφορικές Εξισώσεις
[5] Κεφάλαιο 2



Σχημα 3.3: Το πεδίο δήλωσης των ΣΔΕ

αριθμητικής μεθόδου. Στην περίπτωση αυτή πρέπει να οριστεί τουλάχιστον στα 20-25 καρέ ανα δευτερόλεπτο (frames per second) για να προκύψει animation. Από την άλλη πλευρά ο πολύ υψηλός ρυθμός επαναλήψεων καταναλώνει υπολογιστικούς πόρους, και σε περιπτώσεις δυσεπίλυτων συστημάτων ΣΔΕ μπορεί να επιφέρει κόλλημματα στο τρέξιμο της εξομοίωσης, ειδικά σε "αδύναμους" υπολογιστές.

Κάνοντας κλικ στο πεδίο με την επιγραφή preliminary code ανοίγει ένα παράθυρο ανάπτυξης κώδικα στο οποίο μπορούμε να προσθέσουμε κώδικα Java που αφορά αριθμητικές πράξεις ή εντολές που πρέπει να εκτελεστούν πριν εκκινηθεί ο αλγόριθμος επίλυσης των ΣΔΕ.

Ομοίως αν πατήσουμε στο πλήκτρο events μπορούμε να γράψουμε κώδικα που προετοιμάζει τον αλγόριθμό μας για συμβάντα που πιθανώς να συμβούν κατά τους διαδοχικούς υπολογισμούς από μια κατάσταση του συστήματος στην επόμενη. Για παράδειγμα, έστω ότι η εξομοίωσή μας αφορά μια σφαίρα που προσκρούει σε ένα εμπόδιο και αναπηδά ελαστικά. Η αριθμητική μέθοδος υπολογίζει βηματικά τη νέα θέση/ταχύτητα της σφαίρας και θα πρέπει όταν η σφαίρα αγγίξει τον τοίχο να ενεργοποιηθεί μια δέσμη ενεργειών (στην προκειμένη περίπτωση αλλαγή προσήμου της ταχύτητας) που θα αντιστοιχεί στην ελαστική ανάκρουση. Αυτό είναι ένα συμβάν και στο πεδίο events συμπληρώνουμε τη συνθήκη που πρέπει να ικανοποιηθεί για να συμβεί ένα γεγονός καθώς και την αντίδραση του συστήματος στο γεγονός αυτό. Πρακτικά αυτό που υπολογίζεται είναι η αλλαγή προσήμου μιας συνάρτησης που περιγράφει την κατάσταση του συστήματός μας με κάποια δεδομένη παράμετρο ανοχής (tolerance factor), η οποία επίσης δηλώνεται στη φόρμα για τα συμβάντα.

Ξαναγυρνώντας τώρα στη καρτέλα της εξέλιξης, αφήσαμε για το τέλος το σημαντικότερο τμήμα που *πρέπει* να καθοριστεί για να λειτουργήσει η εξομοίωσή μας: το είδος της

αριθμητικής μεθόδου που θα επιλύσει τις διαφορικές εξισώσεις. Από το μενού επιλογών που υπάρχει στην καρτέλα ο χρήστης μπορεί να επιλέξει κάποιον από τους γνωστότερους αλγορίθμους προσέγγισης διαφορικών εξισώσεων και αυτός ενσωματώνεται αυτόματα στην εφαρμογή που αναπτύσσει. Φυσικά στο παρασκήνιο, το EJS ενσωματώνει στον πηγαίο κώδικα της αναπτυσσόμενης εφαρμογής, κλάσσεις και διαπροσωπείες από τις βιβλιοθήκες του OSP project που υλοποιούν τον συγκεκριμένο αλγόριθμο και τον συνδέει με τις μεταβλητές και τις διαφορικές εξισώσεις που έχουν δηλωθεί μέχρι εκείνη τη στιγμή, μια διεργασία που θα ήταν αρκετά χρονοβόρα για κάποιον που δεν θα διέθετε προχωρημένες προγραμματιστικές ικανότητες. Οι μαθηματικοί αλγόριθμοι που παρέχονται είναι:

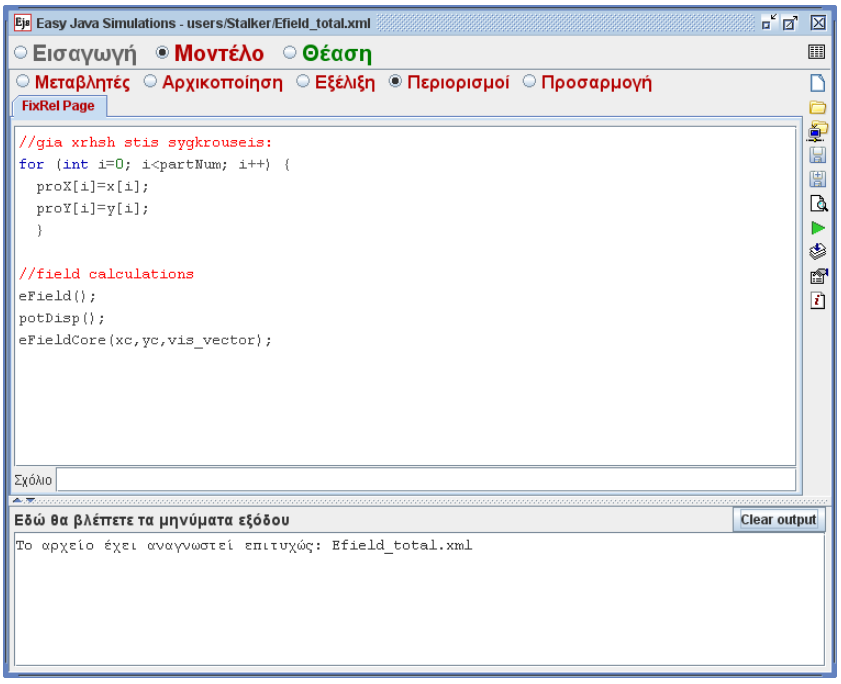
Euler	Ο κλασικός αλγόριθμος του Euler
Euler-Richardson	Αλγόριθμος σταθερού βήματος, δευτέρας τάξης
Runge-Kutta 4	Τεταρτοτάξιος αλγόριθμος R-K σταθερού βήματος
Runge-Kutta 45	Αλγόριθμος προσαρμοσμένου βήματος που χρησιμοποιεί εν παραλλήλω δύο μεθόδους R-K, μια 4ης και μια 3ης τάξης, γνωστός και ως Runge-Kutta-Fehlberg
Fehlberg 8	Αλγόριθμος 8ης τάξης που χρησιμοποιεί παρεμβολή
Dorman-Prince 5	Αλγόριθμος 5ης τάξης μεταβλητού βήματος
Dorman-Prince 853	Αλγόριθμος 8/5/3 δώδεκα σταδίων προσαρμοσμένου βήματος
Radau 5	Έμμεση μέθοδος 5ης τάξης βασισμένη σε R-K για δύσκαμπτες (stiff) διαφορικές εξισώσεις

Στις δύο εξομοιώσεις που αναπτύχθηκαν και παρουσιάζονται στο επόμενο κεφάλαιο χρησιμοποιήθηκαν οι αλγόριθμοι Euler-Richardson και Runge-Kutta-Fehlberg και αναλύονται στο Μαθηματικό Παράρτημα.

3.1.3 Οι καρτέλες Περιορισμών και Προσαρμογής

Στις δύο τελευταίες καρτέλες που υπάγονται στο Μοντέλο υπάρχουν μόνο πεδία για συμπλήρωση πηγαίου κώδικα. Το μόνο βοήθημα που προσφέρει εδώ το EJS είναι ένας code wizzard που εμφανίζει κάποιες γνωστές εντολές της Java . Είναι προφανές πώς εδώ ο χρήστης είναι ελεύθερος να αναπτύξει εκ του μηδενός κώδικα για να δημιουργήσει μεθόδους και δομές ελέγχου που δεν είναι δυνατόν να του δωθούν έτοιμες.

Στην καρτέλα των Περιορισμών Constraints γράφεται κώδικας ο οποίος εκτελείται σε κάθε βήμα της αλγοριθμικής επίλυσης, αφού πρώτα επιλυθούν οι διαφορικές εξισώσεις. Το πιο απτό παράδειγμα είναι ο υπολογισμός της κινητικής ενέργειας, ο οποίος προαπαιτεί τις τρέχουσες τιμές της ταχύτητας όπως προκύπτουν από την αριθμητική μέθοδο. Έτσι σε κάθε κύκλο, υπολογίζεται η θέση και η ταχύτητα από τον αλγόριθμό, στη συνέχεια εκτελείται ο κώδικας στη σελίδα των περιορισμών, ενημερώνονται οι αντίστοιχες μεταβλητές και ξεκινά ο



Σχημα 3.4: Η καρτέλα Περιορισμών.

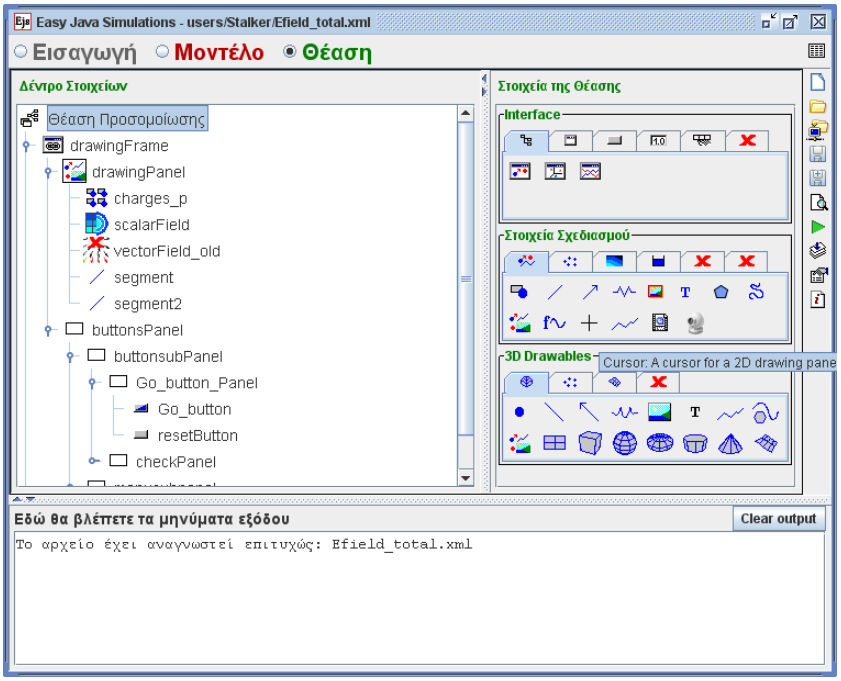
επόμενος κύκλος. Οτιδήποτε λοιπόν γραφεί στην καρτέλα αυτή, θα εκτελεστεί όσες φορές χρειαστεί να τρέξει η αριθμητική μέθοδος μέχρι να φτάσει στην συνθήκη τερματισμού.

Από την άλλη πλευρά, στην καρτέλα *Προσαρμογής* (Custom) ο χρήστης μπορεί να κατασκευάσει μεθόδους ή δέσμες ενεργειών για να τις χρησιμοποιήσει οπουδήποτε μέσα στον ηηγαίο του κώδικα. Μια συνηθισμένη περίπτωση είναι να γράφονται μέθοδοι που καλούνται από το γραφικό περιβάλλον, όταν ο χρήστης αλληλεπιδράσει με αυτό. Καταλαβαίνουμε ότι στην καρτέλα αυτή είναι δυνατόν να αναπτυχθεί ολόκληρη η εξομοίωση, χωρίς να γίνει χρήση της καρτέλας εξέλιξης, αρκεί ο χρήστης να διαθέτει το αναγκαίο προγραμματιστικό υπόβαθρο. Σε περίπτωση που χρειαστεί κλάσεις που δεν παρέχονται από το EJS είναι δυνατόν να φορτώσει τα απαραίτητα πακέτα μέσω του εικονιδίου στην πάνω δεξιά γωνία του EJS, (information about... -> import, .JAR libraries)[4].

3.2 Στήνοντας το γραφικό περιβάλλον της εφαρμογής.

Το γραφικό περιβάλλον αποτελεί ένα από τα απαιτητικότερα τμήματα κατά την ανάπτυξη μιας εφαρμογής. Χρειάζεται εις βάθος γνώση των χαρακτηριστικών της γλώσσας και προχωρημένες προγραμματιστικές τεχνικές για να στηθεί ένα αλληλεπιδραστικό interface και φυσικά αρκετός χρόνος. Καθώς το EJS προορίζεται για εκπαιδευτική χρήση, αλλά και για να διευκολύνει το έργο των προχωρημένων χρηστών, είναι εφοδιασμένο με έναν editor στον

οποίο τα γραφικά στοιχεία που θα αποτελέσουν την εφαρμογή μας επιλέγονται με "drag and drop", δηλαδή πιάσιμο με το ποντίκι και σύρσιμο στο κεντρικό πάνελ σύνθεσης.



Σχημα 3.5: Η καρτέλα view με το δενδρικό διάγραμμα και τα υλικά για το στήσιμο του γραφικού μας περιβάλλοντος

Στο πάνελ αυτό "πιάνουμε και ρίχνουμε" γραφικά αντικείμενα από το δεξιά πλαίσιο στο δενδρικό διάγραμμα που υπάρχει εκεί. Αυτά προστίθενται εκεί ιεραρχικά και ταυτόχρονα εμφανίζεται στην οθόνη μας το γραφικό περιβάλλον της εφαρμογής που κατασκευάζουμε. Έχουμε έτσι άμεση εποπτεία αυτού που κατασκευάζουμε και οποιαδήποτε τροποποίηση ή παραμετροποίηση είναι αυτόματα ορατή.

Η ιεραρχία των γραφικών συστατικών της εφαρμογής μας έχει ιδιαίτερη σημασία: δεν πρέπει να ξεχνάμε ότι καθένα από τα συστατικά που παρατίθενται στα δεξιά μενού, είναι κι από ένα αντικείμενο (Object) της Java . Πιο συγκεκριμένα είναι κλάσεις των πακέτων *java.awt* και javax.swing, βιβλιοθηκών που περιέχουν τα graphical components της Java . Ως εκ τούτου η σειρά με την οποία είναι διατεταγμένα στο δενδρικό διάγραμμα δείχνει πρώτον, τη δομή του πηγαίου κώδικα και δεύτερον τη σειρά με την οποία τα γραφικά στοιχεία διατάσσονται πάνω στην εφαρμογή μας. Εξάλλου, κάποια από τα βασικά στοιχεία αποτελούν πλατφόρμες για να στηθούν τα δευτερεύοντα, οπότε αναγκαστικά ξεκινάμε τη δόμηση από αυτά.

Έστω για παράδειγμα ότι η εφαρμογή μας θέλουμε να εμφανίζει ένα παράθυρο εφοδιασμένο με ένα πλήκτρο εκκίνησης της εξομοίωσης. Δε γίνεται να προσθέσουμε το πλήκτρο στο δενδρικό διάγραμμα αν πρώτα δεν έχουμε προσθέσει από τα συστατικά ένα γραφικό πλαίσιο, που θα υλοποιήσει το παράθυρο. Ή πάλι, δεν είναι δυνατόν να προσθέσουμε διανύσματα που θα περιέγραφαν τα μεγέθη του φυσικού μας συστήματος μέσα σε ένα παράθυρο, αν

πρώτα δεν έχουμε τοποθετήσει ένα πεδίο σχεδιασμού γραφικών αντικειμένων (είναι κι αυτό ένα από τα συστατικά). Κάνοντας δεξί κλίκ στο δενδρικό διάγραμμα έχουμε τη δυνατότητα συν τοις άλλοις, να αλλάξουμε τη διάταξη των γραφικών αντικειμένων μετακινώντας τα προς τα πάνω ή προς τα κάτω.

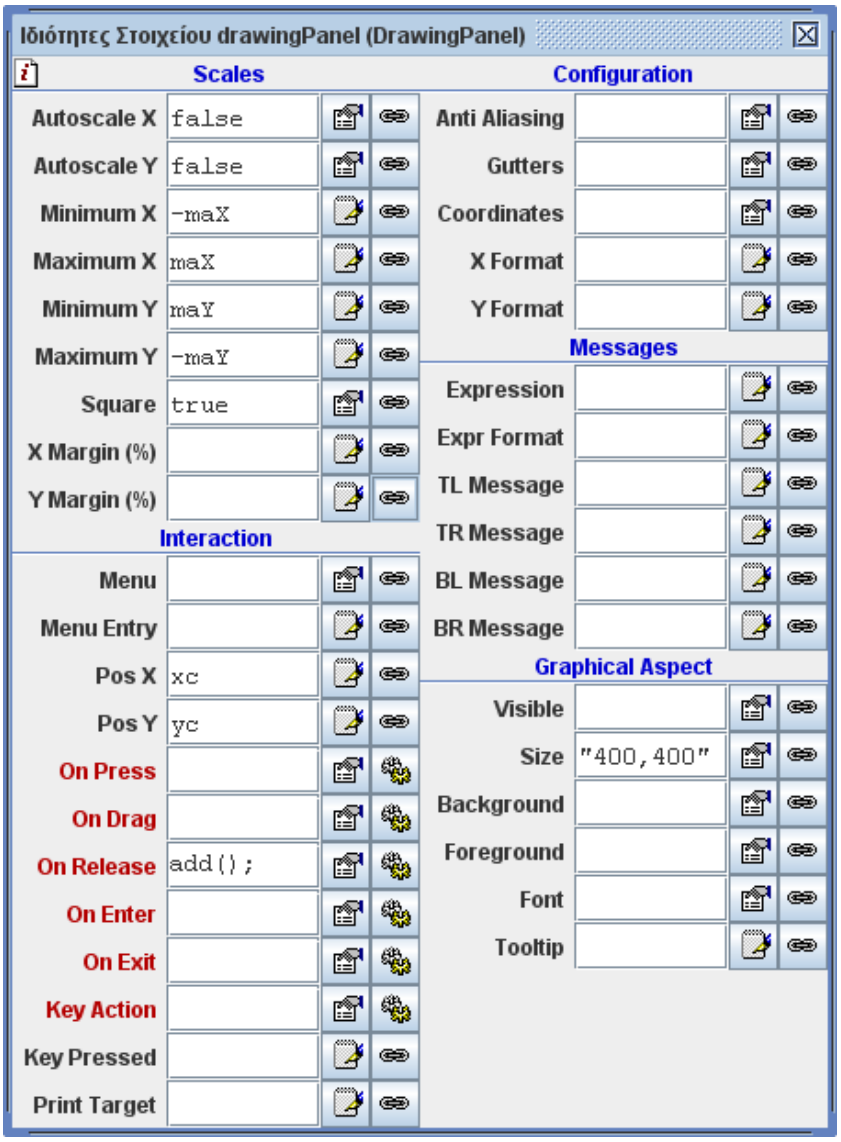
Η δεύτερη σημαντική ευκολία που προσφέρει η καρτέλα της θέασης είναι η τροποποίηση των ιδιοτήτων των γραφικών στοιχείων. Κάθε παράθυρο, πλήκτρο, μενού κτλ που θα προσθέσουμε στο γραφικό περιβάλλον της εφαρμογής διαθέτει παραμέτρους που καθορίζουν τις διαστάσεις του και τη θέση του στην οθόνη, το χρώμα, τον τίτλο που θα φέρει αλλά και τη συμπεριφορά του ως προς τον μαθηματικό πυρήνα της προσομοίωσης: ποιές μεταβλητές θα συνδέονται με αυτό, ποιά μεγέθη θα αναπαριστά ή ποιά συμπεριφορά θα εμφανίζει όταν ο χρήστης θα αλληλεπιδρά με αυτό. Όλες αυτές οι παράμετροι είναι ουσιαστικά μεταβλητές εντός των κλάσεων που υλοποιούν τα γραφικά συστατικά και αν αναπτύσσαμε την εφαρμογή σε πηγαίο κώδικα, θα έπρεπε να ρυθμιστούν και να συσχετιστούν από εκεί. Από την καρτέλα της θέασης όμως έχουμε τη δυνατότητα κάνοντας απλά διπλό κλίκ πάνω στο δενδρικό διάγραμμα, να εμφανίσουμε τη φόρμα παραμετροποίησης που αντιστοιχεί σε κάθε συστατικό.

Στο σχήμα 3.6 βλέπουμε τις ιδιότητες που αντιστοιχούν στο γραφικό στοιχείο *drawingPanel* που υλοποιεί το ένα χώρο μέσα στον οποίο μπορούμε να εντάξουμε άλλα γραφικά. Από το πάνελ αυτό ορίζουμε τις διαστάσεις του είτε δίνοντας αριθμητική τιμή, είτε γράφοντας στο αντίστοιχο πεδίο την αντίστοιχη μεταβλητή (αν έχουμε ορίσει μια για αυτό το σκοπό στην καρτέλα των μεταβλητών). Επίσης καθορίζουμε ποιές ενέργειες - μέθοδοι θα εκτελουνται, αν ο χρήστης κάνει κλίκ μέσα στο στοιχείο *drawingPanel*. Οι μέθοδοι αυτές προέρχονται είτε από την καρτέλα προσαρμογής, όπου ο χρήστης έχει ορίσει τις δικές του μεθόδους, είτε προέρχονται από τη λίστα έτοιμων μεθόδων που παρέχει το EJS, είτε τέλος δύνανται να οριστούν επιτόπου κάνοντας κλίκ στο εικονίδιο με το χέρι που βρίσκεται δίπλα. Γίνεται σιγά σιγά αντιληπτό γιατί το EJS διαθέτει το προσωνύμιο *Easy* στον τίτλο του.

3.3 Εκκινώντας την εφαρμογή μας

Αφού ολοκληρώσουμε την ανάπτυξη όλων των τμημάτων της εξομοίωσης, έρχεται επιτέλους η στιγμή να την "τρέξουμε", για να εξετάσουμε αν λειτουργεί όπως θα θέλαμε. Η εκκίνηση μιας νεόδμητης εφαρμογής του EJS μπορεί να γίνει με πολλούς τρόπους. Είτε κατευθείαν μέσα από το περιβάλλον του EJS, κάνοντας κλίκ στο μικρό πράσινο βέλος στα δεξιά εικονιδια. Είτε επιλέγοντας να κάνουμε export την εφαρμογή υπο μορφή εκτελέσιμου αρχείου `.jar` το οποίο εξάγεται στο φάκελο

```
C:\>...\EJS \workspace \export
```



Σχημα 3.6: Κάνοντας διπλό κλίκ στο στοιχείο drawingPanel του προηγούμενου δενδρικού διαγράμματος εμφανίζονται οι ιδιότητές του.

Εφόσον η εφαρμογή μας λειτουργεί χωρίς σφάλματα εκτελώντας το αρχείο `.jar` που φέρει το όνομά της, αρκεί για την εκκίνησή της. Παρόλα αυτά συνιστάται, μέχρι να σιγουρευτούμε ότι λειτουργεί απρόσκοπτα, η εκκίνησή της να γίνεται μέσα από το EJS όπου ενημερωνόμαστε για τυχόν σφάλματα από την κονσόλα του. Ο τρίτος τρόπος εκτέλεσης της εφαρμογής είναι υπο μορφή applet. Οι applets είναι εφαρμογές που εκκινούνται και λειτουργούν, μέσα από το παράθυρο ενός internet browser όπως ο Mozilla ή ο Internet Explorer. Το EJS κατασκευάζει αυτόματα καθώς αναπτύσσουμε την εφαρμογή μας αρχεία .html που την εμπεριέχουν, και τα οποία αν ανοιχτούν από οποιονδήποτε browser την εμφανίζουν στο παράθυρό τους. Απαιτείται βέβαια μια μικρή παραμετροποίηση αυτών των .html αρχείων, και πιο συγκεκριμένα των εντολών *javascript* που περιέχουν (ιδέ οδ.χρήσεως [4]).

Κεφάλαιο 4

Δύο εξομοιώσεις φυσικής: Ο Νόμος της δύναμης Lorentz. Ηλεκτρικό πεδίο και δυναμικό μεγάλου αριθμού φορτισμένων σωματιδίων

Στο τελευταίο κεφάλαιο θα αναλύσουμε τη δομή και λειτουργία δύο εξομοιώσεων που αναπτύχθηκαν εξ'ολοκλήρου στο EJS με τη βοήθεια των βιβλιοθηκών του OSP project. Πέρα από το γνωστικό υπόβαθρο περί φυσικής που παρατίθεται, οι έννοιες που συναντώνται παρακάτω έχουν παρουσιαστεί στα προηγούμενα κεφάλαια και στο Μαθηματικό Παράρτημα, οπότε αν ο αναγνώστης συναντήσει κάποια δυσκολία καλό θα ήταν να ανατρέξει εκεί. Οι γνωστότεροι συντακτικοί τύποι της Java καθώς και το σύνολο των γραφικών στοιχείων, που διαθέτει το EJS, δεν περιλήφθησαν στην παρούσα διπλωματική καθώς θα τη μετέτρεπαν σε βιβλίο "οδηγιών χρήσης", μπορούν όμως να βρεθούν από την αντίστοιχη βιβλιογραφία.

4.1 Ο Νόμος της δύναμης Lorentz.

Η δύναμη Lorentz αποτελεί τη βασικότερη απόδειξη για την ύπαρξη μαγνητικού πεδίου. Ένα φορτισμένο σωμάτιο που κινείται μέσα σε μαγνητικό πεδίο, δέχεται δύναμη της μορφής:

$$\vec{F_{mag}} = Q \cdot \left(\vec{v} \times \vec{B}\right) \tag{4.1}$$

,

όπου $\vec{v}$ η διανυσματική ταχύτητά του, Q το φορτίο του και **B** το ψευδοδιάνυσμα του μαγνητικού πεδίου, Η δύναμη αυτή ονομάστηκε έτσι προς τιμήν του μεγάλου φυσικού Hendrik Antoon Lorentz ο οποίος τη διατύπωσε στην παραπάνω μορφή[1].

Προσθέτωντας και την επίδραση ενός ηλεκτρικού πεδίου $\vec{E}$

$$\vec{F_{el}} = Q \cdot \vec{E} \tag{4.2}$$

καταλήγουμε στον Νόμο του Lorentz:

$$\vec{F_L} = Q \cdot \left[\vec{E} + \vec{v} \times \vec{B}\right] \tag{4.3}$$

Ο Νόμος της δύναμης Lorentz μαζί με τις εξισώσεις του *Maxwell* αποτελούν την πιο σύντομη διατύπωση της ηλεκτροστατικής και μαγνητοστατικής[9]. Όπως και το Νόμο της δύναμης Coulomb, έτσι το Νόμο του Lorentz τον δεχόμαστε αξιωματικά, "ώς έχει" απλά επειδή επαληθεύεται από όλα τα πειράματα.

Εξαιτίας της δύναμης Lorentz ένα κινούμενο σωμάτιο εντός μαγνητικού πεδίου εκτελεί καμπυλόγραμμη τροχιά καθώς η μαγνητική δύναμη που υφίσταται έχει διεύθυνση κάθετη στη διεύθυνση κίνησής του. Η διεύθυνση και φορά της μαγνητικής δύναμης μπορεί να βρεθεί με τον κανόνα του δεξιού χεριού (λόγω του εξωτερικού γινομένου που υπάρχει στον τύπο), τοποθετώντας τον αντίχειρα στο διάνυσμα της ταχύτητας, το δείκτη στο διάνυσμα του μαγνητικού πεδίου και τη δύναμη $\vec{F}_{mag}$ στον μεσαίο. Η λειτουργία του *κύκλοτρου*, της πρώτης μορφής κυκλικού επιταχυντή σωματίων, στηρίζεται στη δύναμη Lorentz. Επίσης θεωρώντας τη ως κεντρομόλο, η δύναμη Lorentz μας βοηθάει να υπολογίζουμε τις μάζες στοιχειωδών σωματίων, καταγράφοντας την ακτίνα των καμπυλόγραμμων τροχιών τους και συσχετίζοντάς την με ένα γνωστό μαγνητικό πεδίο με τον τύπο:

$$Q \cdot v \cdot B = m \cdot \frac{v^2}{R} \tag{4.4}$$

Στην περίπτωση όπου υπάρχει ταυτόχρονα και ηλεκτρικό πεδίο, η συνδυασμένη επίδραση των δύο δυνάμεων εξαναγκάζει το σωματίδιο να κινηθεί σε κυκλοειδή τροχιά. Στην εξομοίωσή

[1] Η εξίσωση αυτή είχε πρωτοεμφανιστεί σαν μια από τις εξισώσεις του Maxwell συναρτήσει του βαθμωτού δυναμικού φ και του διανυσματικού δυναμικού **B**.

μας θα μοντελοποιήσουμε το σύστημα ενός σωματιδίου που κινείται στον τρισδιάστατο χώρο υπο την επίδραση ομογενούς ηλεκτρικού και μαγνητικού πεδίου, αφήνοντας στην ευχέρεια του χρήστη να ορίζει τις συνιστώσες των δύο πεδίων, καταγράφοντας την τροχιά του στο χώρο, και στα επίπεδα x-y,y-z,x-z

4.1.1 Εξισώσεις και απαιτούμενες μεταβλητές

Για να μοντελοποιήσουμε το παραπάνω σύστημα θα καταφύγουμε στις διαφορικές εξισώσεις κίνησης του σωματιδίου. Από το Νόμο του Νεύτωνα και το Νόμο της δύναμης Lorentz:

$$\vec{F}_{Newt} = m \cdot \vec{a} = \vec{F}_L \tag{4.5}$$

$$F_x = m \cdot \left(\frac{\partial^2 x}{\partial t^2}\right) = Q \cdot \left[\frac{\partial y}{\partial t} \cdot B_z - \frac{\partial z}{\partial t} \cdot B_y\right] \tag{4.6}$$

$$F_y = m \cdot \left(\frac{\partial^2 y}{\partial t^2}\right) = -Q \cdot \left[\frac{\partial x}{\partial t} \cdot B_z - \frac{\partial z}{\partial t} \cdot B_x\right] \tag{4.7}$$

$$F_x = m \cdot \left(\frac{\partial^2 z}{\partial t^2}\right) = Q \cdot \left[\frac{\partial x}{\partial t} \cdot B_y - \frac{\partial y}{\partial t} \cdot B_x\right] \tag{4.8}$$

όπου έχουμε εκτελέσει τις πράξεις του εξωτερικού γινομένου και διαχωρίσαμε τις τρείς συνιστώσες της δύναμης.

Οι μεταβλητές που πρέπει οπωσδήποτε να περιλαμβάνονται στο πρόγραμμά μας είναι οι συντεταγμένες θέσης του σωματιδίου x,y,z, οι συνιστώσες της ταχύτητας u_x, u_y, u_z, της επιτάχυνσης, $\alpha_x, \alpha_y, \alpha_z$, του ηλεκτρικού πεδίου, E_x, E_y, E_z, του μαγνητικού πεδίου B_x, B_y, B_z, της συνολικής δύναμης F_{Lx}, F_{Ly}, F_{Lz} και φυσικά η μάζα m, το φορτίο Q και ο χρόνος t.

Για να προσεγγίσουμε αριθμητικά το παραπάνω σύστημα διαφορικών εξισώσεων, πρέπει να μετατρέψουμε τις διαφορικές εξισώσεις 2ου βαθμού που το αποτελούν, σε συνήθεις (βλ. Μαθ. Παράρτημα).

Θεωρώντας το χρόνο σαν εξαρτημένη μεταβλητή και θέτοντας

$$\frac{dx}{dt} = \upsilon_x, \qquad \frac{dy}{dt} = \upsilon_y, \qquad \frac{dz}{dt} = \upsilon_z \tag{4.9}$$

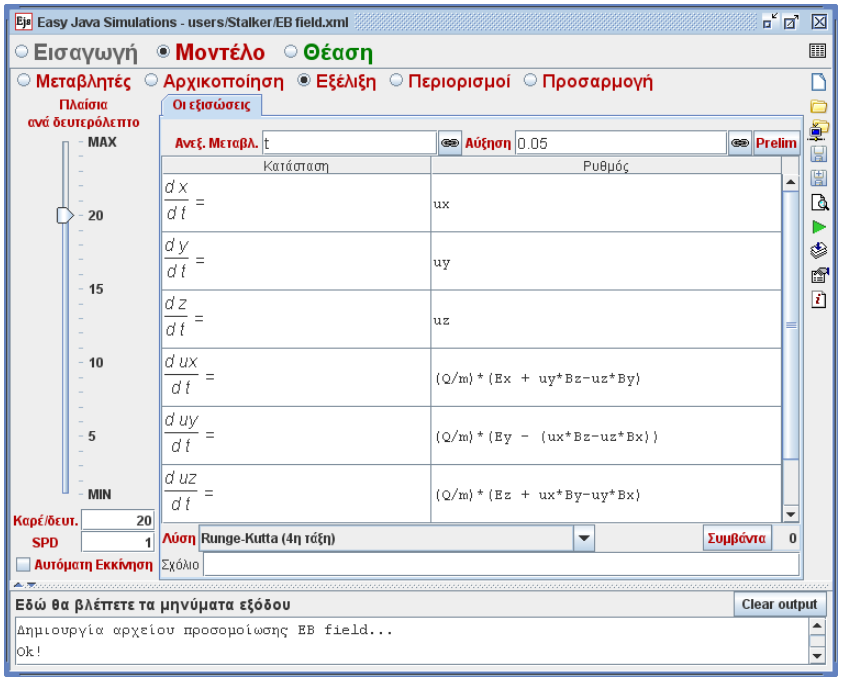
$$\frac{d\upsilon_x}{dt} = \frac{Q}{m} \cdot [\upsilon_y \cdot B_z - \upsilon_z \cdot B_y] \tag{4.10}$$

$$\frac{dv_y}{dt} = -\frac{Q}{m} \cdot [v_x \cdot B_z - v_z \cdot B_x] \tag{4.11}$$

$$\frac{dv_z}{dt} = \frac{Q}{m} \cdot [v_y \cdot B_x - v_x \cdot B_y] \tag{4.12}$$

καθώς και το προφανές $\frac{dt}{dt} = 1$, έχω καταλήξει σε ένα σύστημα από συνήθεις διαφορικές εξισώσεις οι οποίες είναι επιλύσιμες από την αριθμητική μέθοδο που θα χρησιμοποιήσω.

Αφού δηλώσω και αρχικοποιήσω τις μεταβλητές μου στο περιβάλλον ανάπτυξης του EJS, καταγράφουμε στην καρτέλα της εξέλιξης το παραπάνω σύστημα εξισώσεων και επιλέγουμε το χρονικό βήμα (Δt=0.05) και σαν αλγόριθμο αριθμητικής επίλυσης τον Runge-Kutta τέταρτης τάξης.



Σχημα 4.1: Οι διαφορικές εξισώσεις που αποτελούν το μοντέλο της εξομοίωσης Lorentz

Δεν ξεχνάμε να προσθέσουμε στη καρτέλα των Περιορισμών, κώδικα Java για τον υπολογισμό των συνιστωσών της δύναμης F_L που πραγματοποιείται αμέσως μετά την ολοκλήρωση κάθε βήματος της αριθμητικής μεθόδου.

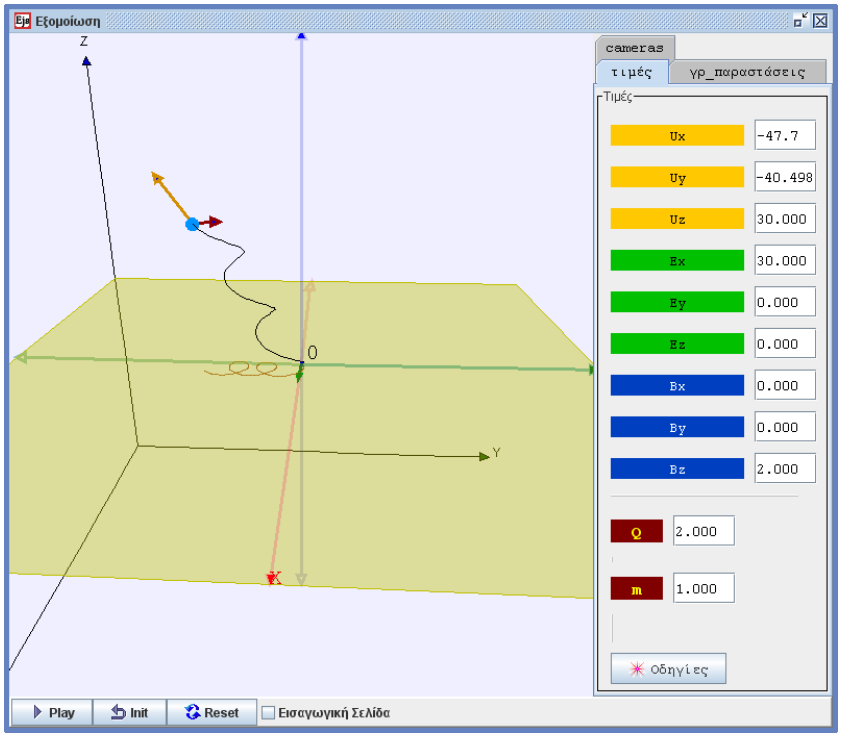
ο κώδικας έχει τη μορφή:

```
Fx = (Q)*(Ex + uy*Bz-uz*By);
Fy = (Q)*(Ey - (ux*Bz-uz*Bx));
Fz = (Q)*(Ez + ux*By-uy*Bx);
```

Οι τιμές όλων των υπολογιζόμενων μεταβλητών συσχετίζονται σε πραγματικό χρόνο με τα γραφικά στοιχεία που συνθέτουν την Θέαση της εφαρμογής μου.

4.1.2 Το γραφικό περιβάλλον της εξομοίωσης.

Η εξομοίωση του Νόμου του Lorentz εφοδιάστηκε με χρήση του EJS με γραφικό περιβάλλον χρήσης (GUI) στο οποίο βλέπουμε σε πραγματικό χρόνο τη θέση και τροχιά του σωματιδίου σε ένα τρισδιάστατο σύστημα αξόνων.



Σχημα 4.2: Το γραφικό περιβάλλον χρήσης GUI της εξομοίωσης που αναπτύξαμε

Η εφαρμογή μας διαθέτει πλήκτρα για εκκίνηση,αρχικοποίηση και παύση του εικονικού πειράματος, καθώς και ένα checkbox το οποίο ενεργοποιεί την εισαγωγική σελίδα. Η σελίδα αυτή ανοίγει αυτόματα με την εκκίνηση του πειράματος και περιγράφει τη φυσική και τις λειτουργίες της εξομοίωσης.

Στο κεντρικό πάνελ αναπαρίσταται το τρισδιάστατο σύστημα αξόνων μέσα στο οποίο διαγράφεται η τροχιά του σωματιδίου και διακρίνονται τα διανύσματα των κρίσιμων για την εξομοίωση μεγεθών: της ταχύτητας (πορτοκαλί διάνυσμα), του μαγνητικού πεδίου (μπλέ), του ηλεκτρικού πεδίου (πράσινο) και της συνολικής δύναμης Lorentz (καφέ). Μπορούμε επίσης να διακρίνουμε την προβολή της τροχιάς του σωματιδίου, στο x-y επίπεδο. Με τη βοήθεια του ποντικιού, το τρισδιάστατο γράφημα γίνεται να περιστραφεί κατά βούληση, για να επιλέξουμε την καταλληλότερη οπτική γωνία.

Στα δεξιά, υπάρχουν τρείς καρτέλες (τιμές, γραφικές παραστάσεις, κάμερες), από τις οποίες ο χρήστης παραμετροποιεί το εικονικό του πείραμα. Υπάρχουν πεδία συμπλήρωσης τιμών για κάθε συνιστώσα της ταχύτητας, του μαγνητικού και του ηλεκτρικού πεδίου, όπως επίσης και του φορτίου και της μάζας του σωματιδίου. Υπάρχει ακόμα και ένα πλήκτρο που ενεργοποιεί μια σελίδα με οδηγίες χρήσης για τη καρτέλα συμπλήρωσης τιμών. Στην καρτέλα των γραφικών παραστάσεων ο χρήστης μπορεί να ενεργοποιήσει επιπλέον συστήματα αξόνων, στις δύο διαστάσεις αυτή τη φορά, που αντιστοιχούν στα επίπεδα x-y, y-z, x-z για να παρακολουθεί την προβολή της τροχιάς του σωματιδίου στο επίπεδο που επιθυμεί.

Παρατηρώντας την εξομοίωση του Νόμου του Lorentz καταλαβαίνουμε ότι παρά το λιτό και εύχρηστο περιβάλλον του, το πρόγραμμα ανάπτυξης εφαρμογών EJS διαθέτει δυνατότητες εφάμιλλες των μεγάλων επαγγελματικών διανομών.

4.2 Ηλεκτρικό πεδίο και δυναμικό μεγάλου αριθμού φορτισμένων σωματιδίων.

Η δεύτερη εξομοίωση που αναπτύχθηκε με τη βοήθεια του λογισμικού OSP/EJS αφορά το Νόμο που μαζί με την αρχή της επαλληλίας αποτελούν το θεμέλιο της ηλεκτροστατικής: το Νόμο της δύναμης *Coulomb*. Περιγράφει την αλληλεπίδραση μεταξύ ακίνητων σημειακών φορτίων [2] και από αυτόν προέκυψαν οι άλλες σημαντικές αρχές της ηλεκτροστατικής όπως ο Νόμος του Gauss και το δυναμικό.

$$\vec{F} = \frac{1}{4\pi\varepsilon_o} \frac{q \cdot Q}{|\vec{r}_Q - \vec{r}_q|^2} \cdot (\vec{r}_Q \hat{-} \vec{r}_q) \tag{4.13}$$

με μοναδιαίο διάνυσμα ίσο με

$$(\vec{r}_Q \hat{-} \vec{r}_q) = \frac{\vec{r}_Q - \vec{r}_q}{|\vec{r}_Q - \vec{r}_q|} \tag{4.14}$$

όπου η απόσταση $\vec{r}$ που χωρίζει τα δύο σημεία δίνεται συναρτήσει των διανυσμάτων θέσης $\vec{r}_q, \vec{r}_Q$ από την αρχή των αξόνων.

Διατυπωμένος με λόγια ο Νόμος του Coulomb μας λέει ότι η δύναμη που ασκείται από ένα ακίνητο σημειακό φορτίο q_1 σε ένα άλλο ακίνητο σημειακό φορτίο Q που βρίσκεται σε απόσταση r είναι ανάλογη του γινομένου των δύο φορτίων και αντιστρόφως ανάλογη του τετραγώνου της απόστασης r. Η διεύθυνσή της βρίσκεται επι της ευθείας που ενώνει τα δύο σημεία και η φορά της εξαρτάται από το πρόσημο των δύο φορτίων: ελκτική όταν

[2] Εξ ου και η έννοια *ηλεκτροστατική*

είναι ετερώνυμα, απωστική για ομώνυμα. Στον τύπο υπεισέρχεται στη σταθερά αναλογίας η διηλεκτρική σταθερά του κενού: $\epsilon_0 = 8.85 \cdot 10^{-12} \frac{C^2}{N \cdot m^2}$.[9]

Όταν έχουμε μια κατανομή ακίνητων σημειακών φορτίων q_i σε ένα σημείο του χώρου, τότε η δύναμη που ασκείται σε φορτίο Q που βρίσκεται σε απόσταση $\vec{r} = \vec{r}_Q - \vec{r}_q$ από αυτά δίνεται από την αρχή της επαλληλίας:

$$\vec{F}_{tot} = \frac{Q}{4\pi\varepsilon_o} \sum_{i=1}^{n} q_i \frac{\hat{r}_i}{|\vec{r}_i|^2} \tag{4.15}$$

Η συνολική ηλεκτροστατική δύναμη δηλαδή ισούται με το άθροισμα όλων των επιμέρους δυνάμεων στο φορτίο Q.

Ο όρος $\frac{1}{4\pi\varepsilon_o} \sum_{i=1}^{n} q_i \frac{\hat{r}_i}{|\vec{r}_i|^2}$ είναι ουσιαστικά το ηλεκτρικό πεδίο $\vec{E}$ που δημιουργείται από την κατανομή των ακίνητων φορτίων στο σημείο που βρίσκεται το Q.

δηλαδή

$$\vec{F}_{tot} = Q \cdot \vec{E} \tag{4.16}$$

Το ηλεκτρικό πεδίο είναι διανυσματικό μέγεθος και ερμηνεύεται σαν δύναμη ανά μονάδα φορτίου που υφίσταται σε ένα σημείο του χώρου εξαιτίας μιας κατανομής φορτίων.Ως τις μέρες μας η φύση του δεν έχει διευκρινιστεί με σαφήνεια. Για να περιγραφεί γραφικά συνήθως χρησιμοποιούνται δυναμικές γραμμές στις οποίες το διάνυσμα του πεδίου είναι πάντα εφαπτόμενο. Στην παρακάτω εξομοίωση, επειδή εμφανίζεται μεγάλος αριθμός σωματιδίων στη οθόνη, επιλέχθηκε η οπτικοποίηση του ηλεκτρικού πεδίου από ένα σύνολο διατεγμένων στο χώρο διανυσμάτων.

Το ηλεκτρικό πεδί αρουσιάζει την ιδιότητα, ο στροβιλισμός του να είναι πάντα μηδέν (ακτινικό πεδίο). Η παραπάνω ιδιότητα προκύπτει αν υπολογίσουμε το επικαμπύλιο ολοκλήρωμα του $\vec{E}$ κατά μήκος κλειστού βρόχου (μηδέν) και εφαρμοσουμε το θεώρημα Stokes σύμφωνα με το οποίο:

$$\int_{surf} (\nabla \times \vec{v}) \cdot d\alpha = \oint_{bound} \vec{v} \cdot d\vec{l} \tag{4.17}$$

το επιφανειακό ολοκλήρωμα μιας παραγώγου ισούται με την τιμή της συναρτήσεως στο σύνορο. Οπότε εφόσον $\oint_{bound} \vec{E} \cdot d\vec{l} = 0$ συνεπάγεται ότι $\nabla \times \vec{E} = 0$. Από το θεώρημα για τα αστρόβιλα πεδία μπορούμε λοιπόν να πούμε ότι αφού $\nabla \times \vec{E} = 0$ το ηλεκτρικό πεδίο θα πρέπει να προκύπτει από την κλίση ενός βαθμωτού μεγέθους $\vec{E} = -\nabla U$. Ολοκληρώνοντας

και δεδομένου ότι το επικαμπύλιο ολοκλήρωμα του είναι ανεξάρτητο διαδρομής ορίζουμε τη βαθμωτή συνάρτηση $V(P)$ όπου

$$V(P) = -\int_{O}^{P} \vec{E} \cdot d\vec{l} \tag{4.18}$$

όπου **O** κάποιο σταθερό σημείο αναφοράς. Η συνάρτηση αυτή είναι το ηλεκτρικό δυναμικό και μας βοηθά να επιλύουμε τα διάφορα προβλήματα της ηλεκτροστατικής, χωρίς να μπλέκουμε με διανυσματικά μεγέθη. Η διαφορά δυναμικού ανάμεσα σε ένα σημείο που έχουμε ορίσει ως V=0 και ενός σημείου πλησίο ενός φορτίου q ισούται με την ενέργεια που θα πρέπει να προσδώσουμε, η να πάρουμε σε ένα άλλο φορτίο Q για να το μεταφέρουμε από το άπειρο κοντά στο q ή το αντίθετο.

Το δυναμικό υπακούει και αυτό στην αρχή της επαλληλίας και για απόσταση r από ένα σημειακό φορτίο q δίνεται από τον τύπο:

$$V(r) = -\int_{\infty}^{r} \vec{E} \cdot d\vec{l} = \frac{1}{4\pi\epsilon_0} \cdot \frac{q}{r} \tag{4.19}$$

όπου θεωρήσαμε σαν σημείο μηδενικού δυναμικού το άπειρο.

4.2.1 Μεταβλητές και διαφορικές εξισώσεις

Στην εξομοίωσή μας ο χρήστης "παράγει" μέσα σε ένα δισδιάστατο χώρο έναν αριθμό φορτισμένων σωματιδίων και στη συνέχεια παρακολουθεί πώς εξελίσσεται το σύστημα. Ο αριθμός των σωματιδίων και το φορτίο τους καθορίζονται από το χρήστη. Το πρόγραμμα παρέχει επιλογές που εμφανίζουν στο χώρο αλληλεπίδρασης των σωματιδίων το ηλεκτρικό πεδίο που παράγει κάθε στιγμή η κατανομή τους στο χώρο, και το δυναμικό σε δύο ή τρείς διαστάσεις.

Για να μοντελοποιηθεί το παραπάνω σύστημα εξήχθησαν οι εξισώσεις κίνησης των σωματιδίων, μέσω του νόμου του Νεύτωνα και του Νόμου της δύναμης Coulomb. Όπως και στη προηγούμενη προσομοιωση οι εξισώσεις 2ου βαθμού αναλύθηκαν σε συνιστώσες και το σύστημα μετατράπηκε σε σύστημα ΣΔΕ προκειμένου να επιλυθεί από την αριθμητική μέθοδο.

Οι εξισώσεις που τελικά εισήχθησαν στη καρτέλα εξέλιξης του EJS είναι:

$$\frac{dx_i}{dt} = \upsilon_{xi}, \qquad \frac{dy_i}{dt} = \upsilon_{yi} \tag{4.20}$$

$$\frac{dv_{xi}}{dt} = K \cdot \sum_{i=1}^{n} \frac{q_i q_n \cdot r_x}{|r|^3} = F_{xi} \tag{4.21}$$

$$\frac{dv_{xi}}{dt} = K \cdot \sum_{i=1}^{n} \frac{q_i q_n \cdot r_y}{|r|^3} = F_{yi} \tag{4.22}$$

με τα i,n να παίρνουν τιμές από τον τρέχοντα αριθμό σωματιδίων στη οθόνη και $|r| = (x_i - x_n)^2 + (y_i - y_n)^2 \qquad r_x = x_i - xn, \qquad r_y = y_i - y_n$

Αυτό που ο αλγόριθμος καλείται να φέρει εις πέρας, είναι να υπολογίσει τις συνιστώσες (Fx,Fy) της δύναμης Coulomb σε κάθε σωματίδιο, όπως αυτές προέρχονται από την αλληλεπίδραση με όλα τα υπόλοιπα σωματίδια του χώρου (αρχή επαλληλίας) και μέσω της επιτάχυνσης να προσδώσει τιμές για την ταχύτητα και τη θέση του. Όλη αυτή η διαδικασία πρέπει να επαναληφθεί για κάθε σωματίδιο που υπάρχει στο χώρο. Στη συνέχεια μέσω μεθόδων που έχουν δηλωθεί στην καρτέλα προσαρμογής γίνεται το ίδιο για τις συνιστώσες του ηλεκτρικού πεδίου και του δυναμικού. Σαν αριθμητική μέθοδος επίλυσης επιλέχθηκε η Euler-Richardson καθώς λόγω του μεγάλου αριθμού σωματιδίων στο χώρο (εως 100) οι υπολογιστικοί πόροι καθίστανται πολύτιμοι.

4.2.2 Εις τα ενδότερα: δομές δεδομένων και ελέγχου.

Για να υλοποιηθεί το παραπάνω μοντέλο σε περιβάλλον Java έγινε ευρεία χρήση πολυδιάστατων πινάκων οι οποίοι αποθηκεύουν τις συνιστώσες των μεγεθών που εμπλέκονται. Η εξομοίωση σχεδιάστηκε ώστε να απεικονίζει το πολύ εκατό σωματίδια (θετικά και αρνητικά) στην οθόνη. Η μάζα και το φορτίο αποθηκεύτηκαν λοιπόν σε αντίστοιχους πίνακες, με μια ακέραια μεταβλητή (τον τρέχοντα αριθμό των σωματιδίων) να αποτελεί το γενικό δείκτη των θέσεων των παραπάνω πινάκων. Για παράδειγμα, στη σελίδα αρχικοποίησης του μοντέλου, περιέχεται κώδικας της μορφής:

```
partNum = 0;
for (int i=0; i<maxparts; i++) {
  if (chargeselector == true) { //positive case
    size[i] = 10.0;
    chargeliszt[i] = 1.0;
    color[i] = java.awt.Color.red;


  }
  else {  //negative case
```

```
    size[i] = 10.0;
    chargeliszt[i] = -1.0;
    color[i] = java.awt.Color.green;
  }
  togglevis[i] = i < partNum;

//tyxaies arxikes theseis
x[i] = 200*Math.random()- 200*Math.random();
y[i] = 200*Math.random()- 200*Math.random();

//tyxaies arxikes taxythtes
vx[i] = 10.0*(Math.random()-0.5);
vy[i] = 10.0*(Math.random()-0.5);

}
```

Αφού αρχικοποιηθεί η μεταβλητή του τρέχοντα αριθμού σωματιδίων, αρχικοποιούνται με επαναληπτικούς βρόχους και οι πίνακες που περιέχουν το φορτίο (`chargeliszt[]`),το μέγεθος του γραφικού στοιχείου που περιγράφει το σωματίδιο (`size[]`), και το χρώμα του κάθε γραφικού στοιχείου (`color[]`, πρόκειται για πίνακες αντικειμένων). Επίσης αρχικοποιείται ένας boolean πίνακας (`togglevis[]`) που δεν επιτρέπει να εμφανίζονται στην οθόνη παραπάνω γραφικά στοιχεία/σωματίδια από τον επιτρεπόμενο αριθμό. Τέλος δίνονται τυχαίες θέσεις και ταχύτητες στα σωματίδια που πρωτοεμφανίζονται στην οθόνη.

Στην καρτέλα της εξέλιξης πέραν των εξισώσεων για την ταχύτητα που συμπληρώθηκαν στο ειδικό πεδίο στη μορφή $\frac{dx[i]}{dt} = \upsilon_x[i]$ οι τύποι για τις εξισώσεις της επιτάχυνσης/δύναμης δεν περιελήφθησαν εκεί. Αντί αυτού η κάθε παράγωγος της ταχύτητας εξισώθηκε με μια μέθοδο $\frac{dv_x[i]}{dt} = \frac{eForce(i,x,y,boolean)}{mass}$ το σώμα της οποίας δηλώνεται στην καρτέλα της Προσαρμογής. Η μέθοδος `eForce()` υπολογίζει τη δύναμη Coulomb ανα σωματίδιο λόγω της αλληλεπίδρασής του με όλα τα υπόλοιπα, αθροίζοντας όλες τις συνιστώσες:

```
private double eForce (int scannedpart, double[] xi, double[] yi, boolean xy)
{//CAUTION!
  double eForce = 0.0, r2;
  double rootr2;


  if (scannedpart >= partNum ) return 0.0; //exception handler
    for (int i=0; i<partNum; i++) {

    if (i==scannedpart) continue;
    r2 = (xi[i]-xi[scannedpart])*(xi[i]-xi[scannedpart])+
```

```
    (yi[i]-yi[scannedpart])*(yi[i]-yi[scannedpart]);
    rootr2 = Math.pow(r2,0.5); //!!

    //threshold exception needed ?  YES!!
    if (r2<limit) continue;
    if (xy) // xy=true -> xCoord / xy=false-> yCoord
      eForce += 1000*(chargeliszt[i]*chargeliszt[scannedpart]*
      (xi[scannedpart]-xi[i])/ (r2*rootr2));
    else
      eForce += 1000*(chargeliszt[i]*chargeliszt[scannedpart]*
      (yi[scannedpart]-yi[i])/ (r2*rootr2));
    //System.out.println(eForce);
  }
  return eForce;

}
```

Στη δεύτερη σελίδα της καρτέλας Evolution, υπάρχουν συνθήκες, που αφορούν την αλληλεπίδραση των σωματιδίων με τα όρια του δισδιάστατου χώρου μέσα στον οποίο κινούνται και οποία εξετάζονται σε κάθε κύκλο υπολογισμών του αλγορίθμου. Οι συνθήκες:

```
//kroush me screen bounds
for (int i=0; i<partNum; i++) {
if (x[i]>maX) { x[i] = maX+(maX-x[i]); vx[i] = -vx[i]; }
if (x[i]<miX) { x[i] = miX+(miX-x[i]); vx[i] = -vx[i]; }
if (y[i]>maY) { y[i] = maY+(maY-y[i]); vy[i] = -vy[i]; }
if (y[i]<miY) { y[i] = miY+(miY-y[i]); vy[i] = -vy[i]; }
 }
```

μας λένε ότι όταν η τεταγμένη ή η τετμημένη καθενός σωματιδίου ξεπεράσει το χώρο που ορίζουν οι μεταβλητές `maX, maY, miX, miY` οι αντίστοιχες συνιστώσες της ταχύτητας αλλάζουν πρόσημο. Τα σωματίδια αλληλεπιδρούν λοιπόν **ελαστικά** με τα τοιχώματα. Για το λόγο αυτό και καθώς τα σωματίδια διαθέτουν μια τυχαία αρχική ταχύτητα, (αρχικοποίηση) πέραν των επιταχύνσεων που αναπτύσσονται λόγω της δύναμης Coulomb, θα βρίσκονται σε μια μονίμη υπερκινητική κατάσταση, που δυσκολεύει την απεικόνιση του ηλεκτρικού πεδίου και του δυναμικού. Έτσι εισήχθη μια παράμετρος επιβράδυνσης στην κίνησή τους, η οποία μειώνει με το χρόνο την ταχύτητά τους. Αυτό έχει σαν αποτέλεσμα τα σωμάτια να χάνουν σιγά σιγά την αρχική ταχύτητά τους και να κινούνται, από κάποιο χρονικό σημείο και μετά, μόνο λόγω της αλληλεπίδρασης Coulomb. Με αυτό το τέχνασμα επιτρέπουμε στο σύστημα να φτάσει σε μια "κατάσταση ισορροπίας" όπου τα φορτισμένα σωματίδια κατανέμονται στο

χώρο **ομογενώς**, έχοντας μεταξύ τους τις μέγιστες δυνατές αποστάσεις[3], με βάση τη φυσική αρχή της ελάχιστης διαδρομής/αλληλεπίδρασης. Στο παρακάτω κομμάτι κώδικα η παράμετρος επιβράδυνσης `slow` μειώνει σταδιακά τις τρέχουσες τιμές των συνιστωσών της ταχύτητας.

```
//epibradynsh gia stamathma
for (int i=0; i<partNum; i++) {
vx[i] *= (1.0-slow);
vy[i] *= (1.0-slow);
 }
```

Μετά την καρτέλα της Εξέλιξης, ακολουθεί η εξίσου σημαντική καρτέλα των Περιορισμών (Constraints tab) στη οποία ορίζονται συμπληρωματικές σχέσεις που ο αλγόριθμος οφείλει να εκτελέσει σε κάθε του βήμα πρίν προχωρήσει στο επόμενο. Εδώ βρίσκονται φυσικά οι δύο μέθοδοι που υπολογίζουν το ηλεκτρικό πεδίο και το δυναμικό (όχι οι ορισμοί τους) `eField(), potDisp()` καθώς επίσης και μια μικρή αλλά απαραίτητη συνθήκη που αποθηκεύει τις τρέχουσες συντεταγμένες κάθε σωματίου για χρήση που θα αναλύσουμε παρακάτω.

```
//gia xrhsh stis sygkrouseis:
for (int i=0; i<partNum; i++) {
proX[i]=x[i];
proY[i]=y[i];
 }
```

Όλες οι μέθοδοι java που κατασκευάστηκαν για την εξομοίωση περιέχονται στην τελευταία καρτέλα του μοντέλου, αυτή της Προσαρμογής (Custom). Εκτός από τη μέθοδο `eForce()` που αναλύθηκε πιο πριν, εδώ ορίζονται οι μέθοδοι `eField()` και `potDisp()` υπεύθυνες για την περιγραφή του διανυσματικού πεδίου και του δυναμικού. Η απεικόνιση του ηλεκτρικού πεδίου είναι στη πραγματικότητα αποτέλεσμα δύο μεθόδων, φωλιασμένων η μία μέσα στην άλλη:

```
public void eFieldCore(double xi,double yi, boolean vec) {
  Ex=0.0;
  Ey=0.0;
  if (vec) {
    for (int k=0; k<partNum; k++) { //particle scan loop
      r2 = (x[k]-xi)*(x[k]-xi)+(y[k]-yi)*(y[k]-yi);
      //System.out.println(r2);
      if (r2<limit) continue;
```

[3] Αν είναι ομώνυμα φορτισμένα, ειδάλλως σχηματίζουν ζεύγη θετικού-αρνητικού που επίσης κατανέμονται ομοιόμορφα στο χώρο.

```
      Ex += 10.0*chargeliszt[k]*(xi-x[k])/r2;
      Ey += 10.0*chargeliszt[k]*(yi-y[k])/r2;
      }
    }
 }
```

Η μέθοδος `eFieldCore()` είναι ένας υπολογιστικός πυρήνας που καλείται 21×21 φορές από τη μέθοδο eField(), της οποίας ο κώδικας παρατίθεται παρακάτω, προκειμένου να να υπολογιστούν οι συνιστώσες, το μέτρο και η διεύθυνση ενός συστήματος από 21×21 διατεταγμένα διανυσματα (γραφικά στοιχεία) μέσα στο χώρο που απεικονίζονται τα σωματίδια. Η `eFieldCore()` παίρνει σαν όρισμα τις συντεταγμένες ενός σημείου και υπολογίζει στο σημείο **αυτό** τις δύο συνιστώσες του ηλεκτρικού πεδίου, όπως αυτές επάγονται εκεί εξαιτίας όλων των κατανεμημένων φορτίων στο χώρο (4.2).

```
public void eField() {
  if (visvector) {
    for (int i=0; i<21; i++) { //x coord loop
      for (int j=0; j<21; j++) { //y coord loop
      eFieldCore(Evgrid[i][j][0],Evgrid[i][j][1],visvector);
      double r = Math.sqrt(Ex*Ex+Ey*Ey);
      Evgrid[i][j][2] = Ex/r;
      Evgrid[i][j][3] = Ey/r;
      Evgrid[i][j][4] = r;
      }
    }
  }
 }
```

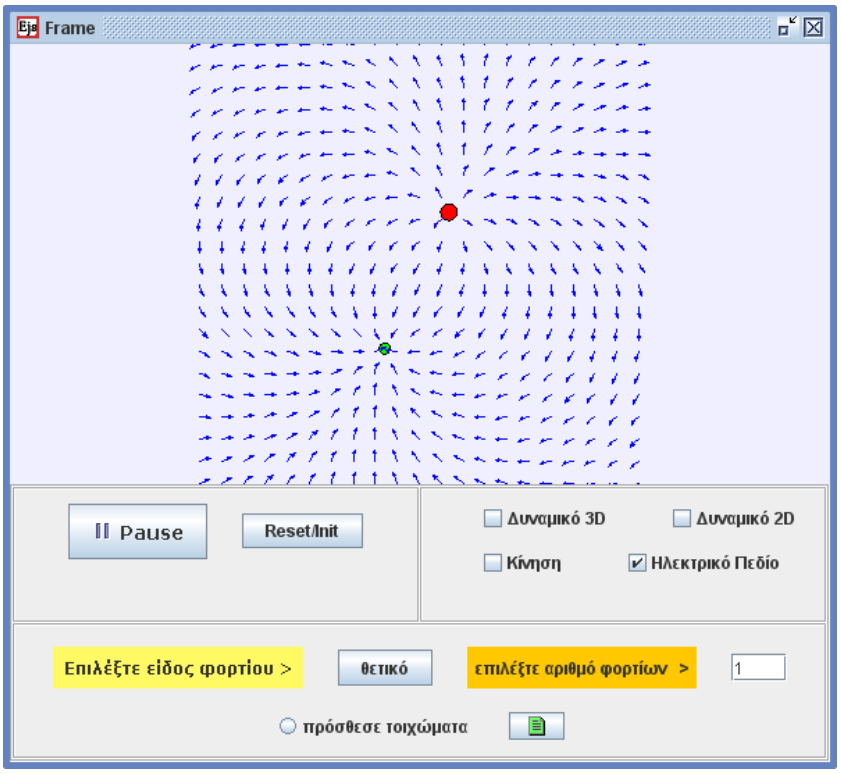
Η μέθοδος `eFieldCore()` καλείται στο εσωτερικό της μεθόδου `eField()` η οποία είδαμε πιο πάνω ότι είναι δηλωμένη στην καρτέλα των Περιορισμών. Η `eField()`, χρησιμοποιώντας έναν διπλό επαναληπτικό βρόχο παίρνει όλες τις συντεταγμένες των θέσεων των διανυσμάτων του ηλεκτρικού πεδίου και τις εισάγει στην `eFieldCore()`, η οποία υπολογίζει το ηλεκτρικό πεδίο στη συγκεκριμένη θέση και επιστρέφει τις δυο συνιστώσες του, Ex, Ey. Υπολογίζονται οι όροι $\frac{E_x}{r}, \frac{E_y}{r}, r$ όπου r το μέτρο του ηλεκτρικού πεδίου, και όλα τα δεδομένα αποθηκεύονται στον πολυδιάστατο πίνακα `Evgrid[21][21][5]`. Προφανώς οι δύο πρώτες διαστάσεις του αφορούν τις συντεταγμένες θέσεων των διανυσμάτων. Στην τρίτη διάσταση, μήκους πέντε θέσεων, αποθηκεύονται οι τιμές των $E_x, E_y, \frac{E_x}{r}, \frac{E_y}{r}, r$. Η επιλογή ενός πολυδιάστατου πίνακα για αποθήκευση όλων των μεταβλητών, υπαγορεύτηκε από τον εξής περιορισμό: Η γραφική κλάση `VectorField` που υλοποιεί το σύστημα των διανυσμάτων

στην οθόνη μας, και η οποία χρησιμοποιήθηκε στην καρτέλα της Θέασης, δέχεται ένα όρισμα της μορφής `double array[][][]`, οπότε είμασταν αναγκασμένοι να της παράσχουμε τα δεδομένα υπο τη μορφή αυτή.

Στις μεθόδους `eFieldCore()` και `eField()` δηλώνονται ή χρησιμοποιούνται κάποιες μεταβλητές ιδιαίτερης βαρύτητας: Η μεταβλητή `limit`, χρησιμοποιείται σε μια συνθήκη/δικλείδα ασφαλείας (exception handler) η οποία αποτρέπει τον απειρισμό, στην περίπτωση που ένα σωματίδιο περάσει πολύ κοντά από μια θέση υπολογισμού του ηλεκτρικού πεδίου. Καθώς στον υπολογισμό του ηλεκτρικού πεδίου που δημιουργεί ένα φορτίο σε μια ορισμένη θέση (στις θέσεις των διανυσμάτων για την ακρίβεια), υπεισέρχεται η απόσταση του φορτίου από τη θέση αυτή, αν το φορτίο περάσει οριακά από τη θέση υπολογισμού, η απόσταση μηδενίζεται στον παρονομαστή και η τιμή του ηλεκτρικού πεδίου απειρίζεται. Το ίδιο συμβαίνει και με τη δύναμη Coulomb όταν δύο ετερόσημα σωμάτια πλησιάσουν πολύ κοντά το ένα το άλλο. Σημειώνουμε ότι τα σωματίδια **δεν** είναι δηλωμένα σαν σκληρά σφαιρίδια, παρόλο στην οθόνη φαίνονται έτσι για πρακτικούς λόγους. Στην πραγματικότητα για τον αλγόριθμο είναι σημειακά, γι'αυτό και στην οθόνη τα σφαιρίδια φαίνονται να περνούν οριακά το ένα μέσα από το άλλο, αποκτώντας αμέσως ισχυρή επιτάχυνση από τη εκθετικά αυξανόμενη δύναμη Coulomb. Η προσέγγιση αυτή επιλεχθηκε ως πιο ρεαλιστική, αναφερόμενοι σε συμπεριφορά στοιχειωδών σωματίων. Η συνθήκη ασφαλείας που χρησιμοποιεί τη μεταβλητή `limit` εξετάζει πότε η απόσταση μειώνεται κάτω από ένα συγκεκριμένο όριο και αναγκάζει τη μέθοδο να συνεχίσει, χωρίς να υπολογίσει τη δύναμη, ή το ηλεκτρικό πεδίο, ή το δυναμικό όταν αυτό συμβεί.

Από την άλλη πλευρά, η boolean μεταβλητή `visvector`, αποτελεί μεταβλητή ελέγχου και είναι συσχετισμένη με το checkbox "Ηλεκτρικό Πεδίο" στο πάνελ ελέγχου της εξομοίωσης. Όταν τσεκάρουμε το κουτάκι, η μεταβλητή αποκτά τιμή true και η μέθοδος `eField()` εκκινείται υπολογίζοντας το ηλεκτρικό πεδίο. Η *ίδια* μεταβλητή ενεργοποιεί και την γραφική κλάση `VectorField` που απεικονίζει το πεδίο ως σύνολο διανυσμάτων.

Παρόμοια λογική με τη μέθοδο `eField()/eFieldCore()` ακολουθείται και για τον υπολογισμό του δυναμικού, αυτή τη φορά από τις μεθόδους `potCore()` και `potDisp()`. Η `potCore()` καλείται από την `potDisp()` για να εκτελέσει τους υπολογισμούς του δυναμικού σε κάθε σημείο ενός πλέγματος το οποίο ορίζεται από τον πίνακα `potgrid[21][21][3]`. Οι τελικές τιμές αποθηκεύονται στον πίνακα `pot[21][21]` προκειμένου να απεικονιστεί το δυναμικό είτε υπο μορφή χρωματικών ισοδυναμικών γραμμών (γραφική κλάση `ScalarField`), είτε από μια τρισδιάστατη επιφάνεια (γραφική κλάση `SurfacePlot`). Ο λόγος που χρησιμοποιείται ο ενδιάμεσος πίνακας `pot[21][21]` είναι επειδή οι δύο προαναφερόμενες γραφικές κλάσεις προκειμένου να απεικονίσουν μια βαθμωτή συνάρτηση, δέχονται ένα όρισμα της μορφής `double array[][]`.



Σχημα 4.3: Οπτικοποίηση του ηλεκτρικού πεδίου από ένα πλέγμα προσανατολισμένων διανυσμάτων: δύο ετερόσημα φορτισμένα σωματίδια

```
private double potCore (double xi,double yi) {
  double potVal = 0.0, r2;
  for (int k=0; k<partNum; k++) { //particle scan loop
    r2 = (x[k]-xi)*(x[k]-xi)+(y[k]-yi)*(y[k]-yi);
    if (r2<limit) continue;
    potVal -= 10.0*chargeliszt[k]*Math.log(r2)*0.5; //Oloklhrwma !!!!
    }
  return potVal;
}
```

Οι μέθοδοι `potCore()` (πάνω) και `potDisp()` (κάτω).

```
public void potDisp () {

  if (vis2D || vis3D)
  for (int i=0; i<21; i++) { //x coord loop
    for (int j=0; j<21; j++) { //y coord loop
    potgrid[i][j][2] = potCore(potgrid[i][j][0], potgrid[i][j][1]);
    pot[i][j] = potgrid[i][j][2];
    }
```

```
  }
}
```

Όπως στις μεθόδους του ηλεκτρικού πεδίου και της δύναμης, έτσι και εδώ αποφεύγουμε τους απειρισμούς στα σημεία μέτρησης μέσω μιας συνθήκης ασφαλείας που χρησιμοποιεί την παράμετρο `limit`. Επίσης σημειώνουμε τη χρήση των boolean παραμέτρων `vis2D, vis3D` οι οποίες γίνονται αληθείς από τα αντίστοιχα checkboxes του πάνελ ελέγχου της εξομοίωσης. Στην περίπτωση του δυναμικού εισάγεται μια επιπλέον μέθοδος, η `Xtremes()` η οποία χρησιμοποιείται από τις γραφικές κλάσεις για να καθορίσει τα όρια απεικόνισης. Η μέθοδος αυτή περιέχει τις μαθηματικές μεθόδους `Math.min()` και `Math.max()` για να υπολογίζει τις μέγιστες και ελάχιστες τιμές του δυναμικού κάθε στιγμή, ρυθμίζοντας ανάλογα τα όρια απεικόνισης των γραφικών κλάσεων `SurfacePlot` και `ScalarField`, μέσω των παραμέτρων `potMin, potMax`. Έτσι το εύρος απεικόνισης ρυθμίζεται δυναμικά ώστε να εμφανίζει όσο το δυνατόν μεγαλύτερο τμήμα της βαθμωτής συνάρτησης. Ακολουθεί η μέθοδος `Xtremes()`:

```
public void Xtremes () {
 if (!(vis2D)||(vis3D)) {
   potMin = 0.0;
   potMax = 1.0;
   return;
 }
 potDisp ();
 double max = Double.MINVALUE;
 double min = Double.MAXVALUE;
 for (int i=0; i<21; i++) { //x coord loop
   for (int j=0; j<21; j++) { //y coord loop
     min = Math.min(min,potgrid[i][j][2]);
     max = Math.max(max,potgrid[i][j][2]);
     }
   }
 potMin = min;
 potMax = max;
}
```

Κλείνοντας αξίζει να αναφερθούμε στις μεθόδους `add()` και `addparts()` της καρτέλας Προσαρμογής. Πρόκειται για μεθόδους που "δημιουργούν και τοποθετούν" τα σωμάτιά μας, αφού εμείς εισάγουμε τον επιθυμητό αριθμό και το φορτίο τους. Και εδώ επιλέχθηκε η φωλιασμένη σχεδίαση, με την `add()` να καλεί την `addparts()` υπό τις παραμέτρους που καθόρισε ο χρήστης.

```
public void add() {
  for (int i=0; i<numberadded; i++) addParts (chargeselector);
}
```

Οι μέθοδοι `add()` (πάνω) και `addparts()` (κάτω).

```
private void addParts (boolean chsel) {

 //prosthetei n swmatia sthn trexousa thesh
 //tou cursora
 // !!exception handler:max number of part!!


  if (partNum>=maxparts) return;
  int n = partNum;
  x[n] = xc + Math.random()- Math.random();
  y[n] = yc + Math.random()- Math.random();
  vx[n] = 10.0*(Math.random()-0.5);
  vy[n] = 10.0*(Math.random()-0.5);
  if (chsel==true) {
    size[n] = 15.0;
    chargeliszt[n] = 1.0;
    //m[n] = massode;
    color[n] = java.awt.Color.red;
  }
  else {
    size[n] = 10.0;
    chargeliszt[n] = -1.0;
    //m[n] = massode;
    color[n] = java.awt.Color.green;
  }
  togglevis[n] = true;
  partNum++ ;
  //afou prosthesei to particle (+/-) aujanei ton arithmo toys
 }
```

Αφού ο χρήστης ορίσει των αριθμό και το είδος των σωματιδίων που επιθυμεί να προσθέσει στο σύστημα, πρέπει κάνει κλίκ με το ποντίκι μέσα στο γραφικό πεδίο απεικόνισης (υλοποιημένο από τη γραφική κλάση `DrawingPanel`) προκειμένου να τα προσθέσει στο σημείο που επιθυμεί. Η κλάση `DrawingPanel` έχει παραμετροποιηθεί στην καρτέλα της Θέασης να εκκινεί υπό την ενέργεια του χρήστη τη μέθοδο `addparts()` (στις ιδιότητες της `DrawingPanel`,

στο πεδίο "Interaction - On Release" προσθέσαμε το όνομα της μεθόδου που θέλουμε να εκκινηθεί). Η μέθοδος `add()` καλεί την `addparts()` με όρισμα την boolean παράμετρο `chargeselector` η οποία καθορίζει το φορτίο που θα έχουν τα νεόδμητα σωμάτια. Η τιμή της `chargeselector` αλλάζει μέσω του πλήκτρου "αρνητικό - θετικό" στο πάνελ ελέγχου της εξομοίωσης. Η `addparts()` καλείται για κάθε καινούριο σωμάτιο μέσω ένος επαναληπτικού βρόχου και υλοποιεί τον αλγόριθμο εισαγωγής των σωματίων εξετάζοντας πρώτα αν ο συνολικός αριθμός τους δεν ξεπερνά το όριο που έχει τεθεί (`maxparts`). Στη συνέχεια προσδίδει τυχαία θέση και ταχύτητα και καθορίζει το φορτίο, τη μάζα, το μέγεθος και το χρώμα του νέου σωματίου και ενεργοποιεί την αντίστοιχη γραφική κλάση (ShapeSet) ώστε αυτό να εμφανιστεί στη οθόνη `togglevis[]`. Φυσικά στο τέλος αυξάνει και τη μεταβλητή-δείκτη του τρέχοντα συνολικού αριθμού των σωματιδίων.

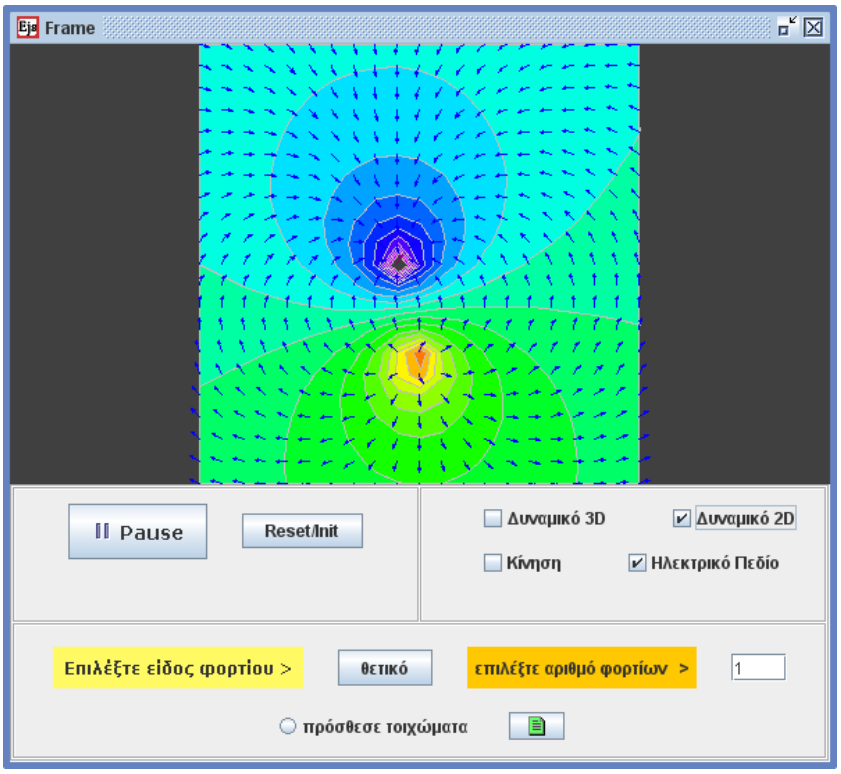
4.2.3 Το γραφικό περιβάλλον.

Το περιβάλλον χρήσης της εξομοίωσης είναι αρκετά λιτό, και δύσκολα φαντάζεται κανείς την πολυπλοκότητα του μοντέλου που του δίνει ζωή.

Αφού εκκινήσει την εφαρμογή ο χρήστης πρέπει να ορίσει τον αριθμό των σωματιδίων που θέλει να προσθέσει στο χώρο του συστήματος καθώς και το είδος τους (αρνητικά ή θετικά). Αφού καθορίσει τον αριθμό και το είδος τους πρέπει να τα "ρίξει" **μέσα** στο χώρο, κάνοντας εκεί κλίκ με το ποντίκι. Τα σωματίδια αρχίζουν να αλληλεπιδρούν και σιγά σιγά επιβραδύνονται καθώς έχει ενσωματωθεί μια παράμετρος επιβράδυνσης, προκειμένου το σύστημα να φτάνει σε μια σχεδόν στάσιμη κατάσταση. Οποιαδήποτε στιγμή θελήσει, ο χρήστης μπορεί να προσθέσει επιπλέον σωματίδια στο χώρο και να παρατηρήσει πώς μεταβάλλεται το υπάρχον σύστημα. Καθώς τα σωματίδια αλληλεπιδρούν, ένα διανυσματικό πλέγμα περιγράφει το ηλεκτρικό πεδίο που δημιουργούν σε πραγματικό χρόνο.

Στο δεξί τμήμα του πλαισίου ελέγχου υπάρχουν checkboxes που παρέχουν επιπλέον πληροφορίες για το σύστημα. Τσεκάροντας το checkbox "δυναμικό 2Δ" εμφανίζεται ένα χρωματικό γράφημα ισοδυναμικών γραμμών που περιγράφει το δυναμικό σε δύο διαστάσεις (οι ψυχρές περιοχές αντιστοιχούν σε χαμηλές τιμές δυναμικού,οι θερμές σε υψηλές). Τσεκάροντας την επιλογή "δυναμικό 3Δ" παρουσιάζεται ένα νέο παράθυρο που αναπαριστά το δυναμικό, αυτή τη φορά στις τρείς διαστάσεις. Το τρισδιάστατο γράφημα γίνεται μάλιστα να περιστραφεί με το ποντίκι για να επιλεγεί η βολικότερη οπτική γωνία.

Τέλος, μπορούμε να ενεργοποιήσουμε δύο εμποδια μέσα στο χώρο μας, από το πλήκτρο "πρόσθεσε τοιχώματα", με τα οποία μπορούμε να εξομοιώσουμε έναν εικονικό πυκνωτή: αρκεί να προσθέσουμε αρνητικά φορτία στο αριστερό τμήμα και θετικά στο δεξί για να μελετήσουμε τη μορφή που αποκτά το πεδίο και το δυναμικό ανάμεσα στις δυο πλάκες. Η εξομοίωση δύο τοίχων μέσα στο πεδίο κίνησης των σωματιδίων είναι στην πραγματικότητα



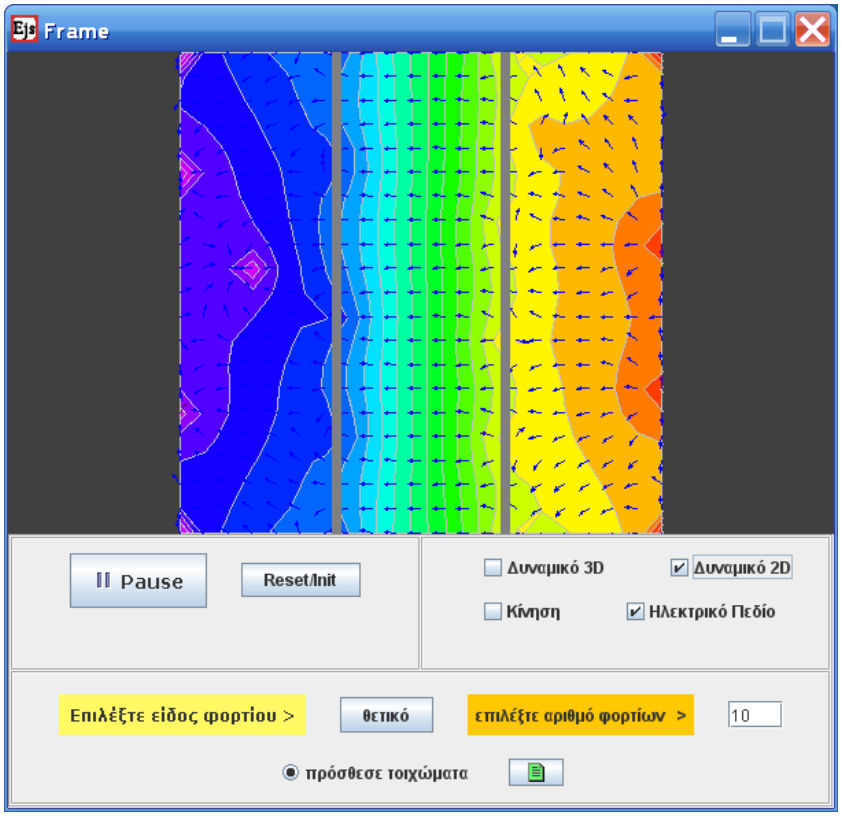
Σχημα 4.4: Απεικόνιση του δυναμικού από χρωματικό γράφημα και ισοδυναμικές γραμμές

ένα τέχνασμα: Στην καρτέλα της Εξέλιξης, προστέθηκε κώδικας ο οποίος αφού ενεργοποιηθεί μέσω μιας `boolean` παραμέτρου (`walls`) συσχετισμένης με το γραφικό πλήκτρο, προσθέτει δύο συνθήκες που εξετάζουν τη φορά κίνησης και το φορτίο του κάθε σωματίου.

```
//kroush me toix-pyknwths
if (walls==true) {
for (int i=0; i<partNum; i++) {
   if ( (x[i]-proX[i])>0  (x[i]> -70.0)  (chargeliszt[i]<0) ) {
     vx[i] = -vx[i];
   }
   if ((x[i]-proX[i])<0  (x[i]< 70.0)  (chargeliszt[i]>0)) {
     vx[i] = -vx[i];
   }
 }
 }
```

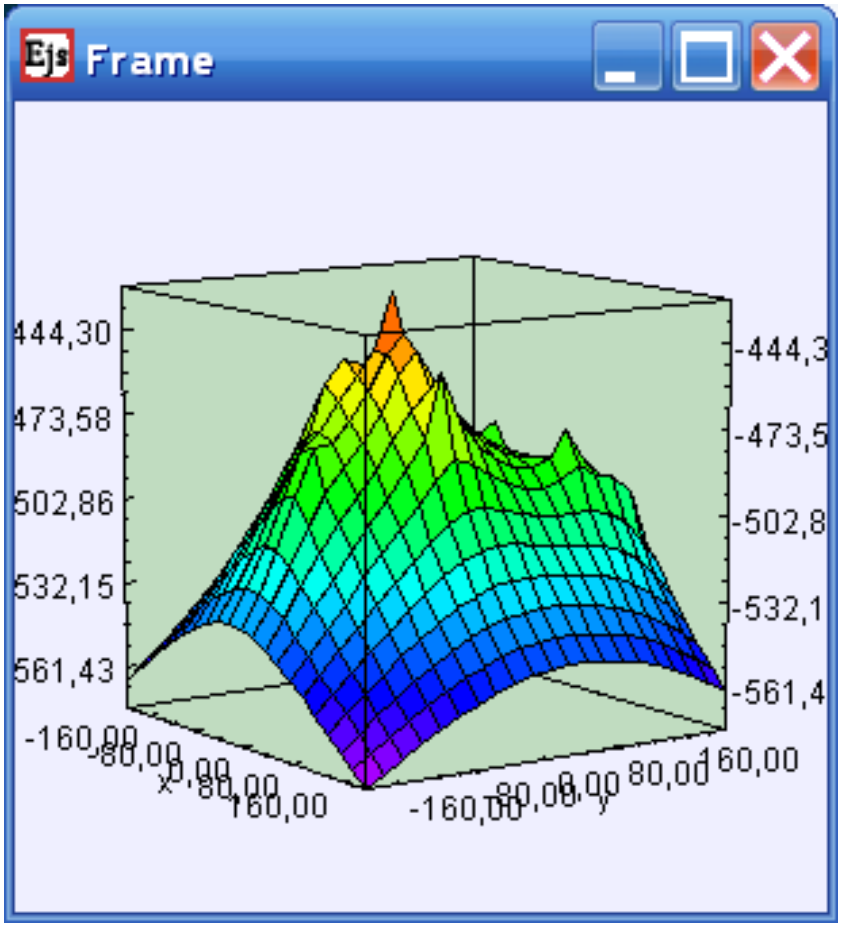
Αρκετά πιο πριν είχαμε σημειώσει ότι στους Περιορισμούς υπάρχει μια δομή που αποθηκεύει τις τρέχουσες θέσεις των σωματιδίων στις μεταβλητές `proX[]` και `proY[]`. Οι δύο αυτές μεταβλητές χρησιμοποιούνται στον πιο πάνω κώδικα για να εξεταστεί η φορά κίνησης (προς τα δεξιά ή προς τα αριστερά) των σωματιδίων. Οι παραπάνω συνθήκες εμποδίζουν τα αρνητικά φορτία που ρίχνονται στο αριστερό τμήμα της οθόνης να κινηθούν δεξιά, και τα θετικά

φορτία που ρίχνονται δεξιά, να κινηθούν αριστερά. Έτσι τα αρνητικά φορτία παγιδεύονται στο αριστερό τμήμα του πεδίου απεικόνισης ενώ τα θετικά στο δεξί (δυστυχώς ο κώδικας στην παρούσα μορφή του δεν λειτουργεί για την αντίθετη περίπτωση, δηλαδή να παγιδεύει τα αρνητικά δεξία και τα θετικά αριστερά). Οι δύο πλάκες του πυκνωτή, δεν είναι τίποτα άλλο παρά δυο γραφικά στοιχεία (`Segments`) τα οποία γίνονται ορατά επίσης μέσω της παραμέτρου `walls` ακριβώς στις **`ευθείες`** που υφίστανται οι παραπάνω οριακές συνθήκες. Παρόλα αυτά δημιουργείται η ψευδαίσθηση στον χρήστη της προσομοίωσης ότι τα σωμάτια προσκρούουν πάνω τους και αναπηδούν ελαστικά εξαιτίας τους.



Σχημα 4.5: Εικονικός πυκνωτής

Παρατηρώντας τη μορφή που αποκτά το ηλεκτρικό πεδίο και δυναμικό ανάμεσα στις πλάκες αντιλαμβανόμαστε ότι μια εξομοίωση εφοδιασμένη με ένα καλά διατυπωμένο φυσικό μοντέλο μπορεί να παράξει αποτελέσματα εφάμιλλα ενός πραγματικού πειράματος, επαληθεύοντας έτσι την αξία των εξομοιώσεων φυσικής σε υπολογιστή.•



Σχημα 4.6: Απεικόνιση του δυναμικού στον τρισδιάστατο χώρο

Παράρτημα Α΄

Αλγόριθμοι αριθμητικής επίλυσης Συνήθων Διαφορικών Εξισώσεων

Οι συνήθεις διαφορικές εξισώσεις (ΣΔΕ ή ODEs) εμπλέκονται στην μοντελοποίηση σχεδών όλων των φυσικών συστημάτων αλλά και σε τομείς όπως η Οικονομία ή η Κοινωνιολογία. Προκειμένου να μαθηματικοποιηθεί ένα σύστημα καταγράφονται τα μεγέθη/μεταβλητές από τα οποία εξαρτάται και παράγονται μαθηματικοί τύποι, που πολύ συχνά περιλαμβάνουν παραγώγους διαφόρων τάξεων. Με τον όρο *Συνήθεις* αναφερόμαστε σε διαφορικές εξισώσεις που περιλαμβάνουν μόνο παραγώγους 1ου βαθμού των μεταβλητών που αφορούν το σύστημα. Ο λόγος για τον οποίο οι ΣΔΕ είναι τόσο διαδεδομένες είναι γιατί, όπως θα δούμε και πιο κάτω, οι διαφορικες εξισώσεις ανωτέρων τάξεων απλοποιούνται με διάφορα τεχνάσματα σε συστήματα από συνήθεις διαφορικές εξισώσεις τα οποία επιλύονται σαφώς ευκολότερα.

Ειδικότερα, όσον αφορά την επίλυση ενός προβλήματος που περιλαμβάνει διαφορικές εξισώσεις, αυτή μπορεί να γίνει είτε αναλυτικά, είτε αριθμητικά: οι κατηγορίες διαφορικών εξισώσεων που επιλύονται με αναλυτικό τρόπο είναι πολύ λίγες, σε σχέση με το συνολικό αριθμό τους, και ακόμα και τα πιο απλά φυσικά συστήματα (π.χ. εξισώσεις κίνησης 3 αλληλεπιδρώντων σωμάτων) δεν έχουν πλήρεις αναλυτικές λύσεις. Για τον παραπάνω λόγο, αλλά και για μεγαλύτερη ευκολία, προτιμώνται οι αριθμητικές μέθοδοι επίλυσης. Σε αυτές, κατασκευάζεται ένας μαθηματικός αλγόριθμος, ο οποίος μετά από ένα πεπερασμένο αριθμό επαναλήψεων, προσεγγίζει τη ζητούμενη λύση με κάποια δεδομένη ακρίβεια. Προκειμένου να καλυφθούν οι ανάγκες μιας εφαρμογής (π.χ. επίλυση των διαφορικών εξισώσεων στατικής μηχανικής, που αφορούν τις τάσεις ενός δικτυώματος που πρόκειται να κατασκευαστεί), επιλέγεται ο κατάλληλος αλγόριθμος για το σύστημα και γίνεται ευρεία χρήση υπολογιστών,

για να επιταχυνθεί η διαδικασία των αριθμητικών πράξεων. Δύναται με αυτόν τον τρόπο να επιτευχθεί ακρίβεια, αρκετές τάξεις μεγέθους κάτω από όσο απαιτεί η εκάστοτε εφαρμογή.

Α′.1 Λίγα λόγια για τις Συνήθεις Διαφορικές Εξισώσεις.

Έστω f(x,y) μια πραγματική συνάρτηση δύο μεταβλητών, ορισμένη στα διαστήματα $a \leq x \leq b$ και $a \leq y \leq b$. Ψποθέτουμε ότι y είναι μια πραγματική συνάρτηση ορισμένη στο διάστημα [a,b]. Η πιο απλή μορφή διαφορικής εξίσωσης είναι:

$$\frac{dy}{dx} = f(x, y) \tag{Α′.1}$$

και ονομάζεται συνήθης διαφορική εξίσωση πρώτης τάξης. Κάθε πραγματική συνάρτηση y(x), η οποία είναι παραγωγίσιμη και ικανοποιεί την διαφορική εξίσωση για όλα τα x του διαστήματος [a,b], λέμε ότι είναι μια λύση της διαφορικής εξίσωσης. Η εξίσωση (Α′.1) ονομάζεται γραμμική αν έχει τη μορφή:

$$\frac{dy}{dx} = Ay + g(x) \tag{Α′.2}$$

όπου A μια σταθερά.

Είναι γνωστό ότι η διαφορική εξίσωση (Α′.1), αν έχει λύση, η λύση αυτή δεν είναι μοναδική, επειδή μια αυθαίρετη σταθερά εισάγεται ολοκληρώνοντάς την. Για παράδειγμα, η συνάρτηση:

$$y(x) = c \cdot exp(\lambda x) \tag{Α′.3}$$

είναι για κάθε τιμή της σταθεράς c, μια λύση της διαφορικής εξίσωσης:

$$\frac{dy}{dx} = \lambda \cdot y \tag{Α′.4}$$

όπου λ μια σταθερά. Για να υπολογίσουμε μια ιδιαίτερη λύση, πρέπει να επιβάλλουμε μια επιπλέον συνθήκη στη λύση, της μορφής:

$$y(x_o) = y_o \tag{Α′.5}$$

όπου $x \in [a, b]$. Η συνθήκη (Α′.5) ονομάζεται αρχική συνθήκη και το y_o αρχική τιμή της y(x). Για το παραπάνω παράδειγμα, η ιδιαίτερη λύση που ικανοποιεί την (Α′.5) είναι:

$$y\left(x\right) = y_o \cdot e^{\lambda \cdot (x - x_o)} \tag{Α′.6}$$

Αν θεωρήσουμε μια διαφορική εξίσωση μαζί με μια αρχική συνθήκη, τότε ορίζουμε το λεγόμενο **πρόβλημα αρχικών τιμών (ΠΑΤ)**. Η γενική μορφή του προβλήματος αρχικών τιμών για συνήθεις διαφορικές εξισώσεις πρώτης τάξης είναι:

$$\frac{dy}{dx} = f\left(x, y\left(x\right)\right), \quad y\left(a\right) = y_o, \quad x \in [a, b], \tag{Α′.7}$$

όπου ο τόνος δηλώνει παραγώγιση ως προς την μεταβλητή x, και y είναι γενικά ένα διάνυσμα διάνυσμα διάστασης n.

Θεώρημα 8.1 (Μοναδικότητας): Θεωρούμε το σύνολο

$$D = \{(x, y) \;\; a \leq x \leq b, -\infty \leq y \leq +\infty\}$$

και υποθέτουμε ότι η f(x,y) είναι συνεχής στο D. Αν η f ικανοποιεί μια συνθήκη *Lipschitz* στο Δ ως προς την μεταβλητή y, τότε το πρόβλημα αρχικών τιμών (Α′.23) έχει μια μοναδική λύση y(x) για $a \leq x \leq b$

Α′.2 Πρόβλημα αρχικών τιμών διαφορικού συστήματος 1ης τάξης

Σε πολλές εφαρμογές συναντάμε όχι μόνο απλές διαφορικές εξισώσεις 1ης τάξης, αλλά και συστήματα διαφορικών εξισώσεων 1ης τάξης με n εξισώσεις και n εξαρτημένες μεταβλητές. Το πρόβλημα αρχικών τιμών στην περίπτωση αυτή γράφεται:

$$\frac{d}{dx}\left(\vec{y}\right) = \vec{f}\left(x, \vec{y}\left(x\right)\right), \quad \vec{y}\left(a\right) = \vec{q}, \quad x \in [a, b], \tag{Α′.8}$$

όπου

$$\vec{y} = [y_1, y_2, \ldots, y_n]^T, \quad \vec{f} = [f_1, f_2, \ldots, f_n]^T, \quad \vec{q} = [q_1, q_2, \ldots, q_n]^T \tag{Α′.9}$$

.

Για την παραπάνω περίπτωση το θεώρημα μοναδικότητας γενικεύεται εύκολα και δίνει τις αναγκαίες συνθήκες για ύπαρξη λύσης.

Όπως προλέχθηκε πολλά φυσικά προβλήματα εκφράζονται από διαφορικές εξισώσεις ανώτερης τάξης. Το ΠΑΤ για μια διαφορική εξίσωσης τάξης n μπορεί να γραφεί υπό τη γενική μορφή:

$$y^{(n)} = f\left(x, y^{(0)}, y^{(1)}, \ldots, y^{(n-1)}\right), \quad y^{(i)}(a) = q_i, \quad i = 0, 1, 2, \ldots \tag{A′.10}$$

και τότε οι συναρτήσεις $y_1(x), y_2(x), \ldots, y_n(x)$ ικανοποιούν το σύστημα

$$\begin{aligned} \frac{dy_1}{dx} &= y_2 \\ \frac{dy_2}{dx} &= y_3 \\ &\vdots \\ \frac{dy_{n-1}}{dx} &= y_n \\ \frac{dy_n}{dx} &= f(y_1, y_2, \ldots, y_n) \end{aligned} \tag{A′.11}$$

με αρχικές συνθήκες:

$$\begin{aligned} y_1(a) &= q_1 \\ y_2(a) &= q_2 \\ &\vdots \\ y_n(a) &= q_n \end{aligned} \tag{A′.12}$$

Έτσι το πρόβλημα (A′.10) τελικά ανάγεται στο ισοδύναμο πρόβλημα:

$$\frac{d}{dx}(\vec{y}) = \vec{f}(x, \vec{y}(x)), \quad \vec{y}(a) = \vec{q}, \quad x \in [a, b], \tag{A′.13}$$

όπου

$$\vec{f} = [y_2, y_3, \ldots, y_n, f(x, y_1, y_2, \ldots, y_n)] \tag{A′.14}$$

Α′.3 Αριθμητικές μέθοδοι επίλυσης ΣΔΕ.

Ας θεωρήσουμε την ακολουθία διακεκριμένων σημείων $\{x_n\}$, τα οποία ορίζονται από την σχέση $x_n = a + n \cdot h$, ν=0,1,2... Η παράμετρος h η οποία θεωρείται σταθερή, εκτός αν ορίζεται διαφορετικά, ονομάζεται βήμα. Μια θεμελιώδης ιδιότητα των περισσοτέρων υπολογιστικών μεθόδων για την αριθμητική επίλυση του προβλήματος (Α′.23) είναι αυτή της **διακριτότητας**. Δηλαδή ζητάμε μια προσεγγιστική λύση, όχι στο συνεχές διάστημα $a \leq x \leq b$, αλλά στο σύνολο των διακεκριμένων σημείων:

$$\left\{ x_n | n = 0, 1, 2 \ldots, \frac{(b-a)}{h} \right\} \tag{Α′.15}$$

Οι βασικότερες κατηγορίες μεθόδων για την αριθμητική επίλυση του προβλήματος (Α′.23) είναι:

- Οι μέθοδοι **απλού βήματος** (one step methods), των οποίων ο κύριος εκφραστής είναι οι μέθοδοι τύπου **Runge - Kutta**.
- Οι μέθοδοι **πολλαπλού βήματος** (multistep methods) διαφόρων τύπων με κύριο εκφραστή τις μεθόδους τύπου **Adams**.

Έστω y_n, μια προσέγγιση της θεωρητικής τιμής $y(x_n)$, της λύσης $y(x)$ του προβλήματος (Α′.23) στο σημείο x_n και $f_n = f\left(x_n, y_{(}n)\right)$. Στην περίπτωση των μεθόδων απλού βήματος, η λύση στο σημείο x_{n+1}, δηλαδή το $y(x_{n+1})$, προσεγγίζεται από το $y(x_{n+1})$ χρησιμοποιώντας πληροφορίες μόνο από την προσέγγιση της λύσης στο προηγούμενο σημείο x_n. Η γενική μορφή μιας μεθόδου αυτής της κατηγορίας μπορεί να εκφραστεί με τη σχέση:

$$y_{n+1} = y_n + h \cdot \Phi(x_n, y_n, h) \tag{Α′.16}$$

Στην περίπτωση των μεθόδων πολλών βημάτων, έστω k, η προσέγγιση της λύσης στο σημείο x_{n+k}, χρησιμοποιεί πληροφορίες από την προσέγγιση της λύσης στα k προηγούμενα σημεία $x_n, x_{n+1}, \ldots, x_{n+k-1}$. Η γενική μορφή μιας μεθόδου από την κατηγορία αυτή μπορεί να γραφεί:

$$\sum_{i=0}^{k} a_i y_{n+i} = h \cdot \Phi(x_n, y_{n+k}, y_{n+k-1}, \ldots, y_n, h) \tag{Α′.17}$$

για $0 \leq n-k$, $b-a = N \cdot h$, όπου $\{a_i\}, i = 0, 1, 2 \ldots, k$ είναι σταθερές και οι αρχικές τιμές $y_o, y_1, \ldots, y_{k-1}$ θεωρούνται γνωστές. Αν στην (Α′.17) η συνάρτηση Φ είναι ανεξάρτητη της y_{n+k} η μέθοδος ονομάζεται άμεση **(explicit)**, διαφορετικά ονομάζεται έμμεση **(implicit)**.

Α′.3.1 Σφάλματα

Έστω x_n ένα σημείο του διαστήματος ορισμού της f. Με $y(x_n)$ συμβολίζουμε την ακριβή λύση του προβλήματος στο σημείο αυτό, με y_n την αντίστοιχη θεωρητική προσεγγιστική λύση η οποία προκύπτει από τη μέθοδο (Α′.16), και με $\bar{y_n}$ την αριθμητική τιμή της προσεγγιστικής λύσης η οποία προκύπτει από τον υπολογιστή. Μπορούμε τώρα να διατυπώσουμε τους επόμενους ορισμούς.

- Το **τοπικό σφάλμα αποκοπής ή διακριτοποίησης** local truncation or discretization error T_{n+1} στο σημείο x_{n+1}, που αντιστοιχεί στην μέθοδο (Α′.16) ορίζεται από τη σχέση:

$$T_{n+1} = y(x_{n+1}) - y(x_n) - h \cdot \Phi(x_n, y(x_n), h) \tag{Α′.18}$$

  και υπολογίζει το κατά πόσο η ακριβής λύση του προβλήματος (Α′.23) αποτυγχάνει να ικανοποιήσει τη μέθοδο (Α′.16). Η ποσότητα αυτή μπορεί να θεωρηθεί σαν ένα πρώτο μέτρο για την ακρίβεια της μεθόδου. Στην περίπτωση αυτή υποθέτουμε ότι δεν υπάρχουν σφάλματα στα προηγούμενα σημεία και με την έννοια αυτή θεωρείται τοπικό.

- Το **τοπικό σφάλμα στρογγύλευσης** (local roundin error) R_{n+1} στο σημείο x_{n+1}, της μεθόδου (Α′.16) ορίζεται από τη σχέση:

$$R_{n+1} = \bar{y_{n+1}} - \bar{y_n} - h \cdot \Phi(x_n, \bar{y_n}, h) \tag{Α′.19}$$

  και υπολογίζει το κατά πόσο η αριθμητική τιμή της λύσης του προβλήματος (Α′.23) που προκύπτει από τον υπολογιστή αποτυγχάνει να ικανοποιήσει τη μέθοδο.

- Το **ολικό σφάλμα διακριτοποίησης** (global truncation error) στο σημείο x_{n+1} δίνεται από τον τύπο

$$\epsilon_{n+1} = y(x_{n+1}) - y_{n+1} \tag{Α′.20}$$

  και εξαρτάται από τα σφάλματα αποκοπής στα *προηγούμενα* σημεία της διαμέρισης.

- Το **ολικό σφάλμα στρογγύλευσης** (global rounding error) στο σημείο x_{n+1} ορίζεται ως

$$r_{n+1} = y_{n+1} - \bar{y}_{n+1} \tag{Α′.21}$$

- Η διαφορά

$$\bar{\epsilon}_{n+1} = y\left(x_{n+1}\right) - \bar{y}_{n+1} \tag{Α′.22}$$

ορίζει το **ολικό σφάλμα διακριτοποίησης** για την υπολογισθείσα αριθμητική τιμή της λύσης στο σημείο x_{n+1}.

Α′.3.2 Δύο γνωστοί αλγόριθμοι αριθμητικής επίλυσης.

Η επονομαζόμενη μέθοδος απλού βήματος υπολογίζει την προσέγγιση της λύσης ενός προβλήματος ΠΑΤ:

$$\frac{dy}{dx} = f\left(x, y\left(x\right)\right), \quad y\left(a\right) = y_o, \quad x \in [a, b], \tag{Α′.23}$$

στο σημείο x_{n+1}, που συσχετίζεται με το y_{n+1}, όταν δίνεται η προσέγγιση y_n ένα κατάλληλα επιλεγμένο βήμα h, και η διαφορική εξίσωση με το θεώρημα **Taylor**:

$$y\left(x_{n+1}\right) = y\left(x_n + h\right) = y\left(x_n\right) + h \cdot \frac{d}{dx} y\left(x_n\right) + \frac{h^2}{2!} \frac{d^2}{dx} y\left(x_n\right) + \ldots \tag{Α′.24}$$

Μιας και ασχολούμαστε με εφαρμογές φυσικής, μπορούμε να δώσουμε σαν γενικό παράδειγμα τις εξισώσεις κίνησης του Νεύτωνα ενός σωματιδίου σε μια διάσταση και χρησιμοποιώντας σαν μεταβλητές το χρόνο $\mathbf{t}$, τη θέση $\mathbf{x}$, την ταχύτητα $\boldsymbol{\upsilon}$ και την επιτάχυνση $\boldsymbol{\alpha}$:

$$\frac{d\upsilon}{dt} = \alpha\left(t\right) \tag{Α′.25}$$

$$\frac{dx}{dt} = \upsilon\left(t\right) \tag{Α′.26}$$

όπου $\alpha\left(t\right) = \alpha\left(x\left(t\right), \upsilon\left(t\right), t\right)$. Στη συγκεκριμένη περίπτωση στόχος της αριθμητική μεθόδου είναι να υπολογίσει τις τιμές των x_{n+1} και υ_{n+1} τη χρονική στιγμή $t_{n+1} = t + \Delta t$. Αν το σύστημα είναι συντηρητικό, το Δt πρέπει να είναι αρκετά μικρό ώστε η ενέργεια να διατηρείται στα πλαίσια του επιτρεπόμενου σφάλματος. Σημειώνουμε εδώ ότι οι αλγόριθμοι που διατηρούν τα μεγέθη του χώρου των φάσεων ονομάζονται *συμπλεκτικοί*. Εφαρμόζοντας το θεώρημα Taylor για τα παραπάνω μεγέθη π.χ. για τη θέση, ο τύπος δίνει:

$$x\left(t + \Delta t\right) = x\left(t\right) + \Delta t \cdot \dot{x}\left(t\right) + \frac{\Delta t^2}{2!} \ddot{x}\left(t\right) + \frac{\Delta t^3}{3!} x^{(3)}\left(t\right) + \ldots \tag{Α′.27}$$

ομοίως και για την ταχύτητα. Αποκόπτουμε τώρα τους μεγιστοβάθμιους όρους και τους αντικαθιστούμε με το αντίστοιχο σφάλμα αποκοπής $O(\Delta t)$.

$$\upsilon_{n+1} = \upsilon_n + a_n \cdot \Delta t + O\left(\Delta t^2\right) \tag{A′.28}$$

$$x_{n+1} = x_n + \upsilon_n \cdot \Delta t + \frac{1}{2}\alpha_n\left(\Delta t^2\right) + O\left(\Delta t^3\right) \tag{A′.29}$$

Κρατώντας τους δύο πρώτους όρους από κάθε σειρά, καταλήγουμε στο διάσημο αλγόριθμο του *Euler*:

$$\upsilon_{n+1} = \upsilon_n + \alpha_n \cdot \Delta t \tag{A′.30}$$

$$x_{n+1} = x_n + \upsilon_n \cdot \Delta t \tag{A′.31}$$

Παρατηρούμε ότι ο αλγόριθμος Euler παρουσιάζει τοπικό σφάλμα αποκοπής της τάξης του $(\Delta t)^2$ σε κάθε βήμα ενώ το ολικό σφάλμα, λόγω συσσώρευσης των διαδοχικών σφαλμάτων ανά βήμα είναι της τάξεως του (Δt). Ο αλγόριθμος χαρακτηρίζεται ως πρώτης τάξης επειδή το ολικό σφάλμα που παρουσιάζεται είναι πρώτης τάξης ως προς το βήμα $(\Delta t)^1$.

Προσπαθώντας να βελτιώσουμε την απόδοση του αλγορίθμου Euler, εφαρμόζουμε το εξής τέχνασμα: χρησιμοποιούμε την αρχική υ_n και τελική υ_{n+1} ταχύτητα και στη συνέχεια υπολογίζουμε τη νέα θέση από τη *μέση τιμή* τους. Η νέα μέθοδος που προκύπτει ονομάζεται *midpoint*:

$$\upsilon_{n+1} = \upsilon_n + \alpha_n \cdot \Delta t \tag{A′.32}$$

$$x_{n+1} = x_n + \frac{1}{2}\left(\upsilon_n + \upsilon_{n+1}\right)\Delta t \tag{A′.33}$$

Αντικαθιστώντας την (A′.32) στην (A′.33) παίρνουμε:

$$x_{n+1} = x_n + \upsilon_n \cdot \Delta t + \frac{1}{2}\alpha_n \Delta t^2 \tag{A′.34}$$

Παρατηρούμε ότι ο αλγόριθμος midpoint παρουσιάζει σφάλμα δευτέρου βαθμού ως προς τη θέση και πρώτου ως προς την ταχύτητα. Γενικά ο συγκεκριμένος αλγόριθμος είναι

αποδοτικός για συστήματα όπου η επιτάχυνση είναι σταθερή, αλλά η γρήγορη συσσώρευση λαθών τον καθιστούν ασταθή για περαιτέρω χρήσεις.

Μια άλλη βελτιωμένη εκδοχή του αλγορίθμου Euler είναι ο επονομαζόμενος αλγόριθμος Euler-Richardson. Γενικά, προσπαθώντας να κατασκευάσουμε πιο αποδοτικούς αλγόριθμους θα πρέπει πάντα να λαμβάνουμε υπόψιν την πολυπλοκότητα των πράξεων που θα έχουν τελικά, ώστε να μην έχουμε σπατάλη υπολογιστικών πόρων. Δεν πρέπει να ξεχνάμε ότι για να επιλυθεί αριθμητικά μια οποιοδήποτε εξίσωση στην επιθυμητή ακρίβεια απαιτούνται δεκάδες, εκατοντάδες ακόμα και χιλιάδες επαναλήψεις της διαδικασίας που ορίζει ο αλγόριθμος.

Πρακτικά, ο Euler-Richardson προκύπτει ακολουθώντας τη λογική του ημίσεως βήματος (half-step algorithm). Έστω $x\left(t\right)$ η θέση τη χρονική στιγμή t και $x_1{=}x\left(t+\Delta t\right)$ μετά από το χρονικό βήμα Δt. Γράφουμε το x_1 ως εξής:

$$x_1 = x\left(t+\Delta t\right) = x\left(t\right) + \upsilon\left(t\right)\cdot\Delta t + \frac{1}{2}\cdot\alpha\left(t\right)\left(\Delta t\right)^2 \tag{A′.35}$$

Χωρίζουμε τώρα το αρχικό χρονικό βήμα Δt στη μέση και ξαναυπολογίζουμε τη θέση για το πρώτο μισό:

$$x\left(t+\frac{\Delta t}{2}\right) \approx x\left(t+\Delta t\right) + \upsilon\left(t\right)\cdot\frac{\Delta t}{2} + \frac{1}{2}\alpha\left(t\right)\left(\frac{\Delta t}{2}\right)^2 \tag{A′.36}$$

Για το δεύτερο μισό του χρονικού βήματος έχουμε:

$$x_2 \approx x\left(t+\frac{\Delta t}{2}\right) + \upsilon\left(t+\frac{\Delta t}{2}\right)\cdot\frac{\Delta t}{2} + \frac{1}{2}\alpha\left(t\right)\left(\frac{\Delta t}{2}\right)^2 \tag{A′.37}$$

συνδυάζοντας τις (A′.37) και (A′.40) προκύπτει:

$$x_2\left(t+\Delta t\right) \approx x\left(t\right) + \frac{1}{2}\left[\upsilon\left(t\right)+\upsilon\left(t+\frac{\Delta t}{2}\right)\right]\cdot\Delta t + \frac{1}{2}\left[\alpha\left(t\right)+\alpha\left(t+\frac{\Delta t}{2}\right)\right]\cdot\left(\frac{\Delta t}{2}\right)^2 \tag{A′.38}$$

θεωρώντας την προσέγγιση $\alpha\left(t+\frac{\Delta t}{2}\right) \approx \alpha\left(t\right) + \frac{1}{2}\acute{\alpha}\left(t\right)\Delta t$ η x_2 γίνεται:

$$x_2\left(t+\Delta t\right) = x\left(t\right) + \frac{1}{2}\left[\upsilon\left(t\right)+\upsilon\left(t+\frac{\Delta t}{2}\right)\right]\cdot\Delta t + \frac{1}{2}\left[2\cdot\alpha\left(t\right)\right]\cdot\left(\frac{\Delta t}{2}\right)^2 \tag{A′.39}$$

Συνδυάζοντας τώρα τις (Α′.36) και (Α′.39), $(2x_2 - x_1)$ καταλήγουμε στον αλγόριθμο Euler-Richardson.

$$x_{ER}(t + \Delta t) = x(t) + \upsilon\left(t + \frac{\Delta t}{2}\right) \cdot \Delta t + O(\Delta t)^3 \tag{Α′.40}$$

$$\upsilon_{ER} = \upsilon(t) + \alpha\left(t + \frac{\Delta t}{2}\right) \cdot \Delta t + O\left(\Delta t^3\right) \tag{Α′.41}$$

Παρατηρούμε ότι παρόλο που στηρίζεται σε αλγορίθμους μικρής τάξης, ο E-R καταφέρνει να αγγίξει ακρίβεια τρίτης τάξης, ισοδύναμης με αυτή του αλγορίθμου *Runge-Kutta* 2ης τάξης. Επιπλέον προκύπτει ότι από τις διαφορές $|x_2 - x_1|$ και $|\upsilon_2 - \upsilon_1|$ μπορουμε να υπολογίζουμε τα αντίστοιχα σφάλματα. Για το λόγο αυτό, ο E-R χρησιμοποιείται σε προχωρημένες αριθμητικές μεθόδους μεταβλητού βήματος, όπου αφού εξεταστεί το μέγεθος του σφάλματος, το βήμα τροποποιείται δυναμικά κατά τη διάρκεια της επίλυσης για να επιτευχθεί περισσότερη ακρίβεια.

Ή πιο διαδεδομένη ίσως μέθοδος αριθμητικής επίλυσης ΣΔΕ, είναι η *Runge-Kutta*. Αναπτύχθηκε στις αρχές του αιώνα, ταυτόχρονα από το γερμανό Karl David Tolme Runge και τον πολωνό Martin Wilhelm Kutta. Η φιλοσοφία της στηρίζεται στο ότι το διαφορικό:

$$\frac{dx}{dy} = f(x, t) \tag{Α′.42}$$

που αποτελεί λύση της δ.ε. εκτιμάται πολλές φορές μέσα στο διάστημα $(t + \Delta t)$. Για παράδειγμα ο αλγόριθμος R-K τετάρτης τάξης, υπολογίζει το διαφορικό τέσσερις φορές μέσα σε κάθε βήμα: μια στην αρχή, μια στο τέλος και δύο ενδιάμεσα, παράγοντας τέσσερις διαφορετικές τιμές k_1, k_2, k_3, k_4 κάθε μια με ξεχωριστό παράμετρο βάρους c_1, c_2, c_3, c_4, από τις οποίες προκύπτει η σταθμισμένη τιμή της f(x,t) για το συγκεκριμένο χρονικό βήμα. Εφαρμόζοντας τον αλγόριθμο στον χώρο (u,x) με $c_1 = \frac{1}{6}, c_2 = \frac{2}{6}, c_3 = \frac{2}{6}, c_4 = \frac{1}{6}$ έχουμε:

$$\upsilon_{n+1} = \upsilon_n + \frac{1}{6}\left(k_{1\upsilon} + 2k_{2\upsilon} + 2k_{3\upsilon} + k_{4\upsilon}\right) \tag{Α′.43}$$

$$x_{n+1} = x_n + \frac{1}{6}\left(k_{1x} + 2k_{2x} + 2k_{3x} + k_{4x}\right) \tag{Α′.44}$$

όπου:

$$
\begin{aligned}
k_{1\upsilon} &= \alpha\left(x_n, \upsilon_n, t_n\right) \cdot \Delta t \\
k_{1x} &= \upsilon_n \cdot \Delta t \\
k_{2\upsilon} &= \alpha \cdot \left(x_n + \frac{k_{1x}}{2}, \upsilon_n + \frac{k_{1\upsilon}}{2}, t_n + \frac{\Delta t}{2}\right) \cdot \Delta t \\
k_{2x} &= \left(\upsilon_n + \frac{k_{1\upsilon}}{2}\right) \cdot \Delta t \\
k_{3\upsilon} &= \alpha\left(x_n + \frac{k_{2x}}{2}, \upsilon_n + \frac{k_{2\upsilon}}{2}, t_n + \frac{\Delta t}{2}\right) \cdot \Delta t \\
k_{3x} &= \left(\upsilon_n + \frac{k_{2\upsilon}}{2}\right) \cdot \Delta t \\
k_{4\upsilon} &= \alpha\left(x_n + \frac{k_{3x}}{2}, \upsilon_n + \frac{k_{3\upsilon}}{2}, t_n + \Delta t\right) \\
k_{4x} &= \left(\upsilon_n + k_{3\upsilon}\right) \cdot \Delta t
\end{aligned}
\tag{Α′.45}
$$

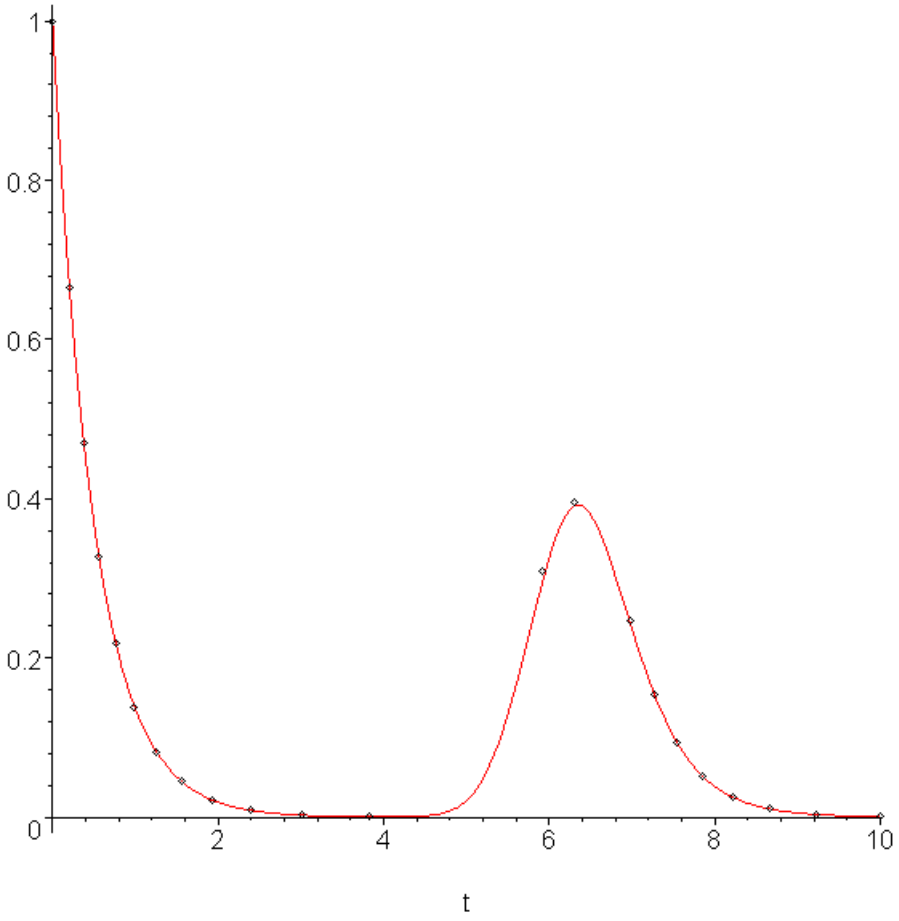
(Α′.46)

Υπάρχουν πολλές εκδοχές των αλγορίθμων R-K ανάλογα με τις τιμές των παραμέτρων c_i και τη συμπεριφορά του αλγορίθμου στο πρόβλημα που επιλύεται. Μια από αυτές, πνευματικό κατασκεύασμα του Fehldberg χρησιμοποιεί εν παραλλήλω δύο αλγορίθμους Runge-Kutta, έναν τετάρτης και έναν πέμπτης τάξης με τη γενική μορφή:

$$y_{n+1} = y_n + c_1 k_1 + c_2 k_2 + c_3 k_3 + c_4 k_4 + c_5 k_5 \tag{Α′.47}$$

$$y^*_{n+1} = y_n + c^*_1 k_1 + c^*_2 k_2 + c^*_3 k_3 + c^*_4 k_4 \tag{Α′.48}$$

Ο σκοπός είναι από τη διαφορά τους (το αποτέλεσμα από τον αλγόριθμο πέμπτης τάξης θεωρείται ακριβέστερο και συγκρίνεται με το αποτέλεσμα του τεταρτοτάξιου) να εκτιμηθεί το σφάλμα και να ρυθμιστεί ανάλογα το εύρος του επόμενου βήματος. Έχοντας ορίσει μια παράμετρο ανοχής σφάλματος, όταν το εκτιμώμενο σφάλμα ανα βήμα την ξεπερνά, το βήμα μειώνεται για να αυξηθεί η ακρίβεια και το αντίθετο. Έτσι ο αλγόριθμος Runge-Kutta-Fehldberg αποκτά δυναμική συμπεριφορά, επιταχύνοντας στα σημεία που η συνάρτηση-λύση προσεγγίζεται εύκολότερα, και επιβραδύντας στα δυσεπίλυτα σημεία (π.χ. ακρότατα).



Σχημα Α΄.1: Προσέγγιση της συνάρτησης $\frac{dy}{dx} = -2y\left(t\right) + e^{-2\cdot\left(t-6\right)^2}$ από τον αλγόριθμο R-K-F

Βιβλιογραφία

[1] Χ.Γ.Τσίτουρας Γ.Σ. Παπαγεωργίου. *Αριθμητική Ανάλυση με εφαρμογές σε Matlab και Mathematica*. Εκδόσεις Συμεών, 2000.

[2] Wolfgang Christian. *Open Source Physics: A User's Guide with Examples*. Pearson-Addison Wesley, 2007. URL http://www.compadre.org/OSP.

[3] Harvey Gould and Jan Tobochnik and Wolfgang Christian. *An Introduction to Computer Simulation Methods*. Pearson - Addison Wesley, 3rd εδιτιον, 2007.

[4] Βασικός ιστοτόπος της εφαρμογής EJS:. http://www.um.es/fem/EjsWiki/index.php.

[5] Βασικός ιστοτόπος της εφαρμογής Tracker:. http://www.cabrillo.edu/ dbrown/tracker/.

[6] Jonathan Knudsen, Patrick Niemeyer. *Learning Java 3rd Edition*. O'Reilly, 2005.

[7] Μερικές πηγές πληροφοριών σχετικά με τη Java στο Διαδίκτυο:.
http://java.sun.com/docs/books/tutorial/
https://java-net.dev.java.net/
http://www.netbeans.org/index.html
http://www.javaworld.com/
http://wiki.java.net/bin/view/Main/WebHome.

[8] Προσωπικός ιστοτόπος του δρ. Fransisco Esquembre:. http://fem.um.es/.

[9] David J. Griffiths. *Εισαγωγή στην Ηλεκτροδυναμική τομ.1*. ΠΕΚ, Prentice Hall International, 1999.