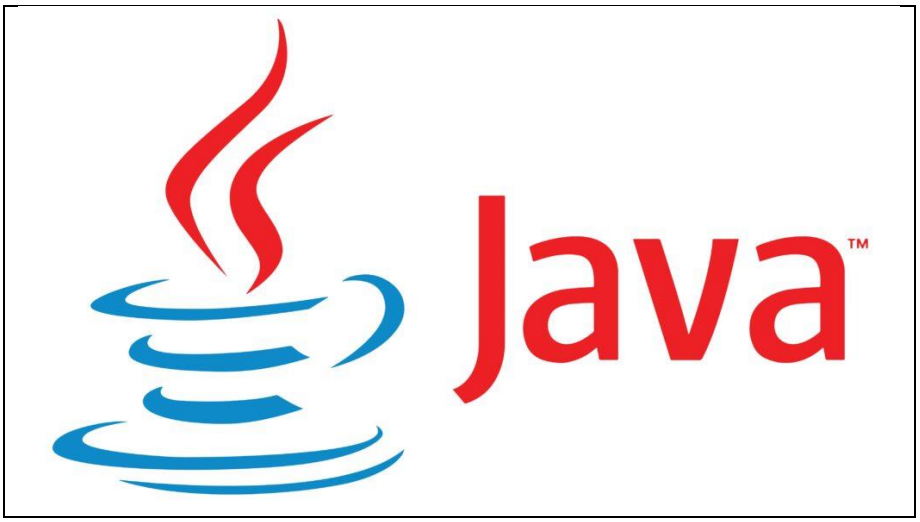
ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

«Ο Αναλυτής JavaParser»



Των φοιτητών
Μάριος Σκουφίτσας 174931 και
Κυριαζής Καραποστόλης 174962

Επιβλέπων
Ονοματεπώνυμο Ιγνάτιος
Δεληγιάννης
Βαθμίδα Καθηγητής

Ημερομηνία 20/6/2022

Τίτλος Δ.Ε. Μελέτη βαθμού Αξιοποίησης βιβλιοθηκών λογισμικού
Κωδικός Δ.Ε. 21293
Ονοματεπώνυμο φοιτητών Μάριος Σκουφίτσας, Κυριαζής Καραποστόλης
Ονοματεπώνυμο εισηγητή Ιγνάτιος Δεληγιάννης
Ημερομηνία ανάληψης Δ.Ε. 13/10/2021
Ημερομηνία περάτωσης Δ.Ε. 20/6/2022

Βεβαιώνω ότι είμαι ο συγγραφέας αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, έχω καταγράψει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών, εικόνων και κειμένου, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επιπλέον, βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά, ειδικά ως διπλωματική εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.

Πρόλογος

Εισαγωγή

Η παρακάτω πτυχιακή εργασία έχει ως σκοπό την μελέτη και την υλοποίηση μια εφαρμογής για την εύρεση μεθόδων βιβλιοθηκών από ένα άλλο project. Η εφαρμογή έχει τη δυνατότητα να δέχεται ένα project του χρήστη, στη συνέχεια να αναλύει όλες τις μεθόδους που γίνονται import και τέλος να μας εμφανίζει το ποσοστό απο αυτές που χρησιμοποιούνται στο project.

Σύντομη Περιγραφή της αρχιτεκτονικής

Η γλώσσα προγραμματισμού που χρησιμοποιήθηκε είναι η java με το εργαλείο neatbeans. Στη συνέχεια οι βιβλιοθήκες που χρησιμοποιήθηκαν για το parse του κώδικα είναι ο javaparser-symbol-solver-core-3.14.9. Για να λειτουργήσει η βιβλιοθήκη που προαναφέραμε χρειάστηκε η εγκατάσταση του Maven. Επιπλέων το Maven χρειάστηκε για να μετατρέψουμε τα jar αρχεία του καλούντος project σε source αρχεία ώστε να μπορέσουμε να δούμε τις μεθόδους που περιλαμβάνει οι βιβλιοθήκες. Παράλληλα, για τη διαχείριση , κοινοποίηση αλλά και το backup του project μας χρησιμοποιήθηκε το github. Τέλος χρησιμοποιήθηκε το swing για τη δημιουργία του GUI(Graphical User Interface) της εφαρμογής μας.

Περίληψη

Στην παρούσα εργασία η θεματολογία αφορούσε τον αναλυτή JavaParser και την βιβλιοθήκη JavaSymbolSolver της γλώσσας προγραμματισμού Java, ως εργαλεία ανάγνωσης, πρόσβασης, ανάλυσης, τροποποίησης και εκτύπωσης των αποτελεσμάτων που αφορούν έναν πηγαίο κώδικα.

Πολλές φορές, οι προγραμματιστές επιθυμούν να δουν την ανάλυση κάθε κόμβου και επιπέδου του προγράμματος που δημιουργούν σε κάθε βήμα εκτέλεσης που πραγματοποιείται. Συγκεκριμένα, η παρούσα εργασία εξετάζει μέσω του JavaParser τρεις μετρικές που αφορούν βιβλιοθήκες JAR και κλήσεις μεθόδων.

Η πρώτη μετρική που εξετάστηκε ήταν ο *αριθμός των μεθόδων* που βρίσκονται σε κάθε βιβλιοθήκη JAR που βρίσκεται στον κατάλογο του project και η εύρεση γίνεται με τη χρήση των MethodDeclarations. Για παράδειγμα, εάν έχουμε τρια JAR στο project μας και το κάθε JAR έχει 1500 μεθόδους, ο συνολικός αριθμός των μεθόδων για το project θα είναι 4500.

Η δεύτερη μετρική αφορά στην *εύρεση των μεθόδων* του project με τη χρήση των MethodCallExpressions *που συσχετίζονται με τις βιβλιοθήκες JAR*. Για παράδειγμα, εάν έχουμε συνολικά 40 MethodCallExpressions, τα 10 ανήκουν στο πρώτο JAR, τα 20 ανήκουν στο δεύτερο JAR και τα υπόλοιπα ανήκουν στο τρίτο JAR. Με βάση αυτούς τους αριθμούς, προκύπτει ένα ποσοστό από την διαίρεση (κλάσμα) με αριθμητή τον αριθμό των μεθόδων για το συγκεκριμένο JAR (π.χ. το πρώτο) και παρονομαστή τον συνολικό αριθμό των μεθόδων που βρίσκονται στο συγκεκριμένο JAR (στο πρώτο). Από αυτή την διαίρεση προκύπτει ένα ποσοστό χρήσης των μεθόδων του συγκεκριμένου JAR στο project που εκτελούμε.

Η τελευταία μετρική βασίζεται στην δεύτερη μετρική, εφόσον βρεθούν οι μέθοδοι στο κάθε JAR, πραγματοποιείται μια διαδικασία κατά την οποία γίνεται μια περαιτέρω *εύρεση με εμφωλευμένο τρόπο για το ποιες παραπάνω μέθοδοι βρίσκονται σε εκείνο το σημείο* πάλι με τη χρήση του MethodCallExpresion. Για παράδειγμα, σε περίπτωση που μια μέθοδος αποτελούσε εμφωλευμένα παραπάνω μεθόδους και αυτές βρίσκονταν μέσα στο ίδιο JAR, αυτές εκτυπώνονται ως αποτέλεσμα της τρίτης μετρικής.

«The JavaParser Analyzer»

«Marios Skoufitsas and Kyriazis Karapstolis»

(στην αγγλική γλώσσα)

Abstract

The Java programming language's JavaSymbolSolver library and JavaParser analyzer—tools for reading, retrieving, analyzing, changing, and publishing the results presented in the source code—were the subject of this study.

Developers frequently request that each executed execution step disclose the analysis of each node and program level. This article specifically looks at three JAR libraries and method call metrics using JavaParser.

The number of methods found in each JAR library in the project directory was the first metric to be looked at. This was done using MethodDeclarations. For instance, if our project has three JARs and each JAR contains 1500 methods, the project will have a total of 4500 methods.

Finding project methods that use MethodCallExpressions connected to JAR libraries is the subject of the second metric. For instance, if there are 40 MethodCallExpressions total, 10 of them will be in the first JAR, 20 in the second, and the rest in the third. Based on these figures, a percentage of the division (fraction) is calculated by using the number of methods for the particular JAR as the numerator (for example, the first) and the total number of methods in the particular JAR as the denominator (in the first). A percentage of the specific JAR's methods being used in the project we are working on is the outcome of this division.

The last metric is based on the second metric; following the discovery of the methods in each JAR, a process is carried out in which a deeper nested search is undertaken to determine which of the aforementioned methods are currently employing MethodCallExpresion once more. The third metric, for instance, prints methods that are nested above other methods and are contained in the same JAR.

Ευχαριστίες

Αρχικά θα θέλαμε να ευχαριστήσουμε τον κ. Δεληγιάννη για την εμπιστοσύνη που μας έδειξε για την ανάληψη και ολοκλήρωση αυτής της διπλωματικής. Καθώς επίσης, την οικογένεια μας που μας υποστήριξε σε όλη την διάρκεια των σπουδών μας.

Περιεχόμενα

Κατάλογος Σχημάτων

Συντομογραφίες

API	Application Programming Interface
GUI	Graphical User Interface
JFC	Java Foundation Classes
AWT	Abstract Window Toolkit
AST	Abstract Syntax Tree
JAR	Java ARchive
POM	Project Object Model
SCM	Source Code Management
CVS	Concurrent Versions System
GPL	General Public License
SHA	Secure Hash Algorithm
SHA1	Secure Hask Algorithm 1
ID	Identification
JSON	Javascript Object Notation
XML	eXtensible Markup Language
YAML	Yet Another Markup Language
DOM	Document Object Model

Κεφάλαιο 1ο: Η Γλώσσα Προγραμματισμού Java

Ενσωματωμένα Πακέτα

Το Java API είναι μια βιβλιοθήκη προγραμμένων κλάσεων, δωρεάν στη χρήση, που περιλαμβάνονται στο περιβάλλον ανάπτυξης Java.

Η βιβλιοθήκη περιέχει στοιχεία για τη διαχείριση εισόδου, τον προγραμματισμό της βάσης δεδομένων και πολλά άλλα. Χωρίζεται σε πακέτα και κλάσεις. Αυτό σημαίνει ότι μπορεί να εισαχθεί είτε μια κλάση (μαζί με τις μεθόδους και τα χαρακτηριστικά της), είτε ένα ολόκληρο πακέτο που περιέχει όλες τις κλάσεις που ανήκουν στο καθορισμένο πακέτο. [1]

Για να χρησιμοποιηθεί μια κλάση ή ένα πακέτο από τη βιβλιοθήκη, πρέπει να χρησιμοποιηθεί η λέξη-κλειδί εισαγωγής **import**:

Πίνακας 1.1: Εισαγωγή πακέτων (import) στην Java

```
import package.name.Class;   // Import a single class
import package.name.*;   // Import the whole package
```

Προϋποθέσεις Java και δηλώσεις IF

Η Java υποστηρίζει τις συνήθεις λογικές συνθήκες από τα μαθηματικά:

- Μικρότερο από: a < b
- Μικρότερο ή ίσο με: a <= b
- Μεγαλύτερο από: a > b
- Μεγαλύτερο ή ίσο με: a >= b
- Ίσο με a == b
- Δεν ισούται με: a != b

Αυτές οι συνθήκες μπορούν να χρησιμοποιηθούν για να εκτελέσει ο χρήστης διαφορετικές ενέργειες για διαφορετικές αποφάσεις.

Η Java έχει τις ακόλουθες δηλώσεις υπό όρους:

- Χρήση του if, για να καθοριστεί ένα μπλοκ κώδικα που θα εκτελεστεί, εάν μια καθορισμένη συνθήκη είναι αληθής.
- Χρήση του else, για να καθοριστεί ένα μπλοκ κώδικα που θα εκτελεστεί, εάν η ίδια συνθήκη είναι ψευδής.
- Χρήση του else if, για να καθοριστεί μια νέα συνθήκη για δοκιμή, εάν η πρώτη συνθήκη είναι ψευδής.
- Χρήση του switch, για να καθοριστούν πολλά εναλλακτικά μπλοκ κώδικα που θα εκτελεστούν.

Πίνακας 1.2: Δομές Επιλογής IF

```
if (condition1) {
  // block of code to be executed if condition1 is true
} else if (condition2) {
  // block of code to be executed if the condition1 is false and
condition2 is true
} else {
  // block of code to be executed if the condition1 is false and
condition2 is false
}
```

Δομή Επανάληψης FOR

Όταν είναι γνωστός ο αριθμός βρόχου μέσω ενός μπλοκ κώδικα, γίνεται χρήση του βρόχου for αντί για τον βρόχο επανάληψης while:

Πίνακας 1.3: Δομή Επανάληψης FOR

```
for (statement 1; statement 2; statement 3) {
  // code block to be executed
}
```

Η δήλωση 1 (statement 1), εκτελείται (μία φορά) πριν από την εκτέλεση του μπλοκ κώδικα.

Η δήλωση 2 (statement 2), ορίζει την συνθήκη για την εκτέλεση του μπλοκ κώδικα.

Η δήλωση 3 (statement 3), εκτελείται (κάθε φορά) μετά την εκτέλεση του μπλοκ κώδικα.

Πίνακες

Οι πίνακες χρησιμοποιούνται για την αποθήκευση πολλαπλών τιμών σε μια μεμονωμένη μεταβλητή, αντί να δηλώνουν ξεχωριστές μεταβλητές για κάθε τιμή.

Για να γίνει δήλωση ενός πίνακα, ορίζεται ο τύπος της μεταβλητής με αγκύλες:

Πίνακας 1.4: Δημιουργία Άδειου Πίνακα Array

```
String[] cars;
```

Τώρα έχει δηλωθεί μια μεταβλητή που περιέχει έναν πίνακα συμβολοσειρών (String). Για να εισαχθούν τιμές σε αυτόν, μπορούμε να χρησιμοποιήσουμε έναν πίνακα κυριολεκτικά με την τοποθέτηση τιμών σε μια λίστα διαχωρισμένη με κόμμα, μέσα σε άγκιστρα:

Πίνακας 1.5: Δημιουργία Πίνακα Array με Αλφαριθμητικά στοιχεία

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
```

Για να δημιουργηθεί ένας πίνακας ακεραίων:

Πίνακας 1.6: Δημιουργία Πίνακα Array με Ακέραια στοιχεία

```
int[] myNum = {10, 20, 30, 40};
```

Πρόσβαση Στοιχείων Πίνακα

Μπορεί να γίνει πρόσβαση σε ένα στοιχείο πίνακα με αναφορά στον αριθμό του δείκτη. Αυτή η δήλωση έχει πρόσβαση στην τιμή του πρώτου στοιχείου στην παρακάτω λίστα αυτοκινήτων:

Πίνακας 1.7: Πρόσβαση Πίνακα

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
System.out.println(cars[0]);
// Outputs Volvo
```

ArrayList

Η κλάση ArrayList είναι ένας πίνακας με δυνατότητα αλλαγής μεγέθους, ο οποίος βρίσκεται στο πακέτο java.util.

Η διαφορά μεταξύ ενός ενσωματωμένου πίνακα και ενός ArrayList στην Java, είναι ότι το μέγεθος ενός πίνακα δεν μπορεί να τροποποιηθεί (π.χ. αν ο χρήστης θέλει να προσθέσει ή να αφαιρέσει στοιχεία σε/από έναν πίνακα, πρέπει να δημιουργήσει έναν νέο πίνακα).

Αντιθέτως, στοιχεία μπορούν να προστεθούν και να αφαιρεθούν από μια ArrayList όποτε είναι επιθυμητό.

Η σύνταξη είναι λίγο διαφορετική:

Πίνακας 1.8: Δημιουργία ArrayList

```
import java.util.ArrayList; // import the ArrayList class

ArrayList<String> cars = new ArrayList<String>(); // Create an ArrayList
object
```

Νήματα

Το Νήμα (Thread) επιτρέπει σε ένα πρόγραμμα να λειτουργεί πιο αποτελεσματικά, κάνοντας πολλά πράγματα ταυτόχρονα.

Τα νήματα μπορούν να χρησιμοποιηθούν για την εκτέλεση περίπλοκων εργασιών στο παρασκήνιο (background) χωρίς διακοπή του βασικού προγράμματος.

Δημιουργία Νήματος

Υπάρχουν δύο τρόποι για να δημιουργηθεί ένα νήμα.

Μπορεί είτε να δημιουργηθεί επεκτείνοντας την κλάση Thread και παρακάμπτοντας τη μέθοδο run():

Πίνακας 1.9: Δημιουργία Thread Α' Τρόπος

```
public class Main extends Thread {
  public void run() {
    System.out.println("This code is running in a thread");
  }
}
```

Ένας άλλος τρόπος για να δημιουργηθεί ένα νήμα, είναι να εφαρμοστεί η διεπαφή Runnable:

Πίνακας 1.10: Δημιουργία Thread Β' Τρόπος

```
public class Main implements Runnable {
  public void run() {
    System.out.println("This code is running in a thread");
  }
}
```

Εκτελέσιμα Νήματα

Εάν η κλάση επεκτείνει την κλάση Thread, το νήμα μπορεί να εκτελεστεί καλώντας τη μέθοδο start():

Πίνακας 1.11: Εκτέλεση Thread

```
public class Main extends Thread {
  public static void main(String[] args) {
    Main thread = new Main();
    thread.start();
    System.out.println("This code is outside of the thread");
  }
  public void run() {
    System.out.println("This code is running in a thread");
  }
}
```

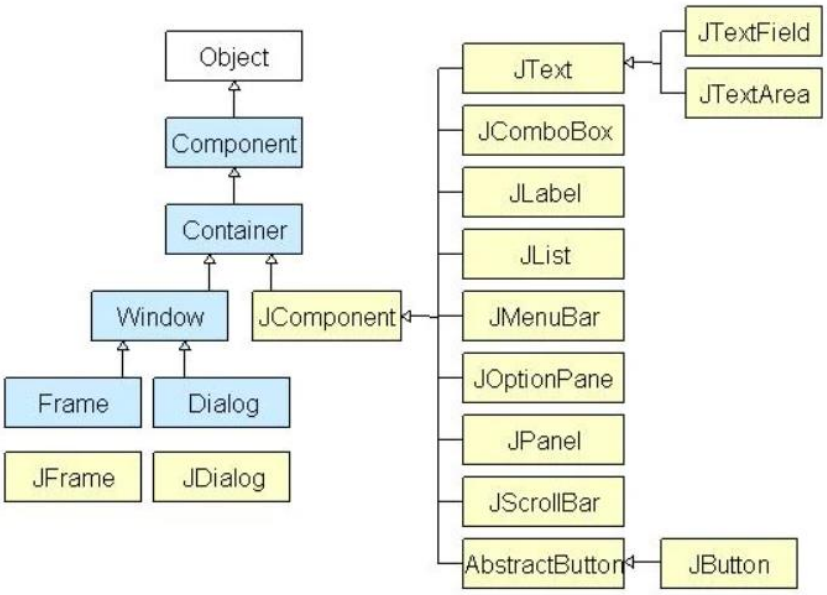
GUI

Το GUI (Graphical User Interface) είναι ένα εύχρηστο οπτικό εργαλείο για εφαρμογές Java. Αποτελείται κυρίως από γραφικά στοιχεία όπως κουμπιά, ετικέτες, παράθυρα κ.λπ. μέσω των οποίων ο χρήστης μπορεί να αλληλεπιδράσει με μια εφαρμογή. Το GUI παίζει σημαντικό ρόλο στη δημιουργία εύκολων διεπαφών για εφαρμογές Java.

Swing

Το Swing είναι μια εργαλειοθήκη γραφικής διεπαφής χρήστη (GUI), που περιλαμβάνει τα στοιχεία GUI (Graphical User Interface). Το Swing παρέχει ένα πλούσιο σύνολο γραφικών στοιχείων και πακέτων για τη δημιουργία εξελιγμένων στοιχείων GUI για εφαρμογές Java. Αποτελεί είναι μέρος των Java Foundation Classes (JFC), το οποίο είναι ένα API για προγραμματισμό Java GUI που παρέχει δυνατότητες GUI. [2]

Η βιβλιοθήκη Java Swing είναι χτισμένη πάνω από το Java Abstract Widget Toolkit (AWT), ένα παλαιότερο πακέτο εργαλείων GUI που εξαρτάται από την πλατφόρμα. Ο χρήστης μπορεί να χρησιμοποιήσει τα στοιχεία προγραμματισμού Java ενός απλού GUI, όπως ένα κουμπί ή ένα πλαίσιο κειμένου από τη βιβλιοθήκη και δεν χρειάζεται να δημιουργεί τα στοιχεία από την αρχή.



Σχήμα 1.1: Διάγραμμα Ιεραρχίας Κλάσης Swing της Java

Κλάση Container

Οι κλάσεις container είναι κλάσεις που μπορούν να περιέχουν άλλα στοιχεία. Έτσι, για τη δημιουργία ενός Java Swing GUI, ο χρήστης χρειάζεται τουλάχιστον ένα αντικείμενο container.

Υπάρχουν 3 τύποι container Java Swing.

- Πάνελ (Panel)

  Είναι ένα καθαρό/διάφανο δοχείο και δεν αποτελεί παράθυρο από μόνο του. Ο μοναδικός σκοπός ενός πάνελ είναι να οργανώσει τα στοιχεία σε ένα παράθυρο.

- Πλαίσιο (Frame)

  Είναι ένα πλήρως λειτουργικό παράθυρο με τον τίτλο και τα εικονίδια του.

- Διάλογος (Dialog)

  Μπορεί να θεωρηθεί σαν ένα αναδυόμενο παράθυρο που αναδύεται όταν πρέπει να εμφανιστεί ένα μήνυμα. Δεν είναι ένα πλήρως λειτουργικό παράθυρο όπως το Frame.

Δημιουργία GUI

Παρακάτω πραγματοπιείται η δημιουργία ενός GUI σε Java με παραδείγματα Swings in Java.

Βήμα 1) Αντιγραφή του κώδικα σε ένα πρόγραμμα επεξεργασίας

Στο πρώτο βήμα, γίνεται αντιγραφή του παρακάτω κώδικα σε ένα πρόγραμμα επεξεργασίας:

Πίνακας 1.12: Δημιουργία Swing

```
import javax.swing.*;
class gui{
    public static void main(String args[]){
       JFrame frame = new JFrame("My First GUI");
       frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
       frame.setSize(300,300);
       JButton button = new JButton("Press");
       frame.getContentPane().add(button); // Adds Button to content pane
of frame
       frame.setVisible(true);
    }
}
```

Βήμα 2) Εκτέλεση του κώδικα

Στο συγκεκριμένο βήμα, γίνεται αποθήκευση, μεταγλώττιση και εκτέλεση του κώδικα.

Βήμα 3) Αντιγραφή του ακόλουθου κώδικα σε ένα πρόγραμμα επεξεργασίας

Έπειτα, θα προστεθεί ένα κουμπί στο πλαίσιο και θα γίνει αντιγραφή του ακόλουθου κώδικα σε ένα πρόγραμμα επεξεργασίας:

Πίνακας 1.13: Τοποθέτηση 1ου Button στο Swing

```
import javax.swing.*;
   class gui{
      public static void main(String args[]){
        JFrame frame = new JFrame("My First GUI");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(300,300);
       JButton button1 = new JButton("Press");
       frame.getContentPane().add(button1);
       frame.setVisible(true);
     }
}
```

Βήμα 4) Εκτέλεση του κώδικα

Στο βήμα αυτό, γίνεται εκτέλεση του κώδικα και θα προκύψει ένα κουμπί.

Σχήμα 1.2: Παράθυρο GUI με Κουμπί

Βήμα 5) Πρόσθεση δύο κουμπιών

Έπειτα, γίνεται αντιγραφή του παρακάτω κώδικα σε ένα πρόγραμμα επεξεργασίας:

Πίνακας 1.14: Τοποθέτηση 2ου Button στο Swing

```
import javax.swing.*;
class gui{
      public static void main(String args[]){
            JFrame frame = new JFrame("My First GUI");
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.setSize(300,300);
           JButton button1 = new JButton("Button 1");
           JButton button2 = new JButton("Button 2");
           frame.getContentPane().add(button1);
           frame.getContentPane().add(button2);
           frame.setVisible(true);
     }
}
```

Βήμα 6) Αποθήκευση και εκτέλεση του προγράμματος

Στη συνέχεια, γίνεται αποθήκευση, μεταγλώττιση και εκτέλεση του προγράμματος ξανά.

Βήμα 7) Έλεγχος της εξόδου

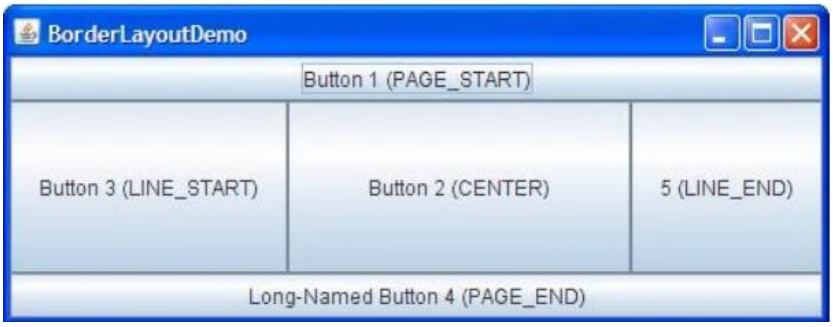
Έλεγχος απροσδόκητης εξόδου, διότι τα κουμπιά μπορεί να επικαλύπτονται.

Layout Manager

Ο Διαχειριστής διάταξης χρησιμοποιείται για τη διάταξη (ή τακτοποίηση) των στοιχείων Java GUI μέσα σε ένα container. Υπάρχουν πολλοί διαχειριστές διάταξης, αλλά παρακάτω δίνονται οι πιο συχνά χρησιμοποιούμενοι.

Java BorderLayout

Το BorderLayout τοποθετεί τα στοιχεία σε έως και πέντε περιοχές: επάνω, κάτω, αριστερά, δεξιά και στο κέντρο. Είναι ο προεπιλεγμένος διαχειριστής διάταξης για κάθε Java JFrame.



Σχήμα 1.3: Παράθυρο GUI με Κουμπιά σε 5 περιοχές

Java FlowLayout

Το FlowLayout από την άλλλη, είναι ο προεπιλεγμένος διαχειριστής διάταξης για κάθε JPanel. Απλώς "απλώνει" τα στοιχεία σε μια σειρά, το ένα μετά το άλλο.

Σχήμα 1.4: Παράθυρο GUI με Κουμπιά στην ίδια σειρά

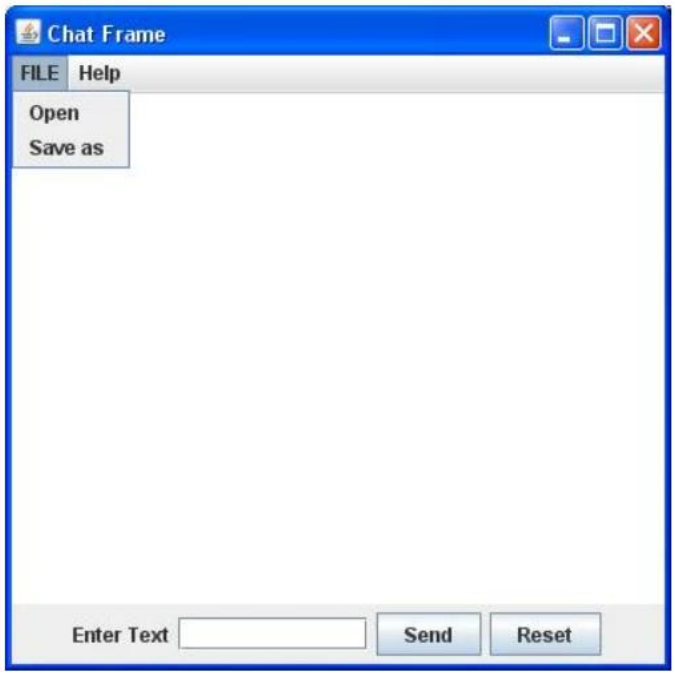
Java GridBagLayout

Το GridBagLayout είναι η πιο εξελιγμένη από όλες τις διατάξεις. Ευθυγραμμίζει τα στοιχεία τοποθετώντας τα μέσα σε ένα πλέγμα κελιών, επιτρέποντας στα συστατικά (components) να εκτείνονται σε περισσότερα από ένα κελιά.

Σχήμα 1.5: Παράθυρο GUI με Κουμπιά σε Πλέγμα (Grid)

Βήμα 8) Δημιουργία πλαισίου συνομιλίας

Παρακάτω ακολουθεί η προσπάθεια δημιουργίας πλαισίου:



Σχήμα 1.6: Παράθυρο GUI με Πλαίσιο (Frame)

Πίνακας 1.15: Δημιουργία Frame

```
//Usually you will require both swing and awt packages
// even if you are working with just swings.
import javax.swing.*;
import java.awt.*;
class gui {
    public static void main(String args[]) {

        //Creating the Frame
        JFrame frame = new JFrame("Chat Frame");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(400, 400);

        //Creating the MenuBar and adding components
        JMenuBar mb = new JMenuBar();
        JMenu m1 = new JMenu("FILE");
        JMenu m2 = new JMenu("Help");
        mb.add(m1);
        mb.add(m2);
        JMenuItem m11 = new JMenuItem("Open");
        JMenuItem m22 = new JMenuItem("Save as");
        m1.add(m11);
        m1.add(m22);

        //Creating the panel at bottom and adding components
        JPanel panel = new JPanel(); // the panel is not visible in
output
        JLabel label = new JLabel("Enter Text");
        JTextField tf = new JTextField(10); // accepts upto 10 characters
        JButton send = new JButton("Send");
        JButton reset = new JButton("Reset");
        panel.add(label); // Components Added using Flow Layout
        panel.add(tf);
        panel.add(send);
        panel.add(reset);

        // Text Area at the Center
        JTextArea ta = new JTextArea();

        //Adding Components to the frame.
        frame.getContentPane().add(BorderLayout.SOUTH, panel);
        frame.getContentPane().add(BorderLayout.NORTH, mb);
        frame.getContentPane().add(BorderLayout.CENTER, ta);
        frame.setVisible(true);
    }
}
```

JavaParser

Οι βιβλιοθήκες JavaParser είναι ένας εύκολος τρόπος εξαγωγής και ανάλυσης Αφηρημένων Συντακτικών Δέντρων (AST) στην Java. Μπορεί να αναλύσει και να μεταμορφώσει τον κώδικα.

Αναλύοντας ένα project

Το JavaParser, ως βιβλιοθήκη προσφέρει πολλές λειτουργίες για την ανάλυση ενός προγράμματος Java. Ωστόσο, εάν υπάρχουν περισσότερα από ένα αρχεία προς ανάλυση που διανέμονται σε πολλούς φακέλους ή μια εφαρμογή που είναι κατανεμημένη σε πολλά πακέτα, πρέπει να καθοριστούν οι φάκελοι προορισμού. Ένας από τους τρόπους για την ανάλυση του έργου είναι η χρήση του SourceRoot, ο οποίος διαχειρίζεται όλους τους φακέλους που περιέχουν αρχεία Java ως ρίζες. [3]

Πίνακας 1.16: Διαχωρισμός με SourceRoot

```
Path projectRoot =
FileSystems.getDefault().getPath(“path\\to\\main\\root\\”);
String[] roots = new String[] {"subdirectory1\\after\\root",
"subdirectory2\\after\\root"};
for (String root : roots) {
SourceRoot sourceRoot = new SourceRoot(projectRoot.resolve(root));
sourceRoot.setParserConfiguration(parserConfiguration);
List<CompilationUnit> allCU=
parseResults.stream().filter(ParseResult::isSuccessful).map(r ->
r.getResult().get()).collect(Collectors.toList());
Iterator<CompilationUnit> cuIter = allCU.iterator();
while (cuIter.hasNext()) {
CompilationUnit compilationunit = cuIter.next();
//work with compilationunit
}
```

Ένα έργο μπορεί να έχει εξωτερικές εξαρτήσεις (dependencies) που θα πρέπει επίσης να υπάρχουν στην καθορισμένη διαδρομή. Αυτές οι εξαρτήσεις προσαρτώνται στον TypeSolver.

Πίνακας 1.17: Χρήση του TypeSolver

```
CombinedTypeSolver typeSolver = new CombinedTypeSolver(new
ReflectionTypeSolver(), new CombinedTypeSolver());
typeSolver.add(new JavaParserTypeSolver(new
File(“directory\\containing\\javafile\\dependency”)));
typeSolver.add(new JarTypeSolver("path\\to\\jar\\dependency.jar"));
JavaSymbolSolver symbolSolver = new JavaSymbolSolver(typeSolver);
ParserConfiguration parserConfiguration = new
ParserConfiguration().setSymbolResolver(symbolSolver);
StaticJavaParser.setConfiguration(parserConfiguration);JavaParser parser
= new JavaParser(parserConfiguration)
```

Έτσι, σε περίπτωση εμφάνισης του σφάλματος "Exception in thread "main" UnsolvedSymbolException", πιθανότατα δεν έχει προστεθεί κάποια εξάρτηση στον TypeSolver. Το ίδιο σφάλμα εμφανίζεται επίσης εάν οριστεί μια νέα μέθοδος στον κώδικα αλλά προηγουμένως δεν έχει αναλυθεί.

Προσεκτική Τροποποίηση του AST

Όπως και άλλα αντικείμενα στη Java, εάν ένα AST (Abstract Syntax Tree) τροποποιηθεί χρησιμοποιώντας JavaParser ενώ πραγματοποιείται ήδη μια επανάληψη πάνω σε αυτό, ενδέχεται να προκύψει σφάλμα εξαίρεσης (exception) "Ταυτόχρονη εξαίρεση τροποποίησης". Επομένως, καλύτερα να διαχωριστεί το κομμάτι της ανάλυσης από το κομμάτι της τροποποίησης.

Προσοχή στις JAR εκδόσεις

Ένας άλλος λόγος για σφάλματα θα μπορούσε να είναι οι αναντιστοιχίες εκδόσεων σε διαφορετικές βιβλιοθήκες. Συνιστάται η χρήση JavaParser JAR με εκδόσεις μεγαλύτερες από 3.20. Σε περίπτωση που προκύψει σφάλμα καταγραφής, θα πρέπει να γίνει έλεγχος για το αν χρησιμοποιείτε το slf4j-ext (έκδοση 1.7.28). Για την ευκολότερη συντήρηση των JAR, θα πρέπει να διατηρούνται στον ίδιο φάκελο. Αυτό θα βοηθήσει όχι μόνο στο να προστεθούν επαναληπτικά όλες οι εξαρτήσεις αντί να κωδικοποιηθούν, αλλά και να προσδιοριστεί εάν υπάρχουν συγκρούσεις έκδοσης.

Μόλις ρυθμιστεί το JavaParser, μπορούν να αξιοποιηθούν όλες οι λειτουργίες του. Με λίγα λόγια, το JavaParser προσφέρει ανάλυση, μετατροπή και δημιουργία πηγαίου κώδικα Java.

Προαπαιτούμενα Maven

Είναι απαραίτητο να γίνει κατανόηση του πώς να εγκατασταθεί το λογισμικό Maven στον υπολογιστή του χρήστη. Σε περίπτωση που δεν υπάρχει η σχετική γνώση εγκατάστασης, θα χρειαστεί να ζητηθεί βοήθεια από κάποιον άλλον χρήστη που γνωρίζει να εγκαταστήσει την συγκεκριμένη τεχνολογία. [4]

Εγκατάσταση Maven

Το Maven είναι ένα εργαλείο Java, επομένως θα χρειαστεί να έχει ήδη εγκατασταθεί η γλώσσα προγραμματισμού Java για να μπορέσει να συνεχίσει ο χρήστης.

Πρώτα, γίνεται λήψη του Maven και έπειτα εκτελούνται οι οδηγίες εγκατάστασης. Μετά από αυτό, πληκτρολογούνται τα ακόλουθα σε ένα τερματικό ή σε μια γραμμή εντολών:

Πίνακας 1.18: Εντολή Έκδοσης Maven

```
mvn --version
```

Θα ακολουθήσει εκτύπωση της εγκατεστημένης έκδοσης του Maven, για παράδειγμα:

Πίνακας 1.19: Εκτύπωαη Έκδοσης Maven

```
Apache Maven 3.6.3 (cecedd343002696d0abb50b32b541b8a6ba2883f)

Maven home: D:\apache-maven-3.6.3\apache-maven\bin\..

Java version: 1.8.0_232, vendor: AdoptOpenJDK, runtime: C:\Program Files\AdoptOpenJDK\jdk-8.0.232.09-hotspot\jre

Default locale: en_US, platform encoding: Cp1250

OS name: "windows 10", version: "10.0", arch: "amd64", family: "windows"
```

Ανάλογα με τη ρύθμιση του δικτύου του χρήστη, μπορεί να χρειαστεί επιπλέον διαμόρφωση. Η καλύτερη επιλογή είναι η ανάγνωση του Οδηγού (Guide) για τη διαμόρφωση του Maven.

Σε περίπτωση χρήσης Windows, θα πρέπει να εξετασθούν οι προϋποθέσεις των Windows για να βεβαιωθεί ότι ο χρήστης είναι έτοιμος να χρησιμοποιήσει το Maven στα Windows.

Δημιουργία Έργου Maven

Αρχικά, χρειάζεται ένας τόπος διαμονής για το project. Δημιουργείται ένας κατάλογος σε μια τοποθεσία και ξεκινάει ένα κέλυφος (shell) σε αυτόν τον κατάλογο. Στη γραμμή εντολών, εκτελείται ο ακόλουθος στόχος Maven:

Πίνακας 1.20: Κέλυφος (Shell) Maven

```
mvn archetype:generate -DgroupId=com.mycompany.app -DartifactId=my-app -
DarchetypeArtifactId=maven-archetype-quickstart -DarchetypeVersion=1.4 -
DinteractiveMode=false
```

Εάν έχει προηγηθεί μόλις η εγκαταστήσει του Maven, μπορεί να χρειαστεί λίγος χρόνος στην πρώτη εκτέλεση. Αυτό συμβαίνει επειδή η Maven κατεβάζει τα πιο πρόσφατα αρχεία στο τοπικό μέρος αποθήκευσης του υπολογιστή. Μπορεί επίσης να χρειαστεί να εκτελεστεί η εντολή μερικές φορές προτού πετύχει. Αυτό συμβαίνει επειδή ο απομακρυσμένος διακομιστής ενδέχεται να λήξει πριν ολοκληρωθούν οι λήψεις που πραγματοποιούνται.

Θα παρατηρηθεί ότι ο στόχος δημιουργίας δημιούργησε έναν κατάλογο με το ίδιο όνομα που δόθηκε με το artifactId. Επομένως, θα χρειαστεί μετάβαση σε αυτό τον κατάλογο:

Πίνακας 1.21: Μετάβαση Καταλόγου Maven

```
cd my-app
```

Κάτω από αυτόν τον κατάλογο παρατηρείται η ακόλουθη τυπική δομή έργου:

Πίνακας 1.22: Τυπική Δομή Έργου Maven

```
my-app
|-- pom.xml
`-- src
    |-- main
    |   `-- java
    |       `-- com
    |           `-- mycompany
    |               `-- app
    |                   `-- App.java
    `-- test
        `-- java
            `-- com
                `-- mycompany
                    `-- app
                        `-- AppTest.java
```

Ο κατάλογος **src/main/java** περιέχει τον πηγαίο κώδικα του έργου, ο κατάλογος **src/test/java** περιέχει τη δοκιμαστική πηγή και το αρχείο **pom.xml** είναι το μοντέλο αντικειμένου του έργου ή αλλιώς το POM του έργου.

POM

Το αρχείο pom.xml είναι ο πυρήνας της διαμόρφωσης ενός έργου στο Maven. Είναι ένα ενιαίο αρχείο διαμόρφωσης που περιέχει την πλειονότητα των πληροφοριών που απαιτούνται για την κατασκευή ενός έργου με τον τρόπο που θέλει ο χρήστης. Το POM είναι τεράστιο και μπορεί να είναι τρομακτικό στην πολυπλοκότητά του, αλλά δεν είναι απαραίτητο να κατανοηθούν όλες οι περιπλοκές για να χρησιμοποιηθεί αποτελεσματικά. Το POM του συγκεκριμένου έργου είναι:

Πίνακας 1.23: POM του Έργου Maven

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <version>1.0-SNAPSHOT</version>

  <properties>
    <maven.compiler.source>1.7</maven.compiler.source>
    <maven.compiler.target>1.7</maven.compiler.target>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.12</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

Έπειτα, εκτελέστηκε το αρχέτυπο του στόχου Maven:generate και ο χρήστης πέρασε σε διάφορες παραμέτρους σε αυτόν τον στόχο. Το αρχέτυπο του προθέματος είναι το πρόσθετο που παρέχει τον στόχο. Αυτός ο στόχος archetype:generate δημιούργησε ένα απλό έργο βασισμένο σε ένα αρχέτυπο maven-archetype-quickstart. Αρκεί να τονιστεί προς το παρόν, ότι ένα πρόσθετο είναι μια συλλογή στόχων με έναν γενικό κοινό σκοπό. Για παράδειγμα το jboss-maven-plugin, του οποίου ο σκοπός είναι "να ασχοληθεί με διάφορα είδη jboss".

Κατασκευή Έργου

Πίνακας 1.23: Κατασκευή (Πακέτου) Έργου Maven

```
mvn package
```

Η γραμμή εντολών θα εκτυπώσει διάφορα αποτελέσματα και θα λήξει ως εξής:

Πίνακας 1.24: Εκτύπωση Αποτελεσμάτων Maven

```
...
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  2.953 s
[INFO] Finished at: 2019-11-24T13:05:10+01:00
[INFO] ------------------------------------------------------------------------
```

Σε αντίθεση με την πρώτη εντολή που εκτελείται (αρχέτυπο:δημιουργία), η δεύτερη είναι απλώς μια λέξη - πακέτο. Αντί για τον στόχο, αυτή είναι μια απλή φάση. Μια φάση, δηλαδή, είναι ένα βήμα στον κύκλο ζωής της κατασκευής, που είναι μια διατεταγμένη ακολουθία φάσεων. Όταν δίνεται μια φάση, ο Maven εκτελεί κάθε φάση στην ακολουθία μέχρι και αυτή που έχει οριστεί. Για παράδειγμα οι φάσεις που εκτελούνται πραγματικά είναι:

1. Η φάση επικύρωσης
2. Η φάση δημιουργίας – πηγές
3. Η φάση διαδικασίας – πηγές
4. Η φάση παραγωγής – πόροι
5. Η φάση διαδικασίας – πόροι
6. Η φάση μεταγλώττισης

Ο χρήστης πλέον μπορεί να δοκιμάσει το πρόσφατα μεταγλωττισμένο και συσκευασμένο JAR με την ακόλουθη εντολή:

Πίνακας 1.25: Δοκιμή JAR του Maven

```
java -cp target/my-app-1.0-SNAPSHOT.jar com.mycompany.app.App
```

Παρακάτω ακολουθεί η εκτύπωση:

Πίνακας 1.26: Εκτύπωση Δοκιμής JAR του Maven

```
Hello World!
```

Για Java 9 ή νεότερες εκδόσεις

Από προεπιλογή, η έκδοση του Maven που διαθέτετε μπορεί να χρησιμοποιεί μια παλιά έκδοση του maven-compiler-plugin που δεν είναι συμβατή με Java 9 ή τις νεότερες εκδόσεις. Αναφορικά με την Java 9 ή κάποια νεότερη έκδοση, θα πρέπει τουλάχιστον να χρησιμοποιηθεί η έκδοση 3.6.0 της προσθήκης maven-compiler και να οριστεί η ιδιότητα maven.compiler.release στην έκδοση Java που στοχεύει ο χρήστης (π.χ. 9, 10, 11, 12….).

Στο παρακάτω παράδειγμα, έχει διαμορφωθεί το έργο Maven ώστε να χρησιμοποιεί την έκδοση 3.8.1 του maven-compiler-plugin και να στοχεύει την Java 11:

Πίνακας 1.27: Προσαρμογή Έκδοσης 3.8.1 του Maven

```
    <properties>
        <maven.compiler.release>11</maven.compiler.release>
    </properties>

    <build>
        <pluginManagement>
            <plugins>
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-compiler-plugin</artifactId>
                    <version>3.8.1</version>
                </plugin>
            </plugins>
        </pluginManagement>
    </build>
```

Εργαλεία Maven

Αν και η παρακάτων δεν αποτελεί μια ολοκληρωμένη λίστα, αυτές είναι οι πιο συνηθισμένες προεπιλεγμένες φάσεις του κύκλου ζωής που εκτελούνται.

- Επικύρωση

Ορθή λειτουργία, όλες οι απαραίτητες πληροφορίες είναι διαθέσιμες.

- Μεταγλώττιση

Μεταγλώττιση πηγαίου κώδικα του έργου.

- Δοκιμή

Δοκιμή του μεταγλωττισμένου πηγαίου κώδικα, χρησιμοποιώντας ένα κατάλληλο πλαίσιο δοκιμής μονάδας. Αυτές οι δοκιμές δεν θα πρέπει να απαιτούν τη συσκευασία ή την ανάπτυξη του κώδικα.

- Πακέτο

Συσκευασία μεταγλωττισμένου κώδικα στη μορφή διανομής του, όπως ένα JAR.

- Έλεγχος Ενσωμάτωσης

Επεξεργασία και ανάπτυξη του πακέτου, εάν είναι απαραίτητο, σε ένα περιβάλλον όπου μπορούν να εκτελεστούν δοκιμές ενοποίησης.

- Επαλήθευση

Εκτέλεση τυχόν ελέγχων για την επαλήθευση ότι το πακέτο είναι έγκυρο και πληρεί τα κριτήρια ποιότητας.

- Εγκατάσταση

Εγκατάσταση του πακέτου στο τοπικό αποθετήριο, για χρήση ως εξάρτηση σε άλλα έργα τοπικά.

- Ανάπτυξη

Γίνεται σε περιβάλλον ενοποίησης ή έκδοσης, αντιγράφει το τελικό πακέτο στο απομακρυσμένο αποθετήριο για κοινή χρήση με άλλους προγραμματιστές και έργα.

Υπάρχουν δύο άλλοι αξιόλογοι κύκλοι ζωής του Maven, πέρα από την προεπιλεγμένη λίστα παραπάνω:

- **Καθαρισμός**: καθαρίζει τα τεχνουργήματα που δημιουργήθηκαν από προηγούμενες κατασκευές
- **Ιστοσελίδα**: δημιουργεί τεκμηρίωση ιστοσελίδας για αυτό το έργο

Οι φάσεις στην πραγματικότητα αντιστοιχίζονται στους υποκείμενους στόχους. Οι συγκεκριμένοι στόχοι που εκτελούνται ανά φάση εξαρτώνται από τον τύπο συσκευασίας του έργου. Για παράδειγμα, το πακέτο εκτελεί jar:jar εάν ο τύπος έργου είναι JAR και war:war εάν ο τύπος έργου είναι WAR.

Ένα ενδιαφέρον πράγμα που πρέπει να σημειωθεί είναι ότι οι φάσεις και οι στόχοι μπορούν να εκτελούνται με τη σειρά:

Πίνακας 1.28: Εκκαθάριση, Αντιγραφή και Συσκευασία Έργου του Maven

```
mvn clean dependency:copy-dependencies package
```

Αυτή η εντολή θα καθαρίσει το έργο, θα αντιγράψει τις εξαρτήσεις και θα συσκευάσει το έργο (φυσικά εκτελώντας όλες τις φάσεις μέχρι το πακέτο).

Δημιουργία Ιστότοπου

Πίνακας 1.29: Δημιουργία Ιστοτόπου (Maven)

```
mvn site
```

Αυτή η φάση δημιουργεί έναν ιστότοπο που βασίζεται σε πληροφορίες σχετικά με το POM του έργου. Τέλος, ο χρήστης μπορεί να δει την τεκμηρίωση που δημιουργείται στον στόχο/ιστότοπο.

Κεφάλαιο 2ο: Το Λογισμικό Git

Git

Το Git είναι ένα σύστημα ελέγχου κατανεμημένων εκδόσεων ανοιχτού (open source) κώδικα. Έχει σχεδιαστεί για να χειρίζεται μικρά έως μεγάλα έργα, με υψηλή ταχύτητα και αποτελεσματικότητα. Αναπτύχθηκε για να συντονίζει την εργασία μεταξύ των προγραμματιστών. Ο έλεγχος έκδοσης επιτρέπει στον χρήστη να παρακολουθεί και να συνεργάζεται με τα μέλη της ομάδας στον ίδιο χώρο εργασίας. [5]

Το Git είναι το θεμέλιο πολλών υπηρεσιών όπως το GitHub και το GitLab, αλλά μπορεί να χρησιμοποιηθεί απλώς το Git χωρίς τις άλλες υπηρεσίες Git. Το Git μπορεί να χρησιμοποιηθεί ιδιωτικά και δημόσια.

Χρησιμοποιείται επίσης ως ένα σημαντικό κατανεμημένο εργαλείο ελέγχου έκδοσης για τα DevOps.

Το Git μαθαίνεται εύκολα και έχει γρήγορη απόδοση. Είναι ανώτερο από άλλα εργαλεία SCM όπως το Subversion, το CVS, το Perforce και το ClearCase.

Χαρακτηριστικά του Git

Σχήμα 2.1: Χαρακτηριστικά του Git

Μερικά αξιοσημείωτα χαρακτηριστικά του Git είναι τα εξής:

- Ανοιχτή πηγή

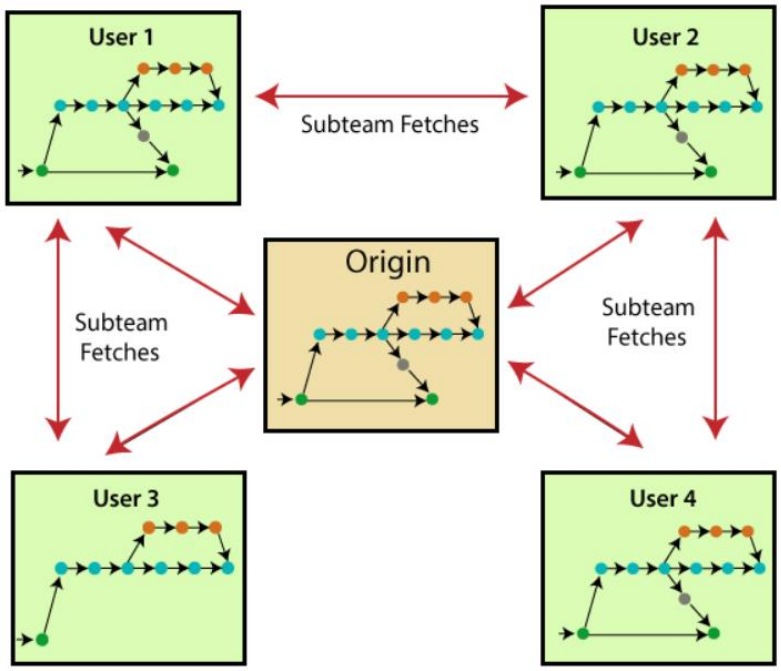
Το Git είναι ένα εργαλείο ανοιχτού κώδικα. Κυκλοφορεί με την άδεια GPL (General Public License).

- Κλιμακούμενος

Το Git είναι επεκτάσιμο, πράγμα που σημαίνει ότι όταν ο αριθμός των χρηστών αυξάνεται, το Git μπορεί εύκολα να χειριστεί τέτοιες καταστάσεις.

- Διανέμονται

Ένα από τα σπουδαία χαρακτηριστικά του Git είναι ότι διανέμεται. Κατανεμημένο σημαίνει ότι αντί να αλλάξει το έργο σε άλλο μηχάνημα, μπορεί να δημιουργηθεί ένας «κλώνος» ολόκληρου του αποθετηρίου. Επίσης, αντί να υπάρχει απλώς ένα κεντρικό αποθετήριο στο οποίο ο χρήστης στέλνει τις αλλαγές και κάθε χρήστης έχει το δικό του αποθετήριο που περιέχει ολόκληρο το ιστορικό δεσμεύσεων του έργου. Δεν χρειάζεται να συνδεθεί στο απομακρυσμένο αποθετήριο, η αλλαγή αποθηκεύεται απλώς στο τοπικό του αποθετήριο. Εάν είναι απαραίτητο, μπορούν να προωθηθούν αυτές οι αλλαγές σε ένα απομακρυσμένο αποθετήριο.



Σχήμα 2.2: Διανομές του Git

- Ασφάλεια

Το Git είναι ασφαλές. Χρησιμοποιεί το SHA1 (Secure Hash Function) για να ονομάσει και να αναγνωρίσει αντικείμενα μέσα στο αποθετήριο του. Τα αρχεία και οι δεσμεύσεις ελέγχονται και ανακτώνται από το άθροισμα ελέγχου κατά τη στιγμή της ολοκλήρωσης αγοράς. Αποθηκεύει το ιστορικό του με τέτοιο τρόπο ώστε το αναγνωριστικό συγκεκριμένων δεσμεύσεων να εξαρτάται από το πλήρες ιστορικό ανάπτυξης που οδηγεί σε αυτήν τη δέσμευση. Μόλις δημοσιευτεί, δεν μπορεί κανένας χρήστης να κάνει αλλαγές στην παλιά του έκδοση.

- Ταχύτητα

Το Git είναι πολύ γρήγορο, επομένως μπορεί να ολοκληρώσει όλες τις εργασίες σε μικρό χρονικό διάστημα. Οι περισσότερες από τις λειτουργίες git γίνονται στο τοπικό αποθετήριο, επομένως παρέχει τεράστια ταχύτητα. Επίσης, ένα κεντρικό σύστημα ελέγχου έκδοσης επικοινωνεί συνεχώς με έναν διακομιστή σε κάποιο μέρος.

Οι δοκιμές απόδοσης που πραγματοποιήθηκαν από τη Mozilla, έδειξαν ότι ήταν εξαιρετικά γρήγορος σε σύγκριση με άλλα VCS. Η ανάκτηση του ιστορικού εκδόσεων από έναν τοπικά αποθηκευμένο χώρο αποθήκευσης, είναι πολύ πιο γρήγορη από την ανάκτησή του από τον απομακρυσμένο διακομιστή.

Το βασικό μέρος του Git είναι γραμμένο σε γλώσσα προγραμματισμού C, το οποίο αγνοεί τα γενικά έξοδα χρόνου εκτέλεσης που σχετίζονται με άλλες γλώσσες υψηλού επιπέδου.

Το Git αναπτύχθηκε για να λειτουργεί στον πυρήνα του Linux. Επομένως, είναι αρκετά ικανό να χειρίζεται αποτελεσματικά μεγάλα αποθετήρια. Από την αρχή, η ταχύτητα και η απόδοση ήταν οι πρωταρχικοί στόχοι του Git.

- Υποστηρίζει τη μη γραμμική ανάπτυξη

Το Git υποστηρίζει απρόσκοπτη διακλάδωση και συγχώνευση, η οποία βοηθά στην απεικόνιση και την πλοήγηση μιας μη γραμμικής ανάπτυξης. Ένας κλάδος στο Git αντιπροσωπεύει μια ενιαία δέσμευση. Μπορεί να κατασκευαστεί η πλήρης δομή του κλάδου με τη βοήθεια της γονικής δέσμευσής του.

- Διακλάδωση και Συγχώνευση

Η διακλάδωση και η συγχώνευση είναι τα σπουδαία χαρακτηριστικά του Git, γεγονός που το κάνει διαφορετικό από τα άλλα εργαλεία SCM. Το Git επιτρέπει τη δημιουργία πολλαπλών κλάδων χωρίς να επηρεάζει το ένα το άλλο. Ένας χρήστης μπορεί να εκτελέσει εργασίες όπως η δημιουργία, η διαγραφή και η συγχώνευση σε κλάδους και αυτές οι εργασίες διαρκούν λίγα μόνο δευτερόλεπτα. Ακολουθούν ορισμένα χαρακτηριστικά που μπορούν να επιτευχθούν με διακλάδωση:

- Μπορεί να δημιουργηθεί ένας ξεχωριστός κλάδος για μια νέα ενότητα του έργου, να δεσμευτεί και να διαγραφεί όποτε γίνει επιθυμητό.
- Μπορεί να υπάρχει ένας κλάδος παραγωγής, ο οποίος διαθέτει πάντα ό,τι μπαίνει στην παραγωγή και μπορεί να συγχωνευθεί για δοκιμή, στον κλάδο δοκιμής.
- Μπορεί να δημιουργηθεί ένας κλάδος επίδειξης για το πείραμα και να γίνει έλγχος αν λειτουργεί. Μπορεί επίσης να αφαιρεθεί αν χρειαστεί.
- Το βασικό πλεονέκτημα της διακλάδωσης είναι εάν γίνει επιθυμητό να προωθηθεί κάτι σε ένα απομακρυσμένο αποθετήριο, δεν χρειάζεται να προωθηθούν όλα τα υποκαταστήματα που υπάρχουν. Μπορούν να επιλεχθούν μερικά από τα υποκαταστήματά ή ακόμη και όλα μαζί.
- Διασφάλιση Δεδομένων

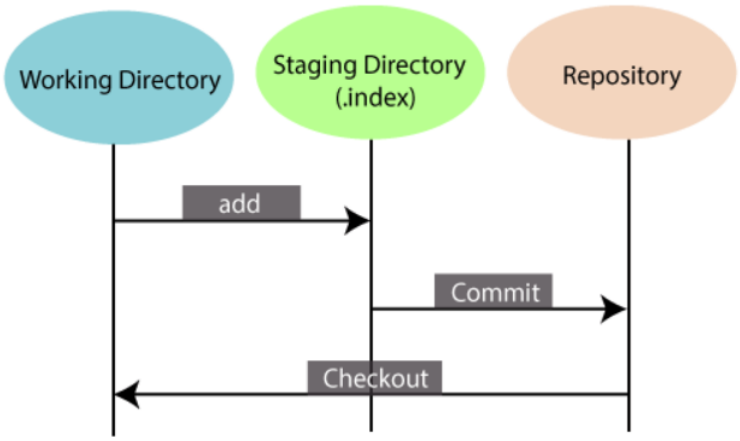
Το μοντέλο δεδομένων Git διασφαλίζει την κρυπτογραφική ακεραιότητα κάθε μονάδας του έργου μας. Παρέχει ένα μοναδικό αναγνωριστικό δέσμευσης σε κάθε δέσμευση μέσω ενός αλγορίθμου SHA.

Μπορούμε να ανακτήσουμε και να ενημερώσουμε το commit με commit ID. Τα περισσότερα από τα κεντρικά συστήματα ελέγχου εκδόσεων δεν παρέχουν τέτοια ακεραιότητα από προεπιλογή.

- Χώρος Σκηνοποίησης

Η περιοχή Staging είναι επίσης μια μοναδική λειτουργικότητα του Git. Μπορεί να θεωρηθεί ως προεπισκόπηση της επόμενης δέσμευσης, επιπλέον, μια ενδιάμεση περιοχή όπου οι δεσμεύσεις μπορούν να μορφοποιηθούν και να αναθεωρηθούν πριν από την ολοκλήρωση. Όταν γίνεται μια δέσμευση, το Git λαμβάνει αλλαγές που βρίσκονται στην περιοχή σταδίου και τις πραγματοποιεί ως νέα δέσμευση. Επιτρέπεται η προσθήκη και η κατάργηση αλλαγών από την περιοχή σταδιοποίησης. Η περιοχή σκηνής μπορεί να θεωρηθεί ως μέρος όπου το Git αποθηκεύει τις αλλαγές.

Παρόλο που, το Git δεν διαθέτει έναν αποκλειστικό κατάλογο σταδιοποίησης όπου μπορεί να αποθηκεύσει ορισμένα αντικείμενα που αντιπροσωπεύουν αλλαγές αρχείων (blobs) χρησιμοποιεί ένα αρχείο που ονομάζεται ευρετήριο.



Σχήμα 2.3: Σταδιοποίηση του Git

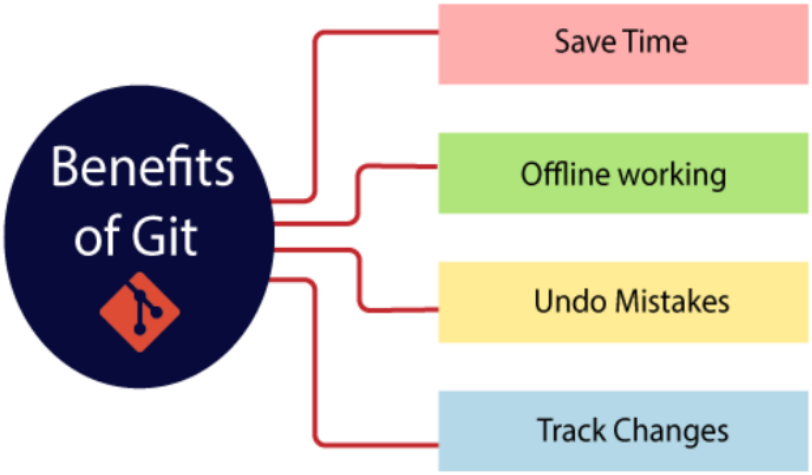
Ένα άλλο χαρακτηριστικό του Git που το κάνει να ξεχωρίζει από άλλα εργαλεία SCM είναι ότι είναι δυνατό να σκηνοθετήσουμε γρήγορα ορισμένα από τα αρχεία και να δεσμευτούν χωρίς να δεσμευτούν άλλα τροποποιημένα αρχεία στον κατάλογο εργασίας.

- Διατήρηση της καθαρής ιστορίας

Το Git διευκολύνει με το Git Rebase. Είναι ένα από τα πιο χρήσιμα χαρακτηριστικά του Git. Παίρνει τις πιο πρόσφατες δεσμεύσεις από τον κύριο κλάδο και βάζει τον κωδικό πάνω από αυτό. Έτσι, διατηρεί ένα καθαρό ιστορικό του έργου.

Οφέλη του Git

Μια εφαρμογή ελέγχου έκδοσης επιτρέπει να παρακολουθηθούν όλες οι αλλαγές που γίνονται στα αρχεία του έργου. Κάθε φορά που γίνονται αλλαγές σε αρχεία ενός υπάρχοντος έργου, γίνεται να προωθηθούν αυτές τις αλλαγές σε ένα αποθετήριο. Άλλοι προγραμματιστές επιτρέπεται να ανασύρουν τις αλλαγές από το αποθετήριο και να συνεχίσουν να εργάζονται με τις ενημερώσεις που προσθέτουν στα αρχεία του έργου.



Σχήμα 2.4: Οφέλη του Git

- Κερδίζει χρόνο

Το Git είναι αστραπιαία τεχνολογία. Κάθε εντολή χρειάζεται μόνο λίγα δευτερόλεπτα για να εκτελεστεί, ώστε να εξοικονομείται πολύς χρόνος σε σύγκριση με τη σύνδεση σε έναν λογαριασμό GitHub και να μάθουμε τις δυνατότητές του.

- Λειτουργία εκτός σύνδεσης

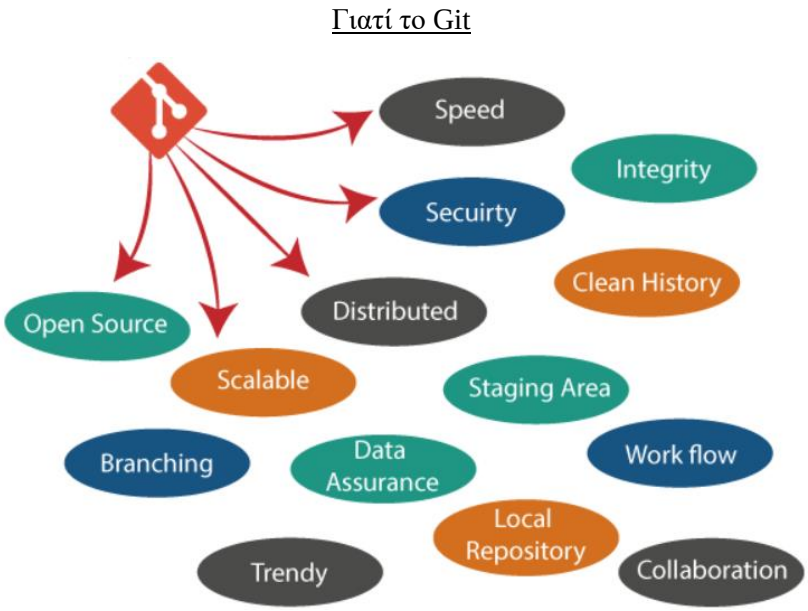
Ένα από τα πιο σημαντικά πλεονεκτήματα του Git είναι ότι υποστηρίζει την εργασία εκτός σύνδεσης. Εάν αντιμετωπιστούν προβλήματα συνδεσιμότητας στο διαδίκτυο, δεν θα επηρεάσει την βασική δουλειά. Στο Git, γίνονται σχεδόν τα πάντα τοπικά. Συγκριτικά, άλλα CVS όπως το SVN είναι περιορισμένα και προτιμούν τη σύνδεση με το κεντρικό αποθετήριο.

- Αναίρεση σφαλμάτων

Ένα επιπλέον πλεονέκτημα του Git είναι ότι γίνεται να αναιρεθούν τα λάθη. Μερικές φορές η αναίρεση μπορεί να είναι μια σωτήρια επιλογή. Το Git παρέχει την επιλογή αναίρεσης σχεδόν για τα πάντα.

- Παρακολούθηση των Αλλαγών

Το Git διευκολύνει με ορισμένες συναρπαστικές λειτουργίες όπως το Diff, το Log και το Status, το οποίο επιτρέπει να παρακολουθούνται οι αλλαγές ώστε να να ελεγχθεί η κατάσταση, να γίνει σύγκριση των αρχείων ή των υποκαταστημάτων μας.



Σχήμα 2.5: Λόγοι Επιλογής του Git

Έχουν συζητηθεί πολλά χαρακτηριστικά και πλεονεκτήματα του Git που καταδεικνύουν αναμφίβολα το Git ως το κορυφαίο σύστημα ελέγχου εκδόσεων. Τώρα, όσον αφορά μερικά άλλα σημεία σχετικά με το γιατί να γίνει επιλογή του Git.

- Ακεραιότητα του Git

Το Git έχει αναπτυχθεί για να διασφαλίζει την ασφάλεια και την ακεραιότητα του περιεχομένου που ελέγχεται από την έκδοση. Χρησιμοποιεί άθροισμα ελέγχου κατά τη μεταφορά ή την παραβίαση του συστήματος αρχείων για να επιβεβαιώσει ότι οι πληροφορίες δεν έχουν χαθεί. Εσωτερικά δημιουργεί μια τιμή αθροίσματος ελέγχου από τα περιεχόμενα του αρχείου και στη συνέχεια την επαληθεύει κατά τη μετάδοση ή την αποθήκευση δεδομένων.

- Σύστημα ελέγχου μοντέρνων εκδόσεων

Το Git είναι το πιο ευρέως χρησιμοποιούμενο σύστημα ελέγχου εκδόσεων. Διαθέτει μέγιστα έργα μεταξύ όλων των συστημάτων ελέγχου εκδόσεων. Λόγω της εκπληκτικής ροής εργασίας και των δυνατοτήτων του, είναι μια προτιμώμενη επιλογή προγραμματιστών.

- Όλα είναι Τοπικά

Σχεδόν όλες οι λειτουργίες του Git μπορούν να εκτελεστούν τοπικά. Αυτός είναι ένας σημαντικός λόγος για τη χρήση του Git. Δεν θα χρειαστεί να διασφαλιστεί σύνδεση στο διαδίκτυο.

- Συνεργασία σε Δημόσια Έργα

Υπάρχουν πολλά δημόσια έργα διαθέσιμα στο GitHub. Υπάρχει δυνατότητα συνεργασίας σε αυτά τα έργα και επίδειξης της δημιουργικότητας στον κόσμο. Πολλοί προγραμματιστές συνεργάζονται σε δημόσια έργα. Η συνεργασία επιτρέπει να σταθεί ο χρήστης δίπλα σε έμπειρους προγραμματιστές και να μάθει πολλά από αυτούς. Έτσι, ανεβάζει τις προγραμματιστικές του ικανότητες στο επόμενο επίπεδο.

- Καλή

Μπορεί να εντυπωσιάσει τους υπεύθυνους προσλήψεων αναφέροντας το Git και το GitHub στο βιογραφικό του. Στέλνει τον σύνδεσμο του προφίλ του στο GitHub στο HR του οργανισμού στον οποίο θέλει να εγγραφεί. Δείχνει τις δεξιότητές του και τις επηρεάζει μέσω της εργασίας του. Αυξάνει τις πιθανότητες πρόσληψης.

- Προαπαιτούμενα

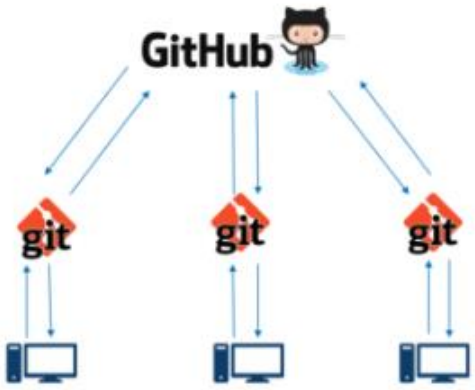
Το Git δεν είναι γλώσσα προγραμματισμού, επομένως θα πρέπει να ο χρήστης να έχει τη βασική κατανόηση μόνο των εντολών των Windows.

Κεφάλαιο 3ο: Ο Κόσμος του Github

Github

Το Github είναι μια υπηρεσία κοινής χρήσης αρχείων ή κώδικα για συνεργασία με διαφορετικούς ανθρώπους.

Συγκεκριμένα, είναι ένα λογισμικό υψηλής χρήσης που χρησιμοποιείται συνήθως για έλεγχο έκδοσης. Είναι χρήσιμο όταν περισσότερα από ένα άτομα εργάζονται σε ένα έργο. Ας πούμε, για παράδειγμα, μια ομάδα προγραμματιστών λογισμικού που θέλει να δημιουργήσει έναν ιστότοπο και όλοι πρέπει να ενημερώνουν τους κωδικούς τους ταυτόχρονα ενώ εργάζονται στο έργο. Σε αυτήν την περίπτωση, το Github τους βοηθά να δημιουργήσουν ένα κεντρικό αποθετήριο όπου όλοι μπορούν να ανεβάσουν, να επεξεργαστούν και να διαχειριστούν τα αρχεία κώδικα. [6]



Σχήμα 3.1: Μικρή Αρχιτεκτονική του Github

Φήμη του Github

Το GitHub έχει διάφορα πλεονεκτήματα, αλλά πολλοί άνθρωποι συχνά έχουν αμφιβολίες για το γιατί να μην χρησιμοποιούν το dropbox ή οποιοδήποτε σύστημα που βασίζεται στο cloud. Ας υποθέσουμε ότι περισσότεροι από δύο προγραμματιστές λογισμικού εργάζονται στο ίδιο αρχείο και θέλουν να το ενημερώσουν ταυτόχρονα. Δυστυχώς, το άτομο που θα αποθηκεύσει πρώτο το αρχείο θα έχει προτεραιότητα έναντι των άλλων. Ενώ στο Github, αυτό δεν συμβαίνει. Το Github τεκμηριώνει τις αλλαγές και τις αντικατοπτρίζει με οργανωμένο τρόπο για να αποφευχθεί το χάος μεταξύ οποιουδήποτε από τα αρχεία που έχουν ανέβει.

Επομένως, χρησιμοποιώντας το κεντρικό αποθετήριο GitHub, αποφεύγεται όλη η σύγχυση και η εργασία στον ίδιο κώδικα γίνεται πολύ εύκολη.

Αναφορικά με τη διαφορά του Github από το Git, το GitHub είναι ένα κεντρικό αποθετήριο, ενώ το Git είναι ένα εργαλείο που επιτρέπει να δημιουργηθεί ένα τοπικό αποθετήριο. Τώρα οι χρήστες συνήθως μπερδεύονται μεταξύ του git και του GitHub, αλλά στην πραγματικότητα είναι πολύ διαφορετικό. Το Git είναι ένα εργαλείο ελέγχου έκδοσης που επιτρέπει στον χρήστη να εκτελέσει όλα τα είδη λειτουργιών για τη λήψη δεδομένων από τον κεντρικό διακομιστή ή την προώθηση δεδομένων σε

αυτόν, ενώ το GitHub είναι μια βασική πλατφόρμα φιλοξενίας για συνεργασία ελέγχου έκδοσης. Το GitHub είναι μια εταιρεία που επιτρέπει στον χρήστη να φιλοξενεί ένα κεντρικό αποθετήριο σε έναν απομακρυσμένο διακομιστή.

Το GitHub παρέχει μια όμορφη οπτική διεπαφή που βοηθάει να γίνει παρακολούθηση ή διαχείριση των τοπκών έργων που ελέγχονται από την έκδοση.

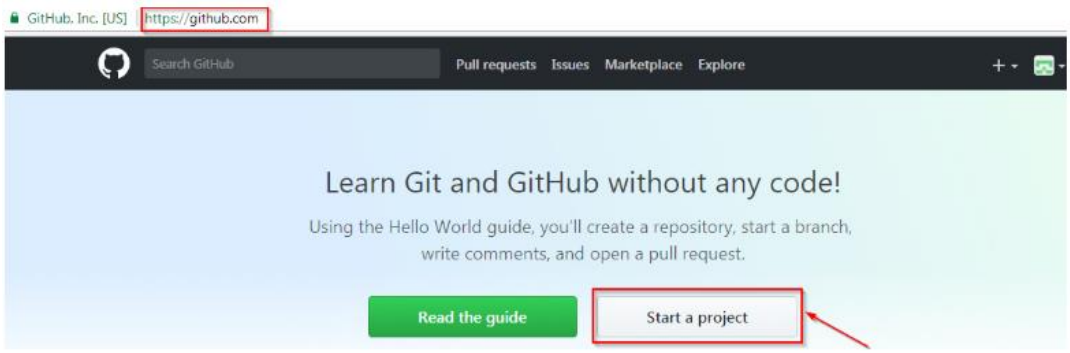
Μόλις ένας χρήστης εγγραφεί στο GitHub, μπορεί να συνδεθεί με το κοινωνικό δίκτυο και να δημιουργήσει ένα ισχυρό προφίλ.

Δημιουργία Αποθετηρίου

Το αποθετήριο είναι ένας χώρος αποθήκευσης όπου ζει το έργο. Μπορεί να είναι τοπικό σε έναν φάκελο στον υπολογιστή ή μπορεί να είναι ένας χώρος αποθήκευσης στο GitHub ή σε άλλο διαδικτυακό κεντρικό υπολογιστή. Μπορεί ένας χρήστης να διατηρήσει αρχεία κώδικα, αρχεία κειμένου, εικόνες ή οποιοδήποτε είδος αρχείου σε ένα αποθετήριο. Χρειάζεται ένα αποθετήριο GitHub όταν έχει κάνει κάποιες αλλαγές και είναι έτοιμος να κάνει μεταφόρτωση. Αυτό το αποθετήριο GitHub λειτουργεί ως το απομακρυσμένο αποθετήριο.

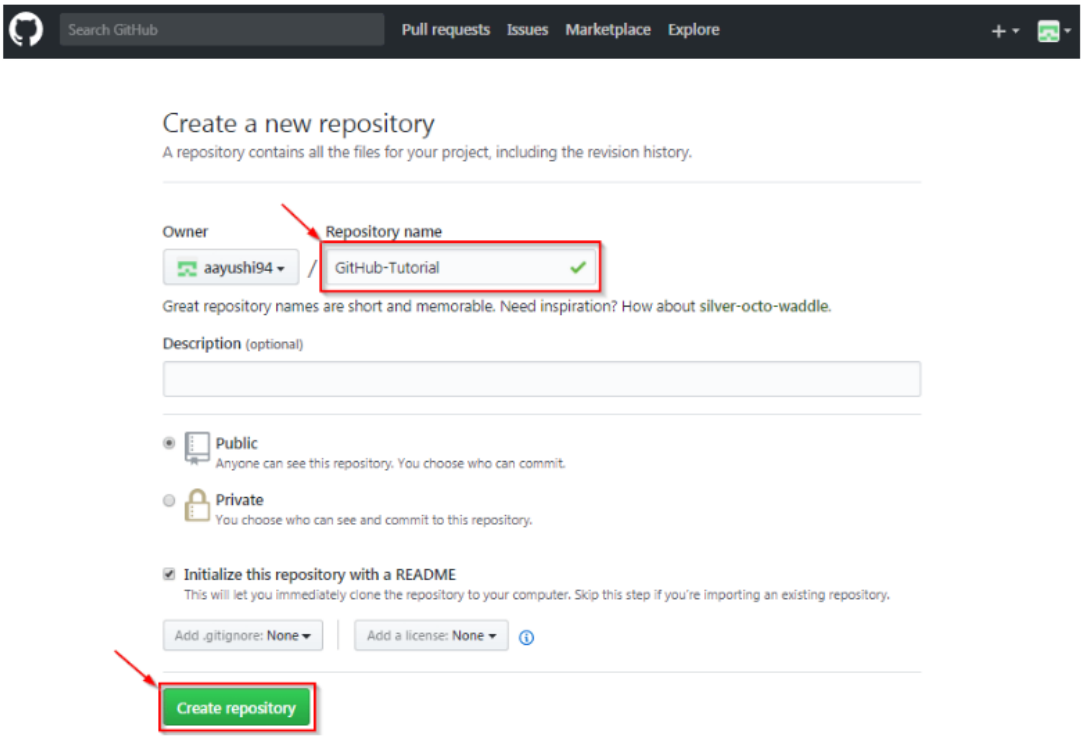
Ακολουθούν μερικά απλά βήματα για τη νδημιουργία αποθετηρίου GitHub:

- Μετάβαση στον σύνδεσμο: https://github.com/ . Συμπλήρωση της φόρμας εγγραφής, κλικ στο «Εγγραφή στο Github» και Κλικ στο «Έναρξη νέου έργου».



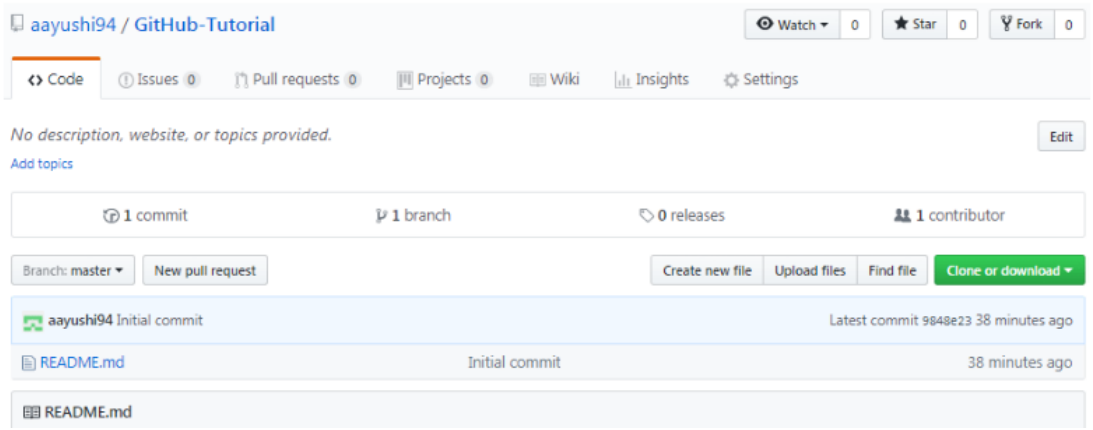
Σχήμα 3.2: Περιβάλλον Github

- Εισαγωγή οποιουδήποτε ονόματος αποθετηρίου και κλικ στο «Δημιουργία αποθετηρίου». Μπορεί να δοθεί επίσης και μια περιγραφή στο αποθετήριο, αν και είναι προαιρετικό.



Σχήμα 3.3: Δημιουργία Αποθετηρίου Github

Τώρα, παρατηρείται από προεπιλογή ότι ένα αποθετήριο GitHub είναι δημόσιο, πράγμα που σημαίνει ότι ο καθένας μπορεί να δει τα περιεχόμενα αυτού του αποθετηρίου, ενώ σε ένα ιδιωτικό αποθετήριο, μπορεί ο χρήστης να επιλέξει ποιος μπορεί να δει το περιεχόμενο. Επίσης, το ιδιωτικό αποθετήριο είναι μια πληρωμένη έκδοση. Επίσης, εάν ανατρέξει στο παραπάνω στιγμιότυπο οθόνης, αρχικοποιεί το αποθετήριο με ένα αρχείο README. Αυτό το αρχείο περιέχει την περιγραφή του αρχείου και μόλις επιλεχθεί αυτό το πλαίσιο, αυτό θα είναι το πρώτο αρχείο μέσα στο αποθετήριο.



Σχήμα 3.4: Επιτυχής Δημιουργία Github

Με αυτό τον τρόπο θα δημιουργηθεί το αποθετήριο με επιτυχία! Μόλις γίνει αυτό, ο χρήστης είναι έτοιμος να δεσμεύσει, να τραβήξει, να σπρώξει και να εκτελέσει όλες τις άλλες λειτουργίες.

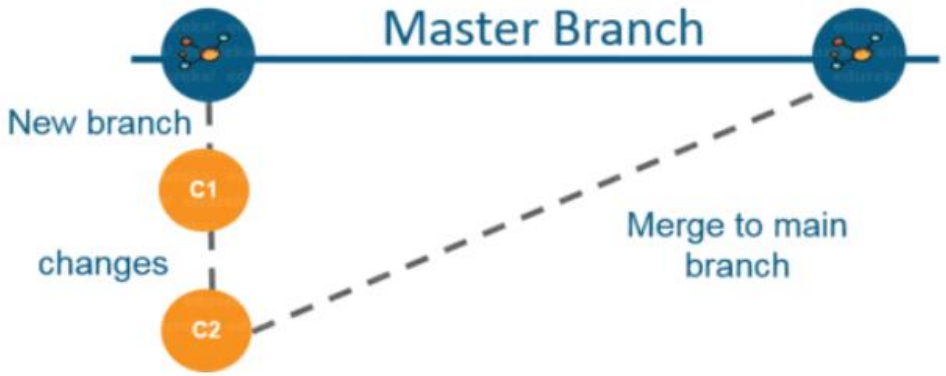
Δημιουργία Υποκαταστημάτων και Εκτέλεση Λειτουργιών

Τα υποκαταστήματα βοηθούν στο να εργάζεται ο χρήστης σε διαφορετικές εκδόσεις ενός αποθετηρίου ταυτόχρονα.

Ας υποθέσουμε ότι ο χρήστης θέλει να προσθέσει μια νέα δυνατότητα (η οποία βρίσκεται στη φάση ανάπτυξης) και ταυτόχρονα φοβάται αν θα κάνετε αλλαγές στο κύριο έργο ή όχι. Εδώ έρχεται η διακλάδωση git.

Οι κλάδοι επιτρέπουν στον χρήστη να μετακινηθεί εμπρός και πίσω μεταξύ των διαφορετικών καταστάσεων/εκδόσεων ενός έργου. Στο παραπάνω σενάριο, ο χρήστης μπορεί να δημιουργήσει έναν νέο κλάδο και να δοκιμάσει τη νέα δυνατότητα χωρίς να επηρεάσει τον κύριο κλάδο. Μόλις τελειώσει με αυτό, μπορεί να συγχωνεύσει τις αλλαγές από τον νέο κλάδο στον κύριο κλάδο. Εδώ ο κύριος κλάδος είναι ο κύριος κλάδος, ο οποίος βρίσκεται εκεί στο αποθετήριο από προεπιλογή.

Υπάρχει ένας κλάδος κύριας/παραγωγής που διαθέτει νέο κλάδο για δοκιμή. Κάτω από αυτόν τον κλάδο, γίνονται δύο σετ αλλαγών και μόλις ολοκληρωθεί, συγχωνεύεται ξανά στον κύριο κλάδο. Έτσι λειτουργεί η διακλάδωση!



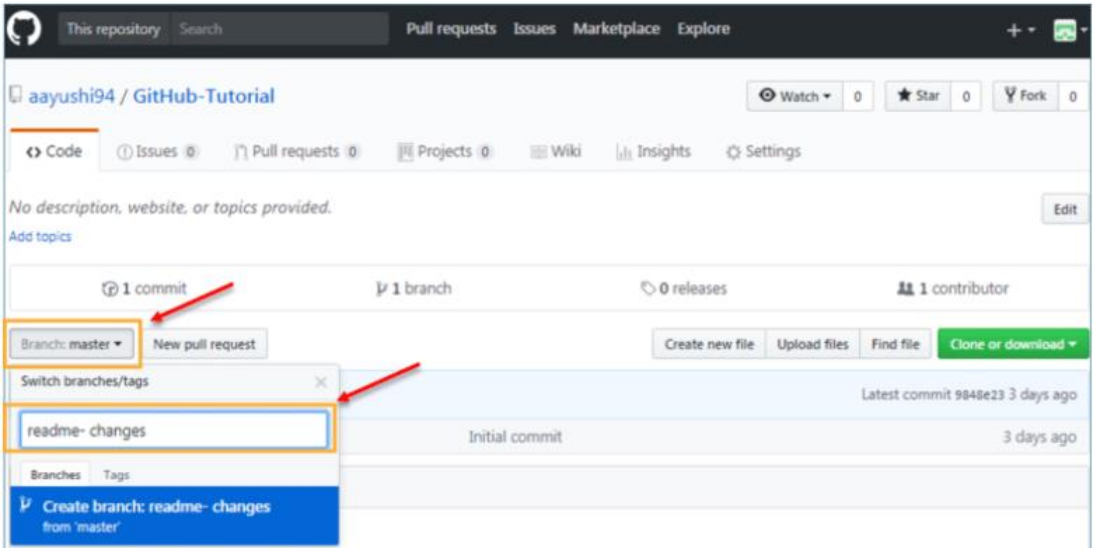
Σχήμα 3.5: Υποκαταστήματα Github

Για να δημιουργηθεί ένα υποκατάστημα στο GitHub, ακολουθούνται τα παρακάτω βήματα:

- Κλικ στο αναπτυσσόμενο μενού "Branch: master"

Μετά το κλικ στο υποκατάστημα, μπορεί κάποιος να βρει ένα υπάρχον υποκατάστημα ή μπορεί να δημιουργήσει ένα νέο. Στην προκειμένη, δημιουργείται ένα νέο υποκατάστημα με όνομα "readme-changes".

- Μόλις δημιουργηθεί ένας νέος κλάδος, υπάρχουν τώρα δύο κλάδοι στο αποθετήριο σας, δηλαδή read-me (κύριο κλάδο) και readme- changes. Ο νέος κλάδος είναι απλώς το αντίγραφο του κύριου κλάδου.



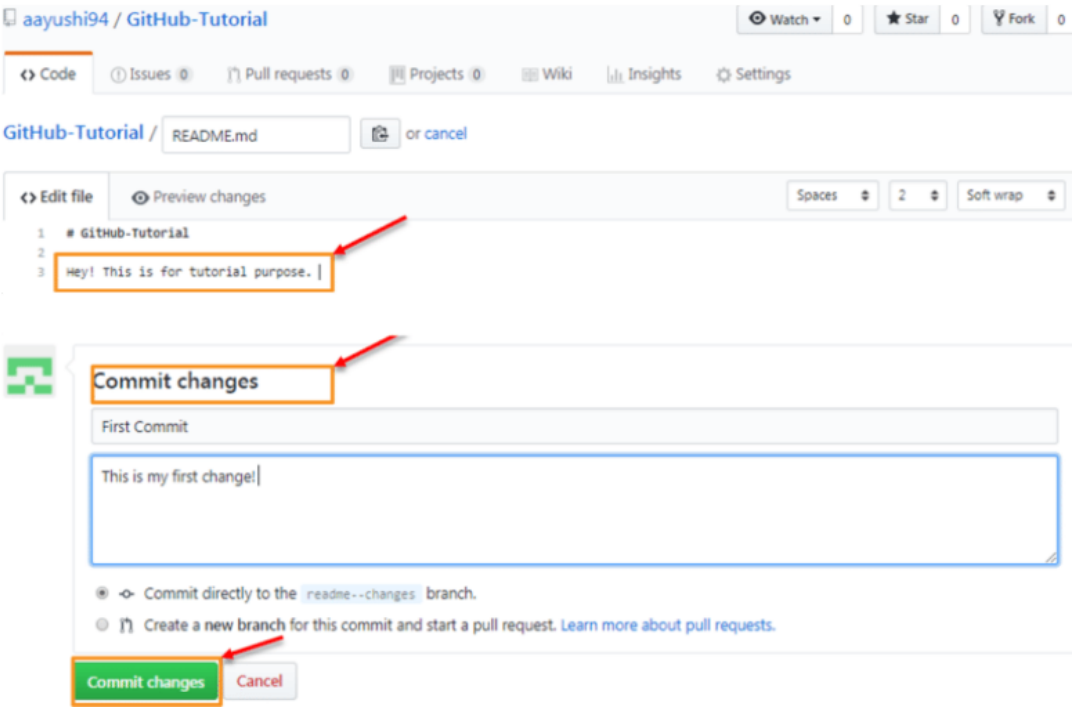
Σχήμα 3.6: Επεξεργασία Υποκαταστημάτων Github

Χρήση και Λειτουργίες του Github

- Εντολή δέσμευσης

Αυτή η λειτουργία βοηθά τον χρήστη να αποθηκεύσει τις αλλαγές στο αρχείο του. Όταν γίνεται δέσμευση ενός αρχείου, θα πρέπει πάντα να παρέχεται το μήνυμα, απλώς για να έχει ο χρήστης υπόψην του τις αλλαγές που πραγματποιήθηκαν. Αν και αυτό το μήνυμα δεν είναι υποχρεωτικό, συνιστάται πάντα για να μπορεί να διαφοροποιήσει τις διάφορες εκδόσεις ή δεσμεύσεις που έχει κάνει μέχρι τώρα στο αποθετήριο του. Αυτά τα μηνύματα δέσμευσης διατηρούν το ιστορικό των αλλαγών που με τη σειρά τους βοηθούν άλλους συνεισφέροντες να κατανοήσουν καλύτερα το αρχείο. Τώρα για την πρώτη δέσμευση, ακολουθούν τα παρακάτω βήματα:

- Κλικ στο αρχείο "readme- changes" που μόλις δημιουργήσαμε.
- Κλικ στο εικονίδιο "επεξεργασία" ή ένα εικονίδιο με το μολύβι στην πιο δεξιά γωνία του αρχείου.
- Θα ανοίξει ένα πρόγραμμα επεξεργασίας όπου ο χρήστης μπορεί να πληκτρολογήσει τις αλλαγές ή οτιδήποτε άλλο.
- Ο χρήστης γράφει ένα μήνυμα δέσμευσης που προσδιορίζει τις αλλαγές του.
- Στο τέλος κάνει κλικ στην επιλογή δέσμευση αλλαγών.



Σχήμα 3.7: Αλλαγές Δεσμεύσεων Github

Πραγματοποιήθηκε η πρώτη δέσμευση του χρήστη. Τώρα αυτό το αρχείο "readme- changes" είναι διαφορετικό από τον κύριο κλάδο. Στη συνέχεια, ακολουθεί η διαδικασία για το αίτημα Pull:

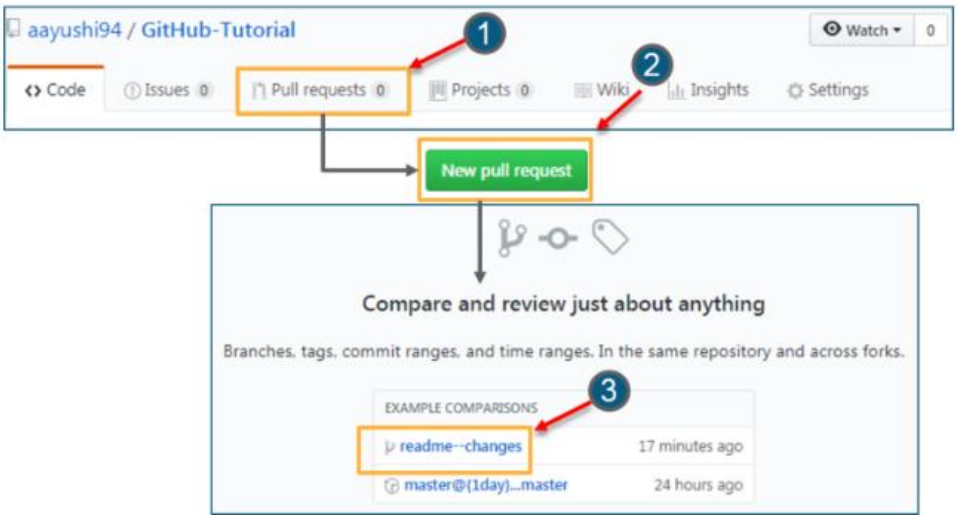
- Εντολή έλξης (Pull)

Η εντολή Pull είναι η πιο σημαντική εντολή στο GitHub. Δίνει τις αλλαγές που έγιναν στο αρχείο και ζητά από άλλους συνεισφέροντες να το δουν καθώς και να το συγχωνεύσουν με τον κύριο κλάδο. Μόλις ολοκληρωθεί η δέσμευση, οποιοσδήποτε μπορεί να τραβήξει το αρχείο και να ξεκινήσει μια συζήτηση για αυτό. Μόλις ολοκληρωθούν όλα, ο χρήστης μπορεί να συγχωνεύσει το αρχείο.

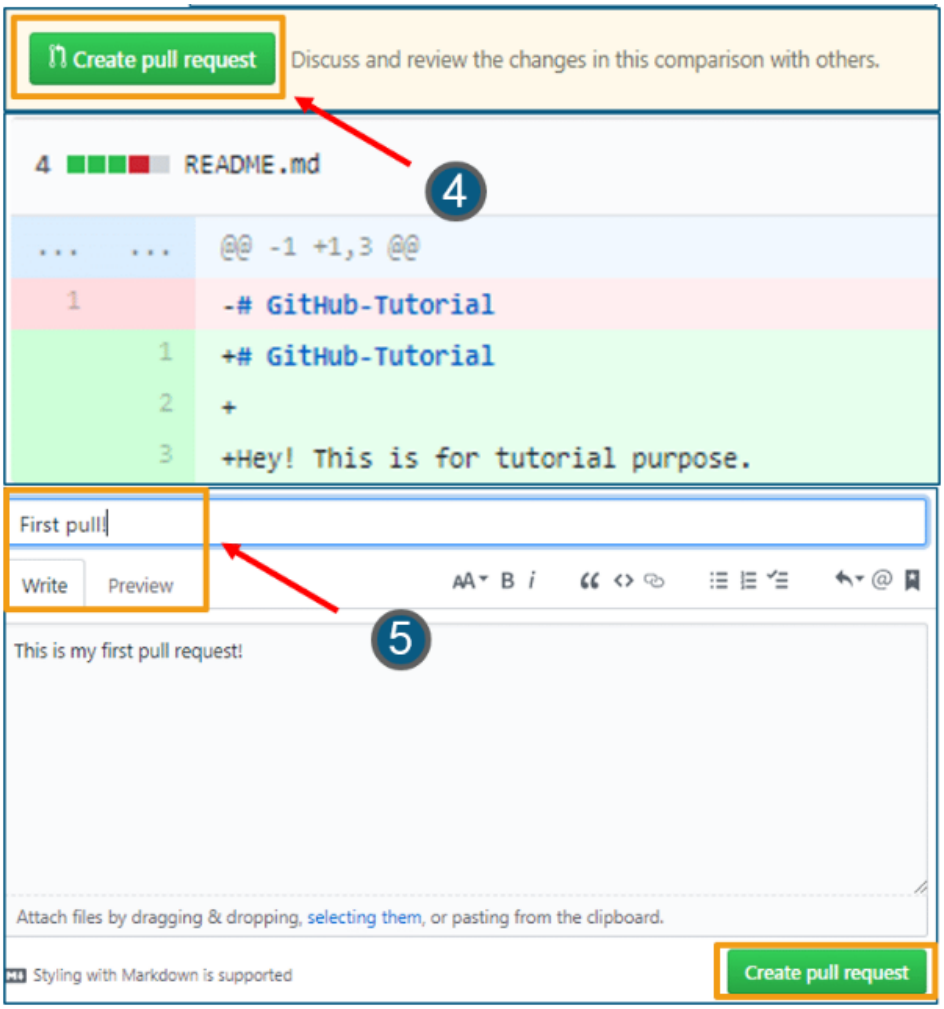
Η εντολή Pull συγκρίνει τις αλλαγές που γίνονται στο αρχείο και εάν υπάρχουν διενέξεις, μπορεί να επιλυθεί με μη αυτόματο τρόπο.

Ακολουθούν τα διάφορα βήματα που εμπλέκονται για την υποβολή αιτήματος στο GitHub.

- Κλικ στην καρτέλα "Αιτήματα έλξης".
- Κλικ στο «Νέο αίτημα έλξης».
- Ο χρήστης επιλέγει τον κλάδο και κάνει κλικ στο αρχείο «readme- changes» για να δει τις αλλαγές μεταξύ των δύο αρχείων που υπάρχουν στο αποθετήριο του.
- Κλικ στο «Δημιουργία αιτήματος έλξης».
- Ο χρήστης εισάγει οποιονδήποτε τίτλο, περιγραφή των αλλαγών του και κάνει κλικ στο «Δημιουργία αιτήματος έλξης».



Σχήμα 3.8: Δημιουργία Αιτήματος Έλξης Github Α'



Σχήμα 3.9: Δημιουργία Αιτήματος Έλξης Github Β'

Στη συνέχεια, ακολουθεί η διαδικασία για τη συγχώνευση του αιτήματος Pull:

- Εντολή συγχώνευσης (Merge)

Εδώ έρχεται η τελευταία εντολή που συγχωνεύει τις αλλαγές στον κύριο κύριο κλάδο. Η συγχώνευση γίνεται στο αρχείο "readme- changes" με τον κύριο κλάδο/ read-me.

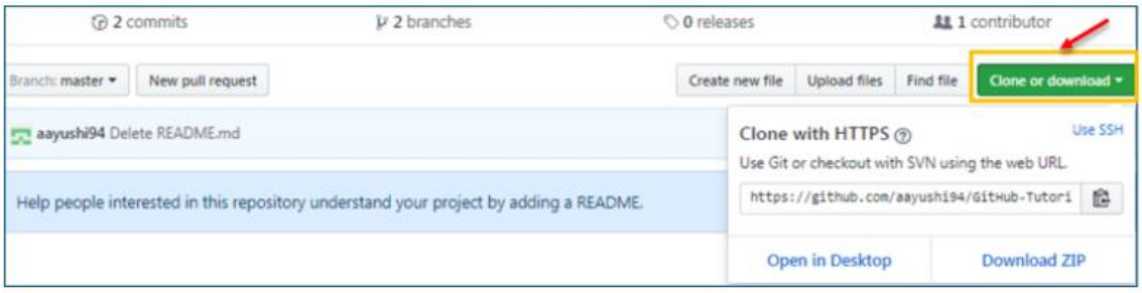
- Κλικ στο «Αίτημα έλξης συγχώνευσης» για την συγχώνευση των αλαλγών στον κύριο κλάδο.
- Κλικ στο «Επιβεβαίωση συγχώνευσης».
- Ο χρήστης μπορεί να διαγράψει τον κλάδο αφού ενσωματωθούν όλες οι αλλαγές και εάν δεν υπάρχουν διενέξεις.

Σχήμα 3.10: Συγχώνευση Αιτήματος Έλξης Github

Όσο ένας χρήστης ακολουθεί τα παραπάνω βήματα, εξοικειώνεται περισσότερο με το περιβάλλον του Github. Στη συνέχεια, ακολουθεί η κλωνοποίηση και η διοχέτευση αποθετηρίου GitHub.

Κλωνοποίηση και Διοχέτευση Αποθετηρίου

Ορισμένες φορές ίσως χρειαστεί να κλωνοποιηθεί ένα αποθετήριο. Εάν κάποιος χρήστης θέλει να χρησιμοποιήσει κάποιον κώδικα που υπάρχει σε ένα δημόσιο αποθετήριο, μπορεί να αντιγράψει απευθείας τα περιεχόμενα με κλωνοποίηση ή λήψη.



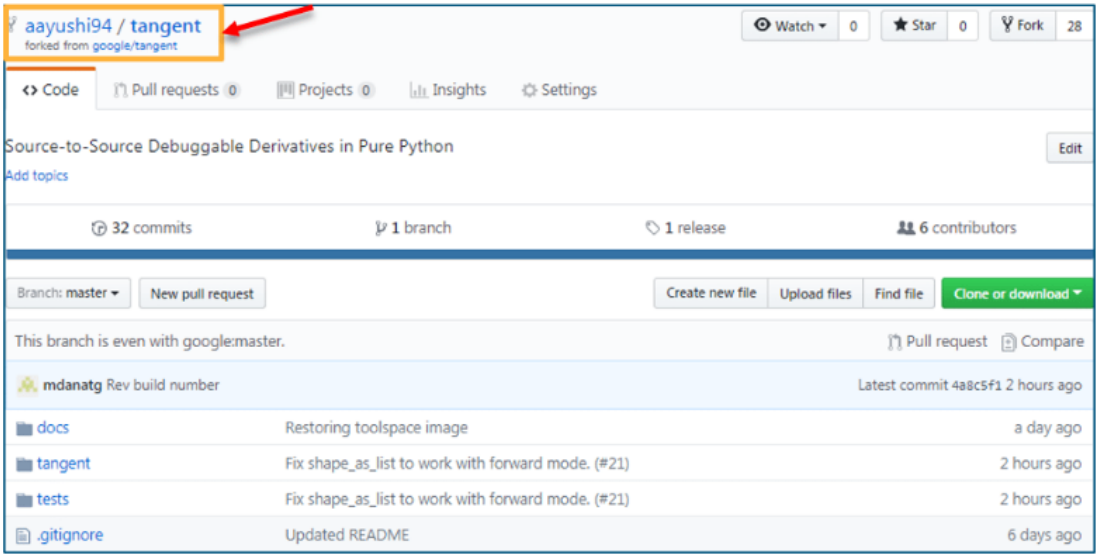
Σχήμα 3.11: Κλωνοποίηση και Λήψη Github

Στη συνέχεια, ακολουθεί η δημιουργία αποθετηρίου με τη χρήση του fork στα παρακάτω βήματα:

- Μετάβαση στην Εξερεύνηση και αναζήτηση για δημόσια αποθετήρια.
- Κλικ στο "fork". Σημείωση ότι αυτό το αποθετήριο "εφαπτομένης" έχει ήδη διαχωριστεί 27 φορές και βρίσκεται στον λογαριασμό "google".
- Κλικ στο "Fork", θα χρειαστεί λίγος χρόνος για να διαχωριστεί το αποθετήριο. Παρατηρείται ότι το όνομα του αποθετηρίου βρίσκεται κάτω από τον λογαριασμό του χρήστη.

Σχήμα 3.12: Fork του Github

Με αυτό τον τρόπο διαχωρίστηκε επιτυχώς ένα υπάρχον αποθετήριο στον λογαριασμό του χρήστη.

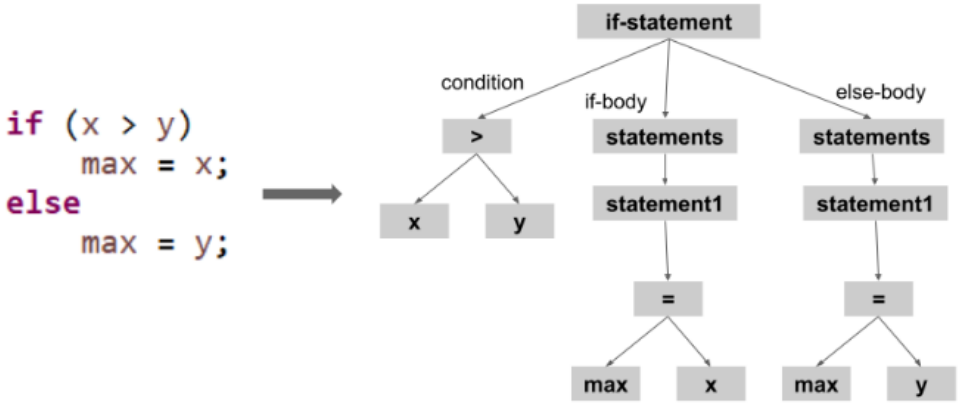


Σχήμα 3.13: Επιτυχής Δημιουργία Αποθετηρίου με Fork Github

Κεφάλαιο 4ο: Ο Αναλυτής JavaParser

JavaParser

Οι βιβλιοθήκες JavaParser είναι ένας εύκολος τρόπος εξαγωγής και ανάλυσης Αφηρημένων Συντακτικών Δέντρων (AST) στην Java. Μπορεί να αναλύσει και να μεταμορφώσει τον κώδικα. [7]



Σχήμα 4.1: Αναπαράσταση Λειτουργίας JavaParser

Αναλύοντας ένα project

Το JavaParser, ως βιβλιοθήκη προσφέρει πολλές λειτουργίες για την ανάλυση ενός προγράμματος Java. Ωστόσο, εάν υπάρχουν περισσότερα από ένα αρχεία προς ανάλυση που διανέμονται σε πολλούς φακέλους ή μια εφαρμογή που είναι κατανεμημένη σε πολλά πακέτα, πρέπει να καθοριστούν οι φάκελοι προορισμού. Ένας από τους τρόπους για την ανάλυση του έργου είναι η χρήση του SourceRoot. Διαχειρίζει όλους τους φακέλους που περιέχουν αρχεία Java ως ρίζες. [8]

Πίνακας 4.1: Διαχωρισμός με SourceRoot

```
Path projectRoot =
FileSystems.getDefault().getPath("path\\to\\main\\root\\");
String[] roots = new String[] {"subdirectory1\\after\\root",
"subdirectory2\\after\\root"};
for (String root : roots) {
SourceRoot sourceRoot = new SourceRoot(projectRoot.resolve(root));
sourceRoot.setParserConfiguration(parserConfiguration);
List<CompilationUnit> allCU=
parseResults.stream().filter(ParseResult::isSuccessful).map(r ->
r.getResult().get()).collect(Collectors.toList());
Iterator<CompilationUnit> cuIter = allCU.iterator();
while (cuIter.hasNext()) {
CompilationUnit compilationunit = cuIter.next();
//work with compilationunit
}
```

Ένα έργο μπορεί να έχει εξωτερικές εξαρτήσεις (dependencies) που θα πρέπει επίσης να υπάρχουν στην καθορισμένη διαδρομή. Αυτές οι εξαρτήσεις προσαρτώνται στον TypeSolver.

Πίνακας 4.2: Χρήση του TypeSolver

```
CombinedTypeSolver typeSolver = new CombinedTypeSolver(new
ReflectionTypeSolver(), new CombinedTypeSolver());
typeSolver.add(new JavaParserTypeSolver(new
File("directory\\containing\\javafile\\dependency")));
typeSolver.add(new JarTypeSolver("path\\to\\jar\\dependency.jar"));
JavaSymbolSolver symbolSolver = new JavaSymbolSolver(typeSolver);
ParserConfiguration parserConfiguration = new
ParserConfiguration().setSymbolResolver(symbolSolver);
StaticJavaParser.setConfiguration(parserConfiguration);JavaParser parser
= new JavaParser(parserConfiguration)
```

Έτσι, σε περίπτωση εμφάνισης του σφάλματος "Exception in thread "main" UnsolvedSymbolException", πιθανότατα δεν έχει προστεθεί κάποια εξάρτηση στον TypeSolver. Το ίδιο σφάλμα εμφανίζεται επίσης εάν οριστεί μια νέα μέθοδος στον κώδικα αλλά προηγουμένως δεν έχει αναλυθεί.

Προσεκτική Τροποποίηση του AST

Όπως και άλλα αντικείμενα στη Java, εάν ένα AST (Abstract Syntax Tree) τροποποιηθεί χρησιμοποιώντας JavaParser ενώ πραγματοποιείται ήδη μια επανάληψη πάνω σε αυτό, ενδέχεται να προκύψει σφάλμα εξαίρεσης (exception) "Ταυτόχρονη εξαίρεση τροποποίησης". Επομένως, καλύτερα να διαχωριστεί το κομμάτι της ανάλυσης από το κομμάτι της τροποποίησης.

Προσοχή στις JAR εκδόσεις

Ένας άλλος λόγος για σφάλματα θα μπορούσε να είναι οι αναντιστοιχίες εκδόσεων σε διαφορετικές βιβλιοθήκες. Συνιστάται η χρήση JavaParser JAR με εκδόσεις μεγαλύτερες από 3.20. Σε περίπτωση που προκύψει σφάλμα καταγραφής, θα πρέπει να γίνει έλεγχος για το αν χρησιμοποιείτε το slf4j-ext (έκδοση 1.7.28). Για την ευκολότερη συντήρηση των JAR, θα πρέπει να διατηρούνται στον ίδιο φάκελο. Αυτό θα βοηθήσει όχι μόνο στο να προστεθούν επαναληπτικά όλες οι εξαρτήσεις αντί να κωδικοποιηθούν, αλλά και να προσδιοριστεί εάν υπάρχουν συγκρούσεις έκδοσης.

Μόλις ρυθμιστεί το JavaParser, μπορούν να αξιοποιηθούν όλες οι λειτουργίες του. Με λίγα λόγια, το JavaParser προσφέρει ανάλυση, μετατροπή και δημιουργία πηγαίου κώδικα Java.

Ανάλυση

Το JavaParser επιτρέπει τη διέλευση αρχείων «.java». Δημιουργείται μια CompilationUnit για κάθε τέτοιο αρχείο που βρίσκεται επίσης στην κορυφή της ιεραρχίας ανάλυσης. Στη συνέχεια, μια CompilationUnit μπορεί να αναλυθεί περαιτέρω για λεπτομέρειες χαμηλότερου επιπέδου, όπως το MethodDeclaration και το FieldDeclaration.

Μεταμόρφωση

Το JavaParser υποστηρίζει επίσης και την επεξεργασία του AST. Οι προγραμματιστές συχνά περιφρονούν τους ανθρώπους που κάνουν επαναλαμβανόμενες εργασίες με το χέρι. Θα πρέπει να προβούν σε αυτοματοποιήσεις.

Ωστόσο, όλες οι δραστηριότητες που σχετίζονται με την κωδικοποίηση γίνονται στο χέρι. Σίγουρα, υπάρχουν IDE που μπορούν να εκτελέσουν κάποια μικρή ανακατασκευή για τον σκοπό του project, αλλά εκεί έχει τέλος.

Η αλλαγή στην συγγραφή του κώδικα θα μπορούσε να γίνει ως εξής:

- Δημιουργία του βαρετού και επαναλαμβανόμενου κώδικα Java που πρέπει να δημιουργηθεί
- Ανάλυση του κώδικα για την απάντηση ορισμένων ερωτήσεων σχετικά με αυτόν
- Επεξεργασία και αναδιαμόρφωση του κώδικα

Αυτό θα επιτευχθεί με ένα σύνολο βιβλιοθηκών: την JavaParser και την JavaSymbolSolver.

Έναρξη

Το μόνο που χρειάζεται, είναι να προστεθεί η JavaSymbolSolver στις εξαρτήσεις του project.

Το JavaSymbolSolver είναι μια βιβλιοθήκη που συμπληρώνει το JavaParser δίνοντάς του μερικές αρκετά ισχυρές δυνατότητες που είναι απαραίτητες για να απαντηθούν πιο περίπλοκες ερωτήσεις σχετικά με τον κώδικα.

Το JavaSymbolSolver εξαρτάται από το JavaParser, επομένως χρειάζεται απλώς να προσθέσετε το JavaSymbolSolver και το Maven.

Περισσότερες πληροφορίες σχετικά με το Maven, δίνονται στο Κεφάλαιο 7.

Δημιουργία Κώδικα

Υπάρχουν πολλές περιπτώσεις στις οποίες μπορεί να χρειαστεί η δημιουργία κώδικα σε Java. Για παράδειγμα, η δημιουργία κώδικα με βάση ορισμένα εξωτερικά δεδομένα, όπως ένα σχήμα βάσης δεδομένων ή ένα REST API.

Όποιος κι αν είναι ο λόγος για τη δημιουργία κώδικα, μπορεί να χρησιμοποιηθεί το JavaParser.

Παρακάτω δημιουργείται μια κλάση με δύο πεδία, έναν κατασκευαστή και δύο getter. Σκοπός είναι να δοθεί μια αίσθηση του τι σημαίνει η χρήση του JavaParser για τη δημιουργία κώδικα.

Πίνακας 4.3: Παράδειγμα με JavaParser

```
CompilationUnit cu = new CompilationUnit();

cu.setPackageDeclaration("jpexample.model");

ClassOrInterfaceDeclaration book = cu.addClass("Book");
book.addField("String", "title");
book.addField("Person", "author");

book.addConstructor(Modifier.PUBLIC)
        .addParameter("String", "title")
        .addParameter("Person", "author")
        .setBody(new BlockStmt()
                .addStatement(new ExpressionStmt(new AssignExpr(
                        new FieldAccessExpr(new ThisExpr(), "title"),
                        new NameExpr("title"),
                        AssignExpr.Operator.ASSIGN)))
                .addStatement(new ExpressionStmt(new AssignExpr(
                        new FieldAccessExpr(new ThisExpr(), "author"),
                        new NameExpr("author"),
                        AssignExpr.Operator.ASSIGN))));

book.addMethod("getTitle", Modifier.PUBLIC).setBody(
        new BlockStmt().addStatement(new ReturnStmt(new NameExpr("title"))));

book.addMethod("getAuthor", Modifier.PUBLIC).setBody(
        new BlockStmt().addStatement(new ReturnStmt(new NameExpr("author"))));

System.out.println(cu.toString());
```

Η τελευταία οδηγία εκτυπώνει τον κώδικά, φρέσκο και έτοιμο για μεταγλώττιση. Ο κώδικας μπορεί επίσης να αποθηκευτεί και σε ένα αρχείο αντί να εκτυπωθεί.

Ανάλυση Κώδικα

Υπάρχουν πολλές διαφορετικές ερωτήσεις που αφορούν τον κώδικα, όπως και πολλούς διαφορετικούς τρόπους για να γίνει η ανάλυσή του.

Μπορούν να αναλυθούν όλα τα πηγαία αρχεία του project:

Πίνακας 4.4: Ανάλυση με JavaParser

```
// Parse all source files
SourceRoot sourceRoot = new SourceRoot(myProjectSourceDir.toPath());
sourceRoot.setParserConfiguration(parserConfiguration);
List<ParseResult> parseResults = sourceRoot.tryToParse("");

// Now get all compilation unitsList
allCus = parseResults.stream()
        .filter(ParseResult::isSuccessful)
        .map(r -> r.getResult().get())
        .collect(Collectors.toList());
```

Μπορεί να δημιουργηθεί επίσης μια μέθοδος για τη λήψη όλων των κόμβων ενός συγκεκριμένου τύπου μεταξύ όλων των μονάδων μεταγλώττισης:

Πίνακας 4.5: Λήψη Κόμβων με JavaParser

```
public static  List getNodes(List cus, Class nodeClass) {
    List res = new LinkedList();
    cus.forEach(cu -> res.addAll(cu.findAll(nodeClass)));
    return res;
}
```

Μερικές ερωτήσεις θα μπορούσαν να είναι οι εξής:

Πόσες μέθοδοι απαιτούν περισσότερες από 3 παραμέτρους;

Πίνακας 4.6: Παράμετροι Μεθόδων

```
long n = getNodes(allCus, MethodDeclaration.class).stream().filter(m ->
m.getParameters().size() > 3).count();
System.out.println("N of methods with 3+ params: " + n);
```

Ποιες είναι οι τρεις κορυφαίες κατηγορίες με τις περισσότερες μεθόδους;

Πίνακας 4.7: Κατηγορίες Μεθόδων

```
getNodes(allCus, ClassOrInterfaceDeclaration.class).stream().filter(c ->
!c.isInterface()).sorted(Comparator.comparingInt(o -> -1 *
o.getMethods().size())).limit(3).forEach(c ->
System.out.println(c.getNameAsString() + ": " + c.getMethods().size() + "
methods"));
```

Μεταμόρφωση Κώδικα

Έστω πως ένας χρήστης προγραμματίζει με τη βοήθεια μιας συγκεκριμένης βιβλιοθήκης. Την έχει προσθέσει στις εξαρτήσεις του πριν από χρόνια και την χρησιμοποιεί αισίως. Μετά από καιρό την χρησιμοποιεί όλο και περισσότερο σε όλο το project του.

Μια μέρα εμφανίζεται μια νέα έκδοση αυτής της χρήσιμης βιβλιοθήκης και αποφασίζει ότι θέλει να ενημερώσει τις εξαρτήσεις του. Πλέον, στη νέα βιβλιοθήκη έχει αφαιρεθεί μία από τις μεθόδους που χρησιμοποιούσε. Έστω ότι καταργήθηκε και ονομάστηκε oldMethod.

Τώρα η oldMethod έχει αντικατασταθεί από την newMethod. Η newMethod παίρνει 3 παραμέτρους, οι δύο πρώτες είναι ίδιες με το oldMethod, απλώς αντιστρέφονται, ενώ η τρίτη είναι ένα boolean, το οποίο πρέπει να οριστεί σε true για να έχουμε την ίδια συμπεριφορά που είχαμε με το oldMethod.

Σε περίπτωση που υπάρχουν εκατοντάδες κλήσεις στο oldMethod, είναι αδύνατον να αλλάξουν μία προς μία. Γιαυτό το λογό μπορεί να χρησιμοποιηθεί το JavaParser.

Με τον παρακάτω τρόπο, μπορεούν να βρεθούν όλες οι κλήσεις προς την παλιά μέθοδο σε ένα συγκεκριμένο αρχείο, γνωστό και ως CompilationUnit στο JavaParser parlanse:

Πίνακας 4.8: Κλήσεις Παλιάς Μεθόδου

```
myCompilationUnit.findAll(ethodCallExpr.class).stream().filter(m ->
m.resolveInvokedMethod().getQualifiedSignature().equals("foo.MyClass.oldM
ethod(java.lang.String, int)")).forEach(m ->
m.replace(replaceCallsToOldMethod(m)));
```

Έπειτα οι παλιές κλήσεις μετατρέπονται σε νέες με τον εξής τρόπο:

Πίνακας 4.9: Μετατροπή Παλιών Κλήσεων σε Νέες

```
public MethodCallExpr replaceCallsToOldMethod(MethodCallExpr methodCall)
{
     MethodCallExpr newMethodCall = new
MethodCallExpr(methodCall.getScope().get(), "newMethod");
     newMethodCall.addArgument(methodCall.getArgument(1));
     newMethodCall.addArgument(methodCall.getArgument(0));
     newMethodCall.addArgument(new BooleanLiteralExpr(true));
     return newMethodCall;
}
```

Το επόμενο βήμα είναι να ληφθεί ο κώδικας για την τροποποιημένη CompilationUnit και απλώς να αποθηκευτεί σε αρχείο Java.

Περισσότερα για το JavaParser

Υπάρχουν πολλές δυνατότητες του JavaParser που δεν αναφέρθηκαν, όπως:

- Το JavaParser μπορεί να χειριστεί σχόλια, υπολογίζοντας σε ποια στοιχεία αναφέρονται.
- Το JavaParser μπορεί να κάνει λεκτική διατήρηση ή όμορφη εκτύπωση (επιλογή του χρήστη).
- Μπορεί να ανακαλύψει σε ποια δήλωση μεθόδου αναφέρεται μια κλήση μεθόδου, ποιους προγόνους έχει μια συγκεκριμένη κλάση και πολλά άλλα, χάρη στην ενσωμάτωση με το JavaSymbolSolver.
- Μπορεί να εξάγει το AST σε JSON, XML, YAML, ακόμη και να δημιουργήσει διαγράμματα χρησιμοποιώντας το Graphviz.

Σύνοψη

Είναι δύσκολο να χρησιμοποιηθεί ένα οποιοδήποτε εργαλείο για την ανάλυση, την δημιουργία και την τροποποίηση κώδικα. Μαθαίνοντας πως χρησιμοποιείται το JavaParser, η συγχώνευση αυτών των τριων λειτουργιών γίνεται επιτεύξιμη.

Τι δεν είναι το JavaParser

Αν και η βιβλιοθήκη μπορεί να χρησιμοποιηθεί με διάφορους τρόπους, δεν αποτελεί μια "εργαλειοθήκη αναδιαμόρφωσης κώδικα". Αυτό είναι ίσως η πιο δημοφιλής παρανόηση για τη συγκεκριμένη βιβλιοθήκη. Η βιβλιοθήκη παρέχει απάντηση στην ερώτηση «τι είναι αυτός ο κώδικας;», ενώ το γιατί και το πώς μπορεί να γίνει χρήση του εξαρτάται από τον προγραμματιστή.

Δεν είναι λύση συμβόλων, δεν θα απαντήσει στην ερώτηση «πού ορίζεται αυτή η μεταβλητή;». Για αυτό χρησιμοποιείται η βιβλιοθήκη JavaSymbolSolver. Επίσης δεν πρόκειται για μεταγλωττιστή. Ένας συντακτικά σωστός πηγαίος κώδικας που μπορεί να αναλυθεί από τη βιβλιοθήκη, δεν το κάνει να σημαίνει απαραίτητα ότι θα μεταγλωττιστεί με επιτυχία. Αν και τα αρχεία που μεταγλωττίζονται με επιτυχία είναι εγγενώς συντακτικά σωστά, η ανάλυση είναι μόνο ένα στάδιο της διαδικασίας μεταγλώττισης. Για παράδειγμα, η μεταβλητή v δεν έχει οριστεί, ενώ αυτό είναι συντακτικά σωστό, θα οδηγήσει σε σημασιολογικό σφάλμα στη διαδικασία μεταγλώττισης.

Κεφάλαιο 5ο: Η Βιβλιοθήκη JavaSymbolSolver

Πώς συμπληρώνει τον JavaParser

Το JavaParser είναι ένας αναλυτής: δεδομένου ενός αρχείου πηγής, αναγνωρίζει το διαφορετικό συντακτικό στοιχείο και παράγει ένα Abstract Syntax Tree (AST).

Το JavaSymbolSolver αναλύει το AST και βρίσκει τις δηλώσεις που συνδέονται με κάθε στοιχείο.

Το foo στο AST είναι απλώς ένα όνομα, το JavaSymbolSolver μπορεί να πει εάν αναφέρεται σε μια παράμετρο, μια τοπική μεταβλητή, ένα πεδίο. Μπορεί επίσης να σας δώσει τον τύπο, να πει πού έχει οριστεί το στοιχείο κλπ. [10]

Ένας Symbol Solver μπορεί να συσχετίσει ένα σύμβολο στον κώδικά με τη δήλωσή του. Αυτό είναι απαραίτητο για να επαληθευτεί ο τύπος μιας έκφρασης ή για να βρεθεί η χρήση ενός συμβόλου (όπως ένα πεδίο ή μια τοπική μεταβλητή):

Για παράδειγμα:

Πίνακας 5.1: Απλό Παράδειγμα σε Java

```
int a = 0;

void foo() {
    while (true) {
        String a = "hello!";
        Object foo = a + 1;
    }
}
```

Στην έκφραση a + 1 ένας αναλυτής (όπως το JavaParser) δεν είναι σε θέση να πει σε ποιον ορισμό του a αναφερόμαστε και κατά συνέπεια δεν μπορεί να πει τον τύπο του a. Το JavaSymbolSolver μπορεί να το κάνει.

Ρίχνοντας μια ματιά στο JavaParserFacade, για παράδειγμα, παρατηρείται πως μπορεί να χρησιμοποιηθεί για να βρεθεί ο τύπος μιας έκφρασης:

Πίνακας 5.2: Εύρεση Τύπου Έκφρασης

```
Node node = <get this node by parsing source code with JavaParser>
Type typeOfTheNode = JavaParserFacade.get(typeSolver).getType(node);
```

Η μόνη διαμόρφωση που απαιτεί είναι μέρος του στιγμιότυπου TypeSolver για να το περάσει.

Το TypeSolver είναι ο μηχανισμός που χρησιμοποιείται για την εύρεση των κλάσεων που αναφέρονται στον κώδικά. Για παράδειγμα, μια κλάση θα μπορούσε να εισάγει ή να επεκτείνει μια δεδομένη κλάση

και το TypeSolver θα τη βρει και θα δημιουργήσει ένα μοντέλο της, το οποίο αργότερα θα χρησιμοποιηθεί για την επίλυση συμβόλων.

Υπάρχουν τέσσερις TypeSolver:

- JavaParserTypeSolver: αναζήτηση του τύπου σε έναν κατάλογο αρχείων προέλευσης
- JarTypeSolver: αναζήτηση του τύπου σε ένα αρχείο JAR
- ReflectionTypeSolver: αναζήτηση τύπου με ανάκλαση.

  Αυτό είναι απαραίτητο επειδή ορισμένες κλάσεις δεν είναι διαθέσιμες με κανέναν άλλο τρόπο (για παράδειγμα η κλάση Object). Ωστόσο, αυτό θα πρέπει να χρησιμοποιείται αποκλειστικά για αρχεία σε πακέτα java ή javax.

- CombinedTypeSolver: επιτρέπει τον συνδυασμό πολλών παρουσιών TypeSolvers

Στις δοκιμές μπορεί να βρεθεί ένα παράδειγμα στιγμιαίας δημιουργίας TypeSolvers:

Πίνακας 5.3: Δημιουργία TypeSolver

```
CombinedTypeSolver combinedTypeSolver = new CombinedTypeSolver();
combinedTypeSolver.add(new ReflectionTypeSolver());
combinedTypeSolver.add(new JavaParserTypeSolver(new
File("src/test/resources/javaparser_src/proper_source")));
combinedTypeSolver.add(new JavaParserTypeSolver(new
File("src/test/resources/javaparser_src/generated")));
```

Συνήθως, για την ανάλυση ενός project, στόχος είναι να δημιουργηθεί μια παρουσία JavaParserTypeSolver για κάθε κατάλογο source, μια παρουσία JarTypeSolver για κάθε exception και ένα ReflectionTypeSolver, τότε μπορούν να συνδεθούν όλα σε ένα CombinedTypeSolver.

Το JavaSymbolSolver έρχεται τώρα σε πακέτο με την βασική βιβλιοθήκη JavaParser. Αυτό συμβαίνει γιατί κάθε έκδοση του JavaSymbolSolver είναι εγγυημένο ότι λειτουργεί με μια συγκεκριμένη έκδοση του JavaParser. Ιστορικά αυτό δεν ήταν έτσι και οι βιβλιοθήκες διανεμήθηκαν χωριστά. Εάν προκύπτουν προβλήματα με dependencies, είναι απαραίτητη η ενημέρωση στην πιο πρόσφατη έκδοση. Επίσης, εάν προηγουμένως δεν έχει συμπεριληφθεί το JavaSymbolSolver και απλώς συμπεριληφθεί ο πυρήνας javaparser στις εξαρτήσεις σας θα πρέπει να να γίνει κατάλληλη ενημέρωση στην έκδοση του πακέτου που θα πρέπει να αντικατασταθεί αυτή η εξάρτηση. [12]

Πίνακας 5.4: Εξαρτήσεις του JavaSymbolSolver

```
dependencies {
    compile group: 'com.github.javaparser', name: 'javaparser-symbol-
solver-core', v\
ersion: '3.13.3'
}
```

Ένας άλλος τρόπος μέσω του εργαλείου Maven θα ήταν ο εξής:

Πίνακας 5.5: Εξαρτήσεις μέσω του Maven

```
<dependency>
    <groupId>com.github.javaparser</groupId>
    <artifactId>javaparser-symbol-solver-core</artifactId>
    <version>3.13.3</version>
</dependency>
```

Πριν από μερικά χρόνια, ο δημιουργός του JavaSymbolSolver, άρχισε να χρησιμοποιεί το JavaParser και μετά άρχισε να συνεισφέρει. Έπειτα, συνειδητοποίησε ότι πολλές λειτουργίες που γίνονται σε κώδικα Java δεν μπορούν να γίνουν μόνο χρησιμοποιώντας το AST που παράγεται από έναν αναλυτή, αλλά πρέπει επίσης να γίνει επίλυση τύπων, συμβόλων και κλήσεις μεθόδων. Για αυτόν τον λόγο δημιουργήθηκε το JavaSymbolSolver.

Δυστυχώς, ένα σημαντικό πράγμα που λείπει είναι η τεκμηρίωση. Οι προγραμματιστές ανοίγουν ζητήματα στο JavaParser ρωτώντας πώς να απαντήσουν σε μια συγκεκριμένη ερώτηση και η απάντηση είναι συχνά "για αυτό πρέπει να χρησιμοποιήσετε το JavaSymbolSolver". Γιαυτό το λόγο θα ακολουθήσουν μερικά παραδείγματα.

Θα δημιουργηθεί μια λίστα με όλες τις κλήσεις σε μια συγκεκριμένη μέθοδο.

Η επίλυση των κλήσεων μεθόδου σε Java με τη χρήση του JavaSymbolSolver, μπορεί να γίνει σε δύο βήματα:

- Χρήση του JavaParser στον πηγαίο κώδικα για την δημιουργία AST
- Κλήση του JavaSymbolSolver στους κόμβους των AST που αντιπροσωπεύουν κλήσεις μεθόδων

Ως παράδειγμα, μια εφαρμογή με δεδομένο ένα αρχείο source θα μπορούσε να παράγει το παρακάτω:

Πίνακας 5.6: Παραγωγή Κλήσεων Μεθόδων

```
* L55 setId(id)    ->
com.github.javaparser.ast.body.VariableDeclarator.setId(com.github.javapa
rser.ast.body.VariableDeclaratorId)
 * L59 setId(new VariableDeclaratorId(variableName))    ->
com.github.javaparser.ast.body.VariableDeclarator.setId(com.github.javapa
rser.ast.body.VariableDeclaratorId)
 * L71 setId(id)    ->
com.github.javaparser.ast.body.VariableDeclarator.setId(com.github.javapa
rser.ast.body.VariableDeclaratorId)
 * L72 setInit(init)    ->
com.github.javaparser.ast.body.VariableDeclarator.setInit(com.github.java
parser.ast.expr.Expression)
 * L76 setId(new VariableDeclaratorId(variableName))    ->
com.github.javaparser.ast.body.VariableDeclarator.setId(com.github.javapa
rser.ast.body.VariableDeclaratorId)
 * L77 setInit(init)    ->
com.github.javaparser.ast.body.VariableDeclarator.setInit(com.github.java
parser.ast.expr.Expression)
```

```
 * L82 setId(id)   ->
com.github.javaparser.ast.body.VariableDeclarator.setId(com.github.javapa
rser.ast.body.VariableDeclaratorId)
 * L83 setInit(init)   ->
com.github.javaparser.ast.body.VariableDeclarator.setInit(com.github.java
parser.ast.expr.Expression)
 * L88 v.visit(this, arg)   ->
com.github.javaparser.ast.visitor.GenericVisitor.visit(com.github.javapar
ser.ast.body.VariableDeclarator, A)
 * L93 v.visit(this, arg)   ->
com.github.javaparser.ast.visitor.VoidVisitor.visit(com.github.javaparser
.ast.body.VariableDeclarator, A)
 * L106 setAsParentNodeOf(this.id)   ->
com.github.javaparser.ast.Node.setAsParentNodeOf(com.github.javaparser.as
t.Node)
 * L112 setAsParentNodeOf(this.init)   ->
com.github.javaparser.ast.Node.setAsParentNodeOf(com.github.javaparser.as
t.Node)
 * L121 setAsParentNodeOf(this.init)   ->
com.github.javaparser.ast.Node.setAsParentNodeOf(com.github.javaparser.as
t.Node)
 * L128 getParentNodeOfType(NodeWithElementType.class)   ->
com.github.javaparser.ast.Node.getParentNodeOfType(java.lang.Class<T>)
 * L130 wrapInArrayTypes(elementType.getElementType(),
elementType.getArrayBracketPairsAfterElementType(),
getId().getArrayBracketPairsAfterId())   ->
com.github.javaparser.ast.type.ArrayType.wrapInArrayTypes(com.github.java
parser.ast.type.Type,
java.util.List<com.github.javaparser.ast.ArrayBracketPair>...)
 * L130 elementType.getElementType()   ->
com.github.javaparser.ast.nodeTypes.NodeWithElementType.getElementType()
 * L131 elementType.getArrayBracketPairsAfterElementType()   ->
com.github.javaparser.ast.nodeTypes.NodeWithElementType.getArrayBracketPa
irsAfterElementType()
 * L132 getId().getArrayBracketPairsAfterId()   ->
com.github.javaparser.ast.body.VariableDeclaratorId.getArrayBracketPairsA
fterId()
 * L132 getId()   ->
com.github.javaparser.ast.body.VariableDeclarator.getId()
 * L137 ArrayType.unwrapArrayTypes(type)   ->
com.github.javaparser.ast.type.ArrayType.unwrapArrayTypes(com.github.java
parser.ast.type.Type)
 * L138 getParentNodeOfType(NodeWithElementType.class)   ->
com.github.javaparser.ast.Node.getParentNodeOfType(java.lang.Class<T>)
 * L142 nodeWithElementType.setElementType(unwrapped.a)   ->
com.github.javaparser.ast.nodeTypes.NodeWithElementType.setElementType(co
m.github.javaparser.ast.type.Type<?>)
 * L143 nodeWithElementType.setArrayBracketPairsAfterElementType(null)
->
com.github.javaparser.ast.nodeTypes.NodeWithElementType.setArrayBracketPa
irsAfterElementType(java.util.List<com.github.javaparser.ast.ArrayBracket
Pair>)
 * L144 getId().setArrayBracketPairsAfterId(unwrapped.b)   ->
com.github.javaparser.ast.body.VariableDeclaratorId.setArrayBracketPairsA
fterId(java.util.List<com.github.javaparser.ast.ArrayBracketPair>)
 * L144 getId()   ->
com.github.javaparser.ast.body.VariableDeclarator.getId()
```

Το αρχείο source είναι το ακόλουθο:

Πίνακας 5.7: Πηγαίο Αρχείο Κλήσεων Μεθόδων

```
package com.github.javaparser.ast.body;

import com.github.javaparser.Range;
import com.github.javaparser.ast.ArrayBracketPair;
import com.github.javaparser.ast.Node;
import com.github.javaparser.ast.expr.Expression;
import com.github.javaparser.ast.expr.NameExpr;
import com.github.javaparser.ast.nodeTypes.NodeWithElementType;
import com.github.javaparser.ast.nodeTypes.NodeWithType;
import com.github.javaparser.ast.type.ArrayType;
import com.github.javaparser.ast.type.Type;
import com.github.javaparser.ast.visitor.GenericVisitor;
import com.github.javaparser.ast.visitor.VoidVisitor;
import com.github.javaparser.utils.Pair;

import java.util.List;

import static com.github.javaparser.ast.type.ArrayType.wrapInArrayTypes;

/**
 * @author Julio Vilmar Gesser
 */
public final class VariableDeclarator extends Node implements
        NodeWithType<VariableDeclarator> {

    private VariableDeclaratorId id;

    private Expression init;

    public VariableDeclarator() {
    }

    public VariableDeclarator(VariableDeclaratorId id) {
        setId(id);
    }

    public VariableDeclarator(String variableName) {
        setId(new VariableDeclaratorId(variableName));
    }

    /**
     * Defines the declaration of a variable.
     *
     * @param id The identifier for this variable. IE. The variables
name.
     * @param init What this variable should be initialized to.
     *             An {@link com.github.javaparser.ast.expr.AssignExpr} is
unnecessary as the <code>=</code> operator is
     *             already added.
     */
    public VariableDeclarator(VariableDeclaratorId id, Expression init) {
        setId(id);
        setInit(init);
    }

    public VariableDeclarator(String variableName, Expression init) {
```

```
        setId(new VariableDeclaratorId(variableName));
        setInit(init);
    }

    public VariableDeclarator(Range range, VariableDeclaratorId id,
Expression init) {
        super(range);
        setId(id);
        setInit(init);
    }

    @Override
    public <R, A> R accept(GenericVisitor<R, A> v, A arg) {
        return v.visit(this, arg);
    }

    @Override
    public <A> void accept(VoidVisitor<A> v, A arg) {
        v.visit(this, arg);
    }

    public VariableDeclaratorId getId() {
        return id;
    }

    public Expression getInit() {
        return init;
    }

    public VariableDeclarator setId(VariableDeclaratorId id) {
        this.id = id;
        setAsParentNodeOf(this.id);
        return this;
    }

    public VariableDeclarator setInit(Expression init) {
        this.init = init;
        setAsParentNodeOf(this.init);
        return this;
    }

    /**
     * Will create a {@link NameExpr} with the init param
     */
    public VariableDeclarator setInit(String init) {
        this.init = new NameExpr(init);
        setAsParentNodeOf(this.init);
        return this;
    }


    @Override
    public Type getType() {
        NodeWithElementType<?> elementType =
getParentNodeOfType(NodeWithElementType.class);

        return wrapInArrayTypes(elementType.getElementType(),
                elementType.getArrayBracketPairsAfterElementType(),
                getId().getArrayBracketPairsAfterId());
    }
```

```
    @Override
    public VariableDeclarator setType(Type type) {
        Pair<Type, List<ArrayBracketPair>> unwrapped =
ArrayType.unwrapArrayTypes(type);
        NodeWithElementType<?> nodeWithElementType =
getParentNodeOfType(NodeWithElementType.class);
        if (nodeWithElementType == null) {
            throw new IllegalStateException("Cannot set type without a
parent");
        }
        nodeWithElementType.setElementType(unwrapped.a);
        nodeWithElementType.setArrayBracketPairsAfterElementType(null);
        getId().setArrayBracketPairsAfterId(unwrapped.b);
        return this;
    }
}
```

Ρύθμιση του project

Το παράδειγμα γίνεται με τη χρήση Kotlin και Gradle και το αρχείο build είναι της παρακάτω μορφής:

Πίνακας 5.8: Ρύθμιση με χρήση Kotlin και Gradle

```
buildscript {
   ext.kotlin_version = '1.0.4'

   repositories {
     mavenCentral()
     maven {
       name 'JFrog OSS snapshot repo'
       url  'https://oss.jfrog.org/oss-snapshot-local/'
     }
     jcenter()
   }

   dependencies {
     classpath "org.jetbrains.kotlin:kotlin-gradle-
plugin:$kotlin_version"
   }
}

apply plugin: 'kotlin'
apply plugin: 'application'
apply plugin: 'java'
apply plugin: 'idea'
apply plugin: 'antlr'

repositories {
    mavenLocal()
    mavenCentral()
    jcenter()
}

dependencies {
    compile "me.tomassetti:java-symbol-solver-core:0.3.1"
    compile "org.jetbrains.kotlin:kotlin-stdlib:$kotlin_version"
    compile "org.jetbrains.kotlin:kotlin-reflect:$kotlin_version"
    testCompile "org.jetbrains.kotlin:kotlin-test:$kotlin_version"
    testCompile "org.jetbrains.kotlin:kotlin-test-junit:$kotlin_version"
```

```
    testCompile "junit:junit:latest.release"
}

idea {
    module {
        excludeDirs += file('src/main/resources')
    }
}
```

Κατασκευή AST

Η κατασκευή ενός AST είναι αρκετά εύκολη και γίνεται με την κλήση της παρακάτω μεθόδου:

Πίνακας 5.9: Μέθοδος Κατασκευής Αφηρημένου Συντακτικού Δέντρου

```
JavaParser.parse(file)
```

Μπορούν επίσης να χρησιμοποιηθούν μερικές μέθοδοι για την πλοήγηση στο AST και τον έλεγχο συγκεκριμένων κόμβων. Συγκεκριμένα, θα χρησιμοποιηθεί για να ληφθούν μονάχα οι κλήσεις μεθόδων, νε τον παρακάτω τρόπο:

Πίνακας 5.10: Μέθοδοι Πλοήγησης Αφηρημένου Συντακτικού Δέντρου

```
// this is a method extension: we had this method to the existing class
"Node"
fun <T> Node.descendantsOfType(type: Class<T>) : List<T> {
    val descendants = LinkedList<T>()
    SpecificNodeIterator(type, object :
SpecificNodeIterator.NodeHandler<T> {
        override fun handle(node: T): Boolean {
            descendants.add(node)
            return true
        }
    }).explore(this)
    return descendants
}
```

Καθορισμός Type Solver

Ένας Type Solver αποτελεί το αντικείμενο που γνωρίζει σε ποιο σημείο να αναζητήσει κλάσεις. Κατά την επεξεργασία του πηγαίου κώδικα υπάρχουν συνήθως αναφορές σε κώδικα που δεν έχει ακόμη μεταγλωττιστεί, αλλά υπάρχει απλώς σε άλλα αρχεία πηγής. Υπάρχει επίσης η δυνατότητα να χρησιμοποιηθούν κλάσεις που περιέχονται σε JAR ή κλάσεις από τις τυπικές βιβλιοθήκες Java. Απλώς πρέπει να ρυθμιστεί ο TypeSolver κατάλληλα, έτσι ώστε να ψάξει για κλάσεις και θα το καταλάβει.

Στο παράδειγμα θα αναλυθεί ο πηγαίος κώδικας από το έργο JavaParser. Φυσικά θα χρησιμοποιηθούν και κλάσεις από τις τυπικές βιβλιοθήκες java. Παρακάτω ακολουθεί ο TypeSolver:

Πίνακας 5.11: Παράδειγμα Καθορισμού TypeSolver

```
fun typeSolver() : TypeSolver {
    val combinedTypeSolver = CombinedTypeSolver()
    combinedTypeSolver.add(JreTypeSolver())

combinedTypeSolver.add(JavaParserTypeSolver(File("src/main/resources/java
parser-core")))

combinedTypeSolver.add(JavaParserTypeSolver(File("src/main/resources/java
parser-generated-sources")))
    return combinedTypeSolver
}
```

Η εφαρμογή

Εδώ γίνεται επίκληση του JavaParserFacade, το οποίο αποτελεί μια από τις κλάσεις που παρέχονται από το JavaSymbolSolver. Απλώς θα ληφθεί μια κλήση μεθόδου εκείνη τη στιγμή και θα περάσει στην επίλυση μεθόδου του JavaParserFacade. Συγκεκριμένα, γίνεται ένα MethodUsage (που είναι βασικά μια δήλωση μεθόδου, μαζί με την τιμή των τύπων παραμέτρων για τη συγκεκριμένη επίκληση). Από αυτό καταλήγει στο MethodDeclaration και εκτύπωση της αναγνωρισμένης υπογραφής, δηλαδή το αναγνωρισμένο όνομα της κλάσης ακολουθούμενο από την υπογραφή της μεθόδου.

Έτσι παίρνουμε το τελικό αποτέλεσμα:

Πίνακας 5.11: Παράδειγμα με JavaParserFacade

```
var solved = 0
var unsolved = 0
var errors = 0

fun processJavaFile(file: File, javaParserFacade: JavaParserFacade) {
    println(file)

JavaParser.parse(file).descendantsOfType(MethodCallExpr::class.java).forE
ach {
        print(" * L${it.begin.line} $it ")
        try {
            val methodRef = javaParserFacade.solve(it)
            if (methodRef.isSolved) {
                solved++
                val methodDecl = methodRef.correspondingDeclaration
                println("  -> ${methodDecl.qualifiedSignature}")
            } else {
                unsolved++
                println(" ???")
            }
        } catch (e: Exception) {
            println(" ERR ${e.message}")
            errors++
        } catch (t: Throwable) {
            t.printStackTrace()
        }
    }
}
```

Συνοπτικά

Υπάρχουν διάφοροι τρόποι για να γίνουν οι παραπάνω δουλειές, αλλά βασικά το JavaSymbolSolver κάνει όλη τη βαριά δουλειά στο παρασκήνιο. Έχοντας μονάχα έναν κόμβο από ένα AST, περνώντας το στην κλάση JavaParserFacade, δίνει πίσω όλες τις πληροφορίες που μπορεί να χρειαστούν, όπως το να βρει τους αντίστοιχους τύπους, πεδία και μεθόδους.

Το πρόβλημα που εμφανίζεται είναι η έλλειψη υλικού από τους χρήστες. Χρειάζεται περισσότερη τεκμηρίωση και σχόλια από τους χρήστες. Είναι απαραίτητο να αρχίσουν οι χρήστες να χρησιμοποιούν το JavaSymbolSolver έτσι ώστε να βρεθούν τρόποι βελτίωσής του.

Πλέον το JavaSymbolSolver μεταφέρθηκε στον οργανισμό JavaParser, επομένως θα υπάρχει μεγαλύτερη συνεργασία μεταξύ των δύο έργων.

Κεφάλαιο 6ο: Το Αφηρημένο Συντακτικό Δέντρο AST

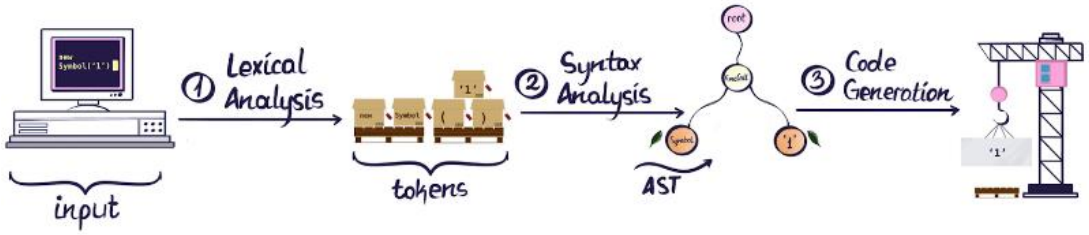
Οι χρήστες που γράφουν κώδικα έχουν συναντήσει τουλάχιστον μια φορά τα AST, κατά τη ροή ανάπτυξης των προγραμμάτων τους. Το AST σημαίνει Abstract Syntax Tree και τροφοδοτεί πολλά μέρη της ροής ανάπτυξής του προγράμματος. Μερικοί άνθρωποι μπορεί να έχουν ακούσει γι 'αυτά στο πλαίσιο των μεταγλωττιστών, αλλά χρησιμοποιούνται σε μια μεγαλύτερη ποικιλία εργαλείων. Ακόμα κι αν ένας χρήστης δεν χρησιμοποιεί γενικά εργαλεία ανάπτυξης, τα AST μπορεί να είναι χρήσιμα στα εργαλείων του. [13]

Παρακάτω θα δοθούν πληροφορίες σχετικά με το τι είναι το AST, που χρησιμοποιείται και πως μπορεί να αξιοποιηθεί.

AST

Τα AST είναι δενδρικές αναπαραστάσεις κώδικα και αποτελούν θεμελιώδες μέρος του τρόπου με τον οποίο λειτουργεί ένας μεταγλωττιστής. Όταν ένας μεταγλωττιστής μετασχηματίζει κάποιο κώδικα, βασικά υπάρχουν τα ακόλουθα βήματα:

1. Λεξιλογική/Λεκτική Ανάλυση (Lexical Analysis)
2. Συντακτική Ανάλυση (Syntax Analysis)
3. Δημιουργία Κώδικα (Code Generation)



Σχήμα 6.1: Αναπαράσταση Λειτουργίας AST

Λεκτική Ανάλυση ή Tokenization

Κατά τη διάρκεια αυτού του βήματος, ο κώδικας που έχει γραφεί θα μετατραπεί σε ένα σύνολο λεκτικών μονάδων (tokens) που περιγράφουν τα διάφορα μέρη του κώδικα. Αυτή είναι βασικά η ίδια μέθοδος που χρησιμοποιεί η βασική επισήμανση σύνταξης. Τα tokens δεν καταλαβαίνουν πώς τα πράγματα ταιριάζουν μεταξύ τους και επικεντρώνονται καθαρά σε στοιχεία ενός αρχείου. Είναι παρόμοιο μια λίστα ή μια σειρά διαφορετικών τύπων tokens.

Είναι παρόμοιο επίσης, με τον διαχωρισμό των λέξεων ενός κειμένου. Η διαφοροποίηση μπορεί να γίνει μεταξύ στίξης, ρημάτων, ουσιαστικών, αριθμών κλπ., αλλά σε αυτό το στάδιο ίσως δεν υπάρχει βαθύτερη κατανόηση του τι είναι μέρος μιας πρότασης ή πώς οι προτάσεις ταιριάζουν μεταξύ τους.

Συντακτική Ανάλυση ή Parsing

Αυτό είναι το βήμα όπου μετατρέπεται η λίστα των tokens σε AST. Συγκεκριμένα, μετατρέπει τα tokens σε ένα δέντρο που αντιπροσωπεύει την πραγματική δομή του κώδικα. Όπου προηγουμένως στα tokens υπήρχε ένα μόνο ζεύγος (), πλέον δημιουργείται μια ιδέα για το αν είναι κλήση συνάρτησης, ορισμός συνάρτησης, ομαδοποίηση ή κάτι άλλο.

Το αντίστοιχο εδώ θα ήταν να μετατραπεί η λίστα των λέξεων σε μια δομή δεδομένων που αντιπροσωπεύει πράγματα όπως προτάσεις, τι ρόλο παίζει ένα συγκεκριμένο ουσιαστικό σε μια πρόταση ή αν πρόκειται για μια λίστα.

Ένα άλλο παράδειγμα με το οποίο μπορεί να συγκριθεί αποτελεί το DOM. Στο προηγούμενο βήμα αναλύει απλώς το HTML σε "ετικέτες" και "κείμενο", αυτό το βήμα θα δημιουργούσε την ιεραρχία που αντιπροσωπεύεται ως DOM.

Ένα πράγμα που πρέπει να σημειωθεί είναι ότι δεν υπάρχει "μία" μορφή AST. Μπορεί να διαφέρουν τόσο ανάλογα με τη γλώσσα που μετατρέπεται σε AST όσο και με το εργαλείο που θα χρησιμοποιηθεί για την ανάλυση.

Πίνακας 6.1: Παράδειγμα με AST

```
{
 "type": "Program",
 "start": 0,
 "end": 14,
 "body": [
   {
     "type": "ExpressionStatement",
     "start": 0,
     "end": 14,
     "expression": {
       "type": "CallExpression",
       "start": 0,
       "end": 13,
       "callee": {
         "type": "Identifier",
         "start": 0,
         "end": 7,
         "name": "isPanda"
       },
       "arguments": [
         {
           "type": "Literal",
           "start": 8,
           "end": 12,
           "value": "🐼",
           "raw": "'🐼'"
         }
       ]
     }
   }
 ],
 "sourceType": "module"
}
```

Γενικά ένα AST είναι μια δομή δέντρου όπου κάθε κόμβος έχει τουλάχιστον έναν τύπο που καθορίζει τι αντιπροσωπεύει. Για παράδειγμα, ένας τύπος θα μπορούσε να είναι ένα Literal που αντιπροσωπεύει μια πραγματική τιμή ή ένα CallExpression που αντιπροσωπεύει μια κλήση συνάρτησης. Ο κόμβος Literal μπορεί να περιέχει μόνο μια τιμή ενώ ο κόμβος CallExpression περιέχει πολλές πρόσθετες πληροφορίες που μπορεί να είναι σχετικές, όπως το "τι καλείται" (callee) ή ποια είναι τα ορίσματα που μεταβιβάζονται.

Δημιουργία Κώδικα

Αυτό το βήμα μπορεί να αποτελεί πολλαπλά βήματα από μόνο του. Μόλις διατίθεται ένα Abstract Syntax Tree, μπορεί να χειριστεί καθώς και να το εκτυπωθεί σε διαφορετικό τύπο κώδικα. Η χρήση AST για τον χειρισμό του κώδικα είναι ασφαλέστερη από το να γίνουν αυτές οι λειτουργίες απευθείας στον κώδικα ως κείμενο ή μια λίστα tokens.

Η χειραγώγηση κειμένου είναι πάντα επικίνδυνη, λόγω του ότι δείχνει το λιγότερο πλαίσιο. Αν χρειαστεί ένας χρήστης να χειριστεί ένα κείμενο χρησιμοποιώντας αντικαταστάσεις συμβολοσειρών ή κανονικές εκφράσεις, είναι αρκετά εύκολο να γίνουν λάθη.

Ακόμη και ο χειρισμός των tokens δεν είναι εύκολος. Ενώ μπορεί να είναι γνωστό τι είναι μια μεταβλητή, αν πρόκειται να μετονομαστεί, δεν υπάρχει ξεκάθαρη εικόνα για πράγματα όπως το εύρος της μεταβλητής ή τυχόν μεταβλητές με τις οποίες μπορεί να έρχεται σε σύγκρουση.

Το AST παρέχει αρκετές πληροφορίες σχετικά με τη δομή του κώδικα και έτσι μπορεί να τροποποιηθεί με μεγαλύτερη σιγουριά. Θα μπορούσε να προσδιοριστεί για παράδειγμα, πού δηλώνεται μια μεταβλητή και να είναι γνωστό σε ποιο μέρος ακριβώς του προγράμματος επηρεάζει λόγω της δομής δέντρου.

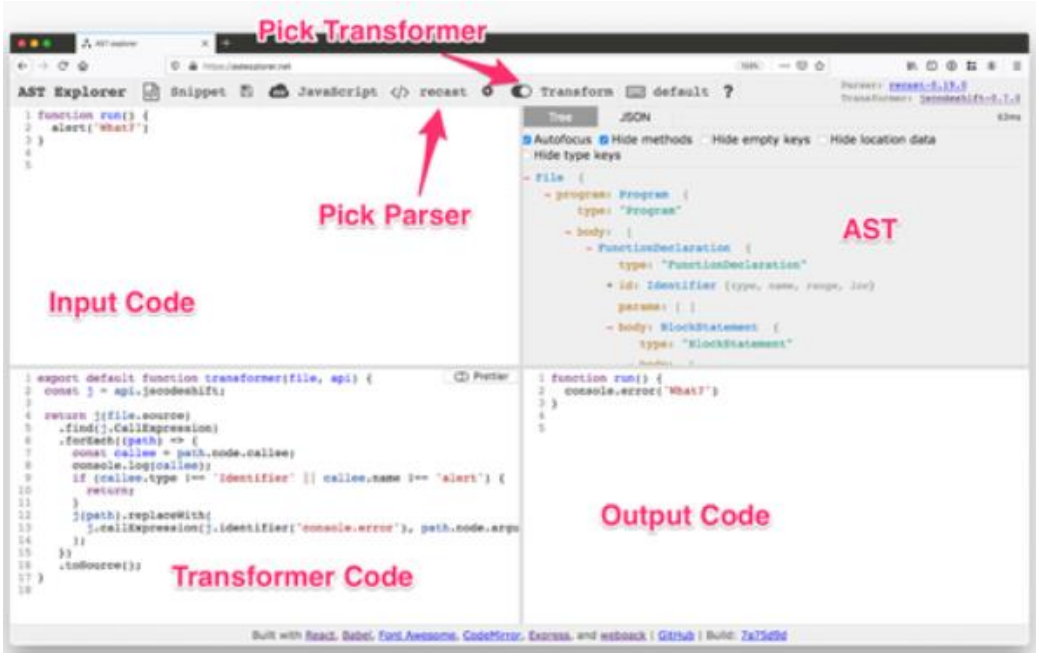
Αφού ο χρήστης χειριστεί το δέντρο, μπορεί στη συνέχεια να εκτυπώσει το δέντρο για να παράγει όποια έξοδο κώδικα αναμένεται. Για παράδειγμα, αν κατασκεύαζε έναν μεταγλωττιστή όπως ο μεταγλωττιστής TypeScript, θα προέκυπτε JavaScript ενώ ένας άλλος μεταγλωττιστής θα μπορούσε να παράγει κώδικα μηχανής.

Και πάλι αυτό επιτυγχάνεται πιο εύκολα με ένα AST επειδή διαφορετικές έξοδοι μπορεί να έχουν διαφορετικές μορφές για την ίδια δομή. Θα ήταν πιο δύσκολο να δημιουργηθεί η έξοδος με μια πιο γραμμική είσοδο όπως κείμενο ή μια λίστα διακριτικών.

Η χρησιμότητα των ASTs

Οι περιπτώσεις χρήσης για τα AST είναι ευρείες και μπορούν γενικά να χωριστούν σε τρεις γενικές ενέργειες: ανάγνωση, τροποποίηση και εκτύπωση. Είναι ένα είδος πρόσθετου, που σημαίνει ότι εάν ένας χρήστης εκτυπώνει AST, είναι μεγάλες οι πιθανότητες να έχει διαβάσει και στο παρελθόν ένα AST και να το έχει τροποποιήσει. Παρακάτω θα δοθούν παραδείγματα για την κάλυψη και των τριών περιπτώσεων.

Για το κάθε παράδειγμα θα δίνεται ο τρόπος της κάθε ενέργειας και τα παραδείγματα θα γίνουν σε JavaScript. Αυτό δεν αποτελεί εμπόδιο, μιας και οι περισσότερες γλώσσες προγραμματισμού έχουν παρόμοια εργαλεία και βιβλιοθήκες.



Σχήμα 6.2: Εργαλείο ASTEXPLORER

Ανάγνωση των AST

Τεχνικά, το πρώτο βήμα της εργασίας με AST είναι η ανάλυση κειμένου για τη δημιουργία ενός AST, αλλά στις περισσότερες περιπτώσεις οι βιβλιοθήκες που προσφέρουν το βήμα ανάλυσης προσφέρουν επίσης έναν τρόπο να διασχίσει το AST.

Η διέλευση ενός AST σημαίνει επίσκεψη στους διαφορετικούς κόμβους του δέντρου για να αποκτηθούν πληροφορίες ή να εκτελεστούν ενέργειες.

Μία από τις πιο συνηθισμένες περιπτώσεις χρήσης για αυτό είναι το linting. Το ESLint, για παράδειγμα, χρησιμοποιεί το espree για να δημιουργήσει ένα AST και αν ένας χρήστης επιθυμεί να γράψει οποιουσδήποτε προσαρμοσμένους κανόνες, θα γράψει αυτούς που βασίζονται στους διαφορετικούς κόμβους AST. Τα έγγραφα ESLint διαθέτουν εκτενή τεκμηρίωση για το πώς μπορεί ένας χρήστης να δημιουργήσει προσαρμοσμένους κανόνες, πρόσθετα και μορφοποιητές (formatters).

Βλέποντας ένα παράδειγμα κώδικα στον εξερευνητή AST που θεσπίζει έναν κανόνα ότι μπορεί να υπλαρχει μόνο μία κλάση στο αρχείο:

Πίνακας 6.2:

```
let numberOfClasses = 0;

export default function (context) {
  return {
    ClassDeclaration(node) {
      numberOfClasses = numberOfClasses + 1;
      if (numberOfClasses > 1) {
        context.report({
          node,
```

```
            message: "You shouldn't use more than one class",
          });
        }
      },
    };
}
```

Σε αυτό το απόσπασμα κώδικα γίνεται αναζήτηση κόμβων ClassDeclaration και κάθε φορά αυξάνεται ένας global μετρητής. Μόλις φτάσει σε ένα συγκεκριμένο όριο, χρησιμοποιείται το API αναφοράς του ESLint για να υποβάλουμε ένα παράπονο.

Τώρα αυτή είναι μια πολύ συγκεκριμένη σύνταξη ESLint, αλλά ένας χρήστης θα μπορούσε να δημιουργήσει ένα παρόμοιο σενάριο χωρίς να δημιουργήσει ένα πρόσθετο ESLint. Θα μπορούσε για παράδειγμα να χρησιμοποιήσει την υποκείμενη βιβλιοθήκη espree για να κάνει την ανάλυση και τη διέλευση κόμβων με μη αυτόματο τρόπο χρησιμοποιώντας ένα βασικό σενάριο Node.js.

Πίνακας 6.3:

```
const fs = require('fs').promises;
const path = require('path');
const espree = require('espree');

function checkTopLevelClasses(ast) {
 let topLevelClassCounter = ast.body.reduce((counter, node) => {
   if (node.type === 'ClassDeclaration') {
     counter++;
   }
   return counter;
 }, 0);

 if (topLevelClassCounter > 1) {
   throw new Error(
     `Found ${topLevelClassCounter} top level classes. Expected not more
than one.`
   );
 }
}

async function run() {
 const fileName = path.resolve(process.cwd(), process.argv[2]);
 const content = await fs.readFile(fileName, 'utf8');
 console.log(fileName);

 const ast = espree.parse(content, { ecmaVersion: 2019 });
 checkTopLevelClasses(ast);
}

run().catch(console.error);
```

Αυτό το σενάριο διαβάζει με μη αυτόματο τρόπο ένα αρχείο, το αναλύει χρησιμοποιώντας το espree και, στη συνέχεια, ελέγχει κάθε κόμβο ανώτατου επιπέδου και αν πρόκειται για ClassDeclaration, οπότε και θα αυξήσει έναν τοπικό μετρητή. Μόλις ολοκληρωθεί, ελέγχει αν το μέτρημα είναι μεγαλύτερο από το αναμενόμενο και θα εμφανίσει σφάλμα.

Όπως φαίνεται και παραπάνω, περιλαμβάνει πολύ πρόσθετο κώδικα σε σύγκριση με το προηγούμενο παράδειγμά. Τα πρόσθετα για τα υπάρχοντα linters είναι σίγουρα οι πιο αποδοτικές λύσεις κώδικα, αλλά μερικές φορές χρειάζεται περισσότερο έλεγχο.

Αναζητώντας για το npm, μπορεί κανείς να βρει μια συλλογή από άλλα εργαλεία για την ανάλυση και τη διέλευση AST. Συνήθως διαφέρουν στο σχεδιασμό API και περιστασιακά στις δυνατότητες ανάλυσης JavaScript. Μερικά κοινά παραδείγματα είναι το acorn και το esprima για ανάλυση ή το estree-walker για τη διέλευση δέντρων συμβατά με ESTree.

Τροποποίηση/Μετασχηματισμός AST

Όπως καλύφθηκε προηγουμένως, η ύπαρξη ενός AST καθιστά την τροποποίηση του εν λόγω δέντρου ευκολότερη και ασφαλέστερη από την τροποποίηση του κώδικα ως διακριτικά ή ακατέργαστη συμβολοσειρά. Υπάρχει μεγάλη ποικιλία λόγων για τους οποίους μπορεί να θέλει κάποιος να τροποποιήσει κάποιο κώδικα χρησιμοποιώντας AST.

Το Babel, για παράδειγμα, τροποποιεί τα AST για τη μετατροπή νεότερων δυνατοτήτων ή για τη μετατροπή του JSX σε κλήσεις συναρτήσεων. Αυτό συμβαίνει όταν για παράδειγμα μεταγλωττίζεται ο κώδικα React ή Preact.

Μια άλλη περίπτωση χρήσης θα ήταν η ομαδοποίηση κώδικα. Στον κόσμο των λειτουργικών μονάδων, η ομαδοποίηση κώδικα είναι συχνά πολύ πιο περίπλοκη από την απλή προσθήκη αρχείων μαζί. Η καλύτερη κατανόηση της δομής των αντίστοιχων αρχείων, διευκολύνει τη συγχώνευση αυτών των αρχείων και την προσαρμογή των εισαγωγών και των κλήσεων λειτουργιών όπου χρειάζεται. Αν παρατηρηθούν οι βάσεις κωδικών των εργαλείων όπως το webpack, το δέμα ή η συλλογή, θα διαπιστωθεί ότι όλα χρησιμοποιούν το AST ως μέρος της ροής εργασιών ομαδοποίησης.

Μια περίπτωση χρήσης που μπορεί να φαίνεται λιγότερο προφανής είναι η κάλυψη δοκιμής. Σε περίπτωση που κάποιος έχει αναρωτηθεί πώς τα εργαλεία κάλυψης κωδικών όπως το Istanbul καθορίζουν την κάλυψη της δοκιμής σας, αυτό που ισχύει είναι πως τα tl;dr εισάγουν πρόσθετο κώδικα που αυξάνει διαφορετικούς μετρητές για κάθε γραμμή, συνάρτηση και πρόταση. Μετά από όλες τις δοκιμές του χρήστη μπορούν να επιθεωρήσουν τους εν λόγω μετρητές και να του δώσουν μια λεπτομερή εικόνα για το τι έχει εκτελεστεί και τι όχι. Το να γίνει αυτό χωρίς AST είναι τόσο απίστευτα δύσκολο όσο και λιγότερο προβλέψιμο.

Τώρα όλα αυτά είναι εργαλεία που πιθανότατα δεν θα χρειαστεί να γράψει μόνος του ένας χρήστης. Αλλά υπάρχει μια περίπτωση χρήσης που μπορεί να είναι επωφελής για την τακτική ροή ανάπτυξης. Και αυτό κάνει τροποποιήσεις κώδικα για βελτιστοποίηση, μακροεντολές ή για ενημέρωση μεγαλύτερων τμημάτων της βάσης του κώδικα ταυτόχρονα.

Η ομάδα React, για παράδειγμα, διατηρεί μια συλλογή σεναρίων που ονομάζεται react-codemod και μπορεί να εκτελέσει κοινές λειτουργίες που σχετίζονται με την ενημέρωση της έκδοσης React που έχει ο χρήστης. Το εργαλείο που χρησιμοποιούν κάτω από την κουκούλα ονομάζεται jscodeshift και μπορεί να χρησιμοποιηθεί για να γράψει ο χρήστης και τα δικά του σενάρια μετασχηματισμού.

Έστω πως ένας χρήστης θέλει να κάνει εντοπισμό σφαλμάτων χρησιμοποιώντας το alert() αλλά θέλει να αποφύγει την αποστολή στους πελάτες του. Θα μπορούσε να γράψει ένα σενάριο όπως το παρακάτω για να αντικαταστήσει όλες τις κλήσεις για ειδοποίηση με το console.error χωρίς να ανησυχεί ότι μπορεί να παρακάμψει κάτι σαν το myalert().

Πίνακας 6.4:

```
export default function transformer(file, api) {
 const j = api.jscodeshift;

 return j(file.source)
   .find(j.CallExpression)
   .forEach((path) => {
     const callee = path.node.callee;
     console.log(callee);
     if (callee.type !== 'Identifier' || callee.name !== 'alert') {
       return;
     }
     j(path).replaceWith(
       j.callExpression(j.identifier('console.error'),
path.node.arguments)
     );
   })
   .toSource();
}
```

Εκτύπωση ASTs

Στις περισσότερες περιπτώσεις, η εκτύπωση και η τροποποίηση AST συμβαδίζουν, καθώς θα πρέπει να εξαχθεί το AST που μόλις τροποποιήθηκε. Ωστόσο, ενώ ορισμένες βιβλιοθήκες, όπως η αναδιατύπωση, εστιάζουν ρητά στην εκτύπωση του AST με το ίδιο στυλ κώδικα με το πρωτότυπο, υπάρχει επίσης μια ποικιλία περιπτώσεων χρήσης όπου μπορεί ένας χρήστης να θέλει να εκτυπώσει το AST διαφορετικά.

Το Prettier, για παράδειγμα, χρησιμοποιεί AST για να διαμορφώσει ξανά τον κώδικα σύμφωνα με τη διαμόρφωση του χρήστη χωρίς να αλλάξει το περιεχόμενο του κώδικά του. Ο τρόπος με τον οποίο το κάνει είναι μετατρέποντας τον κώδικά του σε ένα AST που διαμορφώνει πλήρως αγνωστικιστικό και στη συνέχεια ξαναγράφοντας τον με βάση τους κανόνες του χρήστη.

Συνήθεις άλλες περιπτώσεις χρήσης θα ήταν η εκτύπωση κώδικα σε διαφορετική γλώσσα ή η δημιουργία εργαλείου ελαχιστοποίησης του χρήστη.

Υπάρχουν μερικά διαφορετικά εργαλεία που μπορεί να χρησιμοποιήσει για να εκτυπώσει AST, όπως το escodegen ή το astring. Θα μπορούσε επίσης να προχωρήσει και να δημιουργήσει το δικό του μορφοποιητή ανάλογα με την περίπτωση χρήσης του ή να δημιουργήσει ένα πρόσθετο για το Prettier.

Κεφάλαιο 7ο: Το Εργαλείο Maven της Java

Ανεξάρτητα από το πόσο μικρές ή μεγάλες είναι, όλες οι εφαρμογές πρέπει να υποβληθούν σε μια συγκεκριμένη σειρά διαδικασιών, όπως η δημιουργία και η μεταγλώττιση του πηγαίου κώδικα. Οι προγραμματιστές μπορούν να ρυθμίσουν μη αυτόματα αυτές τις διαδικασίες, αλλά είναι ένα χρονοβόρο βάρος. [14]

Για να λυθεί αυτό το πρόβλημα, εμφανίζεται το Apache Maven, το οποίο αυτοματοποιεί ολόκληρη τη διαδικασία και διευκολύνει την καθημερινή εργασία των προγραμματιστών Java. Παρακάτω δίνονται πληροφορίες σχετικά με το τί είναι το Maven και γιατί πρόκειται για ένα καλό εργαλείο DevOps.

Maven

Το Maven είναι ένα δημοφιλές εργαλείο δημιουργίας ανοιχτού κώδικα που αναπτύχθηκε από τον Όμιλο Apache για τη δημιουργία, τη δημοσίευση και την ανάπτυξη πολλών έργων ταυτόχρονα για καλύτερη διαχείριση έργων. Το εργαλείο παρέχει στους προγραμματιστές τη δυνατότητα να δημιουργήσουν και να τεκμηριώσουν το πλαίσιο του κύκλου ζωής των προγραμμάτων τους. [15]

Σχήμα 7.1: Διάφορες Γνωστές Τεχνολογίες

Είναι γραμμένο σε γλώσσα Java και χρησιμοποιείται για τη δημιουργία έργων γραμμένων σε C#, Scala, Ruby κλπ. Βασισμένο στο Project Object Model (POM), αυτό το εργαλείο έχει κάνει τη ζωή των προγραμματιστών Java ευκολότερη κατά την ανάπτυξη αναφορών, ελέγχων κατασκευής και δοκιμής αυτοματισμού ρυθμίσεις.

Η Maven εστιάζει στην απλοποίηση και τυποποίηση της διαδικασίας δόμησης, μεριμνώντας για τα ακόλουθα:

- Κατασκευές (Builds)
- Τεκμηρίωση (Documentation)
- Εξαρτήσεις (Dependencies)
- Αναφορές (Reports)
- SCMs
- Διανομή (Distribution)
- Εκδόσεις (Releases)
- Λίστα Αλληλογραφίας (Mailing Lists)

Εξέλιξη

Το Maven δημιουργήθηκε για να απλοποιήσει τις διαδικασίες κατασκευής του έργου Jakarta Tribune. Πολλά από τα έργα είχαν ελαφρώς διαφορετικά αρχεία ANT, έτσι η Apache ανέπτυξε το Maven για να χειρίζεται την κατασκευή πολλών έργων μαζί, συμπεριλαμβανομένης της δημοσίευσης πληροφοριών έργων, της διευκόλυνσης της συνεργασίας ομάδας, της ανάπτυξης έργων και της κοινής χρήσης JARS μεταξύ πολλών έργων.

Σκοπός

Σκοπός του Maven είναι να παρέχει στους προγραμματιστές:

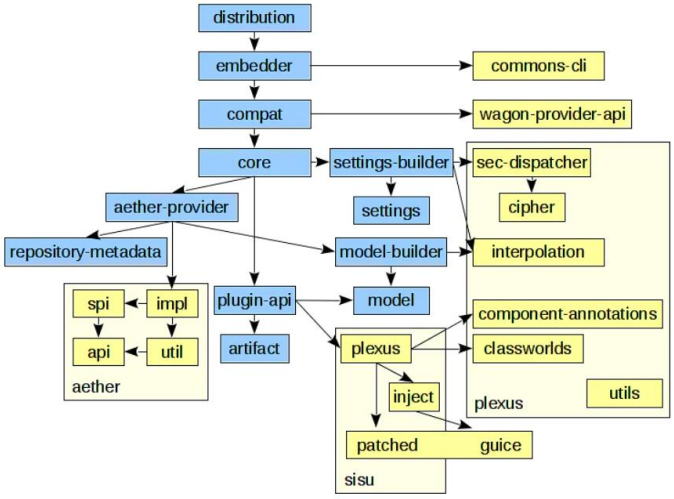
- Ένα ολοκληρωμένο, συντηρήσιμο, επαναχρησιμοποιήσιμο και απλό μοντέλο για έργα.
- Ένα σύνολο εργαλείων και προσθηκών που μπορούν να αλληλεπιδράσουν με το δηλωτικό μοντέλο.

Χαρακτηριστικά

Το Maven είναι γεμάτο με πολλά πολύτιμα και χρήσιμα χαρακτηριστικά, γεγονός που εξηγεί πολύ τη δημοτικότητά του. Εδώ είναι μερικά από τα πιο αξιοσημείωτα χαρακτηριστικά του Maven:

- Διαθέτει ένα τεράστιο, συνεχώς αναπτυσσόμενο αποθετήριο βιβλιοθηκών χρηστών
- Δυνατότητα δημιουργίας έργων εύκολα, χρησιμοποιώντας βέλτιστες πρακτικές
- Διαχείριση εξαρτήσεων, με αυτόματη ενημέρωση
- Συμβατό με προηγούμενες εκδόσεις
- Ισχυρή αναφορά σφαλμάτων και ακεραιότητας
- Αυτόματη γονική έκδοση
- Εξασφαλίζει συνεπή χρήση σε όλα τα έργα
- Είναι επεκτάσιμο και μπορούν εύκολα να γραφούν πρόσθετα χρησιμοποιώντας γλώσσες δέσμης ενεργειών ή Java.

Αρχιτεκτονική



Σχήμα 7.2: Αρχιτεκτονική Maven

Κατασκευή Εργαλείων

Τα εργαλεία δημιουργίας είναι τα εργαλεία ή τα προγράμματα που βοηθούν στη δημιουργία μιας εκτελέσιμης εφαρμογής από τον πηγαίο κώδικα. Όπως υποδηλώνει το όνομα, είναι απαραίτητο για τη δημιουργία ή τη δημιουργία σεναρίων μιας μεγάλης ποικιλίας εργασιών.

Το εργαλείο κατασκευής χρειάζεται για τις ακόλουθες διαδικασίες:

- Δημιουργία πηγαίου κώδικα
- Δημιουργία τεκμηρίωσης από τον πηγαίο κώδικα
- Μεταγλώττιση πηγαίου κώδικα
- Συσκευασία των μεταγλωττισμένων κωδικών σε αρχεία JAR
- Εγκατάσταση του συσκευασμένου κώδικα στο τοπικό αποθετήριο, διακομιστή ή κεντρικό αποθετήριο

Μοντέλο αντικειμένου έργου (POM)

Το Maven είναι τόσο χρήσιμο χάρη στο Project Object Model (POM), το οποίο είναι ένα αρχείο XML που έχει όλες τις πληροφορίες σχετικά με το έργο και τις λεπτομέρειες διαμόρφωσης. Το POM έχει την περιγραφή του έργου, λεπτομέρειες σχετικά με την έκδοση εκδόσεων και τη διαχείριση παραμέτρων του έργου.

Το αρχείο XML βρίσκεται στον αρχικό κατάλογο του έργου. Όταν εκτελείτε μια εργασία, ο Maven αναζητά το POM στον τρέχοντα κατάλογο.

Η ανάγκη για το Maven

Το Maven χρησιμοποιείται κυρίως για έργα που βασίζονται σε Java, βοηθώντας στη λήψη εξαρτήσεων, οι οποίες αναφέρονται σε βιβλιοθήκες ή αρχεία JAR. Το εργαλείο βοηθά στη λήψη των σωστών αρχείων JAR για κάθε έργο, καθώς ενδέχεται να υπάρχουν διαφορετικές εκδόσεις ξεχωριστών πακέτων.

Μετά το Maven, η λήψη εξαρτήσεων δεν απαιτεί επίσκεψη στους επίσημους ιστότοπους διαφορετικού λογισμικού. Επισκέπτοντας το mvnrepository, εντοπίζονται βιβλιοθήκες σε διαφορετικές γλώσσες. Το εργαλείο βοηθά επίσης στη δημιουργία της σωστής δομής έργου σε servlets κλπ., η οποία είναι απαραίτητη για την εκτέλεση.

Τρόπος Χρήσης του Maven

Όταν αρχίζει ένας χρήστης να χρησιμοποιεί το Maven, έχει ως γνώμονα αυτά τα τρία πράγματα:

1. Διαμορφώνει το Maven σε Java, χρησιμοποιώντας το μοντέλο αντικειμένου έργου (POM) που βρίσκεται σε ένα αρχείο pom.xml.
2. Όλες οι ρυθμίσεις διαμόρφωσης που σχετίζονται με το Maven βρίσκονται στο POM. Μπορεί να επεξεργαστεί και να διαμορφώσει πρόσθετα στην ετικέτα <plugins> ενός αρχείου pom.xml.
3. Το Maven παρέχει προεπιλεγμένες ρυθμίσεις για διαμορφώσεις, επομένως δεν χρειάζεται να προσθέσει κάθε διαμόρφωση στο αρχείο pom.xml.

Βήματα Διαδικασίας για την Κατασκευή ενός project

Ακολουθούν τα βήματα που πρέπει να ακολουθήσει ο χρήστης κατά την κατασκευή ενός έργου Maven:

- Προσθήκη του κώδικα για την δημιουργία της εφαρμογής και επεξεργασία του κώδικα στο χώρο αποθήκευσης του πηγαίου κώδικα.
- Επεξεργασία τυχόν απαραίτητων ρυθμίσεων παραμέτρων, όπως το αρχείο pom.XML ή οι λεπτομέρειες προσθήκης.
- Δημιουργία της πραγματικής εφαρμογής.
- Αποθήκευση εξόδου της διαδικασίας κατασκευής ως αρχείο WAR ή EAR σε έναν τοπικό διακομιστή ή σε άλλη τοποθεσία.
- Απόκτηση πρόσβασης στο αρχείο από την τοπική τοποθεσία ή διακομιστή και ανάπυτξή του στην τοποθεσία παραγωγής ή του πελάτη.
- Ενημέρωση εγγράφου της αίτησης αλλάζοντας την ημερομηνία και τον ενημερωμένο αριθμό έκδοσης της εφαρμογής, εάν είναι απαραίτητο.
- Δημιουργία μιας αναφοράς όπως ζητείται για την εφαρμογή ή την απαίτηση.

Πλεονεκτήματα

- Βοηθάει στη διαχείριση όλων των διαδικασιών, όπως η κατασκευή, η τεκμηρίωση, η κυκλοφορία και η διανομή στη διαχείριση έργων.
- Απλοποιεί τη διαδικασία κατασκευής έργου.
- Αυξάνει την απόδοση του έργου και τη διαδικασία κατασκευής.
- Η εργασία λήψης αρχείων JAR και άλλων εξαρτήσεων γίνεται αυτόματα.
- Παρέχει εύκολη πρόσβαση σε όλες τις απαιτούμενες πληροφορίες.
- Διευκολύνει τον προγραμματιστή να δημιουργήσει ένα έργο σε διαφορετικά περιβάλλοντα χωρίς να ανησυχεί για τις εξαρτήσεις ή/και τις διαδικασίες.
- Στο Maven, είναι εύκολο να προσθέσει νέες εξαρτήσεις γράφοντας τον κωδικό εξάρτησης στο αρχείο pom.

Μειονεκτήματα

- Το Maven απαιτεί εγκατάσταση στο σύστημα εργασίας και την προσθήκη Maven για το IDE.
- Εάν ο κωδικός Maven για μια υπάρχουσα εξάρτηση δεν είναι διαθέσιμος, δεν μπορεί να προσθέσει αυτήν την εξάρτηση χρησιμοποιώντας το ίδιο το Maven.
- Ορισμένες πηγές υποστηρίζουν ότι το Maven είναι αργό.

Εταιρείες που χρησιμοποιούν το Maven

Υπάρχουν περισσότερες από 2.000 εταιρείες που χρησιμοποιούν το Maven σήμερα, οι περισσότερες από τις οποίες βρίσκονται στις Ηνωμένες Πολιτείες και στον κλάδο της Επιστήμης Υπολογιστών. Το Maven χρησιμοποιείται επίσης σε βιομηχανίες εκτός της επιστήμης των υπολογιστών, όπως η τεχνολογία πληροφοριών, οι χρηματοοικονομικές υπηρεσίες, οι τράπεζες, τα νοσοκομεία και η περίθαλψη και πολλά άλλα.

Μερικές από τις μεγαλύτερες εταιρείες που χρησιμοποιούν τη Maven περιλαμβάνουν:

- Accenture
- JPMorgan Chase & Co
- Via Varejo
- Craft Base
- Red Hat
- Mitratech Holdings, Inc.
- KRG TECHNOLOGIES
- Radio – Καναδάς

Η Maven απολάμβανε μερίδιο αγοράς 26,88% το 2020, ξεπερνώντας σαφώς τον πλησιέστερο ανταγωνιστή της, την Resolve, η οποία έφτασε στο 19,78%. Σύμφωνα με την ίδια αναφορά, το Maven χρησιμοποιείται σε πάνω από 4.000 ιστότοπους, ενώ περίπου 3.000 ιστότοποι χρησιμοποιούν το Resolve. Άλλοι ανταγωνιστές περιλαμβάνουν τα Webpack, Gulp, Apache Ant και Gulp.

Ρύθμιση του project

Πρώτα θα χρειαστεί ο χρήστης να ρυθμίσει ένα project Java για να δημιουργήσει το Maven. Για να διατηρήσει την εστίαση στον Maven, κάνει το έργο όσο το δυνατόν πιο απλό προς το παρόν. Δημιουργεί αυτήν τη δομή σε έναν φάκελο project της επιλογής του.

Δημιουργία Δομής Καταλόγου

Σε έναν κατάλογο project της επιλογής του, δημιουργεί την ακόλουθη δομή υποκαταλόγου. για παράδειγμα, με το ***mkdir -p src/main/java/hello*** σε συστήματα *nix:

Πίνακας 7.1: Δομή Καταλόγου Maven

```
└── src
    └── main
        └── java
            └── hello
```

Μέσα στον κατάλογο ***src/main/java/hello***, μπορεί να δημιουργήσει όποιες κλάσεις Java θέλει.

Για να διατηρήσει τη συνέπεια με τη συνέχεια αυτού του οδηγού, δημιουργεί αυτές τις δύο κλάσεις:

HelloWorld.java και ***Greeter.java***.

Πίνακας 7.2: *src/main/java/hello/HelloWorld.java*

```
package hello;

public class HelloWorld {
  public static void main(String[] args) {
    Greeter greeter = new Greeter();
    System.out.println(greeter.sayHello());
  }
}
```

Πίνακας 7.3: *src/main/java/hello/Greeter.java*

```
package hello;

public class Greeter {
  public String sayHello() {
    return "Hello world!";
  }
}
```

Τώρα που υπάρχει ένα έργο που είναι έτοιμο να κατασκευαστεί με το Maven, το επόμενο βήμα είναι να εγκατασταθεί το Maven.

Το Maven μπορεί να γίνει λήψη ως αρχείο zip στη διεύθυνση https://maven.apache.org/download.cgi. Απαιτούνται μόνο τα δυαδικά αρχεία, επομένως αναζητήστε τον σύνδεσμο προς apache-maven-{version}-bin.zip ή apache-maven-{version}-bin.tar.gz.

Αφού γίνει λήψη του αρχείου zip, γίνεται αποσυμπίεση στον υπολογιστή και στη συνέχεια, προστίθεται ο φάκελος bin στη διαδρομή.

Για να γίνει δοκιμή της εγκατάστασης του Maven, εκτελείται το mvn από τη γραμμή εντολών:

Πίνακας 7.4: Δοκιμή Εγκατάστασης Maven

```
mvn -v
```

Εάν όλα πάνε καλά, θα πρέπει να παρουσιαστούν ορισμένες πληροφορίες σχετικά με την εγκατάσταση του Maven. Θα μοιάζει (αν και ίσως ελαφρώς διαφορετικό) με το ακόλουθο:

Πίνακας 7.5: Αποτελέσματα Δοκιμής Εγκατάστασης Maven

```
Apache Maven 3.3.9 (bb52d8502b132ec0a5a3f4c09453c07478323dc5; 2015-11-
10T16:41:47+00:00)
Maven home: /home/dsyer/Programs/apache-maven
Java version: 1.8.0_152, vendor: Azul Systems, Inc.
Java home: /home/dsyer/.sdkman/candidates/java/8u152-zulu/jre
Default locale: en_GB, platform encoding: UTF-8
OS name: "linux", version: "4.15.0-36-generic", arch: "amd64", family:
"unix"
```

Με αυτό τον τρόπο έγινε η εγκατάσταση του Maven.

Κεφάλαιο 8ο: Ο Κώδικας της Java

Πίνακας 8.1: Εισαγωγή Πακέτων και Βιβλιοθηκών

```
import com.github.javaparser.ParserConfiguration;
import com.github.javaparser.ast.CompilationUnit;
import com.github.javaparser.ast.body.MethodDeclaration;
import com.github.javaparser.ast.expr.MethodCallExpr;
import com.github.javaparser.ast.visitor.VoidVisitor;
import com.github.javaparser.ast.visitor.VoidVisitorAdapter;
import com.github.javaparser.symbolsolver.JavaSymbolSolver;
import
com.github.javaparser.symbolsolver.resolution.typesolvers.CombinedTypeSol
ver;
import
com.github.javaparser.symbolsolver.resolution.typesolvers.ReflectionTypeS
olver;
import
com.github.javaparser.symbolsolver.utils.SymbolSolverCollectionStrategy;
import com.github.javaparser.utils.ProjectRoot;
import com.github.javaparser.utils.SourceRoot;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.PrintWriter;
import static java.lang.Integer.parseInt;
import java.nio.file.Path;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import org.w3c.dom.Node;
import org.xml.sax.SAXException;
```

Πίνακας 8.2: Δημιουργία Εφαρμογής GUI με τη χρήση JFrame του Swing

```
public class NewJFrame extends javax.swing.JFrame {
. . .
}
```

Πίνακας 8.3: Δημιουργία αρχικών ArrayList

```
//ArrayList<String> soursepath = new ArrayList<String>();
    static ArrayList<String> soursepath = new ArrayList<String>();
    static ArrayList<String> methodmain = new ArrayList<String>();
    static ArrayList<String> arm = new ArrayList<String>();
    static ArrayList<String> methodcallgraph = new ArrayList<String>();
```

Πίνακας 8.4: Δήλωση Μεταβλητών του Project

```
static boolean bolmethee=true;
    static boolean bolmethee2=true;
    static double mainmethod=0;
    static boolean afalse=false;
    static boolean bfalse=false;
    static boolean whileloop=false;
```

Πίνακας 8.5: Δημιουργία υπολοίπων ArrayList

```
static ArrayList<String> namepac = new ArrayList<String>();
    static ArrayList<String> namepacparanomasths = new
ArrayList<String>();
    static ArrayList<String> nameinamain = new ArrayList<String>();
    static ArrayList<String> methodcallgraphcount = new
ArrayList<String>();
    static ArrayList<String> fullnamepacparanomasths = new
ArrayList<String>();
    static ArrayList<String> graphcounter = new ArrayList<String>();
    static int counterarray=0;
```

Πίνακας 8.6: Κλάση MethodVisitor για τις Κλήσεις Μεθόδων

```
//stis koloumenes methodous mas dinei ta method call expresion
    private static class MethodVisitor extends VoidVisitorAdapter<Void>
    {@Override
        public void visit(MethodCallExpr ml, Void arg) {
                whileloop=true;
                String mystring2=ml.getScope().toString();
                    String [] arr2 = mystring2.split("^new *");
                    int N=1;
                    String nWords2="";
                    for(int t=0; t<N ; t++){
                        nWords2 = nWords2 + arr2[t] + "."  ;
                    }
              if(nWords2.startsWith("Optional[new")){
                  String firstcharacter="";
                  firstcharacter=nWords2.substring(13);
                  firstcharacter = firstcharacter.replaceAll("\\(.*\\)",
"");
                  firstcharacter=("Optional["+firstcharacter);
                  firstcharacter =
firstcharacter.substring(0,firstcharacter.length()-1);
                  methodcallgraph.add(firstcharacter+"
"+ml.getName().toString());
              }

              String nWordsString=nWords2;
              nWordsString =
nWordsString.substring(0,nWordsString.length()-1);
              String pattern="\\.";
              Pattern p=Pattern.compile(pattern);
              Matcher m=p.matcher(nWordsString);
             if(m.find()){

nWordsString=nWordsString.substring(0,nWordsString.indexOf("."));
                  nWordsString=nWordsString+"]";
                  methodcallgraph.add(nWordsString+"
"+ml.getName().toString());
```

```
                }
                else{
                    methodcallgraph.add(ml.traverseScope()+"
"+ml.getName().toString());
                }
                methodcallgraph=removeDuplicates(methodcallgraph);
            }
        }
```

Πίνακας 8.7: Αφαίρεση Διπλότυπων Τιμών από το ArrayList

```
//afairoume diplotupa apo array list autos o tropos mporei na alaxtei
    public static <T> ArrayList<T> removeDuplicates(ArrayList<T> list)
    {
        ArrayList<T> newList = new ArrayList<T>();
        for (T element : list) {

            if (!newList.contains(element)) {
                newList.add(element);
            }
        }
        return newList;
    }
```

Πίνακας 8.8: Κλάση MethodVisitor2 για την συνέχεια Κλήσεων Μεθόδων

```
private static class MethodVisitor2 extends VoidVisitorAdapter<Void>{
        @Override
        public void visit(MethodCallExpr n, Void arg) {

        super.visit(n, arg);
        try{
            String mystring2=n.getScope().toString();
                    String [] arr2 = mystring2.split("^new *");
                    int N=1;
                    String nWords2="";
                    for(int t=0; t<N ; t++){
                        nWords2 = nWords2 + arr2[t] + "."   ;
                    }
              if(nWords2.startsWith("Optional[new")){
                    String firstcharacter="";
                    firstcharacter=nWords2.substring(13);
                    firstcharacter = firstcharacter.replaceAll("\\(.*\\)",
"");
                    firstcharacter=("Optional["+firstcharacter);
                    firstcharacter =
firstcharacter.substring(0,firstcharacter.length()-1);
                    arm.add(firstcharacter+" "+n.getName().toString());
              }
              String nWordsString=nWords2;
              nWordsString =
nWordsString.substring(0,nWordsString.length()-1);
              String pattern="\\.";
            Pattern p=Pattern.compile(pattern);
            Matcher m=p.matcher(nWordsString);
            if(m.find()){

nWordsString=nWordsString.substring(0,nWordsString.indexOf("."));
                  nWordsString=nWordsString+"]";
                  arm.add(nWordsString+" "+n.getName().toString());
```

```
                }
                else{
                    arm.add(n.traverseScope()+" "+n.getName().toString());
                }
            }
            catch (Exception e) {
            e.printStackTrace();
            return;}
            }
    }
```

Πίνακας 8.9: Κλάση MethodDeclaration για την Εύρεση Ορισμών Μεθόδων

```
public static String[] myBadGlobalArray = new String[10];
        //statik metablites stous metrites gia methodous kai bibliothikes
         public  static int i=0,i2=0;

        private static class MethodNamePrinter2 extends
VoidVisitorAdapter<Void> {
        @Override
        public void visit(MethodDeclaration mc, Void arg) {
        super.visit(mc, arg);
            String if1,if2;

            for( int i=0; i<methodcallgraph.size(); i++){
                if1=methodcallgraph.get(i).toLowerCase();

if2=("Optional["+mc.resolve().getClassName().toLowerCase()+"]
"+mc.resolve().getName()).toLowerCase();

                if(if1.equals(if2)){

                    String mystring2=mc.resolve().getPackageName();
                    String [] arr2 = mystring2.split("\\.");
                    int N=3;
                    String nWords2="";
                    for(int n=0; n<N ; n++){
                        nWords2 = nWords2 + arr2[n] + "."  ;
                    }

                    boolean newflag=true;

                    for(int counteri=0;
counteri<methodcallgraphcount.size(); counteri++){
                        if(
methodcallgraphcount.get(counteri).equals(mc.resolve().getName())){
                            newflag=false;
                        }

                    }

                    if(newflag)
                        {
                            for(int counti=0; counti<namepac.size();
counti++){
                                if(namepac.get(counti).equals(nWords2)){
                                    String
graphcounterString=graphcounter.get(counti);
                                    System.out.println("Method in Name
Printed: "+mc.resolve().getQualifiedName());
```

```
                                        int
graphcounterInt=parseInt(graphcounterString);
                                        graphcounterInt++;

graphcounter.set(counti,String.valueOf(graphcounterInt));
                                        }
                                    }
                                }
                    methodcallgraphcount.add(mc.resolve().getName());

methodcallgraphcount=removeDuplicates(methodcallgraphcount);
                    VoidVisitor<Void> methodNameVisitor4 = new
MethodVisitor();
                    methodNameVisitor4.visit(mc, null);
                }
            }
            counterarray=methodcallgraph.size()-1;
            }
        }
```

Πίνακας 8.10: Μέτρηση Μεθόδων και Βιβλιοθηκών

```
//metrame to plithos toon methodon kai toon biblhothikon
        private static class MethodNamePrinter extends
VoidVisitorAdapter<Void> {
        @Override
        public void visit(MethodDeclaration md, Void arg) {
        super.visit(md, arg);
        i2++;

        String mystring=md.resolve().getPackageName();
            String [] arr = mystring.split("\\.");
            int N=3;
            String nWords="";
            for(int i=0; i<N ; i++){
                nWords = nWords + arr[i] + "."  ;
            }
            if(bolmethee){
                namepac.add(nWords);
                namepacparanomasths.add("0");
                nameinamain.add("0");
                fullnamepacparanomasths.add("0");
                graphcounter.add("0");
                bolmethee=false;
                }
            int countermain=0;
            int kametr=0,counter;
            for(counter=0; counter<namepac.size(); counter++){
                if(namepac.get(counter).equals(nWords)){
                    String na=fullnamepacparanomasths.get(counter);
                    countermain=counter;
                    int nana=parseInt(na);
                    nana++;

fullnamepacparanomasths.set(counter,String.valueOf(nana));
                    }
                    if(!namepac.get(counter).equals(nWords)){
                        kametr++;
                }
          }
```

```
                    if(counter==kametr){
                        namepac.add(nWords);
                        namepacparanomasths.add("0");
                        nameinamain.add("0");
                        fullnamepacparanomasths.add("0");
                        graphcounter.add("0");
                    }
            if(md.isPublic()){
                i++;

                for(counter=0; counter<namepac.size(); counter++){
                    if(namepac.get(counter).equals(nWords)){
                        String
namepacparanomasthscounter=namepacparanomasths.get(counter);
                        countermain=counter;
                        int
namepacparanomasthscounterParse=parseInt(namepacparanomasthscounter);
                        namepacparanomasthscounterParse++;

namepacparanomasths.set(counter,String.valueOf(namepacparanomasthscounter
Parse));
                    }
                }
                    for(int i=0; i<methodmain.size(); i++){
                        String if1,if2;
                        if1=methodmain.get(i).toLowerCase();

if2="Optional["+md.resolve().getClassName().toLowerCase()+"]
"+md.resolve().getName().toLowerCase();

if(if1.toLowerCase().equals(if2.toLowerCase())){
                                    System.out.println("Method Name Printed:
"+md.resolve().getQualifiedName());
                                    String
nameinamainString=nameinamain.get(countermain);
                                    int
nameinamainInt=parseInt(nameinamainString);
                                    nameinamainInt++;
                                    VoidVisitor<Void> methodNameVisitor4 =
new MethodVisitor();
                                    methodNameVisitor4.visit(md, null);
                                    if(whileloop){
                                        CombinedTypeSolver combinedTypeSolver
= new CombinedTypeSolver();
                                        combinedTypeSolver.add(new
ReflectionTypeSolver());
                                        ParserConfiguration
parserConfiguration = new ParserConfiguration().setSymbolResolver(new
JavaSymbolSolver(combinedTypeSolver));
                                        ProjectRoot projectRoot = new
SymbolSolverCollectionStrategy(parserConfiguration).collect(Path.of(".\\t
arget\\lib\\sources"));
                                        for (SourceRoot sourceRoot :
projectRoot.getSourceRoots()) {
                                            try {
                                            sourceRoot.tryToParse();
                                            List<CompilationUnit>
compilationUnits = sourceRoot.getCompilationUnits();
                                            for(int x=0;
x<compilationUnits.size();x++){
```

```
                                                    VoidVisitor<Void>
methodNameVisitor = new MethodNamePrinter2();

methodNameVisitor.visit(compilationUnits.get(x), null);

                                                }
                                                } catch (IOException e) {
                                                    e.printStackTrace();
                                                    return;
                                                }
                                            }
                                        }

nameinamain.set(countermain,String.valueOf(nameinamainInt));
                                mainmethod=mainmethod+1;
                                System.out.println(mainmethod);
                                }
                        }

                    }
            }
     }
```

Πίνακας 8.11: Εμφάνιση Πλήθους Μεθόδων

```
//emafanizoume to plhthos ton methodon
        public static int count(){
            System.out.println("Number of methods: "+i);
            return i;

        }
```

Πίνακας 8.12: Μεθοδολογία Εκτέλεσης Εντολών Κονσόλας

```
//einai methodos gia na trexoume cmd entoles
    public static void CMD(String cmd)
    {
        String[] command =
        {
            "cmd",
        };
        Process p;
        try {
            p = Runtime.getRuntime().exec(command);
                new Thread(new SyncPipe(p.getErrorStream(),
System.err)).start();
                    new Thread(new SyncPipe(p.getInputStream(),
System.out)).start();
                    PrintWriter stdin = new
PrintWriter(p.getOutputStream());
                        stdin.println(cmd);
                    stdin.close();
                    p.waitFor();
            } catch (Exception e) {
            e.printStackTrace();
    }
    }
```

Πίνακας 8.13: Πρόσβαση σε Μονοπάτια Καταλόγων

```
//pairnoyme ta path pou theloume
    private static final String FILE_PATH4=".\\target\\lib\\sources";

    public NewJFrame() {
        initComponents();
    }
```

Πίνακας 8.14: Έναρξη Συστατικών της GUI Εφαρμογής

```
/**
     * This method is called from within the constructor to initialize
the form.
     * WARNING: Do NOT modify this code. The content of this method is
always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-
BEGIN:initComponents
    private void initComponents() {

        RunButton = new javax.swing.JButton();
        InputTextField = new javax.swing.JTextField();
        jScrollPane1 = new javax.swing.JScrollPane();
        ExportjTextArea = new javax.swing.JTextArea();
        jLabel1 = new javax.swing.JLabel();


setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

        RunButton.setText("Run");
        RunButton.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                RunButtonActionPerformed(evt);
            }
        });

        ExportjTextArea.setColumns(20);
        ExportjTextArea.setRows(5);
        jScrollPane1.setViewportView(ExportjTextArea);

        jLabel1.setFont(new java.awt.Font("Tahoma", 0, 24)); // NOI18N
        jLabel1.setText("Enter Path");

        javax.swing.GroupLayout layout = new
javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LE
ADING)
                    .addGroup(layout.createSequentialGroup()
                        .addGap(63, 63, 63)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LE
ADING, false)
```

```
                                .addComponent(jScrollPane1,
javax.swing.GroupLayout.DEFAULT_SIZE, 692, Short.MAX_VALUE)
                                .addComponent(InputTextField)
                                .addGroup(layout.createSequentialGroup()
                                    .addGap(284, 284, 284)
                                    .addComponent(RunButton,
javax.swing.GroupLayout.PREFERRED_SIZE, 100,
javax.swing.GroupLayout.PREFERRED_SIZE))))
                        .addGroup(layout.createSequentialGroup()
                            .addGap(337, 337, 337)
                            .addComponent(jLabel1)))
                    .addContainerGap(45, Short.MAX_VALUE))
            );
            layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addGroup(layout.createSequentialGroup()
                    .addGap(31, 31, 31)
                    .addComponent(jLabel1)
                    .addGap(18, 18, 18)
                    .addComponent(InputTextField,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addGap(31, 31, 31)
                    .addComponent(RunButton,
javax.swing.GroupLayout.PREFERRED_SIZE, 40,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                    .addComponent(jScrollPane1,
javax.swing.GroupLayout.DEFAULT_SIZE, 306, Short.MAX_VALUE)
                    .addContainerGap())
            );

            pack();
        }// </editor-fold>//GEN-END:initComponents
```

Πίνακας 8.15: Εκτέλεση Ενέργειας κατά το πάτημα Κουμπιού

```
private void RunButtonActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_RunButtonActionPerformed
            // TODO add your handling code here:
            ExportjTextArea.append("Wait");
            String FILE_PATH3;
            FILE_PATH3 = InputTextField.getText();

            //Symbolsover gia th main tou project
            CombinedTypeSolver combinedTypeSolver2 = new
CombinedTypeSolver();
            combinedTypeSolver2.add(new ReflectionTypeSolver());
            ParserConfiguration parserConfiguration2 = new
ParserConfiguration().setSymbolResolver(new
JavaSymbolSolver(combinedTypeSolver2));
            ProjectRoot projectRoot2 = new
SymbolSolverCollectionStrategy(parserConfiguration2).collect(Path.of(FILE
_PATH3+"\\src\\main"));
            for (SourceRoot sourceRoot : projectRoot2.getSourceRoots()) {
        try {
            sourceRoot.tryToParse();
```

```
        List<CompilationUnit> compilationUnits =
sourceRoot.getCompilationUnits();
        for(int x=0; x<compilationUnits.size();x++){
            VoidVisitor<Void> methodNameVisitor2 = new MethodVisitor2();
                methodNameVisitor2.visit(compilationUnits.get(x), null);
        }
    } catch (IOException e) {
        e.printStackTrace();
        return;}}
        methodmain=removeDuplicates(arm);
        //maven entoles gia extract gia jar kai sourses
        String CWD = System.getProperty("user.dir");
        String cmd_Command="cd " + FILE_PATH3 +" & mvn dependency:copy-
dependencies -DexcludeTransitive -DoutputDirectory="+CWD+"\\target\\lib";
        CMD(cmd_Command);
        String cmd2 ="dir /a:-d /s /b "+CWD + "\\target\\lib"+" | find /c
\":\" ";
        CMD(cmd2);
        String cd_Command="cd " + FILE_PATH3+"& mvn dependency:copy-
dependencies -Dclassifier=sources  -DexcludeTransitive -
DoutputDirectory="+CWD+"\\target\\lib\\sources";
        CMD(cd_Command);
        File folder = new File(".\\target\\lib\\sources");
        File[] listOfFiles = folder.listFiles();
        for (int i = 0; i < listOfFiles.length; i++) {
            String cmd3="cd "+FILE_PATH4 +" & jar xf
"+listOfFiles[i].getName();;
            CMD(cmd3);
        }
        //Symbolsover gia ta jar arxeia
        CombinedTypeSolver combinedTypeSolver = new CombinedTypeSolver();
        //combinedTypeSolver.add(new JavaParserTypeSolver(new
File("C:\\Users\\mario\\Desktop\\mavenperser\\target\\lib\\sources")));
        combinedTypeSolver.add(new ReflectionTypeSolver());
        ParserConfiguration parserConfiguration = new
ParserConfiguration().setSymbolResolver(new
JavaSymbolSolver(combinedTypeSolver));
        ProjectRoot projectRoot = new
SymbolSolverCollectionStrategy(parserConfiguration).collect(Path.of(".\\t
arget\\lib\\sources"));
        for (SourceRoot sourceRoot : projectRoot.getSourceRoots()) {
    try {
        sourceRoot.tryToParse();
        List<CompilationUnit> compilationUnits =
sourceRoot.getCompilationUnits();
        //System.out.println(compilationUnits);
        for(int x=0; x<compilationUnits.size();x++){

            VoidVisitor<Void> methodNameVisitor = new
MethodNamePrinter();
                methodNameVisitor.visit(compilationUnits.get(x), null);

        }
    } catch (IOException e) {
        e.printStackTrace();
        return;
    }

        System.out.println("oi  methodoi pou brethikan "+mainmethod+"
SYNOLIKES METHODOI ");
```

```
        ExportjTextArea.setText("Oi methodoi poy xrhsimopoiounte ston
kodika einai "+mainmethod+"\n");
        int a=count();
        ExportjTextArea.append("oi synolikes methodoi twn jar arxeion
"+a+"\n");
        double b=(mainmethod/a)*100;
        System.out.println("Apotelesma "+b+" %");
        ExportjTextArea.append("Apotelesma "+b+"%"+"\n");
        System.out.println("-----------------------------------------
---");
        System.out.println(namepac);

        System.out.println(namepacparanomasths);
        System.out.println(nameinamain);
        double sum;

        for(int i=0; i<nameinamain.size(); i++){

sum=(Double.parseDouble(nameinamain.get(i))/Double.parseDouble(namepacpar
anomasths.get(i)))*100;
            System.out.println(namepac.get(i)+"  "+sum+" %");
            ExportjTextArea.append(namepac.get(i)+"  "+sum+" %"+"\n");
        }

        System.out.println("-----------------------------------------
---");
        ExportjTextArea.append("-------------------------------------
"+"\n");
        ExportjTextArea.append("oi diplotipes methodoi einai
"+methodcallgraphcount.size()+"\n");
        System.out.println("oi diplotipes methodoi einai
"+methodcallgraphcount.size()+"\n");
        System.out.println(fullnamepacparanomasths);
        System.out.println(graphcounter);
        for(int i=0; i<fullnamepacparanomasths.size(); i++){
           if(!graphcounter.get(i).equals("0"))
                {
                String graphcounterString=graphcounter.get(i);
                    int graphcounterInt=parseInt(graphcounterString);
                    graphcounterInt++;
                    graphcounter.set(i,String.valueOf(graphcounterInt));
                    }


sum=(Double.parseDouble(graphcounter.get(i))/Double.parseDouble(fullnamep
acparanomasths.get(i)))*100;
            System.out.println(namepac.get(i)+"  "+sum+" %");
            ExportjTextArea.append(namepac.get(i)+"  "+sum+" %"+"\n");
            System.out.println();
        }
      }
    }//GEN-LAST:event_RunButtonActionPerformed
```

Πίνακας 8.16: Κύρια Συνάρτηση Main του Προγράμματος

```
/**
     * @param args the command line arguments
     */
    public static void main(String args[]) throws FileNotFoundException,
IOException, SAXException, ParserConfigurationException {
        /* Set the Nimbus look and feel */
        //<editor-fold defaultstate="collapsed" desc=" Look and feel
setting code (optional) ">
        /* If Nimbus (introduced in Java SE 6) is not available, stay
with the default look and feel.
         * For details see
http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
         */
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {

javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
        } catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(NewJFrame.class.getName()).log(java.ut
il.logging.Level.SEVERE, null, ex);
        } catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(NewJFrame.class.getName()).log(java.ut
il.logging.Level.SEVERE, null, ex);
        } catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(NewJFrame.class.getName()).log(java.ut
il.logging.Level.SEVERE, null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(NewJFrame.class.getName()).log(java.ut
il.logging.Level.SEVERE, null, ex);
        }
        //</editor-fold>

        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new NewJFrame().setVisible(true);
            }
        });
    }
```

Πίνακας 8.17: Δήλωση Ιδιωτικών Συστατικών της GUI Εφαρμογής

```
// Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.JTextArea ExportjTextArea;
    private javax.swing.JTextField InputTextField;
    private javax.swing.JButton RunButton;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JScrollPane jScrollPane1;
    // End of variables declaration//GEN-END:variables
```

Κεφάλαιο 9ο: Συμπεράσματα

Τα συμπεράσματα που προκύπτουν αφορούν τον τελικό στόχο του project με τον αναλυτή JavaParser και την βιβλιοθήκη JavaSymbolSolver και με βάση αυτά, προκύπτει πως οι προγραμματιστές και οι χρήστες που χρησιμοποιούν τα παραπάνω εργαλεία ανάλυσης, μπορούν πραγματικά να επωφεληθούν από την χρήση του αναλυτή JavaParser και να κάνουν διεργασίες με αυτοματοποιημένο τρόπο.

Οι αυτοματοποιημένες διεργασίες παρέχουν αρκετή αξία για τον χρήστη που θέλει να αναλύσει το project του πέρα από την βασική μεταγλώττιση που του παρέχεται από το γραφικό περιβάλλον συγγραφής κώδικα. Όλα τα στοιχεία που έχει κωδικοποιήσει μπορούν να βρεθούν σε ένα δέντρο ξεκινώντας από την ρίζα του και με ενδιάμεσες διαδρομές (κλαδιά) να φτάσει στο τερματικό επίπεδο (φύλλα) κατά το οποίο τερματίζει το πρόγραμμά τους.

Ο κώδικας του παραπάνω project στο Κεφάλαιο 8, συντελεί στην ολοκλήρωση των τριών μετρικών της παρούσας εργασίας και γίνεται η εκτύπωση των αποτελεσμάτων όλων των μετρικών, η οποία θα προβληθεί εξ ολοκλήρου, στην παρουσίαση της εργασίας.

Κεφάλαιο 10ο: Προτάσεις Βελτίωσης

Στην παρούσα εργασία, υπάρχουν ορισμένα σημεία στα οποία θα μπορούσαν να προσαρμοστούν ορισμένες βελτιώσεις και επεκτάσεις. Αυτές οι βελτιώσεις αφορούν κάποια διαφορετικά μέρη του συνολικού έργου στο κομμάτι της σχεδίασης, της δομής και της λειτουργικότητας του έργου.

Συγκεκριμένα, στο κομμάτι της σχεδίασης, έγινε πρόταση βελτίωσης της εμφάνισης του παραθύρου (Γραφικό Περιβάλλον Χρήστη) που εμφανίζεται κατά την εκτέλεση του προγράμματος. Με τα έως τώρα δεδομενα, το γραφικό παράθυρο διαθέτει την λειτουργικότητα των τριών μετρικών έως ένα σημείο, χωρίς να έχει δοθεί περισσότερη έμφαση στο κομμάτι του Design. Επίσης, έγινε δοκιμή δημιουργίας εκτελέσιμου (executable) αρχείου σε σύγκριση με την μη αυτοματοποιημένη (manual) εκτέλεση του project. Με αυτό τον τρόπο, θα γινόταν πιο άμεση η διαδικασία εκτέλεσης αλλά και γραφικά θα αποτελούσε μια πιο άρτια εικόνα.

Στο κομμάτι της δομής του project, αυτό που μας ενδιαφέρει είναι η δομή του κώδικα, η οποία βασίζεται σε ένα ενιαίο αρχείο .java, έναντι των περισσότερων αρχείων .java με την κατάλληλη σύνδεση και διευθυνσιοδότηση. Αυτό οφείλεται στο γεγονός πως το αντικείμενο του JavaParser αποτελείται από ένα αρκετά μικρό κοινό προγραμματιστών και αναλυτών, συνεπώς εντοπίζεται έλλειψη πληροφοριών και πηγών γύρω από το θέμα και αυτό συνεπάγεται την παραγωγή σύνθετου και περίπλοκου κώδικα για τις λειτουργίες του JavaParser. Η τελική απόφαση, ήταν να χρησιμοποιηθεί ένα ενιαίο αρχείο .java με όλα τα βήματα και την ανάπτυξη του project, έτσι ώστε να γίνονται συνεχείς έλεγχοι για την ορθή εκτέλεση του προγράμματος και να γίνουν οι κατάλληλες διορθώσεις στο ίδιο αρχείο. Αυτό δυστυχώς, καθιστά τον πηγαίο κώδικα λιγότερο ευανάγνωστο και μερικά σημεία του γίνονται λιγότερο ευδιάκριτα.

Στο κομμάτι της λειτουργικότητας του project, θα μπορούσε να χτιστεί καλύτερα η μεθοδολογία των MethodCallExpressions που χρησιμοποιούνται στην 2η και την 3η μετρική της εργασίας. Συγκεκριμένα, θα μπορούσε να διορθωθεί η λειτουργικότητα της GetQualifiedName που βρίσκεται στην MethodCallExpression, έτσι ώστε να προκύπτουν ορθά αποτελέσματα με μεγαλύτερο όγκο μεθόδων και βιβλιοθηκών JAR.

ΒΙΒΛΙΟΓΡΑΦΙΑ

Ο τρόπος γραφής των βιβλιογραφικών αναφορών γίνεται σύμφωνα με τα παρακάτω παραδείγματα (IEEE style):

Βιβλία

[1] W. K. Chen, *Linear Networks and Systems*. Belmont, CA: WadsworthPress, 2003.

[2] J. L. Spudich and B. H. Satir, *Sensory Receptors and Signal Transduction*. New York: Wiley-Liss, 2001.

Application Note

[3] Hewlett-Packard, Appl. Note 935, pp.25-29.

Πατέντες

[4] K. Kimura and A. Lipeles, "Fuzzy controller component," U. S. Patent 14, 860,040, 14 Dec., 2006.

Data Sheet

[5] Texas Instruments, "High speed CMOS logic analog multiplexers/demultiplexers," 74HC4051 datasheet, Nov. 1997.

Internet Site

[6] European Telecommunications Standards Institute, "Digital Video Broadcasting (DVB): Implementation guide for DVB terrestrial services; transmission aspects," *European Telecommunications Standards Institute*, ETSI-TR-101, 2007. [Online]. Available: http://www.etsi.org.

Paper in Conference Proceedings

[7] J. Smith, R. Jones, and K. Trello, "Adaptive filtering in data communications with self-improved error reference," In Proc. IEEE International Conference on Wireless Communications '04, 2004, pp. 65-68.

[8] H. A. Nimr, "Defuzzification of the outputs of fuzzy controllers," presented at 5th International Conference on Fuzzy Systems, Cairo, Egypt, 2006.

Journal Articles

[9] K. A. Nelson, R. J. Davis, D. R. Lutz, and W. Smith, “Optical generation of tunable ultrasonic waves,” *Journal of Applied Physics*, vol. 53, no. 2, pp. 1144-1149, Feb. 2002.

Online Sources

[10] https://www.w3schools.com/java/

[11] https://www.guru99.com/java-swing-gui.html

[12] https://medium.com/@ridhij/getting-started-with-javaparser-what-you-can-and-what-you-shouldnt-do-b943f4cd5122

[13] https://maven.apache.org/guides/getting-started/maven-in-five-minutes.html

[14] https://www.javatpoint.com/git

[15] https://www.edureka.co/blog/how-to-use-github/

[16] https://medium.com/getting-started-with-javaparser/getting-started-with-javaparser-what-you-can-and-what-you-shouldnt-do-21b64eb6305c

[17] https://www.javaadvent.com/2017/12/javaparser-generate-analyze-modify-java-code.html

[18] https://www.livingtech.com.cn/media/javaparservisited.pdf

[19] https://github.com/javaparser/javasymbolsolver

[20] https://www.livingtech.com.cn/media/javaparservisited.pdf

[21] https://tomassetti.me/resolve-method-calls-using-java-symbol-solver/

[22] https://www.twilio.com/blog/abstract-syntax-trees

[23] https://www.simplilearn.com/tutorials/maven-tutorial/what-is-maven

[24] https://spring.io/guides/gs/maven/