Μεθοδολογία Προγραμματισμού

Κλάσεις και αντικείμενα

Νικόλαος Πεταλίδης

Τμήμα Μηχανικών Πληροφορικής και Επικοινωνιών
Διεθνές Πανεπιστήμιο της Ελλάδος

Εισαγωγή
Εαρινό Εξάμηνο

Κλάσεις

- Μια κλάση (ή τάξη) είναι μια συλλογή, μια ομάδα, ένα σύνολο ή ένας τύπος
- Παράδειγμα
  - Άνθρωπος (Η κλάση όλων των ανθρώπων)
  - Φοιτητής (Η κλάση όλων των φοιτητών)
- Ένα αντικείμενο είναι ένα στιγμιότυπο μιας κλάσης, ένα συγκεκριμένο μέλος της κλάσης
- Παράδειγμα
  - ο Άνθρωπος «Νίκος Παπαδόπουλος»
  - ο Φοιτητής «Κώστας Παπαδόπουλος»

Πώς ανακαλύπτουμε τις κλάσεις

- Κοιτάξτε την περιγραφή του προβλήματος που σας έχουν δώσει.
- Υπογραμμίστε ό,τι ουσιαστικά εμφανίζονται
- Υπογραμμίστε ό,τι οργανισμούς και ρόλους εμφανίζονται

Παράδειγμα

- Ποιες κλάσεις αναγνωρίζεται στην παρακάτω περιγραφή;
- Φτιάξτε ένα σύστημα που να το χρησιμοποιούν οι μαθητές για να αξιολογούν τα βιβλία τους. Οι προϊστάμενοι του ΤΕΙ ή οι γραμματείς τους θα μπορούν να εισάγουν νέα βιβλία για αξιολόγηση και να βλέπουν τις αξιολογήσεις που έχουν γίνει. Οι εκδότες θα πρέπει να μπορούν να εισέρχονται στο σύστημα και να βλέπουν τις αξιολογήσεις που έχουν γίνει. Τέλος ένας ειδικός χρήστης θα μπορεί να δημιουργεί νέους χρήστες (πχ προϊσταμένους, εκδότες κτλ).

Βρείτε τις κλάσεις που αναφέρονται στις ακόλουθες περιγραφές

Κατασκευάζετε ένα μηχάνημα ελέγχου πιστωτικών καρτών. Το μηχάνημα επιτρέπει σε χρήστες να εισάγουν τα στοιχεία της πιστωτικής τους κάρτας (αριθμό, τύπο και PIN) μαζί με το ποσό της συναλλαγής που θέλουν να κάνουν. Για λόγους ασφαλείας επίσης για κάθε συναλλαγή καταγράφεται και η ταυτότητα του μηχανήματος που την πραγματοποίησε, δηλαδή ο σειριακός αριθμός του μηχανήματος, και η ταχυδρομική διεύθυνση στην οποία βρίσκεται το μηχάνημα.

Εξάσκηση

Σας ζητούν να σχεδιάσετε ένα σύστημα διαχείρισης πελατών. Το σύστημα θα το χρησιμοποιούν όλοι οι πωλητές ενός γραφείου προκειμένου να δημιουργήσουν νέους πελάτες, ή να επεξεργαστούν παλιούς. Για κάθε πελάτη αποθηκεύονται τουλάχιστον τα ακόλουθα στοιχεία (όνομα, email, τηλέφωνο). Σε κάθε πελάτη αντιστοιχεί και ένας πωλητής ο οποίος είναι και υπεύθυνος για αυτόν τον πελάτη. Κάθε πωλητής μπορεί να εισέλθει στο σύστημα δίνοντας τα στοιχεία του (username, password) και να δει τα στοιχεία των πελατών του ή να δει τα ραντεβού (ώρα, ημέρα, τόπος) που έχει κλείσει με τους πελάτες του ή να φτιάξει νέα.
Ένας ειδικός χρήστης (διευθυντής) μπορεί να δημιουργήσει νέους πωλητές.

Προσοχή

- Πολλοί κάνουν το λάθος και προσπαθούν να ανακαλύψουν ενέργειες όταν διαβάζουν μια περιγραφή ενός προβλήματος
- Αυτός ήταν ο τρόπος δημιουργίας προγραμμάτων στο διαδικασιακό προγραμματισμό (procedural programming)

Παράδειγμα

- Ένας μαθητής εγγράφεται σε ένα μάθημα, ή μπορεί να διαγραφεί από αυτό
- Άσχημη λύση
  - Υπάρχει η κλάση μαθητής, η κλάση εγγραφή και η κλάση διαγραφή
  - Αυτή η προσέγγιση είναι λάθος!
  - Έτσι θα σκεφτόσασταν αν φτιάχνατε συναρτήσεις στη C αλλά όχι στον αντικειμενοστραφή προγραμματισμό!
- Σωστή λύση
  - Υπάρχει η κλάση μαθητής και η κλάση μάθημα

Χαρακτηριστικά

- Κάθε κλάση έχει κάποια χαρακτηριστικά ή ιδιότητες
- Για παράδειγμα μια κλάση «Τραπεζικός Λογαριασμός» θα έχει σα χαρακτηριστικό ένα «Αριθμό Λογαριασμού»
- Για παράδειγμα η «Πιστωτική κάρτα» έχει ως χαρακτηριστικά (αριθμό, τύπο και PIN)

Αντικείμενα

- Μια κλάση είναι ένα σύνολο από αντικείμενα τα οποία έχουν τα ίδια χαρακτηριστικά
- Ένα αντικείμενο είναι ένα συγκεκριμένο μέλος της κλάσης με συγκεκριμένα χαρακτηριστικά.
- Η Πιστωτική Κάρτα με Αριθμό 3424, Τύπο VISA και PIN 343 είναι ένα αντικείμενο της κλάσης Πιστωτική Κάρτα

Λειτουργίες

- Συνήθως τα αντικείμενα μιας κλάσης μπορούν να εκτελέσουν και κάποιες λειτουργίες.
- τα αντικείμενα της κλάσης Πιστωτική Κάρτα μπορούν να εκτελέσουν τη λειτουργία «Χρέωση» ή «Συναλλαγή»

Πως βρίσκουμε τις λειτουργίες;

- Ψάξτε στην περιγραφή του προβλήματος σας για ρήματα.
- Αυτά συνήθως δηλώνουν ενέργειες που μπορούν να γίνουν σε μια κλάση ή λειτουργίες που ξέρει να διεκπεραιώνει.

Παράδειγμα

Φτιάξτε ένα σύστημα που να το χρησιμοποιούν οι μαθητές για να αξιολογούν τα βιβλία τους. Οι προϊστάμενοι του ΤΕΙ ή οι γραμματείς τους θα μπορούν να εισάγουν νέα βιβλία για αξιολόγηση και να βλέπουν τις αξιολογήσεις που έχουν γίνει. Οι εκδότες θα πρέπει να μπορούν να εισέρχονται στο σύστημα και να βλέπουν τις αξιολογήσεις που έχουν γίνει. Τέλος ένας ειδικός χρήστης θα μπορεί να δημιουργεί νέους χρήστες (πχ προϊσταμένους, εκδότες κτλ).

Μια καλή προγραμματιστική πρακτική

- Για κάθε χαρακτηριστικό μιας κλάσης ορίστε και αντίστοιχες μεθόδους set και get. Για παράδειγμα η κλάση "Πιστωτική Κάρτα» περιέχει τα χαρακτηριστικά "PIN", «Σειριακός Αριθμός» και «Τύπος». Ορίστε λειτουργίες getSerial(), setSerial(), getType(), setType(), getPin(), setPin()

Συσχετίσεις

- Μια κλάση μπορεί να συσχετίζεται με μια άλλη
- Στο παράδειγμα με τους μαθητές και τα βιβλία, ένας μαθητής μπορεί να συσχετιστεί με τα βιβλία που έχει αξιολογήσει

Συσχετίσεις

- Μια συσχέτιση απλά τονίζει συνήθως ότι μια κλάση έχει ως χαρακτηριστικό, κάτι που από μόνο του είναι και αυτό μια κλάση.
- Για παράδειγμα ένας πελάτης έχει μια διεύθυνση. Και ο «πελάτης» είναι κλάση και η διεύθυνση είναι «κλάση». Επειδή η κλάση πελάτης περιέχει σα χαρακτηριστικό την κλάση «Διεύθυνση» μπορούμε να πούμε ότι η κλάση πελάτης σχετίζεται με την κλάση διεύθυνση

Κληρονομικότητα

- Κλάσεις που έχουν κοινά στοιχεία μεταξύ τους μας πολύ συχνά μας βοηθούν να μειώσουμε τη δουλειά που έχουμε να κάνουμε
- Η κληρονομικότητα λέμε ότι συνήθως εκφράζει σχέσεις τύπου "είναι".
- Ένα τραπέζι είναι έπιπλο. Άρα η κλάση «τραπέζι» λέμε ότι κληρονομεί από την κλάση «έπιπλο»

Η στρουθοκάμηλος

- Η στρουθοκάμηλος είναι πουλί.
- Όλα τα πουλιά πετούν.
- Άρα η στρουθοκάμηλος πετάει; Που είναι το πρόβλημα;

Κληρονομικότητα

- Η κληρονομικότητα είναι βασική έννοια του αντικειμενοστραφούς προγραμματισμού
- Πάντα όταν ψάχνετε για κλάσεις κοιτάτε να δείτε αν υπάρχουν σχέσεις κληρονομικότητας
- Τι σχέσεις κληρονομικότητας βρήκατε στις προηγούμενες περιγραφές;

[Κλάσεις]

Δηλώσεις κλάσεων

- Η δήλωση μιας κλάσης στη Java έχει την ακόλουθη μορφή:

```
class MyClass {
  //fields
  //constructor
  //method
  }
```

- Είναι πιθανόν επίσης να ορίσεις τροποποιητές (modifiers) όπως public, protected κτλ. Για παράδειγμα:

```
public class MyClass {
  //fields
  //constructor
  //method
  }
```

Πεδία (fields) σε μια κλάση

- Είναι δυνατόν όπως και στη C++ μια κλάση να έχει και δεδομένα όπως στο ακόλουθο παράδειγμα:

```
public class Circle {
        private double x, y;   // centre coordinate
        private double r;      //  radius of the circ

}
```

Μέθοδοι σε μια κλάση

- Είναι δυνατόν όπως και στη C++ μια κλάση να έχει και συναρτήσεις (μεθόδους) όπως στο ακόλουθο παράδειγμα:

```
public class Circle {
   private double x, y; // centre
   private double r;    // radius
   Circle(double x, double y, double r)
   {
      this.x = x;
      this.y = y;
      this.r = r;
    }
   public double circumference() {
^^I^^Ireturn 2*3.14*r;
   }
}
```

Μέθοδοι ή συναρτήσεις

- Οι συναρτήσεις στη Java επιτρέπουν όλες τον επανα-ορισμό τους, και είναι ισοδύναμες με συναρτήσεις που έχουν οριστεί με το keyword `virtual` στη C++
- Για το λόγο αυτό αναφέρονται ως *μέθοδοι* και όχι συναρτήσεις

Υπερφόρτωση μεθόδων

- Είναι δυνατόν όπως και στη C++ μια κλάση να έχει και συναρτήσεις (μεθόδους) με το ίδιο ονομα που διαφέρουν στο είδος και/ή στον αριθμό των ορισμάτων
- Σε αντίθεση με τη C++ δεν είναι δυνατή η υπερφόρτωση τελεστών

Η λέξη κλειδί this

- Η λέξη κλειδί `this` χρησιμοποιείται για να αναφερθούμε στο ίδιο το αντικείμενο που είμαστε
- `this.x = x` δίνει την τιμή της παραμέτρου `x` στο πεδίο `x` του αντικειμένου
- Χρησιμοποιείται όταν το όνομα μιας παραμέτρου είναι ίδιο με το όνομα του πεδίου για να μπορέσουμε να τα διαφοροποιήσουμε

Constructors

- Όπως και στη C++ μπορείτε να ορίσετε δομητές (constructors) σε μια κλάση

```
public class Circle {
   private double x, y; // centre
   private double r;    // radius
   Circle(double x, double y, double r)
   {
      this.x = x;
      this.y = y;
      this.r = r;
    }
   public double circumference() {
^^I^^Ireturn 2*3.14*r;
   }
}
```

Πολλοί constructors

- Όπως και στη C++ έτσι μια κλάση μπορεί να έχει πολλούς υπερφορτωμένους (overloaded) constructors, οι οποίοι διαφέρουν στο είδος και/ή στον αριθμό των ορισμάτων

```
public class Circle {
  private double x, y; // centre
  private double r;    // radius
  Circle(double x, double y, double r) {
    this.x = x;
    this.y = y;
    this.r = r;
   }
  Circle(double x, double y) {
    this.x = x;
    this.y = y;
    this.r = 0;
   }
```

Κλήση constructors

- Όπως και στη C++ έτσι μια κλάση μπορεί να καλέσει έναν constructor από έναν άλλον

```
public class Circle {
   private double x, y; // centre
   private double r;    // radius
   Circle(double x, double y, double r)
   {
      this.x = x;
      this.y = y;
      this.r = r;
    }
   Circle(double x, double y)
   {
      this(x,y,0);
   }
}
```

Δηλώσεις και δημιουργία αντικειμένων

- Είναι δυνατόν να δηλώσετε μεταβλητές μιας κλάσης όπως και στη C++

```
Circle aCircle;
```

- Η αρχική τιμή αυτών των μεταβλητών είναι `null`
- Για τη δημιουργία αντικειμένων μπορείτε να χρησιμοποιήσετε τον τελεστή `new`

```
Circle aCircle = new Circle(1.0,2.0,3.0);
```

Πρόσβαση σε στοιχεία μιας κλάσης

- Γίνεται όπως και στη C++ χρησιμοποιώντας τον τελεστή `.`.

```
Circle aCircle = new Circle();
double circumference = aCircle.circumference();
```

- Ο τελεστής `->` της C++ δε χρησιμοποιείται

Κλήση με τιμή

- Στη Java ουσιαστικά οι κλήσεις των συναρτήσεων είναι με τιμή

```
public class TestClass {
   public void swap(int a, int b) {
      int tmp = a;
      a = b;
      b = tmp;
   }
}
 public static void main(String[] args) {
    TestClass testClass = new TestClass();
    int a = 1;  int b = 2;
    testClass.swap(a,b);
    System.out.println(a); //prints 1
    System.out.println(b); // prints 2
}
```

Κλήση με αναφορά

- Τα πεδία των αντικειμένων όμως καλούνται με αναφορά

```
public class TestClass {
   public int a;
   TestClass(int a) { this.a = a; }
}
static void swap(TestClass one, TestClass two) {
   int tmp = one.a;
   one.a = two.a;   two.a = tmp;
}
 public static void main(String[] args) {
    TestClass testClass1 = new TestClass(1);
    TestClass testClass2 = new TestClass(2);
    swap(testClass1,testClass2);
    System.out.println(testClass1.a);//prints 2
    System.out.println(testClass2.a); //prints 1
 }
```

Κληρονομικότητα

- Για να δηλώσω πως μία κλάση είναι υποκλάση μιας άλλης χρησιμοποιώ τη λέξη κλειδί extends στον ορισμό της

```
public class Shape {
    ...
}
public class Circle extends Shape {
    ...
}
```

Method overriding - Υπέρβαση μεθόδων

- Κάθε φορά που καλείται η μέθοδος ενός αντικειμένου η java αναζητά τη μέθοδο στην τρέχουσα κλάση και
- αν δεν βρεθεί την αναζητά στην υπερκλάση.
- Σε κάποιες περιπτώσεις θέλουμε να διαφοροποιηθεί η συμπεριφορά μιας μεθόδου στην τρέχουσα κλάση από αυτή της υπερκλάσης. Σε αυτή την περίπτωση έχουμε υπέρβαση μεθόδου (overriding)

Method overriding - Παράδειγμα

```
public class Shape {
   int getArea() {   return 0; }
}
public class Circle extends Shape {
   ...
   @Override
   int getArea() {
      return pi*r*r;
   }
}
...
Shape shape = new Shape();
shape.area(); //Returns 0
Circle circle = new Circle();
circle.area(); //Returns pi*r*r
```

Ερώτηση

- Με βάση τις προηγούμενες κλάσεις, τι θα επιστρέψει το ακόλουθο:

```
Shape aShape = new Circle();
aShape.area(); // Returns ???
```

Κλήση αρχικών μεθόδων

- Μπορούμε να καλέσουμε από μια μέθοδο υποκλάσης την μέθοδο της υπερκλάσης που κάνουμε override χρησιμοποιώντας τη λέξη super.

```
public class TwoCircles extends Circle {
    ...
    @Override
    int getArea() {
        double result = super.getArea();
        return 2*result;
    }
```

Κλήση constructor υπερκλάσης

- Μπορούμε να καλέσουμε αντίστοιχα και έναν constructor της υπερκλάσης χρησιμοποιώντας πάλι τη λέξη super.
- Η κλήση της υπερκλάσης πρέπει να είναι η πρώτη εντολή στον constructor

```
public class Parallelogram extends Shape {
  Square(int x, int y) {
    this.height = x;
    this.width = y;
  }
public class Square extends Parallelogram {
  Square(int x) {
    super(x,x);
  }
}
```

Εσωτερικές κλάσεις

- Μια εμφωλευμένη (nested) ή εσωτερική κλάση είναι μια κλάση που ορίζεται μέσα σε μια άλλη
- Έχουν πρόσβαση στις ιδιότητες της εξωτερικής κλάσης αυτόματα
- Μπορούν να κρυφθούν από άλλες κλάσεις
- Χρησιμοποιούνται για να ορίσουμε κλάσεις που δεν θα επαναχρησιμοποιηθούν ή οι οποίες αφορούν μόνο συγκεκριμένο αντικείμενο

Είδη εσωτερικών κλάσεων

- Εσωτερικές (inner)

  Τοπικές Ανώνυμες ή με όνομα

  Μη-τοπικές Μόνο με όνομα

- Στατικές κλάσεις, οι οποίες μπορεί να είναι μη-τοπικές και με όνομα

Μη τοπικές κλάσεις

- Μια κλάση χωρίς τον ορισμό static
- Ορίζεται εκτός μεθόδων
- Μπορεί να είναι private, public, κτλ όπως όλα τα μέλη μιας κλάσης
- Έχει πρόσβαση σε όλα τα πεδία της εξωτερικής κλάσης

Παράδειγμα μη τοπικής κλάσης

```
class Outer{
   private int x1;
   Outer(int x1) {
       this.x1 = x1;
   }
   public void foo(){ System.out.println"("fooing);}
   public class Inner{
     private int x1 = 0;
     void foo() {
      System.out.println("Outer value of x1:"
                                        + Outer.this.x1);
      System.out.println("Inner value of x1:"
                                            + this.x1);
     }
}
```

Παράδειγμα μη τοπικής κλάσης (συνέχεια)

```
public class TestDrive{
 public static void main(String[] args){
// can create in regular way
   Outer outer = new Outer();
   Inner inner =
              outer.new Inner(); \
      //must call new through outer handle
   inner.foo();
   // note that this can only be done
 //if inner is visible
   }
}
```

Που χρησιμοποιούνται

- Συνήθως όταν η εξωτερική κλάση θέλει μια καλύτερη ομαδοποίηση των χαρακτηριστικών της
- Δημιουργούνται συνήθως μέσα στην εξωτερική κλάση (και όχι μέσω του new που είδαμε στο παράδειγμα)

Τοπικές κλάσεις

- Ορίζονται μέσα σε μεθόδους
- Ονομάζονται τοπικές εσωτερικές κλάσεις (local innner clasess)
- Οι κλάσεις αυτές είναι ορατές μόνο στη μέθοδο που ορίζονται
- Δε μπορούν να εχουν πρόσβαση σε τοπικες μεταβλητές (εκτός αν ορίστηκαν final)
- Δε χρειάζεται να έχουν όνομα (anonymous)

Παράδειγμα

```
button.addActionListener(
        new ActionListener(){
            public void actionPerformed(actionEvent ae){
                //do work here
          }
        }
);
```

Λάθη και εξαιρέσεις

Όταν συμβαίνει ένα λάθος στο πρόγραμμα μας υπάρχουν τρεις τρόποι για να το αντιμετωπίσουμε

- Να επιστρέψουμε μια τιμή λάθους
- Να ορίσουμε μια καθολική μεταβλητή λάθους
- Να δημιουργήσουμε μια *εξαίρεση*

Παραδοσιακός χειρισμός λαθών

- Δίνουν στον προγραμματιστή την ευθύνη, αλλά όχι την υποχρέωση, για να γράψει κώδικα που να ελέγχει μετά την κλήση μιας συνάρτησης την επιστρεφόμενη τιμή η την καθολική μεταβλητή λάθους.
- Σε πολλές περιπτώσεις δεν υπάρχει «λογική» τιμή που να επιστρέφεται και να υποδηλώνει λάθος, με αποτέλεσμα να επιλέγεται αυθαίρετα μία τιμή για αυτό το λόγο
- Ο κώδικας που χειρίζεται τα λάθη είναι αναμεμιγμένος με τον κώδικα που χειρίζεται τη φυσιολογική ροή του προγράμματος

Χειρισμός λαθών με εξαιρέσεις

- Δίνουν στον προγραμματιστή την υποχρέωση να γράψει κώδικα που να χειρίζεται το λάθος. Αν δεν το κάνει, τότε σε περίπτωση λάθους το πρόγραμμα θα σταματήσει να εκτελείται τη στιγμή που δημιουργήθηκε το λάθος.
- Μια εξαίρεση είναι μια οντότητα που δε μπορεί να παρανοηθεί σα φυσιολογική τιμή που θα επέστρεφε μια μέθοδος
- Ο κώδικας που χειρίζεται τα λάθη είναι χωριστός από τον κώδικα που χειρίζεται τη φυσιολογική ροή του προγράμματος

Τι είναι μια εξαίρεση (exception)

- Είναι ουσιαστικό ένα αντικείμενο που δε διαφέρει από τα υπόλοιπα
- Μια μέθοδος μπορεί να δημιουργήσει μια εξαίρεση και να την «ρίξει» (throw) όταν βρει ένα λάθος

Throwing an exception

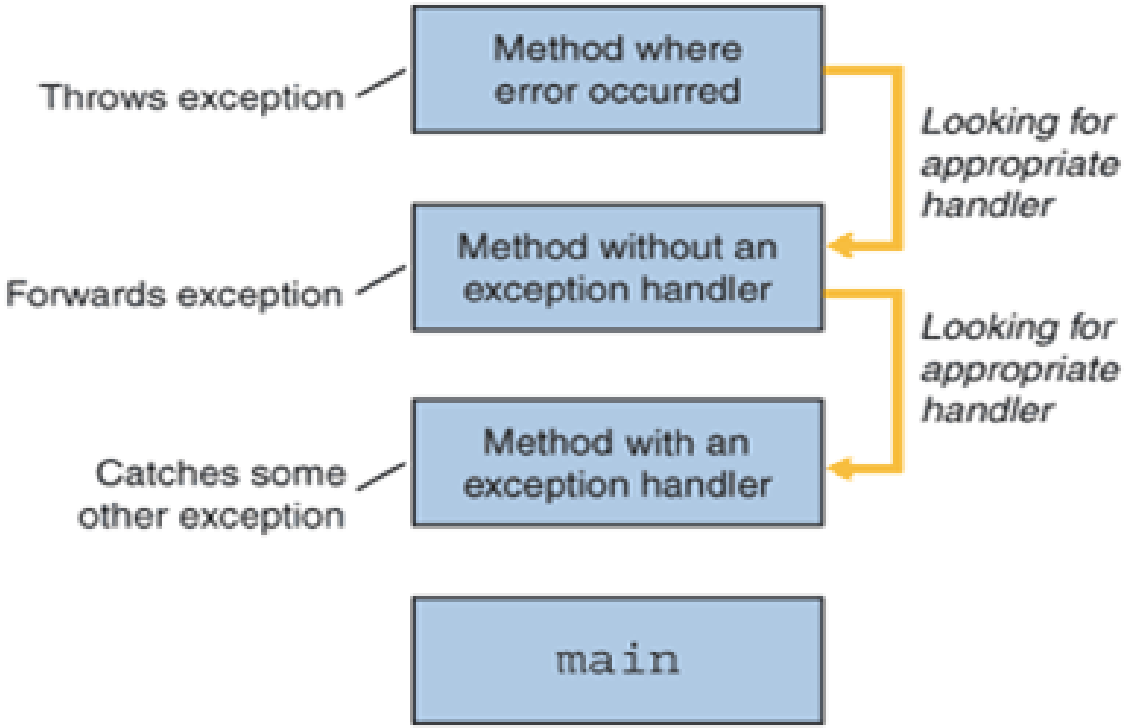
```
public Object pop() throws EmptyStackException {
  Object obj;

  if (size == 0) {
    throw new EmptyStackException();
    }

    ...
```

Χειρισμός εξαιρέσεων

- Το σύστημα ελέγχει τη στοίβα κλήσης συναρτήσεων με σκοπό να βρει κώδικα που να μπορεί να χειριστεί την εξαίρεση που προέκυψε



Χειρισμός εξαιρέσεων - Παράδειγμα

```
public void someMethod() {
   try {
       stack.pop();
   } catch (EmptyStackException e)
   {
      //Χειρισμός εξαίρεσηςεδώ
   }
 }
```

Χειρισμός εξαιρέσεων - finally

- Πολλές φορές θέλουμε να ορίσουμε ότι ένα κομμάτι κώδικα πρέπει να εκτελεστεί είτε δημιουργηθεί exception, είτε όχι

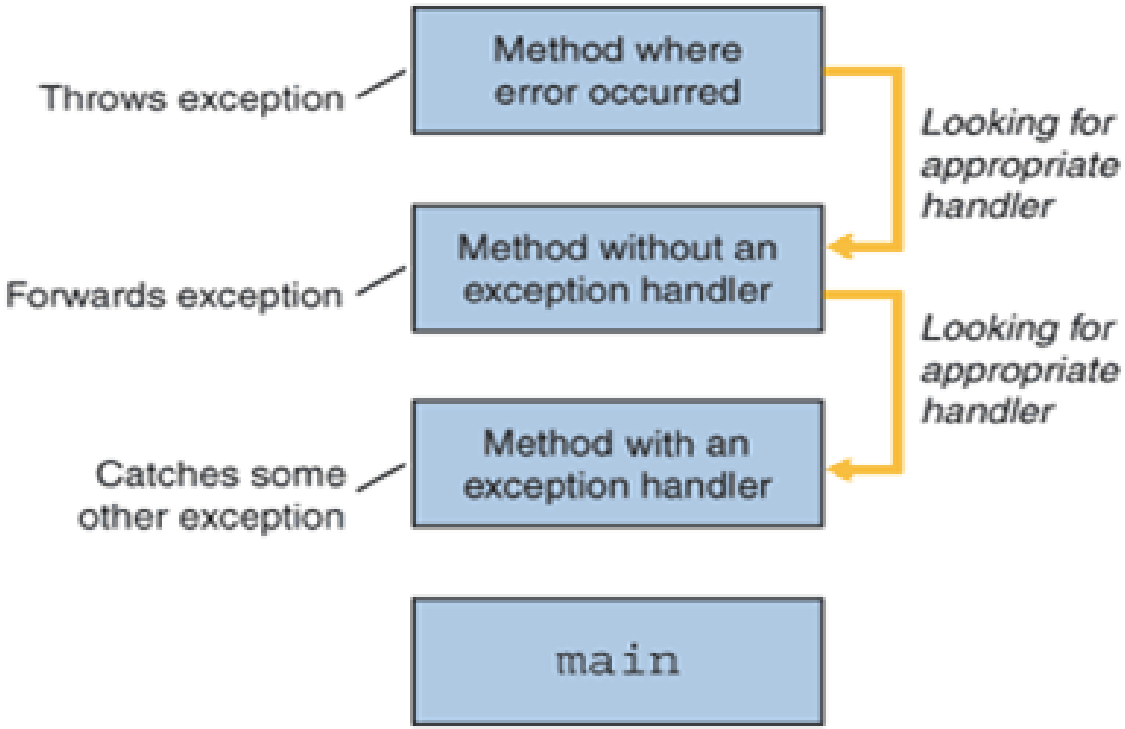
```
public void someMethod() {
    try {
        stack.pop();
    } catch (EmptyStackException e)
    {
        //Χειρισμός εξαίρεσηςεδώ
    }
^^lfinally
    {
^^l// Κώδικαςπουθαεκτελεστείπάντοτε
    }
}
```

Χειρισμός εξαιρέσεων - try-with-resources

- Ένα resource (πόρος) είναι ουσιαστικά ένα αντικείμενο που πρέπει να κλείσουμε μετά τη χρήση του. Μπορούμε, αντί να γράφουμε ειδικό κώδικα στο finally να το ορίσουμε συνοπτικά

```
static String readFirstLineFromFile(String path)
    throws IOException {

try (BufferedReader br =
        new BufferedReader(new FileReader(path))) {
    return br.readLine();
  }
}
```


Method where error occurred
Throws exception
Looking for appropriate handler
Method without an exception handler
Forwards exception
Looking for appropriate handler
Method with an exception handler
Catches some other exception
main

Χειρισμός εξαιρέσεων - Παράδειγμα

```
public void someMethod() {
   try {
       stack.pop();
   } catch (EmptyStackException e)
   {
      //Χειρισμός εξαίρεσηςεδώ
   }
 }
```

Είδη εξαιρέσεων

Checked Exceptions Εξαιρετικές περιπτώσεις όπου ένα καλογραμμένο πρόγραμμα πρέπει να προβλέψει και να ανακάμψει από αυτές (Παράδειγμα: Αδυναμία σύνδεσης στη βάση)

Errors Εξαιρετικές περιπτώσεις όπου ένα καλογραμμένο πρόγραμμα δε μπορεί να προβλέψει (Για παράδειγμα πρόβλημα με το hardware)

Runtime Exceptions Εξαιρετικές περιπτώσεις που δημιουργούνται από την εφαρμογή λόγω πιθανού bug και το πρόγραμμα δε μπορεί ούτε να προβλέψει ούτε να ανακάμψει από αυτές (π.χ. διαίρεση με το 0)

Ορισμός εξαιρέσεων

- Μέθοδοι οι οποίες δημιουργούν Checked Exceptions πρέπει να το ορίζουν ξεκάθαρα στο interface τους

```
public Object pop() throws EmptyStackException {
        ...
}
```