Discord: @maroun#8023

JAVA

a humble

DOCUMENTATION

// Σημειώσεις

// για θεωρία και παραγωγή κώδικα

Αντικειμενοστραφής λογική

Κλάση (class)

Ονομάζουμε ένα μπλοκ κώδικα που λειτουργεί ως μια γενική κατηγορία για να παράγει πιο ειδικές/ συγκεκριμένα πράγματα (αντικείμενα). Για παράδειγμα, μια κλάση «Αυτοκίνητο», μπορεί να δημιουργήσει αντικείμενα «Audi», «BMW», κλπ. Κάθε κλάση δίνει το γενικό πρότυπο, δηλαδή παρέχει αρχικές τιμές, μεταβλητές και συναρτήσεις που καθορίζουν τη συμπεριφοράς των αντικειμένων.

➔ *Παράδειγμα*

```
public class Car {

}
```

Ιδιότητες (attributes)

Τα χαρακτηριστικά μιας κλάσεις, δηλαδή κάποιο στοιχείο που την απαρτίζει και συνθέτει την εικόνα της οντότητας (πχ για μια κλάση αυτοκίνητο, θα μπορούσε να έχει χαρακτηριστικά διάμετρος ρόδας, ταχύτητα, χρώμα περιβλήματος). Κάθε ιδιότητα έχει συγκεκριμένη εμβέλεια, που ορίζει ποιος μπορεί να τη γνωρίζει (δηλαδή να διαβάζει ή να αλλάζει τη τιμή της). Η διάκριση δεν είναι ακριβώς όπως οι περισσότερες γλώσσες δομημένου προγραμματισμού (local, global, static), αλλά τις προσομοιώνει με κάποιες παραπάνω λεπτομέρειες (που δημιουργούν υποκατηγορίες).

➔ *Παράδειγμα*

```
public class Car {

   protected double wheel_diameter;
   private int speed;
   public String color;

}
```

Εμβέλεια μεταβλητών και συναρτήσεων

- ***public***

  Δηλώνει ότι η μεταβλητή (ή συνάρτηση) μπορεί να χρησιμοποιηθεί από οποιαδήποτε άλλη κλάση μέσω ενός αντικειμένου αυτής (πχ αν το car1 είναι αντικείμενο φτιαγμένο βάση το πρότυπο της κλάσης Car με ιδιότητα color (public), τότε καλώ αυτή την ιδιότητα ως car1.color)

  → *Παράδειγμα*

```
// Δημιουργία αντικειμένου Car με τη λέξη "new"
Car car1 = new Car(50.3, 110, "Red");

String whatColorIsTheCar = car1.color;
System.out.println("I like " + whatColorIsTheCar);
```

- ***private***

  Δηλώνει ότι η μεταβλητή (ή συνάρτηση) είναι προσπελάσιμη μόνο από την ίδια τη κλάση (τον κατασκευαστή και τις μεθόδους) στην οποία δηλώνεται. Αν κάποια άλλη κλάση προσπαθήσει να χρησιμοποιήσει αυτούσια τη τιμή της, θα εμφανιστεί μήνυμα λάθους. Για να το αποτρέψουμε αυτό, φτιάχνουμε έξτρα συναρτήσεις (getters) που μας επιτρέπουν «νόμιμα», δηλαδή με τον έγκριση και πλήρη γνώση της κλάσης, τις τιμές των ιδιοτήτων της.

  → *Παράδειγμα*

```
// Δημιουργία αντικειμένου Car με τη λέξη "new"
Car car1 = new Car(50.3, 110, "Red");

// public: Χρησιμοποιούν τη τιμή απευθείας, χωρίς να προκαλούν runtime error

// private: Χρειάζεται getter για να τη χρησιμοποιήσουν κι άλλα (εξωτερικά) τμήματα κώδικα (δλδ άλλες κλάσεις)

int whatSpeedDoesTheCarHave; // private speed
whatSpeedDoesTheCarHave = car1.speed; → ERROR
```

- ***protected***

  Δηλώνει ότι η μεταβλητή (ή συνάρτηση) είναι private για όλες τις κλάσεις ΕΚΤΟΣ από τις κλάσεις-παιδιά της που θα μπορούν να τη χρησιμοποιούν χωρίς πρόβλημα. Για παράδειγμα, βάση το πρότυπο (κλάση) ενός αυτοκινήτου Car, θα μπορούσε να δημιουργηθεί και μια άλλη υπό-κατηγορία, η κλάση ToyCar η οποία θα κληρονομεί τη διάμετρο του αυτοκινήτου από τη Car.

  ΣΗΜΕΙΩΣΗ*: Για τη κατανόηση των protected μεταβλητών, πρέπει να έχει γίνει πρώτα κατανοητή η έννοια της Κληρονομικότητας (ανατρέχουμε στην ανάλογη υπό – ενότητα).*

→ *Παράδειγμα*

```
public class Main {
	public static void main(String[] args) {
		Car car1 = new Car(50.3, 110, "Red");
		ToyCar smallC = new ToyCar(4, 3, "Blue", "LEGO");

		String whatColorIsTheCar = car1.color;
		System.out.println("Color:" + whatColorIsTheCar);
		System.out.println("Color:" + smallC.color);

	}
}


public class ToyCar extends Car {
	private String toyBrand;

	public ToyCar(double wheel_diameter, int speed, String
	color, String toyBrand ) {

		super(wheel_diameter, speed, color);
		this.toyBrand = toyBrand;
	}
}
```

- ***static***

Η μεταβλητή (ή συνάρτηση) ΔΕΝ είναι χαρακτηριστικό του δοθέν αντικειμένου, αλλά της ίδιας της κλάσης (πχ κάποιος counter), όλα της τα αντικείμενα βλέπουν ίδια

→ *Παράδειγμα*

```
public class Product {
   private double price;  // separate attribute for each obj.
   public static total_products = 0;  // counts all these obj.
}
```

- ***final***

Σταθερά, δε μπορείς να αλλάξεις τιμή (πχ καλό για να σταθεροποιήσεις το όνομα ενός αρχείου)

→ *Παράδειγμα*

```
final int minutes_per_hour = 60;
```

Μέθοδος (method)

Συνάρτηση η οποία ορίζεται στο σώμα μιας κλάσης, ώστε να κάνει λειτουργικά τα αντικείμενα

➔ *Παράδειγμα*

```
public class Car {
    private int speed;

    protected void increaseSpeedBy(int increaseNum) {
        speed = speed + increaseNum;

    }

}
```

Κατασκευαστής (constructor)

Ειδική ΑΠΑΡΑΙΤΗΤΗ μέθοδος αρχικοποίησης των πεδίων ενός αντικειμένου. Υπάρχουν σε μια κλάση, και είναι υποχρεωτικό να υπάρχει τουλάχιστον ένας από αυτούς για την ομαλή λειτουργία του προγράμματος. Αυτές οι συναρτήσεις δεν έχουν τύπο (ούτε καν void) κι έχουν ακριβώς ίδιο όνομα με το όνομα της κλάσης. Λόγω πολυμορφισμού, μπορούμε να έχουμε πολλούς κατασκευαστές με διαφορετικές παραμέτρους αρχικοποίησης.

➔ *Παράδειγμα*

```
public class Student {

    private String name;
    private String id;

    public Student(String aName, String someId) {
        name = aName;
        id = someId;
    }
}
```

this

Δεσμευμένη λέξη αυτό-αναφοράς σε αντικείμενο (εκπροσωπεί το εκάστοτε αντικείμενο που κάλεσε κάτι από τη κλάση). Χρήσιμο για όταν η παράμετρος έχει ίδιο όνομα την ιδιότητα.

➔ *Παράδειγμα*

```
    public Student(String name, String id) {
        this.name = name;
        this.id = id;
    }
```

Setters

Μέθοδοι που (ανά)θέτουν τιμές σε ιδιότητες. Προαιρετική η χρήση τους, όχι τόσο συχνή, εκτός κι αν η ανάθεση τιμής απαιτεί μεγάλο τμήμα κώδικα ή σύνθετες διαδικασίες.

➔ *Παράδειγμα*

```
public class Student {
	private String name;
	private String id;

	void setName(String name) {
		this.name = name;
	}

	void setPrice (double price) {
		this.price = price;
	}
}
```

Getters

Μέθοδοι που επιστρέφουν στη καλούσα συνάρτηση τη τιμή μιας ιδιότητας, Προαιρετική, αλλά χρήσιμη, συνήθως υπάρχει για όλες (ή τις περισσότερες) ιδιότητες μιας κλάσης.

➔ *Παράδειγμα*

```
public class Student {

	private String name;
	private String id;

	String getName() {
		return name;
	}

	double getPrice() {
		return price;
	}
}
```

Πολυμορφισμός (polymorphism)

Δυνατότητα να εκτελούμε μια ενιαία ενέργεια με διαφορετικούς τρόπους, να ορίζουμε μια διεπαφή και να έχουμε πολλαπλές υλοποιήσεις. Πρακτικά, αυτό σημαίνει ότι μπορούμε να γράψουμε μια συνάρτηση (πχ initialize), και να την ορίσουμε πολλαπλές φορές για διαφορετικούς τύπους δεδομένων διατηρώντας το ίδιο όνομα (πχ initialize (int age), initialize (String name, int age) , initialize ()) – αυτό λέγεται **υπερφόρτωση συναρτήσεων**, και γίνεται με αλλαγή του αριθμού των παραμέτρων ή/και αλλαγή του τύπου των ορίων.

➔ *Παράδειγμα*

```
public class Student {

	private String name;
	private String id;

	// ΠΟΛΥΜΟΡΦΙΣΜΟΣ σε κατασκευαστές
	public Student(String aName, String someId) {
		name = aName;
		id = someId;
	}

	public Student(String aName) {
		name = aName;
		id = "Not defined yet";
	}

	public Student() {
		name = "Not defined yet";
		id = "Not defined yet";
	}


	// ΠΟΛΥΜΟΡΦΙΣΜΟΣ σε απλές μεθόδους
	// ακριβώς ίδια λογική
	protected void increaseSpeed(int increaseNum) {
		speed = speed + increaseNum;
	}

	protected void increaseSpeed() {
		speed = speed + 1;
	}

}
```

Ενθυλάκωση (encapsulation)

Δυνατότητα κλάσης να ενσωματώνει στο σώμα της ιδιότητες χωρίς οι εσωτερικές λειτουργίες να απασχολούν το υπόλοιποι (εξωτερικό) πρόγραμμα. Υπάρχει για να προστατεύει τη κλάση και να τηρεί κανόνες/ περιορισμούς που να εμφανίζονται σε άλλες μεθόδους (πχ να μην εγγράφονται άλλοι μαθητές σε μάθημα αν ο συνολικός αριθμός τους ξεπεράσει τους 30).

➔ *Παράδειγμα*

```
public class Main {

    public static void main(String[] args) {
        // δίνω σαν όρισμα κάτι (ή και τίποτα, κάποιες φορές), πχ
        τώρα ένα αρχείο, κι αφήνω το κατασκευαστή να κάνει ότι
        αρχικοποιήσεις χρειάζονται
        PoliceDataBase dataBase = new PoliceDataBase
                                  ("file_with_info.txt");

        // δε ξέρουμε πως (δε μπορούμε να δούμε το κώδικα) αλλά
        αυτή η μέθοδος, με κάποιο τρόπο μας επιστρέφει το μέσο
        βάρος των φυλακισμένων
        double avgWeightOfPrisoners;
        avgWeightOfPrisoners = dataBase.calculateAverageWeight();

        // ομοίως, δε μας αφορά το πως, αλλά αυτή η μέθοδος τυπώνει
        στην οθόνη όσους κρατούμενους προέρχονται από την Ελλάδα
        dataBase.printAllSuspectsFromOneCountry("Greece");

        // εδώ επιστρέφεται στο κώδικα μας μια (στατική) ιδιότητα
        μιας κλάσης
        // Καταλαβαίνουμε ότι είναι ιδιότητα κι όχι μέθοδος, γιατί
        δεν έχει τις παρενθέσεις πριν το ερωτηματικό
        int number_of_suspects = dataBase.total_suspects;
    }

}
```

Κληρονομικότητα (Inheritance)

Ικανότητα μιας κλάσης να υιοθετεί τα attributes και τις methods μιας άλλης κλάσης (την οποία αποκαλούμε parent class) χωρίς την ανάγκη copy – paste όλων εκείνων των στοιχείων. Καλό άμα θέλουμε να δημιουργήσουμε μια γενική κατηγορία (πχ κλάση όχημα, με ιδιότητες ταχύτητα, πλήθος_από_ρόδες) και να φτιάξουμε μετά άλλες κλάσεις που είναι υποκατηγορίες αυτής (πχ ποδήλατο – με ΕΞΤΡΑ ιδιότητες πλήθος_από_κουδουνάκια - , και κλάση παιδί αυτοκίνητο – με ΕΞΤΡΑ ιδιότητες είδος_καυσίμου, πρόσφυση – , και κλάση αεροπλάνου – με ΕΞΤΡΑ ιδιότητες).

➔ *Παράδειγμα*

```
public class BankAccount { // parent (super) class

     protected String name;
     protected double balance;

     public BankAccount(String name, double balance) {
          this.name = name;
          this.balance = balance;
     }

    public void printInfo() {
     System.out.println("Name: " + name);
     System.out.println("Balance: " + balance);
    }
}


public class SavingsAccount extends BankAccount { // child class


     private double interestRate; // child class: παίρνει
     αυτόματα τις ιδιότητες – πεδία (μαζί με την εμβέλεια τους)
     της parent class χωρίς να τα δηλώσω εκ νέου

     public SavingsAccount(String name,double cash,double rate)
     {    super(aName, cash);  //καλεί constructor της parent
          interestRate = aRate;
     }

     public void addInterest() {balance+= balance*interestRate;}

     public void printInfo() {
          super.printInfo();   //καλεί μέθοδο της parent (super)
          System.out.println("Interest rate: " + interestRate);
     }
}
```

➔ *Παρατηρήσεις στη κληρονομικότητα*

- Η κλάση Employee από την οποία κληρονομεί η AdministrativeEmployee ονομάζεται υπερκλάση (και συνήθως απεικονίζεται ψηλότερα) ενώ η κλάση που κληρονομεί ιδιότητες και λειτουργίες ονομάζεται υποκλάση
- Η νέα κλάση "απορροφά" από την υπάρχουσα όλες τις ιδιότητες και τη συμπεριφορά της και μπορεί να δηλώσει επιπλέον ιδιότητες ή/και μεθόδους
- Μια ιδιότητα που έχει δηλωθεί με ιδιωτική ορατότητα (private), αν και κληρονομείται από την υποκλάση, δεν επιτρέπεται η πρόσβαση (ανάγνωση ή αλλαγή τιμής). Για να επιτραπεί η πρόσβαση σε ιδιότητες στις υποκλάσεις το προσδιοριστικό ορατότητας θα πρέπει να μεταβληθεί σε protected
- Στη γλώσσα Java δεν επιτρέπεται η πολλαπλή κληρονομικότητα (δηλαδή μια κλάση δεν μπορεί να έχει περισσότερες από μία υπερκλάσεις)
- Χρησιμοποιώντας τη δεσμευμένη λέξη super και το όνομα μιας μεθόδου της υπερκλάσης μπορούμε να καλέσουμε οποιαδήποτε μέθοδο της υπερκλάσης.
- Αρχή της υποκατάστασης: ένα αντικείμενο μιας υπερκλάσης, θα πρέπει να είναι απολύτως επιτρεπτό (δηλαδή να μην δημιουργούνται προβλήματα εκτέλεσης) να υποκαθίσταται με αντικείμενα υποκλάσεων του. Αυτό γατί μια σχέση κληρονομικότητας αποτελεί μια σχέση τύπου "είναι" (is-a), και άρα, τα αντικείμενα των υποκλάσεων "είναι" και στιγμιότυπα των υπερκλάσεών τους. (πχ να επιτρέπεται Employee ref = new Employee("John", 850); αλλά και το Employee ref = new AdministrativeEmployee("John", 850, 10, 15);)
- Μπορούμε να επικαλύψουμε ή να επανορίσουμε μεθόδους της υπερκλάσης ώστε να ταιριάζουν στη κάθε υποκλάση. Δηλαδή να ορίσουμε τη μέθοδο με ένα όνομα στην υπερκλάση, και να πάμε σε όλες τις υποκλάσεις και, σε κάθε μια ξεχωριστά, να ορίζουμε στο σώμα της μια μέθοδο με ακριβώς το ίδιο όνομα που είχε στην υπερκλάση, αλλά εσωτερικά να γράφουμε έξτρα ή εντελώς διαφορετικό κώδικα. Πιο απλά, να παίρνουμε ονομάτα κλάσεων της υπερκλάσης και για κάθε υποκλάση να γράφουμε διαφορετική υλοποίηση. Περισσότερα για αυτός στις αφηρημένες μεθόδους και κλάσεις.

- ✓ Αν έχουμε μια δομή δεδομένων τύπου υπερκλάσης, μπορώ να τη γεμίσω με ποικίλα στοιχεία τύπου υπερκλάσης ή των παιδιών της (λόγω αρχής της υποκατάστασης). Κατά τη μεταγλώττιση, δεν είναι γνωστό τι τύπου δεδομένα/ μεθόδους θα ζητήσει κάθε στοιχείο της δομής, κι έτσι, κατά τη διάρκεια εκτέλεσης η γλώσσα έχει μια μικρή καθυστέρηση για να πάρει τη σωστή απόφαση. Αυτό ονομάζεται "δυναμική διασύνδεση" ή "καθυστερημένη διασύνδεση" μεθόδου και αντίστοιχης κλάσης (dynamic ή late binding). Έτσι προκύπτει αυτό που λέμε «πολυμορφική κλήση».

Αφηρημένες κλάσεις και μέθοδοι (abstract)

Ορίζουμε μια κλάση ή μέθοδο abstract όταν θέλουμε να θέτει μονάχα το γενικό πρότυπο για να χρησιμοποιηθεί από τις υποκλάσεις, χωρίς να θέλουμε να χρησιμοποιήσουμε τη κλάση/ μέθοδο αυτή καθ' αυτή στο σημείο που τη δηλώνουμε. Για παράδειγμα, μια κλάση μαθητής μπορεί να θεωρηθεί abstract αν όλα μου τα αντικείμενα – παιδιά είναι συγκεκριμένο είδος μαθητή (πχ προπτυχιακοί ή μεταπτυχιακοί ή διδακτορικοί) αλλά όχι σκέτο «μαθητές», γενικά κι αόριστα.

➔ *Παράδειγμα*

```
public abstract class Student {
	private String name;
	private String id;

	public Student(String name, String id) {
		this.name = name;
		this.id = id;
	}

	public void printInfo() {
		System.out.println("Name: " + name);
		System.out.println("Id: " + id);
	}

	public abstract void printType();
}


public class UnderGradStudent extends Student {
	private int year;

	public UnderGradStudent(String name, String id, int year) {
		super(name, id);
		this.year = year;
	}

	public void printInfo() {
		super.printInfo();
		System.out.println("year: " + year);
	}

	public void printType() {
		System.out.println("Hello I am an undergraduate");
	}


}
```

```java
public class PhDStudent extends Student {
	private String thesis;

	public PhDStudent(String name, String id, String thesis) {
		super(name, id);
		this.thesis = thesis;
	}

	public void printInfo() {
		super.printInfo();
		System.out.println("Thesis: " + thesis);
	}

	public void printType() {
		System.out.println("Hello I am a PhD student");
	}

}
```

➔ *Παράδειγμα συσχέτισης κλάσεων*

```
// import java.util.ArrayList;  → για χρήση ArrayList

public class Main {

     public static void main(String[] args) {
          Student S1 = new Student("ics1209");

          //S1.printInfo();  προσοχή, αν κληθεί εδώ η printInfo
                             θα προκληθεί Null Pointer Exception

          Course C1 = new Course("Java");
          Course C2 = new Course("Web Programming");

          C1.addStudent(S1); //σύνδεση Course C1 *-* Student S1
          C1.addStudent(S2); //σύνδεση Course C1 *-* Student S2
          C2.addStudent(S1); //σύνδεση Course C2 *-* Student S1
     }
}


public class Course {
     private String name;
     private Student[] enrolledStudents;
     private int stud_in_THIS_course;
     final int max_stud_per_course = 20;
     // ή, αντί για array, Μπορώ να χρησισημοποιήσω ArrayList
     //     → σε αυτή τη περίπτωση, ΔΕ χρειάζομαι μεταβλητή
     //       δείκτη (stud_in_THIS_course) ούτε προκαθορισμένο
     //       μέγιστο όριο θέσεων στη δομή (max_stud_per_course)
     // private ArrayList<Student> enrolledStudents;
     private static int stud_in_ALL_courses = 0;

     public Course(String text) {
          name = text;
          enrolledStudents = new Student[max_stud_per_course];
          stud_in_THIS_course = 0;
          // ή, αντί για array, είχαμε ArrayList
          //   enrolledStudents = new ArrayList<Student>();
     }

     public void addStudent(Student s) {
          if(counter < max_stud_per_course) {
               enrolledStudents[stud_in_THIS_course] = s;
               stud_in_THIS_course ++;
               stud_in_ALL_courses ++;
```

```
        }
    }

    // ή, αντί για array, είχαμε ArrayList
    //   public void addStudent(Student s) {
    //        enrolledStudents.add(s);
    //   }

    public String getName()  { return name; }
}


public class Student {

    private String id;
    //  AN κάθε μαθητής είχε μόνο 1 μάθημα
    //  private Course myCourse;
    //  → φυσικά, σε αυτή τη περίπτωση, η κλάση Course δε θα
    //     είχε Πίνακα μαθητών, και, επειδή θα είχα συσχέτιση
    //     ένας Student δηλώνει 1 μόνο μάθημα, και
    //     ένα Course μπορεί να δηλωθεί από πολλούς μαθητές
    //     δηλαδή Student -> Course
    //     το Course ΔΕΝ θα είχε καμία ιδιότητα τύπου Student
    //     ενώ το Student θα υιοθετούσε 1 ιδιότητα τύπου Course


    public Student(String text) {
        id = text;
    }

    public String getID()  {
        return id;
    }

    //   AN κάθε μαθητής είχε μόνο 1 μάθημα
    //   public void setMyCourse(Course c)  {
    //        myCourse = c; }

    public void printInfo() {
        System.out.println("AM: " + id);
        //  AN κάθε μαθητής είχε μόνο 1 μάθημα
        //  System.out.println("Course: "+myCourse.getName());
    }

}
```

Εκτύπωση στην οθόνη

```
// εκτύπωση στην οθόνη ΧΩΡΙΣ επιπρόσθετο χαρακτήρα αλλαγής
γραμμής στο τέλος
System.out.print("Hello world! ");
// εκτύπωση στην οθόνη κι αυτόματη αλλαγή γραμμής στο τέλος
//   -> ίδιο με την παραπάνω εντολή αλλά ΜΕ το επίθεμα "ln" στο
τέλος της "print"
System.out.println("Hello world! ");
// ίδιο με την σκέτη "print" αν της προσθέσω στο τέλος το \n
System.out.print("Hello world! \n");

// εκτύπωση κειμένου και άλλων στοιχείων (σύνθετη εκτύπωση)
//   -> προσθέτω απλώς "+" ανάμεσα στα διακριτά στοιχεία
System.out.println("ID: " + id);
```

Εισαγωγή δεδομένων από το πληκτρολόγιο

```
// εισαγωγή βιβλιοθηκών
import java.util.Scanner;

// ...

// δημιουργία αντικειμένου κλάσης Scanner – όνομα "in"
Scanner in = new Scanner(System.in);

// διάβασμα συμβολοσειράς (string)
System.out.println("Enter product name: ");
name = in.nextLine();
// διάβασμα ακεραίου (integer)
System.out.println("Enter product score: ");
score = in.nextInt();
// διάβασμα αριθμού κινητής υποδιαστολής (double)
System.out.println("Enter product price: ");
price = in.nextDouble();
```

Δομές Επιλογής

○ *if … else if… else …*

Έλεγχος ισχύς ενός σεναρίου με τη διατύπωση κάθε υπόθεσης ως τμήμα της if ή κάποιου τμήματος if else. Ο έλεγχος κάθε σεναρίου γίνεται «από πάνω προς τα κάτω», δηλαδή πρώτα ελέγχεται άμα ισχύ το σενάριο που έχει γραφεί πρώτο, μετά το δεύτερο, κλπ. Στο πρώτο σενάριο που βρεθεί να επαληθεύεται, σταματά ο έλεγχος κι εκτελείται το τμήμα κώδικα του μπλοκ ακριβώς μετά τον έλεγχο της συγκεκριμένης υπόθεσης. Έπειτα, ο μεταγλωττιστής φεύγει από το μπλοκ της if και συνεχίζει από εκεί που σταματά το μπλοκ του τελευταίου σεναρίου. Το τμήμα else (προαιρετική η χρήση του) εκτελείται άμα όλα τα παραπάνω σενάρια απορριφθούν.

➔ *Παράδειγμα (1)*

```
int price = 22;
if (price < 10) { // ελέγχεται αν  η τιμή είναι <10
   // --> ΑΝ ναι (true),
   //        τότε εκτελείται αυτό το μπλοκ
   // --> ΑΝ οχι (false),
   //        τότε ΔΕΝ εκτελείται αυτό το μπλοκ
   //        και ελέγχεται η ΑΜΕΣΩΣ επόμενη συνθήκη (else if)
   System.out.println("Cheap");
}
else if (price < 15) {
   // αν απορρίφθηκε το πάνω σενάριο,
   // δηλαδή τιμή ΟΧΙ <10 ( -> άρα price>=10)
   // ελέγχεται αν price < 15 (στην ουσία αν 10 <= price < 15)
   // --> ΑΝ true, τότε εκτελείται αυτό το μπλοκ
   // --> ΑΝ false, τότε ελέγχεται το αμέσως επόμενο else if
   System.out.println("Not so cheap");
}
else if (price < 20) {
   // αν απορρίφθηκε το πάνω σενάριο,
   // δηλαδή τιμή ΟΧΙ <10 και ΟΧΙ <15 ( -> άρα price>=15)
   // ελέγχεται αν price < 20
   // --> ΑΝ true, τότε εκτελείται αυτό το μπλοκ
   // --> ΑΝ false, τότε ελέγχεται το αμέσως επόμενο else if
   //               ΑΝ δεν υπάρχει άλλο τμήμα else if,
   //                  εκτελείται το μπλοκ της else
   //               ΑΝ δεν υπάρχει τμήμα else,
   //                  βγαίνουμε από όλη την if χωρίς να
   //                  εκτελέσουμε κανένα απ τα πιθανά σενάρια
   System.out.println("Pretty expensive");
}
else {    // αν ΟΛΑ τα παραπάνω έχουν απορριθεί,
          // εκτελείται το  μπλοκ της else
   System.out.println("So expensive");}
```

➔ *Παράδειγμα (2)*

```
int x = 10, y = 5;
if (x == y) {
   System.out.println("X equal to Y");
}
else { // if ( X != Y )
   System.out.println("X different from Y");
}
```

➔ *Παράδειγμα (3)*

```
int x = 10, y = 5, max;
boolean print_msg = true, extra_cond = false;

if (x > y) {
   max = x;
   System.out.println("X greater than Y");
   if (print_msg)
        System.out.println("(X > Y) AND (print_msg==true)");
}
else if ((x < y) && (extra_cond)) {
// same as: x < y && extra_cond
   max = y;
   System.out.println("(X < Y) AND (extra_cond == true)");
   System.out.println("both need to be true to execute this");
}
else if ((x < y) || (extra_cond)) {
// same as: x < y || extra_cond
   max = y;
   System.out.println("(X < Y) OR (extra_cond == true)");
   System.out.println("(just one condition of these could be
   true (both is okay) to execute this block)");
}
```

- *switch… case… default…*

Μια μορφή απλουστευμένης if για έλεγχο τιμής συγκεκριμένης μεταβλητής. Διαφέρει στο ότι κάθε σενάριο δε μπορεί να δεχτεί σύνθετες συνθήκες (δηλαδή με ΚΑΙ, Η…), συν ότι ακόμη κι αν εκτελεστεί ένα σενάριο, ο μεταγλωττιστής συνεχίζει τους ελέγχους και στα υπόλοιπα σενάρια. Αν κάτι τέτοιο δεν είναι επιθυμητό (δηλαδή θέλουμε αμέσως μόλις εκτελεστεί το μπλοκ ενός σεναρίου να φεύγει από τη switch και να πάει στο κώδικα μετά από τα σενάρια), τότε έπειτα από κάθε case πρέπει να βάζω τη λέξη break, ώστε να διακοπεί η προσπέλαση της switch. Αν δεν εκτελεστεί κανένα σενάριο case, τότε εκτελείται το μπλοκ του default (αν υπάρχει, προαιρετικό στη χρήση), ομοίως με το else της if.

➔ *Παράδειγμα*

```
int day_of_week = 4;
switch ( day_of_week ) {
   case 1:
        System.out.println("It's Monday");
        break;      // βάζουμε break ώστε να διακοπεί // if
                    // προσπέλαση της switch, γιατί αλλιώς
                    // θα συνεχίσει να ελέγχει όλα τα σενάρια
                    // ακόμη κι αν ήδη έχει εκτελέσει κάποιο-α

   case 2:
        System.out.println("It's Tuesday");
        break;      // ξανά, και εδώ βάζουμε break για να
                    // σταματήσει ο έλεγχος των άλλων case
   case 3:
        System.out.println("It's Wednesday");
        break;
   case 4:         // if (day_of_week == 4)
        System.out.println("It's Thursday");
        break;
   case 5:         // if (day_of_week == 5)
        System.out.println("It's Friday");
        break;
   case 6:
        System.out.println("It's Saturday");
        break;
   default: // case 7:
        System.out.println("It's Sunday");
        break;
}
```

Επαναληπτικές δομές

- *while*

➔ *Παράδειγμα*

```
int i = 1;
while ( i<=10 ) {
   System.out.println(i + "th loop");
   i++;
}
```

- *do … while*

➔ *Παράδειγμα*

```
int i = 1;
do {
   System.out.println(i + "th loop");
   i++;
} while ( i<=10 );
```

- *for*

➔ *Παράδειγμα*

```
for ( int i=1; i<=10; i++ )  {
   System.out.println(i + "th loop");
}
```

- *for – each*

➔ *Παράδειγμα*

```
int[] array_int = {1,15, 5, 8,11,5};
int i = 0;
for (int element : array_int) {
   System.out.println("Array[" + i + "] = " + element);
   i++;
   // array_int[i] ίδιο με element
}
```

int – Integer

Κλάση με μεθόδους για αυτοματισμούς στη διαχείριση ακεραίων. Διαφέρει από τον απλό τύπο δεδομένων "int" (ο οποίος παράγει primitive data), γιατί η κλάση διαθέτει επιπλέον λειτουργίες που μπορούν μας διευκολύνουν.

```
// ΔΗΜΙΟΥΡΓΙΑ ακέραιας μεταβλητής
int number;
// Δημιουργία και αρχικοποίηση ακέραιας μεταβλητής
int number2 = 5;

// ΕΞΑΓΩΓΗ ακεραίου από String
// --> Default: το string δίνεται στο δεκαδικό σύστημα
String strNum = "77";
number = Integer.parseInt(strNum);          // → 77

String strNegNum = "-893";
number = Integer.parseInt(strNegNum);       // → -893

// --> το string δίνεται σε διαφορετικό σύστημα αρίθμησης
//     >> επιλογές ως 2η παράμετρος (radix) σχετικά με το
//        το σε ποιο σύστημα αρίθμησης είναι γραμμένο το string:
//        10(δεκαδικό), 2(δυαδικό), 8(οκταδικό), 16(δεκαεξαδικό)
strNum = "1011";
number = Integer.parseInt(strNum, 2);       // → 11
                          // γιατί 1*1 + 1*2 + 0*4 + 1*8)
strNum = "34";
number = Integer.parseInt(strNum, 8);       // → 28
                          // γιατί 4*1 + 3*8)
strNum = "A5B";
number = Integer.parseInt(strNum, 8);       // → 2651

                          // γιατί A(=10)*1 + 5*16 + B(=11)*256
```

String

Ένα σύνολο από ενωποιημένους απλούς χαρακτήρες (char) που κατά βάση αντιμετωπίζονται ως ένα εννιαίο πράγμα, αλλά παρέχουν ταυτόχρονα δυνατότητα αναγνώρισης κάθε μεμονωμένου χαρακτήρα (αν και σε περίπτωση που θέλουμε πιο αυστηρή επεξεργασία κειμένου, προτιμάμε άλλους τρόπους διαχείρισης χαρακτήρων).

```
// Δήλωση String
String aString;
// Δήλωση και δημιουργία primative String
String strPlain = "I love color red";
String strSAMEmsg = "I love color red";
String strPlain3 = "Falala";
// Δήλωση και δημιουργία object String
String strObj2 = new String("I love color red");
String strObj = new String("I love color red");


// ΕΛΕΓΧΟΣ ΙΣΟΤΗΤΑΣ STRING

// ==
// αν βάλω == σε primitive data, συγκρίνει περιεχόμενα
if (strPlain == "I love color red")
     System.out.println("1.A) strPlain == as its msg in plain"
     + " condition is TRUE");
// αν βάλω == σε Object, συγκρίνει ΔΙΕΥΘΥΝΣΕΙΣ, όχι περιεχόμενο
if (strObj == "I love color red")
     System.out.println("1.B) strObj == as its msg in plain"
     + " condition is FALSE");

// .equals()
// συγκρίνει περιεχόμενα string, είτε primitive είτε object
if (strPlain.equals("I love color red"))
     System.out.println("2.i.A) strPlain EQUALS as its msg in
     plain"
     + " condition is TRUE");
if (strObj.equals("I love color red"))
     System.out.println("2.i.B) strPlain EQUALS as its msg in
     plain"
     + " condition is TRUE");
// equals() με string αποθηκευμένο σε μεταβλητή
// --> ίδια συμπεριφορά
if (strPlain.equals(strSAMEmsg))
     System.out.println("2.ii.A) Still TRUE");
if (strObj.equals(strSAMEmsg))
     System.out.println("2.ii.B) Still TRUE");
```

```
// σύγκριση ισότητας string με διαφορετικό περιεχόμενο --> false
// τόσο σε primitive όσο και σε object
if (strPlain.equals(strPlain3))
     System.out.println("3.A) Not same content strings,"
     + " condition is FALSE");
if (strObj.equals(strPlain3))
     System.out.println("3.B) Not same content strings,"
     + " condition is FALSE");


// ΣΠΑΣΙΜΟ STRING

// Η μέθοδος split της κλάσης String
// παίρνει ένα όρισμα τύπου string (delimiter)
// και σπάει το String με βάση το delimiter
// και επιστρέφει ένα πίνακα από Strings
// με τις χωρισμένες λέξεις
String line = "This is such a nice day! Hyrray :)";
String [] words = line.split(" ");
// --> εδώ, χωρίζει λέξεις βάση το " " (κενό)
```

Built-in Δομές δεδομένων

Πίνακας (array)

Μια συλλογή δεδομένων ιδίου τύπου σταθερού και προκαθορισμένου μεγέθους, στην οποία κάθε στοιχείο προσδιορίζεται με ένα δείκτη (το 0ο, 1ο, 2ο, … στοιχείου του πίνακα αυτού). Η αρίθμηση των στοιχείων αρχίζει από το μηδέν (zero – based). Ο τύπος δεδομένων του πίνακα μπορεί να είναι είτε κάτι που προυπάρχει στις βιβλιοθήκες της java (όπως int, String, …) είτε να ακόμη και τύπου μιας κλάσης που δημιουργήσαμε εμείς.

```
int total_places = 20;

// Δήλωση ΜΟΝΟΔΙΑΣΤΑΤΟΥ πίνακα
double[] array1;
// Δημιουργία πίνακα με σύνολο total_places θέσεις
// οι θέσεις του αριθμούνται
// από [0] ως [total_places - 1]
String[] array2 = new String[total_places];
// ίδιο με το να έγραφα
String[] array22 = new String[20];

// Δημιουργία πίνακα που βάζω manually τα στοιχεία
// πχ 10, 20, 35, 42
// που αντοιστοιχούν στις θέσεις
// array3[0], array3[1], array3[2], array3[3]
int[] array3 = {10, 20, 35, 42};

// άντληση μεγέθους πίνακα
int how_many_places_are_in_the_array = array3.length;

// προσπέλαση πίνακα με for
for (int i = 0; i < array3.length; i++)
System.out.println("array[" + i + "] = " + array3[i]);
// και προσδιορισμός του μεγέθους του
// με την .lenght
// προσπέλαση πίνακα με for – each
int i=0;
for (int element : array3)
System.out.println("array[" + i + "] = " + element);

// Δήλωση ΔΙΣΔΙΑΣΤΑΤΟΥ πινάκων
int[][] a1 = { {10, 20, 30},
               { 3,  4,  5} };
int[][] a2 = new int[3][2];
```

```
// προσπέλαση δισδιάστατου πίνακα
// --> οριζόντια διάσχιση (α[0][0], α[0][1],...)
for (int num_row = 0; num_row < a1.length; num_row++ )
     for (int col = 0; col < a1[num_row].length; col++)
          System.out.print(a1[num_row][col]+" , ");
          // -> 10 , 20 , 30 , 3 , 4 , 5

System.out.println();

// --> κάθετη διάσχιση (α[0][0], α[1][0],...)
for (int num_col = 0; num_col < a1[0].length; num_col++ )
     for (int row = 0; row < a1.length; row++)
          System.out.print(a1[row][num_col] +" , ");
          // -> 10 , 3 , 20 , 4 , 30 , 5
```

Στοίβα (υλοποίηση με πίνακα)

Μπορούμε να υλοποιήσουμε πίνακες τη δομή μιας στοίβας, δηλαδή μιας συλλογής δεδομένων (ιδίου τύπου σταθερού και προκαθορισμένου μεγέθους, αν το φτιάξουμε με πίνακες), στην οποία μπορώ να έχω πρόσβαση μόνο στο κορυφαίο (πάνω – πάνω στοιχείο, θέση 0) και άμα θέλω να το ανακτήσω πρέπει αναγκαστικά να το εξάγω από τη δομή. Ακολουθείται η πολιτική FIFO (First In, First Out).

```
class Stack {
	private int capacity;
	private int size = 0;
	private int[] elements;

	public Stack(int capacity){
		this.capacity = capacity;
		elements = new int[capacity];
	}

	public void push(int element){
		if (size == capacity){
			System.out.println("Stack is full");
			return;
		}
		elements[size] = element;
		size ++;
	}

	public int pop(){
		if (size == 0){
			System.out.println("No elements to pop");
			return -1;
		}
		size -- ;
		return elements[size];
	}

	public boolean isEmpty(){
		return (size == 0);
	}
}
```

Στοίβα

Μπορούμε να υλοποιήσουμε πίνακες τη δομή μιας στοίβας, δηλαδή μιας συλλογής δεδομένων (ιδίου τύπου σταθερού

ArrayList

Μοιάζει με πίνακα, αλλά με περισσότερες δυνατότητες, λόγω της δυναμικότητας του. Αρκεί μονάχα η δήλωση του στο χώρο ονομάτων, χωρίς να θέσουμε προκαθορισμένο μέγεθος στοιχείων στη δομή. Πάλι υπάρχουν δείκτες για τη διάταξη τον στοιχειών στη λίστα, τα οποία όμως αντιμετωπίζουμε με τη λογική της αντικειμενοστρέφειας, δηλαδή αξιοποιώντας μεθόδους.

```
// Εισαγωγή βιβλιοθήκης ArrayList
import java.util.ArrayList;
// Προεραιτική εισαγωγή για επέκταση ArrayList
import java.util.Collections;


// Δήλωση ArrayList
// ArrayList<data_type> name = new ArrayList<data_type>();
// --> παίρνει δεδομένα ΜΟΝΟ τύπου κλάσεων
ArrayList<String> things_you_said = new ArrayList<String>();
// .... ή πιο απλά:
ArrayList<String> things_you_said2 = new ArrayList<>();
// ... άρα αν θέλω για ακεραίους, ΔΕ θα βάλω int, αλλά Integer
ArrayList<Integer> listWithIntegers = new ArrayList<>();

// ΕΙΣΑΓΩΓΗ στοιχείου σε ArrayList
// obj.add(<data_type> data);
things_you_said.add("Can I take you out tonight?");
things_you_said.add("Be mine!");
things_you_said.add("...can we break up?");
// obj.add(int index, <data_type> data);
things_you_said.add(2, "I got bored of you...");

// εύρεση ΜΕΓΕΘΟΥΣ (πλήθους στοιχείων) arraylist
// obj.size();
int size_of_arraylist = things_you_said.size();

// ΣΤΟΙΧΕΙΟ ΣΤΗ ΘΕΣΗ No. index σε ArrayList
// obj.get(index);
int index = 1;
String string_in_index_position = things_you_said.get(index);
// --> πχ εδώ, επιστρέφει το "Be mine!"

// Προσπέλαση arraylist με for - each
// for (data_of_arraylist i : arraylist_name)
for (String phrase : things_you_said)
     System.out.print(phrase + ", ");

System.out.println();
```

```
// ΑΝΤΙΚΑΤΑΣΤΑΣΗ στοιχείου σε συγκεκριμένη θέση
// obj.set(index, new_data);
index = (size_of_arraylist - 1) - 1; // Προτελευταίο
String new_data = "I might be seeing another woman...";
things_you_said.set(index, new_data);
// πχ τώρα, στη θέση [2] αντί για "I got bored of you..."
// (το οποίο έχει ΦΕΥΓΕΙ από τη λίστα,
//  ΔΕΝ έχει μετακινηθεί πιο κάτω)
// έχει μπει το "I might be seeing another woman..."
// (και προφανώς το "I got bored of you..." χάθηκε)

// ΔΙΑΓΡΑΦΗ στοιχείου στη θέση No.index
// obj.remove(index);
things_you_said.remove(0);
// --> ΠΡΙΝ: θέση 0 -> "Can I take you out tonight?"
//           θέση 1 -> "Be mine!"
// --> ΜΕΤΑ: θέση 0 -> "Be mine!"
//           θέση 1 -> "I got bored of you..."

// τώρα, αν ξανά κάνω την ίδια εντολή, προκύπτει...
things_you_said.remove(0);
// --> ΠΡΙΝ: θέση 0 -> "Be mine!"
//           θέση 1 -> "I got bored of you..."
// --> ΜΕΤΑ: θέση 0 -> "I got bored of you..."
//           θέση 1 -> "I might be seeing another woman..."

// Διαγραφή όλων των στοιχείων (άδειασμα) του arraylist
// obj.clear();
things_you_said.clear();

// Επέκταση δυνατοτήτων με χρήση Collections
// Ταξινόμηση στοιχείων arraylist σε ΑΥΞΟΥΣΑ σειρά
// Collections.sort(arraylist_name); --> ascending order
Collections.sort(things_you_said);
// Ταξινόμηση στοιχείων arraylist σε φθήνουσα σειρά
Collections.sort(things_you_said); // αύξουσα By default
// ΑΝΤΙΣΤΡΟΦΗ ΣΕΙΡΑΣ στοιχείων σε arraylist
Collections.reverse(things_you_said); // αντιστροφή σειράς
```

LinkedList

Μοιάζει με ArrayList, αλλά είναι πιο γρήγορο κι ευέλικτο.

```
// Εισαγωγή βιβλιοθήκης List, LinkedList
import java.util.List;
import java.util.LinkedList;


List <String> list = new LinkedList<>();
list.add("falala");
list.add(0, "fififi");
list.remove(0);
list.clear();
```

…. *Implement Comparable*

Χρήσιμο για όταν θέλω να κάνω ταξινόμηση σε δικές μου κλάσεις.

```
class ButtonListener implements ActionListener {

	public void actionPerformed(ActionEvent e) {
		Collections.sort(books);
	}
}


//                ΚΑΝΩ implement Comparable<MyClass>
public class Book implements Comparable <Book> {
	private String title;
	private String authorSurname;
	protected Integer code;

	public Book(String title, String authorSurname, int code) {
		this.title = title;
		this.authorSurname = authorSurname;
		this.code = code;
	}

	// ΚΑΝΩ override την CompareTo
	// θέτοντας το δικό μου κριτήριο ταξινόμησης
	@Override
	public int compareTo(Book other) {
		return this.code.compareTo(other.code);
	}
}
```

Εξαιρέσεις

Υπάρχουν φορές που μπορεί να θέλουμε (ή να αναγκαζόμαστε) να τερματίσουμε σκόπιμα το πρόγραμμα μας άμα προκύψει κάποια ειδική περίπτωση (εξαίρεση).

➔ *Παράδειγμα (1)*

```
public class DanceLesson2 {

	public static void main(String[] args) {

		Scanner keyboard = new Scanner(System.in);
		System.out.println("Enter number of male and female
		dancers:");
		int men = keyboard.nextInt();
		int women = keyboard.nextInt();

		try{
			if (men == 0 && women == 0)
				throw new Exception("Lesson is canceled. No
			students.");
			else if (men == 0)
				throw new Exception("Lesson is canceled. No
			men.");
			else if (women == 0)
				throw new Exception("Lesson is canceled. No
			women.");

			if (women >= men)
				System.out.println("Each man must dance with
			" +
				women/(double)men + " women.");
			else
				System.out.println("Each woman must dance
			with " +
			men/(double)women + " men.");
		}
		catch(Exception e){
			e.printStackTrace();
			String message = e.getMessage( );
			System.out.println(message);
			System.exit(0);
		}

		System.out.println("Begin the lesson.");
	}
}
```

➔ *Παράδειγμα (2)*

```
public class BadNumberExceptionDemo {

     public static void main(String[] args) {
          try {
               Scanner keyboard = new Scanner(System.in);
               System.out.println("Enter year of birth:");
               int inputNumber = keyboard.nextInt();
               if (inputNumber > 2020)
               throw new BadNumberException(inputNumber);
               System.out.println("Thank you for entering " +
               inputNumber);
          }
          catch(BadNumberException e) {
               // επιστρέφει τον αριθμό που προκάλεσε πρόβλημα
               System.out.println(e.getBadNumber( ) + " is not
               valid.");
          }
          System.out.println("End of program.");
     }
}
```

Αρχεία

Διάβασμα από αρχείο

Παίρνοντας ως είσοδο ένα αρχείο, μπορώ να εξάγω τη πληροφορία του και να την αποθηκεύσω σε μεταβλητές του προγράμματος μου ώστε να αξιοποιήσω τις πληροφορίες που θέλω

```
// Εισαγωγή βιβλιοθηκών για διάβασμα από αρχείο
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;


// READ FILE
String filename = "test.txt";
String data_from_file = "";

try {
	// Δημιουργία ρεύματος FileReader
	FileReader reader = new FileReader(filename);
	// FileReader = read the contents of a file
	// as a stream of characters.

	int data = reader.read();
	//read() returns an int value which contains the byte value
	// --> if it returns -1, there is no more data to be read
	(EOF)

	while(data != -1) {
		System.out.print((char)data);
		data_from_file += (char)data;

		// keep reading file using read
		data = reader.read();
		// stop reading when data == -1
	}
	// Close file before exiting
	reader.close();
}
catch (FileNotFoundException e) {
	e.printStackTrace();
}
catch (IOException e) {
	e.printStackTrace();
}
```

Γράψιμο (εκ του μηδενός) σε αρχείο

Όταν γράφω σε ένα αρχείο, καλώντας το κατασκευαστή FileWriter, είτε το δημιουργώ στο φάκελο του πρότζεκτ μου, είτε (αν υπάρχει ήδη) χρησιμοποιώ αυτό. **ΠΡΟΣΟΧΗ, ότι η συγκεκριμένη μέθοδος καθαρίζει ΕΝΤΕΛΩΣ όλο το περιεχόμενο του αρχείου και το ξανά γράφει από το μηδέν** (για αυτό και υπάρχει κίνδυνος να χαθεί πληροφορία άμα δεν έχουμε αποθηκεύσει νωρίτερα το περιεχόμενο του σε ένα ασφαλές αντίγραφο). Γράφοντας με την .write() το κείμενο που θέτω ξεκινάει να γράφεται πάντα από την αρχή του αρχείου (ασχέτως αν πριν είχε άλλο κείμενο, το σβήνει, και γράφει εκ νέου), ενώ ή .append() στο να συνεχίσω να προσθέτω κείμενο στο τέλος του αρχείου μου.

Ξανά, **ΠΡΟΣΟΧΗ, γιατί η FileWriter θα διαγράψει αυτόματα όλο το περιεχόμενο του αρχείου ακόμη κι αν αντί για .write() χρησιμοποιήσουμε την .append().**

```
// Εισαγωγή βιβλιοθηκών για γράψιμο σε αρχείο
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;


// WRITE FILE
String filename = "test.txt";
String text_to_write = "lalala\nheyooo";
String new_txt_at_the_end = "\nMy new texttt";

try {
      // Create file connection with FileWriter stream
      FileWriter writer = new FileWriter(filename);
      // Write text to file
      writer.write(text_to_write);
      // Add text at the end of file's text (if needed)
      writer.append(new_txt_at_the_end);
      // close file before exiting
      writer.close();
      System.out.println("File successfully written");
}
catch (IOException e) {
      e.printStackTrace();
      // End program
      System.exit(1);
}
```

*Προσθήκη κειμένου στο τέλος ήδη υπάρχοντος (γραμμένου) αρχείου **ΧΩΡΙΣ απώλεια** της παλιάς πληροφορίας*

Υπάρχει ένας εύκολος τρόπος (άλλα ίσως όχι ο πιο αποδοτικός υπολογιστικά) προκειμένου να διατηρήσουμε τις πληροφορίες ενός αρχείου αλλά ταυτόχρονα να μη διαγράψουμε τις πληροφορίες που ήδη έχει ΚΑΙ να προσθέσουμε στο τέλος του νέες πληροφορίες.

1. Αρχικά, διαβάζουμε όλη τη πληροφορία του αρχείου και την αποθηκεύουμε σε μια μεταβλητή (αντίγραφο ασφαλείας). Την ονομάζω, ενδεικτικά, data_from_file (τύπου String).

```
// READ FILE (+ save ALL info in a back-up variable
String filename = "test.txt";
String data_from_file = "";          // initialize back-up variable

try {
      FileReader reader = new FileReader(filename);
      int data = reader.read();

      while(data != -1) {
      data_from_file += (char)data;    // add new text to variable
            data = reader.read();
      }
      reader.close();
}
catch (FileNotFoundException e) {
      e.printStackTrace();
}
catch (IOException e) {
      e.printStackTrace();
}
```

2. Μετά, συνδέουμε το αρχείο με FileWriter (το οποίο θα διαγράψει αυτόματα όλο το περιεχόμενο του αρχείου ακόμη κι αν αντί για .write() χρησιμοποιήσουμε την .append()). Και το αρχείο αδειάζει.

```
String new_txt_at_the_end = "\nMy new texttt";

try {
      FileWriter writer = new FileWriter(filename);
      // code to be written ...
}
catch (IOException e) {
      e.printStackTrace();
}
```

3. Γράφουμε στο αρχείο το κείμενο που διαβάσαμε πριν και είχαμε αποθηκεύσει στην data_from_file (δηλαδή, ουσιαστικά, του σβήσαμε ότι πληροφορία είχε πριν και τώρα του ξανά

γράφουμε τα ίδια ακριβώς πράγματα, το οποίο για μεγάλα αρχεία σημαίνει μεγάλη καθυστέρηση…)

4. Αφού του ξανά γράψουμε ότι ήδη είχε, κάνουμε με χρήση .append() τη νέα πληροφορία που θέλουμε να εισάγουμε.
5. Στο τέλος κλείνουμε το αρχείο με .close().

```
String new_txt_at_the_end = "\nMy new texttt";

try {
	FileWriter writer = new FileWriter(filename);
	writer.write(data_from_file);
	writer.append(new_txt_at_the_end);
	writer.close();

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

6. Το άθροισμα των data_from_file + new_txt_at_the_end δίνει το νέο επαυξημένο κείμενο που είναι γραμμένο στο αρχείο μας πλέον

```
String whole_file_txt = data_from_file + new_txt_at_the_end;
```

Γραφικά στη Java

Color

Γραφικό που σχετίζεται με το χρωματισμό των γραφικών στοιχείων. Η κατασκευή του αποτελείται από τη χρωματική τριπλέτα RGB που δέχεται 3 παραμέτρους για τη ποσότητα κόκκινου, πράσινου και μπλε που θέλουμε να έχει κάθε χρώμα (τιμές από 0 ως 255)

➔ *Παράδειγμα*

```
Color c1 = new Color(0, 0, 255);      // → «γνήσιο» μπλε
Color c2 = new Color(143, 196, 220); // → γαλαζοπράσινο
Color c3 = Color.BLACK;               // → σταθερά για μαύρο
c2 = Color.CYAN;                      // → άλλη έτοιμη σταθερά
```

Font

Γραφικό που σχετίζεται με τη γραμματοσειρά και την αισθητική του κειμένου

➔ *Παράδειγμα*

```
String letters_font = "Serif";
// other common choices: "SansSerif", "Monospaced",
//                       "Dialog", "DialogInput",
//                       "Garamond"
int letters_style = Font.ITALIC;
// other common choices: Font.BOLD, Font.PLAIN, ...
int size_of_letters = 20;
// other common choices: 11, 12, 14, 16

Font f = new Font(letters_font,
                  letters_style,
                  size_of_letters);
```

Dimension

Γραφικό που ορίζει ένα ορθογώνιο μέγεθος διαστάσεων μορφής X επί Y πίξελ

➔ *Παράδειγμα*

```
int X_pixels = 250, Y_pixels = 50;
Dimension d = new Dimension(X_pixels, Y_pixels);
```

JFrame

Η βάση δημιουργίας ενός βασικού παραθύρου με γραφικά στη Java. ΠΡΟΣΟΧΗ στη διάταξη των εντολών της JFrame. Πάνω του τοποθετούνται μικρότερα πλαίσια που λέγονται πάνελς (JPanel). Κάθε JPanel έχει επάνω του άλλα γραφικά στοιχεία (πχ κουμπιά, κείμενα, λίστες…).

```
// 1. SET UP FRAME
// Δημιουργία παραθύρου JFrame
frame = new JFrame();
//   ΣΗΜΕΙΩΣΗ: δημιουργία ΔΕ σημαίνει και εμφάνιση του
//             απευθείας στην οθόνη μας
//             -> αυτό γίνεται με τη κλήση της .setVisible(true)

// ορισμός τίτλου για το συγκεκριμένο παράθυρο
frame.setSize(420, 250);
// ορισμός τρόπου διάταξης γραφικών στην οθόνη
frame.setLayout(null);
// καθορισμός χρώματος στο παρασκήνιο της οθόνης
Color c = new Color(143, 196, 220);
frame.getContentPane().setBackground(c);
// καθορισμός του «τι γίνεται με το παράθυρο»
// αν πατηθεί το X που το κλείνει
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
// → EXIT_ON_CLOSE: terminate the whole program
// → DISPOSE_ON_CLOSE: just close this frame, but do not
//                     terminate other program's procedures
// → DO_NOTHING_ON_CLOSE: make the close button (X) seems like
//             it doesn't work / press it and take no action
frame.dispose();   // διάλυση του συγκεκριμένου frame
// καθορισμός μεγέθους παραθύρου (pixel X pixel)
frame.setTitle("Find suspect");


// 2. CREATE GRAPHIC ELEMENTS . . .

// 3. ADD ELEMENTS TO PANEL . . .

// 4. ADD PANEL(S) TO FRAME AND MAKE VISIBLE

// ΤΕΛΕΥΤΑΙΑ εντολή από άποψη δηλώσεων και διάταξης στοιχείων
στην οθόνη
// μπαίνει κάτω από τα panels (αφού καθοριστούν και καρφωθούν
στο frame)
// αλλά πριν το λειτουργικό κομμάτι του παραθύρου
// πχ Button listener(s), Mouse listener(s)...
frame.setVisible(true);
```

JPanel

Δημιουργεί πλαίσια που λειτουργούν σαν μεγάλα «πανιά» ή δοχεία, ικανά να στοιβάξουν γραφικά συστατικά, με ορισμένη διάταξη. Τα πάνελ δε μπορούν να σταθούν μόνα τους, οπότε να αποκτήσουν νόημα πρέπει να τοποθετηθούν σε κάποιο παράθυρο JFrame, το οποίο, με τη σειρά του είναι ικανό να στοιβάξει στοιχεία JPanel.

```
// SET UP PANEL
JPanel messagePanel = new JPanel();

// καθορισμός μεγέθους πάνελ
//   -> η 1η και 2η παράμετρος είναι η θέση
//      που θα μπει η πάνω αριστερή γωνία του πάνελ στο frame.
//      Η 3η και 4η παράμετρος είναι το μέγεθος (Χ επί Υ)
//      κάθε πλευράς του ορθογωνίου σε πίξελ
messagePanel.setBounds(50, 50, 450, 100);
// καθορισμός χρώματος πάνελ ως φόντο
messagePanel.setBackground(Color.YELLOW);


// Add panel to frame
frame.add(panel);
```

JButton

Δημιουργεί πλαίσια που έχουν όψη και λειτουργία κουμπιού. Δεν αρκεί μονάχα η δημιουργία και η δήλωση της μορφή του κουμπιού (αυτό είναι μονάχα το «φαίνεσθαι»). Χρειάζεται να υλοποιήσουμε και τη λειτουργικότητα του κουμπιού, αφομοιώνοντας έναν ακροατή, δηλαδή μια μέθοδο ευαισθησίας που να επαγρυπνεί πότε θα πατήσει ο χρήστης το κουμπί – και, όταν το πατήσει, να κάνει ορισμένες λειτουργίες (πχ να κλείσει το τρέχον παράθυρο και να ανοίξει ένα άλλο). ΠΡΟΣΟΧΗ στη σειρά.

```
class aClassNameGUI {
     aClassNameGUI (...) {
          // Δημιουργία κουμπιού που γράφει ένα μήνυμα επάνω του
          JButton button = new JButton("Press here");
          // ή 1. Δημιουργία κουμπιού χωρίς μήνυμα επάνω του
          JButton button = new JButton();

          // εισαγωγή κειμένου στο πεδίο
          button.setText("Just changed text written in the
          box");
          // καθορισμός μεγέθους κουμπιού
          button.setPreferredSize( new Dimension(250, 40) );
          // διατήρηση (true) ή απόκρυψη (false) ενός
          διακριτικού πλαισίου (focus) γύρω από τα μήνυμα που
          είναι γραμμένο μέσα στο κουμπί
          button.setFocusPainted(false);

          // 2. εισαγωγή στοιχείου JButton σε JPanel
          JPanel panel = new JPanel();
          panel.setBounds(50, 50, 300, 150);
          panel.add(button);

          // 3. Add panel to frame
          frame.add(panel);

          // 4. set button listener active
          ButtonListener listener = new ButtonListener();
          findButton.addActionListener(listener);
     }
}

class ButtonListener implements ActionListener {
     public void actionPerformed( ActionEvent e ) {
          // commands...
          frame.setVisible(false);
     }
}
```

JCheckBox

Γραφικό που δημιουργεί ένα διακριτικό τετράγωνο πλαίσιο (κουτάκι) με δυνατότητα επιλογής (check).

```
// Δημιουργία check box
// το κείμενο τοποθετείται (by default) δεξιά από το check box
JCheckBox check = new JCheckBox("Tick me");

Object[] options = {'e', 2, 3.14, 4, 5, "TURTLES!", check};


// Έλεγχος αν έχει επιλεχθεί το check box
if (check.isSelected())
    System.out.println("The box has been ticked");
```

JList

Προβάλει ένα γραφικό σε μορφή λίστας. Πρώτα δημιουργώ ένα μοντέλο λίστας (DefaultListModel), προσθέτω τα στοιχεία μου σε αυτό (δηλαδή τις επιλογές χρήστη από τη λίστα), κι έπειτα δημιουργώ ένα JList και εφαρμόζω πάνω του το μοντέλο μου.

```
// Δημιουργία ΜΟΝΤΕΛΟΥ λίστας
final DefaultListModel<String> l1 = new DefaultListModel<>();
// προσθήκη επιλογών στη λίστα μου
l1.addElement("C");
l1.addElement("C++");
l1.addElement("Java");
l1.addElement("PHP");
// δημιουργία Λίστας
final JList<String> list1 = new JList<>(l1);
// καθορισμός διαστάσεων λίστας
list1.setBounds(100,100, 75,75);
String data;
// ΑΝ εχει επιλαγεί μια τιμή, θα είναι από 0 έως N - 1
// εφόσον έχω N επιλογές στη λίστα μου
if (list1.getSelectedIndex() != -1)
data = "Programming language Selected: " +
list1.getSelectedValue();
```

JLabel

Απεικονίζει κείμενο, χωρίς δυνατότητα αλληλεπίδρασης (επεξεργασίας) από το χρήστη.
Χρησιμοποιείται κυρίως για να επεξηγήσει ή να δώσει οδηγίες.

```
// Δημιουργία κειμένου για JLabel
JLabel errorMsg = new JLabel("This is an explaining sentence.");

// Καθορισμός στυλιστικών επιλογών γραμματοσειράς
errorMsg.setFont(new Font("Garamond", Font.BOLD, 20));
// Καθορισμός χρώματος γραμμάτων
errorMsg.setForeground(new Color(113, 87, 90));
// προσθήκη JLabel σε  JPanel
messagePanel.add(errorMsg);
// Add panel to frame
frame.add(messagePanel);
```

JTextField

Δημιουργεί ορθογώνια πλαίσια με δυνατότητα να απεικονίζουν κείμενο, να το σβήνουν, και να επιτρέπουν στο χρήστη να εισάγει δεδομένα. Η μορφή του θυμίζει κουτάκι σε φόρμα συμπλήρωσης στοιχείων (πχ συμπλήρωση Ονοματεπωνύμου, τηλεφώνου, κλπ)

```
// Δημιουργία πεδίου μαζί με κείμενο (ήδη γραμμένο)
JTextField searchBox = new JTextField("Please enter suspect's
name");
// ή δημιουργία ενός πεδίου χωρίς να γράφει κάτι μέσα
searchBox = new JTextField();

// εισαγωγή κειμένου στο πεδίο
searchBox.setText("Just changed text written in the box");
// καθορισμός γραμματοσειράς στο πεδίο
searchBox.setFont(new Font("Serif", Font.ITALIC, 20));
// καθορισμός χρώματος στα γράμματα του πεδίου
searchBox.setForeground(Color.gray);
// καθορισμός (ιδεατού) μεγέθους διαστάσεων του πεδίου
searchBox.setPreferredSize( new Dimension(250, 40) );


// εισαγωγή στοιχείου JTextField σε JPanel
JPanel panel = new JPanel();
panel.setBounds(50, 50, 300, 150);
panel.add(searchBox);

// Add panel to frame
frame.add(panel);
// set frame visible
frame.setVisible(true);
```

JTextArea

Δημιουργεί, όπως η JTextField, ορθόγωνια πλαίσια που απεικονίζουν κείμενο με δυνατότητα επεξεργασίας. Η κύρια διαφορά μεταξύ του JTextField και του JTextArea είναι ότι το JTextField επιτρέπει την εισαγωγή μιας μόνο γραμμής κειμένου σε μια εφαρμογή GUI, ενώ το JTextArea επιτρέπει την εισαγωγή πολλαπλών γραμμών κειμένου. Η μορφή του θυμίζει φόρμα συμπλήρωσης ερωτήσεων ανοιχτού τύπου (πχ περιγράψτε το πρόβλημα σας σε λίγες γραμμές).

```
String mult_line_txt = ”One\n Twoooo\n Three \n\nFour”

// Δημιουργία πεδίου με κείμενο γραμμένο σε αυτό
JTextArea  txtArea = new JTextArea(mult_line_txt);
// καθορισμός μεγέθους πεδίου
txtArea.setPreferredSize(new Dimension(250, 70));


// εισαγωγή στοιχείου JTextArea σε JPanel
JPanel panel = new JPanel();
panel.setBounds(50, 50, 300, 150);
panel.add(txtArea);


// Add panel to frame
frame.add(panel);
```

JOptionPane

```
import javax.swing.JOptionPane;
```

JOptionPane.showMessageDialog(…)

Ένα έτοιμο frame με ενσωματωμένο JLabel, ένα (σχετικό) εικονίδιο και JButton με μήνυμα «OK», ώστε να λειτουργεί ως ένα άμεσο παράθυρο επικοινωνίας με τον χρήστη. Η 1η παράμετρος είναι για εισαγωγή parentComponent (αν δεν υπάρχει, βάζουμε null), η 2η παράμετρος για εισαγωγή κειμένου στο JLabel, η 3η παράμετρος για εισαγωγή τίτλου στο frame μας, και η 4η παράμετρος για το είδος παραθύρου που θέλουμε να ανοίξει (απλό, πληροφόρησης, ερώτησης, προειδοποίησης, ή λάθους). Όποιο Styling κι αν επιλεχτεί, συνοδεύεται από το αντίστοιχο εικονίδιο (το οποίο μπορούμε να αλλάξουμε καλώντας διαφορετικό κατασκευαστή), και με πλήκτρο «OK» που όταν πατηθεί κλείνει το συγκεκριμένο pop – up παράθυρο.

```
// έτοιμο απλό Pop - Up frame, με JLabel, icon, και JButton "OK"

//   PLAIN_MESSAGE -- μήνυμα χωρίς Styling
JOptionPane.showMessageDialog(null, "Some plain window text",
            "pop up title", JOptionPane.PLAIN_MESSAGE);
//   INFROMATION_MESSAGE -- μήνυμα σε μορφή πληροφόρησης
JOptionPane.showMessageDialog(null, "Here's some info",
            "Info box", JOptionPane.INFORMATION_MESSAGE);
//   QUESTION_MESSAGE -- μήνυμα με εικονίδιο ερωτηματικού
JOptionPane.showMessageDialog(null, "Can I ask smth?",
            "Q title", JOptionPane.QUESTION_MESSAGE);
//   WARNING_MESSAGE -- μήνυμα με εικονίδιο σήμα κινδύνου (!)
JOptionPane.showMessageDialog(null, "Warning! Beware…",
            "W.box", JOptionPane.WARNING_MESSAGE);
//   ERROR_MESSAGE -- μήνυμα με εικονίδιο λάθους (X)
JOptionPane.showMessageDialog(null, "ERROR-- Invalid",
            "Error box", JOptionPane.ERROR_MESSAGE);
```

JOptionPane.showConfirmDialog(…)

Όπως και το showMessageDialog, πρόκειται για ένα έτοιμο frame με ενσωματωμένο JLabel, ένα (σχετικό) εικονίδιο και customised JButton(s). Έχει ακόμα και ίδιο (βασικό) κατασκευαστή. Η διαφοροποίηση εδώ είναι στο ότι μας επιτρέπεται να αποθηκεύσουμε σε μια (int) μεταβλητή το αποτέλεσμα της μεθόδου, ώστε να ξέρουμε ποιο από όλα τα κουμπιά επέλεξε ο χρήστης (πχ το πλήκτρο που κλείνει το παράθυρο, το «OK», το «YES», το «NO», το «CANCEL», κλπ).

```
// έτοιμο απλό Pop - Up frame, με JLabel, icon, και JButton(s)


int choice; // ακέραιος που συμβολίζει το επιλεχθέν πλήκτρο
// αν -1: ο χρήστης έκλεισε το frame πατώντας (X) πάνω δεξιά
// αν <> -1: ο χρήστης επέλεξε ένα πλήκτρο από τις επιλογές του
//           όλα τα άλλα κουμπιά μετρούνται από τα αριστερά προς
//           τα δεξιά, ξεκινόντας το μέτρημα από το μηδέν

choice = JOptionPane.showConfirmDialog(null, "Wanna proceed?",
         "title", JOptionPane.YES_NO_CANCEL_OPTION);

          // πχ εδώ, για το YES_NO_CANCEL_OPTION
          //    αν κλείσει το frame → [X] = -1
          //    υπάρχουν ΜΕ ΑΥΤΗ ΤΗ ΣΕΙΡΑ τα κουμπιά
          //          [ YES ]      [ NO ]     [ CANCEL ]
          //    τότε [ YES ] = 0,
          //          [ NO ] = 1,
          //          [ CANCEL ] = 2

choice = JOptionPane.showConfirmDialog(null, "Wanna proceed?",
         "title", JOptionPane.YES_NO_OPTION);

          // πχ εδώ, για το YES_NO_OPTION
          //    αν κλείσει το frame → [X] = -1
          //          [ YES ]   [ NO ]
          //    τότε [ YES ] = 0, [ NO ] = 1
choice = JOptionPane.showConfirmDialog(null, "Take some info…",
         "title", JOptionPane.INFORMATION_MESSAGE);

          // ίδιο με το να είχα Message αντί του Confirm,
          // ΑΛΛΑ με το έξτρα εδώ ότι στο Confirm μπορώ
          //      να αποθηκεύω το κουμπί που πάτησε ο χρήστης
```

✓ *Άλλες επιλογές πλήκτρων (για τη τελευταία παράμετρο):*

```
JOptionPane.OK_OPTION, JOptionPane.CANCEL_OPTION,
JOptionPane.QUESTION_OPTION, JOptionPane.ERROR_OPTION,
JOptionPane.WARNING_OPTION, JOptionPane.DEFAULT_OPTION,
```

JOptionPane.showInputDialog(…)

Όπως και το showConfirmDialog, πρόκειται για ένα έτοιμο frame με ενσωματωμένο JLabel, ένα (σχετικό) εικονίδιο, JButton(s) ΚΑΙ πεδίο JField για εισαγωγή δεδομένων. Έχει ίδιο (βασικό) κατασκευαστή, αλλά αρκεί απλώς να δώσουμε ως παράμετρο το μήνυμα που θέλουμε να έχει το JLabel. Επίσης, μας επιτρέπεται να αποθηκεύσουμε σε μια μεταβλητή (string) τα δεδομένα που θα εισάγει ο χρήστης στο σχετικό πεδίο.

```
// έτοιμο απλό Pop - Up frame, με JLabel, icon, JButton "OK",
// και JLabel

String answer = null; // για να αποθηκεύσει δεδομένα JField

answer = JOptionPane.showInputDialog(null, "Insert name",
         "title", JOptionPane.DEFAULT_OPTION);

// ή πιο απλά, θέτω σα παράμετρο μόνο το μήνυμα του JLabel
answer = JOptionPane.showInputDialog("Insert name");
```

JOptionPane.showOptionDialog(…)

Το πιο ολοκληρωμένο παράθυρο από τα παραπάνω, το οποίο ουσιαστικά τα συνδυάζει.

```
JCheckBox check = new JCheckBox("Tick me");

// Κάνω πίνακα τύπου Object ώστε να μπορώ να βάλω δεδομένα από
διαφορετικούς τύπους (γιατί κάθε τύπος δεδομένων στη java είναι
υπόκλαση της Object, άρα κληρονομεί τις ιδιότητες της
Object[] options2 = {'e', 2, 3.14, 4, 5, "TURTLES!", check};

int x = JOptionPane.showOptionDialog
                // 1. Parent component
               (null,
                // 2. Κείμενο JLabel
               "Returns position of your choice on the array",
                // 3. Τίτλος του frame
                "Click a button pop up",
                // 4. optionType: Για καθορισμό κουμπιών
                JOptionPane.DEFAULT_OPTION,
                // 5. messageType: Είδος παραθύρου
                JOptionPane.INFORMATION_MESSAGE,
                // 6. Εικόνα ή εικονίδιο
                null,
                // 7. πίνακας με τα κείμενα των επιλογών
                //    AN null, ακολουθεί τη παράμετρο optionType
                //    AN <> null, παίρνει τιμές του options
                options,
                // 8. initial value - default choice
                options[0]);
```

Basics of Eclipse IDE

Δημιουργία νέου Java Project

Τέρμα αριστερά και πάνω, στη μπάρα επιλογών, διαλέγουμε File, πατάμε τη πρώτη επιλογή με τίτλο New, κι από την καρτέλα επιλογής που εμφανίζει διαλέγουμε Java Project. Ανοίγει αυτόματα παράθυρο ρυθμίσεων. Αρκεί μονάχα ένας τίτλος στο πεδίο Project Name για να δημιουργηθεί ένας φάκελος στην αριστερή στήλη με τίτλο Package Explorer (αφότου φυσικά, μετά την ονομασία του έργου, πατήσουμε το κουμπί Finish). Αν κατά τη δημιουργία του παρατηρήσουμε ότι υπάρχει μέσα του ένα αρχείο με όνομα “module-info.java”, κάνουμε δεξί κλικ πάνω του και Delete.

Άνοιγμα ήδη υπάρχοντος Java Project

Τέρμα αριστερά και πάνω, στη μπάρα επιλογών, διαλέγουμε File.

- ➔ … αν έχει επιλογή Open projects from file system, το επιλέγουμε.
- ➔ … αν όχι (δηλαδή αν δεν έχει ψηλά επιλογή Open projects from file system), κοιτάμε πιο χαμηλά στις επιλογές μας και πατάμε πάνω στο Import. Ανοίγει αυτόματα ένα παράθυρο με διάφορους φακέλους, από τους οποίους κάνουμε διπλό κλικ στον General, ώστε να δούμε τους υπό – φακέλους μέσα σε αυτόν. Από τις διαθέσιμες κατηγορίες, διαλέγουμε Existing project into workspace, και κάνουμε διπλό κλικ.

Σε κάθε περίπτωση από εκεί και πέρα, πρέπει να δούμε ένα νέο παράθυρο να ανοίγει, το οποίο μας δίνει τη δυνατότητα, στο πεδίο Select root directory, να εισάγουμε το path που έχουμε αποθηκεύσει το πρότζεκτ μας (πατώντας το κουμπί Browze…). Αφού βρούμε το τη τοποθεσία του πρότζεκτ στο σύστημα αρχείων μας, την επιλέγουμε, πατάμε OK, και βλέπουμε ότι πλέον το πεδίο Select root directory αντί να είναι κενό έχει τη διαδρομή για να ανοίξει το αρχείο μας. Πατάμε Finish στο παράθυρο (κάτω δεξιά).

Δημιουργία κλάσης

Για να μπορεί να δημιουργηθεί μια κλάση, πρέπει να ανήκει σε κάποιο πρότζεκτ πρώτα. Οπότε, αφού δημιουργήσουμε ή ανοίξουμε ένα ήδη υπάρχον πρότζεκτ, δείχνουμε με τον κέρσορα πάνω στο όνομα του (στη στήλη Package Explorer, αριστερά στην οθόνη). Κάνουμε δεξί κλικ πάνω στο όνομα, διαλέγουμε New, και από εκεί επιλέγουμε Class. Ανοίγει ένα παράθυρο στο οποίο αρκεί να συμπληρώσω μονάχα το πεδίο Name και να πατήσω Finish για να παραχθεί ένα νέο αρχείο – κλάση μέσα στο πρότζεκτ μου.

Πριν πατήσω Finish, μπορώ προαιρετικά να επιλέξω superclass (κάνοντας Browze…) αν πρόκειται για κλάση – παιδί μιας άλλης (αν όχι, θεωρείται αυτόματα ότι κάθε κλάση είναι παιδί της Object), ή να επιλέξω στη λίστα επιλογών το Constructors for superclass, ή μπορώ να επιλέξω στη λίστα επιλογών το “public static void main (String args[])” αν φτιάχνω μια Main.