Προγραμματισμός ΙΙ (Java)

10. Πολυνηματικές εφαρμογές – Τεκμηρίωση κώδικα

Multithreading

- Multi-processing
  - Πολλές διεργασίες ταυτόχρονα
  - Κάθε διεργασία έχει το δικό της χώρο μνήμης ή ενδέχεται όλες να μοιράζονται και κάποιο χώρο μνήμης
  - Ουσιαστικά μία διεργασία εκτελείται κάθε φορά
  - Το λειτουργικό αναλαμβάνει τη διαχείριση
- Multithreading
  - Multiple points of execution (threads) within the same memory space

Threads - Νήματα

- Τα threads επιτρέπουν εργασίες να τρέχουν παράλληλα με την κύρια εφαρμογή
- Είναι διεργασίες που τρέχουν στον ίδιο χώρο διευθύνσεων: μοιράζονται τις μεταβλητές στιγμιοτύπων, αλλά διατηρούν και τοπικές μεταβλητές
- Χρησιμοποιούνται από το JVM σε συγκεκριμένες περιπτώσεις
  - Όταν φορτώνουμε μια εικόνα ή ένα ήχο μέσα από ένα applet.
  - Όταν καλούμε την update του αντικειμένου graphics
- Μπορούμε να δηλώσουμε και να χρησιμοποιήσουμε δικά μας νήματα.

Γιατί χρειάζομαι threads

- Single threaded application:
  - Ένα νήμα κάνει τα πάντα
  - Αν υπάρχουν ενέργειες να εκτελεστούν, θα περιμένουν τη σειρά τους
  - Αν μια ενέργεια θέλει χρόνο για να ολοκληρωθεί, η εφαρμογή θα φαίνεται ότι έχει κολλήσει
- Multithreaded application:
  - Ένα νήμα για κάθε εργασία
  - Αν ένα νήμα έχει αναλάβει μια αργή ενέργεια, τα υπόλοιπα νήματα θα συνεχίσουν να εκτελούνται
  - Αν η εφαρμογή τρέχει σε σύστημα με πολλούς επεξεργαστές, το λειτουργικό θα στείλει κάθε νήμα σε άλλο επεξεργαστή.

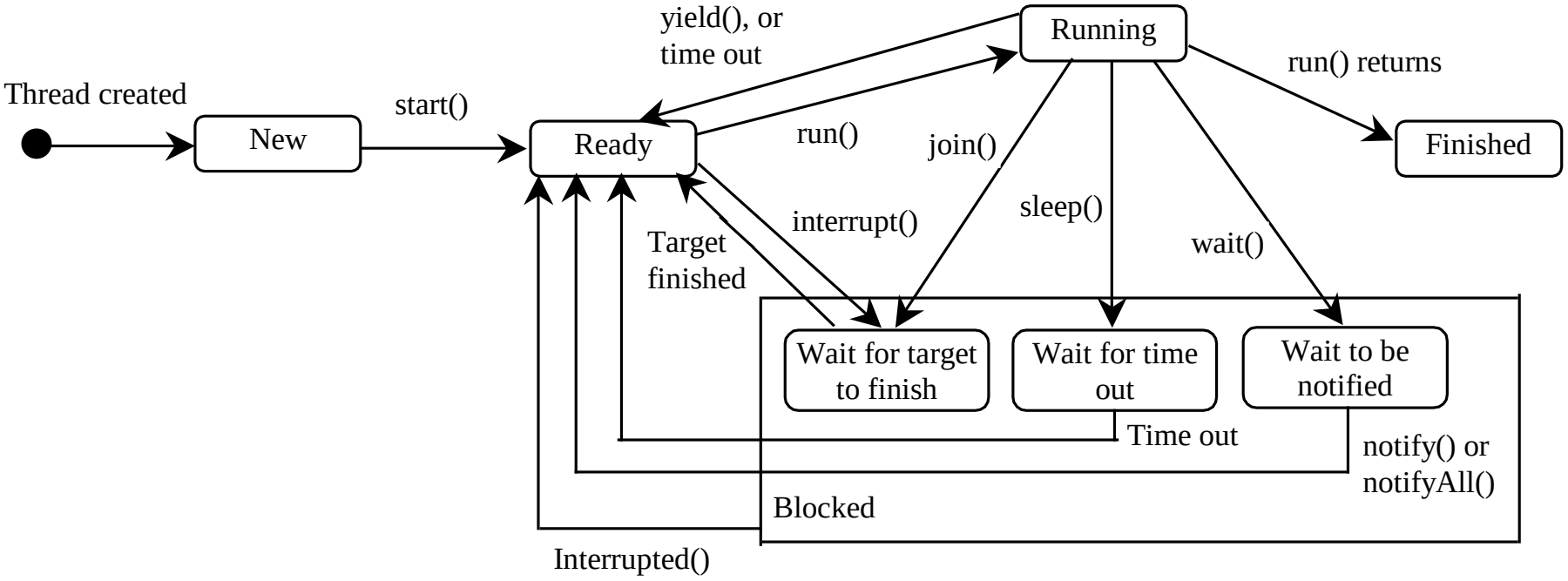
Πότε χρειάζομαι threads

- Όταν η εφαρμογή μου έχει GUI
  - Τα νήματα που αφορούν το GUI διαχειρίζονται τις ενέργειες του χρήστη και αναθέτουν τις εργασίες σε άλλα νήματα
  - Το GUI συνεχίζει να αποκρίνεται στις ενέργειες του χρήστη ακόμη κι αν τρέχουν άλλα νήματα
- Όταν η εφαρμογή μου πρέπει να αποκρίνεται ασύγχρονα (π.χ. Network based applications)
  - Τα δεδομένα που έρχονται από το δίκτυο μπορεί να έρχονται χωρίς ομαλή ροή
  - Ένα νήμα μπορεί να ελέγχει την πόρτα εισόδου δεδομένων
  - Όποτε έρθουν δεδομένα το νήμα ολοκληρώνει τη διαχείρισή τους ή την αναθέτει σε άλλο νήμα
  - Η εφαρμογή μας τρέχει σε άλλο νήμα

Πώς λειτουργεί ο μηχανισμός;

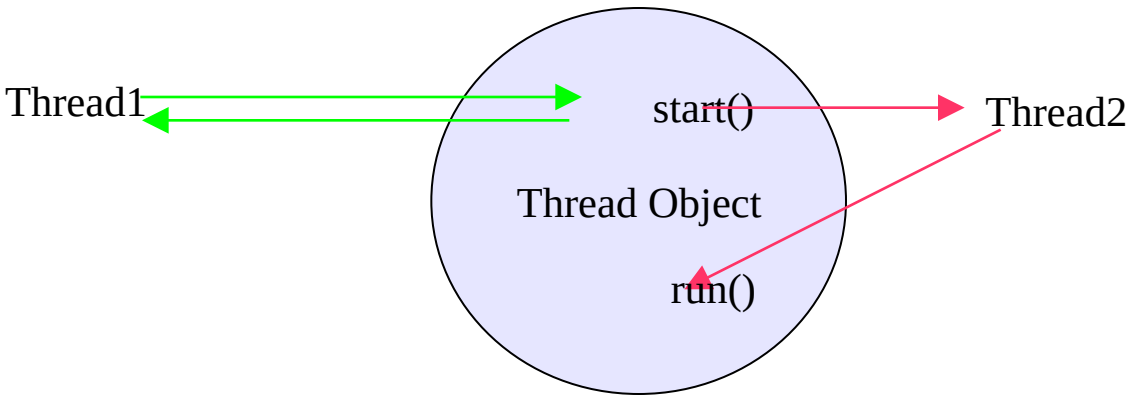
- Κάθε νήμα έχει το δικό του χώρο (virtual registers και calling stack)
- Ο "scheduler" αποφασίζει ποιο νήμα εκτελείται και πότε
- Το VM μπορεί να έχει το δικό του scheduler ή να χρησιμοποιεί το scheduler του λειτουργικού (το JVM έχει δικό του)
- Ο scheduler έχει μια λίστα με τα threads που είναι έτοιμα να τρέξουν (run queue) και μια λίστα με τα threads που περιμένου είσοδο (wait queue)
- Κάθε νήμα έχει μια προτεραιότητα. Τα threads υψηλής προτεραιότητας επιλέγονται κάθε φορά από την run queue
- Δεν μπορούμε να υποθέσουμε από πριν με ποια σειρά θα χρονοπρογραμματιστούν τα threads.

Καταστάσεις ενός thread



Πώς τρέχει ένα thread

- Ξεκινά με τη μέθοδο run() που περιέχει το βασικό σύνολο εντολών του νήματος
  - Ενεργοποιείται με το start()
- Τερματίζει όταν ολοκληρωθεί η run()
  - Αν θέλουμε να μην ολοκληρωθεί το νήμα, πρέπει να μπλοκάρουμε τη run (π.χ. με endless loop)
- Ένα thread καλεί κάποιο άλλο με την κλήση της start()



Ένα Thread για κάθε Task

- Ένα Thread μπορεί να είναι αντικείμενο της κλάσης Thread ή δικών μας κλάσεων που κληρονομούν τη Thread
- Συχνά οι δικές μας κλάσεις μπορεί να έχουν δική τους κληρονομικότητα, και ταυτόχρονα να θέλουμε να είναι threads
  - Τότε χρησιμοποιούμε το Runnable interface
  - Ένα interface με μοναδική μέθοδο την run()
- Χρειαζόμαστε και κάποιον (ένα dummy Thread object) να φτιάξει και να ξεκινήσει το νήμα μας

Τρόποι δήλωσης (1/2)

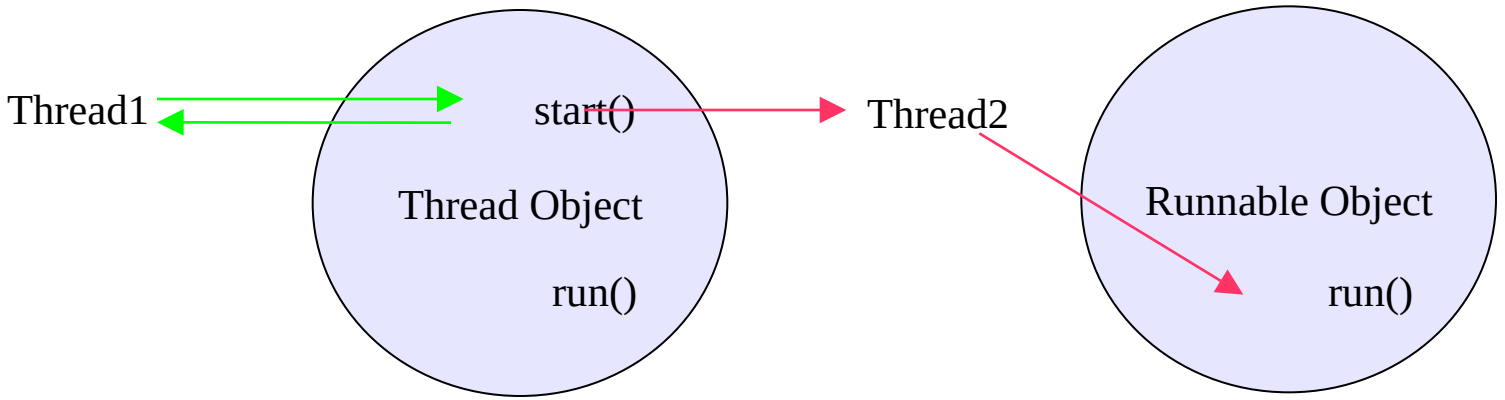
- Επέκταση της Thread

```
public class HelloThread extends Thread {
    public void run() {
        System.out.println("Hello from a thread!");
    }
    public static void main(String args[]) {
        (new HelloThread()).start();
    }
}
```

Τρόποι δήλωσης (2/2)

- Υλοποίηση της Runnable

```
public class HelloRunnable implements Runnable {
    public void run() {
      System.out.println("Hello from a thread!");
    }
    public static void main(String args[]) {
      (new Thread(new HelloRunnable())).start();
    }
}
```



Μέθοδοι

- Thread.yield(): όποτε καλείται δίνει προτεραιότητα στα υπόλοιπα threads εις βάρος του τρέχοντος.
- Thread.sleep(long x): σταματά προσωρινά (για x milliseconds) την εκτέλεση του νήματος
- final void otherThread.join(): διακόπτει το τρέχον νήμα μέχρι να τελειώσει το otherThread
- isAlive(): επιστρέφει true για Ready, Blocked, or Running state
- interrupt(): αν το thread είναι Ready or Running, σταματά. Αν είναι blocked ξυπνά και γίνεται Ready ενώ πετά μια java.io.InterruptedException
- isInterrupt(): ελέγχει αν το νήμα εξακολουθεί να είναι σταματημένο ή ξαναξεκίνησε

Πώς τερματίζω ένα thread

- Στη Java 1.1. η κλάση Thread είχε μέθοδο stop()
  - Ένα thread σταματούσε ένα άλλο thread καλώντας τη stop()
  - Είχε και suspend() και resume()
  - Αυτό όμως οδηγούσε σε αμοιβαίους αποκλεισμούς (deadlocks)
  - Πλέον δε χρησιμοποιούνται (deprecated)
- Αν θέλουμε να τερματίσουμε το thread περιμένουμε να τελειώσει η run
  - Αν θέσω ένα boolean γνώρισμα στο thread και φροντίζω να ελέγχω την τιμή του εντός της run, μπορώ να ελέγξω πότε θα τελειώσει
  - Μέσω μιας μεθόδου set αλλάζω την τιμή στο γνώρισμα (από άλλο νήμα που τρέχει) και τερματίζω τη run() βγαίνοντας από το infinite loop

Κοινή χρήση;

- Υπάρχουν θέματα συγχρονισμού (ταυτόχρονης πρόσβασης στον κοινόχρηστο χώρο) που πρέπει να ληφθούν υπόψη.
- Για να αντιληφθούμε καλύτερα τα νήματα θεωρούμε δύο μόνο αντικείμενα που μοιράζονται τον κοινό χώρο: Το αντικείμενο Thread της Java που ελέγχει το χώρο (***ελεγκτής***) και ένα αντικείμενο που περιέχει τη μέθοδο που θα εκτελέσουμε παράλληλα (***νήμα***).
- Το αντικείμενο νήμα έχει μια μέθοδο run η οποία:
  - πρέπει να είναι public, να μην παίρνει ορίσματα, να μην επιστρέφει τιμές και να μην παράγει εξαιρέσεις.
- Το αντικείμενο ελεγκτής φτιάχνει το νήμα, το ξεκινά (καλώντας τη run) και το καταστρέφει μόλις τελειώσει.

Δήλωση μιας τάξης νήματος

- Μια τάξη νήμα υλοποιεί τη διεπαφή Runnable (ορίζει μια μέθοδο run).
- Περιέχει μια σημαία (flag) επιτυχούς ολοκλήρωσης της διεργασίας.
- Μέσα στη run υλοποιεί την παράλληλη διεργασία και στο τέλος της ενημερώνει τη σημαία.

```
class Animation implements Runnable {
  boolean finished;
  public void run( ) {
        ...
        finished=true;
  }
}
```

Έλεγχος ενός νήματος

- Στην κυρίως εφαρμογή φτιάχνουμε το αντικείμενο νήμα
  Animation happy = new Animation();
- Φτιάχνουμε το αντικείμενο ελεγκτή και του αναθέτουμε να παρακολουθεί το νήμα
  Thread myThread = new Thread(happy);
- Ο ελεγκτής ξεκινά το νήμα
  myThread.start( );
- Η τελευταία εντολή εκτελεί τη run του αντικειμένου happy (νήμα)
- Για να ξεκινήσουμε ένα δεύτερο νήμα (οποιασδήποτε τάξης) ο ελεγκτής συνδέεται με αυτό και το ξεκινά
  myThread = new Thread(new anotherThreadClass());
  myThread.start( );

Παύση και επανεκκίνηση νήματος

- Η μέθοδος sleep( ) σταματά το νήμα για κάποια milliseconds, δημιουργώντας ένα InterruptedException αν το νήμα είναι ήδη σταματημένο
- Η μέθοδος interrupt( ) ξαναξεκινά ένα νήμα που ήταν αδρανές

```
try {
   sleep( 500 ); // περιμένει 0,5"
}
catch ( InterruptedException e ) {
// αν κάποιος καλέσει την interrupt για το νήμα
}
```

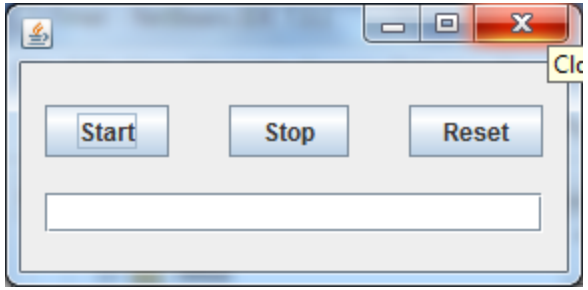
- Οι μέθοδοι wait() και notify( ) συντονίζουν την εκτέλεση δύο ή περισσοτέρων νημάτων

Τερματισμός του Thread

- Πρέπει να μπλοκάρουμε τη run() σε ένα loop
- Πρέπει να φροντίσουμε να μπορεί να επιστρέψει βγαίνοντας από το loop

```
public void stopThread(){
   this.stopThread=true;
}
public void run( ) {
     ….
     while (true) {
        …
        if (stopThread)
           return;

     }
}
```

Παράδειγμα



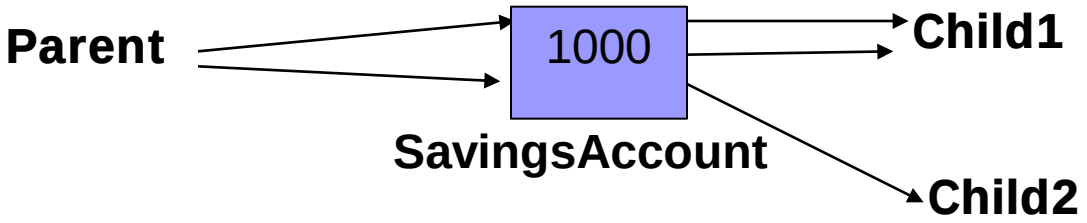
- Δημιουργήστε ένα GUI με 3 buttons και ένα text field
- Με το start: θα ξεκινά ένα thread που θα τυπώνει την τρέχουσα ώρα και ημερομηνία στο text field κάθε 1 sec.
- Με το stop: το thread θα σταματά
- Με το reset: θα καθαρίζει το text field

Ταυτόχρονη πρόσβαση σε δεδομένα

- Δύο threads επιχειρούν να ενημερώσουν τα ίδια δεδομένα (π.χ. να προσθέσουν/διαγράψουν/διαβάσουν αντικείμενα από μια συλλογή)
- Το τελικό αποτέλεσμα εξαρτάται απόλυτα από τον scheduler και δεν μπορούμε να το προκαθορίσουμε
- Όσο το ένα thread γράφει κάτι στα δεδομένα, μπορεί να τα «κλειδώσει» μέχρι να τελειώσει
- Κανείς εκτός από το thread δεν μπορεί να δει τα κλειδωμένα δεδομένα
- Ο μηχανισμός αυτός στην Java λέγεται synchronization

Συγχρονισμός νημάτων

- Έστω ένας τραπεζικός λογαριασμός με ένα αρχικό ποσό, ένα νήμα που κάνει κατάθεση και δύο νήματα που κάνουν ανάληψη




```
public class SavingsAccount
{
    private float balance;
    public synchronized void withdraw(float anAmount)
    {
            if ((anAmount>0.0) && (anAmount<=balance))
                    balance = balance - anAmount;
    }
    public synchronized void deposit(float anAmount)
    {
            if (anAmount>0.0)
                    balance = balance + anAmount;
    }
}
```

Παράδειγμα

- Ο Parent καταθέτει χρήματα κάθε 5" καλώντας την deposit
- Τα Child αφαιρούν χρήματα κάθε 3" καλώντας την withdraw

```
class Parent implements Runnable {
   SavingsAccount account;
   String name;
   boolean bankrupted = false;
   public Parent(String n, SavingsAccount s) {
        this.name = n;
        this.account = s;
    }
    public void run() {
       while (true) {
         try {
             Thread.sleep(5000);
         } catch (InterruptedException ex) {...}
         account.deposit(this.name, 1000);
         if (bankrupted) {
               return;
         }
      }
   }
```

```
class Child implements Runnable {
   SavingsAccount account;
   String name;
   boolean foundajob= false;
   public Child(String n, SavingsAccount s) {
        this.name = n;
        this.account = s;
    }
    public void run() {
       while (true) {
         try {
             Thread.sleep(3000);
         } catch (InterruptedException ex) {...}
         account.withdraw(this.name, 200);
         if (foundajob) {
               return;
         }
      }
   }
```

Η main

- Φτιάχνει
  - το λογαριασμό
  - τον Parent
  - τα Child
- Ξεκινά τα νήματα

```
public class FamilyDemo {
    public static void main(String args[]) throws InterruptedException{
        SavingsAccount acc=new SavingsAccount();
        acc.deposit("bank", 1000);
        Parent p=new Parent("Bill",acc);
        Child c1=new Child("John",acc);
        Child c2=new Child("Mary", acc);
        new Thread(p).start();
        new Thread(c1).start();
        new Thread(c2).start();
    }
}
```

Συγχρονισμένες μέθοδοι

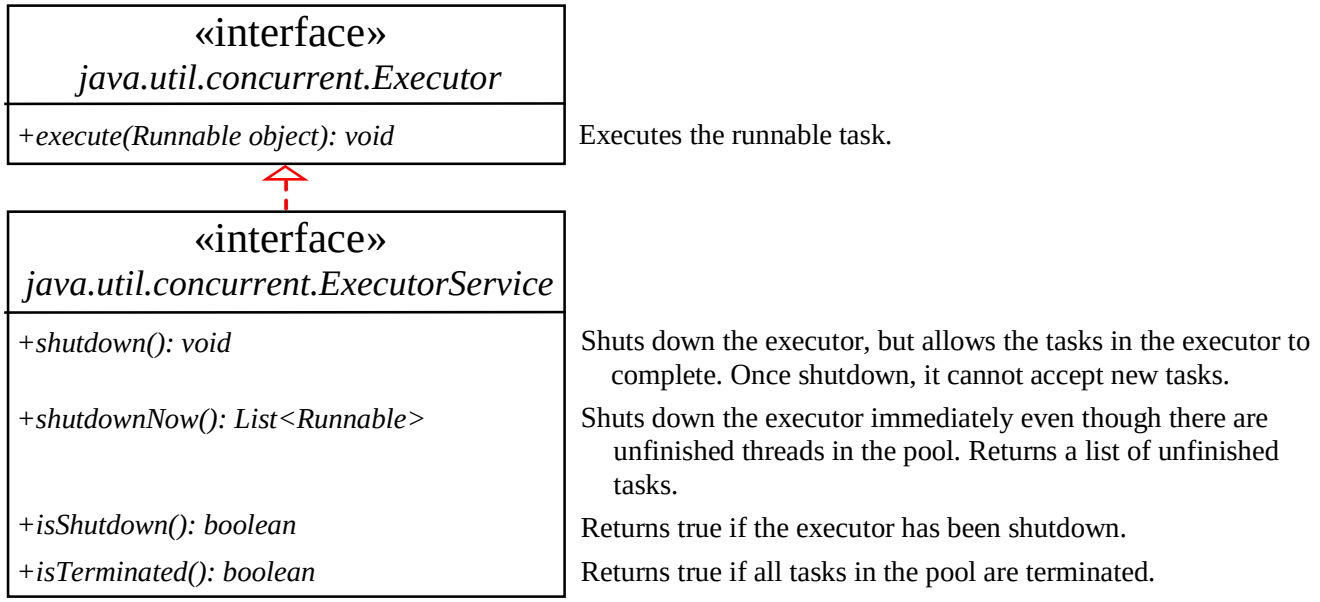
```
public class SavingsAccount {

    private float balance;

    public synchronized void withdraw(String name, float anAmount) throws InterruptedException {
        if ((anAmount > 0.0) && (anAmount <= balance)) {
            balance = balance - anAmount;
            System.out.println(name + " withdraws " + anAmount + " euro.");
            System.out.println("Current balance:" + balance);
            notify();
        } else {
            wait();
        }

    }

    public synchronized void deposit(String name, float anAmount) throws InterruptedException {
        if (anAmount > 0.0) {
            balance = balance + anAmount;
            System.out.println(name + " deposits " + anAmount + " euro.");
            System.out.println("Current balance:" + balance);
            notify();
        } else {
            wait();
        }

    }
}
```

Συνεπώς

- Έχουμε withdraw κλήσεις από τα Child που εξυπηρετούνται όσο υπάρχει υπόλοιπο
- Μόλις αδειάσει ο λογαριασμός η τελευταία κλήση της withdraw παγώνει (wait)
- Μια κλήση της deposit από τον parent καλεί τη notify και ξεπαγώνει τη συγχρονισμένη μέθοδο withdraw
- Πότε παγώνει η deposit και πως βγαίνουμε από την κατάσταση αυτή;

Thread Pools

- Αν ξεκινάμε ένα thread για κάθε task είναι δαπανηρό.
- Θέλουμε μια δεξαμενή με threads, που να μας δίνει ένα thread αν αυτό έχει τελειώσει την προηγούμενη εργασία
- Η υλοποίηση προσφέρεται μέσα από το Executor interface το οποίο ελέγχει τα Runnable αντικείμενα σε ένα thread pool
- Το ExecutorService είναι ένα subinterface του Executor



Δημιουργία εκτελεστή (Executor)

- Φτιάχνουμε Executor αντικείμενα μέσα από static methods της Executors

java.util.concurrent.Executors	
+newFixedThreadPool(numberOfThreads: int): ExecutorService	Creates a thread pool with a fixed number of threads executing concurrently. A thread may be reused to execute another task after its current task is finished.
+newCachedThreadPool(): ExecutorService	Creates a thread pool that creates new threads as needed, but will reuse previously constructed threads when they are available.

```
ExecutorService executor = Executors.newFixedThreadPool(5);
Runnable worker = new MyClassThatImplementsRunnable();
executor.execute(worker);
```

```
public class MyClassThatImplementsRunnable implements Runnable {
public void run() {
        System.out.println(Thread.currentThread().getName() + " Start.");
        processCommand();
        System.out.println(Thread.currentThread().getName() + " End.");
    }
}
```

Τεκμηρίωση κώδικα

javadoc

Σχόλια (Comments)

- Σχόλια είναι σημειώσεις σε φυσική γλώσσα και όχι source code.
- Τα Comments χρησιμοποιούνται για να:
  - Τεκμηριώνουμε: τον σκοπό και τους στόχους του προγράμματος, τον author, ιστορικά στοιχεία για τις αναθεωρημένες εκδώσεις του code (revision history), copyright notices, κτλ.
  - Περιγράφουμε: fields, constructors, και methods
  - Την επεξήγηση μη καθαρών σημείων του code
  - Προσωρινός σχολιασμός μέρους του code, σαν υπενθύμιση για μελλοντική χρήση

Μορφές Σχολίων

- Ένα “block” σχόλιο γράφεται ανάμεσα στα σύμβολα /* και */. Για παράδειγμα:

```
/*  Exercise 5-2  for Java Methods
    Author: Miss Brace
    Date: 3/5/2010
    Rev. 1.0                    */
```

- Ένα single-line σχόλιο γράφεται μετά από το σύμβολο // μέχρι το τέλος της γραμμής. Για παράδειγμα:

```
wt *= 2.2046;          //  Convert to kilograms
```

Σχόλια Τεκμηρίωσης

- Μπορούμε να χρησιμοποιήσουμε το special utility program javadoc για να παράξουμε αυτόματα τεκμηρίωση του source code μας σε HTML format.
- Τέτοιας μορφής σχόλια πρέπει να προηγούνται των: classes, methods, ή fields
- Μπορούν να χρησιμοποιηθούν special javadoc tags:
  - @param – περιγράφει τις παραμέτρους μιας μεθόδου (method)
  - @return - περιγράφει την τιμή που επιστρέφονται από μια μέθοδο (method's return value)

javadoc Comments (συνέχ.)

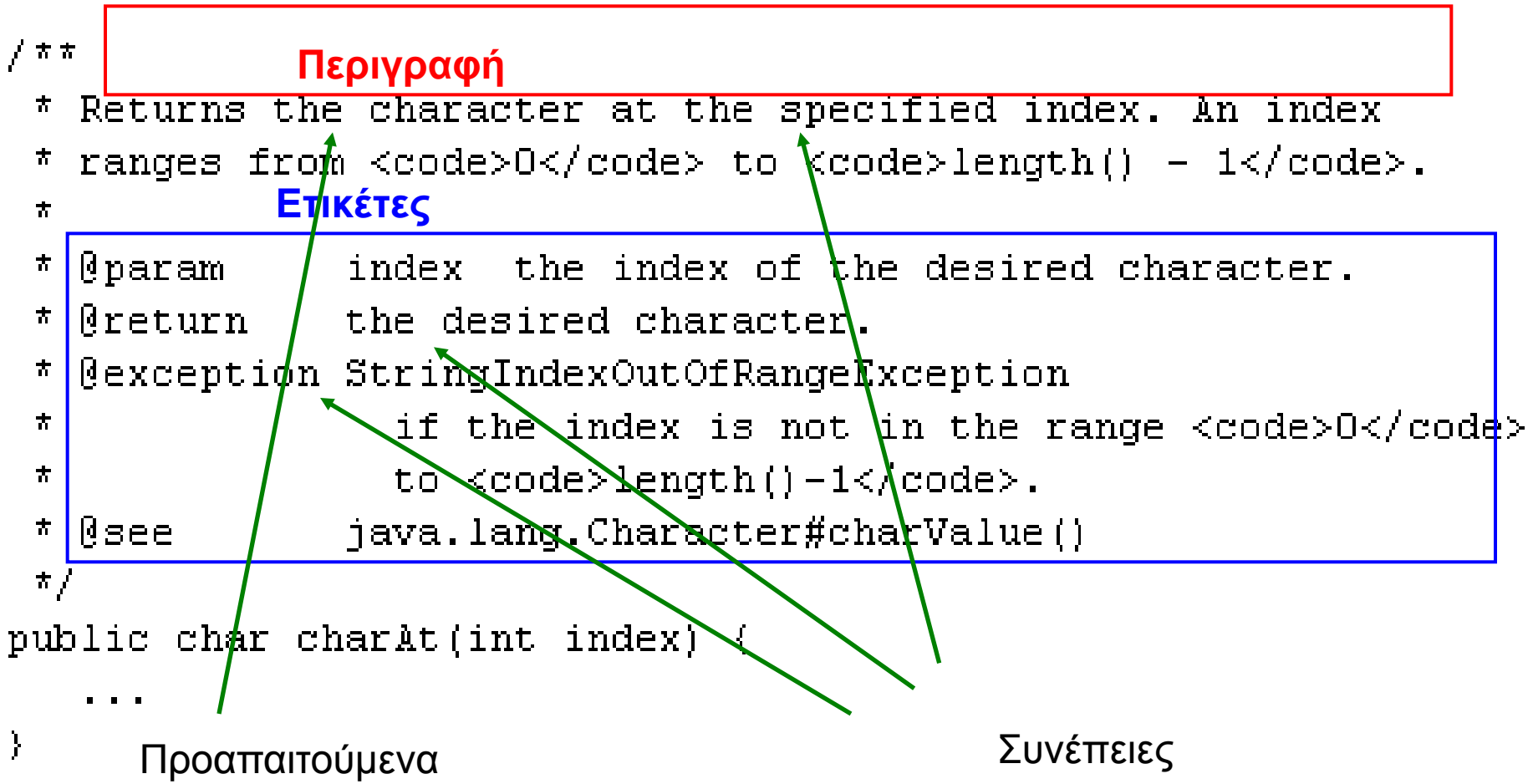
/** δείχνει ένα javadoc comment

```
/**
 *    Returns total sales from all vendors;
 *    sets <code>totalSales</code>
 *    to 0.
 *
 *    @return total amount of sales from all vendors
 */
```

Μπορεί να χρησιμοποιεί HTML tags

Common style

Δομή σχολίων μεθόδου



Ετικέτες Μεθόδων

- @see
- @since
- @deprecated
- @param
- @return
- @throws / @exception
- {@link}
- {@linkplain}
- {@inheritDoc}
- {@docRoot}

Παράδειγμα

```
/**
 * Sample.java – a class that demonstrates the use of javadoc
 * comments.
 * @author Ruth Dannenfelser
 * @version 2.0
 * @see Sample2
 */
 public class Sample extends Sample2 {
  public String words;
  /**
   * Retrieve the value of words.
   * @return String data type.
   */
  public String getWords()
  {
     return words;
   }
```

```
   /**
    * Set the value of words.
    * @param someWords A variable of type String.
    */
    public void setWords(String someWords)
    {
       words = newWords;
    }
 }
```

Δημιουργία τεκμηρίωσης

javadoc –d doc src/package/*.java