Αντικειμενοστρεφής Προγραμματισμός (Object-Oriented Programming)

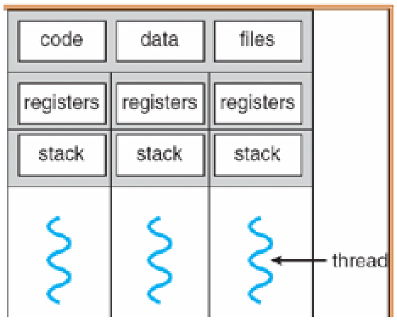
(CEID_NNY106)

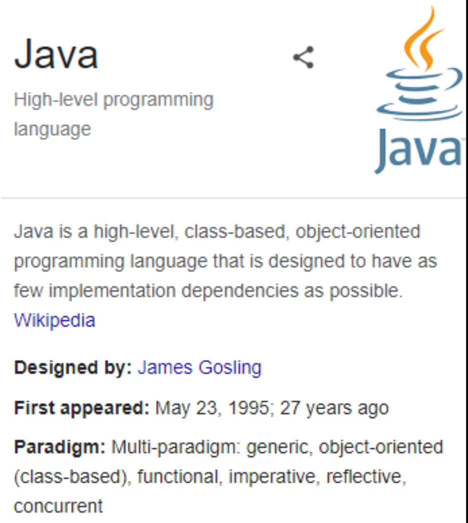
Ταυτόχρονος Προγραμματισμός

Concurrent Programming

Κύρια Πηγή

ΑΠΟ ΤΙΣ ΥΠΟΛΟΓΙΣΤΙΚΕΣ ΣΤΙΣ ΚΥΒΕΡΝΟ-ΦΥΣΙΚΕΣ ΔΙΕΡΓΑΣΙΕΣ ΚΑΙ ΤΟ IoT
Αντικείμενα και Υπηρεσίες
Κεφάλαιο: 10




Java

High-level programming language

Java is a high-level, class-based, object-oriented programming language that is designed to have as few implementation dependencies as possible. Wikipedia

Designed by: James Gosling

First appeared: May 23, 1995; 27 years ago

Paradigm: Multi-paradigm: generic, object-oriented (class-based), functional, imperative, reflective, concurrent

Kleanthis Thramboulidis
Prof. of Software and System Engineering
University of Patras
https://sites.google.com/site/thramboulidiskleanthis/

Concurrent programming is difficult

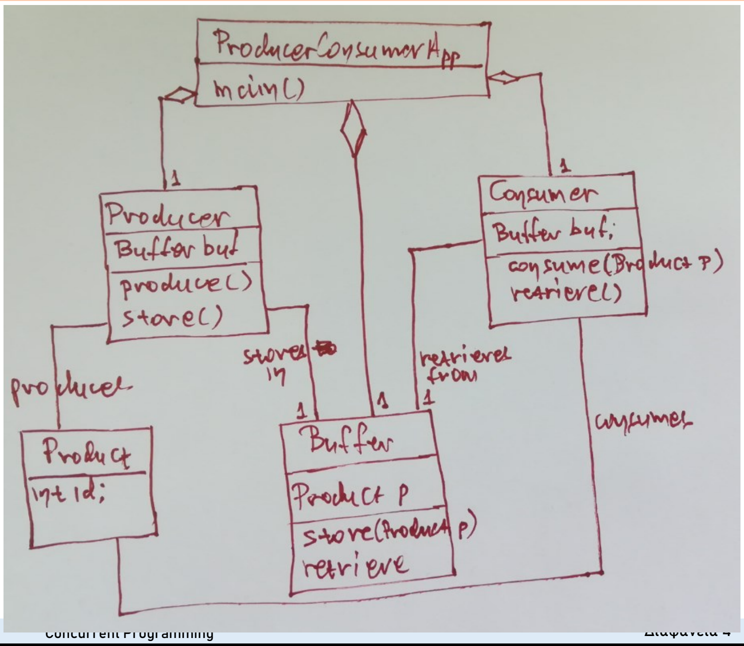
- "It is widely acknowledged that concurrent programming is difficult. Yet the imperative for concurrent programming is becoming more urgent." *Edward A. Lee, IEEE Computer 2006.*
- "humans are quickly overwhelmed by concurrency and find it much more difficult to reason about concurrent than sequential code. Even careful people miss possible interleavings among even simple collections of partially ordered operations." *Sutter and Larus, ACM Queue.*

Οργάνωση Διάλεξης

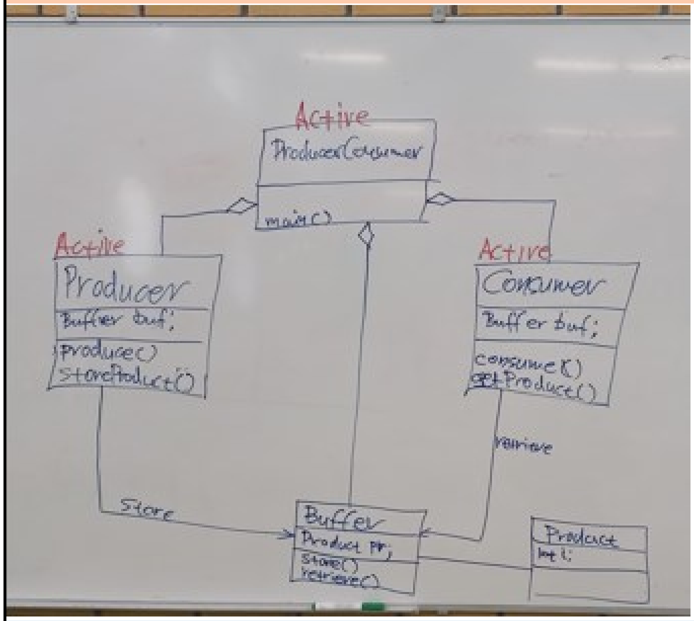
- **Producer/Consumer**
- Η κλάση Thread
- PingPongApp
- Κύκλος ζωής Νήματος
  - Thread Priorities/ Thread Scheduling
- Runnable Interface
- CounterTest Example

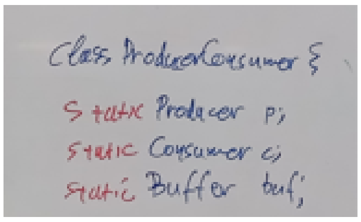
Producer-Consumer

- Αποτελεί την **αφαιρετική διατύπωση ενός συνόλου προβλημάτων**
- Ο παραγωγός παράγει αγαθά τα οποία αποθηκεύει μέχρι να είναι έτοιμος ο καταναλωτής να τα καταναλώσει.
- Ο καταναλωτής παίρνει αγαθά από την αποθήκη και τα καταναλώνει.

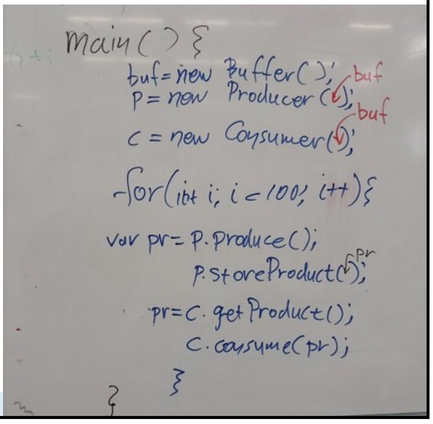


Producer/Consumer – 1st Draft Implementation



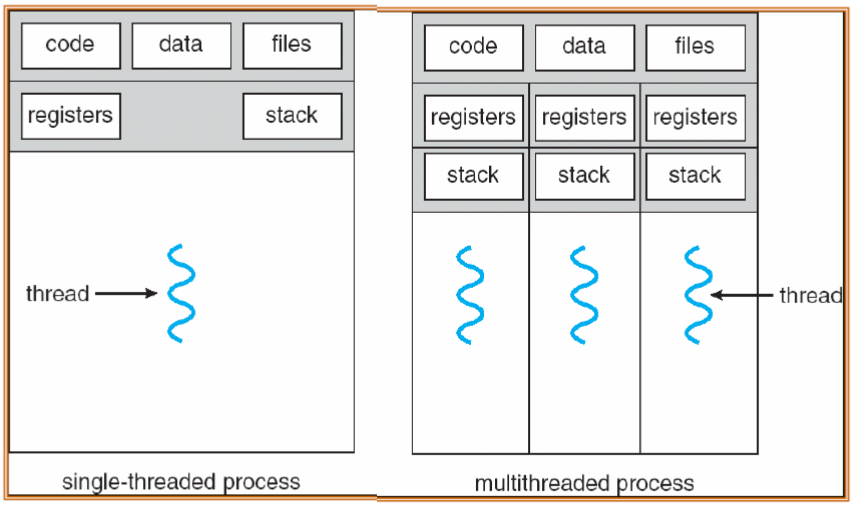


```
Class ProducerConsumer {
  static Producer p;
  static Consumer c;
  static Buffer buf;
```



```
main() {
      buf=new Buffer();  buf
      p=new Producer(); buf
      c=new Consumer();

     for(int i; i<100; i++){
    var pr=p.produce();
           p.storeProduct(); pr
      pr=c.getProduct();
           c.consume(pr);
        }
  }
```

Process and Thread

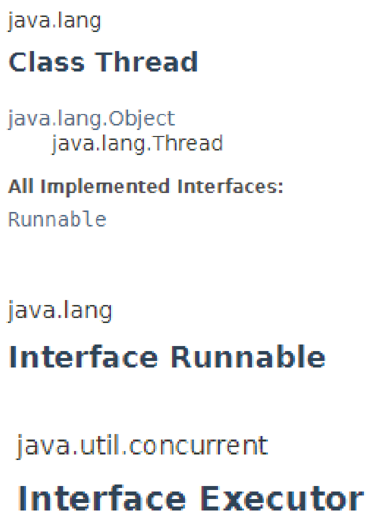


Οργάνωση Διάλεξης

- Producer/Consumer
- **Η κλάση Thread**
- PingPongApp
- Κύκλος ζωής Νήματος
  - Thread Priorities/ Thread Scheduling
- Runnable Interface
- CounterTest Example

Threads in Java

- A **Thread** is a thread of execution in a program
- Η VJM επιτρέπει σε μία εφαρμογή να έχει πολλά νήματα που εκτελούνται παράλληλα
- Η Java υποστηρίζει τα νήματα με
  - ειδικές κατασκευές
    - κλάσεις της βασικής βιβλιοθήκης
    - interfaces της βασικής βιβλιοθήκης



java.lang

Class Thread

java.lang.Object
    java.lang.Thread

All Implemented Interfaces:

Runnable

java.lang

Interface Runnable

java.util.concurrent

Interface Executor

Με την εκκίνηση της VJM υπάρχει ένα non-deamon thread που καλεί την main

Κλάση Thread

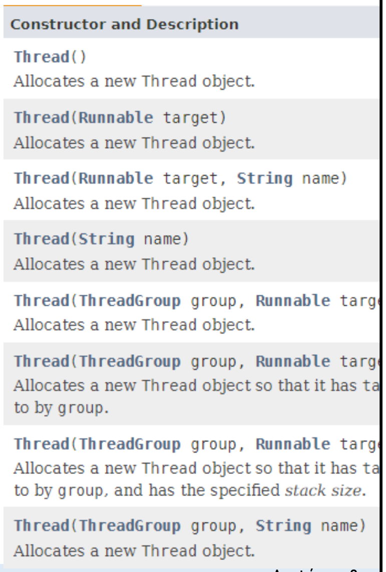
```
public class Thread
extends Object
implements Runnable
```

Δημιουργοί

- public Thread()
  - Το σύστημα δίνει ονόματα Thread-1, Thread-2, ...
- public Thread(String threadName)
  - Ο προγραμματιστής ορίζει το όνομα του Thread

Παράδειγμα

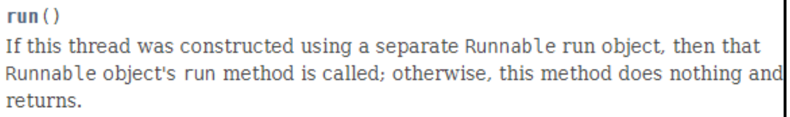
Thread ping = new Thread(new String("ping"));



Constructor and Description
`Thread()` Allocates a new Thread object.
`Thread(Runnable target)` Allocates a new Thread object.
`Thread(Runnable target, String name)` Allocates a new Thread object.
`Thread(String name)` Allocates a new Thread object.
`Thread(ThreadGroup group, Runnable targ` Allocates a new Thread object.
`Thread(ThreadGroup group, Runnable targ` Allocates a new Thread object so that it has ta to by group.
`Thread(ThreadGroup group, Runnable targ` Allocates a new Thread object so that it has ta to by group, and has the specified *stack size*.
`Thread(ThreadGroup group, String name)` Allocates a new Thread object.

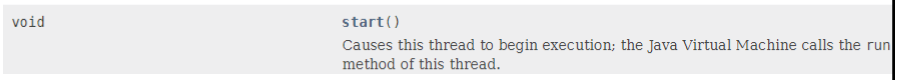
Κλάση Thread – Methods

- **run** - *edit-time*
  - προσδιορίζει το έργο του νήματος
  - Subclasses of Thread should override this method (Ορίζεται από τον προγραμματιστή).
  - Καλείται από το σύστημα σαν αποτέλεσμα της κλήσης της start
  - αν κληθεί άμεσα δεν υποστηρίζεται multithreading



`run()`
If this thread was constructed using a separate Runnable run object, then that Runnable object's run method is called; otherwise, this method does nothing and returns.

- **start** - *run-time*
  - Έχει ως αποτέλεσμα την ενεργοποίηση του νήματος
  - Η JVM καλεί την μέθοδο run του νήματος.
  - Ως αποτέλεσμα 2 νήματα είναι ενεργά ταυτόχρονα:
    - το νήμα που καλεί την start, για το οποίο ο έλεγχος επιστρέφει στο σημείο κλήσης της start, και
    - το νέο που δημιουργήθηκε, το οποίο εκτελεί την μέθοδο run του

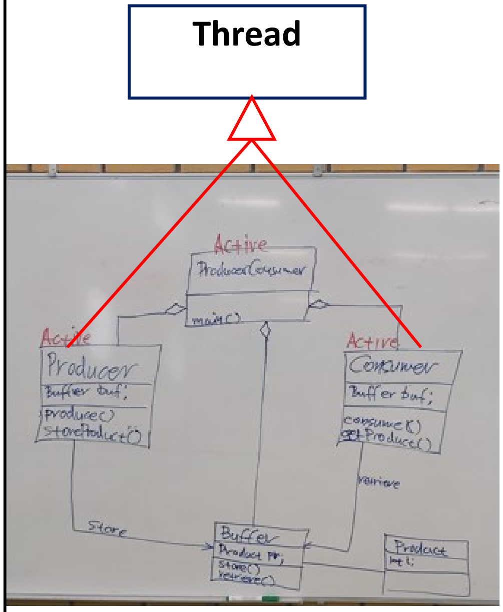


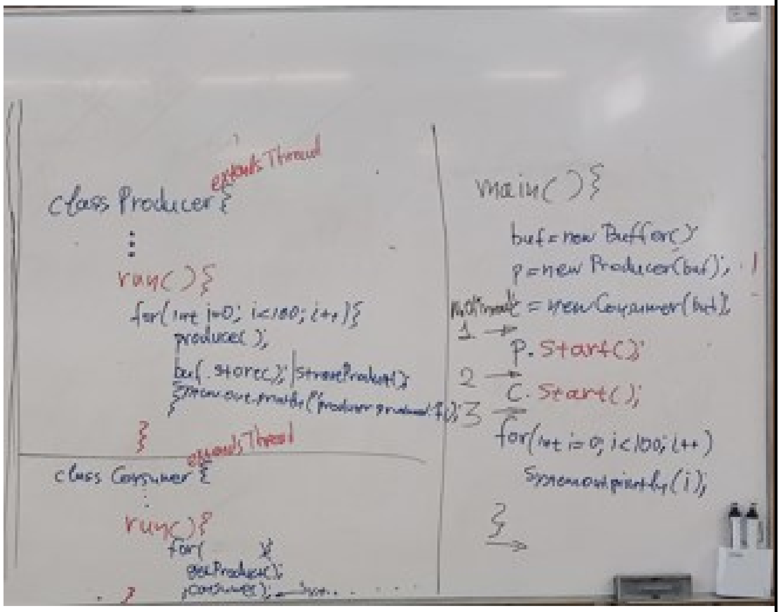
`void`	`start()` Causes this thread to begin execution; the Java Virtual Machine calls the run method of this thread.

Μέθοδοι κλάσης Thread

start - δημιουργεί νέο νήμα εκτέλεσης, ενεργοποιεί την run
interrupt - έχει σαν αποτέλεσμα τη διακοπή ενός νήματος εκτέλεσης.
checkAccess() Determines if the currently running thread has permission to modify this thread.
interrupted – επιστρέφει true αν το νήμα είναι σε διακοπή αλλιώς επιστρέφει false.
suspend - αναστέλλει την λειτουργία ενός νήματος εκτέλεσης *(Deprecated)*
resume - (called by another thread) ξαναρχίζει την εκτέλεση του νήματος *(Deprecated)*
stop - τερματίζει την εκτέλεση ενός νήματος. *(Deprecated)*
isAlive - επιστρέφει true εάν έχει κληθεί η start για ένα συγκεκριμένο νήμα και το νήμα δεν έχει τερματιστεί.
join Waits for this thread to die.
static void **yield**()
Causes the currently executing thread object to temporarily pause and allow other threads to execute.
static **currentThread** επιστρέφει μια αναφορά στο τρέχον νήμα
setName, getName και **setPriority**, **getPriority**
παρέχουν πρόσβαση στο όνομα και στην προτεραιότητα του νήματος

Producer/Consumer with Active objects





Producer-Consumer – 1η προσπάθεια υλοποίησης

```
class Buffer {
        int i=0;

        void store(int i) {
                this.i = i;
                }

        int retrieve(){
                return i;
                }
        }
```

Απαιτήσεις

- Ο παραγωγός δεν πρέπει να βάζει στην αποθήκη προϊόν αν δεν έχει καταναλωθεί το προηγούμενο (product overwrite).
- Ο καταναλωτής δεν πρέπει να καταναλώνει αγαθό που δεν είναι διαθέσιμο (αγαθό που έχει ήδη καταναλωθεί)

Οργάνωση Διάλεξης

- Producer/Consumer
- Η κλάση Thread
- **PingPongApp**
- Κύκλος ζωής Νήματος
  - Thread Priorities/ Thread Scheduling
- Runnable Interface
- CounterTest Example

PingPong App

```
class PingPongApp {
  public static void main(String [] args) {
    PingPong ping=new PingPong("PING",2);
    PingPong  pong = new PingPong("pong",4);

    ping.start();
    pong.start();

    for(int i=0;i<100;i++)
       System.out.print(i + " ");
     ping.stop();
     pong.stop();
    }
  }
```

```
class PingPong extends Thread {
    String word;                // λέξη για εκτύπωση
    int delay;                  // χρόνος ανάπαυσης

    PingPong(String wordToPrint, int pauseTime){
          word = wordToPrint;
          delay = pauseTime;
          }
    public void run() {
          try {
             for(int i=0;;i++){
                   System.out.print(word +"-" + i + " ");
                   sleep(delay);
                   }
          }
          catch (InterruptedException e){
                   return;     // τερματισμός του thread
                   }
          }
    }
```

Μέθοδος sleep

public static void **sleep**(long millis)
throws InterruptedException

Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds. The thread does not lose ownership of any monitors.

Parameters: millis
- the length of time to sleep in milliseconds.

Throws: InterruptedException
- if another thread has interrupted the current thread. The interrupted status of the current thread is cleared when this exception is thrown.

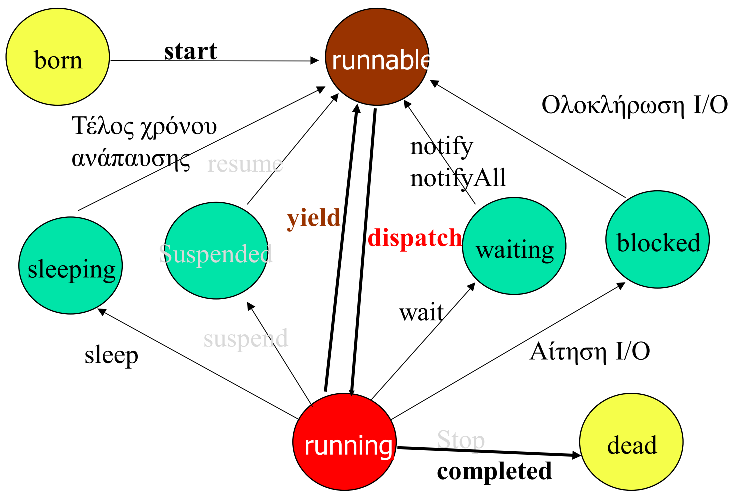
PingPong App – Μία πιθανή έξοδος

PING-0 ***pong-0*** 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 ***pong-1*** **PING-1** 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 ***pong-2*** **PING-2** ***pong-3*** 63 **PING-3** ***pong-4*** **PING-4** 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 ***pong-5*** **PING-5** ***pong-6*** 83 84 85 86 87 88 89 90 91 92 93 94 95 **PING-6** ***pong-7*** 96 97 98 99

Οργάνωση Διάλεξης

- Producer/Consumer
- Η κλάση Thread
- PingPongApp
- **Κύκλος ζωής Νήματος**
  - **Thread Priorities/ Thread Scheduling**
- Runnable Interface
- CounterTest Example

Κύκλος ζωής (life cycle) νήματος



Thread priorities

- It is possible to assign a thread priority
- the Thread class contains three integer priority constants
  - [1] MIN_PRIORITY
  - [5] NORM_PRIORITY
  - [10] MAX_PRIORITY
- the default thread priority is NORM_PRIORITY
- when a thread is created, it takes the priority of the thread which created it
- you can check a threads priority using getPriority()
- you can change a threads priority using setPriority()

Scheduling PingPong

```
public class PriorityTest2 {
        ....
        ping.setPriority(6);
        ping.start();
        pong.start();
```

```
public class PriorityTest3 {
        ....
        ping.setPriority(4);
        ping.start();
```

```
public class PriorityTest4 {
        ....
        ping.setPriority(7);
        pong.setPriority(7);
        ping.start();
```

```
public class PriorityTest5 {
        .....
        ping.setPriority(6);
        pong.setPriority(4);
        ping.start();
```

```
public class PriorityTest6 {
        .....
        ping.setPriority(6);
        pong.setPriority(4);
        ping.start();
        pong.start();
        for(int i=0;i<100;i++)
                System.out.print(i + "
    ");

        ping.stop();
        //pong.stop();
        }
```

PingPong App – Έξοδοι

- **PING-0 PING-1 PING-2 PING-3 PING-4 PING-5 PING-6 PING-7 PING-8 PING-9** 0 1 2 3 4 **pong-0** 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 **pong-1** 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 **pong-2** 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 **pong-3** 87 88 89 **pong-4** 90 91 92 93 94 95 96 97 98 99
- 0 **pong-0** 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 **pong-1** 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 **pong-2** 58 59 60 61 62 63 64 **pong-3** 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 **pong-4** 80 81 82 83 84 85 86 87 88 **pong-5** 89 90 91 92 93 94 95 96 97 98 99
- **PING-0 PING-1 PING-2 PING-3 PING-4 PING-5 PING-6 PING-7 PING-8 PING-9 pong-0 pong-1 pong-2 pong-3 pong-4 pong-5 pong-6 pong-7 pong-8 pong-9** 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99
- **PING-0 PING-1 PING-2 PING-3 PING-4 PING-5 PING-6 PING-7 PING-8 PING-9** 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99
- **PING-0 PING-1 PING-2 PING-3 PING-4 PING-5 PING-6 PING-7 PING-8 PING-9** 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 **pong-0 pong-1 pong-2 pong-3 pong-4 pong-5 pong-6 pong-7 pong-8 pong-9**

Thread Scheduling - Synchronization

- execution of multiple threads in some order on a single CPU system is called **scheduling**
- Thread Scheduling
  - has to do with how individual threads receive CPU attention,
- Thread synchronization
  - has to do with how multiple threads work together.

Producer Consumer example

Thread Scheduling - Rule of thumb

- At any given time,
  - the highest priority thread is running.
  - However, this is not guaranteed.
  - The thread scheduler may choose to run a lower priority thread **to avoid starvation**.
- use priority only to affect scheduling policy for efficiency purposes.
- Do not rely on thread priority for algorithm correctness.

Πηγή: **The Java™ Tutorial**

Οργάνωση Διάλεξης

- Producer/Consumer
- Η κλάση Thread
- PingPongApp
- Κύκλος ζωής Νήματος
  - Thread Priorities/ Thread Scheduling
- **Runnable Interface**
- CounterTest Example

Runnable interface

java.lang

Interface Runnable

- Αν η κλάση που αναπαριστά το νήμα μας είναι απόγονος άλλης κλάσης, η χρησιμοποίηση της κλάσης Thread είναι αδύνατη καθώς η Java δεν υποστηρίζει πολλαπλή κληρονομικότητα.
- Η δημιουργία νήματος ελέγχου σε τέτοιες περιπτώσεις γίνεται με χρήση του interface Runnable. Απλά δηλώνουμε την κλάση μας να υλοποιεί το interface Runnable

public interface **Runnable**

- The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread.
- The class must define a method of no arguments called **run**.
- This interface is designed to **provide a common protocol for objects that wish to execute code while they are active.** For example, Runnable is implemented by class Thread.
- **Being active simply means** that a thread has been started and has not yet been stopped.
- In addition, Runnable provides the means for a class to be active while not subclassing Thread.
- In most cases, the **Runnable interface should be used** if you are only planning to override the run() method and no other Thread methods.
  - This is important because **classes should not be subclassed unless** the programmer intends on modifying or enhancing the fundamental behavior of the class.
- A class that implements Runnable can run without subclassing Thread by instantiating a Thread instance and passing itself in as the target.

Runnable interface – Παράδειγμα

```
public class CountUp implements Runnable {
        public void run() {
                for(int i=0;i<10;i++)
                        System.out.println(“CountUp –“ + i );
                }
        }
```

```
CountUp cu = new CountUp( );
Thread t = new Thread(cu);
t.start();
```

CountUp ως απόγονος της Thread

```
public class CounterThread extends Thread {
   public void run() {
        for(int i=0;i<10;i++)
                System.out.println(“CountUp –“ + i );
        }
   }

class CountUpTest {
   public static void main(String [] args) {
        CounterThread cu = new CounterThread ();
        cu.start();
        }
   }
```

```
CountUp cu = new CountUp( );
Thread t = new Thread(cu);
t.start();
```

Οργάνωση Διάλεξης

- Producer/Consumer
- Η κλάση Thread
- PingPongApp
- Κύκλος ζωής Νήματος
  - Thread Priorities/ Thread Scheduling
- Runnable Interface
- **CounterTest Example**

The CounterTest Example App

```
public class CounterTest {
    private static Counter mc;
    private static CounterUser cu1, cu2;
    private static Thread[] myThread = new Thread[2];

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        mc = new Counter(0);
        cu1 = new CounterUser(mc);
        cu2 = new CounterUser(mc);
        System.out.println("Counter = " + mc.value() );
        myThread[0] = new Thread(cu1);
        myThread[1] = new Thread(cu2);
        myThread[0].start();
        myThread[1].start();
        System.out.println("Counter = " + mc.value() );
```

The Counter Class

```java
public class Counter {
        private int c;

        public Counter(int val){
            c= val;
        }
        public void increment() {
            c++;
        }

        public void decrement() {
            c--;
        }

        public int value() {
            return c;
        }

    }
```

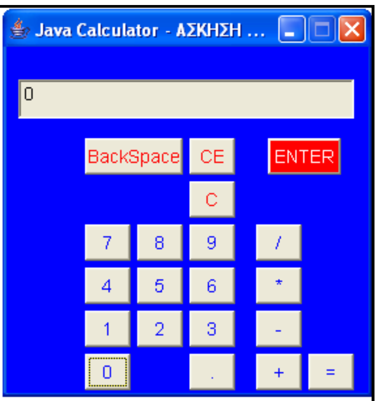
```java
public class CounterUser implements Runnable{
    private Counter counter;

    public CounterUser(Counter c){
        counter=c;
    }

    @Override
    public void run() {
        // TODO Auto-generated method stub
        for(int i=0;i<10;i++){
            counter.increment();
            try {
                Thread.sleep(2);
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
        //System.out.println(Thread.currentThread().
    }
```

Kinds of Threads

- user threads, daemon threads
  - by default, threads are user threads
  - Τα daemon threads παρέχουν γενικές υπηρεσίες και τυπικά δεν τελειώνουν ποτέ
- Τερματισμός εφαρμογής
  - the presence of **a user thread** keeps the application running
  - when the last user thread is finished, any daemon threads are stopped and the application is done
- related methods
  - setDaemon(true)
  - getDaemon()

The case of RPN Calculator

```
public class Calc {
        public static Operand op=new Operand();
        public static Operator add=new Adder();
        public static Operator sub=new Subtracter();
        public static Operator mul=new Multiplier();
        public static Operator div=new Divider();
        public static Operator eq=new Equals();
        public static Stack<Float> stk=new Stack<Float>();
        public static void main(String[] args) {
                new CalculatorGui(op);
        }
```



- JVM initially starts up with a single non-daemon thread, which typically calls the main method of some class.
  - if an application doesn't start any threads itself, the JVM will exit as soon as main terminates.
- **Why Calc doesn't terminate?**
  - AWT encapsulates asynchronous event dispatch machinery to process events AWT or Swing components can fire.
- The exact behavior of this machinery is implementation-dependent.
  - In particular, it can start **non-daemon helper threads** for its internal purposes. In fact, these are the threads that prevent Calc from exiting.

Threads in Java

- Each thread is associated with an instance of the class Thread.
- Two basic strategies for using Thread objects to create a concurrent application.
  - To directly control thread creation and management, simply instantiate Thread each time the application needs to initiate an asynchronous task.
  - To abstract thread management from the rest of your application, pass the application's tasks to an *executor*.

Executors are considered high-level concurrency objects.

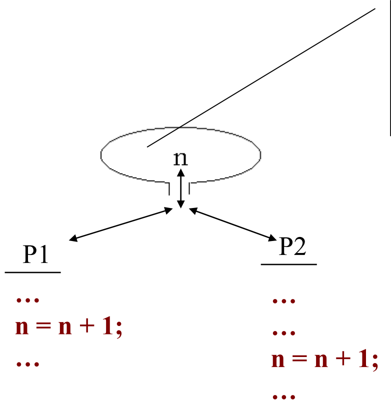
Executor Interface

`public interface Executor`

An object that executes submitted `Runnable` tasks. This interface provides a way of decoupling task submission from the mechanics of how each task will be run, including details of thread use, scheduling, etc. An `Executor` is normally used instead of explicitly creating threads. For example, rather than invoking `new Thread(new(RunnableTask())).start()` for each of a set of tasks, you might use:

```
 Executor executor = anExecutor;
 executor.execute(new RunnableTask1());
 executor.execute(new RunnableTask2());
 ...
```

Interleaving



Arbiter (Διαιτητής)
μηχανισμός σε χαμηλό επίπεδο (υλικό) **που διασφαλίζει τον αμοιβαίο αποκλεισμό** κατά την προσπέλαση μιας μεμονωμένης λέξης της μνήμης.

n = n + 1
Load n
Add 1
Store n

- Η P ακολουθεί μία οποιαδήποτε από τις ακολουθίες εκτέλεσης που προκύπτουν από την διαπλοκή των ακολουθιών εκτέλεσης των P1 και P2