Sincronizzazione con Java

Corse critiche e sincronizzazione

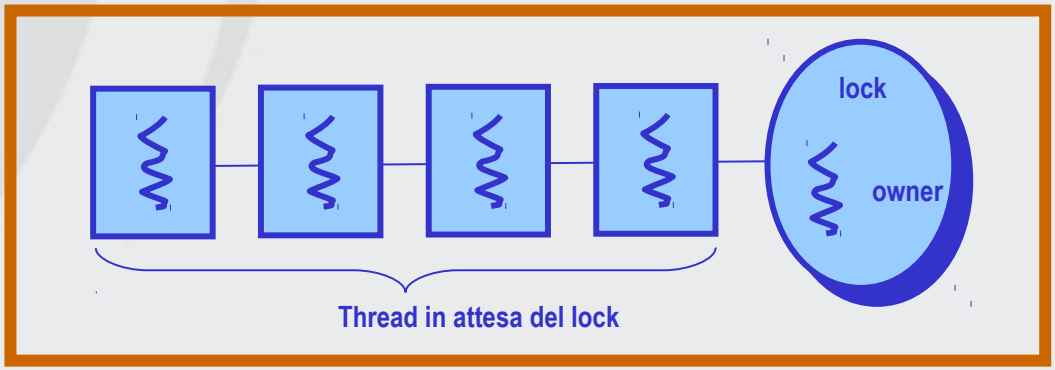
- Java implementa un meccanismo simile al monitor per garantire la sincronizzazione fra *thread*
- Ogni oggetto ha un lock associato ad esso
  - Nelle classi possono essere definiti metodi `synchronized`

```
synchronized int myMethod()
```

  - Un solo thread alla volta può eseguire un metodo synchronized
  - Se un oggetto ha più di un metodo sincronizzato, comunque uno solo può essere attivo

Metodi synchronized

- La chiamata di un metodo sincronizzato richiede il “possesso” del lock
- Se un thread chiamante non possiede il lock (un altro thread ne è già in possesso), il thread viene sospeso in attesa del lock
- Il lock è rilasciato quando un thread esce da un metodo sincronizzato



Metodi synchronized

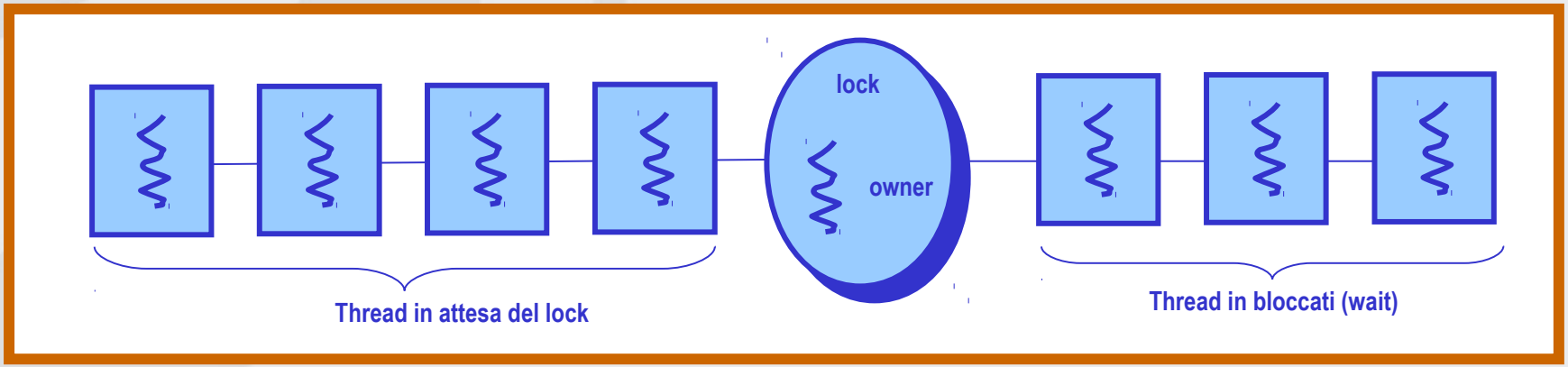
```
    synchronized void incrementaContatore() {
      int tmp = contatore + 1;
      if(tmp>maxContatore) maxContatore = tmp;
      aspetta(milliAspetta);
      contatore = tmp;
    }

    void synchronized decrementaContatore() {
      int tmp = contatore;
      aspetta(milliAspetta);
      contatore = tmp - 1;
    }
```

Sincronizzazione

- **Un Thread può decidere di non proseguire l'esecuzione**
  - **Può allora volontariamente chiamare il metodo `wait` all'interno di un metodo sincronizzato**
    - chiamare `wait` in un contesto diverso genera un errore

Il metodo wait()

- Quando un thread esegue **wait()**, si verificano i seguenti eventi:
  1. il thread rilascia il lock
  2. il thread viene bloccato
  3. il thread è posto in una coda di attesa
  4. gli altri thread possono ora competere per il lock



Il metodo notify()

- Nel momento in cui una condizione che ha provocato una attesa si è modificata si può riattivare un singolo thread in attesa tramite `notify`
- Quando un thread richiama notify():
  1) viene selezionato un thread arbitrario T fra i thread bloccati
  2) T torna fra i thread in attesa del lock
  3) T diventa Eseguibile

insert() con wait/notify

```
public synchronized void insert(Object item) {
   if(count==BUFFER_SIZE) {
      try {
        wait();
      } catch(InterruptedException e) {}
   }
   ++count;
   buffer[in] = item;
   in = (in + 1) % BUFFER_SIZE;
   if(count==1) notify();
}
```

Sincronizzazioni multiple

- notify() seleziona un thread arbitrario.
  Non sempre questo è il comportamento desiderato.
- Java non permette di specificare il thread che si vuole selezionare
- notifyAll() “risveglia” tutti i thread in attesa. In questo modo si permette ai thread stessi di decidere chi deve proseguire.
- notifyAll() è una strategia conservativa utile quando più thread sono in attesa

insert() con wait/notifyAll

```
public synchronized void insert(Object item {
  while(count==BUFFER_SIZE) {
    try {
      wait();
    } catch(InterruptedException e) {}
  }
  ++count;
  buffer[in] = item;
  in = (in + 1) % BUFFER_SIZE;
  if(count==1) notifyAll();
}

private int count=0;
```

Confronto Monitor / Java

- Monitor
- Un solo processo può entrare nel monitor ed eseguire una sua procedura
- Variabili condizione
- wait(variabile_condizione)
- signal(var. condizione)

- (Qualunque) oggetto
- Un solo thread alla volta può eseguire uno dei metodi synchronized
- Una sola implicita (this)
- this.wait()
- this.notify() / notifyAll()

insert() con wait/notify

```
public synchronized void insert(Object item) {
   while(count==BUFFER_SIZE) {
      try {
         wait();
      } catch(InterruptedException e) {}
   }
   ++count;
   buffer[in] = item;
   in = (in + 1) % BUFFER_SIZE;
   if(count==1) notify();
}
```

remove() con wait/notify

```
public synchronized Object remove() {
  while(count==0) {
    try {
      wait();
    } catch(InterruptedException e) {}
  }
  --count;
  Object item = buffer[out];
  out = (out + 1) % BUFFER_SIZE;
  if(count==BUFFER_SIZE-1) notify();
  return item;
}
```

Metodo sleep

- `static void sleep( millisecondi )`
  - Il Thread si sospende (non richiede l'uso del processore) per un certo numero di millisecondi
  - Nel frattempo possono andare in esecuzione thread a bassa priorità
- NB
  - sleep non restituisce il possesso del lock
  - Sleep è un metodo statico della classe Thread
  - Wait è un metodo della classe Object

Sincronizzazione di blocco

- Si definisce **scope** di un lock è il tempo fra la sua acquisizione ed il suo rilascio
- Tramite la parola chiave `synchronized` si possono indicare blocchi di codice piuttosto che un metodo intero
- Ne risulta uno scope generalmente più piccolo rispetto alla sincronizzazione dell'intero metodo

Sincronizzazione di blocco

```
Object mutexLock = new Object();

public void someMethod() {
  nonCriticalSection_1();
  synchronized(mutexLock) {
    criticalSection();
  }
  nonCriticalSection_2();
}
```

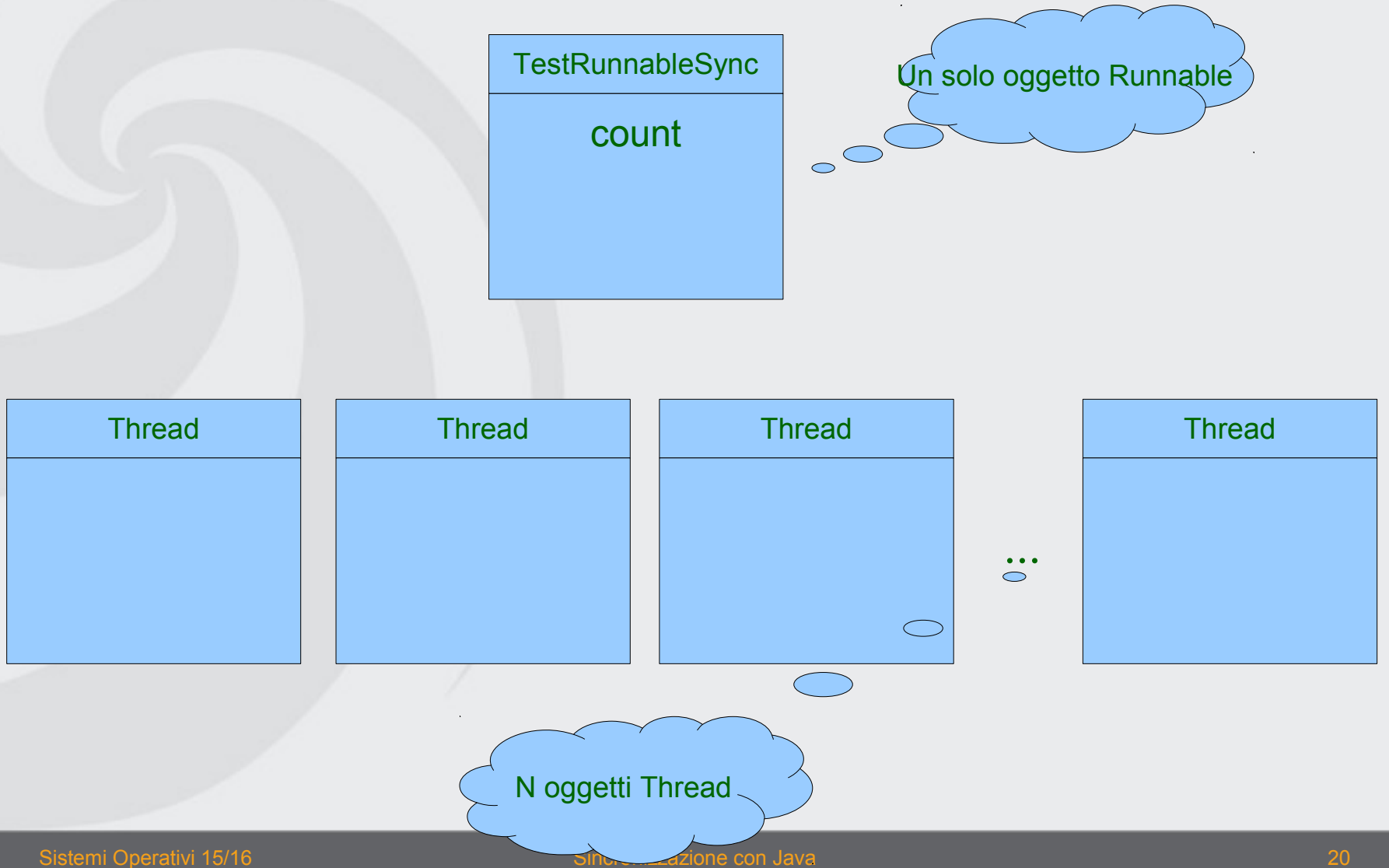
Sincronizzazione di blocco

```
public class TestRunnableSync implements Runnable {
   int count;
   public void run() {
      int tmp;
      synchronized (this) { tmp = count++; }
      String name = Thread.currentThread().getName();
      System.out.println(name + " " + tmp);
   }

   public static void main(String[] args) {
      int n=args.length>0?Integer.parseInt(args[0]):10;
      Runnable runnable = new TestRunnableSync();
      for(int i=0; i<n; i++) {
         Thread th = new Thread(runnable);
         th.start();
      }
   }
}
```

Versione non sincronizzata

```
public class TestRunnableSync implements Runnable {
   int count;
   public void run() {
      int tmp;
      tmp = count++;
      String name = Thread.currentThread().getName();
      System.out.println(name + " " + tmp);
   }

   public static void main(String[] args) {
      int n=args.length>0?Integer.parseInt(args[0]):10;
      Runnable runnable = new TestRunnableSync();
      for(int i=0; i<n; i++) {
         Thread th = new Thread(runnable);
         th.start();
      }
   }
}
```

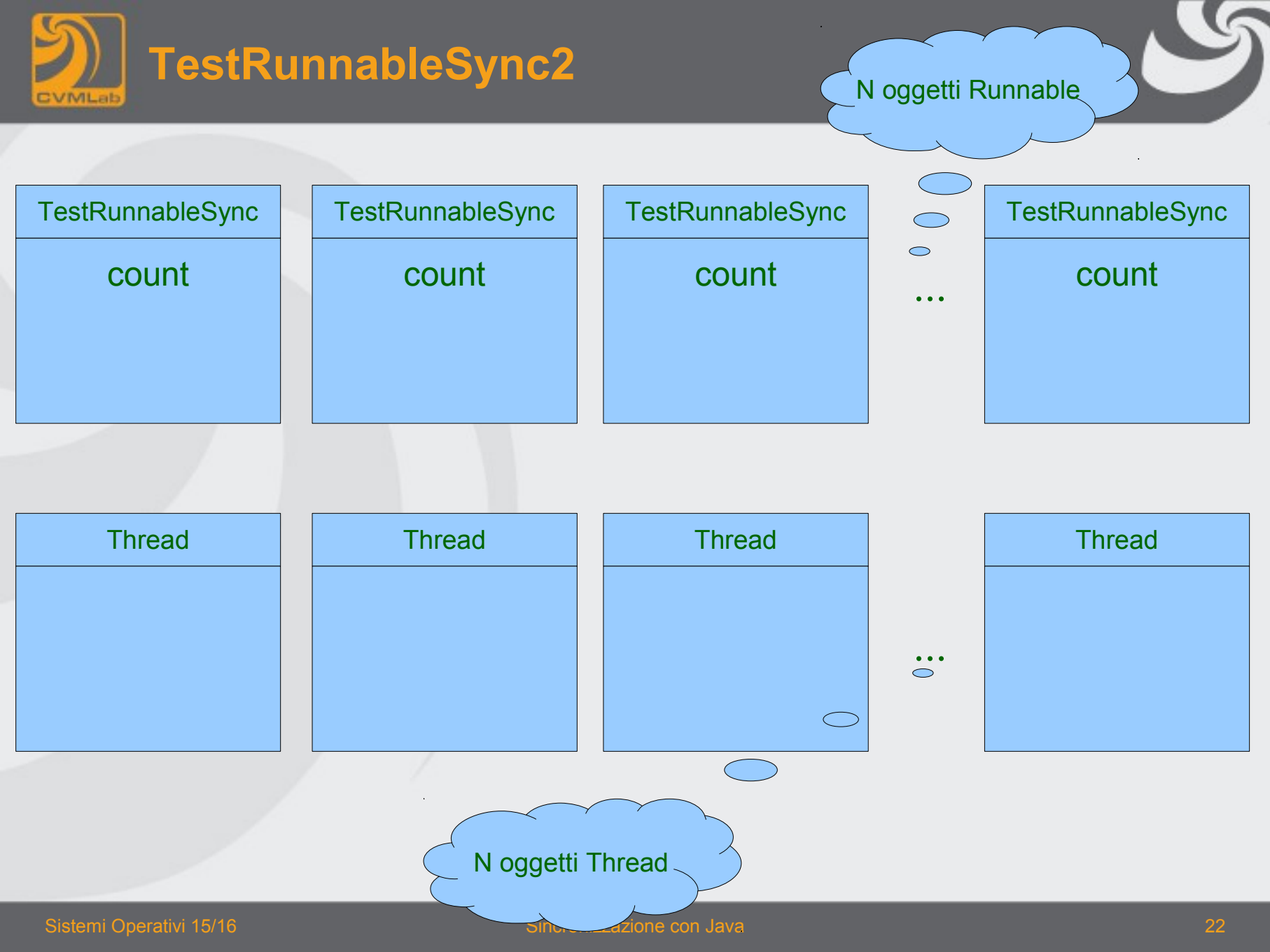
TestRunnableSync



Sincronizzazione di blocco

```
public class TestRunnableSync2 implements Runnable {
   int count;
   public void run() {
      int tmp;
      synchronized (this) { tmp = count++; }
      String name = Thread.currentThread().getName();
      System.out.println(name + " " + tmp);
   }

   public static void main(String[] args) {
      int n=args.length>0?Integer.parseInt(args[0]):10;
      for(int i=0; i<n; i++) {
         Runnable runnable = new TestRunnableSync();
         Thread th = new Thread(runnable);
         th.start();
      }
   }
}
```

TestRunnableSync2



Daemon Threads

- **Sono thread**
  - **Che effettuano operazioni ad uso di altri thread**
    - Per esempio: Garbage collector
  - **Sono eseguiti in background**
    - Usano cioè il processore solo quando altrimenti il tempo macchina andrebbe perso
  - **A differenza degli altri thread non impediscono la terminazione di una applicazione**
    - Quando sono attivi solo thread daemon il programma termina
  - **Occorre specificare che un thread è un daemon prima dell'invocazione di start**
    - `setDaemon( true );`
  - **Il metodo `isDaemon`**
    - Restituisce true se il thread è un daemon thread

La classe Timer

- La classe java.util.Timer permette la schedulazione di processi nel futuro (attenzione esiste anche la classe javax.swing.Timer dal comportamento parzialmente diverso)

  Costruttore: `Timer()`

  `void schedule(TimerTask task, long delay, long period)`

  - Permette di eseguire un task periodico dopo un ritardo specificato
  - Esistono altri costruttori e altri metodi (si veda `scheduleAtFixedRate`)

La classe TimerTask

- TimerTask è una classe astratta occorre estenderla e definire il metodo run (ciò che il task deve fare)

```
long periodo = 1000; // per esempio un secondo
java.util.Timer timer = new java.util.Timer();
timer.schedule(new MyTimerTask(), 0, periodo);

public class MyTimerClass extends
    java.util.TimerTask {

    public void run() {
        // ciò che devo fare
    }

}
```

dovrebbe richiedere un tempo inferiore a periodo

Semafori

- Java, fino alla versione 1.4, non mette a disposizione una implementazione di un semaforo, ma un semplice semaforo può essere costruito tramite il meccanismo della sincronizzazione

```
public interface Semaforo {

    public void up();
    public void down();

}
```

Un semplice semaforo

```
public class SempliceSemaforo {

  private int value;

  public SempliceSemaforo() {
    value = 0;
  }

  public SempliceSemaforo(int v) {
    value = v;
  }
```

Un semplice semaforo

```
  public synchronized void up() {
    value++;
    notify();
  }

  public synchronized void down() {
    while(value==0) {
      try {
        wait();
      } catch (InterruptedException ie) { }
    }
    value--;
  }
}
```

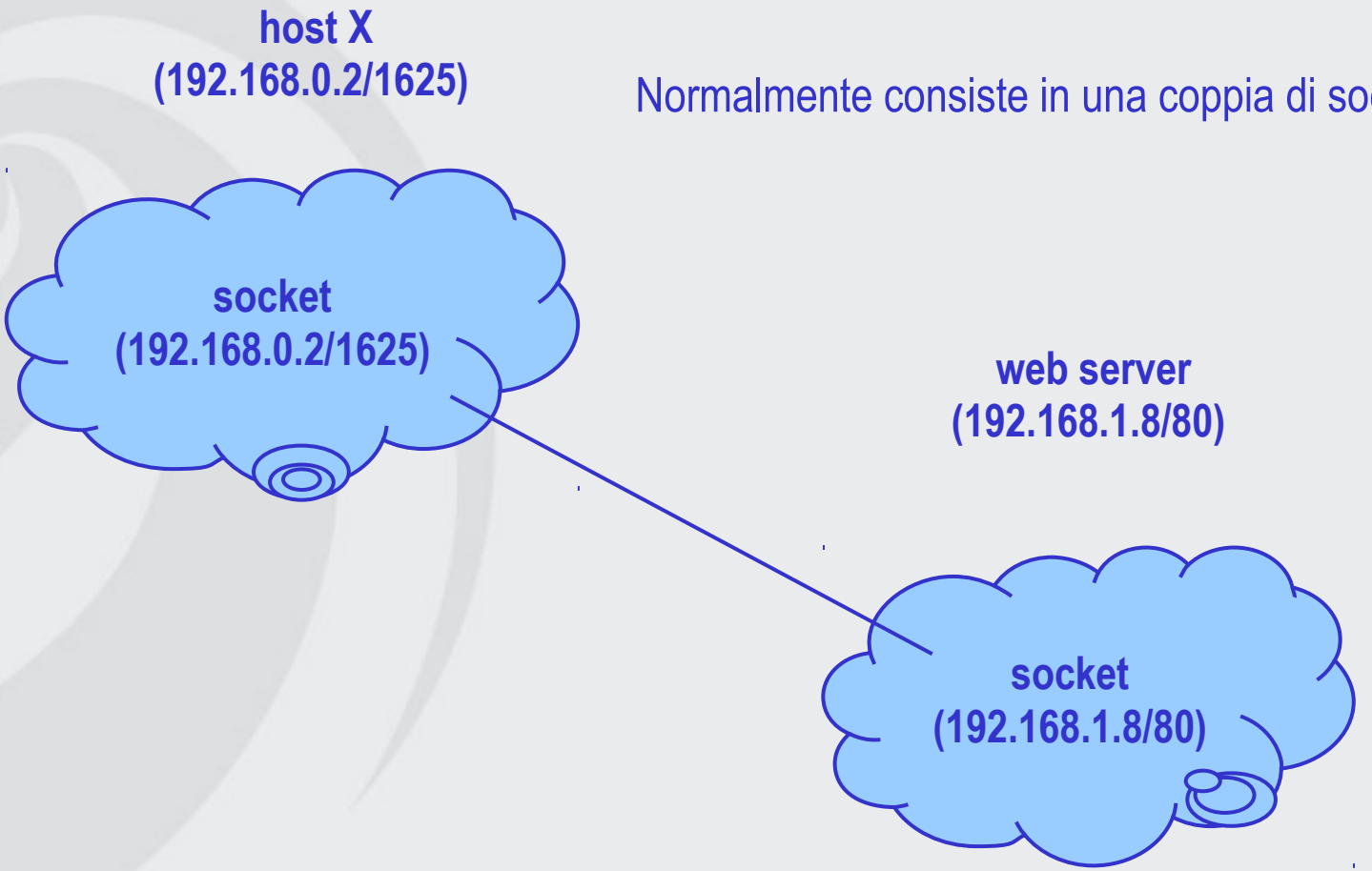
```
public synchronized void down()
        throws InterruptedException {
   while(value==0) wait();
   value--;
}
```

Socket

- Un socket è definito come un estremo di un canale di comunicazione
- È la concatenazione di un indirizzo IP e di una porta
  - Il socket 192.168.0.2:1625 si riferisce alla porta 1625 sull'host 192.168.0.2

Comunicazione fra socket

Normalmente consiste in una coppia di socket



Socket

```
import java.net.*;
import java.io.*;

public class DateServer {
   public static void main(String[] args)
          throws IOException {
      ServerSocket sock = new ServerSocket(6013);
      System.out.println("server = " + sock);
      while (true) {
         Socket client = sock.accept();
         System.out.println("client = " + client);
         PrintWriter pout = new PrintWriter(
            client.getOutputStream(), true);
         pout.println(new java.util.Date().toString());
         pout.close();
         client.close();
      }
      //sock.close();
   }
}
```

Socket

```
import java.net.*;
import java.io.*;

public class DateClient {
   public static void main(String[] args)
          throws IOException {
      InputStream in = null;
      BufferedReader bin = null;
      Socket sock = new Socket("127.0.0.1",6013);
      in = sock.getInputStream();
      bin = new BufferedReader(new InputStreamReader(in));
      String line;
      while( (line = bin.readLine()) != null)
         System.out.println(line);
      sock.close();
   }
}
```

Un server HTTP

```
import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;

public class HTTPD implements Runnable {
   static public void main(String[] args) throws IOException {
      int port=args.length>0?Integer.parseInt(args[0]): 80;
      boolean multiThread = args.length>1 && "-s".equals(args[1]);
      ServerSocket server = new ServerSocket(port);
      System.out.println("Serving in port " + port);
      while (true) {
         HTTPD client = new HTTPD(server.accept());
         if(multiThread) {
            new Thread(client).start();
         } else {
            client.run();
         }
      }
   } ... vedi slide seguenti
}
```

Un server HTTP: il costruttore

```
private Socket client;
private OutputStream output;
private BufferedReader input;

public HTTPD(Socket client) throws IOException {
   this.client = client;
   System.out.println("New client: " + client);
   InputStream inputStream = client.getInputStream();
   output = client.getOutputStream();
   input = new BufferedReader(
     newInputStreamReader(inputStream));
}
```

Un server HTTP: il metodo run

```
public void run() {
   try {
     while(true) {
         String line = input.readLine();
         if (line == null) {
           System.out.println("Connection closed.");
           output.close(); client.close();
           return;
         }
         if (line.startsWith("GET /")) {
          output.write(getAnswer(line).getBytes());
          output.flush();
         }
         System.out.println(line);
      }
   }
   catch (IOException ex) { … }
}
```

Un server HTTP: il metodo getAnswer

```
/**
 * Return the new page.<br>
 * Subclasses of HTTPD should override this method.
 *
 * @param line The request from the client.
 * @return the new page.
 */
protected String getAnswer(String line) {
  String s = new java.util.Date().toString() + '\n';
  String answer =
    "HTTP/1.1 200 OK\n" +
    "Content-Length: " + s.length() + "\n" +
    "Content-Type: text/html\n\n" + s;
  return answer;
}
```

Messaggio di richiesta (Firefox)

```
GET / HTTP/1.1
Host: 127.0.0.1:8080
User-Agent: Mozilla/5.0 (Windows NT 5.1; rv:6.0.2)
                      Gecko/20100101 Firefox/6.0.2
Accept: text/html,application/xhtml+xml,application/xml;
                      q=0.9,*/*;q=0.8
Accept-Language: it-it,it;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Connection: keep-alive
Cache-Control: max-age=0
```

Un server web

- Cosa manca?
  - La gestione delle informazioni che arrivano dal client
  - Il trattamento effettivo delle richieste
  - Uno spegnimento *"gentile"* del server
  - Il trattamento delle eccezioni
  - Riutilizzo dei thread
  - Una applicazione HTTP client