Sistemi Operativi

(modulo di Informatica II)

Laboratorio

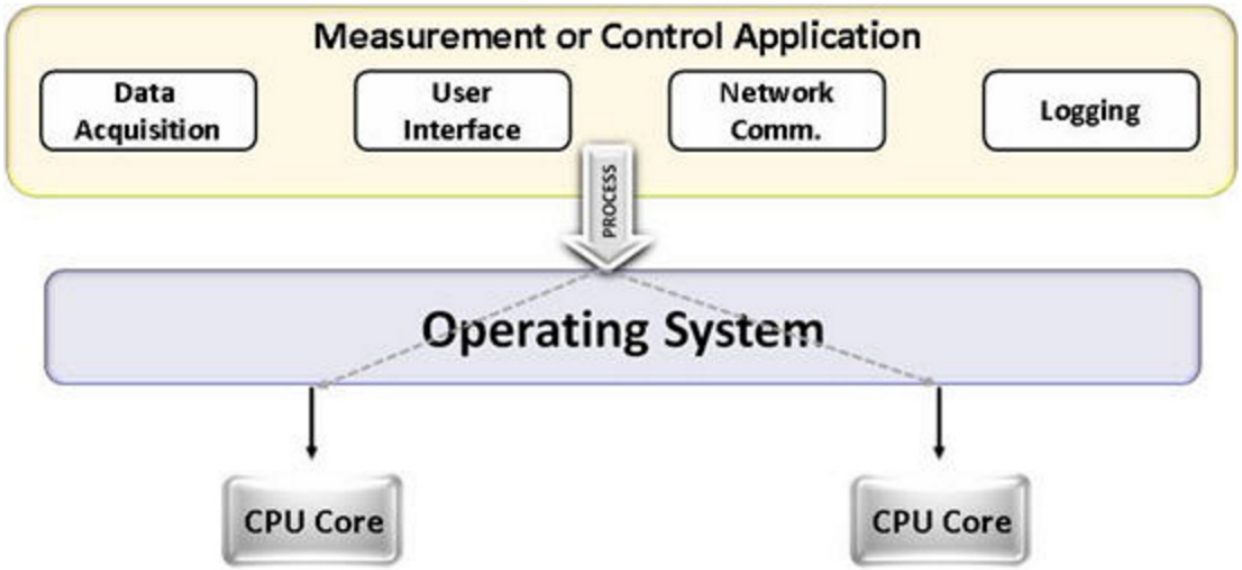
Multithreading in Java

Patrizia Scandurra

Università degli Studi di Bergamo

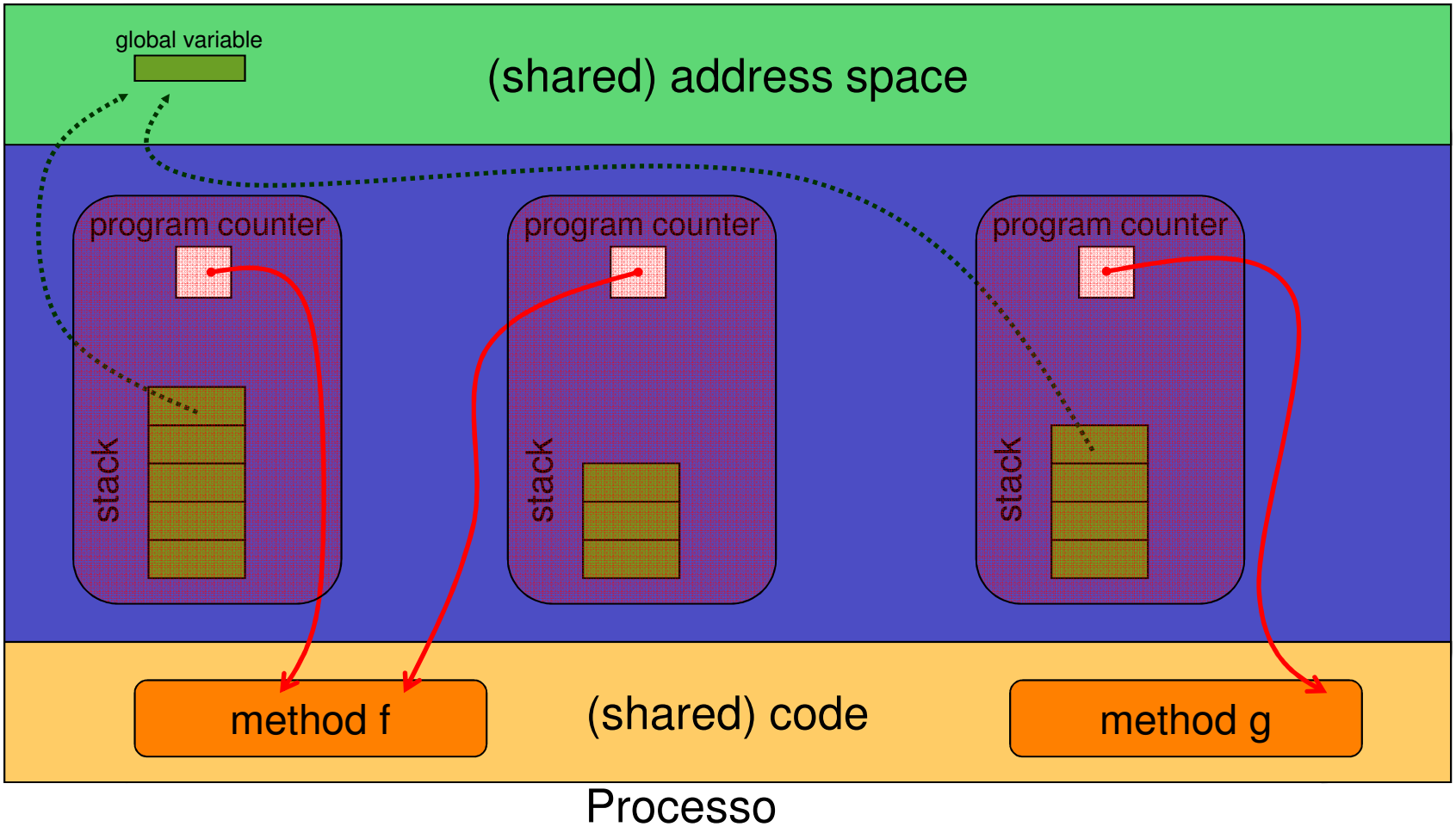
Multithreading

- Multithreading: estende l'idea del **multitasking a livello di processo**, in modo da poter suddividere specifiche operazioni di una singola applicazione in thread separati
- Ogni thread può essere eseguito in parallelo (concorrenza apparente o vera)
- Il SO divide il tempo di processamento non solo tra diversi processi, ma anche tra diversi thread di ogni processo



Concetto di Thread

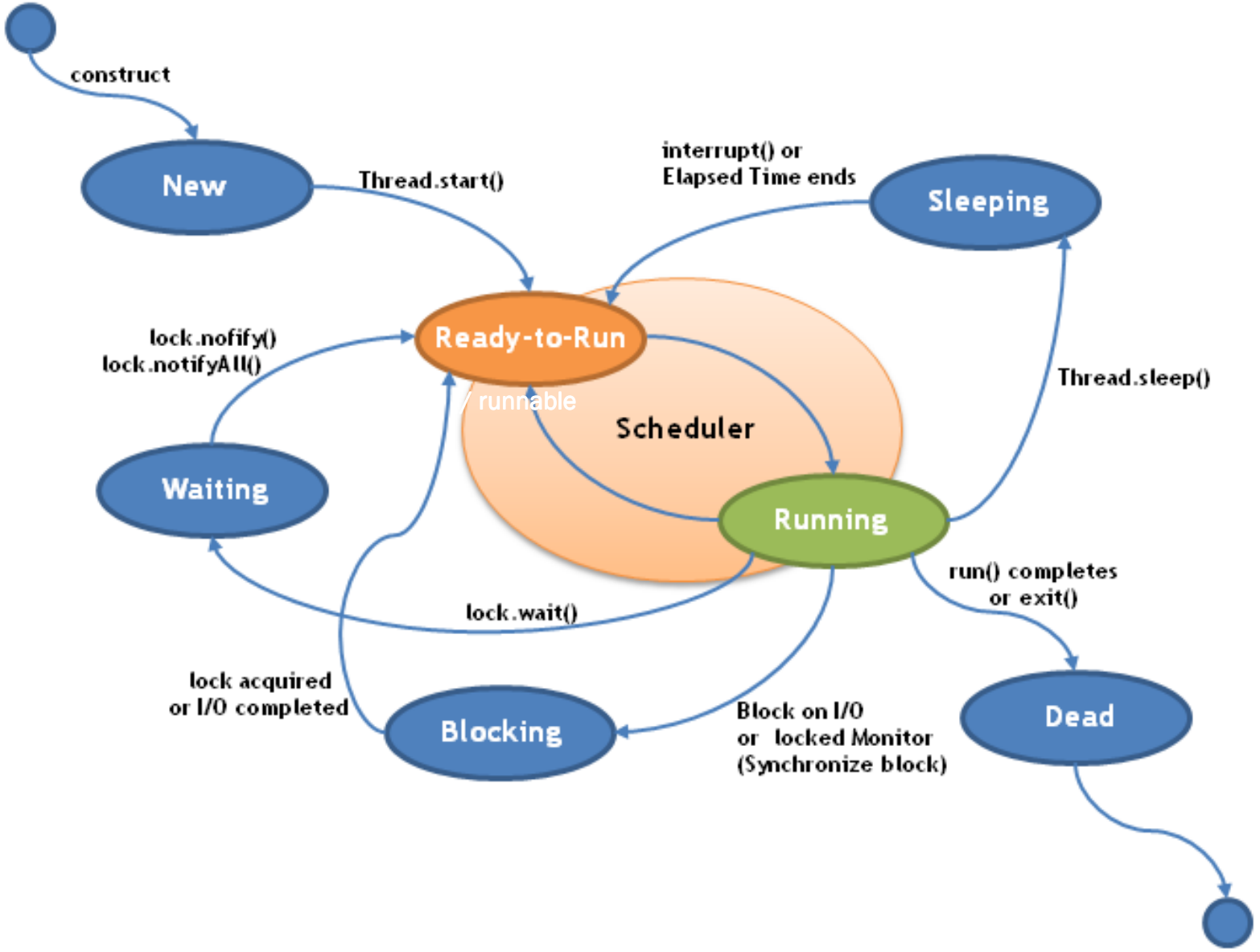
- **Flusso di esecuzione indipendente interno ad un processo**
  - condivide lo spazio di indirizzamento con gli altri thread del processo



La gestione dei thread in Java

- Creazione e avvio
- Ricongiunzione (join)
- Condivisione di oggetti (dati) tra thread
- Terminazione e cancellazione dei thread

Java Thread lifecycle



Creazione di thread

- Ogni programma Java ha almeno un thread corrispondente a **main()** – ***main thread***

- Altri thread possono essere creati mediante oggetti della classe `java.lang.Thread`
  - Attenzione a non confondere il concetto di thread con gli oggetti di tale classe!

- Gli oggetti di tale classe svolgono la funzione di **interfaccia verso la JVM**
  - tali oggetti sono solo lo strumento con il quale è possibile *comunicare* alla JVM
    - di creare nuovi thread
    - di interrompere dei thread esistenti
    - di attendere la fine di un thread (join)
    - …

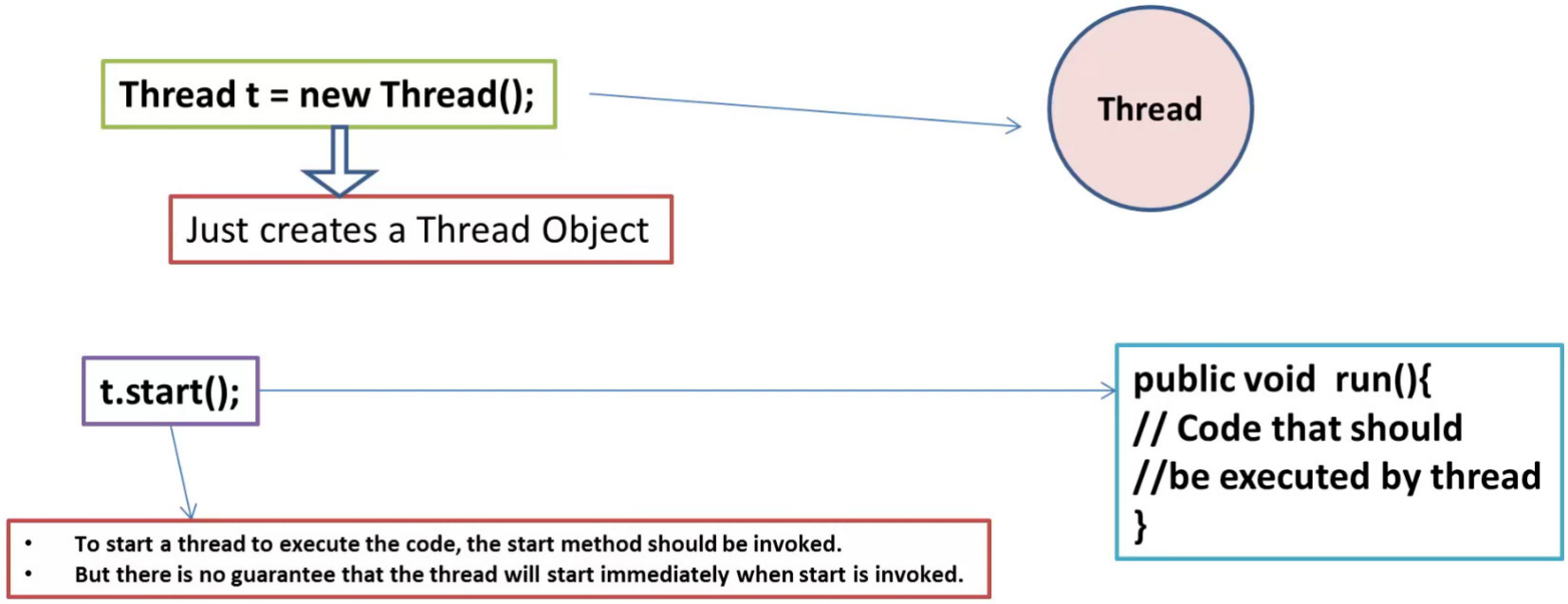
java.lang.Thread (metodi per creare e avviare un thread)

Constructor Summary
`Thread``()` Allocates a new `Thread` object.
`Thread``(Runnable target)` Allocates a new `Thread` object.

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

java.lang.Thread

- E' possibile "creare" dei thread in Java creando oggetti della classe java.lang.Thread
- I thread vengono effettivamente creati dalla JVM quando viene invocato il metodo start() sui corrispondenti oggetti Thread



java.lang.Thread (metodi costruttori)

- Metodi costruttori per creare oggetti Thread
- Parametri durante la creazione di un Thread:
  - Runnable target -> Oggetto che implementa il metodo run()
  - String threadName -> Nome che diamo al thread
  - ThreadGroup group -> Gruppo di cui il thread creato fa parte

Constructors

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 target)` Allocates a new Thread object.
`Thread(ThreadGroup group, Runnable target, String name)` Allocates a new Thread object so that it has `target` as its run object, has the specified `name` as its name, and belongs to the thread group referred to by `group`.
`Thread(ThreadGroup group, Runnable target, String name, long stackSize)` Allocates a new Thread object so that it has `target` as its run object, has the specified `name` as its name, and belongs to the thread group referred to by `group`, and has the specified *stack size*.
`Thread(ThreadGroup group, String name)` Allocates a new Thread object.

Metodi per creare thread

- I thread di Java possono essere creati in due modi:
  1. tramite *estensione* della **classe Thread**
  2. tramite *implementazione* dell' **interfaccia Runnable**

METODO 1

```
class Downloader extends Thread{
                private String url;
                public Downloader(String url){
                this.url = url;
                }
                public void run(){
                FileDownloader fd = new FileDownloader();
                fd.download(this.url);
                }
}
```

- Downloader d = new Downloader(url);
- d.start();

METODO 2

```
import java.lang.Runnable;
class Downloader implements Runnable{
                private String url;
                public Downloader(String url){
                this.url = url;
                }
                public void run(){
                FileDownloader fd = new FileDownloader();
                fd.download(this.url);
                }
}
```

- Thread d = new Thread(new Downloader(url));
- d.start();

Attenzione! Mai invocare direttamente **run()!**
E' il metodo start() ad invocare run()

Creazione di thread: primo metodo

Esempio 1:

```
class Worker1 extends Thread
{
  public void run() {
    System.out.println("Thread ausiliario: hello
  world!");
  }
}
```

Avvio (spawn) di un thread

Esempio 1:

```
public class First
{
  public static void main(String args[]) {
    Thread runner = new Worker1();
    runner.start();
    System.out.println("Thread principale: hello
  world!");
  }
}
```

<DEMO First >

Creazione di thread: secondo metodo (2)

- Instanziare **direttamente** Thread passando al suo costruttore un oggetto di una classe che implementa l'interfaccia Runnable

```
Thread t = new Thread(new MyThread());
 t.start();
```

<DEMO Second>

Creazione e avvio di più thread

Esempio:

```
Thread[] threads = new MyThread[10];
for (int i = 0; i < threads.length; i++) {
        threads[i] = new MyThread();
        threads[i].start();
 }
```

Riassumendo: due modi per creare thread in Java

- *Metodo 1*: Instanziare classi che derivano da java.lang.Thread
  - più semplice
  - ma **non è possibile estendere altre classi**

- *Metodo 2*: Instanziare direttamente Thread passando al suo costruttore un oggetto di interfaccia Runnable
  - la classe può **estendere una qualsiasi altra classe**
  - occorre usare il metodo statico **Thread.currentThread()** per ottenere il riferimento all'oggeto thread corrente

Terzo metodo: con classe anonima

- Creazione dinamica di un thread con classe anonima:

```
Thread thread = new Thread(){
    public void run(){
      System.out.println("Thread Running");
    }
  }

  thread.start();
```

<DEMO Third>

Altri metodi della classe Thread

- DOCUMENTAZIONE
  https://docs.oracle.com/javase/8/docs/api/java/lang/Thread.html

Esempi:

- getName(), setName(String s)
  per ottenere e settare, rispettivamente, il nome di un thread
- isAlive()
  per sapere se un thread è ancora in vita (running); possibile utilizzo per il re-start del thread
- Thread.sleep(int ms)
  consente di bloccare il thread corrente per il numero specificato di millisecondi
- Thread.currentThread()
  restituisce il riferimento del thread attualmente in esecuzione

  Gli ultimi due sono statici

Il metodo isAlive() della classe Thread

- E' possibile sapere se un thread è ancora in vita con il metodo isAlive() : restituisce true se il thread è running, false altrimenti

- Possibile utilizzo: per il restart di un thread

```
if (!t.isAlive()) {
  t.start();
}
```

La gestione dei thread in Java

- Creazione e avvio
- Ricongiunzione (join)
- Condivisione di oggetti (dati) tra thread
- Terminazione e cancellazione dei thread

Ricongiunzione (join)

- Invocando **il metodo join()** su un oggetto thread, il thread corrente si blocca fino alla terminazione del thread associato a tale oggetto

```
Thread t = new MyThread();
 t.start();
 try{
    t.join(); //il thread corrente aspetta che il thread associato a t termini
  } catch (InterruptedExecution e) {}
```

- Utile in situazioni in cui un thread padre vuole attendere la terminazione del thread figlio perché si aspetta dei risultati da esso

<DEMO JoinExample.java>

Join multipli

- Basta ricongiungersi a ciascun thread

Esempio:

```
Thread[] threads = new MyThread[10];
for (int i = 0; i < threads.length; i++) {
        threads[i] = new MyThread();
        threads[i].start();
 }
 try{
          for (Thread thread : threads)
              thread.join();
  } catch (InterruptedExecution e) {}
```

La gestione dei thread in Java

- Creazione e avvio
- Ricongiunzione (join)
- Condivisione di oggetti (dati) tra thread
- Terminazione e cancellazione dei thread

Condivisione di oggetti (senza sincronizzazione)

```
public class ProvaThread {

    public static void main(String[] args) {

        OggettoCondiviso  o = new OggettoCondiviso(33);

        Runnable r1 = new MyRunnable(o);
        Runnable r2 = new MyRunnable(o);

        Thread t1 = new Thread(r1);
        Thread t2 = new Thread(r2);

    }
}

class MyRunnable implements Runnable {
 OggettoCondiviso so;

MyRunnable(OggettoCondiviso so) {
    this.so = so;
}

public void run() {
    // codice del thread: utilizza so
}
```

i parametri vengono passati al costruttore di MyRunnable

lo stesso oggetto viene passato a due thread

Condivisione di oggetti (senza sincronizzazione)

<DEMO Somma.java>

La gestione dei thread in Java

- Creazione e avvio, join
- Condivisione di oggetti (dati) tra thread
- Terminazione e cancellazione dei thread

Terminazione e cancellazione dei thread

- Terminare un thread prima che abbia ancora finito di eseguire il metodo **run()**
- Due approcci:
  1. **Cancellazione asincrona**: terminazione immediata
  2. **Cancellazione differita**: il thread oggetto della cancellazione valuta periodicamente se deve terminare o meno a seguito della ricezione di un segnale di terminazione

Cancellazione asincrona

- Invocando sul thread il metodo **stop()** della classe Thread

- E' però stato *deprecato* (*deprecated*)
  - ancora implementato nelle API Java correnti
  - ma il loro uso è scoraggiato

- Se un thread che sta aggiornando dati condivisi viene "stoppato", libererà l'eventuale lock acquisito per l'accesso esclusivo ai dati ma può lasciare questi ultimi in uno stato inconsistente!

Cancellazione differita (1)

Tramite interruzione con il metodo **interrupt()** della classe Thread

```
Thread t = new InterruptibleThread();
 t.start();
 ...
 t.interrupt(); //setta il flag interruption status
 ...
```

Cancellazione differita (2)

- Il thread target può controllare il suo stato di interruzione con i metodi **interrupted()** o **isInterrupted()** della classe Thread

- E fare pulizia prima di terminare

<DEMO Interrupter.java>

Esercizi

- **Test concorrenza della piattaforma ospitante**: Comprendere ed eseguire il programma Java ***simboli.java*** nella cartella ***Esempi***. E' un'applicazione dove il main thread crea due thread figli. Ciascuno dei due thread figli ha associato un simbolo (ad es. * o #). Nel metodo run(), ciascun thread stampa continuamente su stdout il proprio simbolo, andando a capo ogni 50 simboli stampati.
- **Non determinismo**: creare un programma Java contenente 3 thread come segue. Un thread principale corrispondente al metodo main() che stampa il suo nome, e poi crea ed avvia due thread dello stesso tipo: un thread di nome "A" e un thread di nome "B". Il comportamento dei thread consiste nell'attendere 1 secondo e stampare su stdout il proprio nome. Provare ad eseguire l'applicazione più volte, e a notare la *sequenza di interleaving* dalle stampe dei thread.
- **Somma parallela**: Scrivere un programma Java che dato un array di 50 valori numerici ne esegue la somma in modo parallelo. A tale scopo, il thread main crea 5 thread e assegna ad ogni thread una porzione di 10 valori dell'array. Ogni thread esegue la somma dei propri valori e mette il risultato in una struttura dati del thread main contenente i risultati parziali. Il thread main deve sincronizzarsi sulla terminazione (join) dei thread figli; dopodiché somma i risultati parziali e li stampa su stdout.