SISTEMI OPERATIVI

(MODULO DI INFORMATICA II)

Java multithreading

Prof. Luca Gherardi

Prof.ssa Patrizia Scandurra (anni precedenti)

Università degli Studi di Bergamo

a.a. 2012-13

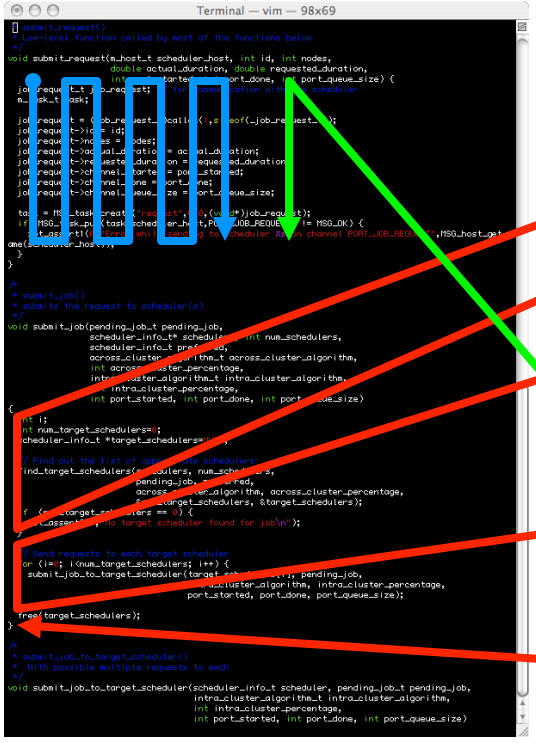
Sommario

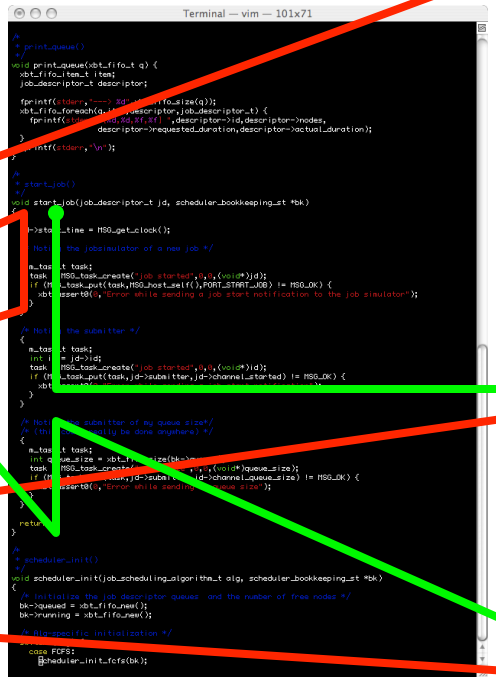
- Programmazione concorrente
- Concetto di thread
- I thread di Java
- Creare e avviare thread in Java
- Terminazione e cancellazione di thread in Java
- Esercizi

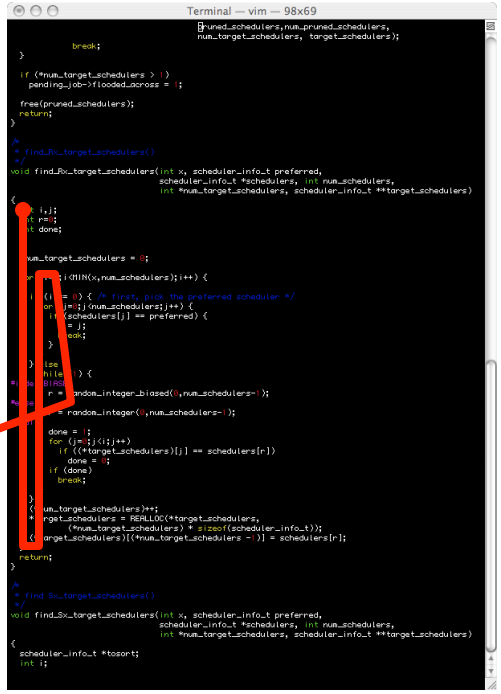
Programmi concorrenti

- Esecuzione simultanea di
  - un task blu
  - un task rosso
  - e un task verde

e così via …







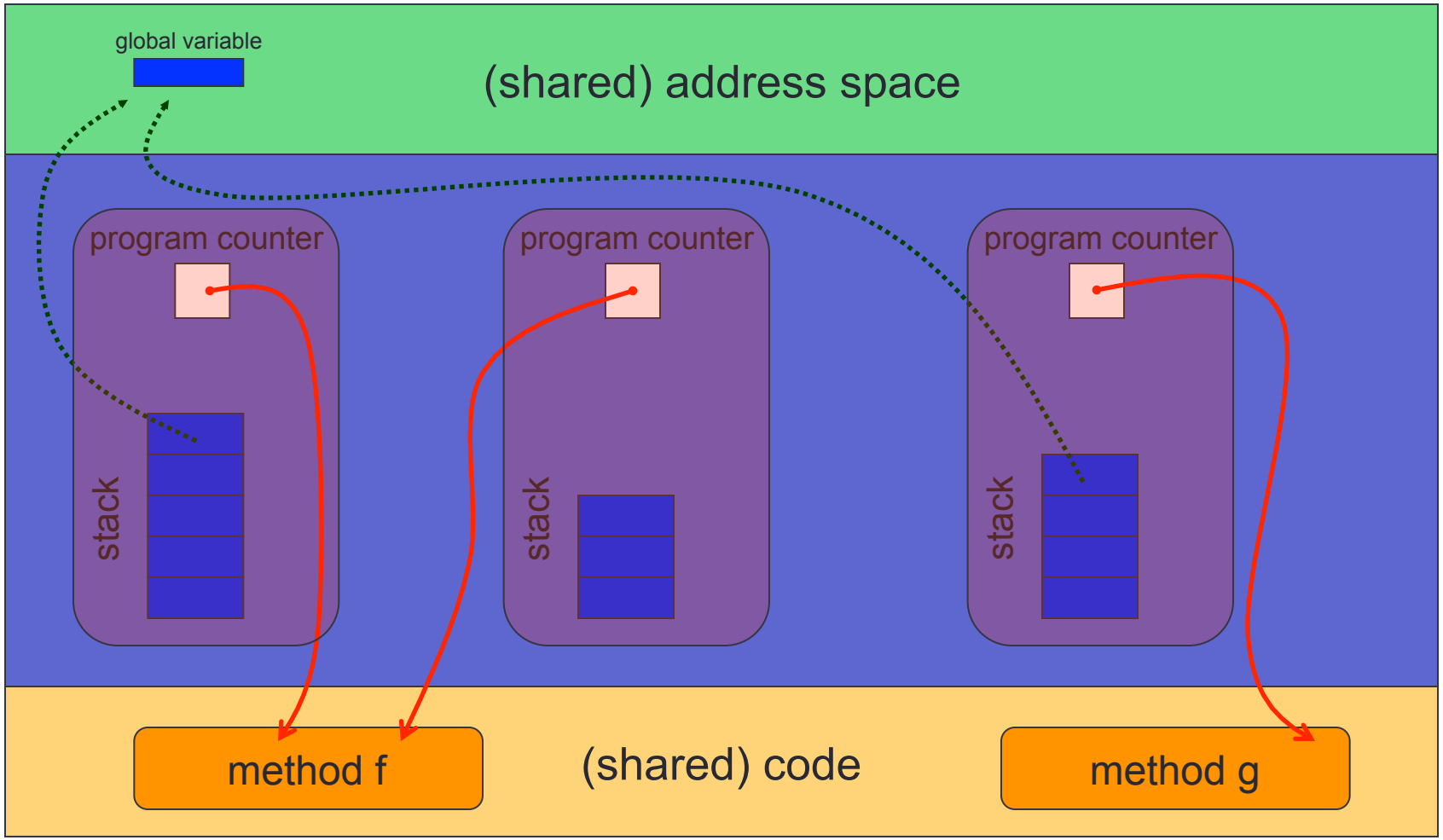
Concorrenza

- **Definizione: esecuzione di "task" multipli nello "stesso" tempo**
- In un programma non-concorrente, o sequenziale
  - In ogni momento è possibile interrompere il programma e dire esattamente quale task si stava eseguendo, quale era la sequenza di chiamate, ecc.
  - Esecuzione **deterministica**
- In un programma concorrente, si individuano un certo numero di task da eseguire come **"flussi indipendenti"**
  - Ogni task ha un compito specifico da portare avanti
  - I task possono comunicare tra loro **"flussi cooperanti"**
  - Regioni differenti del codice eseguite allo stesso tempo
  - Lo stato di un programma ha più di una dimensione
  - Esecuzione **non deterministica**

Concetto di Thread

- Anche chiamati **lightweight process** perché possiedono un contesto più snello rispetto ai processi
- **Flusso di esecuzione indipendente**
  - **interno ad un processo**
  - condivide lo spazio di indirizzamento con gli altri thread del processo
- Per alcuni problemi risultano più semplici ed efficienti rispetto ai processi

Thread in un processo

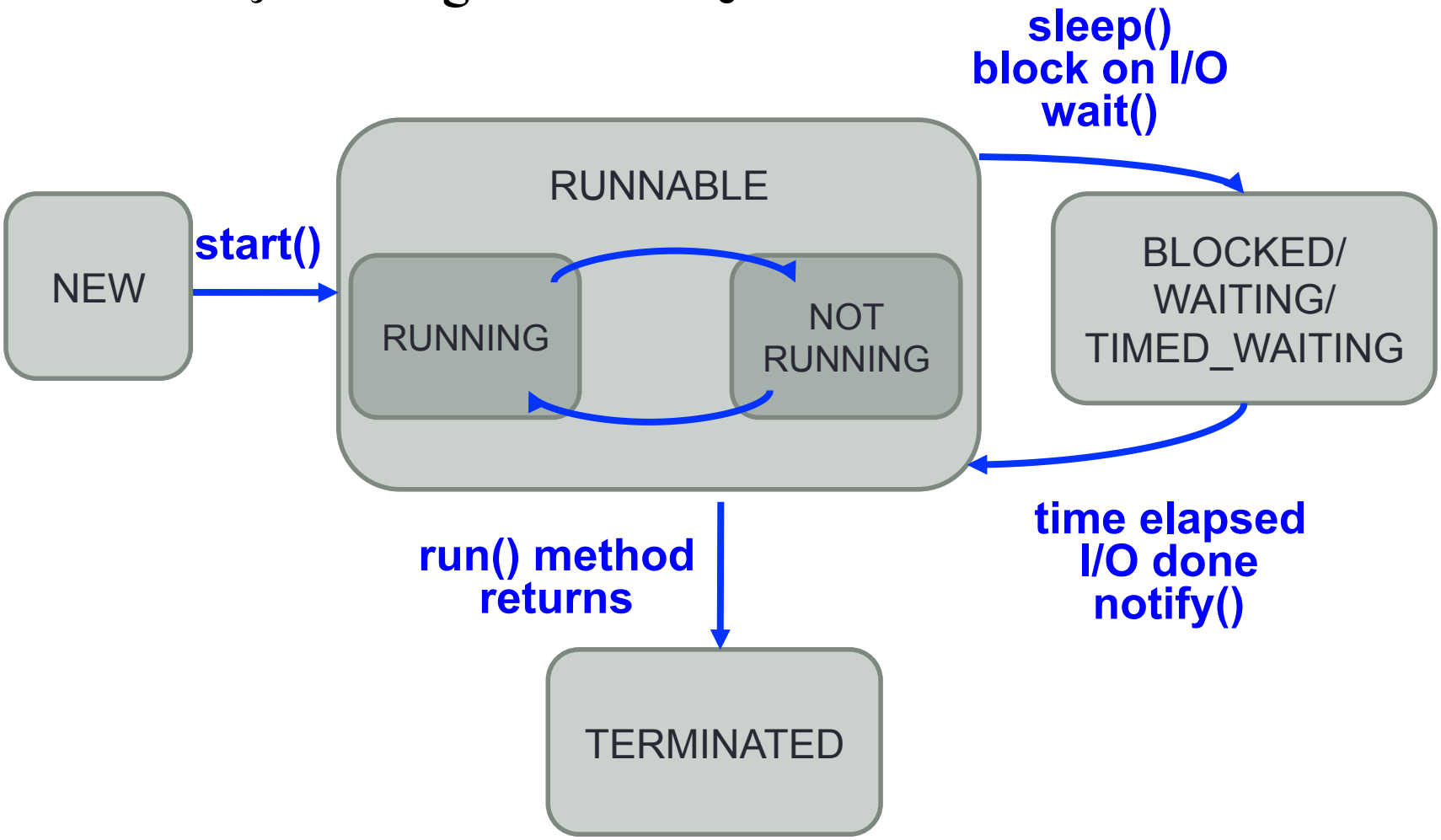


Thread & JVM

- In realtà **qualsiasi programma java è multithread**
  - Il garbage-collector è gestito da un thread separato
- Se si usano GUI, la JVM crea almeno due altri thread
  - Uno per gestire gli eventi della GUI
  - Uno per il rendering grafico

Java Thread Lifecycle: 4 stati

- I thread di Java sono **gestiti dalla JVM**



La gestione dei thread in Java

- Creazione, avvio e join
- Terminazione e cancellazione dei thread
- Dati specifici dei thread
- Schedulazione
- Gruppi di thread
- La gestione dei segnali
- Sincronizzazione

La gestione dei thread in Java

- **Creazione, avvio e join**
- Terminazione e cancellazione dei thread
- Dati specifici dei thread
- Schedulazione
- Gruppi di thread
- La gestione dei segnali
- Sincronizzazione

Creazione di thread

- Ogni programma Java ha almeno un thread corrispondente a **main()** – ***main thread***
- Altri thread possono essere creati dinamicamente
- I thread della JVM sono associati ad istanze della classe `java.lang.Thread`
- I thread di Java possono essere creati in due modi:
  1. L'estensione della classe Thread
  2. L'interfaccia Runnable

java.lang.Thread

- Gli oggetti istanza di tale classe svolgono la funzione di **interfaccia verso la JVM** che è l'unica capace di creare effettivamente nuovi thread
- Attenzione a non confondere il concetto di thread con gli oggetti istanza della classe java.lang.Thread
  - 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

```
package java.lang;

public class Thread implements Runnable {
  public void start();
  public void run();
  public boolean isAlive();
  public static void sleep(long millis);
  public static void sleep(long millis, long
 nanos);
  public Thread.State getState();
  // Ed altri ancora che vedremo via via
 ...
}
```

Creazione di thread: primo metodo

- Il primo metodo per creare un thread consiste nell'estendere la classe Thread e fare l'override del metodo "run()"

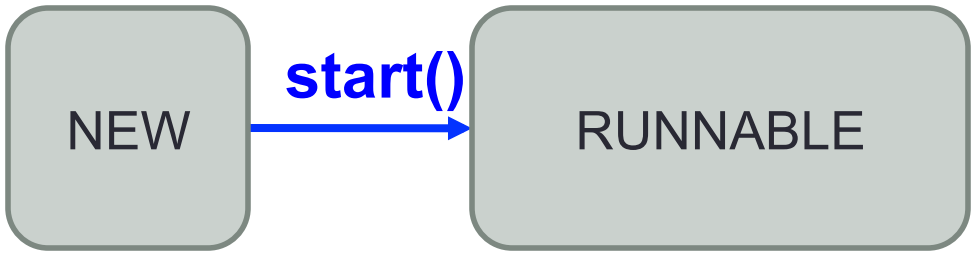
```
class MyThread extends Thread {
    public void run() {
        . . .
    }
    . . .
}

MyThread t = new MyThread();
```

Creazione di thread: primo metodo Esempio

```
class Worker1 extends Thread
{
    public void run() {
        System.out.println("Sono un
            thread ausiliario");
    }
}
```

Avvio (spawn) di un thread

- Una volta creato, basta invocare il metodo **start()**
- **Attenzione**: non invocare direttamente **run()**
  - Avrebbe l'effetto di una normale chiamata di metodo di un oggetto qualunque
- E' il metodo start() (che non va sovrascritto), ad occuparsi delle operazioni necessarie per lo *spawn*
  - E' il metodo start() ad invocare run()

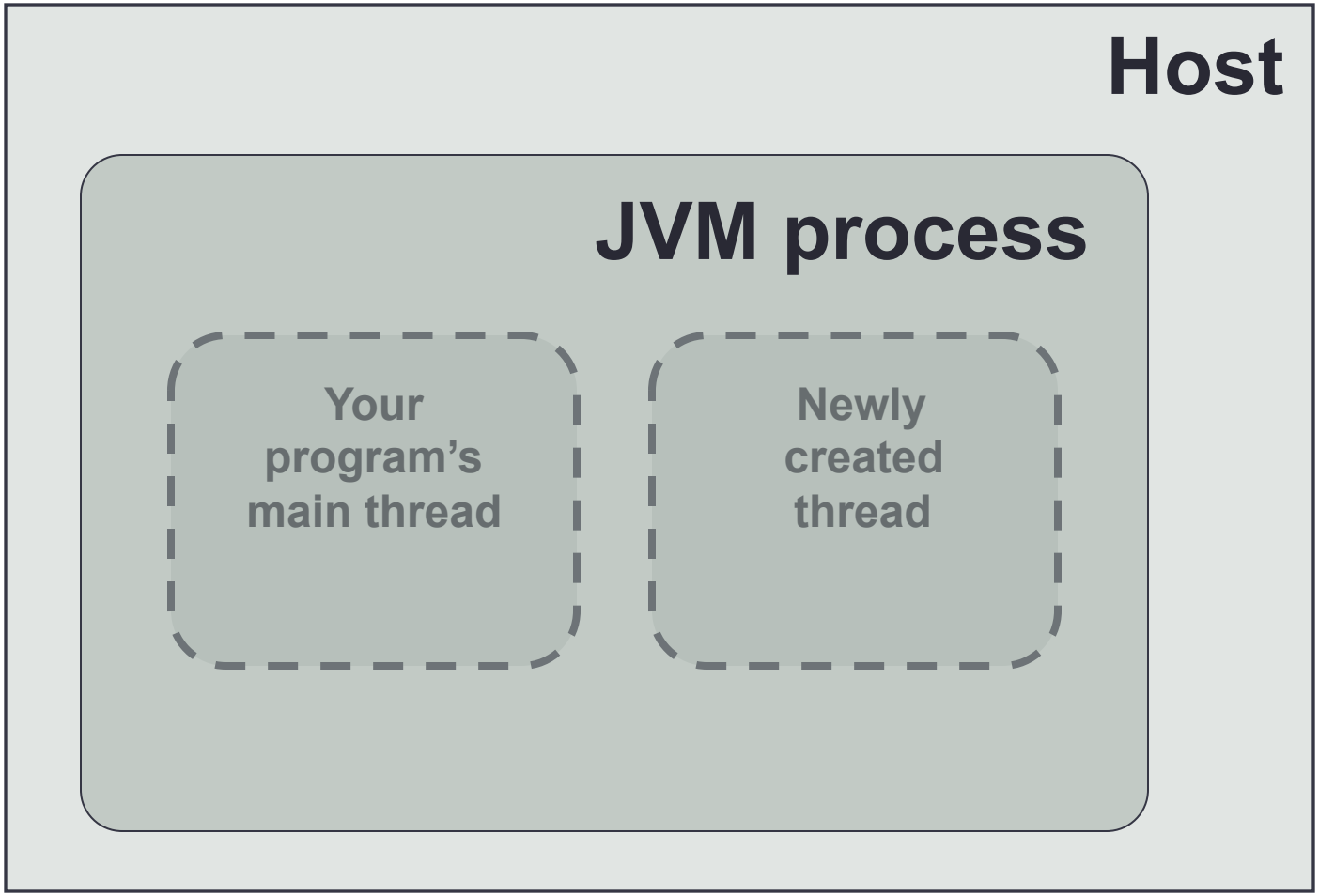


Avvio (spawn) di un thread
Esempio 1

```
public class First
{
    public static void main(String
 args[]) {
        Thread runner = new Worker1();
        runner.start();
    }
}
```

<DEMO First_A>

Cosa succede



Avvio (spawn) di un thread
Esempio 2

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

<DEMO First_B>

Cosa succede

- Entrambi scrivono a video
- **Domanda**: quale sarà l'output?
- **Risposta**: dipende dall'implementazione della JVM sulla particolare macchina su cui l'applicazione esegue
  - Non bisogna fare alcuna assunzione sulle velocità relative dei singoli thread!

Creazione di thread: secondo metodo (1)

- Si crea una classe che implementa l'interfaccia **Runnable**
  - che richiede l' implementazione del solo metodo `run()`
  - l' avvio di un thread si ottiene con la chiamata del metodo `start()`

```
class MyThread implements
Runnable {
        public void run() {
            //do something
            ….
        }
   }
```

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>

java.lang.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.

Riassumendo: due modi per creare thread in Java

- Instanziare classi che derivano da java.lang.Thread
  - più semplice
  - ma **non è possibile derivare da altre classi**

- Instanziare direttamente Thread passando al suo costruttore un oggetto di interfaccia Runnable
  - la classe può **derivare da una qualsiasi altra classe**
  - occorre usare **Thread.currentThread()** per ottenere il riferimento al thread corrente (ed invocare poi su di esso altre azioni)

Il metodo isAlive()

- 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();
}
```

Altre operazioni utili sui thread

- `getName()` e `setName(String s)`: per ottenere e settare, rispettivamente, il nome di un thread
- `Thread.sleep(int ms)`: consente di bloccare il thread corrente per il numero specificato di millisecondi
  - può sollevare una java.lang.InterruptedException
- `Thread.currentThread()`: restituisce il riferimento del thread attualmente in esecuzione

Gli ultimi due sono statici

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

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();
} catch (InterruptedExecution e) {}
```

- Ad es. in situazioni in cui un thread padre vuole attendere la terminazione del thread figlio

<DEMO JoinExample.java>

La gestione dei thread in Java

- Creazione, avvio e join
- **Terminazione e cancellazione dei thread**
- Dati specifici dei thread
- Schedulazione
- Gruppi di thread
- La gestione dei segnali
- Sincronizzazione

Thread in Java: Terminazione

- Un thread termina quando:
  - finisce *spontaneamente* l'esecuzione del metodo `run()`
  - viene *esplicitamente interrotto* ovvero cancellato
  - *eccezioni* ed *errori*

Cancellazione dei thread

- Terminare un thread prima che abbia ancora finito di eseguire il metodo `run()`

- Due approcci:
  - **Cancellazione asincrona**: terminazione immediata
  - **Cancellazione differita**: il thread oggetto della cancellazione valuta lui stesso periodicamente se deve terminare o meno

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 porre le risorse in uno stato consistente prima di terminare

<DEMO Interrupter.java>