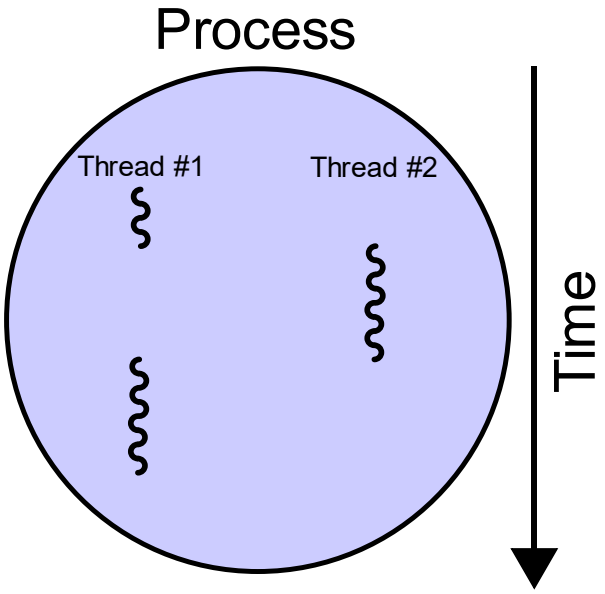
D02
Materiale Integrativo

Luca Grilli

Programmazione di Interfacce Grafiche e Dispositivi Mobili

A.A. 2020-2021



Cenni al *Multithreading* in Java

Concetti di base sulla programmazione multithreading (in Java)

Flusso di controllo (o di *esecuzione*)

- Il ***flusso di esecuzione*** di un programma è l'**ordine** in cui le singole **istruzioni**, o chiamate di funzione, vengono **eseguite** o **valutate**
  - il ***flusso di esecuzione*** è anche detto ***flusso del controllo di esecuzione***, o semplicemente ***flusso di controllo***, o ***flusso del controllo***
  - in inglese ***control flow*** o ***flow of control***
- Un linguaggio di programmazione include (tipicamente) dei **costrutti** per consentire al programmatore di **controllare il flusso di esecuzione**
- Tali costrutti sono pertanto denominati ***strutture di controllo***, e servono a specificare ***se***, ***quando, in quale ordine***, e ***quante volte*** devono essere eseguite determinate istruzioni

Tipiche strutture di controllo 1/2

- Strutture di tipo ***sequenza***
  - stabiliscono l'ordine in cui le istruzioni presenti nel testo del programma devono essere eseguite a tempo di esecuzione
  - tipicamente non hanno un'espressione sintattica esplicita

- Strutture ***condizionali*** (o strutture ***alternative***)
  - per specificare se una data istruzione, o un dato blocco di istruzioni, debba essere eseguito in funzione della validità di una certa condizione booleana (cioè «solo se» è *TRUE*)
  - ***if-then***,
  - ***if-then-else***,
  - ***switch-case***

Tipiche strutture di controllo 2/2

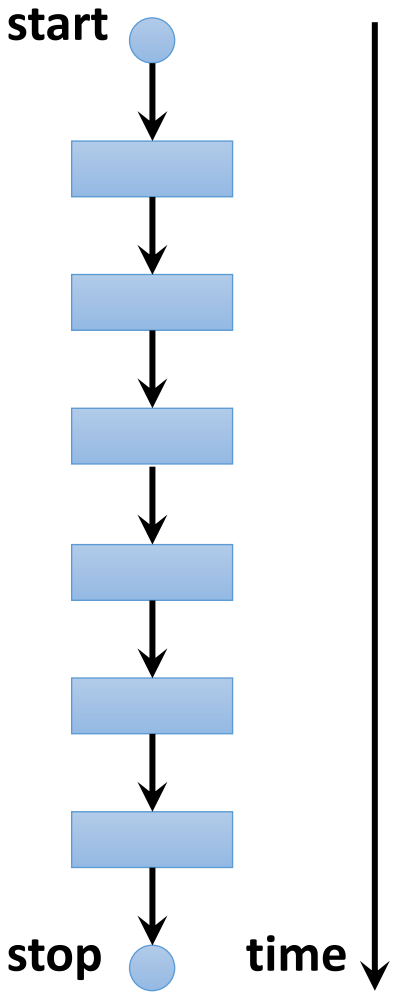
- Strutture ***iterative*** (o ***cicli***)
  - per specificare se una data istruzione, o un dato blocco di istruzioni, debba essere eseguito più volte
    - un numero prefissato di volte, o
    - fintanto che una condizione booleana è valida
  - ***for***, ***while***, ***do-while***

- ***Subroutine*** (chiamate di ***funzioni***/***metodi***)
  - causano un ***salto*** del flusso di controllo, dalla funzione chiamante alla funzione chiamata, per poi ritornare alla funzione chiamante

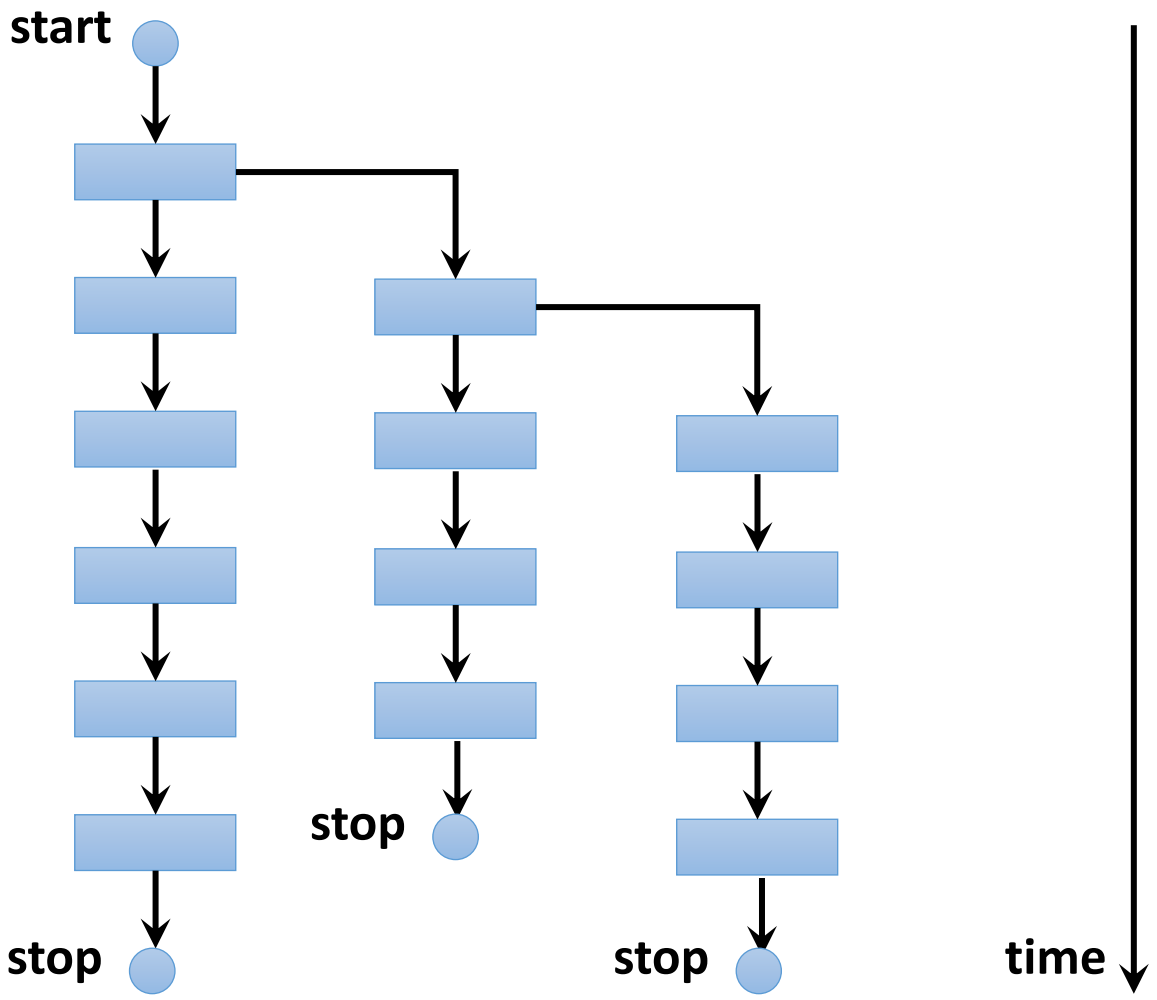
Flusso *sequenziale* di esecuzione

- Un **flusso di esecuzione** (o di **controllo**) si dice ***sequenziale*** se ha un solo inizio, una sola fine, e ad ogni istante durante la sua esecuzione c'è un singolo punto di esecuzione

- Un **programma** si dice ***sequenziale*** se ha un flusso di esecuzione sequenziale, **altrimenti** si dice ***concorrente***

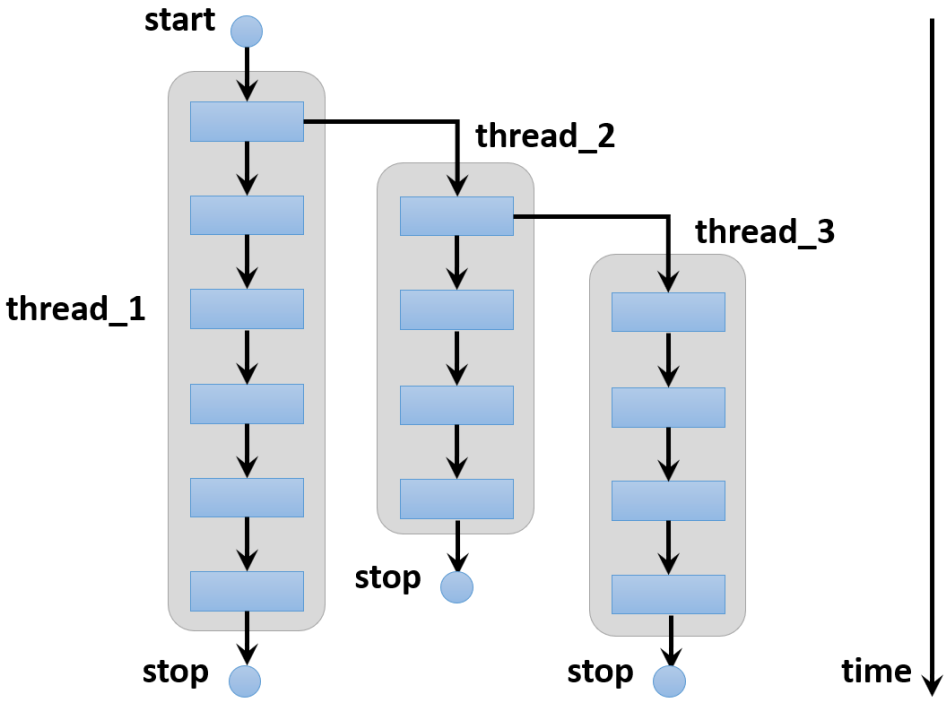


Programma concorrente



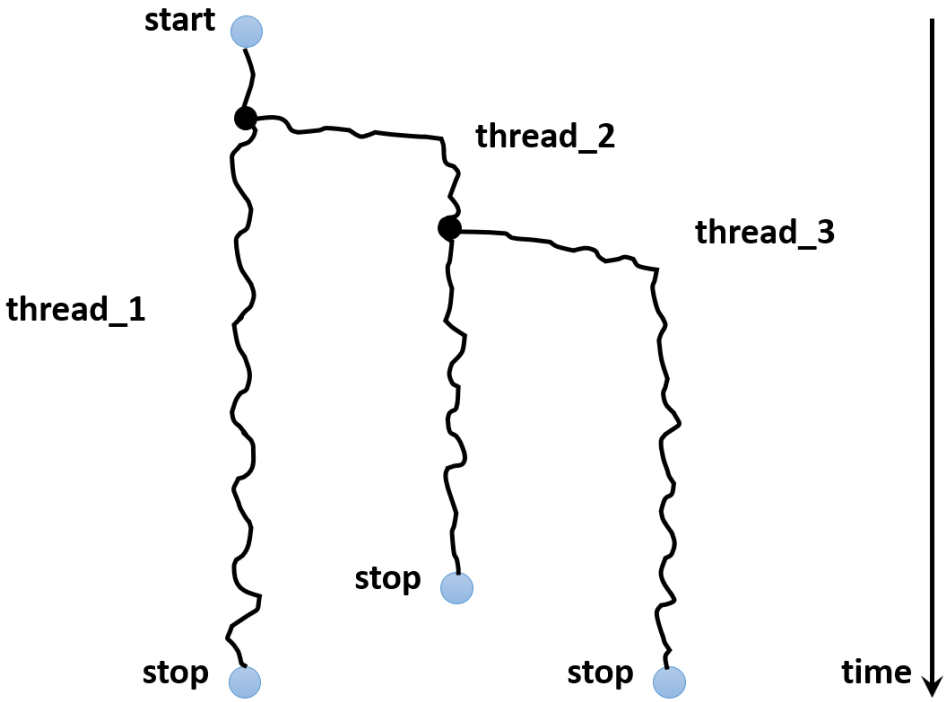
Thread 1/2

- Un ***thread*** (letteralmente «*filo*» o «*trama*») è un **singolo flusso sequenziale** di istruzioni all'interno di un programma
  - un programma sequenziale ha un **singolo** thread
  - un programma concorrente ha **due o più** thread



Thread 2/2

- Un ***thread*** (letteralmente «*filo*» o «*trama*») è un **singolo flusso sequenziale** di istruzioni all'interno di un programma
  - un programma sequenziale ha un **singolo** thread
  - un programma concorrente ha **due o più** thread

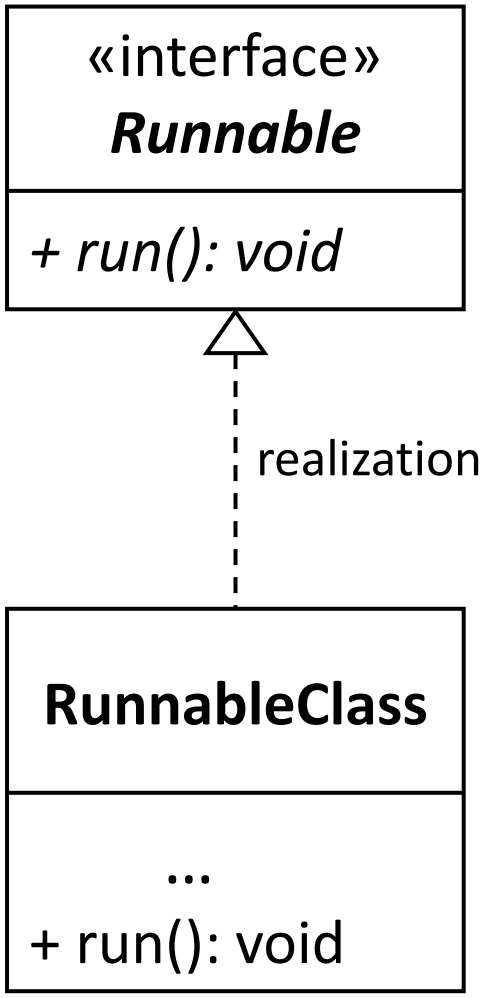


java.lang.*Runnable*

- L'interfaccia ***Runnable*** del pacchetto **java.lang** presenta un solo metodo:

  void run()

- Assicura quindi che un **qualunque oggetto** di una **qualunque classe** concreta che **implementa** tale interfaccia includa una propria implementazione del metodo ***run()***



Incapsulare un task in un oggetto 1/N

- L'interfaccia java.lang.***Runnable*** viene utilizzata tipicamente per **incapsulare** uno specifico **task** (o **comportamento**) in un oggetto
  - un **oggetto** *Runnable* **incapsula** il **task/comportamento** da eseguire nel metodo **run()** che deve obbligatoriamente includere
  - Es.: il metodo **executeTask()** riceve un oggetto ***Runnable*** ed eseguirà il task incapsulato in tale oggetto

```
public class TaskManager {
    ...
    public void executeTask(Runnable task) {
        task.run();
    }
    ...
}
```

Incapsulare un task in un oggetto 2/3

- Senza una sintassi apposita, per ottenere un oggetto *Runnable* che incapsuli un task desiderato è necessario creare preliminarmente un'apposita classe Java
- Tale casse deve implementare l'interfaccia *Runnable*

```
public class MyRunnableClass implements Runnable {
    ...
    public void run() {
        //my specific task
    }
    ...
}
```

Incapsulare un task in un oggetto 3/3

- La sintassi Java appena esaminata illustra come creare una classe *Runnable* i cui oggetti implementano un task specifico
- Sfruttando il polimorfismo (binding dinamico) è possibile realizzare un metodo **executeTask()** il cui comportamento è incapsulato in un oggetto esterno al codice del metodo

```
public void test() {
    ...
    TaskManager taskManager = new TaskManager();
    Runnable mySpecificTask = new MyRunnableClass();
    taskManager.executeTask(mySpecificTask);
    ...
}
```

Istanziazione «veloce» di oggetti *Runnable 1/2*

- Fortunatamente Java offre una sintassi molto più snella per incapsulare un task desiderato in un oggetto di tipo *Runnable*, evitando di definire una classe apposita che implementa tale interfaccia

```
new Runnable() {
    public void run() {
        //my specific task
    }
};
```

Istanziazione «veloce» di oggetti *Runnable 2/2*

- L'**oggetto** *Runnable* così istanziato è associato ad una classe concreta **anonima** generata dal compilatore Java
- Ovviamente, può essere associato ad una variabile riferimento di tipo *Runnable* (es. mySpecificTask)

```
Runnable mySpecificTask = new Runnable() {
                              public void run() {
                                  //my specific task
                              }
                          };
```

Istanziazione di oggetti di classi anonime

- Quanto detto è generalizzabile al caso di oggetti di una generica interfaccia

```
new AnyInterface() {
    public void firstMethod() {
        //first method code
    }
    public void secondMethod() {
        //second method code
    }
    ...
    public void LastMethod() {
        //last method code
    }
};
```

Istanziazione nell'argomento di un metodo

- Java offre una sintassi ancor più compatta: è possibile istanziare l'oggetto che incapsula il comportamento desiderato direttamente nell'**argomento** del metodo che lo utilizza

```
public void test() {
    ...
    TaskManager taskManager = new TaskManager();
    //Runnable mySpecificTask = new MyRunnableClass();

    taskManager.executeTask(new Runnable() {
        public void run() {
            //my specific task
        }
    });
    ...
}
```

Thread in Java *JECs* (*Java Essential Classes*)

- La libreria di base di Java (*JECs Java Essential Classes*), offre **due meccanismi** per eseguire un determinato task in un nuovo **thread**

- **Incapsulare** il codice del task nel metodo ***run()*** di un oggetto java.lang.*Runnable*, e incapsulare a sua volta questo oggetto in un oggetto della classe java.lang.Thread

- **Estendere** la classe java.lang.Thread, che implementa di per sé l'interfaccia java.lang.*Runnable*, implementare il task nel metodo **run()** di tale sottoclasse, e istanziare un oggetto della nuova classe estesa

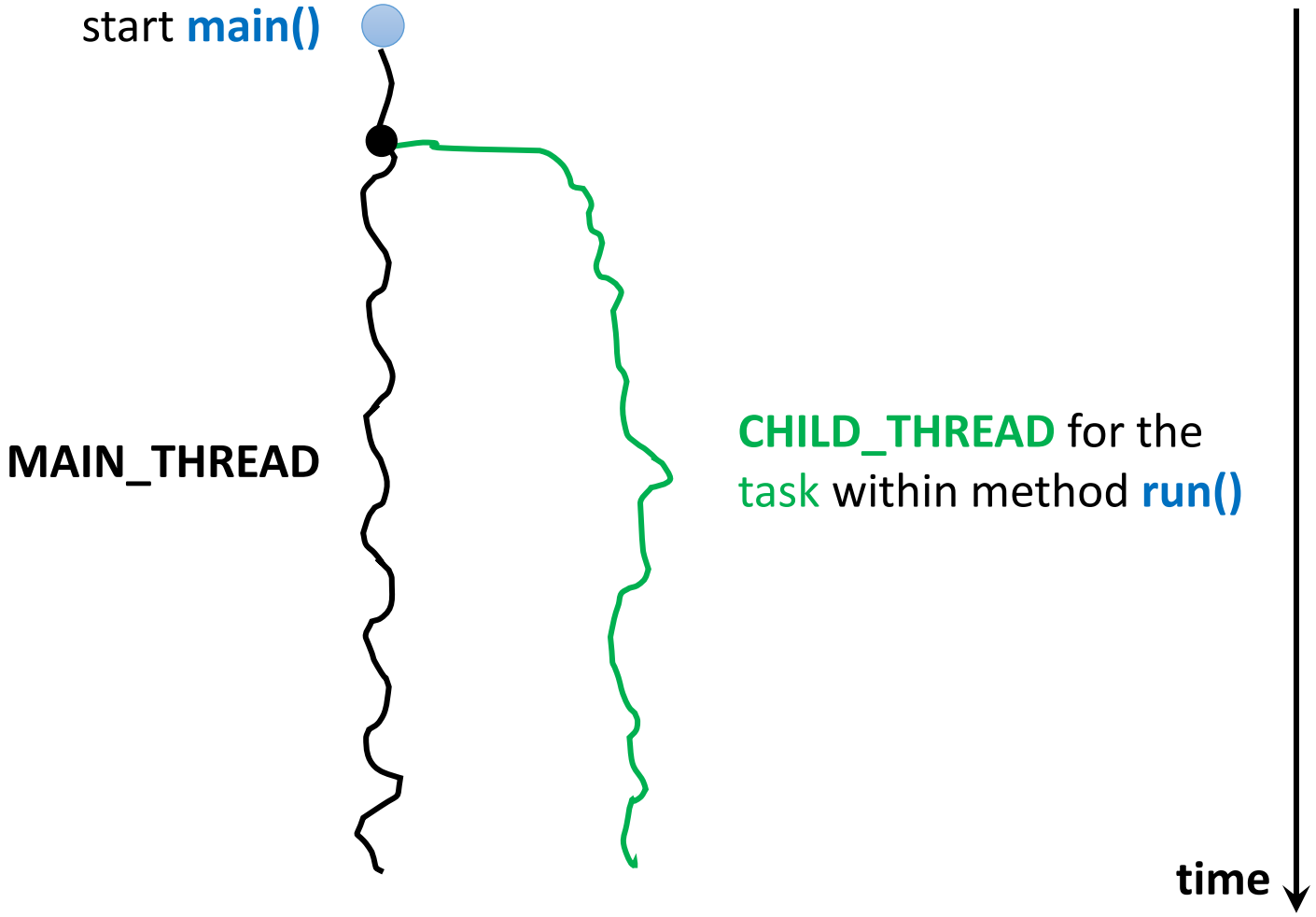
Thread in Java (JECs) – Prima modalità 1/3

```
public class HelloRunnable implements Runnable {

    public void run() {
        //specific task to be parallelized
    }

    public static void main(String args[]) {
        Thread thread = new Thread(new HelloRunnable());
        t.start(); // a new thread starts
        ...
        //other instructions executed in the MAIN thread
        ...
    }
}
```

Thread in Java (JECs) – Prima modalità 2/3



Thread in Java (JECs) – Prima modalità 1/3

```
public class Test {

   public static void main(String args[]) {

      Thread thread = new Thread(new Runnable() {
                                        public void run() {
                                           //specific task
                                        }
                                  });

      t.start(); // a new thread starts
      ...
      //other instructions executed in the MAIN thread
      ...
   }
}
```

The End