# Java Threads

# I Thread in Java

I Thread consentono ad un programma di funzionare più efficientemente facendo più cose contemporaneamente.

I Thread possono essere utilizzati per eseguire compiti complicati in background senza interrompere il programma principale.

# Creazione di un Thread

Ci sono due modi per creare un thread.

1. Può essere creato estendendo la classe `Thread` e sovrascrivendo il suo metodo run():

```
public class Main extends Thread {
  public void run() {
    System.out.println("Questo codice sta eseguendo in un thread");
  }
}
```

2. Oppure può essere creato implementando l'interfaccia `Runnable` e implementando il suo metodo run():

```
public class Main implements Runnable {
  public void run() {
    System.out.println("Questo codice sta eseguendo in un thread");
  }
}
```

## Esecuzione di un Thread

Se la classe estende la classe Thread, il thread può essere eseguito creando un'istanza della classe e chiamando il suo metodo `start()`:

```
public class Main extends Thread {
  public static void main(String[] args) {
    Main thread = new Main();
    thread.start();
    System.out.println("Questo codice è al di fuori del thread");
  }
  public void run() {
    System.out.println("Questo codice sta eseguendo in un thread");
  }
}
```

Se la classe implementa l'interfaccia Runnable, il thread può essere eseguito passando un'istanza della classe al costruttore di un oggetto `Thread` e quindi chiamando il metodo `start()` del thread:

```
public class Main implements Runnable {
  public static void main(String[] args) {
    Main obj = new Main();
    Thread thread = new Thread(obj);
    thread.start();
    System.out.println("Questo codice è al di fuori del thread");
  }
  public void run() {
    System.out.println("Questo codice sta eseguendo in un thread");
  }
}
```

# Differenze tra "estensione" e "implementazione" dei Thread

La differenza principale è che quando una classe estende la classe `Thread`, non è possibile estendere nessun'altra classe, ma implementando l'interfaccia `Runnable`, è possibile estendere anche un'altra classe, ad esempio:

```
class MyClass extends OtherClass implements Runnable
```

## Problemi di Concorrenza

Poiché i thread vengono eseguiti contemporaneamente ad altre parti del programma, non c'è modo di sapere in quale ordine verrà eseguito il codice. Quando i thread e il programma principale leggono e scrivono le stesse variabili, i valori sono imprevedibili. I problemi che ne derivano sono chiamati problemi di concorrenza.

# Esempio

Un esempio di codice in cui il valore della variabile `amount` è imprevedibile:

```
public class Main extends Thread {
  public static int amount = 0;

  public static void main(String[] args) {
    Main thread = new Main();
    thread.start();
    System.out.println(amount);
    amount++;
    System.out.println(amount);
  }

  public void run() {
    amount++;
  }
}
```

Per evitare problemi di concorrenza, è meglio condividere il minor numero possibile di attributi tra i thread. Se gli attributi devono essere condivisi, una possibile soluzione è utilizzare il metodo `isAlive()` del thread per verificare se il thread ha terminato l'esecuzione prima di utilizzare eventuali attributi che il thread può cambiare.

```
public class Main extends Thread {
  public static int amount = 0;

  public static void main(String[] args) {
    Main thread = new Main();
    thread.start();
    // Wait for the thread to finish
    while(thread.isAlive()) {
        System.out.println("Waiting...");
    }
    // Update amount and print its value
    System.out.println("Main: " + amount);
    amount++;
    System.out.println("Main: " + amount);
  }
  public void run() {
    amount++;
  }
}
```

# Sincronizzazione dei Thread

Se invece abbiamo dei thread in continua esecuzione e vogliamo che un thread aspetti che un altro thread abbia finito di eseguire un'operazione, possiamo utilizzare la parola chiave `synchronized`.
Synchronized può essere utilizzato in due modi, su un blocco di codice o su un metodo.
Quando un blocco di codice è sincronizzato, solo un thread può eseguire il blocco di codice alla volta.

```java
public class Main extends Thread {
  public static int amount = 0;

  public static void main(String[] args) {
    Main thread = new Main();
    thread.start();
    // Wait for the thread to finish
    while(thread.isAlive()) {
        System.out.println("Waiting...");
    }
    // Update amount and print its value
    System.out.println("Main: " + amount);
    synchronized(thread) {
        amount++;
    }
    System.out.println("Main: " + amount);
  }
  public void run() {
      synchronized(this) {
          amount++;
      }
  }
}
```