JAVA THREADS IN A NUTSHELL

Due modi per creare thread:

- estendendo la classe `Thread`
- implementando l'interfaccia `Runnable`

I due modi possono essere combinati se si vuole avere una classe che goda di **ereditarietà multipla**.
Entrambi i modi prevedono l'implementazione di un metodo `run()`.

La classe `Thread` è a sua volta un'implementazione di `Runnable`.

Estendere la classe `Thread`

```
// Estendo la classe Thread.
public class MyThread extends Thread {
    public MyThread(String str) {
        super(str); // può essere omesso nel caso non si
sia interessati al nome.
    }
    // Implemento il metodo run.
    public void run() {
        for (int i = 0; i < 10; i++) {
            // getName è un metodo della superclasse
Thread.
            System.out.println(getName() + "@ iterazione
n.: " + i);
            try {  // sleep() può generare eccezioni.
                sleep(1000);
            } catch (InterruptedException e) { }
        }
        System.out.println(getName() + ": passo e
chiudo!");
    }
}
```

```
public class TestMyThread {
```

```
    public static void main(String[] args) {

        // Creo i thread
        MyThread th1 = new MyThread("Thread 1");
        MyThread th2 = new MyThread("Thread 2");

        // l'avvio dei thread avviene attraverso il metodo
start.
        // il metodo start lancia implicitamente il metodo
run().
        th1.start();
        th2.start();

        try { // join() può generare un'eccezione.
            th1.join();
            th2.join();
            System.out.println("main: Thread terminati\n");
        }
        catch (InterruptedException e) {
            e.printStackTrace();
            System.err.println("main: Thread
interrotto\n");
        }
    }
}
```

Metodi della classe `Thread`

Metodo	Descrizione
`void start()`	Richiama il medoto `run()` implementato nella classe figlia. A fine esecuzione del metodo `run()` il thread termina. Invocazioni multiple di `start()` sullo stesso thread generano una `InvalidThreadStateException`
`boolean isAlive()`	Indica lo stato del thread. Se è running oppure no.
`void join()` `void join(long`	Permette di attendere la fine di un thread. Se è specificato il parametro `millis` il chiamante attende

`millis)`	Descrizione solo per il tempo stabilito.
`void sleep(long millis)`	Fa attendere il chiamante per il tempo specificato. Può generare eccezioni. È un metodo `static`.
`Thread currentThread()`	Metodo statico della classe `Thread` che restituisce il thread id del thread chiamante.
`String toString()`	Restituisce una rappresentazione descrittiva del thread, con nome e gruppo.

Implementazione dell'interfaccia `Runnable`

```
// Sviluppo di classe thread attraverso
// l'implementazione dell'interfaccia Runnable
public class ThreadedInt extends MyInt implements Runnable
{

    public ThreadedInt(String name) {
        super((int) (Math.random() * 1000));
        threadName = name;
    }
    // Implementoil metodo run di Runnable.
    public void run() {
        for (int i = 0; i < 10; i++) {
            System.out.println(threadName + "@" + i + "->"
+getVal() );
            try {
                // Utilizzo il riferimento perchè a
differenza del caso
                // precedente non sto estendendo Thread.
                Thread.sleep(100);
            }
            catch (InterruptedException e) { }
        }
    }
}

// Classe MyInt. Inizializza e ritorna un numero intero.
class MyInt {
    int val;
```

```
    MyInt(int initVal) {
        val = initVal;
    }
    int getVal() {
        return val;
    }
}
```

```
public class TestRunnable {
    public static void main(String[] args) {
        ThreadedInt tn1, tn2;
        Thread th1, th2;

        // Istanzio il primo thread, prima generando
l'oggetto Runnable
        // implementato dalla classe ThreadedInt,
passandolo poi al
        // costruttore della classe Thread.
        tn1 = new ThreadedInt("Thread 1");
        th1 = new Thread(tn1);
        th1.start();

        // Istanzio il secondo thread come sopra.
        tn2 = new ThreadedInt("Thread 2");
        th2 = new Thread(tn2);
        th2.start();

        try {
            th1.join();
            th2.join();
            System.out.println("\n" + th1.toString()
                        + " " + th2 + " terminated"
            );
        }
        catch (InterruptedException e) {
        }
    }
}
```

Confronto

`Thread`	`Runnable`
Implementazione più semplice ma più limitata	Flessibile ma più complesso da implementare

Costrutti per la sincronizzazione: `synchronized`

Il costrutto `synchronized` agisce da monitor/semaforo per quanto riguarda l'accesso esclusivo ad una risorsa condivisa. Se un thread è in esecuzione al suo interno nessun'altro può dunque accedervi.

Applica quindi un lock sulla risorsa e lo rilascia in automatico al termine della **sezione critica**.

Può essere:

- applicato alle singole risorse / porzioni di codice
- utilizzato come modificatore di un metodo

`synchronized` applicato agli oggetti

```
synchronized(this) {
    num = 0;
    generate();
    num++;
    if (num > 0) notifica();
}
```

`synchronized` come modificatore di metodo

```
public class EsecSingola {
    private int value;

    public EsecSingola() {
        value = 10;
    }

    // Resetta. Sezione critica.
    synchronized public void reset()
```

```
            if (value == 0) value = 10;
        }

        // Processa il dato. Sezione critica.
        synchronized public void elabora() {
            if (value > 0) {
            --value;
            }
        }

        // Ritorna il valore.
        public int getValue() {
            return value;
        }
}
```

Metodi per la sincronizzazione: `wait()`, `notify()`, `notifyAll()`

La classe `Object` mette a disposizione i metodi di sincronizzazione `wait()`, `notify()` e `notifyAll()`, che possono essere richiamati all'interno di un blocco `synchronized`.

Metodo	Azione
`wait()` `wait(long timeout)`	Blocco l'esecuzione del thread chiamante fino a che un altro thread non lo risveglia attraverso `notify()`. La risorsa interessata dalla `wait()` viene rilasciata, i lock sugli altri oggetti invece persistono. Nella sua versione con timeout blocca il thread per al massimo `timeout` millisecondi
`notify()`	Risveglia uno ed uno solo dei thread in attesa sulla risorsa. Se più thread sono in attesa, la scelta di quale svegliare viene fatta dalla JVM.

Metodo	Azione
`notifyAll()`	Risveglia tutti i thread che aspettano sull'oggetto in questione. Più sicuro rispetto a `notify()` in quanto il thread risvegliato potrebbe non essere in grado di procedere e venire sospeso, generando un blocco.