Esempi (di “runnable”)

Un thread che conta fino a 10

```
public class ContaFinoADieci implements Runnable {

   public void run() {
     for(int i=0;i<10;i++) {
        System.out.print((i+1)+" ");
     }
   }
}
```

Utilizzazione:

```
ContaFinoADieci contatore = new ContaFinoADieci();
Thread threadContatore = new Thread(contatore);
...
threadContatore.start();
```

Alcune osservazioni:

- l'oggetto `contatore` viene creato invocando il costruttore della superclasse (Object in questo caso)
- la creazione dell'oggetto di tipo thread non implica la sua attivazione
- l'attivazione del thread non implica l'immediata esecuzione
- la schedulazione relativa del thread appena attivato rispetto al thread chiamante dipende dalla JVM.

Altro esempio

```
public class DueThread implements Runnable {

    public void run() {
        for(int i=0;i<10;i++) {
            System.out.print(this+" "+(i+1)+" ");
        }
        System.out.println(" ");
    }

    public static void main(String args[]) {
        DueThread t1 = new DueThread();

        Thread threadContatore = new Thread(t1);
        threadContatore.start();
        System.out.println("Thread contatore attivato");

        DueThread t2 = new DueThread();
        t2.run();
        return;
    }
}
```

Quando viene creato il secondo oggetto DueThread e se ne richiama esplicitamente il metodo run: il thread corrente (main) esegue le istruzioni del metodo run dell'oggetto t2 di tipo DueThread come se fossero parte del proprio corpo (senza commutazione di contesto/schedulazione).

Se avessimo fatto girare il codice seguente, invece, allora avremmo effettivamente creato un secondo thread:

```
public class TreThread implements Runnable {

    public void run() {
        for(int i=0;i<10;i++) {
            System.out.print(this+" "+(i+1)+" ");
        }
        System.out.println(" ");
    }

    public static void main(String args[]) {
        TreThread t1 = new TreThread();

        Thread threadContatore = new Thread(t1);
        threadContatore.start();
        System.out.println("Thread contatore attivato");

        TreThread t2 = new TreThread();
        Thread secondoContatore = new Thread(t2);
        secondoContatore.start();

        return;
    }
}
```

Modifichiamo leggermente il metodo run e il main come segue (riportiamo solo l’esempio TreThread):

```
public class TreThread implements Runnable {

    public void run() {
        System.out.println("Thread "+Thread.currentThread());   // <=
        for(int i=0;i<10;i++) {
            System.out.print(" "+(i+1)+" ");
        }
        System.out.println(" ");
    }
    public static void main(String args[]) {
        TreThread t1 = new TreThread();

        System.out.println("Thread main "+Thread.currentThread()); // <=
        Thread threadContatore = new Thread(t1);
        threadContatore.start();
        System.out.println("Thread contatore attivato");

        TreThread t2 = new TreThread();
        Thread secondoContatore = new Thread(t2);
        secondoContatore.start();
        return;
    }
}
```

Mandiamo in esecuzione le due versioni modificate:

```
> java DueThread
Thread main Thread[main,5,main]
Thread contatore attivato
Thread Thread[main,5,main]
1 2 3 4 5 6 7 8 9 10
Thread Thread[Thread-4,5,main]
1 2 3 4 5 6 7 8 9 10

> java TreThread
Thread main Thread[main,5,main]
Thread contatore attivato
Thread Thread[Thread-4,5,main]
 1  2  3  4  5  6  7  8  9  10
Thread Thread[Thread-5,5,main]
 1  2  3  4  5  6  7  8  9  10

>
```

Notare la differenza negli identificatori di thread restituiti (con `Thread.currentThread()`)

```
public class PingPong extends Thread {
    private String word;  // what word to print
    private int delay;    // how long to pause (in milliseconds)

    public PingPong(String whatToSay, int delayTime) {
        word = whatToSay;
        delay = delayTime;
    }

    public void run() {
        try {
            for (;;) {
                System.out.print(word + " ");
                sleep(delay);   // pause before printing next word
            }
        } catch (InterruptedException e) {
            return;   // end this thread
        }
    }
    public static void main(String[] args) {
        new PingPong("ping",  33).start();  // 1/30 second
        new PingPong("PONG", 100).start();  // 1/10 second
    }
}
```

- Possible effect of an execution of PingPong.main:

```
ping PONG ping PONG ping ping PONG ping ping ping
PONG ping ping ping PONG ping ping PONG ping ping
ping PONG ping ping PONG ping ping ping PONG ping
ping PONG ping ping ping PONG ping ping PONG ping
...
```

- Different execution may give different results.
- The interleaving (and interaction) between threads depends on many factors and it is in general unpredictable.

Attesa terminazione di un thread

Dopo aver eseguito un codice tipo:

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

possiamo attendere che le attivita' del thread t siano terminate invocando una

```
try {
  t.join();
} catch (InterruptException e) {
  ...
}
```

Parametri ai thread

- Sia il metodo start che il metodo run sono privi di parametri (void start() e void run())

- Due thread, uno che enumera i numeri pari fino a N e uno che enumera i numeri dispari fino ad N. Entrambi potrebbero avere un corpo tipo:

```
public void run() {
    for(int i=0;i<limiteSuperiore;i++) {
        if(pari) {                  // stampa il numero solo se e' pari
            if(i%2 == 0)
                System.out.println("pari "+i);
        } else {                    // stampa il numero solo se e' dispari
            if(i%2 == 1)
                System.out.println("dispari "+i);
        }
    }
}
```

- il primo thread potrebbe essere attivato in un contesto dove pari vale true e il secondo in un contesto dove vale false

- L’unico modo di rendere disponibili valori specifici ad ogni thread, pur uilizzando lo stesso codice per la classe che implementa Runnable o estende Thread e’ quello di utilizzare le variabili d’istanza

- Esempio: una classe EnumeratorePariDispari

```
public class EnumeratorePariDispari extends Thread {

    private int limiteSuperiore;
    private boolean pari;

    public EnumeratorePariDispari(int n, boolean pari) {
        limiteSuperiore = n;
        this.pari = pari;
    }

    public void run() {
        for(int i=0;i<limiteSuperiore;i++) {
            if(pari) {                  // stampa il numero solo se e' pari
                if(i%2 == 0)
                    System.out.println("pari "+i);
            } else {                    // stampa il numero solo se e' dispari
                if(i%2 == 1)
                    System.out.println("dispari "+i);
            }
        }
    }

}
```

- Utilizzazione:

```
public static void main(String [] args) {
    int n = Integer.parseInt(args[0]);
    Thread tp = new EnumeratorePariDispari(n,true);
    Thread td = new EnumeratorePariDispari(n,false);

    System.out.println("Attivazione thread pari e dispari fino a "+n);
    tp.start(); td.start();
    System.out.println("Attesa terminazione thread pari e dispari");
    try {
        tp.join(); td.join();
    } catch (InterruptedException e) {
        System.out.println(e);
    }
    System.out.println("Thread pari e dispari terminati");
}
```

- Esecuzione:

```
> java EnumeratoreDariDispari 5
Attivazione thread pari e dispari fino a 5
pari 0
pari 2
pari 4
dispari 1
dispari 3
Attesa terminazione thread pari e dispari
Thread pari e dispari terminati
>
```

- Come vedremo, la condivisione di variabili tra thread puo’ portare a seri problemi a livello di programmazione