Definizione di un thread

Come sottoclasse di Thread

```
public class IlThread extends Thread {
   ...
   public void run() {   ... // corpo del thread }
}
```

Come estensione dell'interfaccia Runnable

```
public class IlThread implements Runnable {
   ...
   public void run() {  ... // corpo del thread }
}
```

Creazione di un thread

Definito sottoclassando Thread

```
IlThread tid = new IlThread(...);
```

Definito implementando Runnable

```
Thread tid = new Thread(new IlThread(...));
```

Thread demoni

```
tid.setDaemon(true); // non si farà mai più join sul thread
```

Attivazione di un thread

```
tid.start();
```

Attesa della terminazione di un thread

```
try { tid.join(); } catch (InterruptedException e) { ... }
```

Interruzione di un thread

```
tid.interrupt(); // setta il flag e interrompe se in attesa
```

Controllo da parte di un thread se sia stato interrotto

```
if(isInterrupted()) { ... } // non resetta il flag di interrotto

if(interrupted()) { ... } // resetta il flag di interrotto

try { operazioneBloccante( ... ) }
        catch (InterruptedException e) { ... }
```

Terminazione programma con thread

Main termina a seguito della return se non ci sono più thread non demoni attivi

Main termina a seguito di una exit()

Terminazione di un thread

Quando esegue una return

Terminazione di un thread demone

Quando termina il programma che lo ha lanciato

Passaggio di parametri ad un thread

```
public class IlThread extends Thread {
   String s = null;
   int i = 0;
```

```
  ...
  public IlThread(int i, String s) {
       this.s = s;
       this.i = i;
  }
  ...
  public void run() {
       ... // usa s,i
  }
  ...
}
```

Recupero di parametri di uscita di un thread

```
public class IlThread extends Thread {
  ...
  String s = null;
  int i = 0;
  ...
  public IlThread(int i, String s) {
       this.s = s;
       this.i = i;
  }
  ...
  public void run() {
       ... // usa s,i
  }
  ...
  public String getS() { return(s); }
  public int getI() { return(i); }
  ...
}
```

```
public static void main(String [] args) {
  IlThread tid = new IlThread(19,"ciao");
  tid.start();
  ...
  tid.join();
  System.out.println(tid.getS());
  System.out.println(tid.getI());
  return;
}
```

Priorità di un thread

compresa fra Thread.MIN_PRIORITY e Thread.MAX_PRIORITY, normalmente pari a Thread.NORM_PRIORITY

```
tid.setPriority(int newPriority);
int pri = tid.getPriority();
```

Attesa per un intervallo di tempo

```
try { Thread.sleep(int millis); } catch (InterruptedException E) {...}
```

Utilizzazione di un Timer per schedulare un'attività ad intervalli fissi

Sostituisce un corpo di thread tipo:

```
public void run() {
  while(true) {
       // attività ....
       sleep(intervallo);
  }
}
```

```
TimerTask attivita1 = new TimerTask() {
        public void run () { ... attività ... }};
Timer timer = new Timer();
Timer.schedule(attivita1,0,intervallo);
```

Gruppi di thread

```
ThreadGroup tg = new ThreadGroup("nome");
Runnable tr = new ThreadImplementaRunnable(...);
Thread tid = new Thread(tg,tr);
```

possono essere tutti interrotti con una

```
tg.interrupt();
```

Sincronizzazione fra thread

Attesa di un evento

```
while(condizioneEvento!=true) {
  try {  wait(); } catch(InterruptedException e) { ... }
}  // evento verificato, si può proseguire di conseguenza
```

Segnalazione del verificarsi di un evento (generico)

```
notify();     // una delle wait viene sbloccata

notifyAll(); // tutte le wait vengono sbloccate
```

Mutua esclusione su un blocco di codice

```
synchronized(oggetto) {
        ... // codice in mutua esclusione con lock su oggetto
}
```

Mutua esclusione nell'esecuzione di metodi (aka "Monitor")

```
public class EsempioMutexMetodi {
  ...
  public synchronized metodo1(...) {...}
  ...
  public synchronized metodo2(...) {...}
}
```

equivale sostanzialmente a scrivere:

```
public metodo1(...) {
   synchronized(this) {
        // corpo originale del metodo1
   }
}
```