

# CORSO JAVA EE

# Indice generale

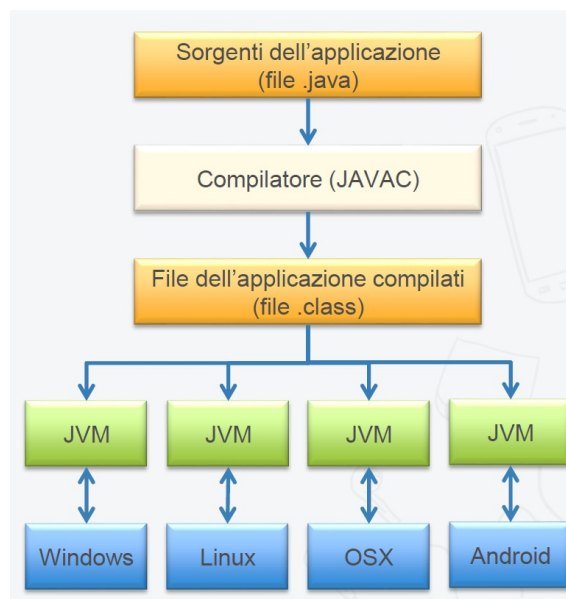
INTRODUZIONE A JAVA.....	3
JAVA PLATFORM.....	4
VANTAGGI E SVANTAGGI JAVA.....	4
JRE, JDK E AMBIENTI DI SVILUPPO.....	5
JAVA RUNTIME ENVIRONMENT (JRE).....	5
JAVA DEVELOPMENT KIT (JDK).....	5
AMBIENTI DI SVILUPPO CON INTERFACCIA GRAFICA.....	5
TIPI DI SOFTWARE.....	6
STANDALONE.....	6
CLIENT / SERVER.....	6
APPLICAZIONE WEB.....	6
SVILUPPO SOFTWARE: WATERFALL VS AGILE.....	7
METODOLOGIA WATERFALL.....	7
METODOLOGIA AGILE.....	8
THREAD E CONCORRENZA.....	9
PROCESSO.....	9
THREAD.....	9
CONCORRENZA.....	11
CICLO DI VITA DI UN THREAD.....	12
THREAD PRIORITY.....	13
CREARE UN THREAD IN JAVA.....	14
MULTITHREADING.....	15
CONCORRENZA IN JAVA.....	18

# INTRODUZIONE A JAVA

JAVA è un linguaggio di programmazione object oriented (orientato agli oggetti). La programmazione a oggetti è un paradigma ancora oggi molto utilizzato, perché ci consente di modellare al meglio delle situazioni reali.

La sintassi del linguaggio JAVA è molto simile ai linguaggi C e C++, da cui eredita parecchie caratteristiche. Per certi versi, però, JAVA è più semplice da utilizzare rispetto i due linguaggi citati precedentemente.

La caratteristica principale che ha reso JAVA così popolare è la portabilità, cioè l'essere indipendente dal sistema operativo su cui viene eseguito un software. Gli elementi fondamentali che rendono JAVA un linguaggio di programmazione portabile sono la Java Virtual Machine (Macchina virtuale o JVM) e la Java Platform.



Appunto, portabilità vuol dire scrivere il proprio codice sorgente una sola volta, compilarlo ed eseguirlo su qualsiasi dispositivo dotato di una macchina virtuale. Un programma JAVA è rappresentato da uno o più file.java, al cui interno sono presenti i nostri sorgenti, ovvero il codice che scriviamo. Attraverso il compilatore chiamato JAVAC, il nostro codice sorgente viene tradotto in un linguaggio intermedio, chiamato bytecode. Tutti questi file compilati avranno l'estensione .class e potranno essere interpretati dalla JVM, che è un processore virtuale che legge il/i nostro/i file.class e traduce il bytecode al loro interno in linguaggio macchina. Quindi, se su dispositivi differenti, con sistema operativo differente, abbiamo la stessa JVM, possiamo eseguire il nostro software senza necessità di ricompilarlo ogni volta.

Oggi JAVA è utilizzato per scrivere:

- applicazioni web: La maggior parte delle applicazioni enterprise (applicazioni aziendali) utilizzano JAVA, soprattutto nella parte backend;
- applicazioni per smartphone e tablet: Android, per esempio, è un sistema operativo scritto in JAVA, così come tutte le app scritte per questo sistema operativo;
- applicazioni per decoder digitali, elettrodomestici e così via.

# JAVA PLATFORM

La Java platform è una piattaforma composta da due componenti:

- JVM, di cui abbiamo già parlato precedentemente;
- API (Application Programming Interface), cioè un set di librerie (componenti software) messi a disposizione degli sviluppatori per poter scrivere software JAVA.

La Java Platform è disponibile in 3 configurazioni:

- Java Standard Edition (JSE): mette a disposizione il set standard di API per poter scrivere applicazioni standalone (programma che può funzionare senza che siano richiesti altri componenti o addirittura senza sistema operativo), client e server, per accesso a database, per il calcolo scientifico e così via;
- Java Enterprise Edition (JEE): mette a disposizione, oltre alle API della Standard Edition, anche quelle per scrivere applicazioni distribuite (ad esempio applicazioni web);
- Java Micro Edition (JME): mette a disposizione le API per sviluppare applicazioni mobile.

## VANTAGGI E SVANTAGGI JAVA

Vantaggi

- indipendenza del linguaggio bytecode: consente di eseguire lo stesso programma su più dispositivi dotati di JVM;
- velocità di sviluppo;
- grande disponibilità di librerie;
- alta integrazione con il web.

Svantaggi

- velocità di esecuzione: il programma viene eseguito ed elaborato dalla JVM, che a sua volta traduce le istruzioni in linguaggio macchina. Pertanto il tempo di esecuzione è leggermente più lento rispetto ad un programma scritto in C++;
- attraverso la decompilazione è possibile risalire al codice sorgente.

# JRE, JDK E AMBIENTI DI SVILUPPO

## JAVA RUNTIME ENVIRONMENT (JRE)

JRE è un'implementazione della JVM, ed è necessario per l'esecuzione dei programmi JAVA.

Il JRE contiene:

- JVM.
- API standard di JAVA.
- Un launcher necessario per avviare i programmi già compilati in bytecode. Tutti i programmi partono sempre da una classe dotata di un metodo main()

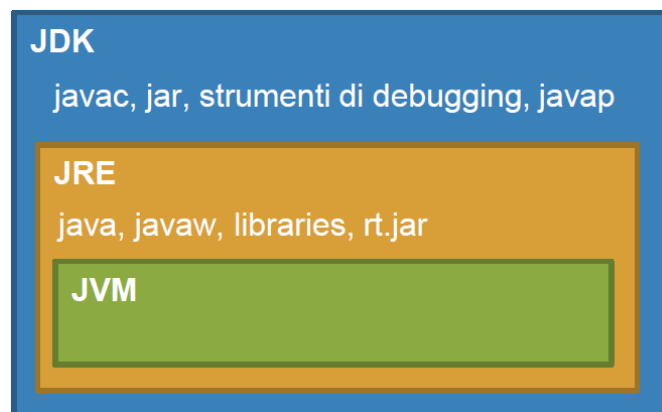
Il JRE deve essere installato su tutti i dispositivi che hanno necessità di eseguire software scritti in JAVA.

## JAVA DEVELOPMENT KIT (JDK)

JDK è un insieme di librerie e software messe a disposizione da Oracle, che consentono di sviluppare software scritti in JAVA.

JDK è un ambiente di sviluppo a tutti gli effetti, poiché appunto contiene tutte queste librerie che ci consentono di sviluppare software, ma è un ambiente di sviluppo a console, ovvero non è dotato di un'interfaccia grafica, ma le istruzioni si eseguono mediante il prompt di comandi. Gli ambienti di sviluppo più famosi, dotati di interfaccia grafica, sono Eclipse, Netbeans e IntelliJ IDEA.

Il JDK contiene librerie importanti per lo sviluppo, il JRE e la JVM.



## AMBIENTI DI SVILUPPO CON INTERFACCIA GRAFICA

Un ambiente di sviluppo integrato o IDE (Integrated Development Environment), rispetto al JDK, è un software dotato di interfaccia grafica che consente agli sviluppatori di creare software JAVA, semplificando la programmazione e la gestione dei file. È utile perché:

- consente di segnalare e visualizzare subito gli errori di sintassi all'interno del codice;
- consente di effettuare il debug in maniera semplice;
- offre una serie di strumenti e funzionalità di supporto allo sviluppatore.

# TIPI DI SOFTWARE

## STANDALONE

Il software standalone è un software che è installato all'interno di un sistema operativo ed è autonomo, perché non richiede l'uso di particolari componenti esterne con cui interagire (per esempio i server).

Esempi di software standalone sono:

- il pacchetto di software Microsoft (Word, Excel, ...);
- i software Adobe (Photoshop, InDesign, ...);
- alcune app per dispositivi mobile (ad es. l'app che visualizza le foto, l'app che gestisce i file del device).

## CLIENT / SERVER

Il software client/server è composto dai due componenti omonimi che definiscono il suo nome. La componente client è installata sul nostro dispositivo personale, mentre la componente server si trova su un dispositivo remoto e fornisce un servizio al client.

Esempi di sistemi client/server:

- File server: per la condividere i file;
- FTP server: per l'upload/download dei file;
- Database server: per la gestione dei dati;

Questa architettura, però, ha un limite importante. Questo limite riguarda la necessità di installare il software della componente client su ciascun terminale che deve accedere a questo software. Questo problema si risolve con le applicazioni web.

## APPLICAZIONE WEB

L'applicazione web è nata con lo scopo di migliorare l'utilizzo dei componenti client, senza la necessità di dover installare nuovi software. Un'applicazione web è accessibile mediante un normale browser.

L'applicazione web è un tipo di client/server evoluto, in cui è sostanzialmente il browser a fare da client, che scarica per noi tutte le varie informazioni (le pagine web) che ci servono per interagire con la componente server. In questo caso, il server e il client comunicano mediante protocollo HTTP.

Per accedere alle applicazioni web si utilizzano URL o link ipertestuali.

# SVILUPPO SOFTWARE: WATERFALL VS AGILE

Dopo aver visto i vari tipi di software, adesso vediamo come si sviluppa un software. Non ha senso parlare di sviluppo nel momento in cui abbiamo a che fare con degli script o piccoli software che devono svolgere un compito particolare, ma solo nel momento in cui abbiamo a che fare con software complessi o strutturati per la gestione di un'intera azienda.

Usare una metodologia per lo sviluppo di un software complesso è importante, perché consente di realizzare il software in maniera più organizzata e strutturata, limitando gli errori che uscirebbero fuori nel caso di uno sviluppo senza criterio.

Le metodologie per lo sviluppo di un software sono 2: waterfall e agile.

## METODOLOGIA WATERFALL

Nella metodologia waterfall (o classica) la sequenza delle fasi del ciclo di vita di un progetto software è strettamente sequenziale e, prima di passare alla fase successiva, è necessario che quella corrente sia terminata completamente.

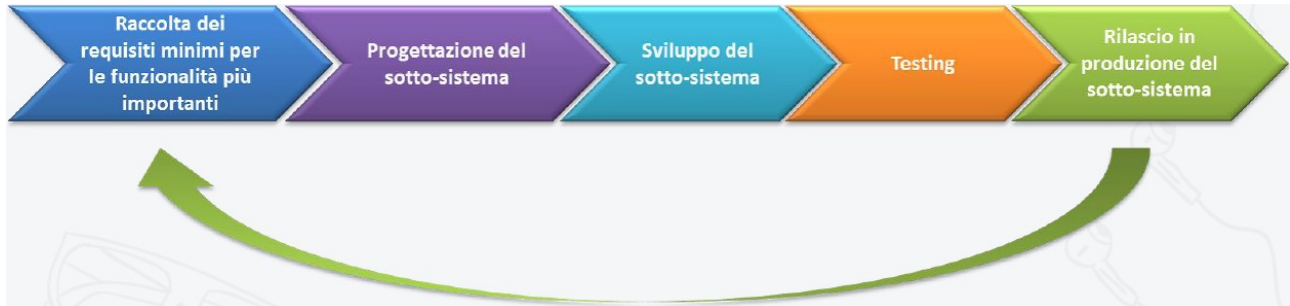


Quindi, in questo tipo di approccio raccogliamo i requisiti all'inizio, sulla base di quei requisiti viene sviluppato il software e, infine, facciamo vedere il nostro software completo solo al termine del rilascio in produzione.

Il limite principale di questa metodologia è quello di aver realizzato un prodotto che soddisfa perfettamente i requisiti inizialmente raccolti che, però, nel frattempo possono essere cambiati e quindi il software è diverso da quello atteso.

# METODOLOGIA AGILE

Nella metodologia agile abbiamo le stesse fasi della metodologia waterfall, ma il ciclo di vita di un progetto software è visto come una sequenza di tante iterazioni, dove ogni iterazione è un arco temporale che va dalle 2 alle 4 settimane.



In ciascuna iterazione prendiamo un pezzettino del nostro software sul quale vengono eseguite tutte le fasi, dopodiché si prende un altro pezzettino e si ritorna indietro nelle fasi per rieseguirle tutte e così via, fino alla realizzazione di tutti i pezzettini che comporranno il software finale.

Vantaggi di questa metodologia:

- coinvolgimento attivo del committente e degli utenti nel processo di sviluppo;
- aggiornamenti regolari e frequenti sullo stato dell'applicazione;
- validazione continua dei requisiti (dopo ogni iterazione);
- consegna rapida delle funzionalità di base;
- pianificazione fissa dei tempi di consegna per funzionalità;
- maggiori test, software migliore.

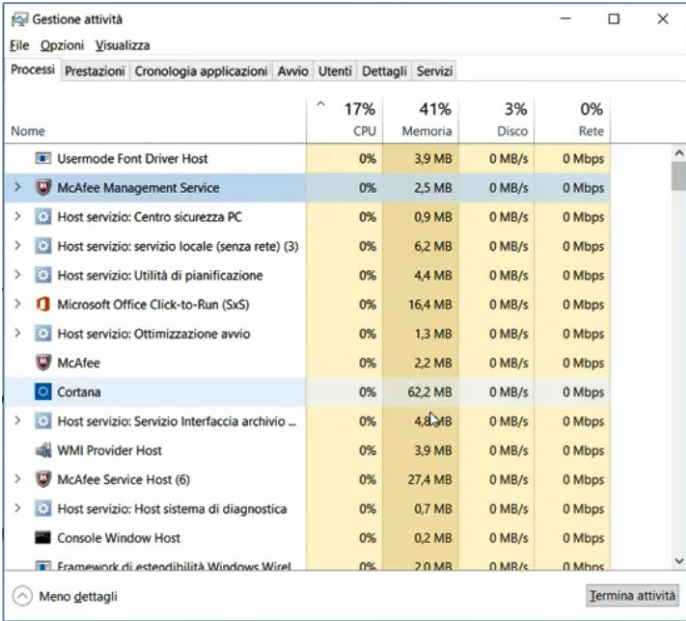


# THREAD E CONCORRENZA

## PROCESSO

Prima di capire cosa è un thread, dobbiamo definire cosa è un processo.

Quando installiamo su un dispositivo un software, copiamo i file compilati del software stesso all'interno del dispositivo, indipendentemente dal tipo di sistema operativo che vi è installato sopra. Il software in stato di esecuzione genera un processo, che avrà un ID univoco e occuperà una porzione di RAM dove salverà le sue informazioni. Quindi, un processo può essere definito come un'istanza del software che contiene le istruzioni e i dati che vengono elaborati durante l'esecuzione del software. Un esempio lo possiamo vedere sulla gestione attività del PC, dove sono elencati tutti i processi attivi in quel dato momento:



The screenshot shows the Windows Task Manager window titled "Gestione attività". The "Processi" tab is selected, displaying a list of running processes. The columns are: Nome, CPU (17%), Memoria (41%), Disco (3%), and Rete (0%). The processes listed include Usermode Font Driver Host, McAfee Management Service, Host servizio: Centro sicurezza PC, Host servizio: servizio locale (senza rete) (3), Host servizio: Utilità di pianificazione, Microsoft Office Click-to-Run (SxS), Host servizio: Ottimizzazione avvio, McAfee, Cortana, Host servizio: Servizio Interfaccia archivio ..., WMI Provider Host, McAfee Service Host (6), Host servizio: Host sistema di diagnostica, Console Window Host, and Framework di estendibilità Windows Wirel. The bottom of the window shows a "Meno gettagli" button and a "Termina attività" button.

Nome	CPU	Memoria	Disco	Rete
Usermode Font Driver Host	0%	3.9 MB	0 MB/s	0 Mbps
McAfee Management Service	0%	2.5 MB	0 MB/s	0 Mbps
Host servizio: Centro sicurezza PC	0%	0.9 MB	0 MB/s	0 Mbps
Host servizio: servizio locale (senza rete) (3)	0%	6.2 MB	0 MB/s	0 Mbps
Host servizio: Utilità di pianificazione	0%	4.4 MB	0 MB/s	0 Mbps
Microsoft Office Click-to-Run (SxS)	0%	16.4 MB	0 MB/s	0 Mbps
Host servizio: Ottimizzazione avvio	0%	1.3 MB	0 MB/s	0 Mbps
McAfee	0%	2.2 MB	0 MB/s	0 Mbps
Cortana	0%	62.2 MB	0 MB/s	0 Mbps
Host servizio: Servizio Interfaccia archivio ...	0%	4.8 MB	0 MB/s	0 Mbps
WMI Provider Host	0%	3.9 MB	0 MB/s	0 Mbps
McAfee Service Host (6)	0%	27.4 MB	0 MB/s	0 Mbps
Host servizio: Host sistema di diagnostica	0%	0.7 MB	0 MB/s	0 Mbps
Console Window Host	0%	0.2 MB	0 MB/s	0 Mbps
Framework di estendibilità Windows Wirel	0%	2.0 MB	0 MB/s	0 Mbps

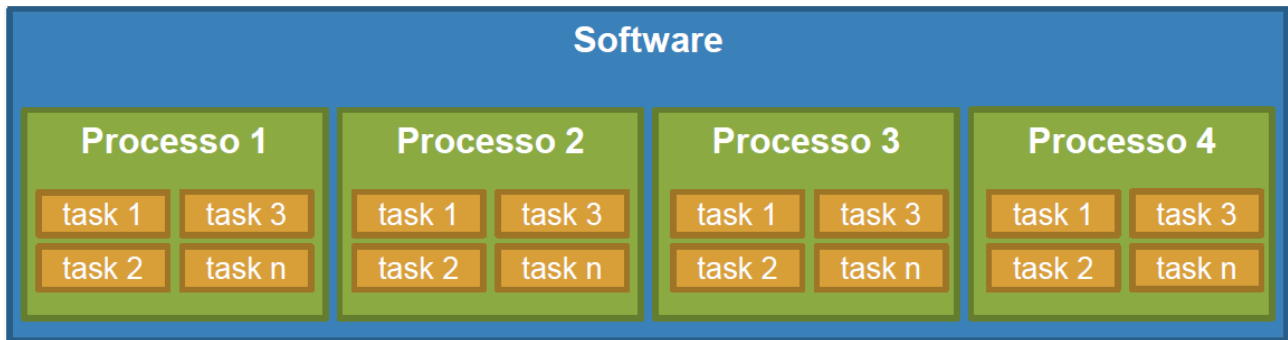
Siccome i sistemi operativi sono multitasking, possiamo eseguire più processi contemporaneamente e il sistema operativo assegnerà la CPU all'i-esimo processo, a seconda delle necessità.

## THREAD

Quando abbiamo la necessità di suddividere un processo in più sottoprocessi (o task), i quali devono essere indipendenti tra loro, a quel punto parliamo di thread. Un thread è un sottoprocesso che ha vita propria e svolge un determinato compito. Più thread possono essere eseguiti in maniera concorrente, a seconda delle necessità.

Quindi, in linea generale lo scopo dei thread è quello di dividere un processo in tante piccole parti, dove ogni parte lavora in maniera autonoma e deve gestire un evento o una risorsa.

Vediamo un esempio:



In questo esempio abbiamo il nostro software e 4 processi, che sono intesi come 4 istanze dello stesso software. Il numero di processi equivale al numero di volte che l'utente avvia lo stesso software. Ogni processo avrà i suoi task per la gestione delle singole attività e dei singoli eventi.

Esempi di utilizzo dei thread sono:

**Nel caso del browser web:**

- un thread che si occupa di scrivere il testo e visualizzarlo a video;
- un thread che si occupa di effettuare la ricerca.

**Nel caso del server web:**

- un thread che si occupa di accettare le richieste e creare altri thread che le gestiscono;
- un thread che si occupa di gestire una richiesta.

**Nel caso di word:**

- un thread che si occupa di scrivere i comandi digitati dall'utente;
- un thread che si occupa di cercare i sinonimi e gli errori di scrittura;
- un thread che si occupa di visualizzare gli errori di ortografia.

# CONCORRENZA

Ovviamente anche il sistema operativo stesso è composto da diversi software. Se il dispositivo non fosse in grado di gestire l'accesso concorrente a questi programmi, potremmo eseguire un solo programma alla volta. Ad esempio, non potremmo navigare su internet e al tempo stesso guardare un video, scrivere un documento, ascoltare musica e così via.

Si capisce a questo punto che la gestione della concorrenza diventa fondamentale, perché consente di sfruttare tutte le risorse hardware a disposizione per l'esecuzione di più software contemporaneamente.

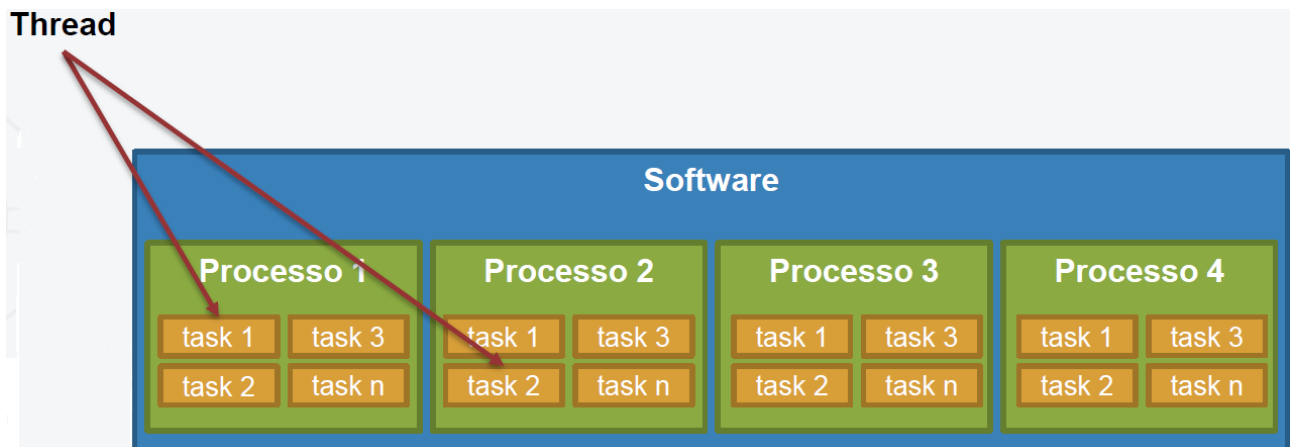
Quindi, la concorrenza è un insieme di concetti che consentono di rappresentare e descrivere l'esecuzione di due o più processi in maniera simultanea o non simultanea, a seconda della tipologia di architettura su cui stiamo eseguendo un software. Due o più processi, quindi due o più istanze di diversi software o dello stesso software, sono in esecuzione concorrente se vengono eseguiti in parallelo. A seconda del tipo di architettura, abbiamo 2 tipi di parallelismo:

- il parallelismo reale avviene quando più processi sono attivi su un dispositivo dotato di più processori, dove ogni processore prende in carico un processo, oppure quando più processi sono attivi su più dispositivi indipendenti tra loro e distribuiti.
- Il parallelismo apparente avviene quando più processi sono attivi su un dispositivo dotato di un processore.

# CICLO DI VITA DI UN THREAD

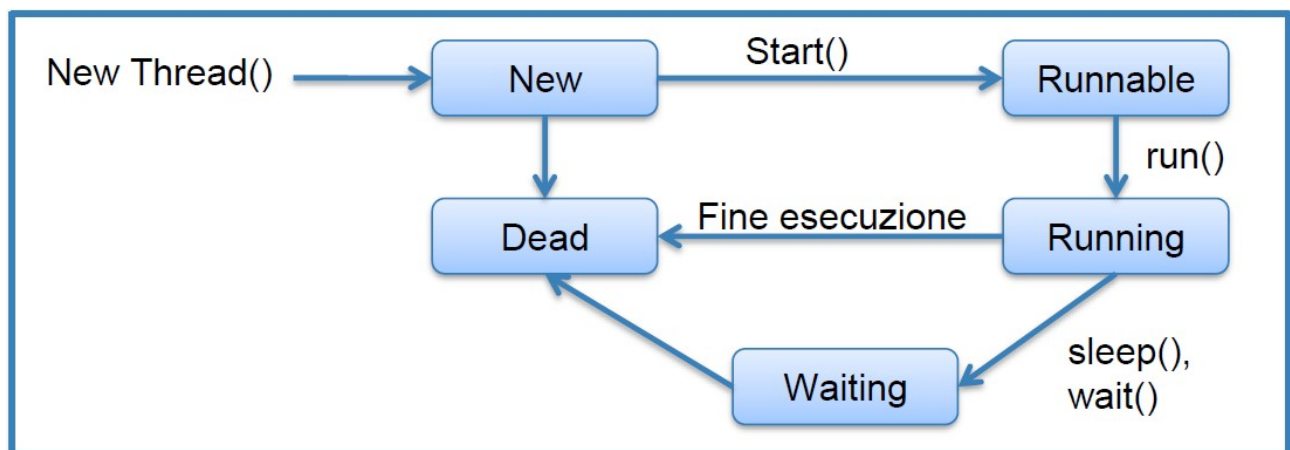
Un thread, dal punto di vista del processo, abbiamo già visto essere un sottoprocesso (o task), quindi un componente di un processo più ampio e complesso che si occupa di gestire una risorsa o un evento.

## Thread



Un thread ha un suo ciclo di vita, che comprende i seguenti stati:

- New
- Runnable
- Running
- Waiting
- Dead



Il primo stato in cui si trova un thread è New quando viene istanziato, poi abbiamo Runnable quando viene avviato, Running quando viene eseguito. Arrivato allo stato di Running, un thread può arrivare a Waiting, oppure a Dead, a seconda del caso se vogliamo sospendere o terminare il suo ciclo di vita.

# THREAD PRIORITY

La priority (o priorità) è l'informazione che indica allo scheduler il livello di importanza di un thread rispetto agli altri. Lo scheduler è un software che si occupa di gestire l'esecuzione dei thread durante l'esecuzione di un processo.

Facendo attenzione, però, su un concetto, non tutti i sistemi operativi garantiscono che l'ordine di esecuzione dei thread è determinato sulla base della thread priority che abbiamo stabilito. Tuttavia, se un certo numero di thread sono bloccati e in attesa di essere eseguiti, il primo che verrà sbloccato dallo scheduler sarà quello che ha priorità maggiore. In questo modo evitiamo la situazione che si chiama starvation, ovvero l'impossibilità di ottenere risorse da parte di un processo, perché tutti i task sono bloccati.

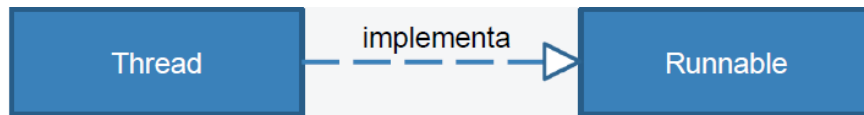
La priority di un thread in JAVA va da 1 a 10, dove la priorità minore è definita `MIN_PRIORITY(1)`, la priorità maggiore è `MAX_PRIORITY(10)` e la priorità normale, che è anche quella di default, è `NORM_PRIORITY(5)`.

# CREARE UN THREAD IN JAVA

Per creare un thread in JAVA abbiamo due possibilità:

- creare una classe che estende la classe Thread
- creare una classe che implementa l'interfaccia Runnable

Ovviamente, la classe Thread implementa l'interfaccia Runnable.



Se creiamo un thread attraverso la classe Thread, la nostra classe deve ereditare attraverso la keyword `extends` la classe Thread. Questo permetterà alla nostra classe di ereditare i metodi della classe Thread, in particolare quello più importante di tutti che è il metodo `run()`. Il metodo `run()` deve essere riscritto nella nostra classe, cioè ci verrà richiesto di fare l'override di tale metodo. Il metodo `run()` contiene le istruzioni che devono essere eseguite dal thread. Il codice presente nel metodo `run()` viene eseguito in maniera concorrente ad altri thread presenti in un programma.

Vediamo un primo esempio di thread in JAVA:

```
public class EsempioThread extends Thread {
    /*Tutto quello che è il blocco di codice che dobbiamo eseguire,
    deve essere messo all'interno del metodo run()
    */
    @Override
    public void run() {
        System.out.println("sono un thread");
    }
}

class MainThread{
    public static void main(String[] args) {
        EsempioThread et = new EsempioThread();

        //Per eseguire il thread, dobbiamo invocare il metodo start()
        et.start(); //sono un thread
    }
}
```

Nell'esempio, l'esecuzione del metodo `start()` nel `main()` ha causato l'invocazione del metodo `run()`.

Quindi, per creare un thread dobbiamo:

- creare un'istanza della nostra classe che estende Thread;
- invocare il metodo `start()` che si occuperà, dopo la configurazione del thread, di invocare il metodo `run()`.

Altri metodi utili che si ereditano dalla classe Thread sono:

- `yield()` che suggerisce allo scheduler di liberare la CPU e renderla disponibile, ovviamente, ad altri thread;
- `sleep()` che mette in standby il thread per un determinato numero di millisecondi.

Possiamo eventualmente creare un thread anche utilizzando direttamente l'interfaccia Runnable. In questo caso la nostra classe deve implementare l'interfaccia Runnable. Questo secondo metodo viene utilizzato, piuttosto che utilizzare la classe Thread, quando la nostra classe estende già un'altra classe e, siccome non possiamo usare l'ereditarietà multipla con l'attributo extends, possiamo utilizzare l'interfaccia Runnable che ci consentirà comunque di implementare un thread. Ovviamente, implementando l'interfaccia, dobbiamo implementare il metodo run(), che è definita all'interno dell'interfaccia Runnable. Per creare un thread in questo caso dobbiamo utilizzare il costruttore della classe Thread, che prende in ingresso come parametro l'istanza della nostra classe che implementa Runnable. Vediamo un esempio:

```
public class EsempioRunnable implements Runnable{
    @Override
    public void run() {
        System.out.println("sono un thread runnable");
    }
}

class MainRunnable {
    public static void main(String[] args) {
        /*Per eseguire questo thread, dobbiamo creare un'istanza della
        classe Thread, passando in ingresso un'istanza della nostra
        classe che implementa l'interfaccia Runnable*/
        Thread t = new Thread(new EsempioRunnable());

        t.start();//sono un thread runnable
    }
}
```

## MULTITHREADING

Multithreading vuol dire eseguire contemporaneamente più thread appartenenti allo stesso processo. Il multithreading può essere:

- collaborativo, ovvero i thread rimangono attivi fino a quando non terminano il task, oppure fino a quando non cedono le risorse occupate ad altri thread;
- preventivo, ovvero la macchina virtuale accede ad un thread attivo e lo controlla attraverso un altro thread.

Le specifiche JAVA stabiliscono che la JVM debba gestire i thread, utilizzando lo scheduling preemptive (o fixed-priority scheduling). Questo vuol dire che lo scheduler ha il compito di interrompere o ripristinare i thread a seconda del loro stato. Quindi, in base in cui si trovano i thread, lo scheduler può attivarli o disattivarli.

Come abbiamo già detto, ogni esecuzione della JVM corrisponde ad un processo, mentre tutto quello che viene eseguito dalla JVM corrisponde ad un thread.

Chiaramente il multithreading ha ragione di esistere perché, se eseguiamo le operazioni in parallelo, ovviamente riusciamo a raggiungere un risultato in maniera più rapida. Oltre a questo, un'altra caratteristica dei software multithreading è che i thread possono scambiarsi informazioni tra loro ed accedere a risorse condivise, ad esempio un database condiviso tra thread, oppure una risorsa hardware.

Vediamo un esempio in codice di multithreading:

```
public class EsempioMultithreading extends Thread{
    @Override
    public void run(){

        System.out.println("Sono il thread " + getName());

        for (int i = 0; i < 10; i++){
            System.out.println(i);
        }
        /*Utilizzo il metodo sleep per dare un tempo di pausa
        tra una stampa di i e un'altra*/
        try {
            Thread.sleep(3000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

class MainMultithreading{
    public static void main(String[] args) {

        EsempioMultithreading em1 = new EsempioMultithreading();
        //setName() serve per poter assegnare un nome al thread
        em1.setName("Thread1");

        EsempioMultithreading em2 = new EsempioMultithreading();
        em2.setName("Thread2");

        EsempioMultithreading em3 = new EsempioMultithreading();
        em3.setName("Thread3");

        EsempioMultithreading em4 = new EsempioMultithreading();
        em4.setName("Thread4");

        EsempioMultithreading em5 = new EsempioMultithreading();
        em5.setName("Thread5");

        em1.start();
        em2.start();
        em3.start();
        em4.start();
        em5.start();

    }
}
```

In questo codice abbiamo 5 thread che cicla la variabile i per 10 volte. Tutti e 5 i thread si eseguono in maniera concorrenziale tra di loro, richiedendo l'accesso alle risorse e, il più delle volte, interrompendosi tra di loro. Ragion per cui, eseguendo più volte questo codice, otterremo sempre un output diverso, perché cambierà sempre l'ordine di esecuzione dei thread e i momenti in cui uno sospenderà l'esecuzione dell'altro.



Vediamo un esempio dei tanti output che possiamo ottenere dal codice precedente:

```
Sono il thread Thread3
0
1
Sono il thread Thread1
0
1
2
3
4
5
6
7
8
9
Sono il thread Thread2
0
1
2
3
4
5
6
7
8
9
Sono il thread Thread4
0
1
2
3
4
5
6
7
8
9
Sono il thread Thread5
0
1
2
3
4
5
6
7
8
9
2
3
4
5
6
7
8
9
```

In questo output in particolare abbiamo:

- il thread 3 che è il primo ad essere eseguito, ma interrotto bruscamente dal thread 1 che ha richiesto l'accesso;
- in seguito, verranno eseguiti in quest'ordine il thread 1, il thread 2, il thread 4 e il thread 5, che non verranno mai interrotti durante la loro esecuzione;
- infine, il thread 3 completa la sua esecuzione dopo il thread 5, interrotta precedentemente dal thread 1.

Alla prossima esecuzione del codice, otterremo un output diverso.

# CONCORRENZA IN JAVA