

Appunti Difroy



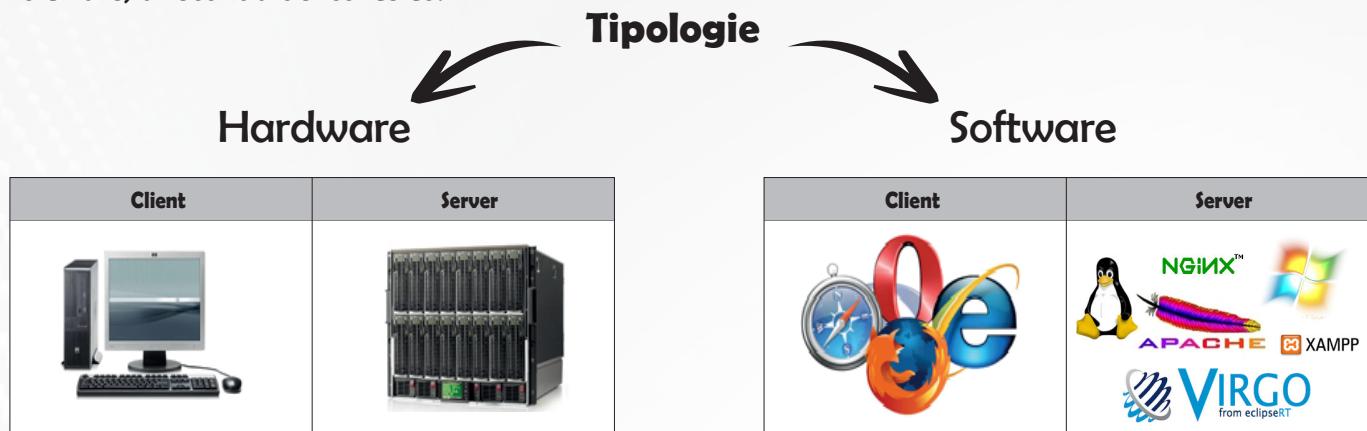
LinkedIn: <https://www.linkedin.com/in/froylan-lamus/>



GitHub: <https://github.com/Difroy>

Server

Un server è un tipo di computer o software system che gestisce le richieste di rete da altri computer, noti come "client". Questi possono richiedere diversi tipi di servizi, come l'accesso a file, condivisione di risorse, gestione di applicazioni o l'accesso a Internet. Il termine "server" si riferisce sia all'hardware che al software, a seconda del contesto.



Server Hardware

I server hardware sono dispositivi fisici progettati specificamente per ospitare, gestire e fornire servizi e risorse a altri computer, noti come client, all'interno di una rete. I server hardware sono ottimizzati per prestazioni, affidabilità, scalabilità e sicurezza, rendendoli fondamentali per l'infrastruttura IT di molte aziende e organizzazioni.

Caratteristiche

Affidabilità: I server sono progettati per funzionare ininterrottamente, 24 ore su 24, 7 giorni su 7. Includono componenti di alta qualità, ridondanza e meccanismi di failover per garantire un'elevata disponibilità.

Scalabilità: Possono essere facilmente scalati per gestire crescenti carichi di lavoro, attraverso l'aggiunta di ulteriori risorse come CPU, memoria e storage.

Sicurezza: Includono funzioni di sicurezza hardware e software, come l'autenticazione avanzata, il controllo degli accessi e la crittografia dei dati.

Gestione del Calore: Sono progettati con sistemi di raffreddamento avanzati per gestire il calore generato dai componenti ad alte prestazioni.

Prestazioni: Utilizzano processori potenti, RAM veloce e storage ad alte prestazioni per gestire compiti intensivi come l'elaborazione di transazioni, l'hosting di applicazioni web e la gestione di database.

Server Software

I server software sono programmi/applicazioni che forniscono servizi a client all'interno di una rete. I server software funzionano su hardware specifico, che può variare da computer personali a server dedicati nei data center.

Caratteristiche

Affidabilità: Sono progettati per essere stabili e disponibili 24/7, con meccanismi di failover e ridondanza.

Scalabilità: Possono essere scalati orizzontalmente o verticalmente per gestire un numero crescente di richieste e utenti.

Sicurezza: Implementano misure di sicurezza per proteggere i dati e garantire l'integrità delle comunicazioni.

Gestione delle Risorse: Ottimizzano l'uso delle risorse hardware come CPU, memoria e storage per garantire prestazioni elevate.

Elaborazione delle Richieste: Gestiscono le richieste dei client, elaborano i dati e rispondono con le informazioni richieste.

Esempi di Server Software

Web Server:

Descrizione: Gestiscono le richieste HTTP da parte dei client (browser) e servono pagine web e contenuti multimediali.

Esempi: Apache/Tomcat, Nginx, Microsoft IIS

Application server:

Descrizione: Forniscono un ambiente per eseguire applicazioni web e enterprise, gestendo la logica di business, le transazioni e la sicurezza.

Esempi: WildFly, GlassFish, Apache Tomcat (relativamente)

Esempi di Server Hardware

Tower Server:

Descrizione: Simile ai PC desktop, ma progettati per essere più robusti e per gestire carichi di lavoro server.

Utilizzo: Piccole aziende o uffici che necessitano di un server dedicato per compiti come la gestione delle stampanti, file sharing, e hosting di applicazioni leggere.

Esempio: Dell PowerEdge T40, HP ProLiant ML350.

Rack Server:

Descrizione: Montati in rack standardizzati, permettono di ottimizzare lo spazio nel data center.

Utilizzo: Medie e grandi aziende con necessità di scalabilità e gestione centralizzata dei server.

Esempio: Dell PowerEdge R740, HPE ProLiant DL380, Lenovo ThinkSystem SR650.

Blade Server:

Descrizione: Moduli server compatti che si inseriscono in un chassis comune che fornisce alimentazione e raffreddamento condivisi.

Utilizzo: Data center che richiedono alta densità di potenza di calcolo in uno spazio ridotto.

Esempio: HPE BladeSystem, Dell EMC PowerEdge MX7000.

Mainframe:

Descrizione: Grandi sistemi di elaborazione utilizzati per applicazioni critiche su larga scala.

Utilizzo: Grandi aziende, banche, istituzioni finanziarie, e agenzie governative che necessitano di elaborazione massiva di dati e transazioni.

Esempio: IBM Z series.

Hyperconverged Infrastructure (HCI):

Descrizione: Combina server, storage e networking in un'unica soluzione integrata, gestita tramite un software centralizzato.

Utilizzo: Data center moderni che cercano di semplificare la gestione e aumentare l'efficienza delle risorse.

Esempio: Nutanix NX Series, Dell EMC VxRail.

Database Server:

Descrizione: Gestiscono la memorizzazione, l'aggiornamento e il recupero dei dati all'interno di database.

Esempi: MySQL/MariaDB, PostgreSQL, Microsoft SQL Server

Mail Server:

Descrizione: Gestiscono l'invio, la ricezione e l'archiviazione di e-mail.

Esempi: Postfix, Sendmail, Microsoft Exchange Server.

File Server:

Descrizione: Gestiscono l'archiviazione e la condivisione dei file all'interno di una rete.

Esempi: Samba, FileZilla Server

Message Server:

Descrizione: Gestiscono la comunicazione asincrona tra applicazioni attraverso messaggi.

Esempi: RabbitMQ, Apache Kafka, ActiveMQ.

Tower Server



Mainstream

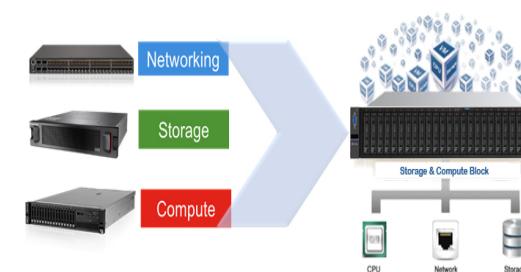


BLADE SERVER

RACK Server



Hyperconverged Infrastructure (HCI)



What Hyperconvergence means

Server Container

Un server container, nel contesto delle applicazioni Java, è un **componente software** che fornisce un ambiente di runtime per eseguire specifici tipi di componenti applicativi, come servlet Java e JavaServer Pages (JSP) e JavaServer Faces (JSF).

Funzioni di un Server Container

Gestione del Ciclo di Vita:

Gestisce la creazione, l'inizializzazione, l'esecuzione, e la distruzione dei componenti applicativi.

Gestione delle Richieste e delle Risposte:

Per i servlet container, riceve le richieste HTTP dal client, le elabora tramite i servlet e JSP, e invia le risposte appropriate.

Gestione delle Sessioni: Mantiene lo stato delle sessioni utente attraverso richieste multiple.

Sicurezza: Implementa misure di sicurezza come l'autenticazione, l'autorizzazione, e la gestione delle connessioni sicure.

Connettività al Database: Fornisce connessioni ai database e gestisce il pooling di connessioni per migliorare le prestazioni.

Servizi di Base: Per i contenitori EJB, offre servizi di gestione delle transazioni, sicurezza, e supporto per la persistenza.

Multithreading: Il container gestisce la creazione e la gestione dei thread per le richieste dei servlet, permettendo l'elaborazione simultanea di multiple richieste, il che aumenta l'efficienza e la scalabilità dell'applicazione.

Gestione delle Risorse: Fornisce servizi come il pooling di connessioni e la gestione delle sessioni, ottimizzando l'uso delle risorse e migliorando le prestazioni delle applicazioni.

Sicurezza: Implementa funzionalità di sicurezza che proteggono le applicazioni dalle minacce comuni come CSRF (Cross-Site Request Forgery) e XSS (Cross-Site Scripting).

Tipi di Server Container

Servlet Container:

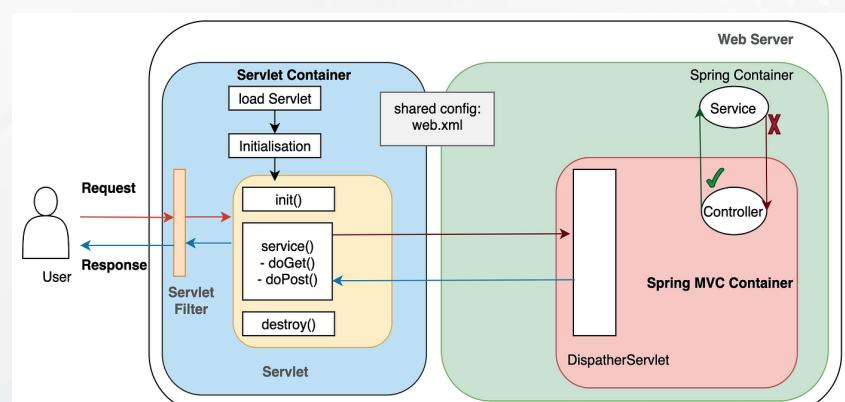
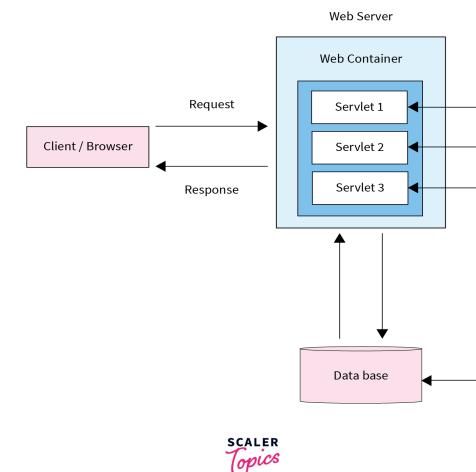
Descrizione: Gestisce le servlet Java e le JSP. Si occupa di ricevere le richieste HTTP dal client, inoltrarle ai servlet appropriati, e restituire le risposte generate al client.

Esempi: Apache Tomcat, Jetty, GlassFish (anche se GlassFish è un application server completo, include un servlet container).

EJB Container (Enterprise JavaBeans Container):

Descrizione: Gestisce i componenti EJB, che sono utilizzati per la logica di business in applicazioni enterprise. Fornisce servizi come la gestione delle transazioni, la sicurezza, il pooling di oggetti, e la persistenza dei dati.

Esempi: WildFly, JBoss EAP, GlassFish.

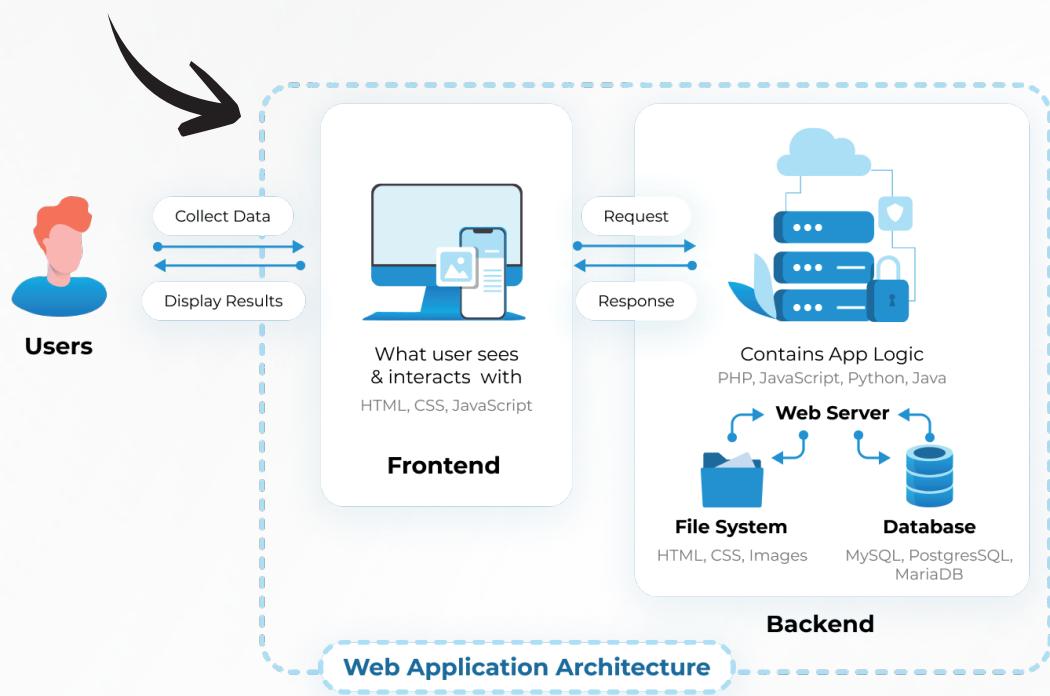


Web Application

Significato

Una web application è un'applicazione software progettata per essere eseguita su un web server e accessibile tramite un browser web. Può essere sviluppata utilizzando una varietà di tecnologie e linguaggi di programmazione come HTML, CSS, JavaScript per il front-end, e linguaggi come PHP, Python, Ruby, Java, o .NET per il back-end..

Architettura



Client/Server Side

Client Side

Definizione: Il client-side si riferisce a tutto ciò che avviene sul dispositivo dell'utente, solitamente all'interno del browser web.

Attività

Rendering della UI:

Generazione e manipolazione del DOM (Document Object Model) per mostrare il contenuto della pagina web.

Aggiornamenti dinamici del contenuto tramite JavaScript e AJAX.

Interattività e Animazioni:

Aggiunta di interattività agli elementi della pagina (e.g., moduli di input, pulsanti).

Creazione di animazioni e transizioni per migliorare l'esperienza utente.

Validazione dei form:

Controllo dei dati inseriti dall'utente nei form prima di inviarli al server.

Utilizzo di JavaScript per assicurarsi che i campi obbligatori siano compilati correttamente.

Gestione degli eventi:

Ascolto e risposta agli eventi utente (e.g., clic, passaggi del mouse, scroll).

Implementazione di callback per gestire le azioni degli utenti.

Richieste AJAX:

Invio di richieste asincrone al server per recuperare o inviare dati senza ricaricare la pagina.

Aggiornamento dinamico del contenuto della pagina basato sulle risposte del server.

Storage Locale:

Utilizzo di LocalStorage e SessionStorage per memorizzare dati lato client.

Gestione di cookie per mantenere lo stato dell'utente.

Gestione del Routing (Single Page Applications):

Utilizzo di framework come React Router per gestire le rotte senza ricaricare la pagina.

Navigazione tra diverse viste in una SPA (Single Page Application).

Linguaggi e tecnologie comuni:

HTML: Struttura il contenuto delle pagine web.

CSS: Stile e design delle pagine web.

JavaScript: Comportamento e interattività delle pagine web.

Frameworks e librerie: React, Angular, Vue.js, ecc.

Pro

Interattività: Consente di creare interfacce utente reattive e dinamiche.

Velocità: Le operazioni sono eseguite sul dispositivo dell'utente, riducendo la latenza.

Riduzione del carico del server: Sposta parte del lavoro computazionale dal server al client.

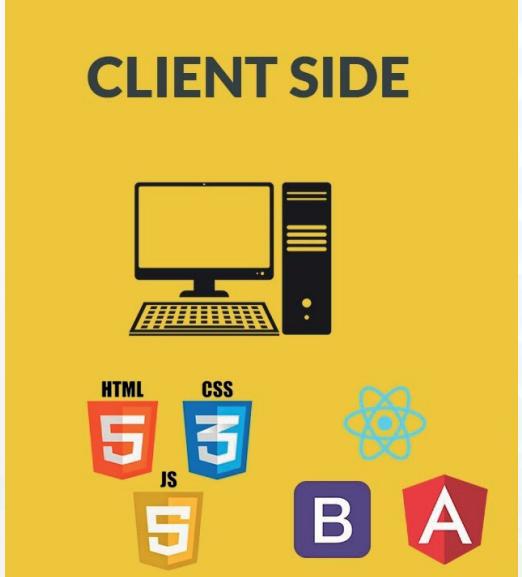
Contro

Sicurezza: Il codice è visibile e accessibile, quindi più vulnerabile a manipolazioni.

Compatibilità: Comportamento variabile a seconda del browser utilizzato dall'utente.

Prestazioni: Dipende dalle risorse del dispositivo dell'utente.

CLIENT SIDE



Client/Server Side

Server Side

Definizione: Il server-side si riferisce a tutto ciò che avviene sul server, che elabora le richieste del client e restituisce le risposte.

Attività

Autentificazione e Autorizzazione:

- Verifica delle credenziali utente e gestione delle sessioni.
- Controllo degli accessi basato sui ruoli utente e sui permessi.

Gestione dei Database:

- Connessione ai database e esecuzione di query per recuperare, inserire, aggiornare o eliminare dati.
- Utilizzo di ORM (Object-Relational Mapping) come Django ORM o Sequelize.

Generazione di Contenuti Dinamici:

- Creazione di pagine web dinamiche basate sui dati recuperati dal database.
- Utilizzo di template engines come EJS, Jinja2, o Handlebars.

Gestione delle Richieste HTTP:

- Ricezione e risposta alle richieste HTTP da parte del client.
- Gestione dei metodi HTTP (GET, POST, PUT, DELETE, ecc.) e dei parametri delle richieste.

Implementazione della Logica di Business:

- Esecuzione della logica di business applicativa, come calcoli complessi o processi di decisione.
- Gestione delle transazioni e delle operazioni aziendali.

Sicurezza del Server:

- Protezione contro attacchi comuni come SQL injection, cross-site scripting (XSS), e cross-site request forgery (CSRF).
- Implementazione di misure di sicurezza come l'uso di HTTPS, firewall, e autenticazione a due fattori.

Integrazione con altri servizi

- Utilizzo di framework come React Router per gestire le rotte senza ricaricare la pagina.
- Navigazione tra diverse viste in una SPA (Single Page Application).

Login e Monitoraggio

- Registrazione delle attività e degli errori del sistema per il monitoraggio e il debugging.
- Utilizzo di strumenti di monitoraggio delle prestazioni e dell'integrità del sistema.

Linguaggi e tecnologie comuni:

Linguaggi di programmazione: PHP, Python, Ruby, Java, Node.js, ecc.

Frameworks: Django, Ruby on Rails, Spring, Express.js, Laravel, ecc.

Database: MySQL, PostgreSQL, MongoDB, ecc.

Pro

Sicurezza: Il codice e i dati sensibili sono mantenuti sul server, lontano dall'accesso diretto degli utenti.

Compatibilità: Il server può gestire le richieste in modo uniforme indipendentemente dal dispositivo dell'utente.

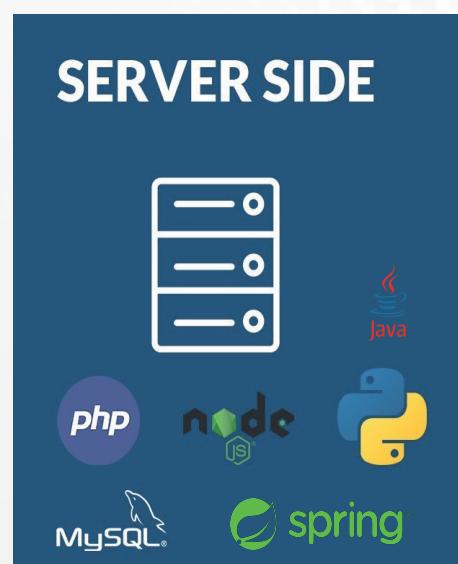
Gestione dei dati: Ideale per operazioni complesse sui dati, come l'accesso e la manipolazione di database.

Contro

Latenza: Ogni richiesta implica una comunicazione con il server, che può introdurre ritardi.

Carico del server: Tutte le operazioni sono eseguite sul server, che può diventare un collo di bottiglia se sovraccaricato.

Scalabilità: La gestione di un grande numero di richieste simultanee può richiedere un'infrastruttura più robusta.





Descrizione

Java è un linguaggio di programmazione ad alto livello e una piattaforma di sviluppo informatico ideata da Sun Microsystems, che è stata successivamente acquisita da Oracle Corporation.

Caratteristiche

Orientato agli Oggetti:

Java è un linguaggio di programmazione orientato agli oggetti (OOP), il che significa che è basato sul concetto di "oggetti" che rappresentano entità del mondo reale con proprietà (campi) e comportamenti (metodi).

Indipendenza dalla Piattaforma:

Uno dei mantra di Java è "Write Once, Run Anywhere" (WORA). Questo è possibile grazie alla Java Virtual Machine (JVM), che permette a un programma Java di essere eseguito su qualsiasi dispositivo che abbia una JVM compatibile, indipendentemente dal sistema operativo sottostante.

Sicurezza:

Java è progettato con caratteristiche di sicurezza avanzate che aiutano a proteggere le applicazioni da minacce come virus e manipolazioni non autorizzate. Questo è particolarmente importante per le applicazioni web.

Robustezza:

Java ha una gestione avanzata degli errori e del garbage collection automatico, che aiutano a creare applicazioni robuste e meno soggette a crash.

Multi-threading:

Java supporta la programmazione multithreading, che permette l'esecuzione simultanea di più thread (piccoli processi) all'interno di un programma. Questo è utile per applicazioni che richiedono un'elevata efficienza e prestazioni.

Alto Livello di Prestazioni:

Sebbene sia un linguaggio interpretato, grazie a ottimizzazioni come il Just-In-Time (JIT) compiler, le prestazioni delle applicazioni Java sono comparabili a quelle dei linguaggi compilati.

Rete e Distribuzione:

Java ha una vasta libreria di API (Application Programming Interface) per sviluppare applicazioni di rete e distribuite, facilitando la costruzione di sistemi scalabili e connessi.

Componenti della Piattaforma Java

JDK (Java Development Kit):

Il JDK è un pacchetto di strumenti necessario per sviluppare applicazioni Java. Include il compilatore (javac), l'interprete/launcher (java), le librerie standard e strumenti di sviluppo come il debugger e il monitor di prestazioni.

JVM (Java Virtual Machine):

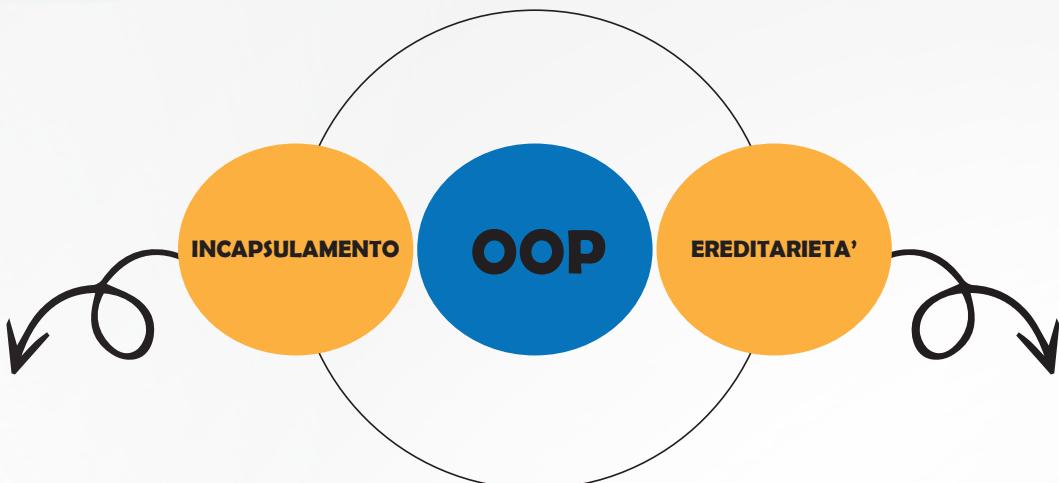
La JVM è un motore di esecuzione che esegue i bytecode Java su qualsiasi piattaforma hardware. La JVM fornisce l'ambiente di runtime necessario per eseguire applicazioni Java.

JRE (Java Runtime Environment):

Il JRE è un subset del JDK che include solo la JVM e le librerie necessarie per eseguire applicazioni Java. Non include strumenti di sviluppo come il compilatore.

		Java Language						Java SE API				
		java	javac	javadoc	jar	javap	JPDA					
JDK	Tools & Tool APIs	JConsole	Java VisualVM	JMC	JFR	Java DB	Int'l	JVM TI				
		IDL	Deploy	Security	Troubleshoot	Scripting	Web Services	RMI				
JRE		Java Web Start						Applet / Java Plug-in				
		JavaFX										
		User Interface Toolkits		Swing	Java 2D	AWT	Accessibility					
		Drag and Drop		Input Methods	Image I/O	Print Service	Sound					
		IDL	JDBC	JNDI	RMI	RMI-IIOP	Scripting					
		Beans	Int'l Support	Input/Output			JMX					
		JNI	Math	Networking			Override Mechanism					
		Security	Serialization	Extension Mechanism			XML JAXP					
		lang and util	Collections	Concurrency Utilities			JAR					
		Logging	Management	Preferences API			Ref Objects					
Java Virtual Machine		Reflection	Regular Expressions	Versioning			Zip	Instrumentation				
		Java HotSpot VM										

I 4 pilastri di Java



Descrizione

L'incapsulamento è il processo di racchiudere dati (variabili) e metodi che operano su quei dati all'interno di una singola unità chiamata classe. Gli attributi di una classe sono privati e accessibili solo tramite metodi pubblici, noti come getter e setter.

Benefici

Protezione dei dati: I dati sensibili possono essere nascosti agli utenti esterni.

Controllo sull'accesso e modifica: I metodi getter e setter possono includere logica per controllare come i dati vengono letti o modificati.

Manutenibilità: Cambiamenti interni alla classe non impattano il codice esterno che usa la classe.

```
public class Person {
    private String name;
    private int age;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        if (age > 0) {
            this.age = age;
        }
    }
}
```

Descrizione

L'ereditarietà è il meccanismo tramite il quale una classe (detta sottoclass o classe derivata) può ereditare campi e metodi da un'altra classe (detta superclasse o classe base). Questo permette la creazione di una gerarchia di classi che condividono un comportamento comune.

Benefici

Riutilizzo del codice: Permette di riutilizzare codice esistente senza doverlo riscrivere.

Gerarchia delle classi: Facilita la creazione di una struttura logica e organizzata del codice.

```
public class Animal {
    public void eat() {
        System.out.println("This animal eats food.");
    }
}

public class Dog extends Animal {
    public void bark() {
        System.out.println("The dog barks.");
    }
}

public class Main {
    public static void main(String[] args) {
        Dog myDog = new Dog();
        myDog.eat(); // Ereditato dalla classe Animal
        myDog.bark(); // Definito nella classe Dog
    }
}
```

I 4 pilastri di Java



Descrizione

Il polimorfismo permette agli oggetti di essere trattati come istanze della loro superclasse piuttosto che della loro classe effettiva. Esistono due tipi principali di polimorfismo: il polimorfismo di compilazione (overloading) e il polimorfismo di runtime (overriding).

Benefici

Flessibilità: Permette di scrivere codice più generale e flessibile.

Estendibilità: Nuove classi possono essere aggiunte con un minimo impatto sul codice esistente.

Descrizione

L'astrazione è il processo di nascondere i dettagli di implementazione e mostrare solo le funzionalità essenziali. In Java, l'astrazione può essere realizzata tramite classi astratte e interfacce.

Benefici

Riduzione della complessità: Facilita la gestione della complessità nascondendo i dettagli non necessari.

Focalizzazione sull'uso: Permette agli sviluppatori di concentrarsi su cosa fa un oggetto piuttosto che su come lo fa.

```
class Animal {
    public void makeSound() {
        System.out.println("This animal makes a sound.");
    }
}

class Dog extends Animal {
    @Override
    public void makeSound() {
        System.out.println("The dog barks.");
    }
}

class Cat extends Animal {
    @Override
    public void makeSound() {
        System.out.println("The cat meows.");
    }
}

public class Main {
    public static void main(String[] args) {
        Animal myDog = new Dog();
        Animal myCat = new Cat();
        myDog.makeSound(); // Output: The dog barks.
        myCat.makeSound(); // Output: The cat meows.
    }
}
```

```
abstract class Animal {
    public abstract void makeSound(); // Metodo astratto

    public void sleep() {
        System.out.println("This animal is sleeping.");
    }
}

class Dog extends Animal {
    @Override
    public void makeSound() {
        System.out.println("The dog barks.");
    }
}

public class Main {
    public static void main(String[] args) {
        Dog myDog = new Dog();
        myDog.makeSound(); // Output: The dog barks.
        myDog.sleep(); // Output: This animal is sleeping.
    }
}
```



Classe

Riduzione della complessità: Facilita la gestione della complessità nascondendo i dettagli non necessari.

Focalizzazione sull'uso: Permette agli sviluppatori di concentrarsi su cosa fa un oggetto piuttosto che su come lo fa.

Una classe è un template o modello che definisce un insieme di variabili e metodi in Java e può essere utilizzato per creare oggetti. Generalmente, una Classe incapsula dati e comportamenti relativi a un'entità o un concetto specifico.

Oggetto

In Java, un oggetto è una istanza di una classe, sono entità concrete che rappresentano esempi specifici di quelle classi.

Caratteristiche

Stato: Definito dalle variabili di istanza (o campi) della classe. Ogni oggetto ha i propri valori per queste variabili.

Comportamento: Definito dai metodi della classe. I metodi operano sui dati all'interno dell'oggetto e possono modificarne lo stato.

Identità: Ogni oggetto ha una identità univoca, anche se due oggetti sono della stessa classe e hanno lo stesso stato.

Class

Persona

Variabili

Nome
Eta

Metodi

mostraDettagli()

Oggetto



Nome: Froy
Eta: 23

```
java
// Definizione della classe
public class Persona {
    // Variabili di istanza
    String nome;
    int eta;

    // Costruttore
    public Persona(String nome, int eta) {
        this.nome = nome;
        this.eta = eta;
    }

    // Metodo per visualizzare i dettagli della persona
    public void mostraDettagli() {
        System.out.println("Nome: " + nome + ", Età: " + eta);
    }
}

// Creazione di un oggetto della classe Persona
public class Main {
    public static void main(String[] args) {
        // Creazione di un nuovo oggetto Persona
        Persona persona1 = new Persona("Mario", 30);

        // Utilizzo del metodo dell'oggetto
        persona1.mostraDettagli(); // Output: Nome: Mario, Età: 30
    }
}
```



Nome: Francesca
Eta: 18



Che cos'è?

Estensione del framework Spring

A cosa serve?

Serve per semplificare lo sviluppo di applicazioni Spring.

Per riuscirci utilizza **configurazioni predefinite** e 'starters'. Con gli starters è possibile:

La semplificazione delle dipendenze.

La configurazione automatica.

L'ottimizzazione della produttività.

Il decremento degli errori.

Differenze



- Configurazione 'Ingombrante'
- Non include il server
- 'Ingombrante' per i microservizi
- Gestione individuale delle dipendenze
- Configurazione manuale per le metriche e monitoraggio
- Flessibile per apps complesse



- Configurazione semplificata
- Server incorporato
- Progettato appositamente per i microservizi.
- Uso di 'Starters' per la gestione delle dipendenze
- Supporto per Actuator
- L'ideale per apps semplici e microservizi

Che cos'è?

Estensione del framework Spring

A cosa serve?

Serve per semplificare lo sviluppo di applicazioni Spring.

Per riuscirci utilizza **configurazioni predefinite** e 'starters'. Con gli starters è possibile:

La semplificazione delle dipendenze.

La configurazione automatica.

L'ottimizzazione della produttività.

Il decremento degli errori.