Cos’è Java?  
  
Java è un linguaggio di programmazione tipizzato ed orientato agli oggetti;  
  
Cosa si intende con tipizzato?  
  
Con tipizzato si intende che una variabile DEVE avere un tipo specifico, rendendo così il codice più robusto e sicuro, in quanto si evitano errori durante l’assegnazione di valori sbagliati e mantiene il codice chiaro in quanto gli errori riguardanti il tipo vengono segnalati direttamente dal compilatore prima dell’esecuzione stessa.  
  
Cosa si intende orientato agli oggetti?

**Questa è la parte centrale, grazie al fatto che Java sia è un linguaggio orientato agli oggetti possiamo strutturare il codice in maniera modulare e riutilizzabile.**  
  
Le caratteristiche base di un linguaggio di programmazione orientato agli oggetti sono:

**Incapsulamento**, ci consente di raggruppare dati e metodi, quindi attributi e comportamenti in una unica entità, definita come: CLASSE, nascondendo dettagli interni di implementazione mostrando solo ciò che è necessario.  
  
**Ereditarietà,** le classi possono ereditare proprietà e comportamenti dalle classi genitore, e non viceversa, consentendo di creare una gerarchia in maniera tale da riutilizzare codice senza doverlo riscrivere, oltre a questo creare una gerarchia ci fa comprendere maggiormente il rapporto che gli oggetti possono avere tra di loro.  
  
**Polimorifsmo,** che consente di usare un oggetto di una classe derivata come se fosse di una classe base.  
**Astrazione:** Mi permette di concentrarmi su cosa un oggetto fa piuttosto di come lo fa.  
***Un esempio di incapsulamento ed astrazione è un DTO***  
  
Cos’è una variabile?  
  
Una variabile è una struttura dati che contiene un valore al suo interno variabile, quindi che può variare, nel tempo a differenza di una costante, è costituita da un **identificatore** (il nome della variabile) ed un **tipo** che definisce la tipologia di dato che un **valore** può avere  
int è il tipo numero è l’identificatore = è l’assegnazione e 10 è il valore.

ITV (identificatore, tipo, valore) DI TIPO PRIMITIVO: int, char, long, double, boolean, ecc

**1) JDK, JRE e JVM  
  
*JDK UTILIZZATA DA NOI   
JRE VA AL DESTINARIO***

* **JDK (Java Development Kit)**: È un pacchetto software che include strumenti per sviluppare , compilare e debuggare applicazioni Java, , tra cui la JVM, il compilatore (javac), e altre utilità di sviluppo.  
  IL COMPILATORE TRASFORMA IL CODICE SORGENTE IN BYTECODE,
* **JRE (Java Runtime Environment)**: È un insieme di software che consente di eseguire applicazioni Java. Include la JVM e ***le librerie necessarie***, ma non contiene strumenti per sviluppare software.
* **JVM (Java Virtual Machine)**: È un motore che esegue le applicazioni Java. Converte il bytecode Java in istruzioni che il sistema operativo può comprendere.

ADESSO CI ALLACCIAMO A HEAP E STACK  
  
In quanto,

**2) Heap and Stack**

* **Heap**: Area di memoria utilizzata per allocare oggetti. La gestore della memoria in heap è più complessa e viene gestita dal garbage collector. Gli oggetti rimangono in memoria finché non sono più referenziati, dopo di ché la GC libererà memoria.
* **Stack**: Area di memoria che gestisce  
  **Variabili locali (chiamata anche Variabile di riferimento) e chiamate di funzione**. Ogni volta che viene chiamato un metodo, viene creata uno stack frame nello stack. La memoria nello stack viene liberata automaticamente al termine del metodo, quindi questo frame viene eliminato.
* Se una variabile locale è un oggetto viene allocata anch’essa nello stack, ma in realtà il suo il riferimento all’oggetto (indirizzo di memoria) che punta all’oggetto vero e proprio nella heap

**3) Passaggio per valore e riferimento**

* **Passaggio per valore**: I valori delle variabili vengono copiati. Modificare una variabile all'interno del metodo non influisce sulla variabile originale.
* **Passaggio per riferimento**: Viene passato un riferimento all'oggetto. Modifiche all'oggetto tramite il riferimento influenzano l'oggetto originale, nel caso delle stringhe no, sono immutabili.

**\*) Esempio di passaggio per valore**

public void modifica(int x) {

x = 10; // non influisce sulla variabile originale

}

**\*) Esempio di passaggio per riferimento**

public void modifica(Prodotto prodotto) {

prodotto.setNome("Nuovo Nome"); // modifica l'oggetto originale

}

**X) Cosa sono gli operatori unari?  
  
Gli operatori unari sono per l’appunto operatori che consente di effettuare operazioni di incremento e decremento.**

**\*) Differenza tra tutti i cicli (for, while, do while, foreach)**

* **for**: Utilizzato quando si conosce il numero di iterazioni.
* **while**: Esegue il blocco finché la condizione è vera. Controlla la condizione prima di eseguire il codice.
* **do while**: Esegue il blocco almeno una volta, poi controlla la condizione.
* **foreach**: Utilizzato per iterare direttamente su una collezione o un array.

**4) Cos'è una variabile (primitiva, reference type e wrapper)**

* **Variabile primitiva**: Contiene dati semplici come int, char, boolean.
* **Reference type**: Contiene un riferimento a un oggetto (e.g., String, ArrayList).
* **Wrapper**: Classi che avvolgono i tipi primitivi in oggetti (e.g., Integer, Double).

**5) Cos'è una classe**

Una classe è un modello o blueprint per creare oggetti. Definisce attributi e metodi.

**6) Cos'è un oggetto**

Un oggetto è un'istanza di una classe. Contiene stati (attributi) e comportamenti (metodi).

**7) Cos'è un metodo**

Un metodo è un blocco di codice che esegue un'operazione e può essere richiamato per riutilizzare il codice.

**\*) Cosa sono i Getters and Setters**

Permettono di accedere a variabili private senza esporle direttamente, contribuendo alla protezione dei dati.

* **Getters**: Metodi per recuperare il valore di un attributo.
* **Setters**: Metodi per impostare il valore di un attributo.

E sono collegati strettamente al concetto di information hiding collegato quindi al pilastro base di Java che è l’incapsulamento.

**\*) Incapsulamento** **(PILASTRO 1/3)**

*un oggetto racchiude tutto il suo funzionamento interno, inclusivo di dati e metodi che su questi agiscono;*

**\*) Cosa sono i modificatori di accesso**

Specificano la visibilità degli attributi e metodi di una classe (public, private, protected, e default).

**\*) Cosa fanno i modificatori di accesso**

Controllano chi può accedere a membri della classe. Ad esempio, private limita l'accesso solo alla classe stessa, protected alla classe stessa, alle classi presenti in quel package, alle sottoclassi (anche in package diversi) mentre il public è il modificatore di accesso più aperto di tutti.

**\*) Cos'è un costruttore e quali conosci**

Un costruttore è un metodo speciale utilizzato per creare e inizializzare oggetti. Esistono costruttori di default e costruttori parametrici con dei parametri formali al suo interno.  
È fondamentale ricordare che una volta definito il nostro costruttore, quello di default non verrà più usato: da adesso in poi, pertanto, l’utilizzo di new Persona() costituirà errore.

**\*) Che cosa vuol dire this nel costruttore**

this si riferisce all'oggetto corrente. È usato per distinguere tra variabili di istanza e parametri del costruttore.

**8) Differenza tra variabile statica e non**

* **Variabile statica**: Appartiene alla classe e non a una specifica istanza. È condivisa tra tutte le istanze.
* **Variabile non statica**: Appartiene a un'istanza specifica della classe.

**9) Differenza tra metodo statico e non (Cos’è un metodo)**

* Un metodo è un blocco di codice all'interno di una classe che definisce un comportamento specifico per gli oggetti creati da quella classe. I metodi possono essere utilizzati per eseguire operazioni, manipolare i dati degli oggetti e restituire valori. Possono avere parametri di input e possono restituire un valore di output *(non void e quindi con return).  
    
  Upcasting è implicito, e quindi abbiamo una conversione automatica.* **Metodo statico**: Può essere chiamato senza creare un'istanza della classe. Non può accedere a membri non statici, usato per applicare regole universali
* **Metodo non statico**: Richiede un'istanza della classe per essere chiamato. Può accedere a membri statici e non statici, usato per applicare regole contestuali

**\*) Cos'è l'overriding**

Overriding è il processo di ridefinire un metodo in una sottoclasse con lo stesso nome e parametri della superclasse. (è una forma di polimorfismo dinamico, poiché permette di usare una variabile di riferimento di tipo genitore, per invocare metodi che verrano eseguti in base all’implementazione del metodo nella classe effettiva dell’oggetto a cui la variabile fa riferimento.

Questo è strettamente correlato con il binding dinamico, è il processo tramite cui la JVM decide a runtime quale metodo invocare, in base al tipo effettivo di oggetto.

**\*) Cos'è l'overloading**

Overloading è il processo di definire più metodi con lo stesso nome ma con parametri diversi nella stessa classe. Serve a dare flessibilità al codice, nel senso, posso avere un costruttore con i requisiti minimi ma anche un costruttore con più requisiti.  
è importante rispettare la firma del metodo quando faccio overloading.

**10) Cos'è l'ereditarietà (PILASTRO 2/3)**

È un meccanismo che consente a una classe (sottoclasse) di ereditare attributi (V.I) e comportamenti (Metodi) da un'altra classe (superclasse).

Una sottoclasse eredita dalla superclasse ma non viceversa.

*Quindi l’ereditarietà prevede che alcune categorie di oggetti non siano sempre progettate da zero ma possano nascere come evoluzione di altre rappresentandone una specializzazione;*

**11) Cos'è il polimorfismo (PILASTRO 3/3)**

*Possiamo definire il polimorfismo come la possibilità che ha un singolo oggetto o un metodo di ottenere risultati diversi dallo stesso meccanismo a seconda dell’oggetto su cui agisce, e quindi ottenere forme diverse a seconda del contesto.*Il polimorfismo può essere applicato sui metodi (Overloading e Overriding)  
e sui dati (Upcasting)   
  
L’upcasting è la possibilità di utilizzare un oggetti mediante un riferimento di classe superiore.

**12) Cos'è il binding dinamico**

Il bind dinamico (o late binding) è un concetto della OOP, in cui la decisione su quale metodo chiamare viene effettuata a runtime, piuttosto che a compile time, e questo è possibile grazie al polimorfismo e all’ereditatietà.

Se ho una variabile di tipo PADRE ma l’oggetto a cui punta è di una CLASSE FIGLIA, Java utilizza il binding dinamico per decidere quale metodo chiamare.

**13) Differenza tra upcast e downcast**

* **Upcast**: Riferire un oggetto di una sottoclasse come se fosse un oggetto della superclasse. Sicuro. Ogni cane è un essere vivente
* **Downcast**: Riferire un oggetto della superclasse come se fosse un oggetto della sottoclasse. Può causare un errore a runtime se non è sicuro. (RunTimeException: ClassCastException) NON OGNI ESSERE VIVENTE E’ UN CANE  
    
  ECCO PERCHE’ NEL PROGETTO HO FATTO QUESTO

**14) Cosa sono le classi astratte**

Le classi astratte in Java sono classi che non possono essere istanziate direttamente e possono contenere sia metodi astratti (senza implementazione) che metodi concreti (con implementazione), ma anche variabili d’istanza, ***definiscono una base parziale di implementazione***.  
  
Sono utilizzate per fornire una base comune per le classi derivate e possono contenere anche stati (variabili di istanza) e comportamenti (metodi). (Una classe può estendere una sola classe astratta per il concetto di ereditarietà singola)

**Caratteristiche delle Classi Astratte:**

* **Non Istanziabili**: Non puoi creare un'istanza di una classe astratta; deve essere estesa da una classe concreta, ISTANZIABILI SOLO CON CLASSI ANONIME, dando un implementazione al volo, qui ed ora, facendo override del metodo e definendo subito.
* **Metodi Astratti e Concreti**: Può contenere metodi astratti (che devono essere implementati dalle classi figlie) e metodi concreti (che possono essere utilizzati direttamente dalle classi figlie).
* **Ereditarietà**: Le classi astratte sono generalmente utilizzate per definire un comportamento comune tra le classi nella stessa gerarchia di ereditarietà.

**15) Cosa sono le interfacce**

Un contratto che definisce un insieme di metodi che una classe deve implementare (UNA COLLEZIONE DI METODI). Non possono contenere implementazione dei metodi, una interfaccia può avere solo metodi astratti e quindi senza corpo. (Una classe può implementare più interfacce).  
Le sue variabili sono implicitamente final e public, e quindi accessibili immediatamente, senza dovere usare super.  
Le interfacce sono utilizzate per definire un comportamento comune che può essere condiviso tra classi diverse, anche se non sono nella stessa gerarchia di ereditarietà.

**16) Si possono istanziare istanze di classi astratte e classi abstract? (classi anonime)**

Non si possono istanziare direttamente, ma è possibile creare classi anonime che estendono classi astratte o implementano interfacce.

**17) Cosa sono i generics e a cosa servono**

I generics consentono di definire classi, interfacce e metodi con parametri di tipo, permettendo di scrivere codice più sicuro e riutilizzabile.

**18) Cosa sono le ArrayList**

Una ArrayList è una collezione dinamica che può contenere oggetti e ridimensionarsi automaticamente.  
l'ArrayList è un esempio di classe generics in Java. Le classi generics permettono di definire una classe con un tipo di dato parametrico, il che significa che puoi specificare il tipo di oggetti che la classe può contenere al momento della creazione dell'istanza.

**\*) Che differenza c'è tra un ArrayList ed un Array**

* **Array**: L’array è un oggetto che contiene elementi indicizzati, ed è fisso in dimensione, contiene solo un tipo di dato. Non ha metodi incorporati per fare operazioni di aggiunta, rimozione di elementi.
* **ArrayList**: Dinamico, può contenere oggetti di qualsiasi tipo e si ridimensiona automaticamente. NON PUO’ CONTENERE PRIMITIVI, e quindi utilizza le wrapper classes. Ha metodi incorporati per la selezione, l’aggiunta e la rimozione di elementi.  
    
  Ah ricorda, potrei scegliere array quando devo essere sicuro del tipo di dato che deve cotenere e quel dato deve essere primitivo, inoltre avere una lunghezza fissa mi da più sicurezza [12] esempio, 12 mesi.

**\*) Che cos'è un Array 2D**

Un array a due dimensioni è un array di array, utile per rappresentare strutture dati come matrici, quindi gli elementi vengono indicizzati in righe e colonne.

**19) Cosa sono le HashMap, HashCode, HashSet**

* **HashMap**: Struttura dati che associa chiavi a valori. Permette un accesso rapido ai dati attraverso l’uso delle chiavi.  
  Ogni chiave in un HashMap deve essere unica, tuttavia, più chiavi possono avere lo stesso valore associato.  
  Un hashmap non mantiene l’ordine degli elementi, nel caso volessimo mantenere l’ordine dovremmo usare una LinkedHashMap o una TreeMap.
* **HashCode**: Invece l’hashCode è un metodo definito nella classe Object, chiaramente overridibile nelle altre classi, che restituisce un valore intero che rappresenta l'oggetto. Utilizzato nelle strutture dati come HashMap. Crea una sorta di FirmaDigitale di quell’oggetto. (Un codice identificato)
* **HashSet**: è una implementazione dell’interfaccia Set in Java, Collezione che non consente duplicati, quindi una collezione di elementi unici e non mantiene l'ordine degli elementi.

**20) Cosa sono le eccezioni**

Le eccezioni sono eventi anomali che si verificano durante l'esecuzione di un programma, interrompendo il flusso normale del programma.  
L’eccezioni in Java derivano dalla classe exception, una sottoclasse di thowable, che a sua volta è una sottoclasse di Object.  
  
Esistono due tipi di eccezioni

1. ***Checked Exc***: Sono quelle che capitano a compile time (quando scrivo il codice), il compilatore quindi obbliga a gestire con un blocco try catch o sia dichiarandole con un throws.
2. ***Unchecked Exc:*** Sono quelle che capitano a runtime (mentre lo eseguo) indicano quindi errori di programmazione.

***Ma il throws?*** Viene inserito nella dichiarazione del metodo, dichiarando che quel metodo può generare delle eccezioni.  
  
La responsabilità di gestire queste eccezioni ricada sul metodo chiamante o su un metodo che ha un blocco try catch in grado di gestire l’eccezione, se non ci fosse un un blocco try catch verrà visualizzato un errore in console con lo stack trace dell’eccezione.   
Ecco quindi spiegato perché esiste il concetto di propagazione delle eccezioni.

**21) Come si usa un'eccezione**

Le eccezioni possono essere gestite utilizzando i blocchi try-catch. In un blocco try si scrive il codice che può generare un'eccezione, mentre nel blocco catch si gestisce l'eccezione.

**22) Che vuol dire throws, throw new, try catch e try with resources**

* **throws**: Dichiarazione di un metodo che può lanciare un'eccezione.
* **throw new**: Utilizzato per generare un'eccezione.
* **try catch**: Struttura per gestire le eccezioni.
* **try with resources**: Permette di dichiarare risorse che devono essere chiuse automaticamente al termine del blocco.

**Quali sono le eccezioni controllate e quelle non controllate?**

Esistono due tipi di eccezioni

1. ***Checked Exc***: Sono quelle che capitano a compile time (quando scrivo il codice), il compilatore quindi obbliga a gestire con un blocco try catch o sia dichiarandole con un throws.
2. ***Unchecked Exc:*** Sono quelle che capitano a runtime (mentre lo eseguo) indicano quindi errori di programmazione.

**23) Gli STREAM I/O**

Gli STREAM di IO sono un'interfaccia fornita dal pacchetto java.io.  
Gli stream di I/O si occupano di leggere e scrivere dati da e verso varie fonti e destinazioni attraverso l’utilizzo di metodi particolari di oggetti e metodi particolari.

**24) Singleton ed MVC**

*Sono dei Pattern architetturali utilizzati nei linguaggi di programmazione*

* **Singleton**: ***PATTERN DI PROGETTAZIONE CREAZIONALE*** che garantisce che una classe abbia una sola istanza e fornisce un punto di accesso globale.
* **MVC (Model-View-Controller)**: ***PATTERN ARCHITETTURALE*** che separa la rappresentazione di un'applicazione dai suoi dati e dalla logica di controllo.

**25) Che cos'è GitHub**

GitHub è una piattaforma di hosting per progetti di sviluppo software che utilizza il sistema di controllo versione Git. Permette la collaborazione tra sviluppatori e la gestione delle versioni del codice