**Domande Esame Programmazione ad oggetti:**

**1) Quali sono le caratteristiche di Java che lo differenziano da altri linguaggi?**

I programmi Java sono compilati in linguaggio bytecode e il bytecode verrà interpretato dalla Java Virtual Machine. Il bytecode può essere compilato in linguaggio macchina staticamente che poi verrà interpretato dalla CPU. Il linguaggio Java utilizza il paradigma orientato agli oggetti impuro (impuro perché esistono i tipi primitivi che non sono oggetti).

**2) Che cosa è una grammatica EBNF, a cosa serve e come viene definita?**

La grammatica Extended Backus-Naur Form (EBNF) è un insieme di produzioni, le produzioni hanno la forma CLASSE ::= ESPANSIONE dove classe è un simbolo non terminale, espansione è una sequenza di simboli terminali e non. Il simbolo| indica una alternativa, il ? elemento opzionale, + elemento ripetuto, \* elemento opzionale ripetuto, () parentesi di raggruppamento. Viene utilizzata per rappresentare la sintassi del linguaggio che la sfrutta.

**3) Quali sono i tipi primitivi del linguaggio Java? Le stringhe sono primitive?**

**Gli array sono primitivi? Qual è la differenza fra tipi primitivi e gli altri tipi?**

I tipi primitivi comprendono: byte, short, int, long, float, double, char, boolean. Gli array sono tipo riferimento e non un tipo primitivo come le String ma le String possono essere dichiarati senza sfruttare l’operatore New.

I tipi primitivi vengono semplicemente dichiarati e contengono un valore della variabile che rappresentano, il valore deve essere pertinente; i tipi primitivi possono fare confronti per identità. I tipi riferimento devono essere dichiarati invocando l’operatore new se si crea un nuovo oggetto mentre se si vuole avere un riferimento devono riferirsi ad un oggetto pertinente alla classe; i confronti devono sfruttare metodi specifici.

**4) Cosa sono i tipi enum? Come si usano? Sono tipi primitivi o riferimento?**

I tipi enum sono una sotto classe di enum e rappresentano un insieme finito di valori che vengono dichiarati alla creazione del enum. La variabile di tipo enum deve contenere un valore corrispondente ad un valore dichiarato. Risultano molto comodi con la struttura switch. I tipi enum sono tipi riferimento.

Esempio:

public enum Seme {CUORI, QUADRI, FIORI, PICCHE;}

Seme carta = Seme.CUORI;

la variabile carta avrà il riferimento a CUORI.

**5) Conversioni di tipo implicite ed esplicite, differenza ed effetti in caso di tipi primitivi e involucro.**

La conversione da un tipo ad un altro è implicita se non produce una perdita di informazioni (esempio negli interi passando da un tipo con minor precisione ad uno con maggior precisione), una conversione esplicita permette di “forzare” il compilatore ad interpretare il risultato di una espressione nel tipo di dati in cui si converte. Nei tipo primitivi si può convertire direttamente seguendo questa direzione: byte->short->int->long->float->double e i char possono essere convertiti in int. Nei tipi involucro dobbiamo prima un-boxarli per ottenere il tipo primitivo, seguire la conversione fino al punto desiderato e poi boxarli nuovamente per l’involucro corrispondente.

**6) Valutazione di un'espressione ed esecuzione di un programma, differenza e**

**possibili esiti. Cosa sono gli effetti collaterali?**

La valutazione di una espressione può non terminare, produrre un errore (lanciare una eccezione), produrre dei side effect o terminare regolarmente producendo un valore o un riferimento a una variabile o non produrre risultati. I side effect sono delle variazioni che si manifestano essendo che si è valutata l’espressione come la variazione del valore di una variabile, il riferimento di una variabile di tipo riferimento ecc...

**7) Differenza fra una classe ed un oggetto. Cosa è la metaclasse di una classe?**

Gli oggetti sono istanze della classe. La classe definisce lo schema di come saranno fatti tutti gli oggetti ma ogni oggetto potrà avere valori diversi nei campi del proprio schema. La metaclasse di una classe è la classe di cui è una istanza la classe, quindi sarebbe la sua superclasse. Per esempio ogni classe ha come metaclasse Class.

In Java there's a single metaclass: the instances of the class Class are used to represent the types of classes and interfaces.

There is only one metaclass: java.lang.Class. Multiple instances (meta-instance?) of the metaclass describe individual classes - including the metaclass itself.

**8) Differenza fra variabili/metodi statici e di istanza. Come vengono usati?**

Le variabili statiche sono variabili contenute nella classe e non nelle istanze, le variabili di istanza vengono contenute negli oggetti. Si utilizzano per esempio per definire nel primo caso una caratteristica uguale a tutte le istanze mentre nel secondo caso caratteristiche diverse per ogni singola istanza. I metodi statici sono metodi che sono mantenuti e utilizzabili solo all’interno della classe mentre i metodi di istanza sono contenuti e utilizzabili all’interno dell’oggetto.

**9) Cosa è un costruttore? Come avviene la creazione di un oggetto e quando viene**

**eseguito il costruttore.**

Un costruttore è un metodo speciale che deve avere lo stesso identificatore della classe, permette di creare un oggetto di quella classe e restituire il suo riferimento. La creazione del oggetto avviene utilizzando il metodo new() che abbia come argomento la classe a cui appartiene l’oggetto che vogliamo costruire, si determina la classe e si verifica che il codice sia presente in memoria se no lo si carica. Si alloca lo spazio necessario dallo heap per contenere lo schema, i campi vengono inizializzati al valore di default e poi eseguite le eventuali espressioni presenti nella inizializzazione e i risultati assegnati ai rispettivi campi. Infine si cerca il costruttore giusto (corrispondente ai parametri forniti al new), si invoca il costruttore che tipicamente inizializza i campi ed eventualmente esegue ulteriori istruzioni alla fine si restituisce il riferimento a questo oggetto

**10) Come funziona la distruzione di oggetti in Java? In che modo e fino a che punto è possibile controllarla?**

Quando un oggetto non ha più nessuna variabile che lo referenzia allora finisce nel garbage collector che lo andrà ad eliminare definitivamente invocando il metodo finalize() se è presente. Nel corpo di questo metodo possono essere inserite ulteriori istruzione che rappresentano le “ultime volontà” in ogni caso all’uscita del metodo l’oggetto viene distrutto e lo spazio liberato viene restituito allo heap

**11) Come funzionano e cosa contengono ambienti, record di attivazione e scope?**

**Come si evolvono durante un'esecuzione single-thread?**

**Cosa cambia in caso di esecuzione multi-thread?**

Gli ambienti contengono i record di attivazione. Un record di attivazione viene creato quando si invoca un metodo e contiene le variabili locali, i parametri formali, il nuovo record viene messo in pila a quello precedente.

Lo scope di una variabile locale si estende dalla dichiarazione alla fine del record di attivazione che la contiene. In un esecuzione single-thread esiste un unico ambiente dove i record di attivazione crescono e decrescono al suo interno seguendo lo stack LIFO mentre in una esecuzione multi-thread esistono più ambienti ma non si risolvono con la stack LIFO ma dipende dalla loro esecuzione inoltre ogni ambiente potrebbe avere visibilità agli oggetti contenuti nello heap se possiede il riferimento specifico.

**12) Cosa sono i metodi sovraccarichi? Come vengono selezionati? In quali situazioni possono produrre errori di compilazione?**

Sono metodi con lo stesso nome ma con firme diverse cioè permettono di accettare parametri diversi (sia nel numero che nel tipo) e di comportarsi diversamente in funzione di essi. Vengono selezionati se si trova la firma corrispondente o 1) per sottotipi, 2) eseguendo il boxing o l’unboxing, 3 argomenti variadici (gli ultimi elementi si tende ad inserirli in un argomento variadico). Si deve rispettare l’ordine presentato e si sceglie la conversione che richiede meno conversioni fra i tipi ma se ci dovessero essere più conversioni minime allora si causa un errore di compilazione perché risulterebbe ambiguo.

**13) Quali sono i quattro strumenti concettuali? Che relazione hanno tra di loro?**

**A quali costrutti Java corrispondono?**

I quattro strumenti sono: L’ereditarietà che sfrutta il concetto delle classi e sottoclassi, una sottoclasse eredita tutti i campi e tutti i metodi della classe da cui discende, tranne i costruttori. L’astrazione permette di memorizzare solo le informazioni rilevanti, permette l’uso del livello corretto nella gerarchia (esempio una automobile ha 4 ruote mentre un veicolo no ma avrà un modo per muoversi). L’incapsulamento protegge lo stato dei campi e protegge l’utilizzo di determinati metodi viene gestito con la visibilità. Il polimorfismo può essere per ereditarietà (una oggetto di una classe può essere considerato come oggetto di una sua superclasse)o parametrico (una classe generica che definisce metodi generici) sfrutta l’upcast, downcast, overriding e overloading.

**14) Quali sono i modificatori di visibilità per componenti di una classe?**

**Per ciascuno di essi, dove rende visibile un suo componente?**

I modificatori di visibilità sono: public, protected, default e private

Se abbiamo un componente X di una classe C nel package P allora il componente X è visibile: 1) se private solo dentro C, 2) se default solo dentro P, 3) se protected anche dentro a classi derivate da C fuori da P, 4) se public ovunque dove sia visibile C

**15) Quali tipi di classi non top-level si possono avere? Quando vengono solitamente usati?**

Le classi non top-level sono: membro(esempio un iteratore è un oggetto di una classe interna), locali (la classe è visibile solo dentro il metodo che la definisce) o anonime (classe che viene definita dentro a new() solitamente usata per i pulsanti)

**16) Cosa sono classi, costruttori e metodi generici? Come si dichiarano e come si usano? Parametri di tipo semplici e vincolati, e uso dei wildcard.**

Classi, costruttori e metodi generici si riconoscono tutti dalla presenza di Parametri Formali di Tipo nella loro intestazione, ad esempio public class Vector <E>, dove E è un parametro formale di tipo. Questa generalizzazione permette a questi costrutti, generici per l’appunto, di poter essere utilizzati con diversi tipi di parametri.

Nella dichiarazione di classi generiche i parametri di tipo vanno posti appena dopo il nome della classe stessa e tra <>(public class Box <E>{…corpo…}).

Nella dichiarazione di un costruttore generico i PFT vanno posti appena prima del nome del costruttore stesso. Un costruttore può essere generico anche se la sua classe non lo è (public <T> Box(E elemento, T arg){…corpo costruttore…}); se nella sua invocazione non viene dichiarato il tipo T quest’ultimo viene inferito dal sistema.

Nella dichiarazione di metodi generici i PFT vengono anteposti al nome del metodo e al suo tipo di ritorno (public static <T> void print(T[] a) {…corpo…}). Anche i metodi possono essere generici indipendentemente dalla loro classe. Nella loro invocazione il tipo potrebbe essere inferito dal sistema se non viene fornito, ma risulterebbe molto meno espressivo (int r = Max.max(1,2,3); contro int r = Max.<Integer>max(1,2,3);).

I parametri di tipo di cui abbiamo parlato finora possono essere semplici, vincolati, o con wildcard. Nelle intestazioni dei costrutti sopra citati i parametri di tipo sono semplici se non sono ulteriormente vincolati, mentre sono vincolati (con vincolo di supertipo) se sono vincolati ad estendere una (o più) particolare classe o interfaccia. Il vincolo viene posto usando la parola chiave extends, indipendentemente che sia riferito ad una classe o ad una interfaccia. In caso di vincoli multipli si usa extends e & per separare le varie classi o interfacce.

Ogni tipo generico ammette un supertipo comune a tutte le sue istanze parametrizzate; questo supertipo si indica scrivendo il carattere ? (la wildcard) al posto dei parametri di tipo.

public static void stampa1(ArrayList<Persona> v){…}

public static <E extends Persona> void stampa2(ArrayList<E> v) {…}

public static void stampa3(ArrayList<? extends Persona> v) {…}

**17) Come viene determinata la compatibilità dei tipi riferimento in Java?**

**Come è influenzata questa regola dalla struttura delle istanze?**

Attraverso l’operatore instanceof (espr instanceof Type); se il tipo dinamico di espr è Type o sottoclasse di Type allora restituisce true; se così non fosse o espr fosse null restituirebbe false. La struttura delle istanze ci assicura che il metodo dinamico di un oggetto sia sempre compatibile con il suo tipo statico.

Quindi se Type o una sua sottoclasse fosse il tipo statico di espr, allora instanceof restituirebbe (quasi) sempre true.

**18) Differenza tra tipo statico e dinamico di un'espressione. Come viene modificato da un cast? In che modo questi tipi definiscono la risoluzione delle chiamate?**

Il tipo statico di una espressione è il tipo che è dichiarato mentre il tipo dinamico è il tipo che viene realmente restituito in una situazione precisa. Per esempio: punto getCoordinate (){return this.Punto()} tipo statico è punto ma tipo dinamico potrebbe essere puntoColorato. Il cast permette di dichiarare esplicitamente come deve essere interpretato il risultato invece che essere considerato come dichiarato nel tipo statico. Se il tipo statico di una espressione è sottoclasse del tipo in cui è fatto il cast allora non ci sono errori di compilazione se a tempo di esecuzione però il tipo dinamico non è compatibile con il tipo castato allora si solleva una eccezione di tipo ClassCastException.

**19) Che cosa è l'override di metodi? Differenza tra overriding e hiding.**

**Se la classe A estende la classe B, quali sono le variabili e i metodi di B?**

L’ovveride è una procedura che permette di ridefinire un metodo di una superclasse in una sua sottoclasse con firma identica. L’Hiding permette di nascondere in una sottoclasse i metodi statici che sono stati definiti in una superclasse. B avrà solo le variabili definite in B e i metodi definiti in B ed eventualmente tutti quelli che potrebbe eventualmente ereditare da una ulteriore classe (ma non quelli di A).

**20) Che significato hanno le metavariabili this e super in un metodo?**

**Che significato hanno invece this(...) e super(...) e dove vengono usati?**

**Cosa succede se quando né this(...) né super(...) sono presenti?**

Nei metodi super permette di invocare un metodo di una superclasse che è stato sovrascritto dalla classe che non sarebbe più possibile accederci (invocando il metodo definito nella superclasse), this permette di invocare un metodo della istanza corrente. this(…)e super(…) vengono usati nei costruttori e il primo permette di richiamare un altro costruttore definito nella classe mentre il secondo richiama un costruttore definito in una superclasse.

Se nessuno dei due è presente allora viene richiamato come primo comando super().

**21) Cosa sono classi e metodi astratti? Qual è la differenza fra classe astratta e interfaccia?**

Una classe è astratta (incompleta) se è dichiarata con il modificatore abstract e contiene metodi astratti, un metodo astratto è un metodo a cui manca il corpo ed è sostituito con “;” il metodo è qualificato abstract. Questi metodi dovranno essere definiti nelle sottoclassi che non avranno più il modificatore abstract.

Una classe può discendere solo da una superclasse e non da 2 diverse se ci fosse la necessità di ereditare 2 o più caratteristiche si può implementare una o più interfaccia le interfacci possono avere solo metodi astratti.

**22) Uso, significato e conseguenze dell'interfaccia Iterable.**

**Quali interfacce possono essere definite "contratti"?**

Una classe che implementa Iterable consente di applicare il ciclo for-each alle istanze della sua classe. L’interfaccia comprende solo il metodo iterator() che deve restituire un oggetto di tipo Iterator<T> il corpo del metodo dovrà essere definito nella classe. Le interfacce che possono essere definite contratti sono:

iterator, LinkedList, Comparable, Serializable, Cloneable

**23) Cosa è un’eccezione? Come si dichiara? Come si solleva?**

Un eccezione è un oggetto che viene creato ogni volta che si verifica un problema, la classe ne descrive la natura del problema. Si dichiara una eccezione lanciando il comando throw e creando una istanza della eccezione. Solitamente la sequenza viene eseguita dopo che si è effettuato un controllo per verificare se sia necessario sollevare l’eccezione.

**24) A cosa serve il costrutto "try/catch"? Cosa succede in sua assenza?**

**Quando e come va usata la clausola "throws"?**

Il costrutto serve per catturare le eccezioni che potrebbero essere sollevate nella esecuzione del corpo del costrutto. Se non è presente e si verifica una eccezione, questa provocherà una uscita anomala dal programma visto che l’eccezione verrà propagata in tutta la pila degli ambienti. La clausola throws informa il compilatore che l’esecuzione del metodo può causare una eccezione controllata, deve essere inserita nella dichiarazione e subito prima del corpo

**25) Come si avvia un nuovo thread in Java? Condivisione della memoria e accesso sincronizzato. Quale altro sistema si usa per sincronizzare computazioni più complesse?**

Un thread viene creato con una espressione new ma viene avviato solo se viene invocato il metodo start() su di esso. Tutti i thread condividono lo stesso heap ma ogni thread può accedere solo agli oggetti di cui possiede un riferimento. Per gestire gli accessi in modo sincronizzato si possono utilizzare dei monitor e per poter accedere ad un certo oggetto o metodo si deve prima acquisire il monitor, se il monitor è libero allora si accede al monitor se invece è già stato acquisito allora il thread viene bloccato e verrà fato ripartire quando il monitor sarà libero. Per sincronizzare computazioni più complesse si possono utilizzare i metodi wait(), notify() e notifyAll() ma questi metodi possono essere eseguiti solo dopo aver acquisito il monitor.