HW#1

1 – Why we need to use OOP ? Some major OOP languages ?

Object oriented Programming (OOP) is the best innovation in software development. OOP concepts address various software crisis.

What is software crisis?

The software development has become very complex to build complex projects. The term software crisis defines exceeding budget of project development, end project not meeting client's requirements and bugs in programs. OOPs programming paradigms are used to rectify all these issues.

What exactly is object?

Now go back to our 8th grade where in physics we have studied about objects as they are something that have mass and occupy space. Object is a real-life entity. Pen, paper, books, desk,laptops everything is object. In programming languages, we exactly want to build programs to represent these real life objects hence we are currently using OOP languages. Object is also a code that has some space in memory and hence it is concrete. Objects have attributes,identity and behavior. Now consider a dog which is again an object. It has color,shape (attributes) , it can bark,wag tile, bite (behavior) and it has breed (identity). Similarly what ever the object we code also has these properties. Attributes define data of object, behavior defined response of object when subjected to simulation. We can say behavior as synonym of function or method.

Can't we do the same using Procedural programming Languages?

Basic procedural languages like C,FORTRAN,BASIC have list of instructions as programs. As length of program increases, complexity increases and whatever the functions we create using such programming paradigms, have unrestricted access meaning anyone can access the code. It is also difficult to create code that represents real-life entities. These languages generally have less extensibility i.e., they have built in data types and it is somewhat difficult to create user-defined data types. So, we move to OOP languages to create complex softwares.

There are certain principles of OOP :

1. **Abstraction :** It hides all the implementation details of the software so that end user can only see what he's intended to see. For example, we use facebook, whatsapp but we only interact with the interface not with the code that is used to develop applications otherwise anyone could easily manipulate the application.
2. **Inheritance:** It enables re-usability. generally, inheritance is something we get from parents/ancestors. Similarly we have parent classes,sub-classes in OOP languages. In case the code written in one class needs to be used again in other class, this concept enables us to re-use the code rather than writing it again in other class.
3. **Encapsulation:** It means hiding data or code into single module,that module can be any class or method. It helps in protecting the code from others to access it. Think encapsulation as a capsule which hides all the drug inside it in one module or capsule.
4. **Polymorphism:** It means "many forms". It can be defined as same thing used many ways. U can think it of as an object exhibiting different behaviors.

2 – Interface vs Abstract class ?

Abstract Class	Interface
An abstract class in Java includes abstract methods and traditional methods.	An interface in Java strictly involves only abstract methods.
The abstract class executes at a faster speed.	An interface is relatively slower than the abstract class.
It is used only for implementing the concept of abstraction and not the multiple inheritance.	You can use an interface in Java for both abstraction and multiple inheritance.
You can use static, non-static, and non-final variables in the abstract class.	You can use the final, public, and static variables in the interface.
The abstract class is used to avoid independence.	The interface can be used for future enrichment.
A single class can extend only one abstract class.	A single class can implement several interfaces.
The declaration of the abstract class uses the keyword 'abstract.'	The interface in Java can be implemented using the keyword 'interface.'
You can use access modifiers in the abstract class.	An interface does not support the use of different access modifiers. Everything defined in the interface is public.
The keyword 'extends' is used to extend the abstract class.	The Keyword 'implements' is used to implement the interface.
An abstract is flexible, which can extend another class and can implement several interfaces.	An interface in Java can extend only other interfaces.
You can declare constructors and destructors in the abstract class.	Using constructors and destructors is not allowed in the Java interface.
In the abstract class, you can define fields and constants.	In the interface, you can't define any field.

3 – Why wee need equals and hashcode ? When to override ?

The composite key class represents the primary key of an entity, and therefore, it should be unique. Hibernate uses equals and hashcode methods internally to identify the uniqueness, compare objects, and check for duplications. To make these operations in Java, you have to implement equals and hashcode methods.

Do it whenever you need to override the default notion of equality: that two objects are equal only if they are the same object.

In other words, when two different instances can, in some sense, be "equal".

For example, Integer overrides equals because new Integer(2) != new Integer(2), but you'd expect new Integer(2).equals(new Integer(2)). Intuitively, an object representing 2 should be equal to another object representing 2.

You need to override hashCode at the same time, in order that your value will work consistently with the equal implementation in hash-based data structures.

4 – Diamon problem in Java ? How to fix it?

For instance, let us assume that Java does support multiple inheritance. And if we have an abstract class named *Sample* with an abstract method ***demo().*** Then if two other classes in the same package extends this class and try to implement its abstract method, *demo()*. Then, as per the basic rule of inheritance, a copy of both demo() methods should be created in the subclass object which leaves the subclass with two methods with same prototype (name and arguments). Then, if you call the demo() method using the object of the subclass compiler faces an ambiguous situation not knowing which method to call. This issue is known as diamond problem in Java.

How to fix it?

You can achieve multiple inheritance in Java, using the default methods (Java8) and interfaces.

From Java8 on wards **default methods** are introduced in an interface. Unlike other abstract methods these are the methods of an interface with a default implementation. If you have default method in an interface, it is not mandatory to override (provide body) it in the classes that are already implementing this interface.

You can have same default methods (same name and signature) in two different interfaces and, from a class you can implement these two interfaces.

If you do so, you must override the default method from the class explicitly specifying the default method along with its interface name.

5 – Why we need Garbagge Collector ? How does it run ?

Java applications obtain objects in memory as needed. It is the task of garbage collection (GC) in the Java virtual machine (JVM) to automatically determine what memory is no longer being used by a Java application and to recycle this memory for other uses. Because memory is automatically reclaimed in the JVM, Java application developers are not burdened with having to explicitly free memory objects that are not being used.

There are 2 ways to run the garbage collector in java.

- You can use the Runtime.getRuntime().gc() method- This class allows the program to interface with the Java Virtual machine. The “gc()” method allows us to call the garbage collector method.
- You can also use the System.gc() method which is common.

The application is simple. There are no syntaxes for these individual methods in Java.

6 – Java ‘static’ keyword usage ?

The static keyword in Java is mainly used for memory management. The static keyword in Java is used to share the same variable or method of a given class. The users can apply static keywords with variables, methods, blocks, and nested classes. The static keyword belongs to the class than an instance of the class.

7 – Immutability means ? Where, How and Why to use it ?

An immutable object is an object whose internal state remains constant after it has been entirely created.

This means that the public API of an immutable object guarantees us that it will behave in the same way during its whole lifetime.

To create an immutable class in Java, you have to do the following steps:

- Declare the class as final so it can't be extended.
- Make all fields private so that direct access is not allowed.
- Don't provide setter methods for variables.
- Make all mutable fields final so that its value can be assigned only once.
- Initialize all the fields via a constructor performing deep copy.
- Perform cloning of objects in the getter methods to return a copy rather than returning the actual object reference.

8 – Composition and Aggregation means and differences ?

An object is a real-time entry, and objects have relationships between them both in programming or real life. Objects are related to each other using more than one relationship, such as Aggregation, Composition, Association, etc.

Aggregation	**Composition**
Association between two objects that defines the "has-a" relationship is referred to as Aggregation.	A specific type of Aggregation that implies ownership is referred to as Composition.
In Aggregation, objects can remain in the scope of a system without each other.	In a composition relationship, objects cannot remain in the scope of a system without each other.
In Aggregation, linked objects are independent of each other.	In Composition, objects are tightly coupled or dependent on each other.
In Aggregation, if we delete the parent object, the child object will still exist in the system. For example, if we remove wheels from a car, the car can function adequately with another wheel.	In Composition, if we delete a parent object, the child object also gets deleted. For example, if we delete a folder, the files will also get deleted which contain in the folder.

We represent it by using the filled diamond.	We represent it by using the empty diamond.
In Aggregation, child objects don't have a lifetime.	In Composition, child objects have a lifetime.
In Aggregation, if we delete an assembly, it will never affect its parts.	In the case of owning a class, it affects the containing class object.

9 – Cohesion and Coupling means and differences ?

Coupling and cohesion are two concepts found in Java (and all other object oriented languages). Coupling measures how much each of the program modules are dependent on the other program modules. Cohesion measures how strongly each of the functions are related within a module. Actually, any object oriented language (including Java) has the two main objectives of increasing cohesiveness and decreasing the coupling at the same time, in order to develop most efficient programmes. These two software engineering metrics were developed by Larry Constantine to reduce the cost of modifying and maintaining software.

What is Cohesion?

Cohesion measures how strongly each of the functions are related within a program module. Well structured classes lead to highly cohesive programs. If a certain class is performing a set of highly related functions, that class is said to be cohesive. On the other hand, if a class is performing a bunch of totally unrelated functionalities that means the class is not cohesive at all. It is important to understand that not having cohesiveness does not mean that the overall application does not have the required functionality. It's just that without cohesion, it will be very difficult to mange the functionality because they will be scattered in many wrong places as the complexity of the application increases over time. Maintaining, modifying and extending behaviors scattered all over the code is very tedious even for the most experience programmers.

What is Coupling?

Coupling measures how much each of the program modules are dependent on the other program modules. Interactions between two objects occur because there is coupling. Loosely-coupled programs are high in flexibility and extensibility. Strong coupling is never good because one object can be highly dependent on some other object. This is a nightmare when the code is modified, because high coupling means that the programmers need to work on several places of code for even a single behavioral modification. Strong coupling always leads to programs with low flexibility and less scalability/extensibility. However, in programming languages like Java, completely avoiding coupling is impossible. But it is recommended that the programmers give their best effort to reduce the coupling as much as possible. It is also possible to have some coupling to help objects interact with each other without hampering its scalability and flexibility.

What is the difference between Coupling and Cohesion?

Even though coupling and cohesion deal with the quality of a module in software engineering, they are entirely different concepts. Cohesion talks about how much the functionality are related to each other within the module, while coupling deals with how much one module is dependent on the other program modules within the whole application. In order to have the best quality software, cohesion and coupling

should reach the two opposite ends of their spectrums. In other words, loose coupling and strong cohesion provides the best software. Having private fields, non-public classes and private methods provide loose-coupling, while making all members visible within the class and having package as the default visibility provide high cohesion.

10 - Heap and Stack means and differences ?

Java Heap Space

Java Heap space is used by java runtime to allocate memory to Objects and JRE classes. Whenever we create an object, it's always created in the Heap space.

Garbage Collection runs on the heap memory to free the memory used by objects that don't have any reference. Any object created in the heap space has global access and can be referenced from anywhere of the application.

Java Stack Memory

Java Stack memory is used for the execution of a thread. They contain method-specific values that are short-lived and references to other objects in the heap that is getting referred from the method.

Stack memory is always referenced in LIFO (Last-In-First-Out) order. Whenever a method is invoked, a new block is created in the stack memory for the method to hold local primitive values and reference to other objects in the method.

As soon as the method ends, the block becomes unused and becomes available for the next method. Stack memory size is very less compared to Heap memory.

Difference between Java Heap Space and Stack Memory

Based on the above explanations, we can easily conclude the following differences between Heap and Stack memory.

- Heap memory is used by all the parts of the application whereas stack memory is used only by one thread of execution.

- Whenever an object is created, it's always stored in the Heap space and stack memory contains the reference to it. Stack memory only contains local primitive variables and reference variables to objects in heap space.

- Objects stored in the heap are globally accessible whereas stack memory can't be accessed by other threads.

- Memory management in stack is done in LIFO manner whereas it's more complex in Heap memory because it's used globally. Heap memory is divided into Young-Generation, Old-Generation etc, more details at Java Garbage Collection.

- Stack memory is short-lived whereas heap memory lives from the start till the end of application execution.

- We can use -Xms and -Xmx JVM option to define the startup size and maximum size of heap memory. We can use -Xss to define the stack memory size.

- When stack memory is full, Java runtime throws java.lang.StackOverFlowError whereas if heap memory is full, it throws java.lang.OutOfMemoryError: Java Heap Space error.

- Stack memory size is very less when compared to Heap memory. Because of simplicity in memory allocation (LIFO), stack memory is very fast when compared to heap memory.

11 – Exception means ? Type of Exceptions ?

When a program violates the semantic constraints of the Java programming language, the Java Virtual Machine signals this error to the program as an *exception*.

An example of such a violation is an attempt to index outside the bounds of an array. Some programming languages and their implementations react to such errors by peremptorily terminating the program; other programming languages allow an implementation to react in an arbitrary or unpredictable way. Neither of these approaches is compatible with the design goals of the Java SE Platform: to provide portability and robustness.

Instead, the Java programming language specifies that an exception will be thrown when semantic constraints are violated and will cause a non-local transfer of control from the point where the exception occurred to a point that can be specified by the programmer.

An exception is said to be *thrown* from the point where it occurred and is said to be *caught* at the point to which control is transferred.

Programs can also throw exceptions explicitly, using throw statements.

Explicit use of throw statements provides an alternative to the old-fashioned style of handling error conditions by returning funny values, such as the integer value -1 where a negative value would not normally be expected. Experience shows that too often such funny values are ignored or not checked for by callers, leading to programs that are not robust, exhibit undesirable behavior, or both.

Every exception is represented by an instance of the class Throwable or one of its subclasses. Such an object can be used to carry information from the point at which an exception occurs to the handler that catches it. Handlers are established by catch clauses of try statements.

During the process of throwing an exception, the Java Virtual Machine abruptly completes, one by one, any expressions, statements, method and constructor invocations, initializers, and field initialization expressions that have begun but not completed execution in the current thread. This process continues until a handler is found that indicates that it handles that particular exception by naming the class of the exception or a superclass of the class of the exception. If no such handler is found, then the exception may be handled by one of a hierarchy of uncaught exception handlers - thus every effort is made to avoid letting an exception go unhandled.

The exception mechanism of the Java SE Platform is integrated with its synchronization model, so that monitors are unlocked as synchronized statements and invocations of synchronized methods complete abruptly.

12 – How to summarize ‘clean code’ as short as possible ?

Clean code can be summarized as a code that any developer can read and change easily.

13 - What is the method of hiding in Java ?

Method hiding means subclass has defined a **class method** with the same signature as a class method in the superclass. In that case the method of superclass is hidden by the subclass. It signifies that : The version of a method that is executed will **NOT** be determined by the object that is used to invoke it. In fact it will be determined by the type of reference variable used to invoke the method.

14 - What is the difference between abstraction and polymorphism in Java ?

- Abstraction allows a programmer to design software better by thinking in general terms rather than specific terms while Polymorphism allows a programmer to defer choosing the code you want to execute at runtime.
- Another difference between Polymorphism and Abstraction is that Abstraction is implemented using abstract class and interface in Java while Polymorphism is supported by overloading and overriding in Java.
- Though overloading is also known as compile-time Polymorphism, method overriding is the real one because it allows a code to behave differently at different runtime conditions, which is known as exhibiting polymorphic behavior.