First Week Homework

Q1 - Why we need to use OOP ? Some major OOP languages ?

OOP is used because it's an excellent way to model the way the real world works in a computer program.

Real programs, solving real problems, are often messy affairs, reflecting the complexity of reality. OOP is a great way to break a problem apart into small units, each effectively a small self-contained computer program in itself (if done right). This makes solving the problems much easier, and then maintaining the entire program also becomes much easier.

OOP isn't perfect, and in the hands of a bad programmer, can do more harm than good, but so far it's one of the best ways we've discovered of making complex, maintainable apps that do something useful.

Most popular OOP languages are `C++`, `Java`, `Javascript`, `C#`, `Python`, `Dart`.

Q2 - Interface vs Abstract class ?

- Abstraction in Java Abstraction is a process of hiding the implementation details and showing only functionality to the user. There are two ways to achieve abstraction in java

1. `Interface`
2. `Abstract class`

Another way, it shows only essential things to the user and hides the internal details, for example, sending SMS where you type the text and send the message. You don't know the internal processing about the message delivery.

- Interface An Interface in Java programming language is defined as an abstract type used to specify the behavior of a class. A Java interface contains static constants and abstract methods. A class can implement multiple interfaces. In Java, interfaces are declared using the interface keyword. All methods in the interface are implicitly public and abstract.

Syntax for Declaring Interface To use an interface in your class, append the keyword `implements` after your class name followed by the interface name.

```
public interface InterfaceName {
  //methods signature...
}
```

- Abstract Class A class which is declared with the abstract keyword is known as an abstract class in Java. It can have abstract and non-abstract methods (method with the body).

Syntax for Declaring Abstract Class

```
abstract class ClassName {
  //code...
}
```

- Difference between interface and abstract class
- Abstract classes can have constants, members, method stubs (methods without a body) and defined methods, whereas interfaces can only have constants and methods stubs.
- Methods and members of an abstract class can be defined with any visibility, whereas all methods of an interface must be defined as public (they are defined public by default).
- When inheriting an abstract class, a concrete child class must define the abstract methods, whereas an abstract class can extend another abstract class and abstract methods from the parent class don't have to be defined.
- Similarly, an interface extending another interface is not responsible for implementing methods from the parent interface. This is because interfaces cannot define any implementation.
- A child class can only extend a single class (abstract or concrete), whereas an interface can extend or a class can implement multiple other interfaces.
- A child class can define abstract methods with the same or less restrictive visibility, whereas a class implementing an interface must define the methods with the exact same visibility (public).

Q3 - Why do we need equals and hashcode ? When to override ?

We know that two objects are considered equal only if their references point to the same object, and unless we override equals and hashCode methods, the class object will not behave properly on hash-based collections like `HashMap`, `HashSet`, and `Hashtable`. This is because hash-based collections are organized like a sequence of buckets, and the hash code value of an object is used to determine the bucket where the object would be stored, and the same hash code is used again to find the object's position in the bucket. The key retrieval is basically a two-step process:

1. Finding the correct bucket using `hashCode()` method.
2. Linearly searching the bucket for the key using `equals()` method.

Joshua Bloch says on Effective Java:

```
You must override `hashCode()` in every class that overrides `equals()`.
Failure to do so will result in a violation of the general contract for
`Object.hashCode()`, which will prevent your class from functioning
```

```
properly in conjunction with all hash-based collections, including HashMap,
HashSet, and Hashtable.
```

- Why we override equals() method? The default implementation is not enough to satisfy business needs, especially if we're talking about a huge application that considers two objects as equal when some business fact happens. (like the above Employee example)

It needs to be overridden if we want to check the objects based on the property.

For example, we want to check the equality of employee object by the id.Then, we need to override the equals() method.

Q4 - Diamon problem in Java ? How to fix it?

As we know, multiple inheritance doesn't support java. To achieve multiple inheritance, we can use the default method and static method in java 8. If you haven't read the article "How to achieve multiple inheritance by inhertface" yet. Then you should read it first, after that, it makes sense for you. By use of default method in the interface, we can achieve multiple inheritances. But the diamond problem is still there.

Q5 - Why we need Garbage Collector ? How does it run ?

Garbage Collection is the process of reclaiming the runtime unused memory by destroying the unused objects. In languages like C and C++, the programmer is responsible for both the creation and destruction of objects. Sometimes, the programmer may forget to destroy useless objects, and the memory allocated to them is not released. The used memory of the system keeps on growing and eventually there is no memory left in the system to allocate. Such applications suffer from "memory leaks". Java Garbage Collection is the process by which Java programs perform automatic memory management. Java programs compile into bytecode that can be run on a Java Virtual Machine (JVM). When Java programs run on the JVM, objects are created on the heap, which is a portion of memory dedicated to the program. Over the lifetime of a Java application, new objects are created and released. Eventually, some objects are no longer needed. You can say that at any point in time, the heap memory consists of two types of objects:

```
Live: these objects are being used and referenced from somewhere else

Dead: these objects are no longer used or referenced from anywhere
```

The garbage collector finds these unused objects and deletes them to free up memory.

- How does Garbage Collection Work in Java? Java garbage collection is an automatic process. The programmer does not need to explicitly mark objects to be deleted. The garbage collection implementation lives in the JVM. Each JVM can implement its own version of garbage collection. However, it should meet the standard JVM

specification of working with the objects present in the heap memory, marking or identifying the unreachable objects, and destroying them with compaction.

Q6 - 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. The static keyword is used for a constant variable or a method that is the same for every instance of a class.

The static keyword is a non-access modifier in Java that is applicable for the following:

`Blocks Variables Methods Classes`

- When a member is declared static, it can be accessed before any objects of its class are created, and without reference to any object. For example, in the below java program, we are accessing static method m1() without creating any object of the Test class.
- Static blocks
  If you need to do the computation in order to initialize your static variables, you can declare a static block that gets executed exactly once, when the class is first loaded.

```
static {
    System.out.println("Static block initialized.");
}
```

- Static variables
  When a variable is declared as static, then a single copy of the variable is created and shared among all objects at the class level. Static variables are, essentially, global variables. All instances of the class share the same static variable.

```
static int a = 20;
```

- Static methods
  When a method is declared with the static keyword, it is known as the static method. The most common example of a static method is the main( ) method. As discussed above, Any static member can be accessed before any objects of its class are created, and without reference to any object.

```
static void m1() {
    //codes...
}
```

- Static Classes
  A class can be made static only if it is a nested class. We cannot declare a top-level class with a static modifier but can declare nested classes as static. Such types of classes are called Nested static classes. Nested static class doesn’t need a reference of Outer class. In this case, a static class cannot access non-static members of the Outer class.

```
static class MyNestedClass {
  //codes
}
```

Q7 - Immutability means ? Where, How and Why to use it ?

An immutable object can’t be modified after it has been created. When a new value is needed, the accepted practice is to make a copy of the object that has the new value.

Functional languages support immutability by design. When using an object-oriented or procedural language, objects and procedural code that accesses data structures (respectively) have to be designed explicity to provide protection against unwanted modification.

Immutable objects offer a number of advantages for building reliable applications. As we don’t need to write defensive or protective code to keep application state consistent, our code can be simpler, more concise, and less error-prone than when we define mutable objects.

Effective Java by Joshua Bloch outlines several reasons to write immutable classes:

- Simplicity - each class is in one state only
- Thread Safe - because the state cannot be changed, no synchronization is required
- Writing in an immutable style can lead to more robust code. Imagine if Strings weren't immutable; Any getter methods that returned a String would require the implementation to create a defensive copy before the String was returned - otherwise a client may accidentally or maliciously break that state of the object.

In general it is good practise to make an object immutable unless there are severe performance problems as a result. In such circumstances, mutable builder objects can be used to build immutable objects e.g. StringBuilder.

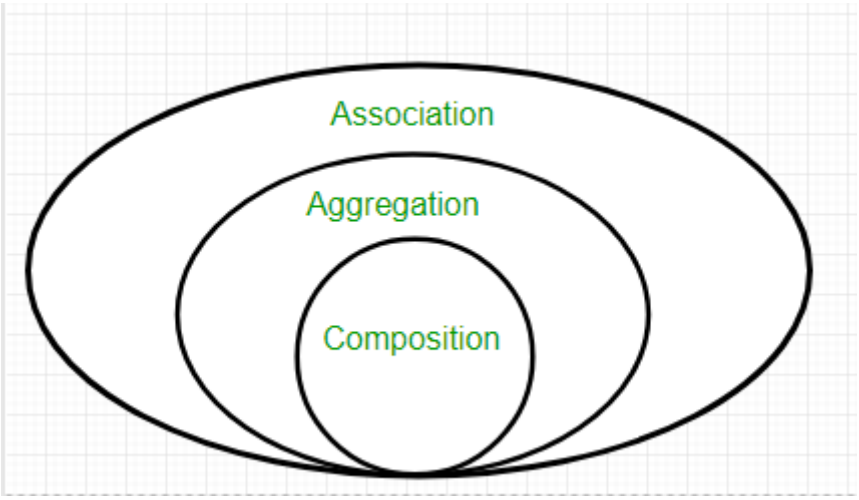
Q8 - Composition and Aggregation means and differences ?

Association is a relation between two separate classes which establishes through their Objects. Association can be one-to-one, one-to-many, many-to-one, many-to-many. In Object-Oriented programming, an Object communicates to another object to use functionality and services provided by that object. Composition and Aggregation are the two forms of association.

Composition is a restricted form of Aggregation in which two entities are highly dependent on each other.

- It represents part-of relationship.
- In composition, both entities are dependent on each other.

- When there is a composition between two entities, the composed object cannot exist without the other entity.



Aggregation vs Composition

- Dependency: Aggregation implies a relationship where the child can exist independently of the parent. For example, Bank and Employee, delete the Bank and the Employee still exist. whereas Composition implies a relationship where the child cannot exist independent of the parent. Example: Human and heart, heart don't exist separate to a Human
- Type of Relationship: Aggregation relation is `has-a` and composition is `part-of` relation.
- Type of association: Composition is a strong Association whereas Aggregation is a weak Association.

Q9 - Cohesion and Coupling means and differences ?

`Cohesion`: Cohesion is the indication of the relationship within module. It is concept of intra-module. Cohesion has many types but usually highly cohesion is good for software.

`Coupling`: Coupling is also the indication of the relationships between modules. It is concept of Inter-module. Coupling has also many types but usually low coupling is good for software.

Differences Cohesion is the concept of intra module. | Coupling is the concept of inter module. Cohesion represents the relationship within module. | Coupling represents the relationships between modules. Increasing in cohesion is good for software. | Increasing in coupling is avoided for software. Cohesion represents the functional strength of modules. | Coupling represents the independence among modules. Highly cohesive gives the best software. | Where as loosely coupling gives the best software. In cohesion, module focuses on the single thing. | In coupling, modules are connected to the other modules.

Q10 - Heap and Stack means and differences ?

When an object is created, it is always created in Heap and has global access. That means all objects can be referenced from anywhere in the application. This is the temporary memory where variable values are stored when their methods are invoked. After the method is finished, the memory containing those values is cleared to make room for new methods.

When a new method is invoked, a new block of memory will be created in the Stack. This new block will store the temporary values invoked by the method and references to objects stored in the Heap that are being used by the method.

- Java Heap Space is used throughout the application, but Stack is only used for the method — or methods — currently running.
- The Heap Space contains all objects are created, but Stack contains any reference to those objects.
- Objects stored in the Heap can be accessed throughout the application. Primitive local variables are only accessed the Stack Memory blocks that contain their methods.
- Memory allocation in the Heap Space is accessed through a complex, young-generation, old-generation system. Stack is accessed through a last-in, first-out (LIFO) memory allocation system.
- Heap Space exists as long as the application runs and is larger than Stack, which is temporary, but faster.

Q11 - Exception means ? Type of Exceptions ?

When program execution ends with an error, an exception is thrown. For example, a program might call a method with a null reference and throw a `NullPointerException`, or the program might try to refer to an element outside an array and result in an `IndexOutOfBoundsException`, and so on.

We use the `try {} catch (Exception e) {}` block structure to handle exceptions. The keyword try starts a block containing the code which might throw an exception. the block starting with the keyword catch defines what happens if an exception is thrown in the try block. The keyword catch is followed by the type of the exception handled by that block, for example "all exceptions" catch (Exception e).

The Finally Block The finally block follows a try block or a catch block. A finally block of code always executes, irrespective of occurrence of an Exception. Using a finally block allows you to run any cleanup-type statements that you want to execute, no matter what happens in the protected code.

```
try {
    // code which possibly throws an exception
} catch (Exception e) {
    // code block executed if an exception is thrown
} finally {
    // The finally block always executes.
}
```

Some exceptions:

1. ClassNotFoundException: happens when a required class cannot be found on the class path.
2. NoSuchMethodException: Trying to access a provided method name that either does not exist or is configured as a private method.
3. NullPointerException: is thrown when a Java program attempts to process an object which contains a null value.
4. ArrayIndexOutOfBoundsException: occurs while processing an array and asking for a position that does not exist within the size of the array.
5. IllegalStateException: is thrown when a method is being called at an illegal or inappropriate time.
6. ArithmeticException: occurs when an exceptional arithmetic condition has occurred.
7. IllegalArgumentException: is often used to capture errors when a provided method value does not meet expectations.

Q12 - How to summarize 'clean code' as short as possible ?

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

Code is easy to extend and refactor, and it’s easy to fix bugs in the codebase.

Q13 - What is the method of hiding in Java ?

- Method hiding is also known as compile-time polymorphism because the compiler is responsible to resolve method resolution based on the reference type.
- It is also known as static polymorphism or early binding.
- In method hiding, method call is always resolved by Java compiler based on the reference type. There is no role of runtime polymorphism in method hiding in java.
- The use of static in method declaration must be the same between superclass and subclass.

Q14 - 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.