HOMEWORK-1

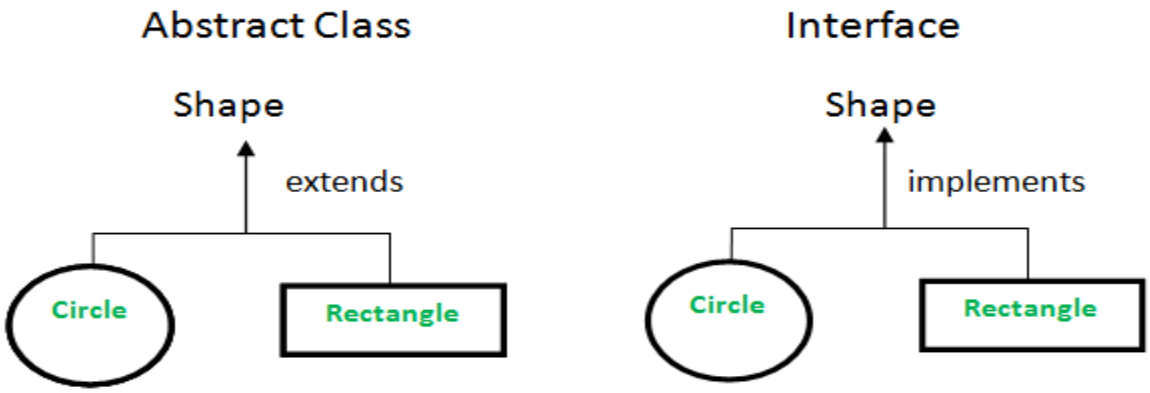
RÜMEYSA TÜRKER

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

The main reason is the increase in the size of the code, the reusability of the written codes and the convenience of working as a team. Therefore we need to use OOP. In addition, OOP language allows to break the program into the bit-sized problems that can be solved easily (one object at a time). It allows you to perform a normal operation in a much shorter time and in a much simpler way. There are 4 main principles for these processes. These are; Encapsulation, Polymorphism, Abstraction and Inheritance.

Some major OOP languages: Java, C++, C#, Python, PHP

2 – Interface vs Abstract class?



Abstract class	Interface
1) Abstract class can **have abstract and non-abstract** methods.	Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also.
2) Abstract class **doesn't support multiple inheritance**.	Interface **supports multiple inheritance**.
3) Abstract class **can have final, non-final, static and non-static variables**.	Interface has **only static and final variables**.
4) Abstract class **can provide the implementation of interface**.	Interface **can't provide the implementation of abstract class**.
5) The **abstract keyword** is used to declare abstract class.	The **interface keyword** is used to declare interface.
6) An **abstract class** can extend another Java class and implement multiple Java interfaces.	An **interface** can extend another Java interface only.

7) An **abstract class** can be extended using keyword "extends".	An **interface** can be implemented using keyword "implements".
8) A Java **abstract class** can have class members like private, protected, etc.	Members of a Java interface are public by default.

9)**Example:**
public abstract class Shape{
public abstract void draw();
}
	Example:
	public interface Drawable{
	void draw();
	}

3 – **Why we need equals and hashcode? When to override?**

equals() is a method used to compare two objects for equality. The default implementation of the equals() method in the Object class returns true if and only if both references are pointing to the same instance. It therefore behaves the same as comparison by ==.

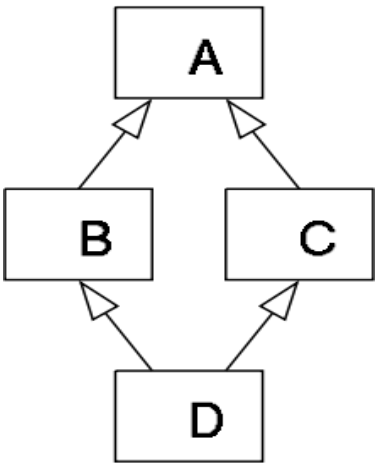
hashCode in Java is a function that returns the hashcode value of an object on calling. It returns an integer or a 4 bytes value which is generated by the hashing algorithm.

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.

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

4 – **Diamond problem in Java? How to fix it?**

The diamond problem in java is coming due to multiple inheritances. Java doesn't support Multiple Inheritance. The problem occurs when there exist methods with the same signature in both the superclasses and subclass. On calling the method, the compiler cannot determine which class method to be called and even on calling which class method gets the priority.



The "**diamond problem**" (sometimes referred to as the "Deadly Diamond of Death") is an ambiguity that arises when two classes B and C inherit from A, and class D inherits from both B and C. If there is a method in A that B and C have overridden, and D does not override it, then which version of the method does D inherit: that of B, or that of C?

The solution to the diamond problem is default methods and interfaces. We can achieve multiple inheritance by using these two things.

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.

Example:

```
interface MyInterface1{
  public static int num = 100;
  public default void display() {
    System.out.println("display method of MyInterface1");
  }
}
interface MyInterface2{
  public static int num = 1000;
  public default void display() {
    System.out.println("display method of MyInterface2");
  }
}
public class InterfaceExample implements MyInterface1, MyInterface2{
  public void display() {
    MyInterface1.super.display();
    //or,
    MyInterface2.super.display();
  }
  public static void main(String args[]) {
    InterfaceExample obj = new InterfaceExample();
    obj.display();
  }
}
```

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

Java garbage collection is the process by which Java programs perform automatic memory management. Java programs compile to bytecode that can be run on a Java Virtual Machine, or JVM for short. When Java programs run on the JVM, objects are created on the heap, which is a portion of memory dedicated to the program. Eventually, some objects will no longer be needed. The garbage collector finds these unused objects and deletes them to free up memory.

Java garbage collection is an automatic process. Automatic garbage collection is the process of looking at heap memory, identifying which objects are in use and which are not, and deleting the unused objects. An in-use object, or a referenced object, means that some part of your program still maintains a pointer to that

object. An unused or unreferenced object is no longer referenced by any part of your program. So the memory used by an unreferenced object can be reclaimed. The programmer does not need to mark objects to be deleted explicitly. The garbage collection implementation lives in the JVM.

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

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

In simple terms, immutability means unchanging over time or unable to be changed. In Java, we know that String objects are immutable means we cant change anything to the existing String objects.

Immutable means that the object's actual value can't be changed, but you can change its reference to another one. In Java, all the wrapper classes (like Integer, Boolean, Byte, Short) and String class is immutable.

There are various reason for immutability:

- Thread Safety: Immutable objects cannot be changed nor can its internal state change, thus there's no need to synchronise it.
- It also guarantees that whatever I send through (through a network) has to come in the same state as previously sent. It means that nobody (eavesdropper) can come and add random data in my immutable set.
- It's also simpler to develop. You guarantee that no subclasses will exist if an object is immutable. E.g. a String class.

So, if you want to send data through a network service, and you want a sense of guarantee that you will have your result exactly the same as what you sent, set it as immutable.

8 – Composition and Aggregation means and differences?

The composition is a design technique in java to implement a has-a relationship.

The composition is achieved by using an instance variable that refers to other objects. If an object contains the other object and the contained object cannot exist without the existence of that object, then it is called composition.

Aggregation in Java is a relationship between two classes that is best described as a "has-a" and "whole/part" relationship. It is a more specialized version of the association relationship. The aggregate class contains a reference to another class and is said to have ownership of that class. Each class referenced is considered to be part-of the aggregate class.

In composition, the dependent object cannot exist without the parent. Whereas, in aggregation, the dependent objects can exists without a parent. The composition is implemented in java by having non-static inner class but aggregation by having a static inner class or object references.

Aggregation vs Composition

1. 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

2. Type of Relationship: Aggregation relation is "has-a" and composition is "part-of" relation.

3. Type of association: Composition is a strong Association whereas Aggregation is a weak Association.

9 – Cohesion and Coupling means and differences?

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 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.

Coupling	Cohesion
Coupling is also called Inter-Module Binding.	Cohesion is also called Intra-Module Binding.
Coupling shows the relationships between modules.	Cohesion shows the relationship within the module.
Coupling shows the relative **independence** between the modules.	Cohesion shows the module's relative **functional** strength.
While creating, you should aim for low coupling, i.e., dependency among modules should be less.	While creating you should aim for high cohesion, i.e., a cohesive component/ module focuses on a single function (i.e., single-mindedness) with little interaction with other modules of the system.
In coupling, modules are linked to the other modules.	In cohesion, the module focuses on a single thing.

10- Heap and Stack means and differences?

The heap is a memory used by programming languages to store global variables. By default, all global variable are stored in heap memory space. It supports Dynamic memory allocation.

The heap is not managed automatically for you and is not as tightly managed by the CPU. It is more like a free-floating region of memory.

A stack is a special area of computer's memory which stores temporary variables created by a function. In stack, variables are declared, stored and initialized during runtime.

It is a temporary storage memory. When the computing task is complete, the memory of the variable will be automatically erased. The stack section mostly contains methods, local variable, and reference variables.

Stack is used for static memory allocation and Heap for dynamic memory allocation, both stored in the computer's RAM.

Feature	Stack	Heap
Cost	Less	More
Variable Deallocation	Not Required	Explicit Deallocation Is Required
Seek Access Time	Faster	Slower
Flexibility	Fixed Size	Resizing Is Available
Locality Of Reference	Automatic Compile Time Instructions	Adequate
Pain Point	Memory Shortages	Memory Fragmentation
Implementation	3 Different Ways - Simple Array-Based, Dynamic Memory, and Linked List Based	2 Different Ways - Array and Trees
Allocation and Deallocation	Automatically Done By Compiler Instructions	Manually Done By Programmer
Memory Allocation	One Contiguous Block	Any Random Order
Variable Resizing	Cannot Be Resized	Can Be Resized
Memory Size	Limited Size-Dependent On OS	No Limit On Memory Size
Variable Access	Local Variables Only	Access Variables Globally
Space Management	OS Efficient Space Management	Blocks Of Memory - FIFO Format
Access Speed	High-Speed Access	Slower Than Slack
Data Structure	Linear Data Structure	Hierarchical D

11 – Exception means? Type of Exceptions?

In Java, exception is an event that occurs during the execution of a program and disrupts the normal flow of the program's instructions. Bugs or errors that we don't want and restrict our program's normal execution of code are referred to as exceptions.

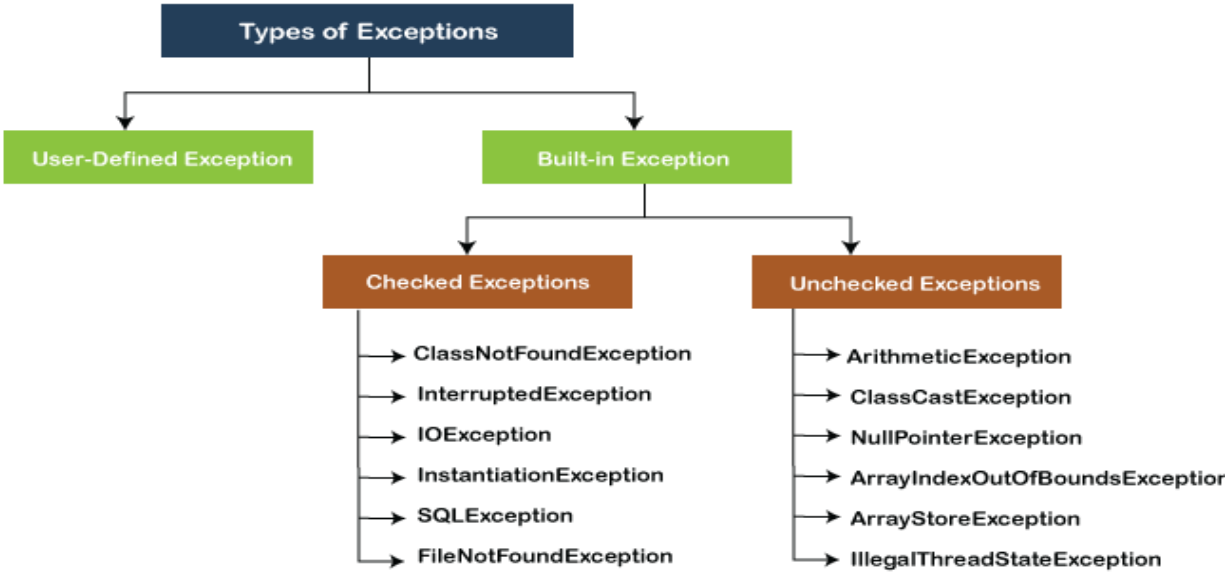
Exceptions can be categorized into two ways:

1. **Built-in Exceptions**
   - *Checked Exception:* Checked exceptions are also known as compile-time exceptions as these exceptions are checked by the compiler during the compilation process to confirm whether the exception is handled by the programmer or not. If not, then the system displays a compilation error. For example, SQLException, IOException, InvocationTargetException, and ClassNotFoundException.
   - *Unchecked Exception:* The unchecked exceptions are those exceptions that occur during the execution of the program. Hence they are also referred to as Runtime exceptions. These exceptions are generally ignored during the compilation process. They are not checked while

compiling the program. For example, programming bugs like logical errors, and using incorrect APIs.

- *Error:* Error is irrecoverable. Some example of errors are OutOfMemoryError, VirtualMachineError, AssertionError etc.

2. **User-Defined Exceptions**: User Defined Exception or custom exception is creating your own exception class and throws that exception using 'throw' keyword. This can be done by extending the class Exception.



12 – How to summarize 'clean code' as short as possible?

Code is clean if it can be understood easily – by everyone on the team. Clean code can be read and enhanced by a developer other than its original author. With understandability comes readability, changeability, extensibility and maintainability.

13- What is the method of hiding in Java?

Method hiding can be defined as, "if a subclass defines a static method with the same signature as a static method in the super class, in such a case, the method in the subclass hides the one in the superclass." The mechanism is known as method hiding. It happens because static methods are resolved at compile time.

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

1) 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.

2) 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.

3) 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.