Kaan Karakaş

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

We use Object-Oriented Programming (OOP) to structure and organize code efficiently. It improves reusability, maintainability, and scalability by encapsulating data and behavior within objects. OOP also enables inheritance and polymorphism, allowing us to extend functionality without modifying existing code. This leads to cleaner, modular, and more maintainable software development. As Object-Oriented languages, we can mention the followings:

Java, C++, Python, Kotlin

2 – Interface vs Abstract class ?

When comparing interfaces and abstract classes, both serve the purpose of abstraction in object-oriented programming, but they have key differences in how they are used.

- Interface is a blueprint that defines what a class should do but not how it should do it. An abstract class can have both abstract methods (without implementation) and concrete methods (with implementation).

- Interfaces typically declare methods without providing implementations (unless using Java 8+ features such as default and static methods), whereas abstract classes allow a combination of abstract and concrete methods, offering the ability for partial implementation.

- Interfaces can only have public static final variables, while abstract classes can have instance variables with various access modifiers.

- An interface is used when multiple classes need to follow the same contract but may have different implementations, while an abstract class is used when classes share common behavior and some methods require a default implementation.

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

We can use the equals() method when we want to compare the contents of two objects. By default, the equals() method checks if the two objects occupy the same memory address. However, when overridden, it can be used to compare the actual contents of the objects. Hash-based collections use the hashCode() method to store and search for objects. Even if two objects are considered equal , if they have different hashCode() values, there may be inconsistencies.

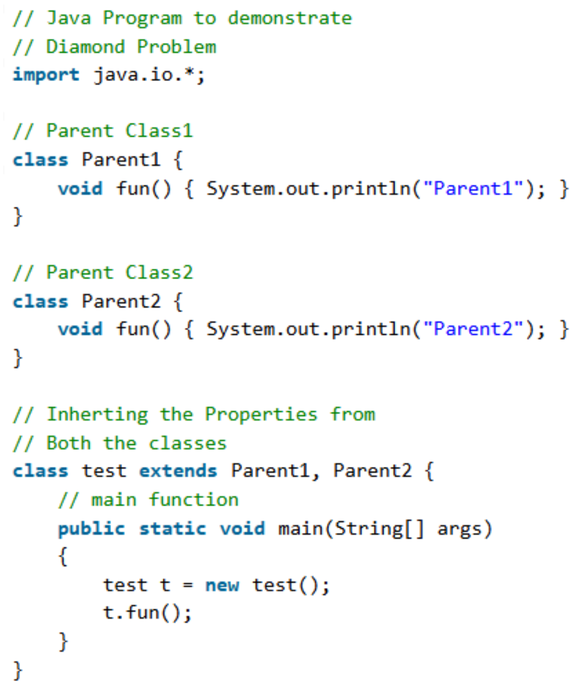
- We should override equals() when we want to compare the actual content of objects, not just their reference equality (memory address).

- If we override equals(), we must also override hashCode() to maintain the contract between the two. If two objects are considered equal, they must return the same hash code. This is critical for the correct functioning of hash-based collections.

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

The Diamond Problem occurs when a class inherits from two or more parent classes that have the same method or field. This leads to a conflict, as the compiler cannot determine which parent class method or field should be used by the subclass.

To solve the Diamond Problem in Java, a good approach is to use interfaces with default methods. While Java doesn't support multiple class inheritance, interfaces allow it. When two interfaces have the same default method, it's necessary for the implementing class to explicitly specify which method to use. This can be done by overriding the method in the class. If needed, both interface methods can be called using the super keyword. By properly managing method overrides and directing which interface method to use, we can avoid confusion and ensure the desired behavior. This way, we can handle multiple inheritance of interfaces in a clear and controlled manner.

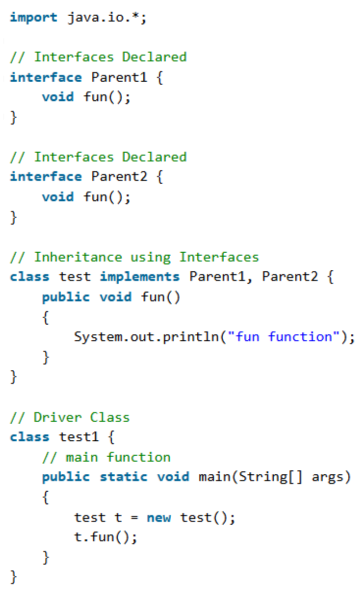


```
// Java Program to demonstrate
// Diamond Problem
import java.io.*;

// Parent Class1
class Parent1 {
    void fun() { System.out.println("Parent1"); }
}

// Parent Class2
class Parent2 {
    void fun() { System.out.println("Parent2"); }
}

// Inheriting the Properties from
// Both the classes
class test extends Parent1, Parent2 {
    // main function
    public static void main(String[] args)
    {
        test t = new test();
        t.fun();
    }
}
```

We can see that the test class is inheriting from two parent classes: parent1 and parent2. Both parent classes define the function fun() , which leads to ambiguity when the test class calls fun(). The compiler is confused about which version of the function it should inherit and use, as there are multiple definitions of fun() in the parent classes. This can result in unexpected behavior or errors in the code.

The Diamond Problem is a common issue in multiple inheritance, but we can resolve it using interfaces. Interfaces, declared with the interface keyword, contain abstract methods by default. Since interfaces cannot be instantiated, a class must implement the interface and provide definitions for all its methods.

We can see how multiple inheritance can be achieved through interfaces.



```
import java.io.*;

// Interfaces Declared
interface Parent1 {
    void fun();
}

// Interfaces Declared
interface Parent2 {
    void fun();
}

// Inheritance using Interfaces
class test implements Parent1, Parent2 {
    public void fun()
    {
        System.out.println("fun function");
    }
}

// Driver Class
class test1 {
    // main function
    public static void main(String[] args)
    {
        test t = new test();
        t.fun();
    }
}
```

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

Garbage Collector (GC) automatically manages memory by removing unused objects. It works by marking reachable objects, deleting unreachable ones, and sometimes compacting memory to avoid fragmentation. Modern GC's use a generational approach, separating objects into young and old generations for efficient collection. There are different GC algorithms like Serial, Parallel, CMS, and G1, each optimized for different performance needs.

6 – Java 'static' keyword usage ?

In Java, the static keyword is used to make a class member shared across all instances of that class. This means you don't need to create an object to access static variables or methods; they belong to the class itself. Static variables are common to all objects of that class, while static methods can be called directly using the class name. A static block is helpful for any initialization that should happen when the class is first loaded into memory. Additionally, static nested classes don't require an instance of the outer class to be created. In short, static makes class members belong to the class itself, meaning they are accessible at the class level rather than objects.

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

-Immutability refers to the concept where an object's state (its data) cannot be altered after it has been created. In other words, once an object is instantiated, its contents cannot be changed. Instead of modifying the original object, new objects are created with updated data, leaving the original object unchanged.

-Immutability is commonly used in functional programming, multithreading and parallel programming, and data security. In multithreading and parallel programming, it prevents data inconsistencies when multiple threads access the same data. It also enhances data security by ensuring that immutable data cannot be altered by other processes, maintaining its integrity.

-Immutability can be used in several ways. First, we create objects only once and avoid changing them, using things like Java's final keyword. Also , we can use immutable collections, such as unmodifiable lists, sets, or maps, to prevent changes. For example, in Java, Collections.unmodifiableList() can make a list unchangeable.

-Immutability is used for several reasons. It helps avoid side effects in functional programming, making functions more predictable because no changes are made to the same object. It ensures data consistency, as data cannot be changed once created, preventing errors or inconsistencies. Immutability reduces the risk of mistakes because changes in one place won't unexpectedly affect the entire program.

8 – Composition and Aggregation means and differences ?

Composition and Aggregation describe the relationships of ownership between objects. In composition, an object has control over another object's lifecycle, meaning if the parent object is destroyed, the child object is also destroyed. In aggregation, an object may own another, but the child object can exist independently, even if the parent object is destroyed. Composition indicates a stronger dependency between objects, while aggregation indicates a looser relationship. In summary, composition links objects closely, while aggregation allows objects to remain independent.

9 – Cohesion and Coupling means and differences ?

Cohesion refers to how well the parts of a class or module are related to each other. High cohesion means the module has one main job or responsibility. Coupling is about how much one class or module depends on others. Low coupling means the modules don't depend on each other much. High cohesion makes the code easier to understand and maintain, while low coupling allows modules to work on their own. In good software design, it's best to have high cohesion and low coupling because this makes the software easier to change and maintain.

10 – Heap and Stack means and differences ?

Heap and Stack are two types of memory in a computer. The Stack is used for storing temporary data like local variables and function calls. It works in a way that the last item added is the first one to be removed (LIFO). The Heap is used for storing larger, more permanent data, and its memory can grow as needed. The Stack is faster and automatically manages memory, while the Heap is slower and needs manual memory management.

11 – Exception means ? Type of Exceptions ?

An exception is an unexpected error that occurs during the execution of a program, disrupting its normal flow. Checked exceptions are checked at compile time and must be caught or passed to the upper levels. Unchecked exceptions occur during runtime and are typically logical errors. Errors represent serious system issues and are usually not caught by the programmer. These types are used to handle errors and ensure the program runs more stably.

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

Clean code should be easy to understand and readable. It should avoid unnecessary complexity and make maintenance easier. Additionally, every line of code should have a specific purpose, and redundant code should be eliminated.

13 – What is the method of hiding in Java ?

In Java, "hiding" refers to the situation where a subclass hides the static method of its superclass. This occurs when the subclass defines its own static method with the same name. Since static methods work at the class level, the method that gets called depends on the class from which it is invoked. Instance methods can be overridden, but static methods can only be hidden

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

Abstraction simplifies complex systems by exposing only the essential and necessary information, while hiding the details. In Java, abstraction is achieved using abstract classes and interfaces. Abstract classes define the method signatures but leave the implementation to subclasses. This allows different subclasses to share the same abstract class while providing different implementations. Polymorphism means the ability for methods with the same name to behave differently based on the objects they are invoked on. There are two types: method overloading and method overriding. Overloading allows the same method to be called with different parameters, while overriding allows a subclass to provide its own implementation of a method inherited from the superclass.

.