Q61. How will you relate unrelated classes or how will you achieve polymorphism without using the base class?

As you progress in an object-oriented design, you will likely encounter objects in the problem domain that contain other objects. In this situation you will be drawn to modeling a similar arrangement in the design of your solution. In an object-oriented design of a Java program, the way in which you model objects that contain other objects is with composition, the act of composing a class out of references to other objects. With composition, references to the constituent objects become fields of the containing object.

Q62.What is Diamond problem in java?

The diamond problem is a common problem in Java when it comes to inheritance. Inheritance is a very popular property in an object-oriented programming language, such as [C++](https://www.javatpoint.com/cpp-tutorial), [Java](https://www.javatpoint.com/java-tutorial), etc. There are different types of inheritance such as, single, multiple, multi-level, and hybrid inheritance. But remember that **Java does not support the multiple inheritance** because of the diamond problem.

As simple inheritance allows a child class to derive properties from one super-class. for example, if class B inherits properties from only one super-class A, then it is called simple inheritance, and Java supports them.

Multi-level inheritance allows a child class to inherit properties from a class that can inherit properties from some other classes. For example, class C can inherit its property from B class which itself inherits from A class. Java also supports them.

What Java does not allow is multiple inheritance where one class can inherit properties from more than one class. It is known as the **diamond problem**. In the above figure, we find that class D is trying to inherit form class B and class C, that is not allowed in Java.

It is an ambiguity that can rise as a consequence of allowing multiple inheritance. It is a serious problem for other OPPs languages. It is sometimes referred to as the **deadly diamond of death**.

Q62. Explain the solution for diamond problem?

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

The default method is similar to the abstract method. The only difference is that it is defined inside the interfaces with the default implementation. We need not to override these methods. Because they are already implementing these interfaces.

The advantage of interfaces is that it can have the same default methods with the same name and signature in two different interfaces. It allows us to implement these two interfaces, from a class. We must override the default methods explicitly with its interface name.

64. Explain the need of abstract class?

Abstract classes permit providing a partial set of default implementations of methods in a class. Since they're incomplete, they can't be instantiated and used as they stand, but they can be subclassed to add the missing details in a way that's specific to that particular implementations, and those subclasses can be instantiated.

Without abstract classes, you would have to provide dummy implementations of the methods you intend to override ... which could be done, but then there'd be the risk of forgetting to implement one of them. Having some methods remain entirely abstract ensures that the real implementations have to fill in the gaps, or continue to be abstract themselves and force their descendents to do so.

It's not something the language couldn't live without. But it's very useful. You'll discover just how useful as you become more proficient in Java and OO design.

65. Why can’t we instantiate abstract class?

It is a class that is specifically designed to be only derived from.that class has pure virtual methods that must be overriden by the class inherits abstract class. instantiating a class that has pure virtual methods is pointless and might cause compiler errors ...

66. Can abstract class have constructors?

* Yes, The main purpose of the constructor is to initialize the newly created object. In abstract class, we have an instance variable, abstract methods, and non-abstract methods. We need to initialize the non-abstract methods and instance variables, therefore abstract classes have a constructor.
* Also, even if we don’t provide any constructor the compiler will add default constructor in an abstract class.
* An abstract class can be inheritedby any number of sub-classes, thus functionality of constructor present in abstract class can be used by them.
* The constructor inside the abstract class can only be called during [constructor chaining](https://www.geeksforgeeks.org/constructor-chaining-java-examples/) i.e. when we create an instance of sub-classes. This is also one of the reasons abstract class can have a constructor.

67. How many instances can be created for an abstract class?

The answer to the question of how many instances of an abstract class can be created is zero. That is, we cannot create an instance of an abstract class as it does not have any complete implementation. An abstract class acts like a template or an empty structure. For an abstract class in the OOP paradigm, we cannot instantiate it.

68. Which keyword can be used for overloading?

If both parent & child classes have the same method, then the child class would override the method available in its parent class. By using the super keyword we can take advantage of both classes (child and parent) to achieve this.

69. Explain the default access specifiers in a class definition?

A default access modifier in Java has no specific keyword. Whenever the access modifier is not specified, then it is assumed to be the default. The entities like classes, methods, and variables can have a default access.

A default class is accessible inside the package but it is not accessible from outside the package i.e. all the classes inside the package in which the default class is defined can access this class.

Similarly a default method or variable is also accessible inside the package in which they are defined and not outside the package.

**class** BaseClass

{

**void** display()      //no access modifier indicates default modifier

       {

           System.out.println("BaseClass::Display with 'dafault' scope");

       }

}

**class** Main

{

**public** **static** **void** main(String args[])

       {

          //access class with default scope

          BaseClass obj = **new** BaseClass();

          obj.display();    //access class method with default scope

       }

}

In the above program, we have a class and a method inside it without any access modifier. Hence both the class and method display has default access. Then we see that in the method, we can directly create an object of the class and call the method.

70. Define all the operators that cannot be overloaded?

In C++ we can overload some operators like +, -, [], -> etc. But we cannot overload any operators in it. Some of the operators cannot be overloaded. These operators are like below

* ? “.” Member access or dot operator
* ? “? : ” Ternary or conditional operator
* ? “::” Scope resolution operator
* ? “.\*” Pointer to member operator
* ? “sizeof” The object size operator
* ? “typeid” Object type operator

These operators cannot be overloaded because if we overload them it will make serious programming issues.