Q. Explain why composition preferred over inheritance?

- Inheritance is tightly coupled whereas composition is loosely coupled.
- There is no access control in inheritance whereas access can be restricted in composition. We expose all the superclass methods to the other classes having access to subclass. So if a new method is introduced or there are security holes in the superclass, subclass becomes vulnerable. Since in composition we choose which methods to use, it's more secure than inheritance.
- One more benefit of composition over inheritance is testing scope. Unit testing is easy in composition because we know what all methods we are using from another class. We can mock it up for testing whereas in inheritance we depend heavily on superclass and don't know what all methods of superclass will be used. So we will have to test all the methods of the superclass. This is extra work and we need to do it unnecessarily because of inheritance.

Q. Can class extend enum? Can Enum implement interface? What is ordinal in Enum?

- All enumerations internally extend a class named Enum the base class of all the language enumeration types. Since Java doesn't support multiple inheritance you cannot extend another class with enumeration if you try to do so, a compile time error will be generated.
- The ordinal () method returns the index of the Enum value.

Q. Explain forLoop, ForEach which one is faster?

- The for loop method is faster when using ArrayList because the for-each is implemented by the iterator, and it needs to perform concurrent modification verification.

- When using LinkedList, the for-each is much faster than the for-loop, because LinkedList is implemented by using a two-way linked list. Each addressing needs to start from the header node. If we need to traverse LinkedList, we need to avoid using the for-loop.

- Using the iterator pattern, for-each does not need to care about the specific implementation of the collection. If there is a need to replace the collection, it can be easily replaced without modifying the code.

Q. Explain the types of association

1. Inheritance: (IS-A Association)

It is the mechanism in java by which one class is allowed to inherit the features (fields and methods) of another class.

A class that inherits from another class can reuse the methods and fields of that class. In addition, you can add new fields and methods to your current class as well.

How to use inheritance in Java?

The **extends keyword** is used for inheritance in java. Using the extends keyword indicates you are derived from an existing class. In other words, "extends" refers to increased functionality.

Syntax :

```
class derived-class extends base-class
{
   //methods and fields
}
```

2. Aggregation:

It is a special form of Association where:

- It represents Has-A's relationship.
- It is a **unidirectional association** i.e. a one-way relationship. For example, a department can have students but vice versa is not possible and thus unidirectional in nature.
- In Aggregation, **both the entries can survive individually** which means ending one entity will not affect the other entity.

3. Composition:

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.

Example of Composition

```
//Car must have Engine
public class Car {
   //engine is a mandatory part of the car
   private final Engine engine;

   public Car () {
      engine = new Engine();
   }
}

//Engine Object
class Engine {}
```

Example of Aggregation

```
//Team
public class Team {
   //players can be 0 or more
   private List players;

   public Car () {
      players = new ArrayList();
   }
}
//Player Object
class Player {}
```

Q. What is Autoboxing and unboxing in java

1. Autoboxing refers to the conversion of a primitive value into an object of the corresponding wrapper class is called autoboxing. For example, converting int to Integer class. The Java compiler applies autoboxing when a primitive value is:

- Passed as a parameter to a method that **expects an object** of the corresponding wrapper class.
- Assigned to a variable of the corresponding **wrapper class**.

2. Unboxing refers to converting an object of a wrapper type to its corresponding primitive value. For example, conversion of Integer to int. The Java compiler applies to unbox when an object of a wrapper class is:

- Passed as a parameter to a method that **expects a value** of the corresponding primitive type.
- Assigned to a variable of the corresponding **primitive type**.