Mouath Sami Masalmah

No: 1220179

Date: 1/12/2023

“Solution for Pre\_Lap 4 ”

Dr: Fadi khalil

Q1:

No, We cannot create an instance of an interface in Java using the `new` keyword. Interfaces are abstract and do not have a constructor that can be used for instantiation.

We can create an instance of a class that implements the interface, but not directly of the interface itself.

Q2:

Yes, you can declare a reference variable `x` with the type of an interface `A` in Java. Example:

public interface A {

// Interface methods

}

// Declaration of a reference variable with the type of the interface

A x;

// You can later assign an object of a class that implements A to x

x = new SomeClassImplementingA();

In this example, `A` is an interface, and `x` is declared as a reference variable of type `A`. You can later assign an instance of a class that implements the `A` interface to the variable `x`. This allows for polymorphism and flexibility in your code.

Q3:

True. If a class implements the Comparable interface, its objects can invoke the compareTo method to define the natural ordering of instances of that class.

Q4:  
No, a class cannot invoke super.clone() when implementing the clone() method if the class does not implement java.lang.Cloneable. Attempting to do so would result in a CloneNotSupportedException. To support cloning, the class must implement the Cloneable interface, indicating its compatibility with the cloning mechanism.

Q5:

Yes, the Date class in Java implements the Cloneable interface. This means that instances of the Date class can be cloned using the clone() method

Q6:

interface Engine {

void start();

}

// Class implementing Engine interface

class Car implements Engine {

public void start() {

System.out.println("Car engine started.");

}

}

In this example, using an interface (Engine) allows the Car class to implement the start() method, promoting flexibility and avoiding limitations associated with single inheritance.

Q7: **Similarities:**

1. Both support abstraction and polymorphism.
2. Both can be used for inheritance.

**Differences:**

1. Abstract classes can have constructors and fields; interfaces cannot.
2. Abstract classes can have both abstract and concrete methods; interfaces can have abstract, default, and static methods.
3. A class can extend only one abstract class but implement multiple interfaces.
4. Abstract class members can have different access modifiers; interface members are implicitly public.
5. Abstract classes are used for code sharing; interfaces define a contract for multiple implementations.