Ch. 8 CSE3130 - Object Oriented Programming 2: CRT Questions

Zephram Gilson

Questions 1, 2, 3, 4, & 6

1.

Has-a Relationship (Composition): A class contains or owns an instance of another class. It is used when one object uses another object as part of its functionality but doesn't inherit from it. For example, a `Car` class might have a has-a relationship with an `Engine` class, meaning `Car` contains an `Engine` but is not a type of `Engine`.

Is-a Relationship (Inheritance): Indicates that a class is a specialized form of another class and inherits its behavior and characteristics. It's implemented using inheritance. For example, a `Dog` class is-a Animal, meaning `Dog` inherits from `Animal` and represents a more specific type.

2.

If a base class has a public method `go()`, and a derived class has a public method `stop()`, an object of the derived class will have access to both methods:

Available Methods: The derived class object can call both `go()` (from the base class) and `stop()` (from the derived class) because the derived class inherits all accessible public and protected methods from the base class in addition to its own methods.

3.

Implementing an Abstract Method: When a class inherits an abstract method (declared in an abstract class or interface), it must provide a concrete implementation of that method. The method has no body in the abstract class/interface, so the subclass supplies the logic.

Overriding a Method: Occurs when a subclass provides a specific implementation for a method that it inherits from its superclass. The base method has an existing implementation, but the subclass can redefine it to add specialized behavior. Overriding is optional and used to alter the behavior of the inherited method.

4.

Abstract Class:

- Can contain both abstract (unimplemented) methods and concrete (implemented) methods.
- Allows instance variables, constructors, and modifiers on methods.
- Inheritance is restricted to single inheritance (a class can only extend one abstract class).

Interface:

- Abstract by default
- Cannot contain instance variables (only constants, i.e., `public static final` variables).
- Supports multiple inheritance (a class can implement multiple interfaces).

6.

a) In `Wo`, `doThat()` is an **abstract** method.
b) `Wo` is an **interface**.
c) The `doThat()` method is implemented in `Roo` because `Roo` implements the `Wo` interface, which mandates that any class implementing it must provide an implementation for all its abstract methods.
d) A `Roo` object will have access to the following methods:
    1) **`doThis():`** The overridden version in `Roo` (returns `10`).
    2) **`doThat():`** The method from the `Wo` interface
    3) **`doNow()`**: The inherited method from the `Bo` class (returns `15`).
e) The implementation of `doThis()` in `Roo` **overrides** the `doThis()` method in `Bo`. When `doThis()` is called on a `Roo` object, Java will use the overridden version in `Roo`, returning `10` instead of `2`.
f) The statement `super(1);` in the `Roo` constructor calls the constructor of the superclass `Bo` with an argument of 1. This initializes the `x` variable in `Bo` with the value 1.
g) Yes, the `doThis()` method in `Bo` can be called from a `Roo` object using the `super` keyword within a method of `Roo`. For example, `super.doThis()` can be called inside a `Roo` method to access `Bo`'s implementation of `doThis()`.
h) Yes, a method in `Roo` can call the `doThis()` method in `Bo` by using `super.doThis()`. This allows access to