INFO 5100 - Application Engineering & Dev

Homework to Chapter 10

Submitted By:

Abdul Haseeb Khan

NUID: 002844724

khan.abdulh@northeastern.edu

In the following pages, I will answer the questions in the Midterm Exam, providing detailed responses. Your feedback is greatly appreciated, and I am open to any additional guidance or expectations you may have for this assignment.

Non-Programming Assignment

1. Can a class be derived (inherit) from two different independent classes?

Ans: The phenomenon of a class that is derived from a single class is called single inheritance. When a class that inherits from more than one independent class is known to have multiple inheritance. The Java programming language does not support multiple inheritance. One class can only inherit from one parent class in this programming language. Although, Java has a concept that is called interface which consists of multiple abstract methods. Each class can implement multiple interfaces.

2. Can a class be derived (inherit) from a “final” class?

Ans: Classes that has a non-access modifier as “final” can not be inherited by any class.

3. What is polymorphism, how it works, and why is it needed?

Ans: Polymorphism in Java refers to the ability of different classes to be treated as objects of a common interface or superclass. It allows a single interface to represent different types of objects, providing a mechanism for flexibility and code reuse in object-oriented programming. Polymorphism in Java is implemented through two main concepts: method overloading and method overriding.

1. Method Overloading (Compile-time Polymorphism):
    - Involves defining multiple methods in the same class with the same name but different parameter types or numbers of parameters.
    - The correct method to be called is determined at compile-time based on the method signature.
    - Example:

```
public class Calculator
{
     public int add(int a, int b)
     {
          return a + b;
     }
     public double add(double a, double b)
     {
          return a + b;
     }
}
```

2. Method Overriding (Runtime Polymorphism):
    - Occurs when a subclass provides a specific implementation for a method that is already defined in its superclass.
    - The decision about which method to call is made at runtime based on the actual type of the object.
    - Example:

```
// Superclass
public class Animal
{
     public void makeSound()
     {
          System.out.println("Some generic sound");
     }
}
// Subclass
public class Dog extends Animal
{
     @Override
     public void makeSound()
     {
          System.out.println(“Woof, Woof”);
     }
}
```

Polymorphism in Java provides a mechanism for designing flexible and extensible code by allowing objects of different classes to be treated uniformly through a common interface. It enhances code readability, maintainability, and promotes the principles of object-oriented programming.

4. What is an inner (nested) class?

Ans: An inner or nested class is a class that is declared inside an existing class. In Java, There are different types of inner classes, such as:

- Static Nested Class: A static class that is nested within another class.
- Non-static (Inner) Class: A class that is a member of another class and has access to the instance variables and methods of the outer class.
- Local Inner Class: A class defined within a block of code, typically within a method.
- Anonymous Inner Class: A class without a name that is declared and instantiated at the same time.

5. What is an abstract class and why is it needed?

Ans: In Java, an abstract class is a class that cannot be instantiated on its own and is typically used as a base class for other classes. It is declared using the abstract keyword. Abstract classes can have both abstract methods and concrete methods.

Abstract classes in Java serve several important purposes:

Abstraction: Abstract classes allow you to define abstract methods, which are methods without a body. This enforces that concrete subclasses must provide an implementation for these methods. This mechanism enables you to abstract away the implementation details in the abstract class, providing a high-level interface while leaving the details to be implemented by subclasses.

Code Reusability: Abstract classes promote code reusability by allowing you to define common functionality in a base class. Subclasses can then inherit this functionality without having to re-implement it. This helps to avoid code duplication and makes the codebase more maintainable.

Polymorphism: Abstract classes support polymorphism, which means that a reference variable of the abstract class type can refer to an object of any concrete subclass. This is useful for creating code that can work with objects of different but related types through a common interface.

6. What is an “interface” class in Java?

Ans: An **interface** is a collection of abstract methods. It is similar to an abstract class in that it declares methods without providing implementations. However, unlike abstract classes, an interface cannot have any concrete methods or instance variables. Java interfaces provide a way to achieve abstraction and define a contract that implementing classes must adhere to.

Here are key characteristics of interfaces in Java:

Abstract Methods: Interfaces can declare abstract methods, which are methods without a body. These methods are implicitly public and abstract. Classes that implement an interface must provide concrete implementations for all the methods declared in the interface.

No Constructors: Interfaces cannot have constructors because they cannot be instantiated on their own. They are meant to be implemented by classes.

No Instance Variables: Interfaces cannot have instance variables (fields) other than constants, which are implicitly public, static, and final.

Multiple Inheritance: Unlike classes, Java allows a class to implement multiple interfaces. This supports a form of multiple inheritance, where a class can inherit from multiple sources of functionality.