Assignment 8 Greg Olivares

1. Class and Object:

A class is a data type that holds different functions / methods that operate with the internal data. A collection of classes that interact with each other is known as a Java Program. Inside of a class may also be Constructors and objects.
Objects refer to a variable whose data type is a class. The object will represent a Constructor by passing a reference to call the class, however each object is in its own state.

2. Class members

These are the different components that all contribute to making a class. Some examples include Variables, Constructors, methods, and Nested classes. All Combine to the structure and behavior of objects within a class. They do so through the use of encapsulation, abstraction, and modularity to create a Java Program.

3. Encapsulation / Method hiding

Encapsulation is one of the four main principles of OOD. It refers to the combination of data and methods that fall within one class. Within a Java program for example, data can be encapsulated by creating private variables but then be modified only by public methods.

Method hiding can be better understood by hiding a method within a class that has the same name / parameter of another method. In this case, the subclass will be able to call the same named method however it will refer to a static method in another class that extends an initial class

4. Generalization

Generalization can be percieved the same as method hiding that creates multiple subclasses which extend a single class however these subclasses share similar behaviors. Generalization promotes code reusability due to if the shared superclass gets altered, all superclasses will also be equally changed

5. Composition and Aggregation

Both of these relate to how the objects in one class can be related to another class. In a Composition manner, the composite class contains and owns a component class object. The component object cannot be made without there being a composite class. Aggregation however is the opposite, where the aggregate class does not own the component class, meaning any change to the aggregate class will not affect the component object.

6. Dynamic Allocation

This refers to allocating memory for objects at runtime rather than during compile time. The benefit of this is that it provides memory flexibility in managing memory resources based on program logic. Within a Java Program for example, creating variables / arrays with "new" will dynamically allocate resources for it.

7. Static Method matching

By using a reference type, it involves the process of invoking static methods. This is resolved during compile time that is then resolved by dynamic Method dispatch. Similary to generalization, the methods have the same name but are called using different methods

8. Dynamic binding

Here, the execution of a method is determined by runtime which is based on the type of object rather than a reference. This also achieves Polymorphism. Another term for this is "late binding" due to the method being called late in a programs execution.

9. Polymorphism

This allows different classes to be treated as objects of a singular Superclass. It achieves doing so From two techniques. Method overriding implements a method that is already defined in the Superclass. Polymorphism allows code flexibility and prevents duplication to be easier to read.

10. Deep Copy / Shallow copy

These both are different techniques that are meant to duplicate objects or data structures. Deep Copy refers to the duplication of objects through recursion whether they are directly or indirectly referenced. On the other hand, Shallow copy only duplicates the top-level structure of the object or structure. It is used when creating new objects without duplicating what they refer to

11. Fat Interface

This refers to the bad habit / inefficient use of various Methods that do not relate to each other and all serve different purposes. Fat interface is a bad coding practice that will lead to the code being difficult to understand and to maintain.

12. Open-Closed Principle

This involved the Notion that classes, Modules, functions, should be open for extension but closed for modification. Essentially, the source code should not be modified. One way to achieve this is through following the principles of Composition between classes and methods. The benefit of this principle is it helps with code maintenance.

13. Dynamic/static linking

Both linking techniques involve how different components of code relate to each other at different times. Dynamic linking resolves dependencies at runtime which reduces the memory footprint of the program. Static linking combines code and data from multiple libraries onto one executable program. This technique is faster than dynamic and more portable in environments.

14. Fragile base Class problem

This is a software design issue where the subclasses of a superclass become dependent on it for implementation. Any changes made to the base class will directly affect the fucctionality of its subclasses. It challenges maintenance based on the subclass all depending on the main class and can break the inheritance of those subclasses.