Assignment 8 - OOP

Claire Moscare April 25, 2024

Class and Object - In object-oriented programming, a class is like a cookie cutter for creating objects. It defines the properties and behaviors that object of the class will have. Object is an instance of a class. It represents a specific entity with its own unique state and behavior. For example, a class of "pen" might have attributes such as "color" and "Tip point". An object created from this class could represent a particular pen with its own color and methods.

Class members - Class members are the variables/attributes and functions that belong to a class. They define the structure and behavior of objects created from that class. Class members can be either static or non-static. Static members are shared among all instances of the class, while non-static members are unique to each other. These members can be accessed using the dot notation, specifying the object followed by the member name.

Encapsulation, Information hiding - Encapsulation is the bundling of data and methods that operate on the data into a single unit, Also known as class. It hides the internal state of objects from the outside world and only exposes the necessary functionalities through methods. Information hiding is a principle of OOP that emphasizes restricting access to certain parts of an object, such as its data, and only exposing the necessary interfaces for interacting with it. This promotes modularity, reduces complexity, and enhances security by preventing unintended access and modification of internal data.

Generalization - Generalization is a fundamental concept in OOP that involves creating a more general class from a set of more specialized classes. It allows common attributes and behaviors to be grouped together in superclass, which can then be inherited by subclasses. This promotes code reuse and helps in organizing classes in hierarchical manner, where subclasses inherit properties and methods from their parent classes.

Composition and Aggregation - Both are two forms of association between classes in OOP. Composition implies a strong relationship where the child object cannot exist independently of the parent object. In contrast, aggregation implies a weaker relationship where the child object can exist independently of the parent object. Composition is often represented as "has-a" relationship while aggregation is represented as a "has a" or "uses a" relationship.

Dynamic Binding - It is also known as late binding or runtime polymorphism. It is a mechanism where the method call is resolved at runtime rather than compile-time. It allows a program to determine which implementation of a method to call based on the actual type of the object being referenced, rather than the reference type. Dynamic binding is a key feature of inheretence and polymorphism in OOP languages like Java. It enables flexibility and extensibility in object behavior.

Dynamic Allocation - it refers to the process of allocation memory for objects at runtime, as opposed to static allocation where memory is allocated at compile-time. In languages like Java, dynamic allocation is typically done using keywords such as 'new'. It allows for flexible memory management and is often used when the size of an object is not known until runtime or when objects need to be created and destroyed dynamically.

Static Method Matching - It refers to the process by which the correct version of a static method is selected at compile-time based on the type of reference variable used to call the method. In languages like Java, static methods are resolved based on the refere type rather than the actual object type. This means that if a subclass overrides a static method from its superclass, calling the static method using a reference variable of the superclass will still invoke the superclass's method.

Polymorphism - it is a core concept in OOP that allows objects of different classes to be treated as objects of a common superclass. It enables a single interface to represent multiple underlying forms. They're two main types of polymorphism. First, compile-time polymorphism, it is achieved through method overloading and operator overloading, and second, runtime polymorphism it is achieved through method overriding and inheritance.

Deep copy, shallowcopy — Both methods of copying objects in programming. A shallow copy creates a new object but does not recursively copy the contents of original object. Instead it copies references to the original objects data. A deep copy creates a new object and resursively copies all the contents of the original object, including any nested object resulting in two independent objects with no shared references.

Fat Interface - It is an anti-pattern in software design where an interface contains more methods than the implementing classes actually need. It violates the interface segregation principle which states that client shouldn't be forced to depend on methods they don't use. Fat interfaces can lead to unecessary dependencies and make the system more difficult to maintain and extend. To address this issue, interfaces should be kept focused and cohesive containing only the methods that are relevant to the implementing classes.

Open-closed Principle - It is a design principle in OOP that states that software entities such as classes and function should be open for extension but closed for modification. This means that a class should be easily extensible to accomodate new behavior or requirements without requiring changes to its existing code. OOP encourges the use of abstraction and polymorphism to achieve flexibility and maintainability in software systems allowing them to evolve over time w/o breaking existing functionality.

Dynamic linking and Static linking - Both are two methods of linking libraries or modules with a program. Static linking involves combining the object code of the program and the libraries into a single executable file before the program is run. Dynamic linking links the program to the libraries at runtime allowing multiple programs to share a single copy of the library code. This results in smaller executable files and allows for easier updates to the library code w/o recompiling the programs that use it.

Fragile Base class Problem - It is a software design issue that arises when changes to a base class can unintentionally break subclasses that depend on it. This can occur when subclasses rely on implementation details of the base class such as internal data representation or behaviour which are not part of the base class or public interface. Changes to the base class such as adding new methods or modifying existing ones can cause unexpected side effects in the subclasses, leading to errors or malfunctions.