REPORT

BY: ABLA SAAD

1. Abstract Classes:

Definition: An abstract class is a class in object-oriented programming that cannot be instantiated on its own. It serves as a blueprint for other classes and may contain abstract methods (methods without a body) that must be implemented by its subclasses. Abstract classes can also have concrete methods with a defined implementation.

Purpose: Abstract classes provide a way to define common behavior and structure for a group of related classes, ensuring consistency and allowing for code reuse. Subclasses that inherit from an abstract class are required to implement the abstract methods, enforcing a specific contract.

2. Static

Definition: In the context of object-oriented programming, the keyword "static" is used to declare members (variables or methods) that belong to the class itself rather than to instances of the class. Static members are shared among all instances of the class and can be accessed without creating an object.

Purpose: Static members are used to represent class-wide properties or behaviors that are independent of a specific instance. They are commonly used for utility methods, constants, or variables that should be consistent across all instances of a class.

3. Encapsulation:

Definition: Encapsulation is one of the fundamental principles of object-oriented programming that involves bundling the data (attributes) and methods (functions) that operate on the data into a single unit known as a class. It restricts access to the internal details of an object and only allows interaction through well-defined interfaces.

Purpose: The main purpose of encapsulation is to achieve information hiding and protect the integrity of an object. By encapsulating the internal details, a class can control access to its data and methods, preventing external code from directly manipulating the object's state. This enhances modularity and makes it easier to maintain and evolve the code.

4. Polymorphism:

Definition: Polymorphism is a concept in object-oriented programming that allows objects of different types to be treated as objects of a common base type. It enables a single interface to represent various types and allows methods to be written to handle objects of a generic type, providing flexibility and extensibility.

Types of Polymorphism:

1. Compile-time (Static) Polymorphism: This is achieved through method overloading and operator overloading. It involves the compiler determining the appropriate method or operator to be called at compile time based on the method signature.

2. Runtime (Dynamic) Polymorphism: This is achieved through method overriding. It involves the selection of the appropriate method at runtime based on the actual type of the object.

Purpose: Polymorphism allows for code reusability and flexibility by allowing objects to be treated at a higher level of abstraction. It simplifies code maintenance and promotes a more modular and extensible design.