Java OOPS Concepts

- Java OOPS stands for Java Object Oriented Programming System
- The main purpose of Object Oriented Programming System is to implement real world entities (i.e. Objects) such as Chairs, Laptops, Pens, Cars etc.
- Java OOPS concepts can be categorized as below:
    - Object
    - Class
    - Inheritance
    - Polymorphism
    - Abstraction
    - Encapsulation
- **Object**
    - Object is nothing but a real world entity that has **state** and **behavior**.
        - Example: Chairs, Laptops, Pens, Cars, Dogs etc.
        - **state**: Example: A Dog has states like Color, Name, Breed etc.
        - **behavior**: Example: A Dog has behaviors like Barking, Eating, Sleeping etc.
    - Technically speaking Object is an instance of a Class
    - Example for Object will be provided while explaining Class
- **Class**
    - Class is a template/blueprint which can be used for creating Objects
    - Example for Objects and Class - View here
    - Demonstrate creating Class and Objects - View code here
- **Inheritance**
    - Inheritance is a mechanism in which one class acquires the properties (i.e. variables and methods) of another class.
        - Instead of recreating the common variables and methods in a new class, we use inheritance concept to inherit the properties from the earlier created class
            - Advantage: Code Re-usability to avoid duplication
        - Child class acquires the properties (variables and methods) of Parent class
        - Child class uses **extends** keyword for inheriting the properties from parent class
        - Demonstrate a child class which inherits the properties from Parent Class - Demonstrate here
            - Child class can have specific properties (i.e. variables and methods) which are not available in the parent class
            - Object created for parent class can access the variables and methods that are created in parent class only. It cannot access the child class properties.
            - Object created for child class which is inheriting the parent class can access the variables and methods of both parent class and child class.
    - 'IS A' relationship
        - Example: **Benz** is a **Car**
    - Types of Inheritance:
        - There are 3 types of inheritance:
            - Single Inheritance - View here
                - Single level of hierarchy
                - Demonstrate using a practical program
            - Multilevel Inheritance - View here
                - Multiple levels of hierarchy
                - Demonstrate using a practical program
            - Hierarchical Inheritance - View here
                - Multiple class inheriting the same class
                - Demonstrate using a practical program
        - Multiple Inheritance and Hybrid Inheritance are not supported in Java - View here
            - Demonstrate the non possibility of Multiple Inheritance
        - Privatizing the data will stop the data from inheriting
            - Demonstrate
- **Polymorphism**
    - Polymorphism in simple terms is 'one thing in multiple forms'
    - Types of polymorphism:
        - Compile Time Polymorphism
            - Overloading
                - Multiple methods having the same name but with different signatures (i.e. number of parameters, types of parameters, order of parameters)

            - Demonstrate Method Overloading by creating add methods having different number or types of parameters or order of parameters
            - Demonstrate constructor Overloading
    - Runtime Polymorphism
        - Overriding
            - When a method in the Child class (i.e. sub-class) is duplicate of a method in Parent class (i.e. super-class) , then the method in the sub-class is said to override the method in super-class.
                - When we create an Object for Sub-class and call the overridden method, the method in the sub-class will be called - Demonstrate here
                - Even though the name of the method in the sub-class has the same name as a method in super-class, if the type of parameters or number of parameters differ, then the method in the sub-class will overload the method in super-class instead of overriding
                - Constructors cannot be overridden as the name of the constructor needs to be same as the name of the Class.

- **Abstraction**
    - Abstraction is a methodology of hiding the lower level/implementation details from the user and only providing the functionality to them.
        - The user will have information on what it does without knowing how it does
        - i.e. Implementation is abstracted/hidden from the user
    - Ways to achieve abstraction in Java:
        - Abstract class
        - Interface
    - Abstract class
        - variables cannot be specified with 'abstract' non-access modifier - Demonstrate
        - On specifying a method with abstract modifier, we can just declare the method without implementing it - Demonstrate here
        - Classes having at-least one abstract specified method must be specified as abstract
        - Sub-Class inheriting the Super-Class needs to implement the abstract specified methods in Super-Class - Demonstrate here
            - Purpose of abstract methods - Used when the super-class dont have to implement everything, and when the sub-classes inheriting the super-class needs to implement them.
        - Objects cant be created for abstract classes, we have to create a Sub-Class and access its variables/methods using Sub-Class object reference - Demonstrate here
    - Interface
        - The purpose of an interface is to just to declare all the functionalities required before actually implementing them.
            - Interfaces looks similar to Classes and are extensions of abstract classes
            - Interfaces provides 100% abstraction
            - Create an interface say 'Bank' in Eclipse IDE and create variables & methods inside it as shown here
                - Variables in the interfaces are of static and final type by default
                - In abstract classes, we can have both methods (i.e. implemented and non-implemented), where as in interfaces, we cannot implement any methods.
            - Classes use **implements** keyword to implement any interface - Demonstrate here
            - Classes implementing an interface can have their own specific methods apart from methods which are acquired from an interface - Demonstrate here
            - Objects cannot be created for an interface - Demonstrate
            - Object can be created for the Classes which are implementing the interfaces, for accessing interface defined methods and class specific methods - Demonstrate
            - Follow the below steps to provide the access to the interface specific methods and not to access the class specific methods
                - Create an object for the Class which is implementing the interface
                - Assign the object of the class to the interface reference variable
                - Using the interface reference variables, we can now access only the methods which are declared in the interface - Demonstrate here
- **Encapsulation**
    - Encapsulation is a mechanism in java, which wraps the data (i.e. variables) and code (i.e. methods) together as a single unit.
        - i.e. In Encapsulation, you cannot access data (i.e. variables) as a single unit, instead we have to take the help of methods to access the data (i.e. variables)
    - Encapsulation is achieved by privatizing the variables and accessing them by using getters and setters methods to modify and get the variable values

- Demonstrate Encapsulation by privatizing the variables and accessing them using setters and getters methods - Demonstrate here

---