Java - What is OOP?

OOP stands for Object-Oriented Programming

Object-oriented programming has several advantages over procedural programming:

* OOP is faster and easier to execute
* OOP provides a clear structure for the programs
* OOP helps to keep the Java code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug

OOP makes it possible to create full reusable applications with less code and shorter development time.

Java - What are Classes and Objects?

Classes

Classes and objects are the two main aspects of object-oriented programming.

* Class is nothing but a blueprint / a template from which object is created. Contains attributes and methods.
* The object is an instance of a class. It’s nothing but data

Example: 1. **Class (Blueprint) in Java**

A screenshot of a computer code

Description automatically generated

2. **Object (Building) in Java**

A computer screen shot of text

Description automatically generated

**Visualizing the Difference**

| **Aspect** | **Class (Blueprint)** | **Object (Building)** |
| --- | --- | --- |
| **Definition** | A template that defines properties and behavior | A specific instance of the class with actual values |
| **Example** | House class | house1 and house2 objects |
| **Memory** | No memory allocated until an object is created | Memory is allocated for each object |
| **Real-world analogy** | A house blueprint | A real house built from the blueprint |

Java OOPS Concepts:

Java Inheritance Java Encapsulation Java Polymorphism Java Abstraction

## Java Inheritance (Subclass and Superclass)

* **Inheritance** in Java is a mechanism that allows one class (called a **subclass** or **child class**) to inherit the attributes and methods of another class (called the **superclass** or **parent class**).
* We group the "inheritance concept" into two categories:
* subclass (child) - the class that inherits from another class
* superclass (parent) - the class being inherited from
* Its only possible by using Keyword “extends”

|  |
| --- |
| A computer code with text  Description automatically generated |

Java Encapsulation

* **Encapsulation: Means binding code and data together in a single unit.**
* Encapsulation means hiding sensitive data from the user.
* Declare class variables/attributes as private
* provide public **get** and **set** methods to access and update the value of a private variable

Person.java

|  |
| --- |
| public class Person {  private String name; // private = restricted access  // Getter  public String getName() {  return name;  }  // Setter  public void setName(String newName) {  this.name = newName;  }  } |

MyClass.java

|  |
| --- |
| public class Main {  public static void main(String[] args) {  Person myObj = new Person();  myObj.name = "John";  System.out.println(myObj.name);  }  } |

**Java Polymorphism**

* Polymorphism means many forms.
* It allows the same method to perform different operations in different scenarios.
* Polymorphism is a technique that allows to get rid of long if-else-condition OR Switch & case Statements.
* Inheritance lets us inherit attributes and methods from another class.
* Polymorphism uses those methods to perform different tasks. This allows us to perform a single action in different ways.

In Java, polymorphism manifests in two primary ways:

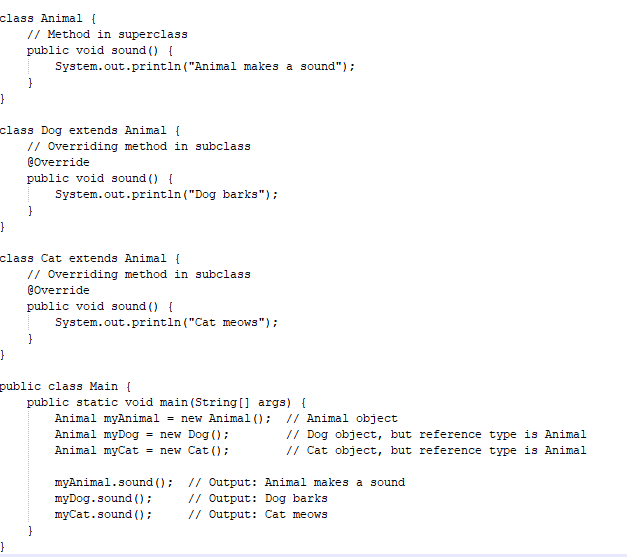
1. **Method Overloading (Compile-time Polymorphism)**
2. **Method Overriding (Runtime Polymorphism)**

* **Method Overloading (Compile-time Polymorphism)**
* This is also called compile time polymorphism
* Static binding
* Can have multiple methods with same name but different variations in parameters such as Type/order/number

A screenshot of a computer program

Description automatically generated

* **Method Overriding (Runtime Polymorphism)**
* This is also called run time polymorphism
* Dynamic binding
* Method signatures remain same, but which method is called is determined at the run time based on the class object

****

**Java Abstraction**

* **Abstraction** is the process of hiding certain details and showing only essential information to the user.
* Abstraction can be achieved with either **abstract classes** or [**interfaces**](https://www.w3schools.com/java/java_interface.asp)

The abstract keyword is a non-access modifier, used for classes and methods:

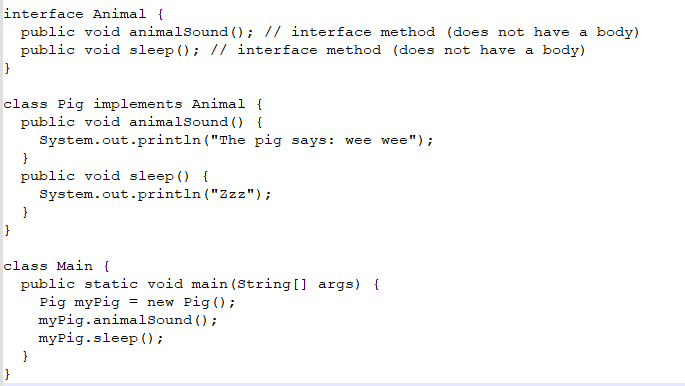
* **Abstract class:** is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).
* **Abstract method:** can only be used in an abstract class, and it does not have a body. The body is provided by the subclass (inherited from).

A screenshot of a computer program

Description automatically generated

**Java Interface**

* Another way to achieve abstraction in Java, is with interfaces.
* An interface is a completely "abstract class" that is used to group related methods with empty bodies
* Interface in java is like a blueprint of a class.
* And it is declared with the keyword before the class name.
* The method in the interface is declared and does not have the body.
* The class that implements the interface needs to implement all methods declared in the interface.
* The class that uses the interface needs to use the keyword “Implements.”



**Homework Questions or Assignments to practice**

**1. Basic Class and Object Creation**

Objective: Understand how to create classes and instantiate objects.

Task:

* Create a class Book with properties such as title, author, and price.
* Add a method to display the details of the book (displayDetails()).
* Instantiate an object of Book and call displayDetails() method.

Requirements:

* Implement a constructor for initializing the book properties.
* Print the details of the book.

**2. Encapsulation**

Objective: Practice the concept of encapsulation (data hiding) using getters and setters.

Task:

* Create a class Person with the fields name (String) and age (int).
* Make the fields private and provide getters and setters for each field.
* Create an instance of Person, set the values using setter methods, and retrieve the values using getter methods.

Requirements:

* Implement validation in setters (for example, ensuring age is positive).
* Demonstrate access control and encapsulation.

**3. Inheritance and Method Overriding**

**Objective**: Demonstrate inheritance and method overriding.

**Task**:

* Create a base class Animal with a method sound().
* Create two subclasses Dog and Cat, which override the sound() method to provide specific behaviors (e.g., "bark" and "meow").
* Create objects of Dog and Cat and demonstrate polymorphism by calling sound() on both.

**Requirements**:

* Use annotation when overriding methods.
* Demonstrate polymorphism using an array or a collection of Animal references.

**4. Polymorphism - Method Overloading**

**Objective**: Understand compile-time polymorphism via method overloading.

**Task**:

* Create a class Calculator with overloaded methods add().
* Implement different add() methods with varying parameter types (e.g., one with two integers, another with two doubles, and another with three integers).
* Instantiate the Calculator class and call the overloaded add() methods.

**Requirements**:

* Demonstrate method overloading based on parameter types and count.
* Print results for each method call.

**5. Abstraction using Abstract Classes**

**Objective**: Understand abstraction and its use in Java using abstract classes.

**Task**:

* Create an abstract class Shape with an abstract method area().
* Create two concrete subclasses Circle and Rectangle, each implementing the area() method based on their respective formulae.
* Instantiate the subclasses and display the area of the shapes.

**Requirements**:

* The Shape class should be abstract, and it should contain a concrete method printArea().
* The Circle and Rectangle classes must implement the area() method.
* The printArea() method should be used to display the area of the shape.

**6. Interface Implementation**

**Objective**: Learn how to implement interfaces in Java.

**Task**:

* Create an interface Playable with methods play() and stop().
* Create two classes MusicPlayer and VideoPlayer that implement the Playable interface.
* Demonstrate calling play() and stop() on objects of both classes.

**Requirements**:

* Implement both play() and stop() methods in both MusicPlayer and VideoPlayer.
* Create an array or list of Playable objects and iterate over them to call play() and stop().

**7. Creating a Banking System (Class and Objects)**

**Objective**: Apply OOP concepts (encapsulation, inheritance, and polymorphism) to design a banking system.

**Task**:

* Create a base class Account with methods deposit() and withdraw().
* Create subclasses SavingAccount and CurrentAccount, both inheriting from Account. Implement additional behavior in each subclass (e.g., SavingAccount may have an interest rate).
* Demonstrate polymorphism by creating an array of Account objects and performing transactions (deposit, withdraw).

**Requirements**:

* Use encapsulation for account balance and transaction details.
* Implement method overriding for interest calculation in SavingAccount.

**10 possible Interview Questions for oops in java**

* 1. **What are the four main principles of Object-Oriented Programming (OOP)? Explain each of them.**
  2. **2. What is the difference between method overloading and method overriding in Java?**
  3. **Explain the concept of inheritance in Java. How does it promote code reusability?**
  4. **What is an abstract class in Java? How does it differ from an interface?**
  5. **What is polymorphism in Java? Can you give an example?**
  6. **What is encapsulation and how is it implemented in Java?**
  7. **What is the significance of the super keyword in Java?**
  8. **What is the purpose of the final keyword in Java?**
  9. **What are constructors in Java? What is the difference between a default constructor and a parameterized constructor?**
  10. **How do you implement the OOP principles in a Selenium framework?**
  11. **How can you apply inheritance in a Selenium test framework?**
  12. **What is Page Object Model (POM), and how does it relate to OOP principles in Selenium?**