OOPS

Object Oriented Programming Language is a type of programming that is based on objects rather than functions and procedures.

Individual objects are grouped into classes.

OOPs implements real world entities like inheritance, polymorphism, hiding into programming

It allows binding data and code together

WHY WE USE OOPS

* OOPS allow us to bound data and code together by encapsulation
* OOPS allows data hiding, so private data is kept confidential
* Code can be reused through inheritance, thereby reducing redundancy.
* The concept of polymorphism gives flexibility to the program by allowing entities to have multiple forms
* A large problem can be divided into different parts, thus making it simple to solve.

FEATURES OF OOPS

* Inheritance
* Encapsulation
* Polymorphism
* Data Abstraction

OBJECT

An object is a real world entity which is the basic unit of OOPS

Different objects have different state and behaviours

Example: Car,Chair.

CLASS

* A class is a user defined blueprint or prototype from which objects are created.   
  Class does not occupy memory.  
  Class is a group of variables of different data types and group of methods.

 In general, class declarations can include these components, in order:

1. **Modifiers**: A class can be public or has default access
2. **Class keyword:**class keyword is used to create a class.
3. **Class name:** The name should begin with an initial letter (capitalized by convention).
4. **Superclass(if any):** The name of the class’s parent (superclass), if any, preceded by the keyword extends. A class can only extend (subclass) one parent.
5. **Interfaces(if any):** A comma-separated list of interfaces implemented by the class, if any, preceded by the keyword implements. A class can implement more than one interface.
6. **Body:** The class body is surrounded by braces, { }.

**INHERITANCE**

Java Inheritance is **a mechanism in which one class acquires the property of another class**.

**Note: A class can extend only one class,but can extend multiple interfaces**

**Keyword:**

* extends

**Purpose:**

* Reusablility and Maintenance

**Types:**

* Single,Mutilple,MultiLevel

**Java Example:**

* Runtime exception extends Exception
* ArrayList extends AbstractList

**Selenium Example:**

* (Class)ChromeDriver extends ChromiumDriver extends RemoteWebDriver(Class)

**Interface Example:**

* (Interface)WebDriver extends SearchContext(Interface)

**Framework Example:**

* Tests and Pages extends BaseClass(ProjectSpecificMethods)

IS MULTIPLE INHERITANCE SUPPORTED IN JAVA?

Multiple Inheritance is a feature of an object-oriented concept, where a class can inherit properties of more than one parent class.

Java does not support multiple inheritance by itself, it is achieved through interfaces only

POLYMORPHISM

Polymorphism is an ability of an object to take multiple forms.Multiple definitions can be given to a single interface

Types

Compile-Time Polymorphism(overloading)

**Method Overloading :**

Method Overloading is a feature that allows a class to have more than one method having the same name, with a different type of parameters or with a different number of parameters or both.

ADVANTAGES

It helps programmers reuse the code and classes once written.

Single variable name can be used to store variables of multiple data type

**Java Example:**

**substring(startindex int), substring(startindex int, endindex int)**

**Selenium Example:**

You will use **driver.switchTo().frame(String id or Name), frame(Webelement element**, **names of the methods** are the same but they accept different types of parameters.  
  
The first **frame(String id or Name)** accepts a String parameter, but the second frame(WebElement element) accepts the Webelement as a parameter.

**Framework Example:**

**locateElement()🡪 default by xpath**

**locateElement(String,String)🡪by any locator**

Run-Time Polymorphism(overriding)

**Java Example:**

**Equals() method is String class overrides the Equals method in Object Class**

**Selenium Example:**

**Quit() method in chromiumdriver overrides the quit() method in RWD**

The differences between Method Overloading and Method Overriding in Java are as follows:

| Method Overloading | Method Overriding |
| --- | --- |
| Method overloading is a compile-time polymorphism. | Method overriding is a run-time polymorphism. |
| It helps to increase the readability of the program. | It is used to grant the specific implementation of the method which is already provided by its parent class or superclass. |
| It occurs within the class. | It is performed in two classes with inheritance relationships. |
| Method overloading may or may not require inheritance. | Method overriding always needs inheritance. |
| In method overloading, methods must have the same name and different signatures. | In method overriding, methods must have the same name and same signature. |
| In method overloading, the return type can or cannot be the same, but we just have to change the parameter. | In method overriding, the return type must be the same or co-variant. |
| Static binding is being used for overloaded methods. | Dynamic binding is being used for overriding methods. |
| It gives better performance. The reason behind this is that the binding of overridden methods is being done at runtime. | Poor performance |
| Private and final methods can be overloaded. | Private and final methods can’t be overridden. |
| Argument list should be different while doing method overloading. | Argument list should be same in method overriding. |

ENCAPSULATION

Encapsulation refers to binding the data and code together in a single unit

It also allows data hiding as data specified in one class is hidden from other classes

In the encapsulation technique, we declare fields as private in the class to prevent other classes from accessing them directly. The required encapsulated data can be accessed by using public Java getter and setter method.

Step 1: Create a package named 'numberTest' and make a class 'Number'.

package encapsulationTest;

public class Number

{

Step 2: Declare field/variable as private.

private int y;

Step 3: Create a public getter method for private variable.

public int get() {

return y;

}

Step 4: Create a public setter method for private variable and declare the parameter.

public void set(int y) {

this.y = y;

}

}

Step4: Create another class EncapTest.

public class Encaptest

{

public static void main(String[] args)

{

Step 6: Create an object of class Number using the new keyword.

Number n = new number(); // Here, n is a reference variable of Number and pointing to the object of class Number.

Step 7: Now call setter method and set value of the variable.

n.set(6);

Step 8: Call getter method to read the value of variable.

int num = n.get();

Step 9: Print the output.

System.out.println(num);

}

}

ABSTRACT CLASS

Abstract class is a class that consist of Abstract methods. Abstract methods are methods which are declared but not defined(no implementation).If these methods are to be used in some subclass then,they need to be exclusively defined in subclass.An abstract class is declared with the keyword “abstract”.

Since it doesn’t have complete implementation , instances of subclass cannot be created.However,instances of subclass inheriting the abstract class can be created.

Syntax: access\_modifier abstract class\_name

Example: pubic abstract Shape

INTERFACE

Interface in java is a blueprint of a class,what a class must do and not how to do. It has static constants and abstract methods

Interface is a mechanism to achieve abstraction and multiple inheritance in java.

Syntax: interface interface\_name

Example:

**Abstract class vs Interface**

* **Type of methods:** Interface can have only abstract methods. An abstract class can have abstract and non-abstract methods.
* **Final Variables:** Variables declared in a interface are by default final. An abstract class may contain non-final variables.
* **Type of variables:**Abstract class can have final, non-final, static and non-static variables. The interface has only static and final variables.
* **Implementation:** Abstract class can provide the implementation of the interface. Interface can’t provide the implementation of an abstract class.
* **Inheritance vs Abstraction:** A Java interface can be implemented using the keyword “implements” and an abstract class can be extended using the keyword “extends”.
* **Multiple implementations:** An interface can extend one or more Java interfaces; an abstract class can extend another Java class and implement multiple Java interfaces.
* **Multiple Inheritance:**  Interface supports multiple inheritance; an abstraction does not support multiple inheritance.
* **Accessibility of Data Members:** Members of a Java interface are public by default. A Java abstract class can have class members like private, protected, etc.

Henceforth Interface and Abstract Class are prerequisite to achieve Abstraction

**Data Abstraction:** [Abstraction](https://www.scientecheasy.com/2020/05/java-abstraction.html/) is a technique by which we can hide the unnecessary data that is not needed for a user and expose only the necessary.

Abstraction permits the user to use an object without knowing its internal details. It helps to reduce complexity by not including background details.

Note:[Unneceassary data means->internal implementation of the feature; necessary data🡪functionality of the feature. What is works(showing), How is works(hiding)]

Example: A realtime example of abstraction is “sending SMS”. When you need to send SMS  from your mobile, you only type the text and send the message. But you don’t know the internal processing of the message delivery.