CSC 311 - OBJECT-ORIENTED PROGRAMMING WITH JAVA (2 CREDIT UNITS)

COURSE OUTLINE

Basic OOP Concepts: Classes, Objects, inheritance, polymorphism, Data Abstraction, Tools for developing, Compiling, interpreting and debugging, Java or C++ Programs, C++ or Java Syntax and data objects, operators. Central flow constructs, objects and classes programming, Window Toolkit, Laboratory exercises in an OOP Language.

Object Oriented Programming:

Object-oriented programming (**OOP**) is a programming paradigm based on the concept of *objects*, which can contain data and code: data in the form of fields (often known as attributes or properties), and code in the form of procedures (often known as methods). In OOP, computer programs are designed by making them out of objects that interact with one another.

Many of the most widely used programming languages (such as C++, Java,[4] and Python) are multi-paradigm and support object-oriented programming to a greater or lesser degree, typically in combination with imperative programming, procedural programming and functional programming.

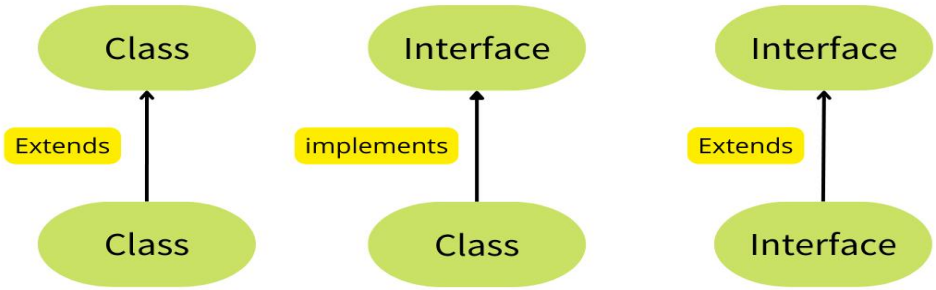
Significant object-oriented languages include Ada, ActionScript, C++, Common Lisp, C#, Dart, Eiffel, Fortran 2003, Haxe, Java, JavaScript, Kotlin, Logo, MATLAB, Objective-C, Object Pascal, Perl, PHP, Python, R, Raku, Ruby, Scala, SIMSCRIPT, Simula, Smalltalk, Swift, Vala and Visual Basic.NET.

Some basic concepts of object-oriented programming (OOP) include:

- **Interface**

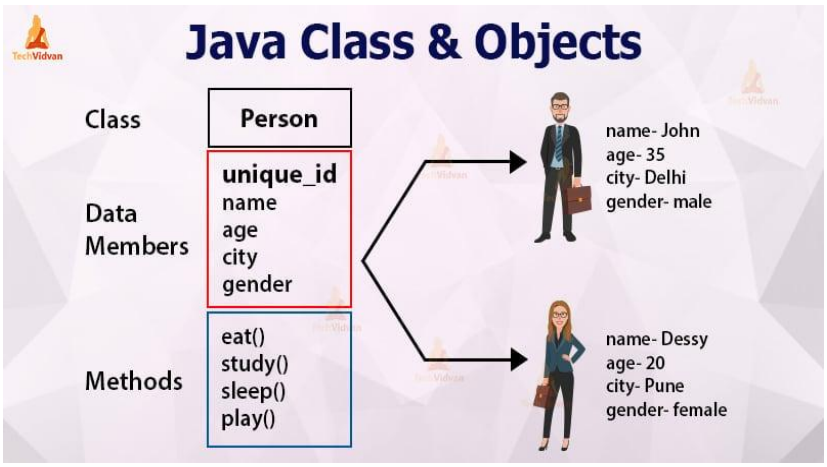
Tells the object what it can do. For example, an interface tells a tree that it can grow.

Understanding OOP concepts is key to understanding how languages like Java work.



- **Class**

A blueprint for an object. For example, all cars have some common properties, such as 4 wheels, speed limit, and mileage range.



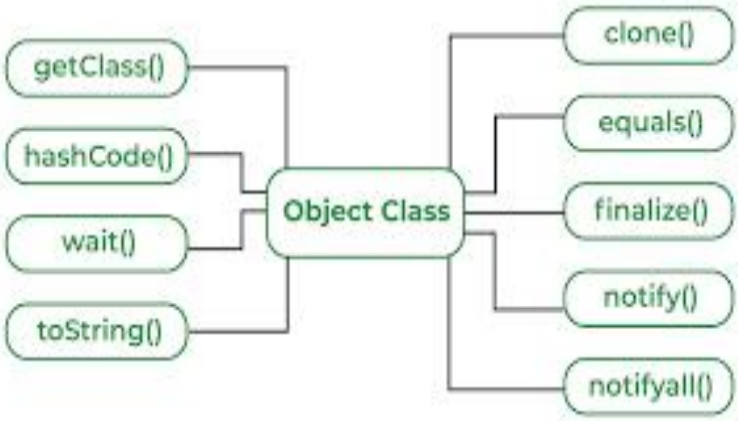

```
// simplest Java class
class Student { }

// in file Student.java
public class Student {
  String name;
  public String getName() {
    return name;
    }
  public void setName(String theName) {
    name = theName;
```

return type

signature

- **Object**

An instance of a class. Each object has all the properties and methods defined in the class, but they will have different property values.



- **Inheritance**

A mechanism that allows one class to inherit the features of another class. This makes it easier to manage information in a hierarchical order.

Types of Inheritance in Java: Here, Animal is the superclass, and Dog is the subclass. The Dog class inherits the eat() method from the Animal class. Additionally, the Dog class introduces a new method called bark(). This is an example of a single inheritance, as Dog extends only one superclass.

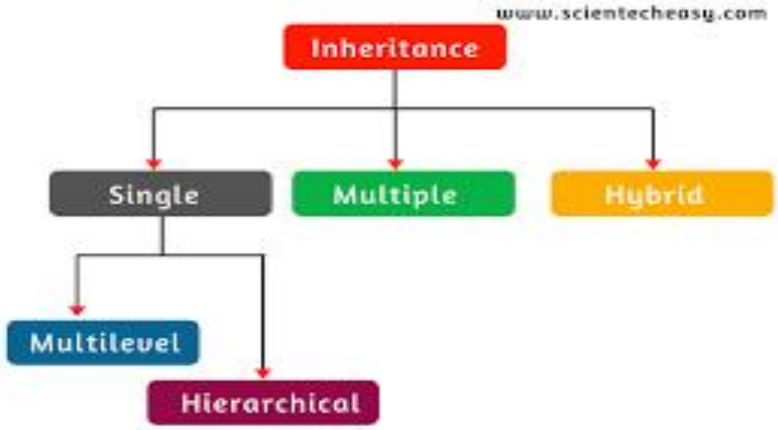


Fig: Classification of Inheritance in Java

```
// Parent class (Base class)
class Animal {
    void eat() {
        System.out.println("This animal eats food.");
    }
}

// Child class (Derived class)
class Dog extends Animal {
    void bark() {
```

```
        System.out.println("The dog barks.");
    }
}

// Main class to test inheritance
public class InheritanceExample {
    public static void main(String[] args) {
        Dog myDog = new Dog();

        // Calling method from the parent class
        myDog.eat();

        // Calling method from the child class
        myDog.bark();
    }
}
```

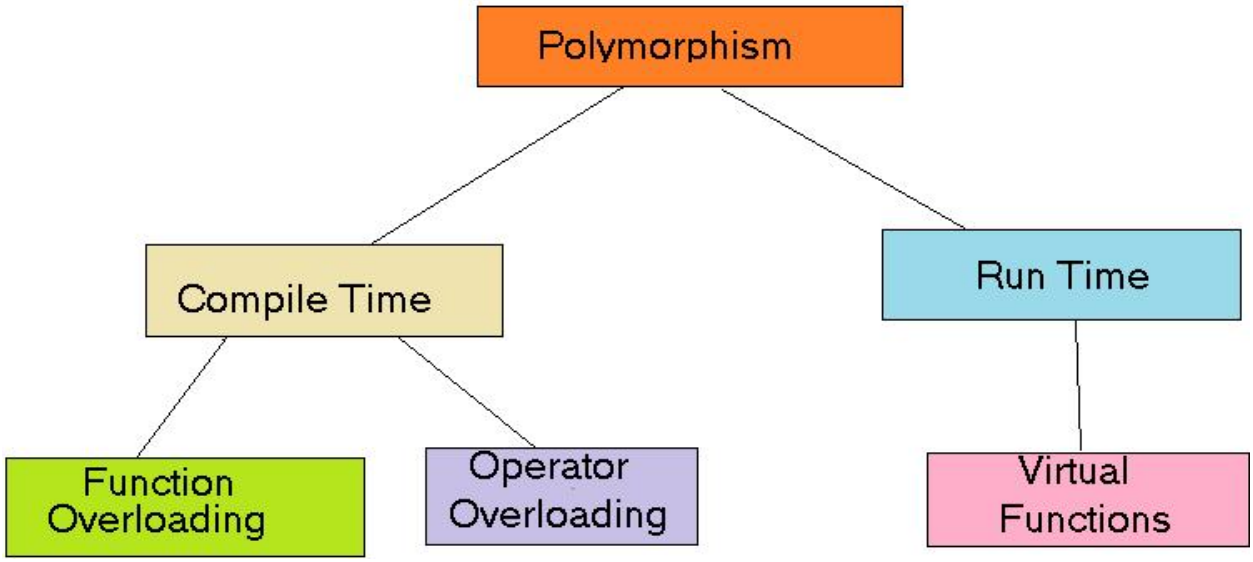
Output:

This animal eats food.
The dog barks.

- **Polymorphism**

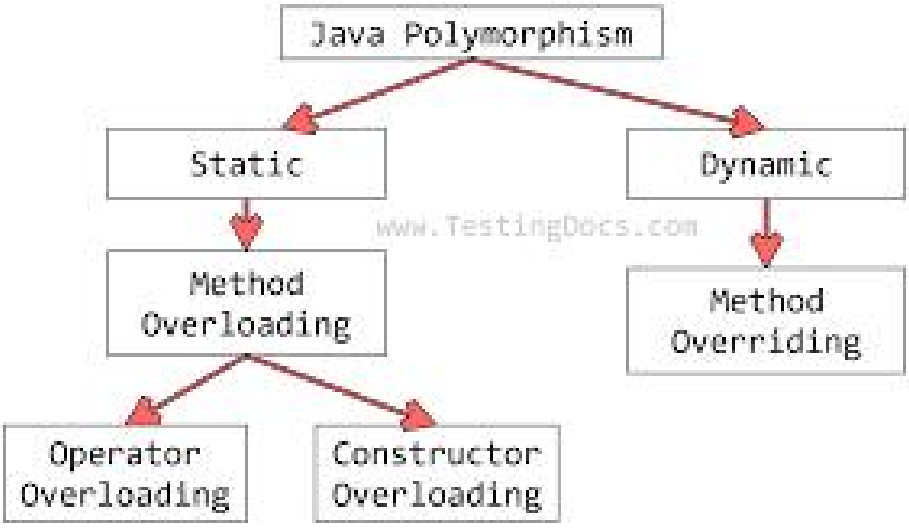
The ability of an object to behave differently depends on the context in which it is used. For example, a person can be a teacher in one context and a student in another.

There are two different types of Polymorphism in Java. They are: Compile-Time Polymorphism. Run-Time Polymorphism.



- **Encapsulation**

The process of hiding the variables or data of a class from other classes. This is also known as data-hiding.

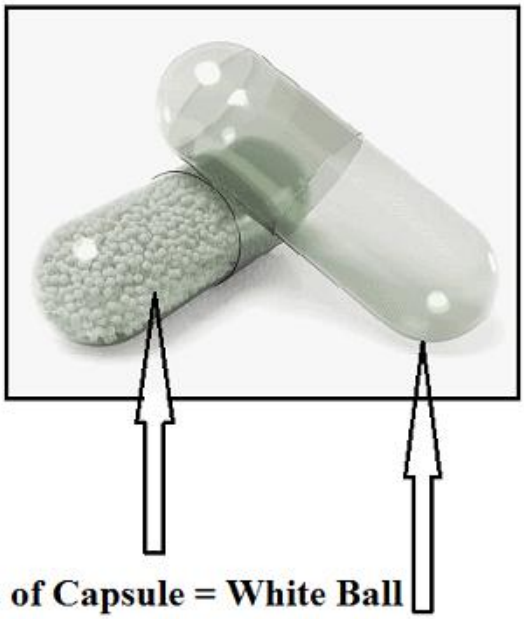


Object-oriented programming has three ways to implement encapsulation: member variable, function, and class.

- Member variable encapsulation. In member variable encapsulation—also called data member encapsulation—data members are class members. ...
- Function encapsulation. ...
- Class encapsulation.

```
public class Student{

 private int age;

 public void setAge(int studentAge){
       this.age = studentAge;
     }

     public int getAge() {
       return this.age;
     }
}
```

Methods and variables are memebers of Class.



Members of Capsule = White Ball

Class = Single Unit

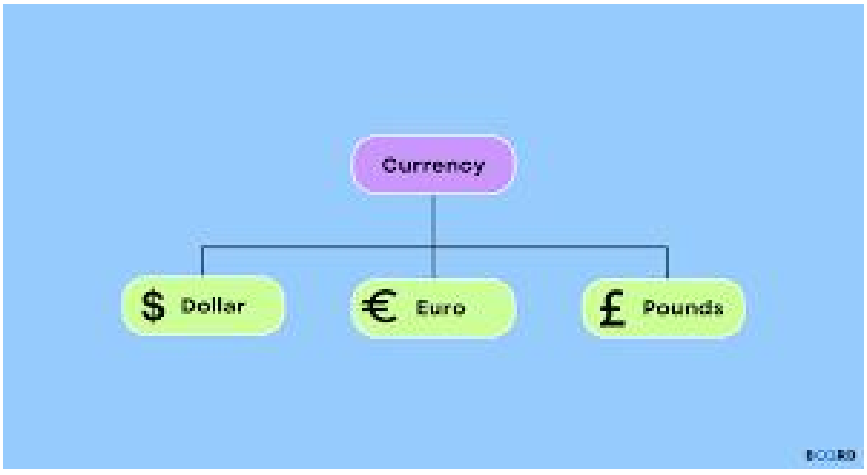
1 capsule = Single Unit

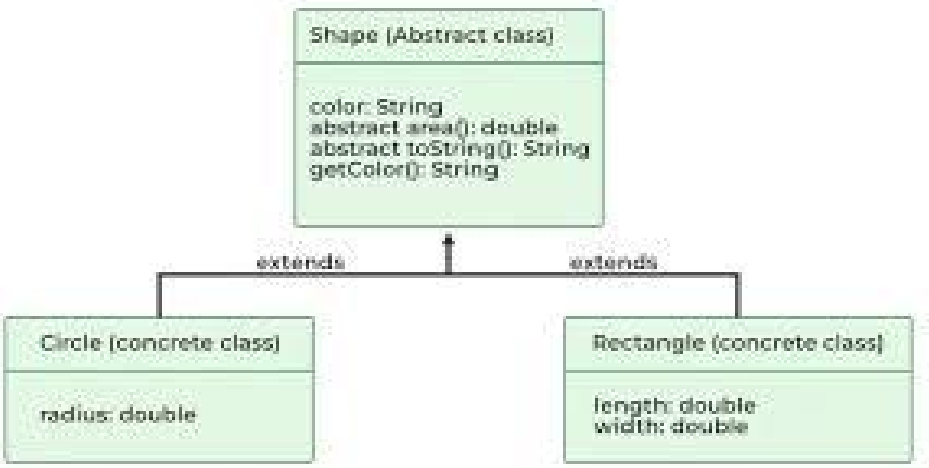
- **Abstraction**

Abstraction in Java is a fundamental Object-Oriented Programming (OOP) concept that focuses on hiding the complex implementation details and showing only the essential features of an object.

The process of hiding the implementation details and only showing functionality to the user. For example, pressing the accelerator increases the speed of a car.

There are two main types of abstraction in Java: interface and abstract class. Interface Abstraction is a way of defining the public methods that a particular type of object must implement without having to define how those methods work.





Here are some tools for developing, compiling, interpreting, and debugging in OOP:

- **Integrated development environments (IDEs) and code editors**

These tools provide features like debugging, testing, syntax highlighting, and code completion. Some popular IDEs and code editors for OOP languages include:

  - Visual Studio
  - Eclipse
  - IntelliJ IDEA
  - NetBeans
  - PyCharm
  - VS Code
  - Sublime Text

- **JDK**

Contains all the tools needed to compile, debug, and run a program developed using the Java platform

- **Print statements**

Also known as "Print Debugging", this method inserts code lines into a program to output information to the console or log files

- **Pair debugging**

Involves working with another developer to debug code

- **Airbrake**

A cloud-based debugging tool that offers a bug-reporting solution for small and midsize businesses

- **Other debugging tools**

Includes:

- WinDbg (Windows Debugger)
- Visual Studio Debugger
- IntelliJ IDEA Debugger
- PyCharm Debugger

Some tips for debugging efficiently include:

- Reproducing the bug before changing code
- Understanding stack traces
- Writing a test case that reproduces the bug
- Knowing error codes
- Using Google, Bing, or DuckDuckGo

Unified Modeling Language (UML):

Unified Modeling Language (UML) is a visual modeling language that helps software developers visualize, construct, and document software systems. It's an industry standard that provides a standard way to write blueprints for systems, including business processes, programming language statements, and database schemas.

UML is a collection of best engineering practices that help programmers and application architects make blueprints for projects. It's a "dictionary" of symbols with defined meanings that helps represent complex issues, processes, and systems. UML can help identify areas for optimization by depicting a system's structure and behavior.

UML can help improve software quality and lower business costs by eliminating the need for expensive tools.

Some common UML notations include:

- Activity (act)
- Class
- Component (cmp)
- Deployment (dep)
- Interaction (sd)
- Package (pkg)
- State Machine (stm)
- Use Case (uc)

Java Programming:

Java is a multi-platform, object-oriented, and network-centric language. It's known for its reliability, ease of use, and simplicity. One major advantage of Java is its portability, meaning that code written for a Java program on one device can be easily moved to another.

Java Syntax:

ava syntax is the set of rules and conventions that define how to write and interpret Java programs. It includes guidelines for using keywords, operators, data types, control structures, and other components to create valid Java programs.

Java syntax is important for developing robust Java applications because it provides a consistent framework for writing efficient and error-free code. Some key elements of Java syntax include:

- **Classes**

All code belongs to classes, and all values are objects, with the exception of primitive data types.

- **Access modifiers**

There are four access modifiers in Java: public, protected, default, and private.

- **Curly braces**

Curly braces ({}) are used to define blocks of code. The opening curly brace marks the beginning of a block, and the closing curly brace marks the end.

- **Variables**

Variables are used to name, store, and reference different types of data. Primitive data types include int, double, boolean, and char.

Java's syntax and structure are designed to encourage best practices in coding, making it a good language for beginners.

```
public void processData() {
    do {
        int data = getData();

        if (data < 0)
            performOperation1(data);
        else
            performOperation2(data);
    } while (hasMoreData());
}
```

Java is a programming language used for a wide variety of tasks, including:

- **Web applications**: Java is a popular choice for coding web applications.
- **Mobile applications**: Java is used to build and run mobile applications.

- **Cloud applications**: Java is a good choice for building cloud applications due to its performance, scalability, and reliability.
- **Enterprise software**: Java is used to create enterprise software.
- **Internet of Things (IoT) devices**: Java is used to support IoT devices.
- **Artificial intelligence (AI)**: Java is a popular choice for AI development, especially for large-scale projects.
- **Big data**: Java is used for big data applications.
- **Gaming**: Java is used for gaming.
- **Chatbots and marketing tools**: Java is used to develop chatbots and other marketing tools.

Java Operators

Operators are used to perform operations on variables and values.

In the example below, we use the + **operator** to add together two values:

ExampleGet your own Java Server

```
int x = 100 + 50;
```

Try it Yourself »

Although the + operator is often used to add together two values, like in the example above, it can also be used to add together a variable and a value, or a variable and another variable:

Example

```
int sum1 = 100 + 50;        // 150 (100 + 50)
int sum2 = sum1 + 250;      // 400 (150 + 250)
int sum3 = sum2 + sum2;     // 800 (400 + 400)
```

Java divides the operators into the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Bitwise operators

Arithmetic Operators

Arithmetic operators are used to perform common mathematical operations.

Operator	Name	Description	Example	Try it
+	Addition	Adds together two values	x + y	Try it »
-	Subtraction	Subtracts one value from another	x - y	Try it »
*	Multiplication	Multiplies two values	x * y	Try it »
/	Division	Divides one value by another	x / y	Try it »
%	Modulus	Returns the division remainder	x % y	Try it »
++	Increment	Increases the value of a variable by 1	++x	Try it »
--	Decrement	Decreases the value of a variable by 1	--x	Try it »

Java Assignment Operators

Assignment operators are used to assign values to variables.

In the example below, we use the **assignment** operator (=) to assign the value **10** to a variable called **x**:

Example

```
int x = 10;
```

The **addition assignment** operator (+=) adds a value to a variable:

Example

```
int x = 10;
x += 5;
```

Try it Yourself »

A list of all assignment operators:

Operator	Example	Same As	Try i

=	x = 5	x = 5	Try i
+=	x += 3	x = x + 3	Try i
-=	x -= 3	x = x - 3	Try i
*=	x *= 3	x = x * 3	Try i
/=	x /= 3	x = x / 3	Try i
%=	x %= 3	x = x % 3	Try i
&=	x &= 3	x = x & 3	Try i
|=	x |= 3	x = x | 3	Try i
^=	x ^= 3	x = x ^ 3	Try i
>>=	x >>= 3	x = x >> 3	Try i
<<=	x <<= 3	x = x << 3	Try i

Java Comparison Operators

Comparison operators are used to compare two values (or variables). This is important in programming, because it helps us to find answers and make decisions.

The return value of a comparison is either true or false. These values are known as *Boolean values*, and you will learn more about them in the Booleans and If..Else chapter.

In the following example, we use the **greater than** operator (>) to find out if 5 is greater than 3:

Example

```
int x = 5;
int y = 3;
System.out.println(x > y); // returns true, because 5 is higher than 3
```

Try it Yourself »

Operator	Name	Example	Try i
==	Equal to	x == y	Try i
!=	Not equal	x != y	Try i
>	Greater than	x > y	Try i
<	Less than	x < y	Try i
>=	Greater than or equal to	x >= y	Try i
<=	Less than or equal to	x <= y	Try i

Java Logical Operators

You can also test for true or false values with logical operators.

Logical operators are used to determine the logic between variables or values:

Operator	Name	Description	Example	Try
&&	Logical and	Returns true if both statements are true	x < 5 && x < 10	Try
||	Logical or	Returns true if one of the statements is true	x < 5 || x < 4	Try
!	Logical not	Reverse the result, returns false if the result is true	!(x < 5 && x < 10)	

Control Flow Constructs in java programming:

A control-flow construct refers to a programming feature that allows the programmer to connect different sequences of code, or basic blocks, in a useful way. Common control-flow constructs include if-then-else statements, loops, and case statements.

For controlling the flow of a program, the Java programming language has three loop constructs, a flexible if - else statement, a switch statement, exception-handling statements, and branching statements.

Objects and Classes Programming:

- **Classes**

A class is a user-defined data type that acts as a blueprint for objects. It defines the properties and behaviors of objects, including the range of valid values and default values for those properties.

- **Objects**

An object is a specific instance of a class that has all the extra information filled in, such as the values for the properties and what the methods will return. Objects can correspond to real-world objects or an abstract entity.

Here are some examples of how classes and objects work in programming:

- **Creating a person**

If you want to use a person in your program, you can create a class called "person" that describes what a person looks like and what they can do. Then, you can create an object of the type "person" to use in your program.

- **Creating a door**

You can create a door class that contains all the methods a door can perform and all the attributes a door might have, such as height, width, color, and whether it's closed or locked. Then, you can create objects of the door class to represent specific doors, like a front door or a back door.

Window Toolkit in Java Programming:

Java's Abstract Windowing Toolkit (AWT) provides support for programs that use Graphical User Interfaces (GUIs), rather than simply communicating with the user via the keyboard or via files. The AWT includes classes for commonly used objects like windows, labels, buttons, and checkboxes