

Object Oriented Programming using JAVA

Course Code	22CS33 / 22IS33	Course type	IPCC	Credits L-T-P	3 - 0- 1
Hours/week: L-T-P	3 - 0 - 2			Total credits	4
Total Contact Hours	L = 40Hrs; T = 0Hrs; P = 20Hrs Total = 60Hrs			CIE Marks	100
Flipped Classes content	10 Hours			SEE Marks	100

Required Knowledge of: Procedure Oriented Programming Languages

Course Learning Objectives

- | | |
|----|---|
| 1. | To understand the fundamentals of object-oriented programming and String class in Java. |
| 2. | To demonstrate the object-oriented features such as encapsulation, inheritance and polymorphism to design and develop programs in Java. |
| 3. | To understand exception handling mechanism supported in Java. |
| 4. | To learn to use the data structures to organize data in the program using the collections framework in Java. |
| 5. | To understand the concept of Packages, Interfaces and Lambda expressions in Java. |

Course Outcome (COs)

Learning Levels:

Re - Remember; Un - Understand; Ap - Apply; An - Analysis; Ev - Evaluate; Cr - Create

At the end of the course, the student will be able to		Learning Level	PO(s)	PSO(s)
1.	Explain classes, objects, members of a class and relationships among them needed for a specific problem.	Un	1,2,3,9,10,12	1,3
2.	Apply OOP principles (encapsulation, inheritance, polymorphism etc.) and proper program structure to write application programs.	Ap	1,2,3,5,9,10,12	1,2,3
3.	Develop skills in writing programs using exception handling techniques.	Ap	1,2,3,5,9,10,12	1,2,3
4.	Make use of the type hierarchy in the Collections Framework and Lambda expressions.	Ap	1,3,9,10,12	1,3
5.	Experiment with the concept of packages and interfaces.	Ap	1,3,9,10,12	1,3

Scheme of Continuous Internal Evaluation (CIE):

For integrated courses, a lab test also will be conducted at the end of the semester. The lab test **COMPULSORY**) will be part of the CIE. **No SEE for Lab.**

THEORY (60 marks)			LAB (40 marks)		Total
IA test 1	IA test 2	Assignment (OBA/Lab Project/ Industry assignment)	Conduction	Lab test	
25 marks	25 marks	10 marks	15 marks	25 marks	100 marks

IA Test:
1. No objective part in IA question paper
2. All questions descriptive

Conduct of Lab:
1. Conducting the experiment and journal: 5 marks
2. Calculations, results, graph, conclusion and Outcome: 5 marks
3. Viva voce: 5 marks

Lab test: (Batchwise with 15 students/batch)
1. Test will be conducted at the end of the semester
2. Timetable, Batch details and examiners will be declared by Exam section
3. Conducting the experiment and writing report: 5 marks
4. Calculations, results, graph and conclusion: 10 marks
5. Viva voce: 10 marks

Eligibility for SEE:
1. 40% and above (24 marks and above) in theory component
2. 40% and above (16 marks and above) in lab component
3. **Lab test is COMPULSORY**
4. Not eligible in any one of the two components will make the student **Not Eligible** for SEE

Scheme of Semester End Examination (SEE):

- | | |
|----|--|
| 1. | It will be conducted for 100 marks of 3 hours duration |
| 2. | Minimum marks required in SEE to pass: 40 out of 100 |
| 3. | Question paper contains two questions from each unit each carrying 20 marks. Students have to answer one full question from each unit. |

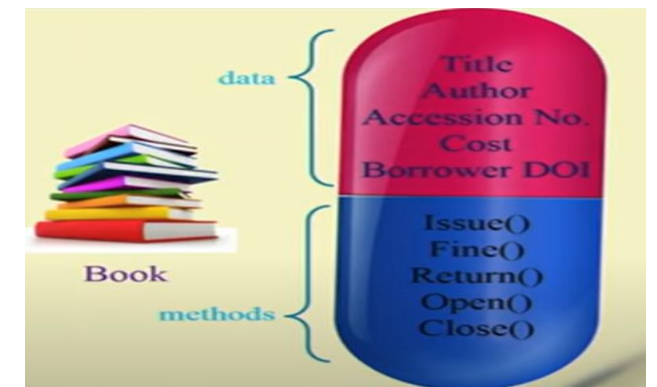
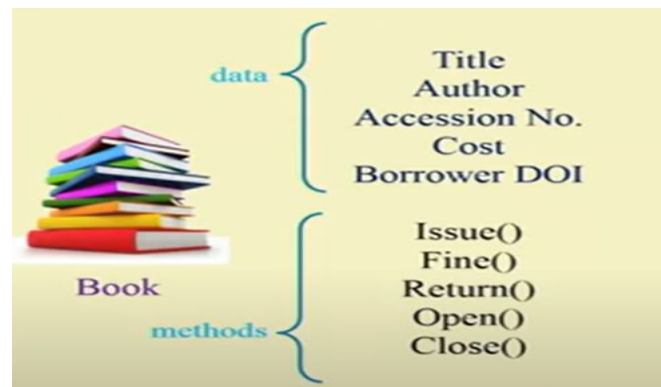
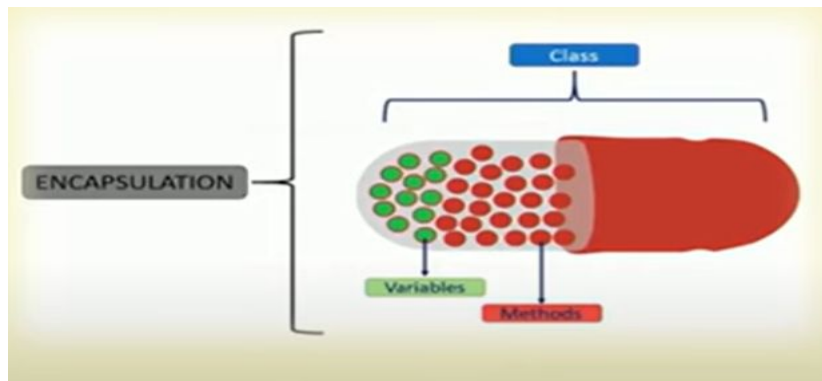
Unit – I Contact Hours = 8 Hours

- **OOP Paradigm:** The key attributes of object-oriented programming.
- **Java basics:** The Java language, JDK, arrays, multidimensional arrays, alternative array declaration, assigning array references, using the length member, the for-each loop.
- **Introducing classes and objects:** Class fundamentals, how objects are created, reference variables and assignment, String class

OOP Paradigm: The key attributes of object-oriented programming

- **Encapsulation:** Encapsulation is a programming mechanism that **binds together both code and data it manipulates**. When code and data are linked together an object is created. The data is not accessible to outside world and only those methods which are wrapped in class can access it.
 - **Data Hiding:** Encapsulation allows you to hide the internal details of an object, exposing only the necessary interface to the outside world.
 - achieved through access modifiers (public, private, protected) in many object-oriented languages.
 - **Information Hiding:** It provides a mechanism to control the visibility of object internals, preventing unintended interference and ensuring data integrity.
 - The meaning of Encapsulation, is to make sure that "sensitive" data is hidden from users.
- To achieve this, you must:
- declare class variables/attributes as **private**
 - provide public get and set methods to access and update the value of a **private** variable

Example:



Data Hiding

```
class Circle {  
    private:  
        double radius;  
    public:  
        void setRadius(double r) {  
            if (r >= 0) {  
                radius = r;  
            }  
        }  
        double getRadius() const {  
            return radius;  
        }  
};
```

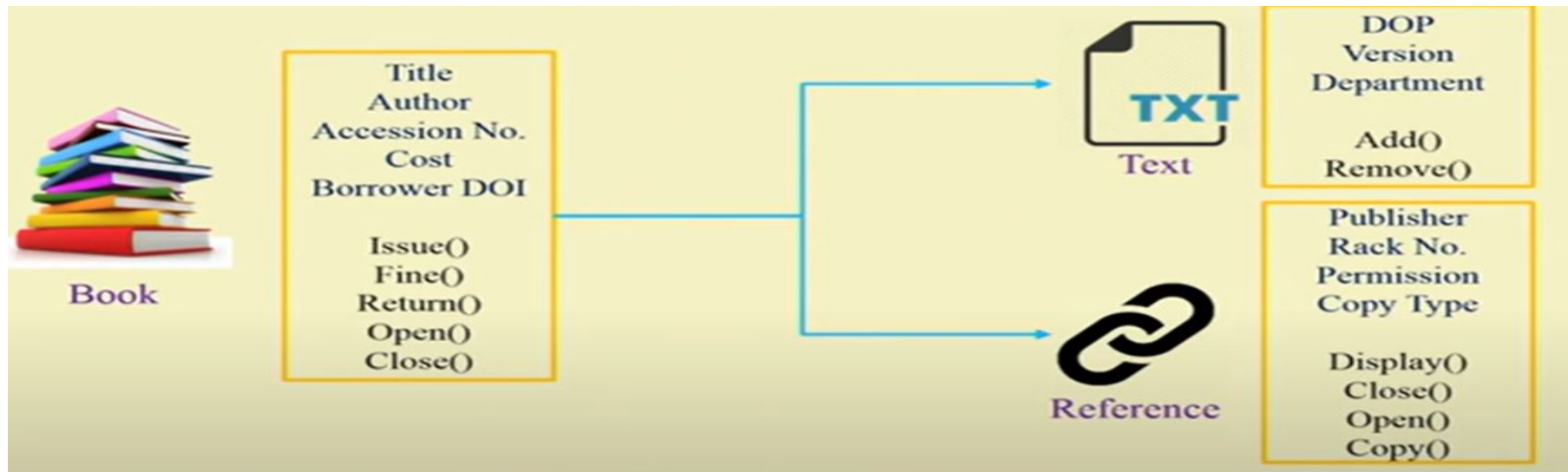

Information Hiding

```
class Circle {  
private:  
    double radius;  
public:  
    void setRadius(double r) {  
        if (r >= 0) {  
            radius = r;  
        }  
    }  
    double getRadius() const {  
        return radius;  
    }  
    double getArea() const {  
        return 3.14159265359 * radius * radius;  
    }  
};
```

Benefits of Encapsulation

- **Modularity:** Encapsulation promotes modularity, allowing you to break down complex systems into smaller, more manageable components (objects).
- **Security :** By restricting access to an object's data, you can control how data **Security** is modified and prevent unauthorized changes.
- **Ease of Maintenance:** Changes to an object's internal implementation can be made without affecting other parts of the program that use the object's interface.

- **Inheritance** : Inheritance is the process by which one object can acquire the properties of another object. In Java, it is possible to inherit attributes and methods from one class to another. We group the "inheritance concept" into two categories:
 - subclass (child) - the class that inherits from another class
 - superclass (parent) - the class being inherited from
 - To inherit a class ,use the **extend** keyword.



Polymorphism

- **Polymorphism:** Polymorphism means "many forms", or one Interface multiple methods, and it occurs when we have many classes that are related to each other by inheritance. [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.
(One interface multiple methods)
- **Definition:** Ability of different objects to respond to the same method or message in their own unique way.
- **Key Aspects:**
 - **Method Overriding:** Polymorphism is often achieved through method overriding, where a subclass provides a specific implementation for a method defined in its superclass.
 - **Interfaces:** Interfaces and abstract classes provide a way to achieve polymorphism by defining a common set of methods that multiple classes can implement.

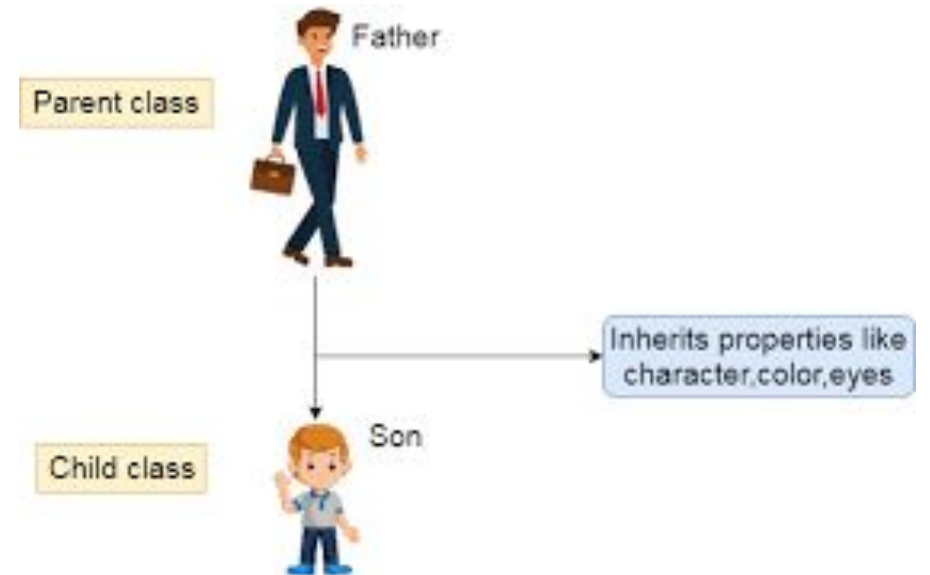
Benefits of Polymorphism

- **Code Reusability:** Polymorphism promotes code reusability by allowing different classes to share a common interface. This reduces code duplication.
- **Flexibility:** Polymorphism enables dynamic method invocation, allowing you to write code that can adapt to different object types at runtime.
- **Extensibility:** It makes it easier to add new classes that conform to an existing interface, enhancing the extensibility of a system.

```
class A {  
    void display() {  
        System.out.println("Inside class A");  
    }  
}  
  
class B extends A {  
    @Override  
    void display() {  
        System.out.println("Inside class B");  
    }  
}  
  
public class PolymorphismExample {  
    public static void main(String[] args) {  
        A obj1 = new A();  
        A obj2 = new B();  
        obj1.display(); // Calls the display method of class A  
        obj2.display(); // Calls the overridden display method of class B  
    }  
}
```

Inheritance

- **Definition:** Inheritance is a mechanism in OOP that allows a new class (subclass or derived class) to inherit properties and behaviors from an existing class (superclass or base class).
- The subclass can extend or override the inherited attributes and methods.



Key Aspects:

- **Superclass-Subclass Relationship:** Inheritance establishes an "is-a" relationship between the derived class and the base class.
- **Code Reuse:** Inheritance promotes code reuse by allowing a subclass to inherit the attributes and methods of a superclass, reducing redundancy.

Benefits of Inheritance:

- **Code Reusability:** Inheritance allows you to build new classes based on existing ones, reducing the need to write similar code from scratch.
- **Hierarchy:** It helps create class hierarchies that model real-world relationships, improving the organization of code.
- **Polymorphism:** Inheritance is closely related to polymorphism, enabling the use of common interfaces and dynamic method dispatch.

Inheritance(Sample)

```
class Employee{  
    float salary=40000;  
}  
class Programmer extends Employee{  
    int bonus=10000;  
    public static void main(String args[]){  
        Programmer p=new Programmer();  
        System.out.println("Programmer salary is:"+p.salary);  
        System.out.println("Bonus of Programmer is:"+p.bonus);  
    }  
}
```

Java basics: The Java language

- The Java was chosen for two primary reasons:

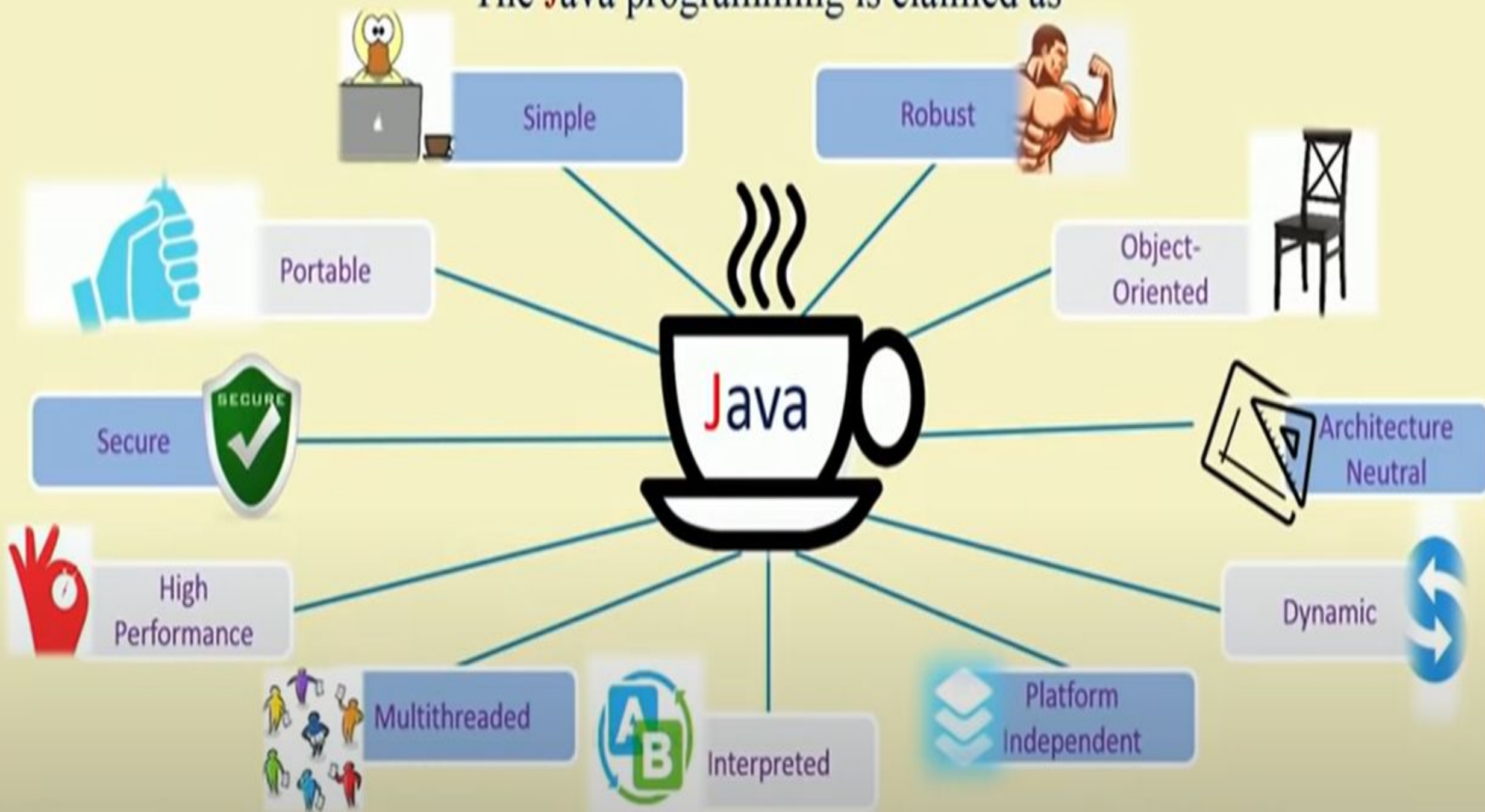
- 1st it is one of the worlds most widely used computer language.
- 2nd its features are designed and implemented in such a way that makes it easy to demonstrate the foundations of programming.

- The Origin of Java
- Java's Contribution to the Internet
- Java Applets
- Security
- Portability
- Java's Solution :The Bytecode
- The Evolution of Java.

The Origin of Java

Year	Development
1991	Java was conceived by James Gosling & others @ Sun Microsystems in 1991 . It was initially called “ oak ” by James Gosling. ”. Oak was renamed “Java”. Gosling and others began to work on portable ,cross-platform language that could run on variety of CPU’s under different platform. The WWW (World Wide Web) appeared on internet due to which Java programs were portable.
1993	Web applets(tiny programs) using the new language that could run on all the types of computer connected to Internet.
1994	The team developed a Web browser called “Hot Java” to locate & run applet program on Internet.
1995	Oak was renamed “Java” Java is an island of Indonesia where first coffee was produced.
1996	JDK(Java Development Kit) 1.0 released in January 23,1996.

The **J**ava programming is claimed as

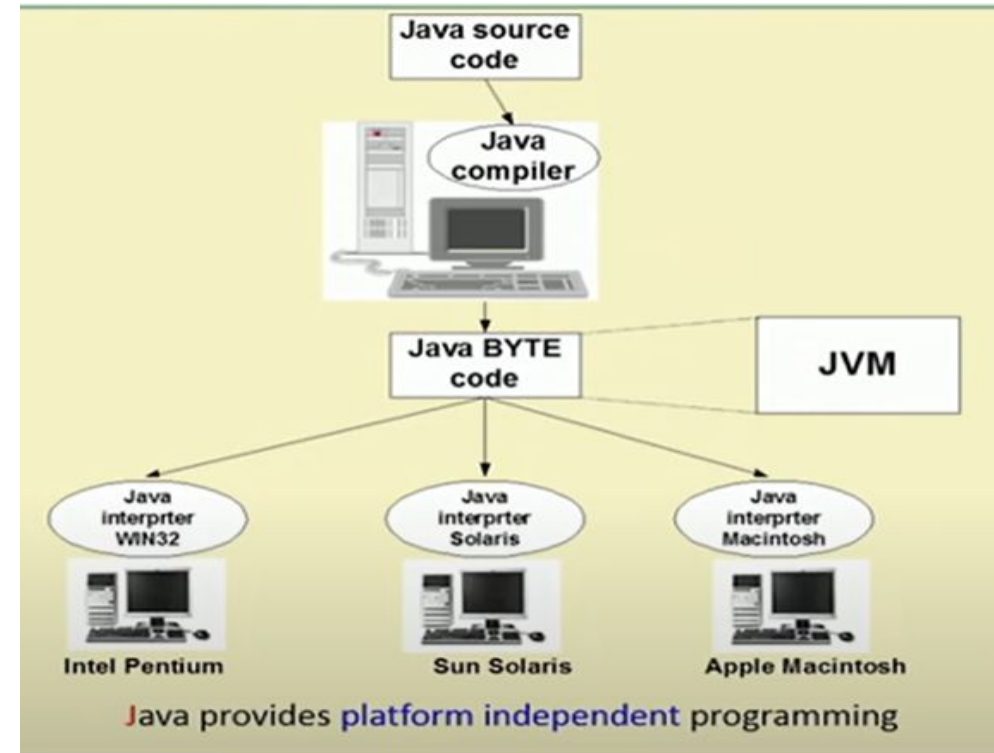
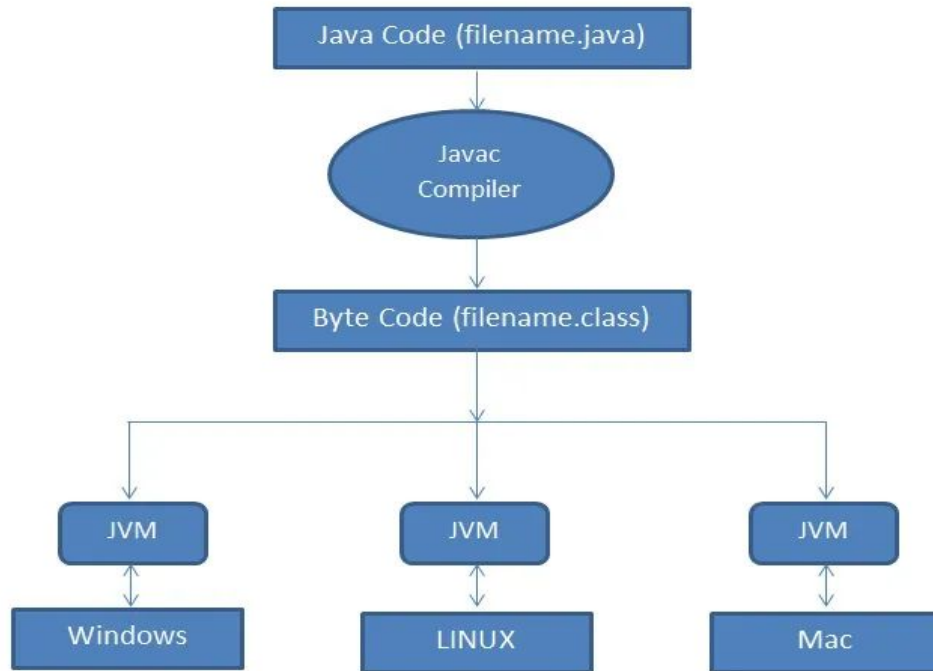


- **Java's Contribution to the Internet:** Java innovated a **new type of networked program called applet**, but it had issues with the Internet I,e. Portability and security.
- **Java Applets:** Applet is a special type of a program that is embedded in the webpage to generate a dynamic content .It runs inside the browser and works at client side.

Drawbacks :Applets presented a serious challenges interms of **Security** and **Portability**.

- **Security** : Every time when you download the “normal program ” it may contain virus like Torjan Horse or some harmful code which may cause damage to the system.
- ✓ Java achieved this protection by confining an applet to the Java execution environment and not allowing it to access to other parts of the computer.
- **Portability:** Portability is a major aspect of the Internet because many different types of computer and operating system are connected to it. When ever a program is downloaded it must be able to run on that system. Therefore some means of generating portable executable code was needed. Same mechanism that ensures security also helps create portability.

• Java's Solution :The Bytecode



- **As soon as a Java program is compiled bytecode is generated.**
- Bytecodes are non-runnable codes that rely on the availability of an interpreter, this is where JVM comes into play.
- Byte code is a machine-level language code that runs on the JVM.
- Bytecode adds portability to Java which resonates with the saying, “write once, read anywhere”.
- **JVM, loads, verifies and executes Java bytecode.** It is known as the interpreter or the core of Java programming language because it executes Java programming.

- **The Evolution of Java**: Java is a programming language that was first released in 1995 by Sun Microsystems (now owned by Oracle).
- Since its initial release, Java has undergone several updates and changes, with the most recent version being Java 15, which was released in September 2020.
- Some of the major updates and changes to Java over the years include:
 - **Java 1.1 (1997)**: Introduced the event-delegation model for event handling, inner classes, and JavaBeans.
 - **Java 2 (1998)**: Included the introduction of the Java Standard Edition (J2SE) and the Java Enterprise Edition (J2EE).
 - **Java 5 (2004)**: Introduced generics, annotations, and the for-each loop.
 - **Java 7 (2011)**: Introduced the try-with-resources statement and the diamond operator for generic types.
 - **Java 8 (2014)**: Included the introduction of lambda expressions and functional interfaces, as well as the Streams API.
 - **Java 9 (2017)**: Included the introduction of the Java Platform Module System (JPMS), as well as improvements to the Java shell (JShell) and the HTTP client.
 - **Java 10 (2018)**: Introduced local-variable type inference and additional improvements to the JPMS.
 - **Java 11 (2018)**: Java 11 was the first Long-term support release with a new release cycle, where Oracle releases a new version every six months, and supports each release for three years. This release introduced the removal of the JavaEE and CORBA modules, and the introduction of the HTTP client.
 - **Java 12 (2019), Java 13 (2019), Java 14 (2020) and Java 15 (2020)** : These releases have introduced new features and improvements such as new switch expressions, text blocks, and pattern matching for instanceof.

JDK: The Java Development Kit.

- **The JDK (Java Development Kit) is a software development kit that develops applications in Java**
- The JDK provides two primary programs.
- First is **javac** which is a Java compiler which is used to compile java programs, it takes .java file as input and produces bytecode.
- Following is the syntax of this command .It converts your source code into bytecode.

javac sample.java

- Second The **java command** is used to execute the bytecode of java. It takes byte code as input and runs it and produces the output. Following is the syntax of this command.

java sample

A First Example

A program in C to display message

```
#include <stdio.h>

int main()
{
    printf("Hello, World!");
    return 0;
}
```


A program in Java to display message

```
import java.lang.*;

class HelloWorldApp
{
    public static void main(String args[]){
        System.out.println("Hello, World!");
    }
}
```

Note: Both the languages are case sensitive

Aspects	C	Java
Paradigms	Procedural	Object-oriented
Platform Dependency	Dependent	Independent
Datatypes : union, structure	Supported	Not supported
Pre-processor directives	Supported (#include, #define)	Not supported
Header files	Supported	Use packages (import)
Storage class	Supported	Not supported

Aspects	C	Java
 Inheritance	No inheritance	Supported (Simple inheritance)
Pointers	Supported	No Pointers
Code translation	Compiled	Interpreted
Multi-threading and Interfaces	Not supported	Supported
Exception Handling	No exception handling	Supported
Database Connectivity	Not supported	Supported

A First Example:

- Every application begins with a class name and that class name must match the file name.
- Let us first create a Java file as Example.java that prints "Hello, World!" to the console:

```
public class Example {  
    public static void main(String[] args) {  
        System.out.println("Hello, World!");  
    }  
}
```

- Every line of the code must be included inside the class.
- The class begins with opening { and closing } braces. The elements between the two braces are members of class.

- `public static void main(String[] args) {`

- The line starts with the `public` keyword which indicates access modifier.

1.static: the keyword **static** allows **main()** to be executed independently of any object..

2.void: This is a return type. The main method **doesn't return any value**, so it is declared with void.

3.main: This is the name of the method. The main method is the entry point of the Java program. It's the method that gets executed when you run the program.

- If class is private ,which prevents the members from being used by code defined outside of its class.
- Main method has one parameter i.e `String[] args`, which declares the parameter named args. This is an array of objects of type `String`. i.e args receives any command line arguments present when the program is executed.

- `{`: This symbol is an opening curly brace, marking the beginning of the main method block. All the code related to the main method is enclosed within these curly braces.
- `System.out.println("Hello, World!");`
- The line outputs the string “Hello , World!” followed by new line on the screen.
- `System` – is a predefined class that provides access to the system
- `out`– is an output stream that is connected to the console.
- `println`—displays the string.

- `}`: This closing curly brace marks the end of the main method block.
- `}`: This closing curly brace marks the end of the Example class block.

Converting Gallons to Liters

```
public class GallonsToLiters {  
    public static void main(String[] args) {  
        double gallons = 5;  
        double liters = gallons * 3.78541;  
        System.out.println(gallons + " gallons is equal to " + liters + "  
liters.");  
    }  
}
```

Explanation: The program begins by declaring a variable **gallons** and assigning a value of 5 to it .

- The program then performs the conversion by multiplying the number of gallons by the conversion factor (3.78541) and storing the result in the variable **liters**..
- Finally, the program prints the result of the conversion to the console using the **System.out.println** method .

WAP in Java to find the largest of 3 numbers

```
public class FindLargest {  
    public static void main(String[] args) {  
        int num1 = 10, num2 = 20, num3 = 30;  
        int largest = num1;  
        if (num2 > largest) {  
            largest = num2;  
        }  
        if (num3 > largest) {  
            largest = num3;  
        }  
        System.out.println("The largest number is: " + largest);  
    }  
}
```

- The program begins by declaring three variables **num1**, **num2**, **num3** and assigning values 10, 20, 30 to them respectively.
- The program then declares a variable **largest** and assigns the value of **num1** to it.
- The program then uses an **if-else** statement to check if **num2** is greater than **largest**. If it is, then the value of **num2** is assigned to **largest**.
- The program then uses another **if-else** statement to check if **num3** is greater than **largest**. If it is, then the value of **num3** is assigned to **largest**.
- Finally, the program prints the largest number using the **System.out.println** method.
- When you run this program, it will display the largest number among num1, num2 and num3 which are assigned in the program.

Find the largest of 3 numbers using scanner class

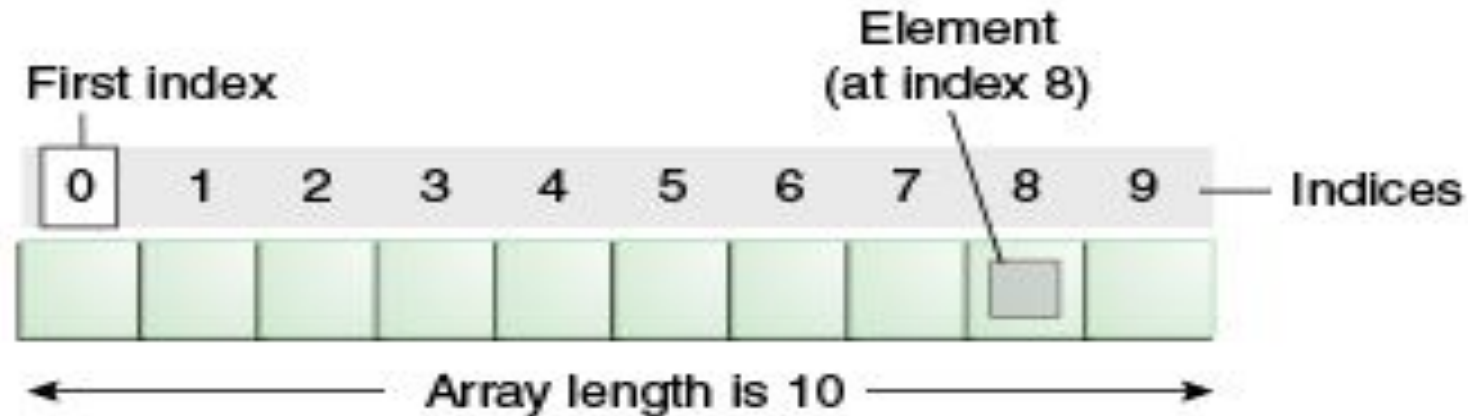
```
public class Main {  
    public static void main(String[] args) {  
        Scanner input = new Scanner(System.in);  
        System.out.print("Enter the first number: ");  
        int num1 = input.nextInt();  
        System.out.print("Enter the second number: ");  
        int num2 = input.nextInt();  
        System.out.print("Enter the third number: ");  
        int num3 = input.nextInt();  
        input.close();  
  
        int largest = num1;  
        if (num2 > largest) {  
            largest = num2;  
        }  
        if (num3 > largest) {  
            largest = num3;  
        }  
  
        System.out.println("The largest number is: " + largest);  
    }  
}
```

In this program, we are using a scanner object to take input from the user for three numbers. Then using if-else statement to find the largest of three numbers and print the largest number.

```
public class Main {  
    public static void main(String[] args) {  
        int num1 = 10, num2 = 20, num3 = 30;  
  
        if (num1 >= num2 && num1 >= num3) {  
            System.out.println(num1 + " is the largest number.");  
        } else if (num2 >= num1 && num2 >= num3) {  
            System.out.println(num2 + " is the largest number.");  
        } else {  
            System.out.println(num3 + " is the largest number.");  
        }  
    }  
}
```


Arrays

- In Java, **an array is an object which is a collection of elements of the same data type.** The elements of an array are stored in contiguous memory locations, and each element can be accessed by its index.
- Array in Java is index-based, the first element of the array is stored at the 0th index, 2nd element is stored on 1st index and so on. We can store only a fixed set of elements in a Java array.
- Like C/C++, we can also create single-dimensional or multidimensional arrays in Java.



Advantages

- **Code Optimization:** It makes the code optimized, we can retrieve or sort the data efficiently.
- **Random access:** We can get any data located at an index position.

Disadvantages

- **Size Limit:** We can store only the fixed size of elements in the array. It doesn't grow its size at runtime. To solve this problem, a collection **framework** is used in Java which grows automatically.

One Dimensional Array

Syntax to Declare an Array in Java

```
datatype arrayName[] = new datatype[size];
```

```
dataType[] arr; (or)
```

```
dataType []arr; (or)
```

```
dataType arr[];
```

- **datatype:** is the type of the elements that we want to enter in the [array](#), like int, [float](#), [double](#), etc.
- **arrayName:** is an identifier.
- **new:** is a keyword that creates an instance in the memory.
- **size:** is the length of the array.

// Declaring an array example

```
int[] myArray;
```

Instantiation of an Array in Java

```
arrayRefVar=new datatype[size];
```

```
// Initializing an array
```

```
myArray = new int[5];
```

```
public class Main{  
    public static void main(String[] args) {  
  
        int myarray[]; //Declaring an array  
  
        myarray= new int[5]; //Initializing an array//  
  
        //Assign values to the array  
        myarray[0]=1;  
        myarray[1]=2;  
        myarray[2]=3;  
  
        //Accessing elements of an array  
        for(int i=0;i<3;i++)  
            System.out.println(myarray[i]);  
        }  
    }
```

Example of Java Array

/Java Program to illustrate how to declare, instantiate, initialize
//and traverse the Java array.

```
class Main{  
    public static void main(String args[]){  
        int a[]=new int[5];//declaration and instantiation  
        a[0]=10;//initialization  
        a[1]=20;  
        a[2]=70;  
        a[3]=40;  
        a[4]=50;  
        //traversing array  
        for(int i=0;i<a.length;i++) //length is the property of array  
            System.out.println(a[i]);  
    }  
}
```

//Let's create a program that takes a single-dimensional array as input

import java.util.Scanner;

public class scanne1 {

public static void main(String[] args) {

int n;

Scanner sc= new Scanner(System.in);

System.out.println("Enter how many elements you want to store");

n=sc.nextInt(); //The nextInt() method scans the next token of the input data as an “int”.

int[] array =new int[10];

System.out.println("Enter the elements of array");

for(int i=0;i<n;i++){

array[i]= sc.nextInt();

}

System.out.println("Array elements are");

for(int i=0;i<n;i++)

{

System.out.println(array[i]);

}

}

}

```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {
        int n;
        Scanner sc= new Scanner(System.in);
        System.out.println("Enter how many elements you want to store");
        n=sc.nextInt(); //The nextInt() method scans the next token of the input data as an "int".
        int[] array =new int[10];
        System.out.println("Enter the elements of array");
        for(int i=0;i<n;i++){
            array[i]= sc.nextInt();
        }
        System.out.println("Array elements are");
        for(int i=0;i<n;i++)
        {
            System.out.println(array[i]);
        }
    }
}
```


Find the minimum and maximum value in an array

```
import java.util.Scanner;

public class MinMax {
    public static void main (String[] args) {
        int[] nums=new int[10];
        int min , max , n;
        Scanner sc = new Scanner(System.in);

        System.out.println("Enter the number of elements you want
        to store: ");

        n= sc.nextInt();

        System.out.println("Enter the elements of the array: ");
        for(int i=0;i<n ; i++)
            nums[i]=sc.nextInt();

        System.out.println("Array elements are: ");
        for(int i=0 ; i<n ; i++) {
            System.out.println(nums[i]); }
```

```
        min=max=nums[0];
        for (int i=1;i<n;i++) {
            if(nums[i]<min)
                min=nums[i];
            if(nums[i]>max)
                max=nums[i]; }
        System.out.println("Min And
        Max=" + min + "    " + max);  } }
```

```
import java.util.Scanner;
public class Main {
    public static void main (String[] args) {
        int[] nums=new int[10];
        int min , max , n;
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number of elements you want to store: ");
        n= sc.nextInt();
        System.out.println("Enter the elements of the array: ");
        for(int i=0;i<n ; i++)
            nums[i]=sc.nextInt();
        System.out.println("Array elements are: ");
        for(int i=0 ; i<n ; i++) {
            System.out.println(nums[i]); }
        min=max=nums[0];
        for (int i=1;i<n;i++) {
            if(nums[i]<min)
                min=nums[i];
            if(nums[i]>max)
                max=nums[i]; }
        System.out.println("Min value from array is =" + min + " and Max value from array is =" + max);
    } }
```

Find minimum and maximum to from array without using scanner function.

```
public class MinMaxArray {  
    public static void main(String[] args) {  
        int[] myArray = {1, 2, 3, 4, 5, 6, 7, 8, 9};  
        int max = myArray[0];  
        int min = myArray[0];  
        for (int i = 1; i < myArray.length; i++) {  
            if (myArray[i] > max) {  
                max = myArray[i];  
            }  
            if (myArray[i] < min) {  
                min = myArray[i];  
            }  
        }  
        System.out.println("The maximum value in the array is: " + max);  
        System.out.println("The minimum value in the array is: " + min);  
    }  
}
```

- Array boundaries are strictly enforced in Java.
- It is run time error to over run or under run an array.
- For example demonstration of ARRAY OVERRUN

```
class ArrayErr{  
    public static void main(String[] args) {  
        int[] sample =new int[10];  
        int i;  
        // generate an array over run  
        for(i=0;i<100;i++)  
            sample[i]= i;  
        }  
    }
```

As soon as **i** reaches **10**, an **ArrayIndexOutOfBoundsException** is generated and the program is terminated.

Sorting an Array using Bubble sort.

```
public class BubbleSort {  
    public static void main(String[] args) {  
        int[] myArray = {5, 3, 2, 8, 1, 9, 4};  
        bubbleSort(myArray);  
        for (int i = 0; i < myArray.length; i++) {  
            System.out.print(myArray[i] + " ");  
        }  
    }  
    public static void bubbleSort(int[] arr) {  
        int n = arr.length;  
        for (int i = 0; i < n - 1; i++) {  
            for (int j = 0; j < n - i - 1; j++) {  
                if (arr[j] > arr[j + 1]) {  
                    int temp = arr[j];  
                    arr[j] = arr[j + 1];  
                    arr[j + 1] = temp;  
                }  
            }  
        }  
    }  
}
```

```
public class Main {  
    public static void main(String[] args) {  
        int[] arr = {12,02,53,74,15};  
  
        System.out.println("Original Array:");  
        printArray(arr);  
  
        bubbleSort(arr);  
  
        System.out.println("Sorted Array:");  
        printArray(arr);  
    }  
}
```

```
public static void bubbleSort(int[] arr) {  
    int n = arr.length;  
    boolean flag;  
    for (int i = 0; i < n - 1; i++) {  
        flag = false;  
        for (int j = 0; j < n - i - 1; j++) {  
            if (arr[j] > arr[j + 1]) {  
                // Swap arr[j] and arr[j+1]  
                int temp = arr[j];  
                arr[j] = arr[j + 1];  
                arr[j + 1] = temp;  
                flag = true;  
            }  
        }  
        // If no two elements were swapped in inner loop, the  
        array is already sorted  
        if (!flag) {  
            break;  
        }  
    }  
}
```

```
public static void printArray(int[] arr) {  
    for (int value : arr) {  
        System.out.print(value + " ");  
    }  
    System.out.println();  
}  
}
```

```
class Main {  
    public static void main(String[] args) {  
        int[] myArray = {1, 2, 3, 4, 5, 6, 7, 8, 9};  
        int keyelem = 2;  
        int result = binarySearch(myArray, keyelem);  
        if (result == -1) {  
            System.out.println("Element not found");  
        } else {  
            System.out.println("Element found at index: " + result);  
        }  
    }  
}
```



```
public static int binarySearch(int[] arr, int target) {  
    int left = 0;  
    int right = arr.length - 1;  
    while (left <= right) {  
        int mid = left + (right - left) / 2;  
        if (arr[mid] == target) {  
            return mid;  
        }  
        if (arr[mid] < target) {  
            left = mid + 1;  
        } else {  
            right = mid - 1;  
        }  
    }  
    return -1;  
}
```

Multidimensional Array

- In Java, a multidimensional array is an array of arrays. It can be used to represent a table with rows and columns, or a matrix, or a grid.
- Example of how to declare and initialize a two-dimensional array in Java

```
int[][] myArray = new int[3][3];
```

- This creates a 2-dimensional array with 3 rows and 3 columns. Each element of the array is initialized with the default value for the int data type, which is 0.

Syntax to Declare and Initialize a two-dimensional array :

```
int[][] myArray = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
```

- In this example, the array named "myArray" is being declared as a 2-dimensional array of integers, and it is being initialized with the values 1, 2, 3, 4, 5, 6, 7, 8, 9.

Assigning array references

- In Java, arrays are objects that store multiple values of the same data type. You can create an array by specifying the type of data it will hold, followed by square brackets and the name of the array. **For example, to create an array of integers called "numbers", you would use the following syntax:**
 - `int[] numbers;`
- You can also initialize an array when you create it by specifying its size and the values it will hold. For example:
 - `int[] numbers = {1, 2, 3, 4, 5};`
- Once you have an array, you can access its elements by using the array name followed by the index of the element in square brackets. For example:
 - `System.out.println(numbers[0]);` // prints 1

- You can also assign new values to specific elements of an array using the same syntax:
 - `numbers[0] = 10;`
- You can also **create a reference to an array** using the same syntax.
 - `int[] numbers = {1, 2, 3, 4, 5};`
 - `int[] copy = numbers;`

Using the length member

length is an instance variable that contains the size of array.

Example:

```
int[ ] a={10,20,30,40,50};
```

```
int len = a.length //get the size of array.
```

```
system.out.println(len); //outputs 5
```

for-each loop

- In Java, the **for** loop and the **for-each** loop (also known as the enhanced for loop) are both used to iterate over the elements of an array or any other iterable object.
- The syntax for a for each loop (also known as a "for each" loop) is as follows:

```
for (type variable : collection) {  
    // code to be executed  
}
```

- ‘**type**’ is the data type of the items in the collection.
- ‘**variable**’ is the variable that will be assigned the value of each item in the collection, one at a time.
- ‘**collection**’ is the collection of items over which the loop will iterate. This could be an array, a list, a set, or any other type of collection.

- For example, the following for each loop iterates over an array of integers and prints each value:

```
int[ ] numbers = {1, 2, 3, 4, 5};  
    for (int num : numbers)  
    {  
        System.out.println(num);  
    }
```

//The output would be

```
1  
2  
3  
4  
5
```

- It's important to note that the for-each loop is used for only iterating over elements of an array or collections, it can't be used to iterate over the range of integers or anything like that.

Q. Write a program to add sum of array using for-each loop.

```
public class ForEach {  
    public static void main(String[] args) {  
        int [] nums={1,2,3,4,5,6,7,8,9,10};  
        int sum=0;  
        //use for each loop to display and sum the value.  
        for(int x:nums){  
            System.out.println(x);  
            sum=sum+x;  
        }  
        System.out.println("Summation =" +sum);  
    }  
}
```


- Write a Java program display a pattern as shown below using a looping structure.

1

22

333

4444

55555

```
public class Main {  
    public static void main(String[] args) {  
        int rows = 5; // Number of rows in the pattern  
  
        for (int i = 1; i <= rows; i++) {  
            for (int j = 1; j <= i; j++) {  
                System.out.print(i);  
            }  
            System.out.println();  
        }  
    }  
}
```

- Define the process of building and running Java program. And write a Java program to accept a number and check whether it is prime or not.

```
import java.util.Scanner;
public class PrimeChecker {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int number = scanner.nextInt();

        if (isPrime(number)) {
            System.out.println(number + " is a prime number.");
        } else {
            System.out.println(number + " is not a prime number.");
        }
    }

    public static boolean isPrime(int number) {
        if (number <= 1) {
            return false;
        }
        for (int i = 2; i <= Math.sqrt(number); i++) {
            if (number % i == 0) {
                return false;
            }
        }
        return true;
    }
}
```

- Write a Java program display a pattern as shown below using a looping structure

Output:

0 1 2 3 4

5 6 7 8 9

10 11 12 13 14

15 16 17 18 19

Program to find row sum and column sum in 2D array .

- public class Main{
- public static void main(String[] args) {
- int a[][]={{1,2,3},{4,5,6},{7,8,9}};
- for(int i=0;i<3;i++) {
- int rsum=0,csum=0;
- for(int j=0;j<3;j++) {
- rsum+=a[i][j]; csum+=a[j][i];
- }
- System.out.println("rowsum= "+rsum+"\t colsum= "+csum); }
- System.out.println("\n");
- }
- }

```
class Main {  
  
    public static void main(String args[]) {  
        int twoD[][]= new int[4][5];  
        int i, j, k = 0;  
        for(i=0; i<4; i++)  
            for(j=0; j<5; j++) {  
                twoD[i][j] = k; k++;  
            }  
        for(i=0; i<4; i++) {  
            for(j=0; j<5; j++)  
                System.out.print(twoD[i][j] + " ");  
            System.out.println();  
        }  
    }  
}
```

Introducing classes and objects:

The General Form of a Class

```
class classname {  
    // declare instance variables  
    type var1;  
    type var2;  
    // ...  
    type varN;  
  
    // declare methods  
    type method1(parameters) {  
        // body of method  
    }  
    type method2(parameters) {  
        // body of method  
    }  
    // ...  
    type methodN(parameters) {  
        // body of method  
    }  
}
```



```
class Vehicle{  
    int passengers;  
    int fulecap;  
    int mpg;  
}
```

```
class SimpleIntrest{  
  
    double p,t,r;  
    double calcSI() {  
        return (p*t*r)/100;  
    }  
}
```

How objects are created

In Java, objects are created using the **new** keyword, followed by the constructor of the class.

The general syntax for creating an object is as follows:

ClassName objectName = new ClassName(parameters);

- **'className'** is the name of the class.
- **'objectName'** is the name of the object (variable) that refers to the created object.
- **'new ClassName(parameters)'** is a call to the constructor of the class, passing in any necessary parameters.

For example, using the Student class , we can create an object of the class as follows:

```
Student student1 = new Student("John",20,"Computer Science");
```

Accessing class members

Once an object is created, we can call its methods or access its fields using the dot (.) operator.

The general form of the dot operator is
Object.member

For example,

```
System.out.println(student1.getName());
```

```
/*program that uses the vehicle class*/
```

```
class Vehicle{
```

```
    int passengers;
```

```
    int fulecap;
```

```
    int mpg;
```

```
}
```

```
Class VehicleDemo{
```

```
public static void main(String[] args){
```

```
    Vehicle minivan= new Vehicle();
```

```
    int range;
```

```
    minivan.passangers=7;
```

```
    minivan.fulecap=16;
```

```
    minivan.mpg=21;
```

```
//compute the range assuming full tank of gas
```

```
range= minivan.fulecap*minivan.mpg;
```

```
System.out.println("Minivan can carry"+minivan.passangers+ "with a range of "+range);
```

```
class Calculator {  
    int x,y;  
  
    int Sum() {  
        return x+y;  
    }  
}  
  
public class Main{  
    public static void main(String[] args) {  
        Calculator calc = new Calculator();    calc.x = 10;  
  
        calc.y =20;  
        System.out.println("Sum is:"+calc.Sum());  
    }  
}
```

```
import java.util.Scanner;
class SI{
double p,t,r;
double calcSI(){
return (p*t*r)/100;
}
}
public class Main{
    public static void main(String[] args){
        double pi,ti,re, interest;
        Scanner sc = new Scanner(System.in);

        pi = sc.nextDouble();
        ti = sc.nextDouble();
        re = sc.nextDouble();
```

```
SI s = new SI();
    s.p = pi;
    s.t= ti;
    s.r = re;
    interest = s.calcSI();
    System.out.println("Interest is:"+interest);
}
}
```

```
import java.util.Scanner;
class SI{
    double calcSI(int p,int t , int r){
return (p*t*r)/100;
}
}
public class Main{
    public static void main(String[] args){

        double pi,ti,re, interest;

        SI s = new SI();
            interest = s.calcSI(100,10,5);
System.out.println("Interest is:"+interest);        }
}
```

Methods

- Methods are the subroutines that manipulates the data defined by the class and in many cases control access to that data.
- **main()** is reserved for the method that begins execution of the program.

- **The general form is as shown here:**

return-type name(parameter list)

{

//body of method

}

Reference variables and assignment

- When one primitive-type is assigned to another, i.e., the statements `int x=y` means that
 - **x** receives a copy of the value contained in **y**.
- Hence, after the assignment, both **x** and **y** will contain their own, independent copies of the value. Changing one does not affect the other.
- But, when one object variable is assigned to another, the situation is bit different because you are assigning references.
- This means that you are changing the object that the reference variable refers to, not making a copy of that object
- For example,

```
Box b1=new Box();
```

```
Box b2=b1;
```

-
- In this case **b1** and **b2** will both refer to the same object. Therefore, object can be changed through **b1** or **b2**.
- - `b1.width = 12;`
 - `System.out.println(b1.width);`
 - `System.out.println(b2.width);`
- After executing both of these `println()` statements, we get the **same value for both** the statements as 12.

STRINGS

- In Java, a string is defined as an object
- Constructing Strings: by using **new** keyword and calling a string constructor

For example:

```
String str1 = "Hello World"; //Using String literal
```

```
String str2 = new String("Hello World"); //Using new operator
```

```
char[] charArray = {'H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l', 'd'};
```

```
String str3 = new String(charArray);
```

Example:

```
class Main {  
    public static void main(String args[]) {  
        String str1 = new String("hello");  
        String str2 = new String(str1);  
        String str3 = "Well come to Java";  
        System.out.println(str1);  
        System.out.println( str2+"\n"+ str3);  
    }  
}
```

Once a String object is created, you can perform various operations on it such as:

- **Concatenation** using the "+" operator or the **concat() method**
- **Comparison** using the **equals() method** or the "==" operator
- Finding the **length** of a string using the **length() method**
- Extracting a **substring** using the **substring() method**
- Replacing part of a string using the **replace() method**
- **Converting to lowercase or uppercase** using the **toLowerCase() or toUpperCase() method**

etc.

- Few examples are:

```
String str1 = "Hello";
```

```
String str2 = "World";
```

```
String str3 = str1 + " " + str2; // "Hello World"
```

```
System.out.println(str1.length()); // 5
```

```
System.out.println(str3.substring(0, 5)); // "Hello"
```

```
System.out.println(str3.toUpperCase()); // "HELLO WORLD"
```

boolean equals(str)	Returns true if the invoking string contains the same character sequence as str
int length()	Returns the number of characters (size) in the string
char charAt(index)	Returns the character at the place specified by “index”
int compareTo(str)	Returns negative value if the invoking string is less than str, positive value if the invoking string is greater than str, and zero if the strings are equal
int indexOf(str)	Searches the invoking string for the substring specified by str and returns the index of the first match or -1 on failure
int lastIndexOf(str)	Searches the invoking string for the substring specified by str and returns the index of the last match or -1 on failure
String toUpperCase()	Returns a string in upper case letters
String toLowerCase()	Converts and returns a string in lower case letters
char[] toCharArray()	Returns a string in a new character array
String substring(int start, int end)	Returns a new string that contains a specified portion of the invoking string

Consider the below example

```
String str1 = "Java";
```

```
String str2 = "JAVA";
```

```
String str3 =JavaJava;
```

boolean equals(str)

- str1.equals(str2) returns false, as invoking string str1 does not have same character sequence as str2.

int length()

- str1.length() returns 4

char charAt(index)

- str1.charAt(2) returns character "v" as index starts from 0

int compareTo(str)

- str1.compareTo(str2) returns 32, as ASCII value of "a" is 32 times greater than "A"

int indexOf(str)

- str3.indexOf("Java") returns 0, as it returns the index of the first match

int lastIndexOf(str)

- str3.lastIndexOf("Java") returns 4, as it returns the index of the last match

Program on String Operations

```
public class StrOps {  
    public static void main(String[] args) {  
String str1="when it comes to web programming ,java is  
1";  
        String str2 =new String(str1);  
String str3="Java strings are powerfull";  
        int result,idx;  
        char ch;  
        System.out.println("length of str1 " + str1.length());  
//display str1 , one character at a time  
        for(int i=0;i<str1.length();i++)  
            System.out.print(str1.charAt(i));  
        System.out.println();  
        if(str1.equals(str2))  
            System.out.println("str1 equals str2");  
        else  
            System.out.println("str1 does not equals str2");  
        if(str1.equals(str3))  
            System.out.println("string1 equals string3");
```

```
        if(result== 0)  
            System.out.println("string1  
and 3 are equal");  
        else if(result<0)  
            System.out.println("string1  
is less than string3");  
        else  
            System.out.println("string1  
is greater than string3");  
        //assign a new string to str2  
        str2="One Two Three One";  
        idx=str2.indexOf("one");  
        System.out.println("Index of  
first occurance of One:" +idx);  
        idx=str2.lastIndexOf("one");  
        System.out.println("Index of  
first occurance of One:" +idx);  
    }  
}
```


Output:

- **Length of str1: 45**
- **when it comes to web programming ,java is 1**
- **str1 equals str2**
- **string1 does not equals string3**
- **string1 is greater than string3**
- **Index of first occurrence of One: 0**
- **Index of first occurrence of One: 14**

Arrays Of Strings

- Like any other data type strings can be assembled into arrays.

For example:

- `public class StringArrays {`
- `public static void main(String[] args) {`
- `String[] str={"This" , "is" , "a" , "test."};`
- `System.out.println("Original array");`
- `for (String s:strs)`
- `System.out.print(s + " ");`
- `System.out.println("\n");`
- `//change a string in the array`
- `str[1]="was";`
- `str[3]="test,too !";`
- `System.out.println("Modified array:");`
- `for (String s:strs)`
- `System.out.print(s + " "); } }`

OutPut:

Original array
This is a test.

Modified array:
This was a test,too !

Strings are Immutable

- Java, strings are immutable, meaning their value cannot be changed once they are created.
- Therefore, all the methods that are defined above don't alter the original String object
- This means that any operations that appear to change the value of a string, such as concatenation or replacement, will actually create a new string object with the modified value.

Previous Year Questions

- Explain the following terms with respect to string in JAVA.

charAt()

indexOf()

compareTo()

toCharArray()

toLowerCase()

3b. Describe the for-each loop with an example. Write a Java program to sum only the first six elements of array $a = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$ using for-each loop.

OR

```
public class SumFirstSixElements {  
    public static void main(String[] args) {  
        // Given array  
        int[] a = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};  
  
        // Variable to store the sum  
        int sum = 0;  
  
        // Using for-each loop to sum the first six elements  
        int count = 0;  
        for (int element : a) {  
            if (count < 6) {  
                sum += element;  
                count++;  
            } else {  
                break; // Break the loop after summing the first six elements  
            }  
        }  
  
        // Displaying the result  
        System.out.println("Sum of the first six elements: " + sum);  
    }  
}
```

Q. write a java program to sum only the first six elements of array a={1,2,3,4,5,6,7,8,9,10} using for each loop

1a. Explain the three key attributes of object-oriented principles. Describe the general form of a java program. Write a java program to add 5 numbers.

[2] [1] [1] [10]

1b. Explain any five methods of String. Write a java program to display the numbers from 1 to 9 in the form of a table. The table has 3 rows and 3 columns.

[3] [2] [2] [10]

OR

2a. Define a class. Describe the general form of a class. Write a program to create a class called "Vehicle" and create two objects from the Vehicle class.

[3] [1] [2] [10]

2b. Describe for-each style for loop with an example. Write a Java program to perform addition of two matrices of size 3*3.

[3] [1] [2] [10]

```
public class NumberTable {  
    public static void main(String[] args) {  
        // Constants for rows and columns  
        int rows = 3;  
        int columns = 3;  
  
        // Variable to track the number  
        int number = 1;  
  
        // Nested loops to print the table  
        for (int i = 0; i < rows; i++) {  
            for (int j = 0; j < columns; j++) {  
                // Print the current number and increment  
                System.out.print(number + "\t");  
                number++;  
            }  
            // Move to the next line after printing a row  
            System.out.println();  
        }  
    }  
}
```

Write a java program to display the numbers in the from 1 to 9 in the form of table. the table has 3 rows 3 columns.

OutPut

1	2	3
4	5	6
7	8	9

- a. Explain the three key attributes of object oriented principles. Describe the general form of a java program. Write a java program to convert 10 gallons to liters and display the result. There are approximately 3.7854 liters in a gallon.

(2) (1) (1) (10)

- b. Explain methods of String class that operate on strings. Write a java program to display twelve names, stored in a one-dimensional array, in a tabular form that has 3 rows and 4 columns.

(2) (2) (3) (10)

OR

- a. Describe the creation of a two dimensional array. Write a java program to sort elements of an array `num[]={99, -10, 100123, 18, -978, 5623, 463, -9, 287, 49}` using Bubble sort and print the sorted array.

(2) (1) (3) (10)

- b. Describe for-each style for-loop with an example. Write a Java program to perform multiplication of two matrices.

(2) (2) (1) (10)


```

public class NamesArray {
    public static void main(String[] args) {
        // 1D array with 12 names
        String[] names = {
            "Alice", "Bob", "Charlie", "David",
            "Eva", "Frank", "Grace", "Hank",
            "Ivy", "Jack", "Kelly", "Leo"
        };
        // Displaying the names in a 3x4 table
        int rowCount = 3;
        int colCount = 4;
        int index = 0; // Index to iterate through the names array
        for (int i = 0; i < rowCount; i++) {
            for (int j = 0; j < colCount; j++) {
                // Check if there are still names in the array
                if (index < names.length) {
                    System.out.print(names[index] + "\t");
                    index++;
                }
            }
            System.out.println(); // Move to the next line after printing a row
        }
    }
}

```

Write a java program to display 12 names stored in 1-d array in a tabular form that has 3 rows ,4 columns .

OutPut

Alice	Bob	Charlie	David
Eva	Frank	Grace	Hank
Ivy	Jack	Kelly	Leo

- What is the difference between a class and an object? How is class defined in JAVA? Write a Java program to define a class called myTriangle to model a triangle geometrical object with three sides. Include functions to:
 - Initialize the three sides of triangle.
 - Compute and return the area of the triangle.

```
import java.util.Scanner;

class MyTriangle {
    private double side1;
    private double side2;
    private double side3;

    // Constructor to initialize the sides of the triangle
    public MyTriangle(double s1, double s2, double s3) {
        side1 = s1;
        side2 = s2;
        side3 = s3;
    }

    // Function to compute and return the area of the triangle
    public double calculateArea() {
        // Using Heron's formula to calculate the area of the triangle
        double s = (side1 + side2 + side3) / 2.0;
        double area = Math.sqrt(s * (s - side1) * (s - side2) * (s - side3));
        return area;
    }
}

public class TriangleDemo {
```

TW2: Design a class by name myTriangle to model a triangle geometrical object with three sides. Include functions to:

1. Initialize the three sides of the triangle.
2. Determine the type of triangle represented by the three sides (Equilateral/ Isosceles/ Scalene triangle).
3. Compute and return the area of the triangle.

Note: When three sides are given we use the following formula:

$$s = (a+b+c) / 2;$$

$$\text{area} = \text{sqrt}(s*(s-a)*(s-b)*(s-c));$$

```
import static java.lang.System.exit;

import java.util.Scanner;

class Triangle {
    double a,b,c;
    void getSides(){
        Scanner in=new Scanner(System.in);
        System.out.println("Enter 3 sides of a triangle:");
        a=in.nextDouble();
        b=in.nextDouble();
        c=in.nextDouble();
    }
}
```

```
void checkTriangle() {  
    if ((a+b)>c && (b+c)>a && (a+c)>b) {  
        if (a==b && b==c && c==a)  
            System.out.println("Triangle is equilateral");  
        else if (a==b || b==c || c==a)  
            System.out.println("Triangle is isosceles");  
  
        else  
            System.out.println("Triangle is scalene");  
    }  
    else {  
        System.out.println("Triangle cannot be formed");  
        exit(0);  
    }  
}
```

```
double computeArea() {  
    double s=(a+b+c)/2;  
    double area=Math.sqrt(s*(s-a)*(s-b)*(s-c));  
    return area;  
}  
  
}
```

```
public class tw2 {  
    public static void main(String[] args) {  
        Triangle t=new Triangle();  
        t.getSides();  
        t.checkTriangle();  
        if((t.computeArea())!=0){  
            System.out.println("Area is "+t.computeArea());  
        }  
    }  
}
```


IA-1 QP with solutions

- “Strings are immutable in Java.” Explain with suitable example to support this statement.
- public class Main {
 - public static void main(String[] args) {
 - // Creating a string literal
 - String s1 = "Hello, ";
 -
 - // Concatenating strings creates a new string
 - String s2 = s1 + "World!";
 -
 - // Printing the original and modified strings
 - System.out.println("Original String: " + originalString);
 - System.out.println("Modified String: " + modifiedString);
 -
 - // The original string remains unchanged
 - System.out.println("After modification, Original String: " + originalString);
 - }
- }

Explain the concept of instance variable hiding in Java using an example. Discuss why it is important to be aware of this behaviour and how it can be mitigated.

- It is illegal in Java to declare two local variables with the same name inside the same or enclosing scopes
- Interestingly, you can have local variables, including formal parameters to methods, which overlap with the names of the class' instance variables
- When a local variable has the same name as an instance variable, the local variable hides the instance variable
- For example, here is another version of Box(), which uses width, height, and depth for parameter names and then uses this to access the instance variables by the same name

```
// Use this to resolve name-space collisions.  
Box(double width, double height, double depth) {  
    this.width = width;  
    this.height = height;  
    this.depth = depth;  
}
```

- Use this to overcome the instance variable hiding

Analyse the following code and find the number of objects in it

```
public class Strings_Java {  
    public static void main(String[] args) {  
        String s1 = "KLSGIT";  
        String s2 = s1;  
        String s3 = display();  
        String s4 = show();  
        System.out.println(s2);  
        System.out.println(s3);  
        System.out.println(s4);  
    }  
    static String display(){  
        String s4 = "GIT";  
        return s4.concat("KLS");  
    }  
    static String show(){  
        return new String();  
    }  
}
```

- S1=KLSGIT
- S4=GIT (in display scope)
- KLS
- S3=GITKLS
- S4= “ ” (in main scope)

Analyse the following program and write its output

```
public class Strings_Java {  
    public static void main(String[] args) {  
        String s1 = "KLSGIT";  
        String s2 = new String("KLSGIT");  
        if(s1==s2)  
            System.out.println("S1 and S2 are equal");  
        else  
            System.out.println("S1 and S2 are not equal");  
        char[] target = {'a','b'};  
        System.out.println(target);  
        char[] target1 = new char[5];  
        s1.getChars(3, 6, target1, 2);  
        System.out.println("The output is");  
        System.out.println(target1[2]);  
        s1.getChars(3, 6, target, 2);  
        System.out.println(target[2]);  
    }  
}
```

S1 and S2 are not equal

ab

The output is

G

Runtime exception (StringIndexOutOfBoundsException) / Error