Experiment No. 1

Aim of the Experiment: Write some simple programs in java such as

i) To find factorial of number

ii) To display first 50 prime numbers

iii) To find sum and average of N numbers

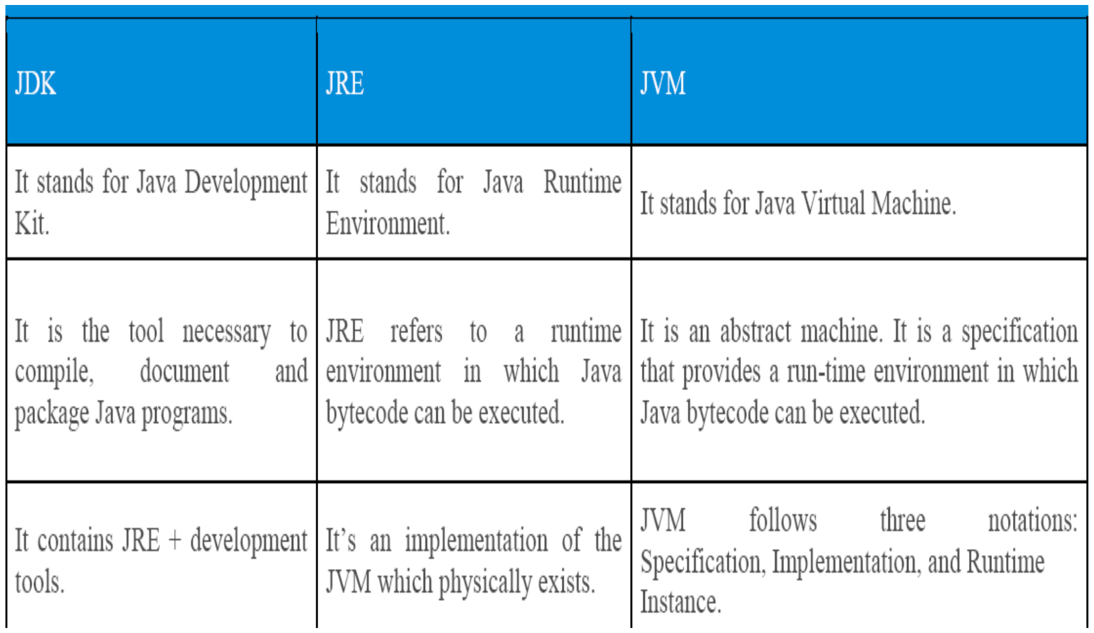
Objective: To write simple programs in Java.

Software used: Eclipse IDE 2018, JDK 1.8.0 is required

Course Outcome Addressed: CO1

Theory:

- JAVA was developed by James Gosling at Sun Microsystems Inc in the year 1991, later acquired by Oracle Corporation.
- It is a simple programming language. Java makes writing, compiling, and debugging programming easy. It helps to create reusable code and modular programs.
- Java is a class-based, object-oriented programming language and is designed to have as few implementation dependencies as possible.
- A general-purpose programming language made for developers to write once run anywhere (WORA) that is compiled Java code can run on all platforms that support Java.
- Oak is initial name of java.
- Java applications are compiled to byte code that can run on any Java Virtual Machine.



JDK	JRE	JVM
It stands for Java Development Kit.	It stands for Java Runtime Environment.	It stands for Java Virtual Machine.
It is the tool necessary to compile, document and package Java programs.	JRE refers to a runtime environment in which Java bytecode can be executed.	It is an abstract machine. It is a specification that provides a run-time environment in which Java bytecode can be executed.
It contains JRE + development tools.	It's an implementation of the JVM which physically exists.	JVM follows three notations: Specification, Implementation, and Runtime Instance.

Integrated Development Environments

An integrated development environment, or IDE, is a graphical user interface program that integrates all these aspects of programming and probably others (such as a debugger, a visual interface builder, and project management). A command-line environment is just a collection of commands that can be typed in to edit files, compile source code, and run programs

There are sophisticated IDEs for Java programming that are available:

Eclipse IDE - An increasingly popular professional development environment that supports Java development, among other things. Eclipse is itself written in Java. It is available from http://www.eclipse.org/.

NetBeans IDE - A pure Java IDE that should run on any system with Java 1.7 or later. NetBeans is a free, "open source" program. It is essentially the open source version of the next IDE. It can be downloaded from www.netbeans.org.

IntelliJ- IntelliJ IDEA is an integrated development environment written in Java for developing computer software written in Java, Kotlin, Groovy, and other JAR based languages.

public static void main(String args[]) in Java.

- main() in Java is the entry point for any Java program. It is always written as public static void main(String[] args).

- public: Public is an access modifier, which is used to specify who can access this method. Public means that this Method will be accessible by any Class.

- static: It is a keyword in java which identifies it is class-based. main() is made static in Java so that it can be accessed without creating the instance of a Class. In case, main is not made static then the compiler will throw an error as main() is called by the JVM before any objects are made and only static methods can be directly invoked via the class.

- void: It is the return type of the method. Void defines the method which will not return any value.

- main: It is the name of the method which is searched by JVM as a starting point for an application with a particular signature only. It is the method where the main execution occurs.

- String args[]: It is the parameter passed to the main method.

Algorithm to Find the Factorial of a Number:

In this program, we will find the factorial of a number using recursion with user-defined values. Here, we will ask the user to enter a value and then we will calculate the factorial by calling the function recursively.

1. Start
2. Create an instance of the Scanner Class.
3. Declare a variable.
4. Ask the user to initialize the variable.
5. Declare a loop variable and another variable to store the factorial of the number.
6. Initialize both the variables to 1.
7. Use a while loop to calculate the factorial.
8. Run the loop till the loop variable is less than or equal to the number.
9. Update the factorial in each iteration.
10. Increment the loop variable in each iteration.
11. Print the factorial of the number.
12. Stop.

Conclusion:

References:

Herbert Schildt , “Java : The Complete Reference” Tata McGraw-Hill (7th Edition).

Questions:

1. What are the differences between C++ and Java?
2. Explain features of Java?
3. Why Java is platform independent?
4. Why main method declared as static in Java?
5. Explain control statements in Java?

Experiment No. 2

Aim of the Experiment: Write a program in Java to implement a Calculator with simple arithmetic operations such as add, subtract, multiply, divide, factorial etc. using switch case and other simple java statements.

Objective: To implement a Calculator with arithmetic operations and using switch case in Java.

Software used: Eclipse IDE 2018, JDK 1.8.0 is required

Course Outcome Addressed: CO1

Theory:

Java Operators:

Operators are used to perform operations on variables and values. In Java, an expression has two parts: operand and operator. The variables or constants that operators act upon are called **operands**.

An operator in java is a symbol or a keyword that tells the compiler to perform a specific mathematical or logical operations.

They are used in programs to manipulate data and variables. They generally form mathematical or logical expressions. An expression is a combination of variables, constants, and operators.

For example, an expression is x+5. Here, the operand x is a variable, operand 5 is a constant, and + is an operator that acts on these two operands and produces the desired result.

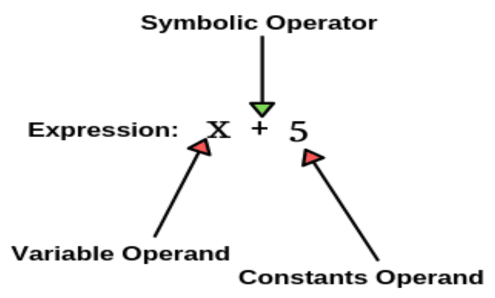


Figure: Operator and Operands

Types of Operators in Java:

There are three types of operators in java based on the number of operands used to perform an operation.

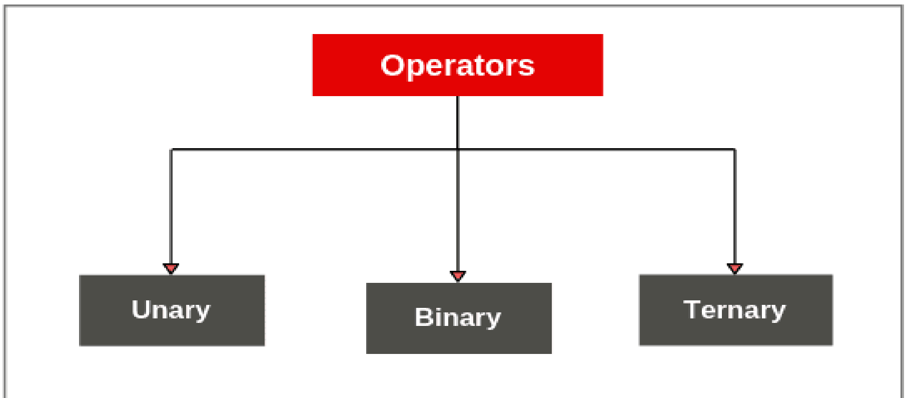


Fig: Classification of java operators based on number of operands

1. **Unary operator:** The operator that acts on a single operand is called unary operator. A unary operator uses a single variable.

2. **Binary operator:** The operator that acts on two operands is called binary operator. A binary operator uses two variables.

3. Ternary operator: The operator that acts on three operands is called ternary operator. A ternary operator uses three variables.
Based on the notation used, Java operator has been divided into two categories: Symbolic and named.

If a symbol like +, -, *, etc is used as an operator, it is called symbolic operator. If a keyword is used as an operator, it is called named operator.

Classification of Operators based on Symbols and Named in Java:

The Java operators can be classified on the basis of symbols and named. They are as follows:

1. Symbolic operator in Java
   - Arithmetic operators: +, -, *, /, etc.
   - Relational operators: <, >, <=, >=, = =, !=.
   - Logical operators: &&, ||, !.
   - Assignment operators: =,
   - Increment and decrement operators: + +, – –
   - Conditional operators: ?:
   - Bitwise operators: &, !, ^, ~, <<, >>, >>>
   - Shift operators: <<, >>, >>>.

2. Named operators in Java
   - Instanceof operator

Arithmetic Operators in Java:

- Java Arithmetic operators are used to performing fundamental arithmetic operations such as addition, subtraction, multiplication, and division on numeric data types.
- The numeric data types can be byte, short, int, long, float, and double. Java provides five arithmetic operators.

Operators	Meaning	Description
1. +	Addition or unary plus	Performs addition operation.
2. –	Subtraction or unary minus	Performs subtraction operation.
3. *	Multiplication	Performs multiplication operation.
4. /	Division	Performs division operation.
5. %	Modulo division (Remainder)	Performs remainder after division operation.

Java Switch Statement

The Java *switch statement* executes one statement from multiple conditions. It is like if-else-if ladder statement. The switch expression is evaluated once.

- The value of the expression is compared with the values of each case.
- If there is a match, the associated block of code is executed.
- After each case, a break statement is necessary inside the switch block to come out switch block. The break statement is optional. Even if you remove the break, the control of the program will flow to the next case.
- The default statement sequence is optional and is executed when none of the previous cases are matched.

Syntax:

```
switch(expression)
  {
      case value1:
      //code to be executed;
```

```
       break;
      case value2:
       / /code to be executed;
       break;
           ......
      default:
    code to be executed if all cases are not matched;
 }
```

Conclusion:

References:

Herbert Schildt , "Java : The Complete Reference" Tata McGraw-Hill (7th Edition).

Questions:

1. What is an Operator in Java?
2. What are the types of operators based on the number of operands?
3. What is an Arithmetic operator in Java?
4. What are the types of arithmetic operators? What are the priority levels of arithmetic operation in Java?
5. What is a symbolic operator in Java? What are the types of operators based on symbols?
6. What is a Java Switch statement?

Experiment No. 3

Aim of the Experiment: Write a program in Java with class Rectangle with the data fields width,length, area and colour. The length, width and area are of double type and colour is of string type. The methods are get_length(), get_width(), get_colour() and find_area(). Create two objects of Rectangle and compare their area and colour. If the area and colour both are the same for the objects then display "Matching Rectangle", otherwise display "Nonmatching Rectangle".

Objective: To understand the concept of multiple objects in Java.

Software used: Eclipse IDE 2018, JDK 1.8.0 is required

Course Outcome Addressed: CO2

Theory:

Java Classes:

Java is an object-oriented programming language. Everything in Java is associated with classes and objects, along with its attributes and methods. For example: in real life, a car is an object. The car has attributes, such as weight and color, and methods, such as drive and brake.

A Class is like an object constructor, or a "blueprint" for creating objects. To create a class, use the keyword class.

Syntax of Class in Java:

```
class <class_name>
{

   field;
   method;
}
```

A class can contain any of the following variable types.

- **Local variables** − Variables defined inside methods, constructors or blocks are called local variables. The variable will be declared and initialized within the method and the variable will be destroyed when the method has completed.
- **Instance variables** − Instance variables are variables within a class but outside any method. These variables are initialized when the class is instantiated. Instance variables can be accessed from inside any method, constructor or blocks of that particular class.
- **Class variables** − Class variables are variables declared within a class, outside any method, with the static keyword.

A class can have any number of methods to access the value of various kinds of methods.

Java Objects:

Object is an instance of a class. An object in OOPS is nothing but a self-contained component which consists of methods and properties to make a particular type of data useful.

Object Syntax in Java

ClassName ReferenceVariable = new ClassName();

There are three steps when creating an object from a class :

- **Declaration** − A variable declaration with a variable name with an object type.
- **Instantiation** − The 'new' keyword is used to create the object.
- **Initialization** − The 'new' keyword is followed by a call to a constructor. This call initializes the new object.

Multiple Objects:

You can create multiple objects of one class:

Example of Multiple Objects:

Create two objects of class Main:

```
   public class Main {
      int x = 5;


       public static void main(String[] args) {
       Main myObj1 = new Main();      // Object 1
       Main myObj2 = new Main();       // Object 2
       System.out.println(myObj1.x);
       System.out.println(myObj2.x);
    }
}
```

Using Multiple Classes:

You can also create an object of a class and access it in another class. This is often used for better organization of classes (one class has all the attributes and methods, while the other class holds the main() method (code to be executed)).

Remember that the name of the java file should match the class name

Conclusion:

References:

Herbert Schildt , "Java : The Complete Reference" Tata McGraw-Hill (7th Edition).

Questions:

1. What is the Difference between Object and Class in Java?
2. Why do we create multiple objects to single class?
3. How classes organization is better?
4. Can we declare the main method of our class as private?
5. What are the logical operators "and", "or", and "not" in Java?

Experiment No. 4

Aim of the Experiment: Write a program in JAVA to demonstrate the method and constructor overloading.

Objective: To understand the method and constructor overloading in Java.

Software used: Eclipse IDE 2018, JDK 1.8.0 is required

Course Outcome Addressed: CO2

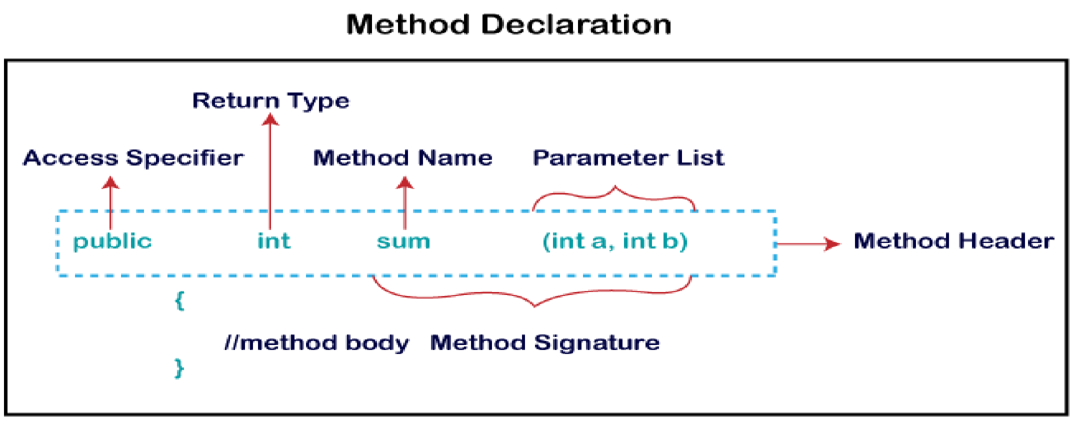
Theory:

Method in Java:

A **method** is a block of code or collection of statements or a set of code grouped together to perform a certain task or operation. It is used to achieve the **reusability** of code. We write a method once and use it many times. We do not require writing code again and again. It also provides the **easy modification** and **readability** of code, just by adding or removing a chunk of code. The method is executed only when we call or invoke it. The most important method in Java is the **main()** method.

Method Declaration:

The method declaration provides information about method attributes, such as visibility, return-type, name, and arguments. It has six components that are known as **method header**, as we have shown in the following figure.



Method Signature: Every method has a method signature. It is a part of the method declaration. It includes the method name and parameter list.

Access Specifier: Access specifier or modifier is the access type of the method. It specifies the visibility of the method. Java provides four types of access specifier:

- Public: The method is accessible by all classes when we use public specifier in our application.
- Private: When we use a private access specifier, the method is accessible only in the classes in which it is defined.
- Protected: When we use protected access specifier, the method is accessible within the same package or subclasses in a different package.
- Default: When we do not use any access specifier in the method declaration, Java uses default access specifier by default. It is visible only from the same package only.

Return Type: Return type is a data type that the method returns. It may have a primitive data type, object, collection, void, etc. If the method does not return anything, we use void keyword.

Method Name: It is a unique name that is used to define the name of a method. It must be corresponding to the functionality of the method. Suppose, if we are creating a method for subtraction of two numbers, the method name must be subtraction(). A method is invoked by its name.

Parameter List: It is the list of parameters separated by a comma and enclosed in the pair of parentheses. It contains the data type and variable name. If the method has no parameter, left the parentheses blank.

Method Body: It is a part of the method declaration. It contains all the actions to be performed. It is enclosed within the pair of curly braces.

Naming a Method:

While defining a method, remember that the method name must be a verb and start with a lowercase letter. If the method name has more than two words, the first name must be a verb followed by adjective or noun. In the multi-word method name, the first letter of each word must be in uppercase except the first word. For example:
Single-word method name: sum(), area()

Multi-word method name: areaOfCircle(), stringComparision()

Types of Method
There are two types of methods in Java:

1. Predefined Method
2. User-defined Method

1. Predefined Method

In Java, predefined methods are the method that is already defined in the Java class libraries is known as predefined methods. It is also known as the **standard library method** or **built-in method**. We can directly use these methods just by calling them in the program at any point.

Some pre-defined methods are **length(), equals(), compareTo(), sqrt(),** etc. When we call any of the predefined methods in our program, a series of codes related to the corresponding method

runs in the background that is already stored in the library.Each and every predefined method is defined inside a class. Such as **print()** method is defined in the **java.io.PrintStream** class. It

prints the statement that we write inside the method. For example, **print("Java")**, it prints Java on the console.

2. User-defined Method

The method written by the user or programmer is known as **a user-defined** method. These methods are modified according to the requirement.

Method Overloading:

Method Overloading is a feature that allows a class to have **more than one method with the same name**, if their **argument lists are different**. This is an important feature in java, there are several cases where we need more than one methods with same name, for example if we are building an application for calculator, we need different variants of add method based on the user inputs such as add(int, int), add(float, float) etc. Argument list means the parameters that a method has: For example the argument list of a method add(int a, int b) having two int parameters is different from the argument list of the method add(int a, int b, int c) having three int parameters.

Three ways to overload a method:

In order to overload a method, the argument lists of the methods must differ in either of these:

1. **Number of parameters.**

   For example: This is a valid case of overloading

   add(int, int)

   add(int, int, int)

2. Data type of parameters.

For example:

add(int, int)
add(int, float)

3. Sequence of Data type of parameters.

For example:
add(int, float)
add(float, int)

Constructor in Java:

A constructor in Java is a **special method** that is used to initialize objects. The constructor is called when an object of a class is created. It can be used to set initial values for object attributes. Every time an object is created using the new() keyword, at least one constructor is called. Constructors must have the same name as the class within which it is defined, Constructors do not return any type, Constructors are called only once at the time of Object creation.

Types of Java Constructors:

1. **Default constructor** (no-arg constructor)

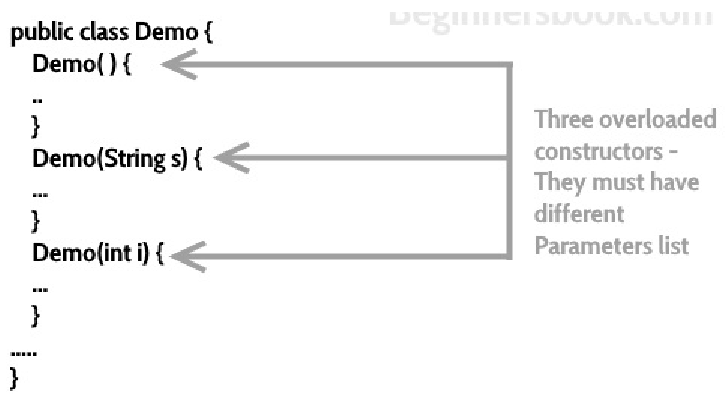
   This type of constructor is called whenever we create an object without passing any argument to the constructor.

2. **Parameterized constructor**

   This type of constructor is called if we want to initialize an object by passing a few arguments in the constructor.

Constructor overloading:

The **constructor overloading** can be defined as the concept of having more than one constructor with different parameters so that every constructor can perform a different task.



```
public class Demo {
   Demo( ) {
   ..
   }
   Demo(String s) {
   ...
   }
   Demo(int i) {
   ...
   }
.....
}
```

Three overloaded constructors - They must have different Parameters list

Benefits of Constructor Overloading in Java:

The key advantages of making use of constructor overloading while writing Java programs are:

- The constructor overloading enables the accomplishment of static polymorphism.
- The class instances can be initialized in several ways with the use of constructor overloading.
- It facilitates the process of defining multiple constructors in a class with unique signatures.
- Each overloaded constructor performs various tasks for specified purposes.

Conclusion:

References:

Herbert Schildt , “Java : The Complete Reference” Tata McGraw-Hill (7th Edition).

Questions:

1. What is a Method in Java?
2. Can we declare the main method of our class as private?
3. Explain method overloading in Java?
4. What is constructor in Java?
5. Explain constructor overloading in Java?

Experiment No. 5

Aim of the Experiment: Write programs in Java to sort i) List of integers ii) List of names. The objective of this assignment is to learn Arrays and Strings in Java

Objective: To learn Arrays and Strings in Java

Software used: Eclipse IDE 2018, JDK 1.8.0 is required

Course Outcome Addressed: CO3

Theory:

Arrays: An array is collection of homogenous or similar data items which are related and share same common name. Arrays are single types of integers, floats or characters.

Creation of an array involves three steps:

1. Declaring an array.

   typearrayname[];

   or

   type [] arrayname;

   For eg: intn[];

   int [] n;

2. Creating a new object of the array.

   arrayname = new type [size];

   For eg: a = new int[5];

3. Initialization of arrays.

   arrayname [index] = value;

   For eg: a[0] = 35;

4. You can also initialize the complete array directly:

   Type arrayname[] = {list of values};

   For eg: intn[] = {35,20,40,12,26};

5. It is also possible to assign an array object to another

   int a[] = {1,3,5};

   int b[];

   b = a;

6. We can obtain the size of array:

   Int variable = arrayname.length;

   For eg: intlen = a.length;

Two-dimensional Arrays: You need two-dimensional arrays to store a table of values.

For eg: int table[][];

table = new int [2][3];

or

int table = new int[2][3];

or

int table[2][3] = {0,0,0,1,1,1};

or

int table[][] = {{0,0,0},{1,1,1}};

String Class / String literal:

- A combination of characters is a string. Strings are constant, their values cannot be changed after they are created.
- String may be declared and created as follow:

  String firstname = new String(“Hello”);

- You can concatenate the Java strings as follows:

  String city = "New" + "Delhi";

  String fullname = firstname + lastname;

- We can also create empty strings.

  String s = new String();

String Arrays:

- We can also create and use string arrays.

  String s[] = new String[3];

  This will create a string array which can hold 3 string constants.

- We can also directly initialize the string array as follows:

  String s[] = {"Jay","Vijay","Anil"};

Conclusion:

References:

Herbert Schildt , "Java : The Complete Reference" Tata McGraw-Hill (7th Edition).

Experiment No. 6

Aim of the Experiment: Write programs in Java to add two matrices

Objective: To learn Arrays in Java Java.

Software used: Eclipse IDE 2018, JDK 1.8.0 is required

Course Outcome Addressed: CO3

Theory:

Arrays: An array is collection of homogenous or similar data items which are related and share same common name. Arrays are single types of integers, floats or characters.

Creation of an array involves three steps:

1. Declaring an array.

   typearrayname[];

   or

   type [] arrayname;

   For eg: intn[];

   int [] n;

2. Creating a new object of the array.

   arrayname = new type [size];

   For eg: a = new int[5];

3. Initialization of arrays.

   arrayname [index] = value;

   For eg: a[0] = 35;

4. You can also initialize the complete array directly:

   Type arrayname[] = {list of values};

   For eg: intn[] = {35,20,40,12,26};

5. It is also possible to assign an array object to another

   ```
   int a[] = {1,3,5};
   int b[];
   b = a;
   ```

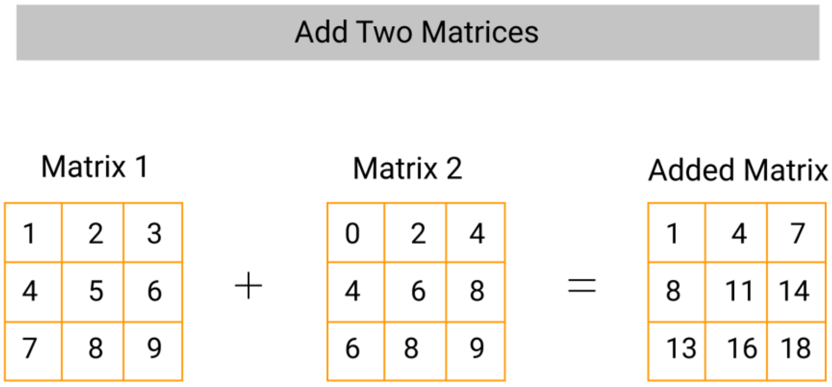
6. We can obtain the size of array:

   Int variable = arrayname.length;

   For eg: intlen = a.length;

Two-dimensional Arrays: You need two-dimensional arrays to store a table of values.

```
For eg: int table[][];
table = new int [2][3];
        or
int table = new int[2][3];
        or
int table[2][3] = {0,0,0,1,1,1};
        or
int table[][] = {{0,0,0},{1,1,1}};
```



Conclusion:

References:

Herbert Schildt , “Java : The Complete Reference” Tata McGraw-Hill (7th Edition).

Experiment No.7

Aim of the Experiment: Write a program in Java to create a player class. Inherit the classes Cricket_player, Football_player and Hockey_player from player class

Objective: To learn the concepts of inheritancein Java.

Software used: Eclipse IDE 2018, JDK 1.8.0 is required

Course Outcome Addressed: CO3

Theory:

Inheritance

- Inheritance is a mechanism of reusing something that already exists rather than creating the same all over again.
- Inheritance refers to properties sharing. It is possible to derive a new class from old one. Thus, the new class will get the properties from the old one.
- The old class is called *base class* or *super class* or *parent class.*
- The new one is called the *derived class* or *sub class* or *child class.*

Types of Inheritance:

1. **Single Inheritance:** It has one parent class and one child class.

   Syntax: class classname extends parentclassname
```
            {
                Variable declaration;
                Method declaration;
            }
```

The **extends** keyword signifies that the properties of the parent class are extended to the child class. Now the child class will contain its own variables and methods plus that of the parent class.

2. **Multilevel Inheritance:** A child class is derived from a derived class. This allow us to build a chain of classes.

   Syntax:
   ```
   class A
   {---------}
   class B extends A
   {---------}
   class C extends B
   {---------}
   ```

3. **Hierarchical Inheritance:** Many childs is derived from same parent.

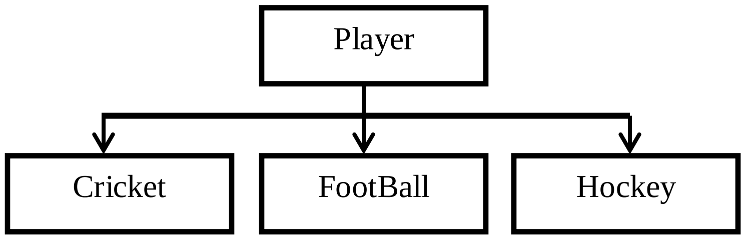
   Syntax:
   ```
   class A
   {---------}
   class B extends A
   {---------}
   class C extends A
   {---------}
   class D extends A
   {---------}
   ```

Superclass and Subclasses:

- The existing class is called ***superclass.*** It is also called as ***parent class*** or ***base class.***
- The class derived from superclass is called the ***subclass.*** It is also called as ***child class*** or ***extended class*** or ***derived class.***

Use of Super Keyword:

- To call a superclass constructor.
- To call a superclass method.



Algorithm:

Create Base Class as "Player"

Step1] Create member variable "gname" as "String"datatype

Step2] Write Parameterized Constructor"Player" with argument "gname" and store this value into member variable by using "this" keyword.

Step3] Stop

Create Derived Class as "Cricket_player "

Step1] Create member variable "no_of_player" as "int"datatype

Step2] Write Parameterized Constructor "Cricket_player "with arguments"gname" and "no_of_player". Pass the value of "gname"to Base Class "Player"using "super" keyword and Store"no_of_player"value into member variable by using "this" keyword.

Step3] Write Function "display" which will display the value of variable "gname" from Base class "Player" and "no_of_player" from derived class "Cricket_player "

Step4] Stop

Create Derived Class as "Football_player"

Step1] Create member variable "no_of_player" as "int"datatype

Step2] Write Parameterized Constructor "Football_player " with arguments "gname" and "no_of_player". Pass the value of "gname" to Base Class "Player" using "super" keyword and Store"no_of_player" value into member variable by using "this" keyword.
Step3] Write Function "display" which will display the value of variable "gname" from Base class "Player" and "no_of_player" from derived class "Football_player "
Step4] Stop

Create Derived Class as "Hockey_player"
Step1] Create member variable "no_of_player" as "int"datatype
Step2] Write Parameterized Constructor "Hockey_player" with arguments "gname" and "no_of_player". Pass the value of "gname" to Base Class "Player" using "super" keyword and Store"no_of_player" value into member variable by using "this" keyword.
Step3] Write Function "display" which will display the value of variable "gname" from Base class "Player" and "no_of_player" from derived class "Hockey_player"
Step4] Stop

Conclusion:

References:

Herbert Schildt , “Java : The Complete Reference” Tata McGraw-Hill (7th Edition).

Experiment No. 8

Aim of the Experiment: Write a Java Program which implements Interface

Objective: To learn the concept of interface in Java.

Software used: Eclipse IDE 2018, JDK 1.8.0 is required

Course Outcome Addressed: CO4

Theory:

Introduction:

An Interface is a kind of class that only contains the methods without any implementation. An interface is a named collection of method definitions (without implementations). An interface can also include constant declarations. Each variable that is defined in interface should be final and static.

Defining Interfaces:

- Java does not support multiple inheritance. Instead, it provides an alternative approach known as *interfaces.*
- **Interface** is a kind of class.
- **Interface** contains methods and variables but the methods are only abstract and variables are final.
- The class that implements **interface** define the code for these abstract methods.
- The syntax for defining an **interface**:

```
interfaceinterfacename
{   variable declaration;
methods declaration;
```

```
    }
```

- Variables are declared as follows:

  static final type variablename = value;

- Methods declaration will contain only a list of methods without any body statements.

  return_typemethodname(param);

  For eg:
```
interface area
{
final static float pi = 3.14;
floatcalc(float x, float y);
voiddisparea();
}
```

Algorithm:

Step1] Create interface named "Shape" with constant variable as "pi=3.14" and abstract method "area()"

```
 interface Shape
 {
        final float pi=3.14f;
        public void area();
  }
```

Step2] Create Class "Circle" with data member as "r" and constructor to initialize r and code for abstract method is implemented. In that, area of circle is calculated and display.

```
 public void area()
  {
       System.out.println("Area of Circle = "+(pi*r*r));
         }
```

Step3] Create Class “Sphere” with data member as “r” and constructor to initialize r and code for abstract method is implemented. In that, area of circle is calculated and display.

```
 public void area()
   {
       System.out.println("Area of  Sphere = "+(4*pi*r*r));
   }
```

Step4] In main function, interface Shape-create reference variable and refer to implemented class named ”Circle” and call method “area”

```
    Shape s;
    s = new Circle(r);
    s.area();
```

Step5] In main function, interface Shape-create reference variable and refer to implemented class named ”Sphere” and call method “area”

```
       s=new Sphere(r);
       s.area();
```

Conclusion:

References:

Herbert Schildt , “Java : The Complete Reference” Tata McGraw-Hill (7th Edition).

Experiment No. 9

Aim of the Experiment: Write a program which use try and catch for exception handling

Objective: To learn the concept of exception handling in Java.

Software used: Eclipse IDE 2018, JDK 1.8.0 is required

Course Outcome Addressed: CO5

Theory:

- An exception is an abnormal condition that occurs during the execution of a program.
- An exception is an run-time error.
- A Java exception is an object that is generated at runtime to describe a problem encountered during the execution of a program.
- When an exceptional condition occurs, an object representing that exception is created and thrown in the method that caused the error.

Exception Handling:

- The advantage of using the try and catch keywords are that, it fixes the error and prevents the program from terminating abruptly.
- The try statement contains a block of statement enclosed by braces. This is the code you want to monitor for exceptions. If a problem occurs during its execution, an exception is thrown.
- Immediately following the try block is a sequence of catch blocks. Each of these begins with the catch keyword. An argument is passed to each catch block. That argument is the exception object that contains information about the problem.

- The finally block is optional. However, in some applications it can provide a useful way to relinquish resources. For eg, you may wish to close files or databases at this point.
- Each try block must have at least one catch or finally block.

Syntax:

```
try
  {…….
  }catch(ExceptionType1 param1)
  {  // exception handling block }
  catch(ExceptionType2 param2)
  {  // exception handling block }
  ….
  Finally
  {   // finally block   }
```

Creating User Defined Exceptions:

- Java provides most of the errors but most of the times standard exception cannot handles most of the exceptions.
- Such exceptions are designed by the user itself. This type of exception is also called as unchecked exception.
- For eg if we put date 30 to month February then we have error InvalidDateException, but no such kind of exception.
- A user defined exception class must extend Exception.

  Syntax:

```
class userdefinedexception extends exception
  {     …………       }
```

Algorithm:

Step1] Create user defined exception named as "InvalidException" with toString() method which return message "Invalid Password"

Step2] Create Scanner object "s" to read from user

Step3] Inside try block, read user name in variable "user" and read password in variable "pass".

Step4] if (user=="Hello" and pass=="a123") then goto step5 otherwise goto step6

Step5] Print message "Login successful", goto step 8

Step6] throw user defined error to catch() method

Step7] catch() method will catch error and display message "Error: Invalid Password"

Step8] Stop

Conclusion:

References:

Herbert Schildt , "Java : The Complete Reference" Tata McGraw-Hill (7th Edition).

Experiment No. 10

Aim of the Experiment: Write a Java program to draw oval, rectangle, line, text using graphics class

Objective: To learn the concepts of graphics programming in Java.

Software used: Eclipse IDE 2018, JDK 1.8.0 is required

Course Outcome Addressed: CO6

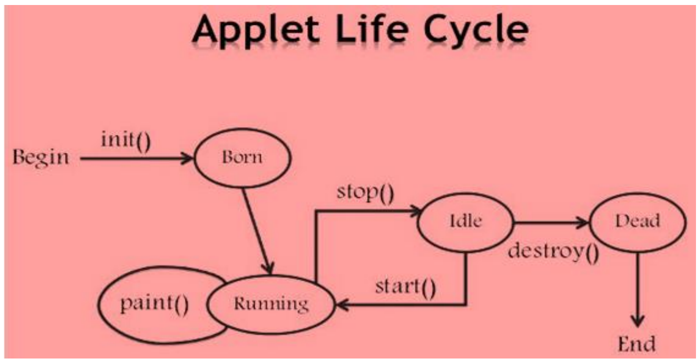
Theory:

Applet:

- An applet is a special kind of java program that is primarily used in Internet computing.
- It can be transported from one computer to another on internet and run using the *applet viewer* or any java capable *web browser* such as HotJava or Netscape.

Applet Life Cycle:

- Java applets are derived from class Applet. Any kind of applet which are either local applet or remote applet they undergoes following phases in their life cycle.



- **Initialization State:**
  - Applet enters the initialization state when it is first loaded.
  - This is achieved by calling the init() method of Applet Class.
  - The applet is born.
  - The initialization occurs only once in the applet's life cycle.
  - This method is executed only once in applet life cycle.
  - public void init()
  - {……}

- **Running State:**
  - Applet enter the running state when the system calls the **start()** method of Applet class.
  - This occurs automatically after the applet is initialized.
  - The **start()** method may be called more than once.

    public void start()

    {…….}
- **Idle or Stopped State:**
  - An applet becomes idle when it is stopped from running.
  - We can also do so by calling the **stop()** method explicitly.
  - If we use a thread to run the applet, then we must use **stop()** method to terminate the thread.
  - We can achieve by overriding the **stop()** method:

    public void stop()

    {………}
- **Dead State:**
  - An applet is said to be dead when it is removed from memory.
  - This occurs automatically by invoking **destroy()** method when we quit browser.
  - Like initialization, destroying stage occurs only once in the applet's life cycle.

  - If the applet has created any resources like threads, we may override the **destroy()** method to clean up these resources.
    public void destroy()
    {………}
- **Display State:**
  - Applet moves to the display state whenever it has to perform some output operations on the screen.
  - This happens immediately after the applet enters into the running state.
  - The **paint()** method is called to accomplish this task.
    public void paint(Graphics g)
    {………}

Algorithm:

Step1] Inside multiline comment, add applet tag

```
/*
<applet code="Slip12" width=400 height=400>
</applet>
*/
```

Step2] In paint() method, change color and call drawOval(), fillOval() method with co-ordinates with the help of graphics "g".

```
g.setColor(Color.red);
g.drawOval(200 , 10, 50,80);
g.setColor(Color.yellow);
g.fillOval(201 , 11, 49,79);
```

Step3] Next, change color and call drawRect(), fillRect() method with co-ordinates with the help of graphics "g".

```
g.setColor(Color.blue);
g.drawRect(100 , 100, 50,100);
```

```
        g.setColor(Color.cyan);
        g.fillRect(101 , 101, 49,99);
```

Step4] Next, change color and call drawLine()method with co-ordinateswith the help of graphics "g".

```
        g.setColor(Color.black);
        g.drawLine(0 , 220 , 100 , 220);
```

Step5] Next, change color and call drawString() method with co-ordinates with the help of graphics "g".

```
        g.setColor(Color.green);
        g.drawString("Welcome" , 100 ,250);
```

Step6] Stop

Conclusion:

References:

Herbert Schildt , "Java : The Complete Reference" Tata McGraw-Hill (7th Edition).

Experiment No. 11

Aim of the Experiment: Write a Java program in which data is read from one file and should be written in another file line by line

Objective: To learn the concepts of file handling in Java.

Software used: Eclipse IDE 2018, JDK 1.8.0 is required

Course Outcome Addressed: CO6

Theory:

Files:

Stream:

- A sequence of bytes is called as stream.
- Stream in which writing data to a stream is called as **Output Stream.**
- Stream in which reading data from a stream is called as **Input Stream.**
- Stream in which buffer in memory, such stream is called as a **Buffered Stream.**
- A **binary Stream** contains the binary data, while **Character Stream** contains the character data. Character Streams are basically used to storing and retrieving text.
- There are four types of Stream:
    - **Reader:** A stream to read characters.
    - **Writer:** A stream to write characters.
    - **InputStream:** A stream to read binary data.
    - **OutputStream:** A stream to write binary data.

- Java supports two streams:

- **Byte based Stream:** Row bytes read and write by byte based stream. For eg: InputStream, OutputStream.
- **Character based Stream:** Characters i.e. Unicode read and write by character based stream. For eg: Reader & Writer.

- **Predefined Stream:** System class define by the package java.lang package automatically by the java programs.
  - **System.out:** Standard Output by default, i.e. console.
  - **System.in:** Standard Input provide by keyboard by default.
  - **System.err:** For the outputting error.

1. **FileReader:**

- The FileReader class is a subclass of InputStreamReader.
- It is used to read characters from a file.
- Initially file opening must essential before read from or write to a file.
  Syntax: FileReaderfileptrname = new FileReader(filename);
  For eg: FileReaderfp = new FileReader("sample.java");

2. **FileWriter:**

- FileWriter provides a convenient way to writing characters to a file.
- To open a file for writing use the FileWriter class and create an instance from it.
  FileWriter f = new FileWriter("sample");
- Constructors of FileWriter:
  publicFileWriter(File file)
  publicFileWriter(File file, boolean append)
  publicFileWriter(String path)
  publicFileWriter(String path, boolean append)
  publicFileWriter(FileDescriptorfd)

- Methods in FileWriter Class:

  write(String str) – Writes string or array of chars to file.

  write(int n) – Writes a int value into a file.

  flush – Writes any buffered chars to file.

  close – Closes the file stream after performing a flush.

Algorithm:

```
Step1] Create object of FileReader as “fr”, in that open file “Slip13.java”.
Step2] Create object of FileWriter as “fw”, in that open file “11.java”.
Step3] Read character by character with help of read() method from file object “fr”and store it in character type variable “c”.
Step4] Store character type “c” into file object “fw” with help of write() method and also print it on screen
Step5] Repeat from Step 3 until (c = fr.read())!=-1)
Step6] Print message "1 File get Copied...."
Step7] Close all files
        fr.close();
        fw.close();
Step8] Stop
```

Conclusion:

References:

Herbert Schildt , “Java : The Complete Reference” Tata McGraw-Hill (7th Edition).

Experiment No. 12

Aim of the Experiment: To understand the Inheritance in Java.

Objective: To learn the concepts of inheritance.

Software used: Virtual Labs

Course Outcome Addressed: CO3

Theory:

Inheritance in Java:

- It is an important part of OOPs (Object Oriented programming system).
- It is the mechanism in java by which one class is allowed to inherit the features(fields and methods) of another class.
- The idea behind inheritance in Java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class.

Why use inheritance in java?

- For Method Overriding (so runtime polymorphism can be achieved).
- For Code Reusability.

Important terminology:

- **Super Class:** The class whose features are inherited is known as superclass(or a base class or a parent class).

- **Sub Class:** The class that inherits the other class is known as a subclass(or a derived class, extended class, or child class). The subclass can add its own fields and methods in addition to the superclass fields and methods.

- **Reusability:** Inheritance supports the concept of "reusability", i.e. when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class.

How to use inheritance in Java?

The keyword used for inheritance is **extends**.The extends keyword indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.

The syntax of Inheritance in JAVA:

```
class Subclass-name extends Superclass-name
       {
         //methods and fields
       }
```

Types of inheritance in java:

On the basis of class, there can be three types of inheritance in java:

- Single
- Multilevel
- Hierarchical

In java programming, multiple and hybrid inheritance is supported through interface only. We will learn about interfaces later.

1. Single Inheritance: When a class inherits another class, it is known as a **single inheritance**. In the example given below, Dog class inherits the Animal class, so there is the single inheritance.

2. Multilevel Inheritance: When there is a chain of inheritance, it is known as **multilevel inheritance**. In the example given below, BabyDog class inherits the Dog class which again inherits the Animal class, so there is a multilevel inheritance.

3. Hierarchical Inheritance: When two or more classes inherits a single class, it is known as **hierarchical inheritance**. In the example given below, Dog and Cat classes inherits the Animal class, so there is hierarchical inheritance.

Why multiple inheritance is not supported in java?

- To reduce the complexity and simplify the language, multiple inheritance is not supported in java.
- Consider a scenario where A, B, and C are three classes. The C class inherits A and B classes. If A and B classes have the same method and you call it from child class object, there will be ambiguity to call the method of A or B class.Since compile-time errors are better than runtime errors, Java renders compile-time error if you inherit 2 classes. So whether you have same method or different, there will be compile time error.

Procedure:

The following steps will be followed on the virtual lab simulator:

- The experiment begins with previewing different components of java programs in Simulator section of preview tab.
- Click Start Experiment button in Simulator to start the experiment.
- Read and understand the first line of the java program and click Add to insert syntax in Syntax section.
- Read and understand the subsequent lines of the java program and click Add button to insert syntax in Syntax section.
- Insert values in input boxes wherever prompted in the Section.
- After understanding the whole program click Execute button to run the program and see output in Output Section.

Conclusion:

References:

Link of the Virtual Lab: https://java-iitd.vlabs.ac.in/

Experiment No. 13

Aim of the Experiment: Implementing method overloading through polymorphism

Objective: To learn the concepts of method overloading

Software used: Virtual Labs

Course Outcome Addressed: CO2

Theory:

Java Method Overloading:

- If a class has multiple methods having same name but different in parameters, it is known as Method Overloading.
- If we have to perform only one operation, having same name of the methods increases the readability of the program.
- Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as a(int,int) for two parameters, and b(int,int,int) for three parameters then it may be difficult for you as well as other programmers to understand the behavior of the method because its name differs.

You can create same method name with different argument list example:

```
voidfunc() { ... }
voidfunc(int a) { ... }
floatfunc(double a) { ... }
floatfunc(int a, float b) { ... }
```

Why method overloading?

Suppose, you have to perform the addition of given numbers but there can be any number of arguments (let’s say either 2 or 3 arguments for simplicity). In order to accomplish the task, you can create two methods sum2num(int, int) and sum3num(int, int, int) for two and three parameters respectively. The better way to accomplish this task is by overloading methods. And, depending upon the argument passed, one of the overloaded methods is called. This helps to increase the readability of the program.

Advantage of method overloading:

- Method overloading increases the readability of the program.
- We don’t have to create and remember different names for functions doing the same thing.

How to perform method overloading in Java?

- Overloading by changing the number of arguments
- By changing the datatype of parameters

1. Overloading by changing the number of arguments

Java program to demonstrate working of method overloading in Java. If overloading was not supported by Java, we would have to create method names like sum1, sum2, … or sum2Int, sum3Int, … etc. Example..

```
public class Sum {


        // Overloaded sum(). This sum takes two int parameters
        public int sum(int x, int y)
        {
           return (x + y);
        }

        // Overloaded sum(). This sum takes three int parameters
        public int sum(int x, int y, int z)
        {
            return (x + y + z);
        }

        // Overloaded sum(). This sum takes two double parameters
        public double sum(double x, double y)
        {
            return (x + y);
        }

        // Driver code
        public static void main(String args[])
        {
            Sum s = new Sum();
            System.out.println(s.sum(10, 20));
            System.out.println(s.sum(10, 20, 30));
            System.out.println(s.sum(10.5, 20.5));
        }
    }
```

2. By changing the datatype of parameters

```
  classMethodOverloading
    {
        // this method accepts int
    private static void display(int a)
    {
        System.out.println("Got Integer data.");
    }

    // this method  accepts String object
    private static void display(String a)
    {
      System.out.println("Got String object.");
    }
```

Can we overload java main() method?

Yes, by method overloading. You can have any number of main methods in a class by method overloading. But JVM calls main() method which receives string array as arguments only. Let's see the simple example:

```
class TestOverloading4
        {
        public static void main(String[] args)
        {
        System.out.println("main with String[]");
        }
        public static void main(String args)
        {
        System.out.println("main with String");
        }
        public static void main()
        {
```

```
System.out.println("main without args");
}
}
```

Procedure:

The following steps will be followed on the virtual lab simulator:

- The experiment begins with previewing different components of java programs in Simulator section of preview tab.
- Click Start Experiment button in Simulator to start the experiment.
- Read and understand the first line of the java program and click Add to insert syntax in Syntax section.
- Read and understand the subsequent lines of the java program and click Add button to insert syntax in Syntax section.
- Insert values in input boxes wherever prompted in the Section.
- After understanding the whole program click Execute button to run the program and see output in Output Section.

Conclusion:

References:

Link of the Virtual Lab: https://java-iitd.vlabs.ac.in/