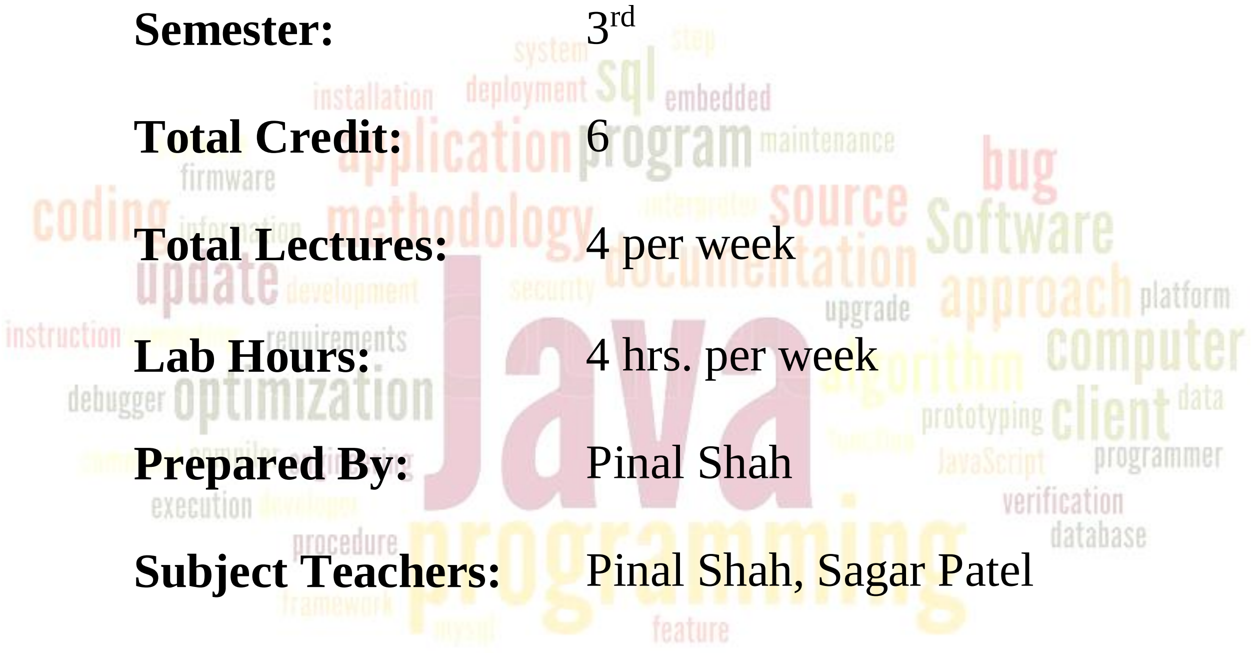
CHARUSAT

CSPIT

Department of Information Technology

Laboratory Manual



Subject:	Java Programming
Code:	IT 242
Offered to:	2nd Year Students
Semester:	3rd
Total Credit:	6
Total Lectures:	4 per week
Lab Hours:	4 hrs. per week
Prepared By:	Pinal Shah
Subject Teachers:	Pinal Shah, Sagar Patel
Lab Teachers:	Sagar Patel, Nehal Patel, Jalpesh Vasa, Ayesha Shaikh, Harsh Patel

CHARUSAT

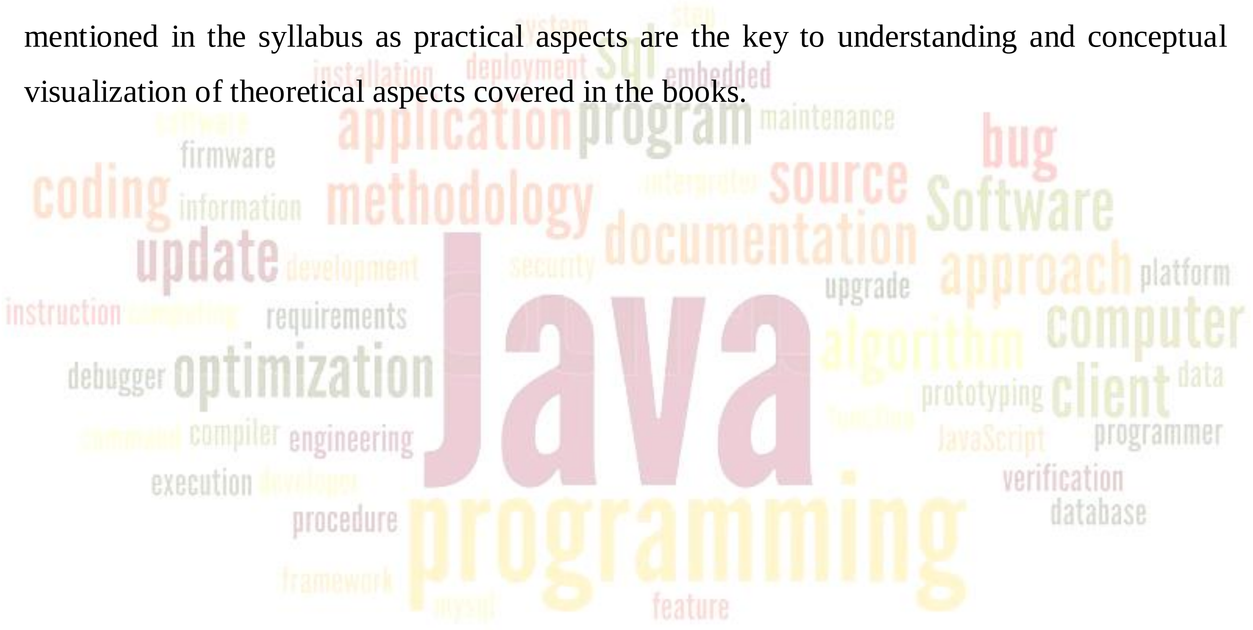
CSPIT

Department of Information Technology

<u>Laboratory Manual Content</u>

This manual is intended for the Second year students of Information Technology Department in the subject of Java Programming. It contains various practical and Lab Sessions related to Java Programming which covers various aspects related the subject to enhanced understanding.

Students are advised to thoroughly go through this manual rather than only topics mentioned in the syllabus as practical aspects are the key to understanding and conceptual visualization of theoretical aspects covered in the books.



<u>Dr. Parth Shah</u>
Hod, IT Department
CSPIT, Charusat, Changa

<u>Mr. Pinal Shah</u>
Assistant Professor, IT Department,
CSPIT, Charusat, Changa

CHARUSAT

CSPIT

Department of Information Technology

Vision of Department

To inculcate excellent education to enhance professional behavior, strong ethical values, innovative research capabilities and leadership abilities in the young minds so as to work with a commitment to the progress of the country.

Mission of Department

To impart quality education in both the theoretical and applied foundations of IT and educate students to effectively apply this education to solve real-world problems thus strengthening their lifelong high-quality careers in global work environment of the 21st century.

Objectives of the Course

Objective of the Course:

This subject introduces OOP using Java as the implementation language. It emphasizes proper formulation and abstraction of the problem domain in the programming process in order to build programs that are robust, secure, and portable.

The objective of course is,

- To teach the model of object oriented programming concepts like abstract data types, encapsulation, inheritance and polymorphism.
- To deliver the knowledge about fundamental features of core Java like object classes and interfaces, exceptions, libraries of object collections ,GUI and Lambda Expression.
- To teach how to take the statement of a business problem and from this determine suitable logic for solving the problem; then be able to proceed to code that logic as a program.
- To demonstrate how to test and prepare a real time application using java.

CHARUSAT

CSPIT

Department of Information Technology

INDEX

Week 1: Write a simple Java program which covers the basic syntactical structure of programming like…

- Operators & Expressions
  (Arithmetic, logical, bitwise, ternary, relational etc.…)
- Looping Statement
  (while, do while, for, switch)
- Decision Making Statement
  (if, if else if, if elseif ladder etc. …)

Week 2: Write a Java program to define a class, describe its constructor & overload its constructor.

Week 3: Write Java programs for:

- Command line argument
- String class Methods

Week 4: Write Java programs for multilevel and hierarchical inheritance.

Week 5: Write a Java program to implement Multiple Inheritance. **(Interface)**

Week 6: Write a Java program to create a package & use it in another program.

Week 7: Write Java programs to demonstrate use of exception and user defined exception.

Week 8: Write a Java program to implement concept of multithreading.

Week 9: Producer Consumer Problem **(Thread Synchronization)**

Week 10: Write a Java program to demonstrate Applet.

Week 11: Write a Java program to implement Event Handling.

Week 12: Write a Java Program to demonstrate the concept of Lambda Expression in Java 8.

Week 13: Write a Java program to implement Wrapper classes and their methods.

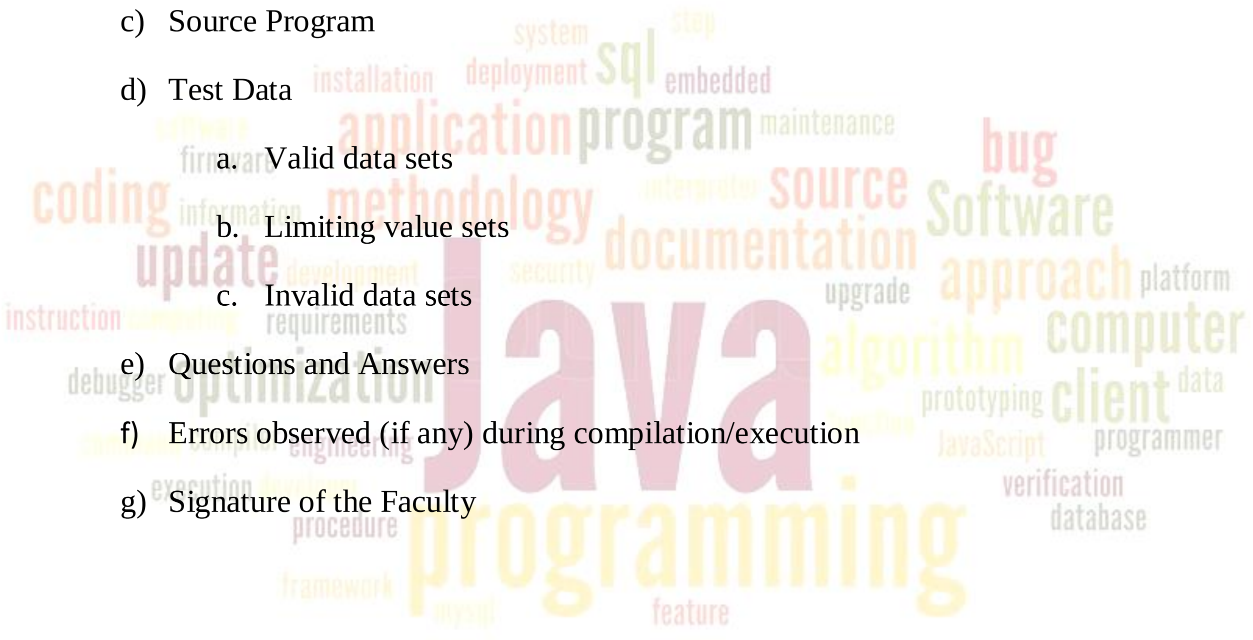
CHARUSAT

CSPIT

Department of Information Technology

File Format for Students

The students will write the today's experiment in the Observation book as per the following format:



a) Name of the experiment/Aim
b) Algorithm
c) Source Program
d) Test Data
    a. Valid data sets
    b. Limiting value sets
    c. Invalid data sets
e) Questions and Answers
f) Errors observed (if any) during compilation/execution
g) Signature of the Faculty

Week 1

Basics of Java Programming

AIM: Write a simple Java program which covers the basic syntactical structure of programming like…

1. Operators & Expressions
2. Looping Statement
3. Decision Making Statement

OBJECTIVE:

- Understand fundamentals of programming such as variables, operators, loops.
- Be able to use the Java SDK environment to create, debug and run simple Java programs.

THEORY:

1. **Operators**

Operator	Result
+	Addition
-	Subtraction (also unary minus)
*	Multiplication
/	Division
%	Modulus
++	Increment
+=	Addition assignment
_=	Subtraction assignment
*=	Multiplication assignment
/=	Division assignment
%=	Modulus assignment
--	Decrement

Arithmetic Operators

Operator	Result
~	Bitwise unary NOT
&	Bitwise AND
|	Bitwise OR
>>	Shift right
>>>	Shift right zero fill
<<	Shift left
&=	Bitwise AND Assignment
|=	Bitwise OR Assignment

Bitwise Operator

Operator	Result
==	Equal to
!=	Not equal to
>	Greater than

Relational Operator

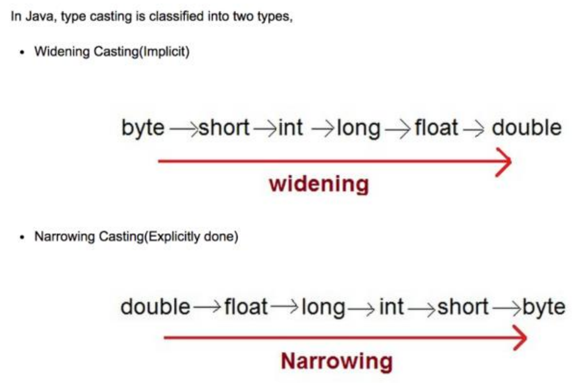
Operator	Results
&&	Logical AND
||	Logical OR
!	Logical unary NOT
?:	Ternary if-then-else

Boolean Logical Operator

<u>Data Types available in Java</u>

Data Type	Memory Size	Default value	Declaration
Byte	8 bits	0	byte a=9;
Short	16 bits	0	short b=89;
Int	32 bits	0	int c=8789;
Long	64 bits	0	long=9878688;
Float	32 bits	0.0f	float b=89.8f;
Double	64 bits	0.0	double c =87.098
Char	16 bits	'\u0000'	char a ='e';
boolean	JVM Dependent	false	boolean a =true;

<u>Type Conversion in Java</u>


In Java, type casting is classified into two types,

- Widening Casting(Implicit)

$$\begin{tikzcd}\text{byte} \arrow[r] & \text{short} \arrow[r] & \text{int} \arrow[r] & \text{long} \arrow[r] & \text{float} \arrow[r] & \text{double}\end{tikzcd}$$

- Narrowing Casting(Explicitly done)

$$\begin{tikzcd}\text{double} \arrow[r] & \text{float} \arrow[r] & \text{long} \arrow[r] & \text{int} \arrow[r] & \text{short} \arrow[r] & \text{byte}\end{tikzcd}$$

Java allows mixing of constants and variables of different types in an expression, but has during evaluation it applies rules of type conversion. If the operands are of two different types the lower type is automatically converted to higher type before operation proceeds.

The following changes are introduced during assignment

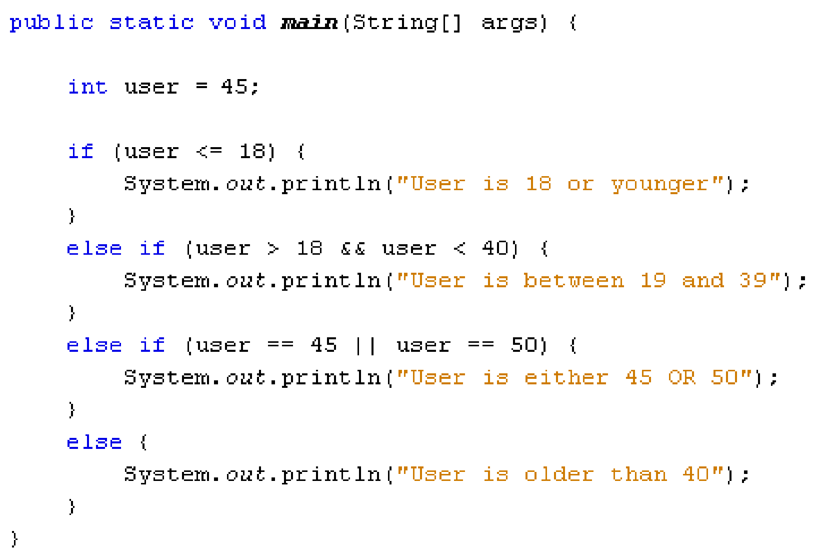
- float to int causes truncation of fractional part.
- double to float causes rounding of digits.
- long to int causes dropping of excess higher order bits.

2. Decision Making Statement & Looping

if statement

- Simple if
- if..else statement
- Nested if..else statement
- else if ladder

<u>Sample Code </u>----------->



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

    int user = 45;

    if (user <= 18) {
        System.out.println("User is 18 or younger");
    }
    else if (user > 18 && user < 40) {
        System.out.println("User is between 19 and 39");
    }
    else if (user == 45 || user == 50) {
        System.out.println("User is either 45 OR 50");
    }
    else {
        System.out.println("User is older than 40");
    }
}
```

Switch Statement:

The switch statement is Java's multiway branch statement. Here is the general form of a switch statement:

```
switch (expression)
{
case value1:

statement sequence break;

case value2:

statement sequence break;

...
case valueN:

statement sequence break;

default:

default statement sequence

}
```

The While statement:

The while loop is an entry- control loop statement. It takes the following form**:**

```
Initialization;
while (test condition) {
body of the loop;
}
```

The do statement:

The do statement takes the following

```
type do {
body of the loop;
}
while (test-condition);
```

for statement:

```
for(initialization; test condition ; iteration)
{
body of the loop;
}
```

ALGORITHM:

1. Declare and define a class. Define main method.
2. Declare variables and initialize variables.
3. Implement different operators in Java for e.g. Arithmetic, logical, conditional etc. with the help of variables.
4. Print all the values of variables, by using basic syntax of Java language.

SAMPLE PROGRAMS:

1. Write a simple Java program which covers all the supported operators like Arithmetic, logical or bitwise using the concept of switch statement.
2. Write a Java Program to calculate simple interest.
3. Write a Java Program to convert given number into characters. **ex:**(i/p: 1….o/p: one)
4. Write a Java Program to print whether the given number is Armstrong or not.

STEPS TO EXECUTE JAVA PROGRAM USING COMMAND PROMPT:

1. Open Command Prompt and check whether the corrected version of JDK has been installed or not.

   Type javac and check whether you get an error or other message.
2. Go to the program directory where you have stored your .java file.

   Type cd Directory_name
3. Compile your first Java program and check your program directory for class file.

   Javac filename.java
4. Execute the Java Program.

   Java filename

QUESTIONS

1. What is the role of JVM? How many types of memory allocated by JVM?
2. Define class in broader way.
3. What is the difference between instance variable and class variable?
4. Write any five Java supported Ide's name.

Week 2

Object, Method & Constructor Overloading

AIM: Write a Java program to define a class, describe its constructor & overload its constructor.

OBJECTIVE:

- Understand fundamentals of object-oriented programming in Java, including defining classes, creating objects, invoking methods.
- To understand the concept of constructor and constructor overloading.

THEORY:

1. **Defining Class:**
   A class is a user defined data type with template that serves to define its properties. Once the class has been defined, we can create "variables" of that type using declarations that are similar to the basic type declarations.

   The basic definition of class is:

```
class class_name
{
[fields declarations;]
[method declarations;]
}
```

2. **Initializing Variable:**
   Data is encapsulated in a class by placing the data fields inside the body of the class definition. These variables are called instance variables because they are created whenever an object of the class is instantiated.

   For example:

```
class Rectangle
{
int length;
int width;
}
```

3. **Declaration of Method:**
   The methods are declared inside the body of the class but immediately after the declaration of instance variables.

   Method declaration has four basic parts:

   - The name of the method(method name)
   - The type of the value the method returns(type)
   - A list of parameters (parameter-list)

- The body of the method

General form for declaring method is:

```
type method_name (parameter-list)
{
Method-body;
  }
```

4. **Creating an Object:**
An object in java is block of memory that contains space to store the instance of variables. Creating an object is also known as instantiating an object. Objects are created using a new operator.

For example:

```
Rectangle rect1;  //declare the object

rect1 = new Rectangle (); //instantiate the object
```

Both statements can be combined into one as shown below:

```
Rectangle rect1 = new Rectangle ();
```

The method Rectangle () is the default constructor of the class. We can create any number of objects of Rectangle. Each object has its own copy of variables of its class.

5. **Accessing a Class Members using Object:**
As we have created objects each containing its own set of variables, we should assign values to these variables in order to use them in program. All variables must be assigned values before they are used.

We must use the concerned object and the dot operator as shown below:

```
objectname.variablename = value;

objectname.methodname (parameter-list);
```

6. **Defining a Constructor:**

A *constructor* initializes an object immediately upon creation. It has the same name as the class in which it resides and is syntactically similar to a method. Once defined, the constructor is automatically called immediately after the object is created, before the **new** operator completes.

Constructors look a little strange because they have no return type, not even **void**. This is because the implicit return type of a class' constructor is the class type itself. It is the constructor's job to initialize the internal state of an object so that the code creating an instance will have a fully initialized, usable object immediately.

<u>**Default Constructor:**</u> The default constructor automatically initializes all instance variables to zero. The default constructor is often sufficient for simple classes, but it usually won't do for more sophisticated ones. Once you define your own constructor, the default constructor is no longer used.

<u>**Parameterized Constructors:**</u> For example consider **Box ( )** constructor it is not necessary—all boxes have the same dimensions. What is needed is a way to construct **Box** objects of various dimensions. The easy solution is to add parameters to the constructor.

Box defines a parameterized constructor that sets the dimensions of a box as specified by those parameters.

```
/* Here, Box uses a parameterized constructor to initialize the dimensions of
a box. */

class Box {

double width;
double height;
double depth;

//    This is the parameterized constructor for Box.

Box(double w, double h, double d) {

width = w;
height = h;
depth = d;
}
 //   compute and return volume double volume () {
return width * height * depth;
}
}
```

7. **Method Overloading:**

- In Java it is possible to define two or more methods within the same class that share the same name, as long as their parameter declarations are different.
- When this is the case, the methods are said to be *overloaded*, and the process is referred to as *method overloading.*
- When an overloaded method is invoked, Java uses the type and/or number of arguments as its guide to determine which version of the overloaded method to actually call.
- Thus, overloaded methods must differ in the type and/or number of their parameters.

8. **Constructor Overloading:**

In addition to overloading normal methods, you can also overload constructor. In fact, for most real-world classes that you create, overloaded constructors will be the normal, not the exception.

ALGORITHM:

- Create any class, describe it’s constructor, define class variables, and methods of class.
- Initialize the class variables
- Also overload the constructor.
- Define main class, create object of above class.
- Access members of above class in main class.
- Print the result.
- End of the Program.

SAMPLE PROGRAMS:

1. Demonstrate the use of public, private and protected specifier with suitable example. List down the cases where public, private or protected variables are not accessible.
2. Write a Java Program to demonstrate the Method Overloading with different numbers of parameters in argument list.
3. Write a Java Program to demonstrate the Method Overloading with difference in datatypes of parameters.
4. Write a Java Program to demonstrate the Method Overloading by changing the sequence of datatypes of arguments.
5. Similarly perform above three programs with the concept of constructor overloading.

Note: *In above program, students are allowed to use their own class, field variables and methods. There should be unique class definition for each students.*

QUESTIONS

1. List down few valid and invalid cases of Method overloading in Java.
2. Explain the role of Default constructor.
3. Class is the best example for providing encapsulation. Justify.
4. What is Compile time polymorphism in Java?
5. What are the different types of access specifiers in Java?

Week 3

Command Line Argument & String Class Methods

AIM: Write Java programs for Command line argument and String class Methods.

OBJECTIVE:

- Understand how to pass parameters using command line arguments.
- Learn & implement various methods of String class.

THEORY:

Command line Arguments:

Sometimes you will want to pass information into a program when you run it. This is accomplished by passing *command-line arguments* to **main ( )**. A command-line argument is the information that directly follows the program's name on the command line when it is executed. To access the command-line arguments inside a Java program is quite easy— they are stored as strings in a **String** array passed to the **args** parameter of **main( )**. The first command-line argument is stored at **args [0]**, the second at **args [1]**, and so on. For example, the following program displays all of the command-line arguments that it is called with:

```
//    Display all command-line arguments.
class CommandLine
{
public static void main(String args[])
{
for(int i=0; i<args.length; i++)
System.out.println("args[" + i + "]: " +args[i]);
}
}
```

String Class & Methods:

The most direct way to create a string is to write:

```
String greeting=”Hello World”;
```

Whenever it encounters a string literal in your code, the compiler creates a String object with its value in this case, "Hello world!'.

The String class is immutable, so that once it is created a String object cannot be changed. If there is a necessity to make a lot of modifications to Strings of characters, then you should use String Buffer & String Builder Classes.

String Length:

Methods used to obtain information about an object are known as accessor methods. One accessor method that you can use with strings is the length () method, which returns the number of characters contained in the string object.

Concatenating Strings:

The String class includes a method for concatenating two strings:

```
String1.concat(string2);.
```

Different String Methods:

Here is the list of methods supported by String class:

S.N.	Methods with Description
1	char charAt(int index) Returns the character at the specified index.
2	int compareTo(Object o) Compares this String to another Object.
3	String concat(String str) Concatenates the specified string to the end of this string.
4	boolean equals(Object anObject) Compares this string to the specified object.
5	int length() Returns the length of this string.
6	String replace(char oldChar, char newChar) Returns a new string resulting from replacing all occurrences of oldChar in this string with newChar.
7	String replaceAll(String regex, String replacement

Replaces each substring of this string that matches the given regular expression with the given replacement.

8	String replaceFirst(String regex, String replacement)
	Replaces the first substring of this string that matches the given regular expression with the given replacement.
9	String substring(int beginIndex)
	Returns a new string that is a substring of this string.
10	String substring(int beginIndex, int endIndex)
	Returns a new string that is a substring of this string.
11	String toLowerCase()
	Converts all of the characters in this String to lower case using the rules of the default locale.
12	String toString()
	This object (which is already a string!) is itself returned.
13	String toUpperCase()
	Converts all of the characters in this String to upper case using the rules of the default locale.
14	String trim()
	Returns a copy of the string, with leading and trailing whitespace omitted.

ALGORITHM:

- Declare a class
- Define & declare main method.
- Create an object of String class & initialize it.
- Use various methods on the object.
- Print the result.
- End of the program.

SAMPLE PROGRAMS:

1. Write a program in Java to demonstrate at least 10 string class methods with proper output.
2. Write a program in Java to arrange the numbers in ascending order. (User needs to supply five numbers through command line).

3. Write a program in Java to find total available prime numbers between the given ranges. (Range should be given in command line.)

QUESTIONS

1. How to perform command line argument program in NetBeans Ide. Write down the steps for the same.
2. String is immutable in Java. Justify. Give proper example.
3. In how many ways you can create string object in Java?
4. Does String is Thread safe in Java?
5. What do you mean by mutable and immutable objects in Java? Explain with example.
6. What is the difference between Java String and C++ string?
7. Explain the concept of String constant Pool.

Week 4
Inheritance

AIM: Write Java programs for multilevel and hierarchical inheritance.

OBJECTIVE:

- Discuss various types of Inheritance.
- Write Java classes which extend existing classes by changing the behavior of their methods using inheritance.
- Understand the superclass-subclass relationship.
- Write a Java classes which adds extra methods to existing Java classes using inheritance.

THEORY:

Inheritance may take different forms:

- Single Inheritance(only one super class)
- Multiple inheritance(several super classes)
- Hierarchical inheritance(one super class, many sub classes)
- Multilevel inheritance(Derived from a derived class)

Java does not directly implement multiple inheritance. It is implemented by using a secondary inheritance path in form of interfaces.

Defining a Subclass:

A subclass is defined as follows:

```
class subclassname extends superclassname {
Variables declaration;
methods declaration;
}
```

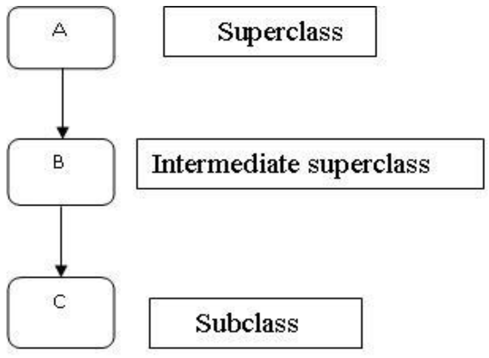
The keyword extends specifies that the properties of the superclassname are extended to the subclassname. The sub class now contains its own variables or methods as well those of the super class.

Subclass Constructor:

A subclass constructor is used to construct the instance variables of both the subclass and the superclass. The subclass constructor uses the keyword super to invoke the constructor method of superclass.

- super may only be used within a subclass constructor method.
- The call to superclass constructor must appear as first statement within the sub class constructor.
- The parameters in the super call must match the order and type of the instance variables declared in super class.

Multilevel Inheritance: A common requirement in object oriented programming is the use of a derived class as super class. Java supports the concept and uses it extensively in building its class library.



The class A serves as a base class for derived class B which in turn serves as a base class for the derived class C. The chain ABC is known as inheritance path.

A derived class with multilevel base classes is declared as follows.

```
class A
{
}
class B extends A // First level
{
}
class C extends B //Second level
{
}
```

Hierarchical Inheritance:

Deriving many subclasses from one super class is known as Hierarchical Inheritance. Another interesting application of inheritance is to use it as a support to hierarchical design of a program. Many programming problems can be cast into hierarchy where certain features of one level are shared by many other below the level.

For example:

Below diagram shows a hierarchical classification of accounts in a commercial bank. This is possible because all the accounts possess certain common features.

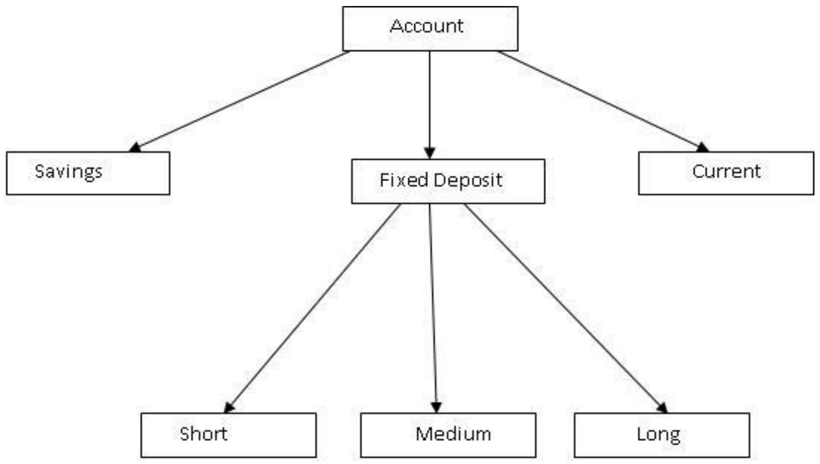


Fig. 4.2 Hierarchical classification of Bank Account

ALGORITHM:

Multilevel Inheritance:

- Define base class/ Parent class, Define members of base class.
- Derive child class from base class with the help of extend keyword.
- Derived another class from already derived one. (Derive more classes if required).
- Declare and define members of classes.
- Define main class, create object of class which you want to access.
- You can access members of base class in derived class just by creating object of that class.
- End of the program.

Hierarchical Inheritance:

- Define base class/ Parent class , Define members of base class.
- Derive child class from base class with the help of extend keyword.
- Derived another child class from already declared parent class with the help of extends keywords.
- Derived another child class from already declared parent class with the help of extends keywords. (create as many child classes as you want)

- Declare and define members of classes.
- Define main class, create object of class which you want to access.
- You can access members of base class in derived class just by creating object of that class.
- End of the program.

SAMPLE PROGRAMS:

1. Demonstrate the use of single level inheritance by taking proper class and sub class along with variables. Create the object of both the class and try to identify valid and invalid cases in terms of method calling.
2. Write a Java program to facilitate the multilevel inheritance. (Also demonstrate the use of constructor in multilevel inheritance)
3. Demonstrate the use of super keyword in inheritance and try to implement the program which can able to perform following operations…
   a. Use super to call base class variable.
   b. Use super to call base class method.
   c. Use super to call base class constructor.
4. Write a Java Program to demonstrate the concept of hierarchical inheritance. (implement this program with super keyword, final keyword, constructor and method overriding concept)

QUESTIONS

1. Explain the use of final keyword with method, variable and class.
2. How method overriding is differ from method overloading?
3. Explain the various types of inheritance supported by Java with example.
4. Can we override static method in Java?
5. Show the comparison between C++ inheritance and Java inheritance.

Week 5

Interface & Abstract Class

AIM: Write a Java program to implement Multiple Inheritance. **(Interface)**

OBJECTIVE:

- Understand the concept of interfaces.
- Learn how to use interfaces to achieve multiple inheritance.

THEORY:

Interface:

Java does not support multiple inheritance but it can be achieved using interfaces. Interfaces are syntactically similar to classes, but they lack instance variables, and their methods are declared without any body.

Defining an Interface:

An interface is defined much like a class. This is the general form of an interface:

```
access interface name
{
return-type method-name1(parameter-list);
return-type method-name2(parameter-list);
type final-varname1 = value;
type final-varname2 = value;
// ...
return-type method-nameN(parameter-list);
type final-varnameN = value;
}
```

When no access specifier is included, then default access results, and the interface is only available to other members of the package in which it is declared. When it is declared as **public**, the interface can be used by any other code.

Implementing Interfaces:

Once an **interface** has been defined, one or more classes can implement that interface. To implement an interface, include the **implements** clause in a class definition, and then create the methods defined by the interface. The general form of a class that includes the **implements** clause looks like this:

```
class classname [extends superclass] [implements interface [,interface...]]
{
// class-body
}
```

If a class implements more than one interface, the interfaces are separated with a comma.

Interfaces Can Be Extended:

One interface can inherit another by use of the keyword **extends**. The syntax is the same as for inheriting classes. When a class implements an interface that inherits another interface, it must provide implementations for all methods defined within the interface inheritance chain.

Abstract class important points:

1. abstract keyword is used to create an abstract class in java.
2. Abstract class in java can't be instantiated but you can create the reference of it.

```
abstract class Base {
    abstract void fun();
}
class Derived extends Base {
    void fun() {
System.out.println("Derived fun() called");
}
}

class Main {

    public static void main(String args[]) {

        // Uncommenting the following line will cause compiler error as the
        // line tries to create an instance of abstract class.
        // Base b = new Base();

        // We can have references of Base type.
        Base b = new Derived();
        b.fun();

    }

}
```

3. We can use abstract keyword to create an abstract method, an abstract method doesn't have body.
4. If a class have abstract methods, then the class should also be abstract using abstract keyword, else it will not compile.

5. It’s not necessary to have abstract class to have abstract method.

```
abstract class Base {
    void fun() { System.out.println("Base fun() called");
}

}

class Derived extends Base {


}


class Main {
    public static void main(String args[]) {
        Derived d = new Derived();
        d.fun();

    }
}
```

6. If abstract class doesn’t have any method implementation, it’s better to use interface because java doesn’t support multiple class inheritance.
7. The subclass of abstract class in java must implement all the abstract methods unless the subclass is also an abstract class.
8. All the methods in an interface are implicitly abstract unless the interface methods are static or default. Static methods and default methods in interfaces are added in Java 8.
9. Java Abstract class can implement interfaces without even providing the implementation of interface methods.
10. Java Abstract class is used to provide common method implementation to all the subclasses or to provide default implementation.
11. We can run abstract class in java like any other class if it has main () method.

ALGORITHM:

- Define an interface along with its variables & methods.
- Define a class which implements the interface methods.
- Define base class/ Parent class, Define members of base class.
- Derive child class from base class with the help of extend keyword.
- Declare and define members of classes.
- Define main class, create object of class which you want to access.
- Print the variable values.
- End of the program.

SAMPLE PROGRAMS:

1. Write a java program to demonstrate dynamic method dispatch and abstract keyword with class and methods.
2. Try to implement the concept of multiple inheritance in Java with the use of interface.
3. Write a Java program to demonstrate the hybrid inheritance with example.

QUESTIONS

1. Differentiate abstract class with interface.
2. Can we create the object of abstract class? Justify.
3. Is it compulsory for a class which is declared as abstract to have at least one abstract method?
4. Can we use final and abstract together with class name? Why? Explain with example.
5. Can we declare any abstract method as static method?
6. Compare virtual function of C++ with abstract class in Java with example.

Week 6
Package

AIM: Write a Java program to create a package & use it in another program.

OBJECTIVE:

- Use of built-in packages in Java
- To create user defined packages & use them in another program
- Learn how to extend packages.
- Use of various access protections with packages.

THEORY:

Packages are Java's way of grouping a variety of classes and/or interfaces together. The grouping is usually done according to functionality. By organizing our classes into packages we can achieve the following benefits:

- The classes contained in the packages can be easily reused.
- Packages, classes can be unique compared with classes in other packages. That is two classes in two different packages can have same name. They may be referred by their fully qualified name, corresponding package name & the class name.
- Packages provide a way to "hide" classes thus protecting other programs or packages from accessing classes.
- Java packages are therefore classified into two types. The first category is known as Java API packages and second is user defined packages.

Frequently Used API Packages:

Package Name	Content
java.lang	Language support classes. Java compiler itself uses classes & therefore they are automatically imported. They include classes for primitive types, String, math functions, threads & exceptions.
java.util	Language utility classes such as vectors, hash tables, random numbers, date etc.

java.io	Input/output support classes. They provide facilities for input & output of data.
java.awt	Set of classes for implementing graphical user interface.
java.net	Classes for networking. They include classes for communicating with local computers as well as internet servers.
Java.applet	Classes for creating & implementing applet.

Defining a Package:

To create a package is quite easy: simply include a **package** command as the first statement in a Java source file. Any classes declared within that file will belong to the specified package. The **package** statement defines a name space in which classes are stored. If you omit the **package** statement, the class names are put into the default package, which has no name.

This is the general form of the **package** statement:

```
package pkg;
```

Here, *pkg* is the name of the package. For example, the following statement creates a package called **MyPackage**.

```
package MyPackage;
```

Java uses file system directories to store packages. For example, the **.class** files for any classes you declare to be part of **MyPackage** must be stored in a directory called **MyPackage**.

You can create a hierarchy of packages. To do so, simply separate each package.

The general form of a multileveled package statement is shown here:

```
package pkg1[.pkg2[.pkg3]];
```

Here Myclass can directly accessed by using class name or its object.

Another approach is as follows:

Accessing a Package

Java system either using a fully qualified class name or system package can be accessed either using a fully qualified class name or using a shortcut approach through the import statement.

This same approach can be used with user defined package. The general form is:

```
Import package1[.package2][.package3].classname;
```

Here package1 is the name of top level package; package2 is the name of the package that is inside the package1 & so on. We can have any number of hierarchies. Finally the explicit classname is specified.

For example:

```
import firstpackage.secondpackage.Myclass;

Import packagename.*;
```

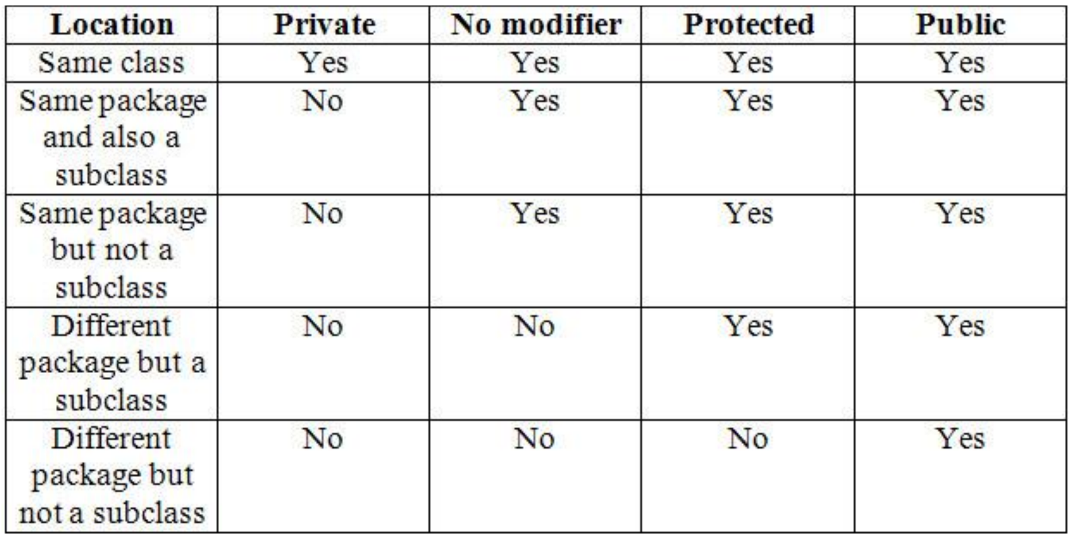
Here packagename may denote a single package or hierarchy of packages. The star indicates that the compiler should search this entire package hierarchy when it encounters its class name.

ALGORITHM:

- Create a package containing various classes & methods in it.
- Define one more class & its members.
- Import the classes & members of the package in the class under development.
- Print the result.
- End of the program.

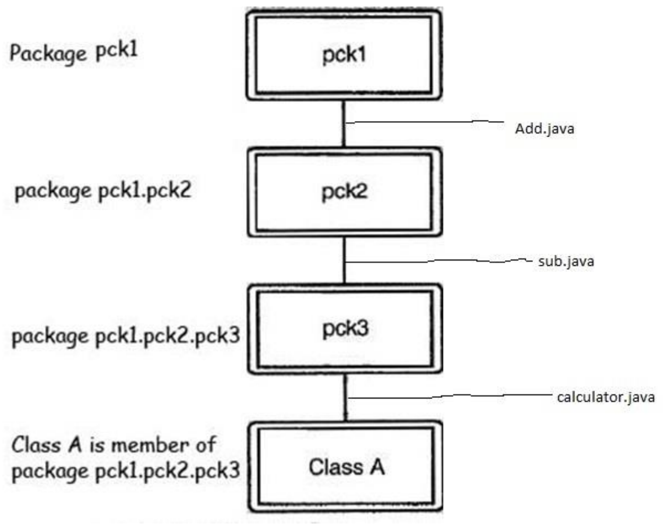
SAMPLE PROGRAMS:

1. Write a program that demonstrates use of packages & import statements. (Simple addition program).
2. Write java program to check all the available access specifier with package and try to prove that the below given table is true.



Location	Private	No modifier	Protected	Public
Same class	Yes	Yes	Yes	Yes
Same package and also a subclass	No	Yes	Yes	Yes
Same package but not a subclass	No	Yes	Yes	Yes
Different package but a subclass	No	No	Yes	Yes
Different package but not a subclass	No	No	No	Yes

3. Write a java program to create the hierarchy like below figure to perform addition and subtraction operation.



QUESTIONS

1. Explain the concept of static import with example.
2. Raise the various types of difficulties faced by the programmer if we do not have the concept of package in Java.
3. Name few classes which are the part of java.lang package.
4. What is the basic difference between public class and class?

Week 7
Exception Handling

AIM: Write Java programs to demonstrate use of exception and user defined exception.

OBJECTIVE:

- Learning various types of exception in java.
- Learn how to handle exceptions using try-catch block.

THEORY:

Common Exceptions

Exception	**Meaning**
ArithmeticException	Arithmetic error, such as divide-by-zero.
ArrayIndexOutOfBoundsException	Array index is out-of-bounds.
ArrayStoreException	Assignment to an array element of an incompatible type
EnumConstantNotPresentException	An attempt is made to use an undefined enumeration value.
IllegalArgumentException	Illegal argument used to invoke a method.
IllegalMonitorStateException	Illegal monitor operation, such as waiting on an unlocked thread
IllegalStateException	Environment or application is in incorrect state.
IllegalThreadStateException	Requested operation not compatible with current thread state.
IndexOutOfBoundsException	Some type of index is out-of-bounds.
NegativeArraySizeException	Array created with a negative size
NullPointerException	Invalid use of a null reference
NumberFormatException	Invalid conversion of a string to a numeric format.

The try Block:

The try block can have one or more statements that could generate an exception. If anyone statement generates an exception, the remaining statements in the block are skipped & execution jumps to the catch block that is placed next to the try block.

The catch Block(s):

The catch block too have one or more statements that are necessary to process the exception. The catch statement works like a method definition. Every try statement should have at least one catch statement otherwise compilation error will occur.

The catch statement is passed a single parameter, which is reference to the exception object thrown by the try block.

```
try{
. . .
}
catch (. . .){
. . .
}
catch (. . .){
. . .
} . . .
```

The finally Block:

Java supports another statement known as finally statement that can be used to handle an exception that is not caught by any previous catch statements. Finally block can be used to handle any exception generated within a try block. It may be added immediately after try block or after the last catch block. When a finally block is defined, this is guaranteed to execute, regardless of whether or not in exception is thrown.

User Defined Exception

User-defined exception class is used to show custom messages to the end users. It is used to hide the other exceptions like null pointer exception; arithmetic exception etc. User-defined exception class is so common that it will be used in every project.

Only one step will be enough to create user-defined exception class i.e. extend the Exception class.

```
/**
 * To create user defined exception
 * one step is enough
 * that is just to extend Exception class
 */
public class UserDefinedException extends Exception{

}
```

```
/**
 * This example is used to show
 * How to Create User Defined Exception class
 *
 * @author Examples On Java
 *
 */
public class UserDefinedExceptionExample {

      public static void main(String[] args) throws Exception {

            try {
                  /**
                   * Throw user defined exception
                   */
                  throw new UserDefinedException();

            } catch (Exception e) {
                  e.printStackTrace();
            }
      }
}
```

ALGORITHM:

- Define a class; define variables and members of class.
- Write down the code which might throw an exception at a time of program execution in try block which throws exceptions.
- The user defined exception class should extend from Exception class.
- Write down multiple catch statements for handling all exceptions thrown by try block.
- End of the program.

SAMPLE PROGRAMS:

1. Write a different java program for generating following types of exception
   a. NullPointerException
   b. ArrayIndexOutOfBoundException
   c. ArithmeticException
   d. NumberFormatException
   e. StringIndexOutOfBoundException

2. Write a program to demonstrate user define exception.
3. Write a java program to demonstrate the use of nested try block.
4. Can a method be overloaded on the basis of an exception? Explain with an example.

QUESTIONS

1. What is chained exception? Explain with example.
2. Explain the difference between compile time and ruin time exception.
3. Is it necessary that each try block must be followed by the catch block? Explain with example.
4. Is there any cases where finally block will not be executed?
5. What is OutOfMemoryError in Java?
6. What are the best practices for exception handling?

Week 8 & 9

Multithreading

AIM: Write a Java program to implement concept of multithreading.

OBJECTIVE:

- Understand the concept of multithreading & thread life cycle.
- Using various methods of thread class.
- Creating multi-threaded program.

THEORY:

Creating a Thread:

Java defines two ways in which this can be accomplished:

- You can implement the Runnable interface.
- You can extend the Thread class itself.

Create Thread by Implementing Runnable:

The easiest way to create a thread is to create a class that implements the **Runnable** interface. To implement Runnable, a class needs to only implement a single method called **run ( )**, which is declared like this: public void run ();

You will define the code that constitutes the new thread inside run() method. It is important to understand that run () can call other methods, use other classes, and declare variables, just like the main thread can.

After you create a class that implements Runnable, you will instantiate an object of type Thread from within that class. Thread defines several constructors. The one that we will use is shown here:

```
Thread (Runnable threadOb, String threadName);
```

Here, *threadOb* is an instance of a class that implements the Runnable interface and the name of the new thread is specified by *threadName*.

After the new thread is created, it will not start running until you call its **start ( )** method, which is declared within Thread. The start ( ) method is shown here: void start ();

Create Thread by Extending Thread:

The second way to create a thread is to create a new class that extends **Thread**, and then to create an instance of that class.

The extending class must override the **run ( )** method, which is the entry point for the new thread. It must also call **start ( )** to begin execution of the new thread.

Thread Methods:

Following is the list of important methods available in the Thread class.

SN	Methods with Description
1	**public void start ()** Starts the thread in a separate path of execution, then invokes the run () method on this Thread object.
2	**public void run ()** If this Thread object was instantiated using a separate Runnable target, the run () method is invoked on that Runnable object.
3	**public final void setName(String name)** Changes the name of the Thread object. There is also a getName () method for retrieving the name.
4	**public final void setPriority(int priority)** Sets the priority of this Thread object. The possible values are between 1 and 10.
5	**public final void setDaemon(boolean on)** A parameter of true denotes this Thread as a daemon thread.
6	**public final void join(long millisec)** The current thread invokes this method on a second thread, causing the current thread to block until the second thread terminates or the specified number of milliseconds passes.
7	**public void interrupt ()** Interrupts this thread, causing it to continue execution if it was blocked for any reason.
8	**public final boolean isAlive()** Returns true if the thread is alive, which is any time after the thread has been started but before it runs to completion.

The previous methods are invoked on a particular Thread object. The following methods in the Thread class are static. Invoking one of the static methods performs the operation on the currently running thread.

1. **public static void sleep (long millisec):** Causes the currently running thread to block for at least the specified number of milliseconds
2. **public static void yield ():**Causes the currently running thread to yield to any other threads of the same priority that are waiting to be scheduled
3. **public static Thread currentThread ():**Returns a reference to the currently running thread, which is the thread that invokes this method

ALGORITHM for Normal Thread Program:

- Create thread by extending thread class.
- Create as many threads as you want & declare their variables & methods.
- Declare a class which contains main thread.
- From this class, start & control the execution of other threads by creating their objects.
- Print the result.
- End of the program.

ALGORITHM for Inter Process Thread Communication:

- Creating a class q and declare variable n as int and declare value set (Boolean) as false.
- Create synchronized method in get ().
- Create another synchronized method void put (int n).
- If (value set) then go to try and catch.
- Inside catch call notify () this tells produces that it away put more about in the queue.
- Create a class producer, that implements runnable.
- Create object q. create a constructor producer where object is passed.
- Create a function run (), initialize and declare i= 0.
- Create a class consumer that implements runnable.
- Create constructor consumer, pass the object and create a function run ().
- Create a class multi2 inside the main function and create object q.

SAMPLE PROGRAMS:

1. Write a program for creating three threads randomly using following methods:
    a. By extending Thread class
    b. By implementing Runnable interface
2. Write a thread program to demonstrate isAlive () and join ().

3. Write a program to create a new thread by extending a thread class.
    a. Get the current thread name
    b. Set the highest priority to the newly created thread
    c. Pause a thread for 2.5 seconds.
    d. Check whether the thread is in running state or not.
    e. Verify your newly created thread must be completed first before your main thread is completed.
4. Write a Java program to create deadlock type situation using multiple threads. Also provide solution to come out from the deadlock.
5. Write a Java Program to implement the concept of inter thread communication with very famous producer consumer problem.

QUESTIONS

1. Why thread synchronization is required?
2. Explain thread life cycle in detail.
3. What is the basic difference between sleep (), yield () and wait ()?
4. What is the role of Thread Scheduler in Java?
5. Explain the difference between preemptive scheduling and slicing in Thread.
6. Can we start thread twice?
7. How you can define Race condition?
8. What happen if exception occurs in Thread program?
9. Why wait () and notify () are called only from synchronized block?
10. What is volatile variable in Java?
11. How daemon thread is differ from normal thread?