**Hyderabad Karnataka Education Society’s**

**PoojyaDodappaAppa Engineering College, Kalaburagi**

**(An Autonomous Institution)**

**Aiwan-E-Shahi Area, Kalaburagi, Karnataka 585102**

**Department of Computer Science & Engineering**

****

**OBJECT ORIENTED PROGRAMMING WITH JAVA LAB(22CS33)**

**LAB MANUAL 2023-2024**

**Prepared By:**

**Dr.Priyadarshani Patil**

**Smt.Smitha Padshetty**

**Pankaj R Chapparbandi**

**Subject code : IPCC(22CS33) Credits :4**

**Hours/Week : 3 :0 :2 C.I.E : 50 Marks**

**Laboratory Experiments**

1. Write a Java Program to demonstrate the creation of class for student information.

2. Write a program in Java for String handling which performs the following:

i) Checks the capacity of String Buffer objects.

ii) Reverses the contents of a string given on console and converts the resultant string in upper case.

iii) Reads a string from console and appends it to the resultant string of ii.

3 a. Write a JAVA Program to demonstrate Constructor Overloading and Method Overloading.

b. Write a JAVA Program to implement Inner class and demonstrate its Access Protections.

4. a. Write and execute a JAVA Program to demonstrate Inheritance.(single level and multilevel)

b. Write and execute a JAVA program to demonstrate method overriding.

5. Write a JAVA Program to demonstrate multiple inheritance using interfaces to calculate the area of a rectangle and triangle.

6. Write a JAVA program to create and import packages in JAVA.

7. Write a JAVA program which has

i. A Class called Account that creates account with 500Rs minimum balance, a deposit() method to deposit amount, a withdraw() method to withdraw amount and also throws Less Balance Exception if an account holder tries to withdraw money which makes the balance become less than 500 Rs. Amount (\_\_\_Rs) is not valid.

ii. A Class which creates 2 accounts, both account deposit money and one account tries to withdraw more money which generates a Less Balance Exception take appropriate action for the same.

iii. A Class which creates 2 accounts, both account deposit money and one account tries to withdraw more money which generates a Less Balance Exception take appropriate action for the same.

8. Write a Java program to implement multithreading in JAVA which demonstrate built in methods available for thread.

9. Write a JAVA program using Synchronized Threads, which demonstrates Producer Consumer concept.

10. Write a JAVA applet program to create a basic Applet having buttons, text area GUI controls to add & subtract two nos. Use appropriate event listeners.

**1. Laboratory Experiments:** Laboratory experiments are framed on par with theory courses to enhance the knowledge and adopt the theoretical concept in practical. Laboratory experiments are framed which covers the entire syllabus Lab notebook are maintained. Students are provided Lab manuals for reference during practical sessions.

**2. Open Ended Experiments :**One or two open ended experiments are given to student based on real life applications of the theoretical concept studies.

**3. Performance Evaluation of Students in Lab courses :**Student performance in laboratory is evaluated based on grading to each experiments/programs, lab records , continuous internal assessment test with viva voce which will be conducted at the end of each lab session.

|  |  |
| --- | --- |
| **SL No** | **Rubrics adopted for evaluation of lab courses** |
| 1 | Problem Solving approach |
| 2 | Methodology/Tool used |
| 3 | Implementation |
| 4 | Data Interpretation or analysis |

## **What is Eclipse?**

In the context of computing, Eclipse is an integrated development environment (IDE) for developing applications using the Java programming language and other programming languages such as C/C++, Python, PERL, Ruby etc.

The Eclipse platform which provides the foundation for the Eclipse IDE is composed of plug-ins and is designed to be extensible using additional plug-ins. Developed using Java, the Eclipse platform can be used to develop rich client applications, integrated development environments and other tools. Eclipse can be used as an IDE for any programming language for which a plug-in is available.

The Java Development Tools (JDT) project provides a plug-in that allows Eclipse to be used as a Java IDE, PyDev is a plugin that allows Eclipse to be used as a Python IDE, C/C++ Development Tools (CDT) is a plug-in that allows Eclipse to be used for developing application using C/C++, the Eclipse Scala plug-in allows Eclipse to be used an IDE to develop Scala applications and PHPeclipse is a plug-in to eclipse that provides complete development tool for PHP.

**Downloading and installing**

The main page is <http://www.eclipse.org/> ; start there for everything.

Click on "downloads" in either the main panel or the navigation panel; this will bring you to a page where you can select either the Main Eclipse download site or one of several mirrors. I suggest that you get the "Latest Release." I'm using **Release 3.0.1**, which is the latest release at the time I'm writing this.

Installation is simple, but you must already have the Java SDK installed. Unzip the file and double-click **eclipse.exe**.

**Getting started**

The following is slightly modified from <http://www.eclipse.org/eclipse/faq/eclipse-faq.html#users_4>:

|  |
| --- |
| **How do I write a simple "Hello World" program?**  To write a "Hello World" program follow these steps:   1. Start Eclipse. 2. Create a new Java Project:    1. **File->New->Project**.    2. Select "**Java**" in the category list.    3. Select "**Java Project**" in the project list. Click "**Next**".    4. Enter a project name into the **Project name** field, for example, "**Hello World Project**".    5. Click "**Finish**"--It will ask you if you want the Java perspective to open. (You do.) 3. Create a new Java class:    1. Click the "**Create a Java Class**" button in the toolbar. (This is the icon below "**Run**" and "**Window**" with a tooltip that says "New Java Class.")    2. Enter "**HelloWorld**" into the **Name** field.    3. Click the checkbox indicating that you would like Eclipse to create a "**public static void main(String[] args)**" method.    4. Click "**Finish**". 4. A Java editor for **HelloWorld.java** will open. In the main method enter the following line. **System.out.println("Hello World");** 5. Save using **ctrl-s**. This automatically compiles **HelloWorld.java**. 6. Click the "**Run**" button in the toolbar (looks like a little man running). 7. You will be prompted to create a Launch configuration. Select "**Java Application**" and click "**New**". 8. Click "**Run**" to run the Hello World program. The console will open and display "**Hello World**". |

**Introduction to Java programming**

\*JAVA was developed by Sun Microsystems Inc in 1991, later acquired by Oracle Corporation. It was developed by James Gosling and Patrick Naughton. It is a simple programming language.  Writing, compiling and debugging a program is easy in java.  It helps to create modular programs and reusable code.\*

## Java platform overview

Java technology is used to develop applications for a wide range of environments, from consumer devices to heterogeneous enterprise systems. In this section, get a high-level view of the Java platform and its components.

* Java is an object-oriented programming language developed by Sun Microsystems and released in 1995.
* Java was originally developed by **James Gosling** at **Sun Microsystems** (which has since merge into Oracle Corporation).
* Java programs are platform independent which means they can be run on any operating system with any type of processor as long as the Java interpreter is available on that system.
* Java code that runs on one platform does not need to be recompiled to run on another platform, it’s called **write once, run anywhere(WORA)**.
* Java Virtual Machine (JVM) executes Java code, but is written in platform specific languages such as C/C++/ASM etc. JVM is not written in Java and hence **cannot be platform independent** and Java interpreter is actually a part of JVM.

## Types of Java Applications

1. **Web Application –** Java is used to create server-side web applications. Currently, servlet, jsp, struts, jsf etc. technologies are used.
2. **Standalone Application –** It is also known as desktop application or window-based application. An application that we need to install on every machine or server such as media player, antivirus etc. AWT and Swing are used in java for creating standalone applications.
3. **Enterprise Application –** An application that is distributed in nature, such as banking applications etc. It has the advantage of high level security, load balancing and clustering. In java, EJB is used for creating enterprise applications.
4. **Mobile Application –** Java is used to create application softwares for mobile devices. Currently Java ME is used for creating applications for small devices, and also Java is programming language for Google Android application development.

## Different Editions of Java Technology

* **Java SE**– Java SE or Java Standard Edition provides tools and API’s that you can use to create server applications, desktop applications, and even applets. These programs developed using Java SE can be run on almost every popular operating system, including Linux, Macintosh, Solaris, and Windows.
* **JEE** – Based on the foundation framework of the standard edition, Java Enterprise Edition helps in web application service, component model and enterprise class service oriented architecture (SOA).
* **JME** – Java Micro Edition or JME for short is an accumulation of Java APIs that are used for the development of software for devices like mobile phones, PDAs, TV set-top boxes, game programming. The platform of micro edition generally consists of an easy user interface, a robust security model and a wide variety of built-in networks for running Java based application.)

## Main Features of JAVA

### Java is a platform independent language

Compiler(javac) converts source code (.java file) to the byte code(.class file). As mentioned above, JVM executes the bytecode produced by compiler. This byte code can run on any platform such as Windows, Linux, Mac OS etc. Which means a program that is compiled on windows can run on Linux and vice-versa. Each operating system has different JVM, however the output they produce after execution of bytecode is same across all operating systems. That is why we call java as platform independent language.

### Java is an Object Oriented language

Object oriented programming is a way of organizing programs as collection of objects, each of which represents an instance of a class.

4 main concepts of Object Oriented programming are:

1. Abstraction
2. Encapsulation
3. Inheritance
4. Polymorphism

portable.

## Java terminology

Before we start learning Java, lets get familiar with common java terms.

**Java Virtual Machine (JVM)**

This is generally referred as JVM. Before, we discuss about JVM lets see the phases of program execution. Phases are as follows: we write the program, then we compile the program and at last we run the program.

1) Writing of the program is of course done by java programmer like you and me.

2) Compilation of program is done by javac compiler, javac is the primary java compiler included in java development kit (JDK). It takes java program as input and generates java bytecode as output.

3) In third phase, JVM executes the bytecode generated by compiler. This is called program run phase.

So, now that we understood the primary function of JVM is to execute the bytecode produced by compiler**. Each operating system has different JVM, however the output they produce after execution of bytecode is same across all operating systems.** That is why we call java **bytecode**

As discussed above, javac compiler of JDK compiles the java source code into bytecode so that it can be executed by JVM. The bytecode is saved in a .class file by compiler

1. Write a Java Program to demonstrate the creation of class for student information.

**CLASSES and OBJECTS**

This section focused on Java object and class, In this lesson our major focus on Objects and Classes.

**What is Object?**

In real-world an entity that has state and its behavior is known as an object.

For Example:

* A Car is an object. It has states (name, color, model) and its behavior (changing gear, applying brakes).
* A Pen is an object. Its name is Parker,color is silver etc. known as its state. It is used to write, so writing is its behavior.

In real world object and software object have conceptually similar characteristics. In terms of object-oriented programming, software objects also have a state and behavior.

**What is a class?**

* A class is a template or blueprint that is used to create objects.
* Class representation of objects and the sets of operations that can be applied to such objects.
* Class consists of Data members and methods.

Primary purpose of a class is to held data/information. This is achieved with attributes which is also known as data members.

The member functions determine the behavior of the class i.e. provide definition for supporting various operations on data held in form of an object.

**Defining a Class in Java**

Syntax:

publicclassclass\_name

{

DataMembers;

Methods;

}

Example:

publicclassCar

{

public:

double color;// Color of a car

double model;// Model of a car

}

* Private, Protected, Public are called visibility labels.
* The members that are declared private can be accessed only from within the class.
* Public members can be accessed from outside the class also.

### Class Members

Data and functions are members.

Data Members and methods must be declared within the class definition.

Example:

publicclassCube

{

int length;// length is a data member of class Cube

int breadth;// breadth is a data member of class Cube

intlength ;// Error redefinition of length

};

Program:

import java.util.Scanner;

public class Student

{

public String vtuno;

public String fullName;

public intsem;

public String branchcode;

public String address;

public Student()

{

Scanner scanner=new Scanner(System.in);

System.out.print("VTU NO:");

vtuno=scanner.nextLine();

System.out.print("FullName:");

fullName=scanner.nextLine();

System.out.print("Address:");

address=scanner.nextLine();

System.out.print("branch:");

branchcode=scanner.nextLine();

System.out.print("semester:");

sem=scanner.nextInt();

}

public void show()

{

System.out.println("Entered Data");

System.out.println("VTU No. "+vtuno);

System.out.println("FullName:"+fullName);

System.out.println("sem:"+sem);

System.out.println("Branch:"+branchcode);

System.out.println("Address:"+address);

}

public static void main(String[] args)

{

Student std=new Student();

std.show();

}

}

**Output:**

VTU NO:3PD22CS000

FullName:XYZ

Address:KLBG

branch:cse

semester:3

Entered Data

VTU No. 3PD22CS000

FullName:XYZ

sem:3

Branch:cse

Address:KLBG

2. Write a program in Java for String handling which performs the following:

i) Checks the capacity of String Buffer objects.

ii) Reverses the contents of a string given on console and converts the resultant string in upper case.

iii) Reads a string from console and appends it to the resultant string of ii.

**Java String class methods**

The java.lang.String class provides a lot of methods to work on string. By the help of these methods, we can perform operations on string such as trimming, concatenating, converting, comparing, replacing strings etc.

Java String is a powerful concept because everything is treated as a string if you submit any form in window based, web based or mobile application.

**Java StringBuffer class**

Java StringBuffer class is used to create mutable (modifiable) string. The StringBuffer class in java is same as String class except it is mutable i.e. it can be changed.

### Important Constructors of StringBuffer class

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| StringBuffer() | creates an empty string buffer with the initial capacity of 16. |
| StringBuffer(String str) | creates a string buffer with the specified string. |
| StringBuffer(int capacity) | creates an empty string buffer with the specified capacity as length. |

### Java String toUpperCase() and toLowerCase() method

The java string toUpperCase() method converts this string into uppercase letter and string toLowerCase() method into lowercase letter.

String s="Sachin";

System.out.println(s.toUpperCase());//SACHIN

System.out.println(s.toLowerCase());//sachin

System.out.println(s);//Sachin(no change in original)

### What is mutable string

A string that can be modified or changed is known as mutable string. StringBuffer and StringBuilder classes are used for creating mutable string.

### a) StringBufferappend() method

The append() method concatenates the given argument with this string.

**class** StringBufferExample{

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

StringBuffer sb=**new** StringBuffer("Hello ");

sb.append("Java");//now original string is changed

System.out.println(sb);//prints Hello Java

}

}

### 2) StringBufferinsert() method

The insert() method inserts the given string with this string at the given position.

**class** StringBufferExample2{

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

StringBuffer sb=**new** StringBuffer("Hello ");

sb.insert(1,"Java");//now original string is changed

System.out.println(sb);//prints HJavaello

}

}

### 3) StringBufferreplace() method

The replace() method replaces the given string from the specified beginIndex and endIndex.

**class** StringBufferExample3{

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

StringBuffer sb=**new** StringBuffer("Hello");

sb.replace(1,3,"Java");

System.out.println(sb);//prints HJavalo

}

}

### 4) StringBufferdelete() method

The delete() method of StringBuffer class deletes the string from the specified beginIndex to endIndex.

**class** StringBufferExample4{

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

StringBuffer sb=**new** StringBuffer("Hello");

sb.delete(1,3);

System.out.println(sb);//prints Hlo

}

}

### 5) StringBufferreverse() method

The reverse() method of StringBuilder class reverses the current string.

**class** StringBufferExample5{

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

StringBuffer sb=**new** StringBuffer("Hello");

sb.reverse();

System.out.println(sb);//prints olleH

}

}

### 6) StringBuffercapacity() method

The capacity() method of StringBuffer class returns the current capacity of the buffer. The default capacity of the buffer is 16. If the number of character increases from its current capacity, it increases the capacity by (oldcapacity\*2)+2. For example if your current capacity is 16, it will be (16\*2)+2=34.

**class** StringBufferExample6{

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

StringBuffer sb=**new** StringBuffer();

System.out.println(sb.capacity());//default 16

sb.append("Hello");

System.out.println(sb.capacity());//now 16

sb.append("java is my favourite language");

System.out.println(sb.capacity());//now (16\*2)+2=34

 i.e (oldcapacity\*2)+2

}

}

Program:

import java.util.Scanner;

public class StringOps

{

public static void main(String [] args)

{

StringBuffer s=new StringBuffer();

String str2;

Scanner input = new Scanner(System.in);

s.append("Java Programming");

System.out.println("The current capacity of StringBuffer is:"+s.capacity());

System.out.println("Initial string without append:"+s);

System.out.println("Enter a string to append:");

str2=input.nextLine();

s.append(" "+str2);

System.out.println("Initial string with append:"+s);

s.reverse();

System.out.println("Initial string after reversing:"+s);

inti=0;

do

{

s.replace(i,i+1,s.substring(i,i+1).toUpperCase());

i=i+1;

}

while(i>0 &&i<s.length());

System.out.println("After reversing and converting to upper case:"+s);

}

}

**Output**:

The current capacity of String Buffer is:16

Initial string without append: Java Programming

Enter a string to append:

Java Programming Language

Initial string with append: Java Programming Java Programming Language

Initial string after reversing: egaugna LgnimmargorPavaJgnimmargorPavaJ

After reversing and converting to upper case: EGAUGNAL GNIMMARGORP AVAJ GNIMMARGORP AVAJ

3 a. Write a JAVA Program to demonstrate Constructor Overloading and Method Overloading.

b. Write a JAVA Program to implement Inner class and demonstrate its Access Protections.

Program 3a:

import java.util.Scanner;

class Prog5 {

public int a, b;

Prog5() {

Scanner S1 = new Scanner(System.in);

System.out.println("Enter any 2 integers:");

a = S1.nextInt();

b = S1.nextInt();

}

void display() {

System.out.println("Addition=" + (a + b));

System.out.println("Subtraction=" + (a - b));

System.out.println("Multiplication=" + (a \* b));

System.out.println("Division=" + ((float) a / b)); // Convert one operand to float to get float division

}

Prog5(float a1, float b1) {

System.out.println("Addition=" + (a1 + b1));

System.out.println("Subtraction=" + (a1 - b1));

System.out.println("Multiplication=" + (a1 \* b1));

System.out.println("Division=" + (a1 / b1));

}

void display(int x) {

System.out.println("Square of " + x + " is " + (x \* x));

}

}

class Program3rd {

public static void main(String[] args) {

Scanner S1 = new Scanner(System.in);

System.out.println("\nArithmetic operations on float");

System.out.println("Enter any two float values");

float a1 = S1.nextFloat();

float a2 = S1.nextFloat();

Prog5 arth1 = new Prog5(a1, a2);

System.out.println("\nArithmetic operations on integers");

Prog5 arth2 = new Prog5();

System.out.println("\nEnter number to find Square");

int x = S1.nextInt();

arth1.display(x);

}

}

Program3b:

class Outer

{

private int x=10;

protected int z=30; class inner

{

private int x=20;

protected int z=85;

}

public static void main(String args[])

{

Outer obj1=new Outer();

inner obj2=new Outer().new inner();

System.out.println("Through Outer Class, x = "+obj1.x);

System.out.println("Through Inner Class, x = "+obj2.x);

} }

class Prog5b

{

public static void main(String args[])

{

Outer ob1=new Outer();

Outer.inner ob2=new Outer().new inner();

System.out.println("Through Different Class, Outer's protected z = "+ob1.z);

System.out.println("Through Different Class, Inner's protected z = "+ob2.z);

}

}

**Output :**

Arithmetic operations on float

Enter any two float values

2.0145

5.2136

Addition=7.2281

Subtraction=-3.1991003

Multiplication=10.502797

Division=0.38639325

Arithmetic operations on integers

Enter any 2 integers:

2

3

Enter number to find Square

6

Square of 6 is 36

4. a. Write and execute a JAVA Program to demonstrate Inheritance.(single level and multilevel)

b. Write and execute a JAVA program to demonstrate method overriding.

# Java - Inheritance

Inheritance can be defined as the process where one class acquires the properties (methods and fields) of another. With the use of inheritance the information is made manageable in a hierarchical order.

The class which inherits the properties of other is known as subclass (derived class, child class) and the class whose properties are inherited is known as superclass (base class, parent class).

## extends Keyword

**extends** is the keyword used to inherit the properties of a class. Following is the syntax of extends keyword.

**Syntax**

class Super {

.....

}

class Sub extends Super {

.....

.....

}

## Sample Code

Following is an example demonstrating Java inheritance. In this example, you can observe two classes namely Calculation and My\_Calculation.

Using extends keyword, the My\_Calculation inherits the methods addition() and Subtraction() of Calculation class.

**Types of inheritance in Java: Single, Multiple, Multilevel**

Below are Various types of inheritance in Java. We will see each one of them one by one with the help of examples and flow diagrams.

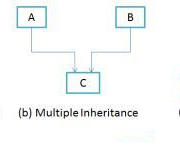
### 1) Single Inheritance

**Single inheritance** is damn easy to understand. When a class extends another one class only then we  call it a single inheritance. The below flow diagram shows that class B extends only one class which is A. Here A is a **parent class** of B and B would be  a **child class** of A.

### [Single Inheritance](https://beginnersbook.com/wp-content/uploads/2013/05/Single-Inheritance.png)

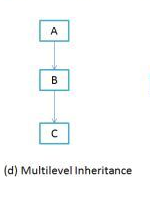
### 2) Multiple Inheritance

“**Multiple Inheritance**” refers to the concept of one class extending (Or inherits) more than one base class. The inheritance we learnt earlier had the concept of one base class or parent. The problem with “multiple inheritance” is that the derived class will have to manage the dependency on two base classes.



### 3) Multilevel Inheritance

**Multilevel inheritance** refers to a mechanism in OO technology where one can inherit from a derived class, thereby making this derived class the base class for the new class. As you can see in below flow diagram C is subclass or child class of of B and B is a child class of A



Program4a:

class sup

{

int x;

sup(int x)

{

this.x=x;

}

void display()

{

System.out.println("sup+x = " +x);

}

}

class supr extends sup

{

int y;

supr(intx,int y)

{

super(x);

this.y=y;

}

void display()

{

System.out.println("sup-x = "+x); System.out.println("sup-y = "+y);

}

}

class prog4a

{

public static void main(String args[])

{

supr s1 = new supr(100,200); s1.display();

}

}

**Output** :

sup-x = 100

sup-y = 200

Program4b:

class Language {

public void displayInfo() {

System.out.println("Common English Language");

}

}

class Java extends Language {

@Override

public void displayInfo() {

System.out.println("Java Programming Language");

}

}

class MainMain {

public static void main(String[] args) {

// create an object of Java class

Java j1 = new Java();

j1.displayInfo();

// create an object of Language class

Language l1 = new Language();

l1.displayInfo();

}

}

**Output :**

Java Programming Language

Common English Language

5. Write a JAVA Program to demonstrate multiple inheritance using interfaces to calculate the area of a rectangle and triangle.

# Java - Interfaces

An interface is a reference type in Java. It is similar to class. It is a collection of abstract methods. A class implements an interface, thereby inheriting the abstract methods of the interface.

Along with abstract methods, an interface may also contain constants, default methods, static methods, and nested types. Method bodies exist only for default methods and static methods.

Writing an interface is similar to writing a class. But a class describes the attributes and behaviors of an object. And an interface contains behaviors that a class implements.

Unless the class that implements the interface is abstract, all the methods of the interface need to be defined in the class.

An interface is similar to a class in the following ways −

* An interface can contain any number of methods.
* An interface is written in a file with a **.java** extension, with the name of the interface matching the name of the file.
* The byte code of an interface appears in a **.class** file.
* Interfaces appear in packages, and their corresponding bytecode file must be in a directory structure that matches the package name.

However, an interface is different from a class in several ways, including −

* You cannot instantiate an interface.
* An interface does not contain any constructors.
* All of the methods in an interface are abstract.
* An interface cannot contain instance fields. The only fields that can appear in an interface must be declared both static and final.
* An interface is not extended by a class; it is implemented by a class.
* An interface can extend multiple interfaces.

Program:

interface compute

{

double calculate();

}

class Rectangle implements compute

{

double l,b;

void getvalues()

{

l = 10.5F;

b=7.3F;

}

public double calculate()

{

return (l\*b);

}

}

class TRI extends Rectangle implements compute

{

public double calculate()

{

return (0.5\*b\*l);

}

}

class prog4b

{

public static void main(String[] args)

{

Rectangle R = new Rectangle();

R.getvalues();

System.out.println("Area of Rectangle = " + R.calculate());

TRI T =new TRI();

T.getvalues();

System.out.println("Area of triangle = " + T.calculate());

}}

**Output:**

Area of Rectangle = 76.65000200271606

Area of triangle = 38.32500100135803

6. Write a JAVA program to create and import packages in JAVA.

# Java – Packages

Packages are used in Java in order to prevent naming conflicts, to control access, to make searching/locating and usage of classes, interfaces, enumerations and annotations easier, etc.

A **Package** can be defined as a grouping of related types (classes, interfaces, enumerations and annotations ) providing access protection and namespace management.

Some of the existing packages in Java are −

* **java.lang** − bundles the fundamental classes
* **java.io** − classes for input , output functions are bundled in this package

Programmers can define their own packages to bundle group of classes/interfaces, etc. It is a good practice to group related classes implemented by you so that a programmer can easily determine that the classes, interfaces, enumerations, and annotations are related.

Since the package creates a new namespace there won't be any name conflicts with names in other packages. Using packages, it is easier to provide access control and it is also easier to locate the related classes.

## Creating a Package

While creating a package, you should choose a name for the package and include a **package** statement along with that name at the top of every source file that contains the classes, interfaces, enumerations, and annotation types that you want to include in the package.

The package statement should be the first line in the source file. There can be only one package statement in each source file, and it applies to all types in the file.

Then a folder with the given package name is created in the specified destination, and the compiled class files will be placed in that folder.

Program:

class Square

{

int a, A;

public void inputvalues (int x)

{

a=x;

}

public void displayArea ()

{

A=a\*a;

System.out.println ('Area of square : " +A);

}

}

class Triangle

{

double a, b, A;

public void inputvalues (int x, int y)

{

a=x;

b=y;

public void displayArea ()

{

A=0.5\*a\*b;

System.out. println("Area of Triangle: "+A);

}

}

Class Circle

{

Double r,A;

Public void inputvalues(int x)

{

r=x;

}

Public void displayArea()

{

A=3.14\*r\*r;

System.out.println(“Area of circle:”+A);

}

}

Public class shapeex1

{

Public static void main(String args[])

{

Square S=new Square();

Triangle T=new Triangle();

Circle C=new Circle();

S.inputvalues(10);

T.inputvalues(9,10);

C.inputvalues(12);

S.displayArea();

T.displayArea();

C.displayArea();

}

}

**Output:**

Area of Square: 100

Area of Triangle: 45

Area of Circle: 452.16

7. Write a JAVA program which has

i. A Class called Account that creates account with 500Rs minimum balance, a deposit() method to deposit amount, a withdraw() method to withdraw amount and also throws Less Balance Exception if an account holder tries to withdraw money which makes the balance become less than 500 Rs. Amount (\_\_\_Rs) is not valid.

ii. A Class which creates 2 accounts, both account deposit money and one account tries to withdraw more money which generates a Less Balance Exception take appropriate action for the same.

iii. A Class which creates 2 accounts, both account deposit money and one account tries to withdraw more money which generates a Less Balance Exception take appropriate action for the same.

**Exception Handling in Java**

1. [Exception Handling](https://www.javatpoint.com/exception-handling-in-java)
2. [Advantage of Exception Handling](https://www.javatpoint.com/exception-handling-in-java#exceptionad)
3. [Hierarchy of Exception classes](https://www.javatpoint.com/exception-handling-in-java#exceptionhierarchy)
4. [Types of Exception](https://www.javatpoint.com/exception-handling-in-java#exceptiontypes)
5. [Scenarios where exception may occur](https://www.javatpoint.com/exception-handling-in-java#exceptionscenarios)

The **exception handling in java** is one of the powerful *mechanism to handle the runtime errors* so that normal flow of the application can be maintained.

In this page, we will learn about java exception, its type and the difference between checked and unchecked exceptions.

What is exception?

**Dictionary Meaning:** Exception is an abnormal condition.In java, exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.

What is exception handling?

Exception Handling is a mechanism to handle runtime errors such as ClassNotFound, IO, SQL, Remote etc.

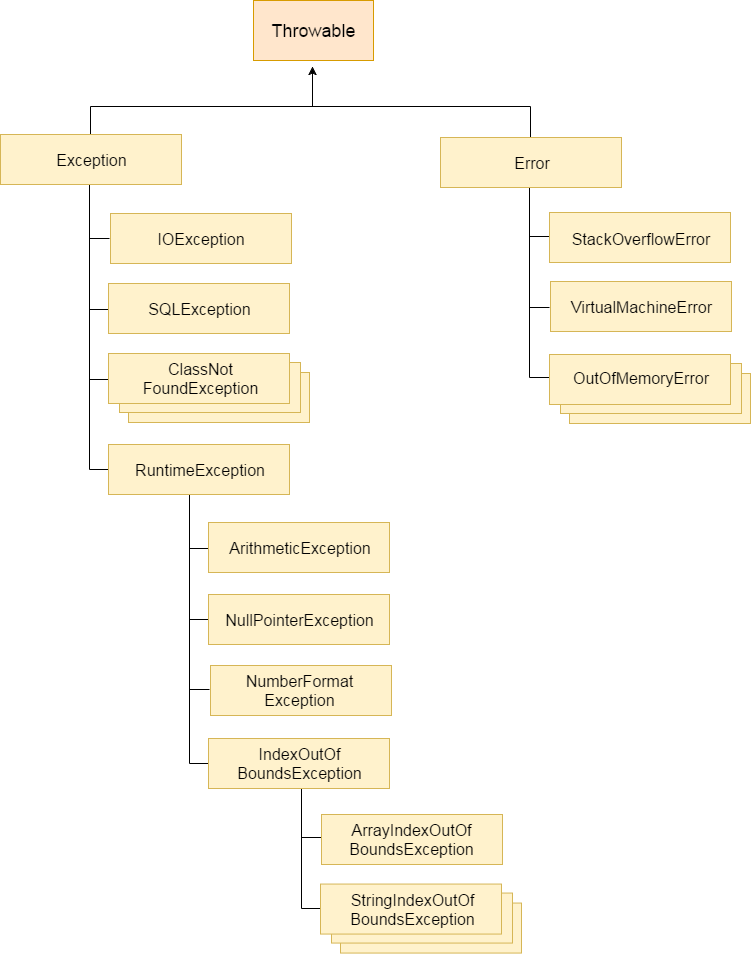
**Advantage of Exception Handling**

The core advantage of exception handling is **to maintain the normal flow of the application**. Exception normally disrupts the normal flow of the application that is why we use exception handling. Let's take a scenario:

1. statement 1;
2. statement 2;
3. statement 3;
4. statement 4;
5. statement 5;
6. statement 6;
7. statement 7;
8. statement 8;
9. statement 9;
10. statement 10;

Suppose there is 10 statements in your program and there occurs an exception at statement 5, rest of the code will not be executed i.e. statement 6 to 10 will not run. If we perform exception handling, rest of the statement will be executed. That is why we use exception handling in java.

**Hierarchy of Java Exception classes**

****

import java.util.Scanner;

class Account {

static intcurBalance = 1000; // Initial balance

int amt;

public Account() {

Scanner s = new Scanner(System.in);

System.out.println("\nEnter the amount:");

amt = s.nextInt();

}

void deposit(int amount) {

curBalance += amount;

System.out.println("Deposit successful.");

}

void withdraw(int amount) {

try {

if ((curBalance - amount) < 500)

throw new LessBalanceException(amount);

curBalance -= amount;

System.out.println("Withdrawal successful.");

} catch (LessBalanceException le) {

System.out.println(le);

}

}

void checkBal() {

System.out.println("\nBalance in your a/c: " + curBalance);

}

}

class LessBalanceException extends Exception {

int am;

public LessBalanceException(int x) {

am = x;

}

public String toString() {

return ("You cannot withdraw the amount, less balance " + am + " now.");

}

}

public class Main3 {

public static void main(String[] args) {

intch;

Scanner s = new Scanner(System.in);

Account a = new Account();

while (true) {

System.out.println("1: Deposit\t2: Withdraw\t3: Check Balance\t4: Exit");

System.out.println("Enter your choice:");

ch = s.nextInt();

switch (ch) {

case 1:

System.out.println("Enter deposit amount:");

intdepositAmount = s.nextInt();

a.deposit(depositAmount);

break;

case 2:

System.out.println("Enter withdrawal amount:");

intwithdrawAmount = s.nextInt();

a.withdraw(withdrawAmount);

break;

case 3:

a.checkBal();

break;

case 4:

return;

default:

System.out.println("Invalid choice");

break;

}

}

}

}

**Output:**

Enter the amount:

5000

1: Deposit 2: Withdraw 3: Check Balance 4: Exit

Enter your choice:

1

Enter deposit amount:

1000

Deposit successful.

1: Deposit 2: Withdraw 3: Check Balance 4: Exit

Enter your choice:

2

Enter withdrawal amount:

3000

You cannot withdraw the amount, less balance 3000 now.

1: Deposit 2: Withdraw 3: Check Balance 4: Exit

Enter your choice:

3

Balance in your a/c: 2000

1: Deposit 2: Withdraw 3: Check Balance 4: Exit

Enter your choice:

4

8. Write a Java program to implement multithreading in JAVA which demonstrate built in methods available for thread.

# Java – Multithreading

Java is a *multi-threaded programming language* which means we can develop multi-threaded program using Java. A multi-threaded program contains two or more parts that can run concurrently and each part can handle a different task at the same time making optimal use of the available resources specially when your computer has multiple CPUs.

By definition, multitasking is when multiple processes share common processing resources such as a CPU. Multi-threading extends the idea of multitasking into applications where you can subdivide specific operations within a single application into individual threads. Each of the threads can run in parallel. The OS divides processing time not only among different applications, but also among each thread within an application.

Multi-threading enables you to write in a way where multiple activities can proceed concurrently in the same program.

## Life Cycle of a Thread

A thread goes through various stages in its life cycle. For example, a thread is born, started, runs, and then dies. The following diagram shows the complete life cycle of a thread.

Following are the stages of the life cycle −

* **New** − A new thread begins its life cycle in the new state. It remains in this state until the program starts the thread. It is also referred to as a **born thread**.
* **Runnable** − After a newly born thread is started, the thread becomes runnable. A thread in this state is considered to be executing its task.
* **Waiting** − Sometimes, a thread transitions to the waiting state while the thread waits for another thread to perform a task. A thread transitions back to the runnable state only when another thread signals the waiting thread to continue executing.
* **Timed Waiting** − A runnable thread can enter the timed waiting state for a specified interval of time. A thread in this state transitions back to the runnable state when that time interval expires or when the event it is waiting for occurs.
* **Terminated (Dead)** − A runnable thread enters the terminated state when it completes its task or otherwise terminates.

## Thread Priorities

Every Java thread has a priority that helps the operating system determine the order in which threads are scheduled.

Java thread priorities are in the range between MIN\_PRIORITY (a constant of 1) and MAX\_PRIORITY (a constant of 10). By default, every thread is given priority NORM\_PRIORITY (a constant of 5).

Threads with higher priority are more important to a program and should be allocated processor time before lower-priority threads. However, thread priorities cannot guarantee the order in which threads execute and are very much platform dependent.

## Create a Thread by Implementing a Runnable Interface

If your class is intended to be executed as a thread then you can achieve this by implementing a **Runnable** interface. You will need to follow three basic steps −

### Step 1

As a first step, you need to implement a run() method provided by a **Runnable** interface. This method provides an entry point for the thread and you will put your complete business logic inside this method. Following is a simple syntax of the run() method −

public void run( )

### Step 2

As a second step, you will instantiate a **Thread** object using the following constructor −

Thread(Runnable threadObj, String threadName);

Where, *threadObj* is an instance of a class that implements the **Runnable** interface and **threadName** is the name given to the new thread.

### Step 3

Once a Thread object is created, you can start it by calling **start()** method, which executes a call to run( ) method. Following is a simple syntax of start() method −

void start();

Program:

class MyThread implements Runnable{

Thread thrd;

MyThread(String name)

{

thrd=new Thread(this,name);

thrd.start();

}

public void run()

{

System.out.println(thrd.getName() +"Starting.");

try {

for(int count=0;count<10;count++)

{

Thread.sleep(400);

System.out.println("In " + thrd.getName() + " count is" + count );

}

}

catch(InterruptedExceptionexc)

{

System.out.println(thrd.getName()+" interrupted");

}

System.out.println(thrd.getName() + " terminating");

}

}

public class JoinThreads{

public static void main(String[] args) {

System.out.println("Main thread starting");

MyThread mt1=new MyThread("Child #1");

MyThread mt2=new MyThread("Child #2");

MyThread mt3=new MyThread("Child #3");

try

{

mt1.thrd.join();

System.out.println("Child #1 joined");

mt2.thrd.join();

System.out.println("Child #2 joined");

mt3.thrd.join();

System.out.println("Child #3 joined");

}

catch(InterruptedExceptionexc)

{

System.out.println("Main thread interrupted");

}

System.out.println("Main thread ending");

}

}

**Output:**

Main thread starting

Child #2Starting.

Child #3Starting.

Child #1Starting.

In Child #1 count is0

In Child #3 count is0

In Child #2 count is0

In Child #1 count is1

In Child #2 count is1

In Child #3 count is1

In Child #1 count is2

In Child #3 count is2

In Child #2 count is2

In Child #3 count is3

In Child #2 count is3

In Child #1 count is3

In Child #2 count is4

In Child #3 count is4

In Child #1 count is4

In Child #3 count is5

In Child #2 count is5

In Child #1 count is5

In Child #1 count is6

In Child #3 count is6

In Child #2 count is6

In Child #1 count is7

In Child #2 count is7

In Child #3 count is7

In Child #2 count is8

In Child #1 count is8

In Child #3 count is8

In Child #1 count is9

In Child #3 count is9

Child #3 terminating

In Child #2 count is9

Child #2 terminating

Child #1 terminating

Child #1 joined

Child #2 joined

Child #3 joined

Main thread ending

9. Write a JAVA program using Synchronized Threads, which demonstrates Producer Consumer concept.

class Consumer extends Thread

{

private Sync sync;

private int number;

public Consumer(Sync c,int number)

{

sync=c;

this.number=number;

}

public void run()

{

int value=0;

for(inti=0;i<5;i++)

{

value=sync.get();

System.out.println("Consumer#" +this.number+ "got:" +value);

try{

sleep((int)(Math.random()\*10000));

}catch(InterruptedException e)

{}

}

}

}

class Producer extends Thread{

private Sync sync;

private int number;

public Producer(Sync c,int number){

sync=c;

this.number=number;

}

public void run(){

for(inti=0;i<5;i++){

sync.put(i);

System.out.println("Produces" +this.number+ "put:" +i);

try{

sleep((int)(Math.random()\*10000));

}catch(InterruptedException e){}

}

}

}

class ProducerConsumerTest{

public static void main(String [] args){

Sync c=new Sync();

Producer p1=new Producer(c,1);

Consumer c1=new Consumer(c,1);

p1.start();

c1.start();

}

}

class Sync{

private int contents;

private boolean available=false;

public synchronized int get(){

while(available==false){

try{

wait();

}catch(InterruptedException e){}

}

available=false;

notifyAll();

return contents;

}

public synchronized void put(int value){

while(available==true){

try{

wait();

}catch(InterruptedException e){}

}

contents=value;

available=true;

notifyAll();

}

}

**Output:**

Produces1put:0

Consumer#1got:0

Produces1put:1

Consumer#1got:1

Produces1put:2

Consumer#1got:2

Consumer#1got:3

Produces1put:3

Produces1put:4

Consumer#1got:4

10.Write a JAVA applet program to create a basic Applet having buttons, text area GUI controls to add & subtract two nos. Use appropriate event listeners.

**Java Applet**

Applet is a special type of program that is embedded in the webpage to generate the dynamic content. It runs inside the browser and works at client side.

**Advantage of Applet**

There are many advantages of applet. They are as follows:

* It works at client side so less response time.
* Secured
* It can be executed by browsers running under many plateforms, including Linux, Windows, Mac Os etc.

**Drawback of Applet**

* Plugin is required at client browser to execute applet.

Program:

import java.applet.Applet;

import java.awt.\*;

import java.awt.event.\*;

public class CalculatorApplet extends Applet implements ActionListener {

TextField num1Field, num2Field, resultField;

Button addButton, subtractButton;

public void init() {

// Set layout

setLayout(new GridLayout(4, 2, 5, 5));

// Create text fields

num1Field = new TextField(10);

num2Field = new TextField(10);

resultField = new TextField(10);

resultField.setEditable(false);

// Create buttons

addButton = new Button("Add");

subtractButton = new Button("Subtract");

// Add components to the applet

add(new Label("Number 1:"));

add(num1Field);

add(new Label("Number 2:"));

add(num2Field);

add(new Label("Result:"));

add(resultField);

add(addButton);

add(subtractButton);

// Add action listeners

addButton.addActionListener(this);

subtractButton.addActionListener(this);

}

public void actionPerformed(ActionEvent e) {

if (e.getSource() == addButton) {

int num1 = Integer.parseInt(num1Field.getText());

int num2 = Integer.parseInt(num2Field.getText());

int result = num1 + num2;

resultField.setText(String.valueOf(result));

} else if (e.getSource() == subtractButton) {

int num1 = Integer.parseInt(num1Field.getText());

int num2 = Integer.parseInt(num2Field.getText());

int result = num1 - num2;

resultField.setText(String.valueOf(result));

}

}

}

/\*

<applet code="CalculatorApplet.class" width="300" height="200"></applet>

\*/