**What is Java?**

Java is a **technology** from Sun MicroSystems developed by James Gosling in 1991 with initial name ‘Oak’. It was designed with the goal to run any kind of device.

It was renamed to **Java** in 1995.

It is divided in three sub technologies

1. Java SE (Standard Edition)
2. Java EE (Enterprise Edition)
3. Java ME (Mobile Edition)

**What are various resources for Java Development?**

<http://java.sun.com>

<http://www.java.com>

<http://developers.sun.com>

**What are different tools required for Java SE?**

1. Java Development Kit (JDK)
2. Java Runtime Environment (JRE)
3. Editor and IDEs (Integrated Development Environment)
   1. Notepad Editor
   2. EditPlus Editor
   3. Kawa IDE
   4. NetBeans IDE
   5. Eclipse
   6. Visual Age
   7. JDeveloper
   8. JCreator
   9. Etc.

**Who is the compiler for Java Programming? Where to find it?**

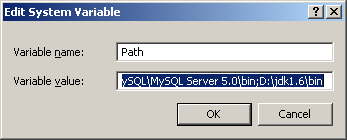
JAVAC.EXE is the compiler for JAVA development found in <jdk folder>\BIN folder

**How to set the PATH for JAVAC?**

PATH is an **environmental variable** to hold a list of folder to search the executable files (.exe, .com and .bat).

Add your folder of JDK into the PATH using the following methods

1. Using command prompt
   1. SET PATH=%PATH%;d:\jdk1.6\bin
2. Using MyComputer
   1. MyComputer 🡪 Properties 🡪 Advanced 🡪 Environmental Variables 🡪 System Variables 🡪 Path 🡪 Edit 🡪 Add your folder name after semicolon



Using IDEs

1. Kawa IDE from Alliare
2. NetBeans 6.0 from Sun

**Basic rules of Java Programming**

1. Java is a case sensitive language
2. Every file will have **.java** extension
3. Java is Object Oriented Language. Every executable program must have a class
4. Every executable class must have an entry point **main()**

public static void main(String args[])

* 1. Here args an array which are treated like objects and each array provides **length** property to get size of array

**Syntactical rules of Java**

1. All keywords and package names must be in lower case
   1. public keyword
   2. int keyword
   3. java.lang package
   4. java.util package
   5. etc.
2. All class names and interface names starts with capital letter
   1. Math class
   2. String class
   3. StringBuilder class
   4. InputStreamReader class
   5. Etc.
3. All field names and method names starts with lower letter
   1. length
   2. printf()
   3. readLine()
   4. etc.

**Hierarchy in Java**

JAR File (Java Archive)/ZIP file

* Package
  + Class/Interface/Enumerator
    - Field/Methods/Static Objects

**JAR file**

* A compressed file like a ZIP file that is created using JAR.EXE tool of JDK

Package

* A collection of similar set of classes
  + java.lang (default)
    - Math, String, System, Thread, Runnable etc.
  + java.io
    - For input/output operators
    - File, FileReader, FileWriter, InputStreamReader, BufferedReader, IOException etc.
  + java.util
    - Provides collections and utilities
    - Stack, Queue, LinkedList
    - Timer, TimerTask
    - Date
    - Scanner
    - Etc.
  + java.awt
    - For graphics programming in Java
    - Button, Checkbox, TextField, Color, Font etc.

**Using the package in our programs**

* Use **import** command to import the class from the package

import <packagename.classname>; - to import only one class

import <packagename.\*>; - to import all classes of a package

Example

import java.util.Scanner;

import java.io.\*;

**General Input/Output Operations**

* Use **System** class with its static objects **out** and **in**
* **out** is a static object of PrintStream class java.io package contained in System class
* **in** is a static object of InputStream class of java.io package contained in System class

Using out we can access method of PrintStream class

* print()
* println()
* printf()

Using in we can read some data using methods of InputStream class

* read()

**Writing First Java Program**

//First.java

class Test

{

public static void main(String args[])

{

System.out.printf("Welcome to Java");

}

}

Save this program using an editor like Notepad into your folder

**Compile the program** using JAVAC.EXE

JAVAC First.java 🡪 Test.class (Byte Code Language file)

**Running the class file**

JAVA Test

**How Java System works**

.java 🡪 JAVAC 🡪 .class 🡪 JVM (Class Loader 🡪 Code Verifier 🡪 JIT (Just-in-time) compiler 🡪 Binary code 🡪 Execution)

**Features of Java**

1. Simple and sober
2. Object Oriented Programming
3. Multi Threaded
4. Java is portable
5. Java is **platform Independent** or Write Once, Run Anywhere
   1. Java Virtual Machine (JVM) is the software responsible for such feature
   2. .java 🡪 JAVAC 🡪 .class (byte code) 🡪 JVM 🡪 Binary 🡪 Execute
   3. JVM is a machine dependent software but make the Java Platform Independent
6. Java is secure
7. Java is Robust
   1. Java provides a big set of classes for almost every purpose
   2. Built-in support of Exception handling makes the Java more powerful
   3. No more memory leakage. Memory is handled a software called as Garbage Collector (GC)

**Data Types in Java**

* Special keyword used to define the kind of data and range of data of some variable or constant or some expression is called as data type
* Can be of two types
  + Primitive Type
  + User Defined Types
* Primary Types can
  + Integrals (All are signed)
    - byte – 1 byte
    - short – 2 byte
    - int – 4 byte
    - long – 8 byte
  + Floatings
    - float – 4 bytes
    - double – 8 bytes
  + characters
    - char -2 byte (Unicode)
  + Booleans
    - boolean – 2 byte
    - Can have **true** or **false** only
* User Defined Types
  + Class
  + Interface
  + Enumerators

**Providing values to the variables**

1. Using Literals
2. Using keyboard Input
3. Using command line argument

**What are the literals?**

* The values that we use from our side is called as literal or literal values or constant value
* Literals can be of different types
  + Integral literals
    - When using numbers without decimal point
    - Default is **int**
      * int num=67;
    - Use l or L with long
      * long k=68L;
  + Floating Literals
    - Default is **double**
    - Use f or F with floats
      * double n=5.6;
      * float k=5.6; //error
      * float k=5.6f; //correct
  + Character Literals
    - Enclosed in single quotes
      * char ch=’A’;
  + Boolean literals
    - Can be **true** or **false** only
      * boolean married=true;
  + String literals
    - Enclosed in double quotes
    - Managed by **String** class
      * String name=”Vikas”;
    - Use + as concatenation operator

**File Naming Rules**

* Program name must be .java
* A program file can have many classes and all or none of the classes can have the main()
* Program can be same as class name or different except **if class is public then both must be same**

**Reading data from Keyboard**

* We can read data using two methods
  + **Scanner** class of java.util package
  + **BufferedReader** class of java.io package

**Creating object of Scanner class**

Scanner sc=new Scanner(System.in);

s

**Use methods of Scanner class**

* String next()
* int nextInt()
* double nextDouble()

Example

Write a program to get name and age of a person check it to be valid voter

import java.util.Scanner;

public class Voter

{

public static void main(String args[])

{

Scanner sc=new Scanner(System.in);

System.out.print("Enter your name : ");

String name=sc.next();

System.out.print("Enter your Age : ");

int age=sc.nextInt();

if(age>=18)

System.out.println("Dear "+name+" you can vote");

else

System.out.printf("Dear %s you cannot vote",name);

}

}

**Reading data using BufferedReader class**

* When a key is pressed from keyboard (System.in), a combination of bits (stream) get produced and send an object of class InputStreamReader that converts the stream into a character
* Those characters get passed to BufferedReader class and stored in buffer area till we hit the enter key
* To read the data from buffer area we need a method called as readLine() which returns a string value
  + String readLine()
* To convert data from string type to some numeric type Java provides special classes called as ***Wrapper classes***

**What is a Wrapper Class?**

* Special classes corresponding to data types that provides advance functioning of that data types called as Wrapper Class

Data Type Wrapper Class

byte Byte

short Short

int Integer

long Long

float Float

double Double

char Character

boolean Boolean

Example 1

Get a number from keyboard and print that number into Decimal, Octal, Binary and Hexa Decimal

import java.util.Scanner;

public class WrapperTest

{

public static void main(String []args)

{

Scanner sc=new Scanner(System.in);

System.out.print("Enter a number : ");

int num=sc.nextInt();

System.out.println("Number is Octal is :"+Integer.toOctalString(num));

System.out.println("Number is Binary is :"+Integer.toBinaryString(num));

System.out.println("Number is Hex is :"+Integer.toHexString(num));

}

}

Wrapper classes help in data conversion from String type to value type

Conversion Formula

datatype variable=wrapperclass.**parse**Datatype(stringdata);

Example

Write a program to get name and age of a person check it to be valid voter using BufferedReader class

import java.io.\*;

public class VoterBR

{

public static void main(String []args) throws IOException

{

InputStreamReader isr=new InputStreamReader(System.in);

BufferedReader br=new BufferedReader(isr);

System.out.print("Enter Name : ");

String name=br.readLine();

System.out.print("Enter Age : ");

int age=Integer.parseInt(br.readLine());

if(age>=18)

System.out.println("Dear "+name+" you can vote");

else

System.out.printf("Dear %s you cannot vote",name);

}

}

**Try**

Write a program to get two numbers and print the biggest one

Case 1: Using Scanner class

Case 2: Using BufferedReader class

**Static Objects**

* An object of a class created in another class as static that can be re-used again and again without re-creating it
* Use **static** keyword to define such objects

Example

Create an object of **Scanner** class and another object of **BufferedReader** in a class as MyClass. Make both the object as static object.

Now use this class in another class General in another file to get name and age of a person and check it as valid voter.

//MyClass.java

import java.util.\*;

import java.io.\*;

class MyClass

{

public static Scanner sc=new Scanner(System.in);

public static BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

}

//General.java

class General

{

public static void main(String arg[]) throws IOException

{

System.out.print("Enter the Name : ");

String name=MyClass.br.readLine();

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

int age=MyClass.sc.nextInt();

if(age>=18)

System.out.println("Dear "+name+" you can vote");

else

System.out.println("Dear "+name+" you cannot vote");

}

}

**Importing Static members of a class**

* Used to avoid writing the class name every time
* Using **import static** keyword

Example

import static java.lang.System.\*;

import java.io.\*;

import static java.lang.System.\*;

class ImportStaticTest

{

public static void main(String arg[]) throws IOException

{

out.print("Enter the Name : ");

String name=MyClass.br.readLine();

out.print("Age : ");

int age=MyClass.sc.nextInt();

if(age>=18)

out.println("Dear "+name+" you can vote");

else

out.println("Dear "+name+" you cannot vote");

}

}

**Getting Input From Command Line**

* When run a class from the command prompt, we can pass different arguments to the program
* These arguments get passed to the main() method into args variable
  + public static void main(String args[])
* Use args variable to read data from command line
* In Java, arrays are treated like objects
* Each array provides **Length** property to get size of an array

Example

Write a program to read name and age of a person from command line and check it to be valid voter. Lets assume class name as Voter

Java Voter Amit 34

* public static void main(String args[])
* args[0] 🡪 Amit
* args[1]🡪 34

**Passing dynamic number of arguments**

* Use **…** to define the dynamic array

Example

Write a method that accepts any number of arguments as number and print sum of those number

class DynamicArgs

{

public static void sum(int... num)

{

int s=0;

for(int i=0;i<num.length;i++)

{

s=s+num[i];

}

System.out.println("Sum is : "+s);

}

public static void main(String args[])

{

sum(2,3,4,5,6,7,8,9);

}

}

**Using for-each loop for arrays and collections**

* Java provides a new loop to work with array and collections without knowing site of it and any array indexing

for(datatype variable : arrayname)

statement;

class DynamicArgs

{

public static void sum(int... num)

{

int s=0;

for(int t : num)

s=s+t;

System.out.println("Sum is : "+s);

}

public static void main(String args[])

{

sum(2,3,4,5,6,7,8,9);

}

}

Write a program to get some numbers from command line and print biggest number out of those numbers.

class Sum

{

public static void main(String args[])

{

int b=Integer.parseInt(args[0]);

int n;

for(int i=1;i<args.length;i++)

{

n=Integer.parseInt(args[i]);

if(b<n)

b=n;

}

System.out.println("Biggest one is "+b);

}

}

**Arrays**

* A variable that can hold multiple values of similar data type
* In Java Arrays are treated like object and created with new keyword and provide **length** property to get size of array
* Can be of two types
  + Single Dimensional
  + Multi-Dimensional

Examples

int []num=new int [5]; // Single dimensional

int [][]num=new int[3][4]; //two dimensional array

Write a program to get an array of 3X4 and print transpose of it

1 2 3 4

5 6 7 8

9 0 1 2

Output

1 5 9

2 6 0

3 7 1

4 8 2

**OOPs (Object Oriented Programming System)**

* A system used for better project management by mapping the real world entities in computer programming
* It is independent of any programming language
* It has several components
  + Inheritance
  + Abstraction
  + Encapsulation
  + Polymorphism
* It get implemented using class and objects

**Encapsulation**

* All the elements of a class must be placed at one place
* Enclose all the members under a pair of braces inside a class

class <classname>

{

//members

}

**Types of Members inside a class**

1. Field
2. Method

Fields are used to hold some data about an object. They can be of two types

1. Variables
2. Constants
   1. Use **final** keyword to declare a constant

Example

class Number

{

int num;//variable

final double pi=3.14; //constant

}

**Methods**

* Denotes some action
* Can be of two types
  + General methods
  + Special methods
* General methods can be three types
  + Concrete methods
  + Abstract Methods
  + Final Methods
* Special methods can be as Constructor

**Constructor**

Special method of a class having some features

1. Same name as class name
2. No return type
3. Used to initialize fields of a class
4. If no constructor is created then default constructor is created automatically
5. If we create any parameterized constructor then default constructor is not created automatically, we have to create it, if required

Note: Use **JAVAP** tool to view contents inside a class

**What is a class?**

* A class is a blueprint or template to create similar set of objects about an ***entity***
* A class manages two things
  + Data Members or fields
  + Methods
* It is a user defined data type

**What is an object?**

* An object is the real entity created based on class specification
* To create an object we need two things
  + **new** keyword
  + constructor

classname refererencename=new constructor(<parameter>);

Example

String s=new String();

Scanner sc=new Scanner(System.in);

**Passing data to the class fields**

1. Using Constructors
2. Using General Methods

Example

Create a class as **Number** having a field as ***num.*** Now create constructors to pass data to num along with some method to pass data to num.

Now write the method to operate on this number as factorial, square, cube etc.

**Solution**

class Number

{

int num; //field

public Number()

{

}

public Number(int n)

{

num=n;

}

public void setNumber(int n)

{

num=n;

}

public int getNumber()

{

return num;

}

public long factorial()

{

long f=1;

for(int i=1;i<=num;i++)

f=f\*i;

return f;

}

public int square()

{

return num\*num;

}

}

class Test

{

public static void main(String args[])

{

Number x=new Number();

x.setNumber(8);

Number y=new Number(7);

System.out.printf("Factorial of %d is %d\n",x.getNumber(), x.factorial());

System.out.printf("Factorial of %d is %d",y.getNumber(), y.factorial());

}

}

Note: Use **this** keyword to refer object of current class

class Number

{

int num;//field

public Number()

{

}

public Number(int num) //parameter

{

this.num=num;

}

public void setNumber(int num)

{

this.num=num;

}

public int getNumber()

{

return num;

}

public long factorial()

{

long f=1;

for(int i=1;i<=num;i++)

f=f\*i;

return f;

}

public int square()

{

return num\*num;

}

}

class Test

{

public static void main(String args[])

{

Number x=new Number();

x.setNumber(8);

Number y=new Number(7);

System.out.printf("Factorial of %d is %d\n",x.getNumber(), x.factorial());

System.out.printf("Factorial of %d is %d",y.getNumber(), y.factorial());

}

}

**Categories of members inside the class**

* Non static or instance members
* Static or class members

Non static or instance members always need an object to call them while static members do not need an object to call them but can be called by the object.

Static members can be called using the class name.

Use **static** keyword to declare as member as static

**P1**

Create a class **Customer** having fields **name** and **balance**. Create constructor to pass name and balance. Create a method **Deposit()** to deposit some money and **ShowBalance()** to show the current balance. Create another method as **CalcInterest()** that get amount, rate of interest and period and returns the interests paid for that periods.

Decide yourself for static and non-static members.

class Customer

{

String name;

double balance;

public Customer(String name, double opamount)

{

this.name=name;

balance=opamount;

}

public void deposit(int amount)

{

balance+=amount;

}

public void showBalance()

{

System.out.println("Current balance is "+balance);

}

public static void calcInterest(int p, double r, double t)

{

double intr=(p\*r\*t)/100;

System.out.println("Interest is "+intr);

}

}

class CustomerTest

{

public static void main(String args[])

{

Customer c1=new Customer("Amit Verma",9000);

c1.deposit(5000);

c1.showBalance();

Customer.calcInterest(10000,9,5);

}

}

**P2**

1. Create a class for a customer of a home shoppe to manage their information like customer id, name, mobile, credit limit, current credit.
2. Create a menu to make a purchase and get payment, show the current credit available, show general customer details, open as account

**Abstraction**

* Defines the scope or visibility of the class and its members
* Java provides four abstraction layers
  + Private
  + Public
  + Protected
  + Default or Package
* Private means accessible within the class
* Public means anywhere access in same or different package. Can be used by an object in any package
* Protected means in same class or in child class in same or different package. Can be used by an an object in same folder or package only.
* Package or default means within the same folder or package

**What is a package?**

* A package is a folder having related set of classes
* Each class in a package must have **package** command on top

package <packagename>;

class <classname>

{

//members

}

**Creating a Custom Package**

* Create a folder to hold your packages
  + Example e:\b1packages
* Now decide the package name and create a folder with that name
  + Example e.g. general
* Now create the classes and place them into this folder

package general;

class MyMath

{

int num;

MyMath()

{

}

MyMath(int num)

{

this.num=num;

}

double squareRoot()

{

return Math.pow(num,0.5);

}

}

**Using a package**

Import a package and use it classes

Set the classpath to define the folder name having the packages

For DOS

SET CLASSPATH=%CLASSPATH%;e:\b1packages

For Kawa

Package🡪 classpath 🡪 Add Dir…

NetBeans

Select the Project Name 🡪 Library Properties 🡪 Add Jar/Folder…

**Distributing the Packages**

* Convert the packages into a .ZIP file or .JAR file
* Use WINZIP.EXE to create a ZIP file and JAR.EXE to create a JAR File

**Options of JAR command**

* c 🡪 create
* t 🡪 tabulate
* x 🡪 extract
* v 🡪 verbose
* f 🡪 specify the file name

Create a JAR file for all packages

JAR cvf pkg.jar .

Now give path of pkg.jar instead of the folder

**Polymorphism**

* When an item can perform multiple activities is called as polymorphism
* Java allows only Method Overloading to implement the polymorphism

**Method Overloading**

* When two or more method have the same name but different number of arguments or type of arguments is called as method overloading

void area(int side) 🡪 area of square

void area(double r) 🡪 area of circle

void area(int l, int w) 🡪 area of rectangle

**Inheritance**

* Provides re-usability of code
* Java provides single inheritance
* Use **super** keyword to refer the super class
* All classes in Java are child of Object class
* Use **extends** keyword to inherit a class into other class

Note: Use **JAVAP** tool to view contents inside a class file

JAVAP classname

* Classes are divided at three levels
  + Current Class
  + Super Class
  + Parent Class

Create a class as Num2 having two numbers. Write some methods like g2() and p2()

Create another class Num3 by inheriting class Num2 and create the methods g3() and p3()

class Num2

{

int a,b;

public Num2(int a, int b)

{

this.a=a;

this.b=b;

}

public int g2()

{

return a>b?a:b;

}

public int p2()

{

return a\*b;

}

}

class Num3 extends Num2

{

int c;

public Num3(int x, int y, int z)

{

super(x,y);

c=z;

}

public int g3()

{

return g2()>c?g2():c;

}

public int p3()

{

return p2()\*c;

}

}

class InhTest

{

public static void main(String args[])

{

Num3 x=new Num3(4,5,6);

System.out.println(x.p2());

}

}

**Method Overriding**

* A method of parent or super class, re-written in child class with same signature but different body contents is called as method overriding
* It is best used for runtime polymorphism or dynamic method dispatch (DMD)
* While overriding we can increase scope of the overridden method but cannot decrease it

class Num2

{

int a,b;

public Num2(int a, int b)

{

this.a=a;

this.b=b;

}

public int greatest()

{

return a>b?a:b;

}

public int product()

{

return a\*b;

}

}

class Num3 extends Num2

{

int c;

public Num3(int x, int y, int z)

{

super(x,y);

c=z;

}

public int greatest() //method overriding

{

return super.greatest()>c?super.greatest():c;

}

public int product() //method overriding

{

return super.product()\*c;

}

}

class InhTest

{

public static void main(String args[])

{

Num3 x=new Num3(4,5,6);

System.out.println(x.product());

}

}

**Runtime Polymorphism**

* In inheritance, a parent can hold reference to its childs and can invoke all those methods of childs whose signature is provided from parent to the child
* Single reference of a parent class can be used to hold reference of childs and invoke its methods

final **methods**

* A method that can never be overridden
* Use **final** keyword with such methods

final class

* The class can never be inherited in other class

abstract methods

* A method having the signature but no body contents
* Can be created at two places
  + Inside an abstract class
  + Inside an interface
* If created inside an abstract class, **abstract** keyword is required. If a class contains any abstract method, it must be declared as abstract class.
* If created inside an interface, no keyword is required. All methods inside an interface are public and abstract by default

**Abstract class**

* A class which can never be instantiated
* An abstract class may or may not have any abstract method
* Use **abstract** keyword to declare such classes
* Such classes are used for inheritance purpose only

Example

Institute Management System

* Entities
  + Student
    - Alumni
    - Current
  + Faculty
    - Permanent
    - Visiting

**Common to All**

1. Name
2. Email
3. Mobile

Common to student

1. rollno
2. course

Common to Faculty

1. empid
2. subject

Alumni

* orgname
* designation

Current Student

* currsem

Permanent Faculty

* leaves

Visiting Faculty

* orgname
* designation

Example

abstract class Common

{

String name,email,mobile;

public Common(String name,String email,String mobile)

{

this.name=name;

this.email=email;

this.mobile=mobile;

}

public String getName()

{

return name;

}

public String getEmail()

{

return email;

}

public String getMobile()

{

return mobile;

}

}

class Faculty extends Common

{

int empid;

String subject;

public Faculty(int empid, String name,String email,String mobile, String subject)

{

super(name,email,mobile);

this.empid=empid;

this.subject=subject;

}

public int getEmpid()

{

return empid;

}

public String getSubject()

{

return subject;

}

}

class Test

{

public static void main(String args[])

{

Faculty f=new Faculty(123,"Rakesh Verma","rakesh@yahoo.com","98988998","Oracle");

System.out.println(f.getName());

System.out.println(f.getMobile());

System.out.println(f.getSubject());

}

}

**Interfaces**

* A user defined data type very similar to class but contains only abstract methods and final fields
* It allows to implement multiple inheritance in Java
* A class can inherit any number of interfaces using **implements** keyword
* Use **interface** keyword to create an interface
* One interface can also inherit other interface using **extends** keyword
* Can never be instantiated but can create its reference
* In an interface all members are public and abstract by default

Example

Create an ERP software for different departments of an organization

1. Finance🡪budget()
2. HR🡪 salary()

interface Finance

{

void budget();

}

interface Hr

{

void salary();

}

class ERP implements Finance,Hr

{

public void budget()

{

System.out.println("Budget is 5cr");

}

public void salary()

{

System.out.println("7th of Every month");

}

}

class AbcIndia

{

public static void main(String args[])

{

Hr x=new ERP();

x.salary();

}

}

**Terms in inheritance**

abstract🡪 class, method

final🡪 field, method, class

interface

super

extends

implements

**Exception Handling**

* A method to find runtime errors in a program using some special kind of classes called exceptions
* Each exception is a class inherited from **Exception** class and have Exception world associated with them
  + IOException
  + IndexOutOfBoundException
  + FormatException
* All such classes provide common methods
  + String getMessage()
  + String toString()
  + void printStackTrace()
* Java provides five keywords for exception handling
  + try
  + catch
  + throw
  + throws
  + finally
* Here try-catch is a block of statements to execute them and trap the runtime errors

try

{

//statements

}catch(classname reference)

{

//message

}

* One try block can have many catch statements
* ***finally*** is a block to always execute some irrespective of an error
* A try can have only finally block at the bottom

**Test Case**

Write a program to input two numbers and print division of that numbers

import java.io.\*;

class ExpTest

{

public static void main(String args[])

{

BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

try

{

System.out.print("Number 1 : ");

int a=Integer.parseInt(br.readLine());

System.out.print("Number 2 : ");

int b=Integer.parseInt(br.readLine());

int c=a/b;

System.out.println("Divisions : "+c);

return;

}

catch(NumberFormatException ex)

{

System.out.println("Sorry!Only Numbers are allowed");

}

catch(ArithmeticException ex)

{

System.out.println("Sorry! Denominator cannot zero");

}

catch(Exception ex)

{

System.out.println("Some error has occurd. contact customer care at 8989898");

//System.out.println(ex.toString());

//ex.printStackTrace();

}

finally

{

System.out.println("Thanks");

}

}

}

**Creating Custom Exceptions**

* Create a class as exception to send a message ***Low Balance Unable to withdraw*** when a user tries to get more money than balance.
* Being an exception class, inherit it from Exception class
* Override the methods of Exception class
  + String getMessage()
  + String toString()

Note:

* Use **throw** keyword to throw an object of some exception kind of class
* Use **throws** keyword to mark a method that the method is throwing some kind of exception to be trapped by the compiler

class LowBalanceException extends Exception

{

public String getMessage()

{

return "Sorry! Low Balance. Unable to Withdraw";

}

public String toString()

{

return "LowBalanceException: Sorry! Low Balance. Unable to Withdraw";

}

}

class Customer

{

String name;

int balance;

public Customer(String name, int opamt)

{

this.name=name;

balance=opamt;

}

public void deposit(int amount)

{

balance+=amount;

}

public void withdraw(int amount) throws LowBalanceException

{

if(amount>balance)

throw new LowBalanceException();

balance-=amount;

}

public void showBalance()

{

System.out.println("Current Balance is "+balance);

}

}

class SBI

{

public static void main(String args[])

{

Customer c1=new Customer("Amit",8000);

c1.deposit(5000);

c1.showBalance();

try

{

c1.withdraw(30000);

}catch(LowBalanceException ex)

{

System.out.println(ex.getMessage());

}

c1.showBalance();

}

}

**P4**

Create class a InvalidNumberException showing a message **Invalid Number Try Again**.

Create another class a Number having a field num and a method factorial(). Throw an exception is a factorial() is called without passing any data to num as InvalidNumberException

Use Number class in another class as TestExp and test it

**String Handling**

class StringTest

{

public static void main(String args[])

{

String s1="Vikas";

String s2="Vikas";

if(s1==s2)//value comparison

System.out.println("Equal");

else

System.out.println("Unequal");

}

}

Output

Equal

class StringTest

{

public static void main(String args[])

{

String s1=new String("Vikas");

String s2=new String("Vikas");

if(s1==s2) //address comparison

System.out.println("Equal");

else

System.out.println("Unequal");

}

}

Output

Unequal

class StringTest

{

public static void main(String args[])

{

String s1=new String("Vikas");

String s2=new String("Vikas");

if(s1.equals(s2)) //value at the address

System.out.println("Equal");

else

System.out.println("Unequal");

}

}

Output

Equal

**Types of String**

* Strings are of two types
  + Immutable String or Fixed Size String
  + Mutable String or Expandable Size String
* String class is used to managed immutable strings while StringBuilder and StringBuffer classes are used from mutable strings
* When we concatenate two strings using String class, a memory is de-referenced and send to Garbage Collector
* Use StringBuffer (network applications) or StringBuilder (local applications) to manage expandable strings using Append() method

class StringTest

{

public static void main(String args[])

{

String s="Amit";

s=s+" Kumar";

System.out.println(s);

StringBuilder sb=new StringBuilder();

sb.append("Amit");

sb.append(" Kumar");

System.out.println(sb);

String x=sb.toString();

}

}

**Methods of String class**

* int length()
* String toUpperCase()
* String toLowerCase()
* char charAt(int index)
* int indexOf(String s) returns -1 if not found
* int indexOf(String s, int startpoint)
* int lastIndexOf(String s)
* String substring(int start, int end)
  + To read a substring from a string as (end-start) characters
* boolean equals(String s)
* boolean equalsIgnoreCase(String s)
* boolean startsWith(String s)
* boolean endsWith(String s)

Example

class StringMethods

{

public static void p(String s)

{

System.out.println(s);

}

public static void main(String args[])

{

String name="Vikas kumar Chauhan";

p("Length is "+name.length());

p("In upper case :"+name.toUpperCase());

p("Lower case : "+name.toLowerCase());

p("third character is : "+name.charAt(2));

int pos=name.indexOf("kumar");

if(pos==-1)

p("Not found");

else

p("Found at "+pos);

p("Characters from 2 to 9 is " +name.substring(2,9));

StringBuilder sb=new StringBuilder(name);

String rname=sb.reverse().toString();

p("Reverse Stirng is : "+rname);

}

}

**Interview Questions**

1. What are static blocks?
2. Can we run any code before main()?
3. Can we run a program without main()?
4. Can we have private class?
5. Can be we private constructors?
6. What is singleton object? How to create it?
7. What is the difference among final, finally and finalizer?
8. What are the factory methods?
9. Can we invite Garbage Collect to collect the memory?

**Static Block**

Special block inside a program that always execute before main() and used to initialize static variables and execute some code that need to be called before main().

class StaticBlocks

{

static

{

System.out.println("Hi to all");

}

public static void main(String args[])

{

System.out.printf("Hello to all");

}

static

{

System.out.println("Bye to all");

}

}

**A program can run even without main() using static blocks**

class NoMain

{

static

{

System.out.println("Hi to all");

System.exit(0);

}

}

Private classes cannot be used outside the package

**Singleton Object**

* If we want to allows only single object of a class, it is called as singleton object
* We need to make the ***constructor as private*** to disallow creation of object outside class
* Create a static object of the class and create another method to return it to outside world
* If a method, other than constructor, of a class returns an object of its own class is called ***factory method***.

class MyClass

{

private static MyClass mc=new MyClass();//Singleton object

private MyClass()

{

}

public static MyClass getObject() //factory method

{

return mc;

}

public long factorial(int n)

{

if(n==1)

return 1;

else

return n\*factorial(n-1);

}

}

class SingletonTest

{

public static void main(String args[])

{

MyClass mc=MyClass.getObject();

System.out.println(mc.factorial(6));

}

}

**Finalizer**

* Special method of Object class with the name as finalize() which get called automatically when an object is garbage collected
* It is used to write some code that has to be executed when an object is released from memory
  + protected void finalize()

**Inviting GC for Garbage Collection?**

* Use System.gc() method

class FinalizerTest

{

public void finalize()

{

System.out.println("Bye Bye");

}

public static void main(String args[])

{

FinalizerTest t1=new FinalizerTest();

FinalizerTest t2=new FinalizerTest();

t1=null;

t2=null;

System.gc();

}

}

**Next Topics**

1. AWT (Abstract Window Toolkit)
2. Event Handling
3. Java Applets
4. JDBC
5. Java IO
6. Multi Threading
7. Collections and Generics
8. Java Utilities
9. Java Networking
10. Java Swing
11. RMI
12. Servlets
    1. Tomcat Server

**Abstract Window Toolkit (AWT)**

* AWT is a set of classes for GUI Programing
* All such classes are provided under **java.awt** package
* Such package provide three kind of classes
  + Containers
    - Frame, Panel, Applet, Dialog, Window etc.
  + Components
    - Used for user interaction
    - TextField, Checkbox, Choice, Menu, PopupMenu etc.
  + Supporting Classes
    - Color, Font, Dimension etc.

**Types of GUI Application**

* Desktop Applications
* Web Applications or Java Applets

To create a desktop based application, create a class as child of Frame class.

To create an applet it must inherit from java.applet.Applet class

Note: All containers and components are child of a class named as **Component** that provides all common methods for components and containers. It is an abstract class.

Some of the methods

* void setSize(int w, int h)
* void setVisible(boolean value)
* void setEnabled(boolean value)
* void setBackground(Color c)

Object 🡪 Component 🡪 Container 🡪 Window 🡪 Frame

**Frame class**

* To define a window
  + Frame()
  + Frame(String title)
* void setTitle(String title)
* void setResizable(boolean value)

**Label** class

* To define fixed text
  + Label()
  + Label(String text)
* void setText(String text)
* String getText()

TextField class

* To create single text and password field
  + TextField()
  + TextField(int columns)
* void setText(String text)
* String getText()
* void setEchoChar(char ch)
* char getEchoChar()
* char setEditable(boolean value)

Button class

* To create the push button
  + Button(String label)
* void setLabel(String label)
* String getLabel()

**Test Application**

Create a screen for Login and Password

Login : [ ]

Password : [ ]

<Check Login>

Note: Every container type class is child of **Container** class that provides common methods for all containers.

public void add(Component c)

public void setLayout(LayoutManager lm)

**Layout Managers**

1. BorderLayout (default for Frame)
2. FlowLayout (default for Applet)
3. CardLayout
4. GridLayout
5. GridBagLayout

To set the layout as FlowLayout use the command

setLayout(new FlowLayout());

**Naming the controls**

* Use **Hungarian Notation** given by Charles Simony of Hungary to name the controls
* It states that use three character prefix to define the controls type along with purpose of control
  + Label 🡪 lbl
  + TextField 🡪 txt
  + Button 🡪 cmd
  + Etc.

**Event Handling**

* A system to take some action when some activity is done using keyboard, mouse or some system activity
* Each event is an abstract method defined inside special interfaces called as ***Listeners*** under **java.awt.event** package
  + MouseListener
  + MouseMotionListener
  + KeyListener
  + AdjustmentListener
  + ItemListener
  + WindowListener
  + ActionListener
* ActionListener
  + public void actionPerformed(ActionEvent e)
    - Methods of ActionEvent class
      * Object getSource()
      * String getActionCommand()

**How events work in Java**

* In Java, events get handled using ***Event Delegation Model***
* We need to place a ***delegate*** on the controls on which events need to be trapped.
* Register the listeners with the control to place a delegate
  + Controlname.addXXXListener(object of the class having methods of listener);
* Implement the interface
* Override the methods

Example

cmdSquare.addActionListener(this);

cmdCube.addActionListener(this);

public void actionPerformed(ActionEvent e)

{

if(e.getSource()==cmdSquare)

System.out.println("Square Clicked");

else

System.out.println("Cube Clicked");

}

public void actionPerformed(ActionEvent e)

{

if(e.getActionCommand().equals("Square"))

System.out.println("Square Clicked");

else

System.out.println("Cube Clicked");

}

**Full Sample Code**

import java.awt.\*;

import java.awt.event.\*;

class TestApp extends Frame implements ActionListener

{

Label lblNumber,lblResult;

Button cmdSquare,cmdCube;

TextField txtNumber,txtResult;

public TestApp()

{

lblNumber=new Label("Number");

lblResult=new Label("Result");

txtNumber=new TextField(10);

txtResult=new TextField(10);

cmdSquare=new Button("Square");

cmdCube=new Button("Cube");

this.setResizable(false);

txtResult.setEditable(false);

//registering the events

cmdSquare.addActionListener(this);

cmdCube.addActionListener(this);

setLayout(new FlowLayout());

add(lblNumber);add(txtNumber);

add(lblResult);add(txtResult);

add(cmdSquare);add(cmdCube);

setSize(200,200);

setVisible(true);

}

public static void main(String args[])

{

new TestApp();

}

public void actionPerformed(ActionEvent e)

{

double num=Double.parseDouble(txtNumber.getText());

double result=0.0;

if(e.getSource()==cmdSquare)

result=num\*num;

else

result=num\*num\*num;

txtResult.setText(Double.toString(result));

}

}

Write a program to trap the current mouse position on a form when we move the mouse pointer on a frame.

Activity 🡪 Mouse Movement on Frame

Listener 🡪 MouseMotionListener

Methods of MouseMotionListener

* public void mouseMoved(MouseEvent e)
* public void mouseDragged(MouseEvent e)
  + Methods of MouseEvent class
    - int getX()
    - int getY()
    - boolean isPopupTrigger()

**Solutions**

import java.awt.\*;

import java.awt.event.\*;

class EventTest extends Frame implements MouseMotionListener

{

public void mouseMoved(MouseEvent e)

{

String p="Mouse is on "+e.getX()+","+e.getY();

setTitle(p);

}

public void mouseDragged(MouseEvent e)

{

}

public EventTest()

{

addMouseMotionListener(this);

setSize(300,300);

setVisible(true);

}

public static void main(String args[])

{

new EventTest();

}

}

Test Case 2

Write a program to have a text box which allows to type only numbers.

Activity 🡪 Keyboard

Listener 🡪 KeyListener

Methods

public void keyReleased(KeyEvent e)

public void keyPressed(KeyEvent e)

public void keyTyped(KeyEvent e)

**Methods of KeyEvent class**

char getKeyChar()

void consume() – to avoid unwanted character to view in text box

**Solution**

import java.awt.\*;

import java.awt.event.\*;

class KeyTest extends Frame implements KeyListener

{

TextField t;

public KeyTest()

{

t=new TextField(10);

t.addKeyListener(this);

setLayout(new FlowLayout());

add(t);

setSize(200,200);

setVisible(true);

}

public void keyReleased(KeyEvent e)

{

}

public void keyPressed(KeyEvent e)

{

}

public void keyTyped(KeyEvent e)

{

int code=(int)e.getKeyChar();

//setTitle(Integer.toString(code));

if(!((code>=48 && code<=57) || code==8))

e.consume();

}

public static void main(String args[])

{

new KeyTest();

}

}

**Implementing listeners at different locations**

1. Same class
2. Inner class
3. Outer class
4. Anonymous class

**Inner class**

* A class within a class that can access all the members of containing class

Test case 1

Create an application having a button as **Close**. On click on Close button quit the application. Use Inner class mode to implement the interfaces.

**Anonymous class**

* A class without any name is called as anonymous class
* If we override a method of a listener while registering the event, then a class get created automatically called as anonymous class

Example

b.addActionListener(new ActionListener()

{

public void actionPerformed(ActionEvent e)

{

System.exit(0);

}

}

);

**Working with Adapter classes**

* Special classes corresponding to listeners
* They have pre-implemented methods of listeners with blank body
  + WindowListener 🡪 WindowAdapter
  + KeyListener 🡪 KeyAdapter
* All those methods who have only one method, do not have their adapters
  + ActionListener
  + AdjustmentListener
  + Etc.
* They are based used in all other places except same class

Example

Create an application having a single window. Write the code to quit the window when we press x key of the form.

Hint : WindowListener or WindowAdapter

Event: windowClosing

import java.awt.\*;

import java.awt.event.\*;

class AdapterTest extends Frame

{

public AdapterTest()

{

addWindowListener(new WindowAdapter()

{

public void windowClosing(WindowEvent e)

{

System.exit(0);

}

});

setSize(200,200);

setVisible(true);

}

public static void main(String args[])

{

new AdapterTest();

}

}

**Other controls**

**Checkbox class**

* To create a check box and radio button
* Checkbox allows to select all or none
* Radio buttons is a group of controls where we can select only one item from group

For Checkbox

Checkbox(String text)

Checkbox(String text, boolean state)

boolean getState()

**Sample case**

Create a window form having a check box [ ] Agree to terms and conditions and a button. Initially the button is disabled. When checkbox is checked enable it and unchecked disable it

Listener -> ItemListener

Method : itemStateChanged

import java.awt.\*;

import java.awt.event.\*;

class CheckboxTest extends Frame implements ItemListener

{

Button b;

Checkbox c;

public CheckboxTest()

{

c=new Checkbox("Agree to terms and conditions");

c.addItemListener(this);

b=new Button("Next");

setLayout(new FlowLayout());

add(c);add(b);

b.setEnabled(false);

setSize(200,200);

setVisible(true);

}

public static void main(String args[])

{

new CheckboxTest();

}

public void itemStateChanged(ItemEvent e)

{

b.setEnabled(c.getState());

}

}

**Creating Radio Buttons**

* Create a CheckboxGroup class object
* Checkbox(String text, CheckboxGroup cg, boolean state)

Checkbox male,female;

CheckboxGroup gender;

gender=new CheckboxGroup();

male=new Checkbox("Male",gender,true);

female=new Checkbox("Female",gender,false);

**Choice class**

* To create a drop down list
* It allows to select only one item
  + Choice()
  + public void add(String s)
  + String getSelectedItem()
  + String getSelectedIndex()

**List** class

* To create a list box that allows to select one or more items
  + List()
  + List(int size)
  + List(int size, boolean multiple)
  + void add(Strings )
  + String []getSelectedItems()
  + int []getSelectedIndices()

TextArea class

* To create multiline Text box
  + TextArea(int rows, int columns)
  + void setText(String s)
  + String getText()

**Creating Menus**

* Menus can be of two types
  + Dropdown Menu
  + Popup Menu

MenuBar

* Menu
  + MenuItem
  + Menu
* To place the Menubar on the Frame use **setMenuBar()** method of Frame class

**Creating a Popup Menu**

* Use PopupMenu class to create the popup menu
* To show a popup menu trap the right click using isPopupTrigger() method MouseEvent class with mouseRealesed() event from MouseListener or MouseAdapter
* Use show() method of PopupMenu class to show the popup menu at the same location the mouse key is released

import java.awt.\*;

import java.awt.event.\*;

class PopupTest extends Frame implements ActionListener

{

PopupMenu pm;

MenuItem red,green,blue;

public PopupTest()

{

pm=new PopupMenu();

red=new MenuItem("Red");

green=new MenuItem("Green");

blue=new MenuItem("Blue");

pm.add(red);

pm.add(green);

pm.add(blue);

add(pm);

red.addActionListener(this);

green.addActionListener(this);

blue.addActionListener(this);

addMouseListener(new MouseAdapter()

{

public void mouseReleased(MouseEvent e)

{

if(e.isPopupTrigger())

pm.show(e.getComponent(), e.getX(),e.getY());

}

});

setSize(300,300);

setVisible(true);

}

public void actionPerformed(ActionEvent e)

{

MenuItem b=(MenuItem)e.getSource();

if(b==red)

setBackground(Color.red);

else if(b==blue)

setBackground(Color.blue);

else

setBackground(Color.green);

}

public static void main(String args[])

{

new PopupTest();

}

}

**Setting positions of controls**

* Remove the existing layout manager
  + setLayout(null);
* Use setBounds() method on the controls to define the location and size of controls
  + public void setBounds(int x, int y, int w, int h)

Sample

import java.awt.\*;

import java.awt.event.\*;

class GUIApp extends Frame

{

Label lblNumber,lblResult;

TextField txtNumber,txtResult;

public GUIApp()

{

addMouseMotionListener(new MouseMotionAdapter()

{

public void mouseMoved(MouseEvent e)

{

String s=e.getX()+","+e.getY();

setTitle(s);

}

});

setLayout(null);

lblNumber=new Label("Number");

lblNumber.setBounds(30,50,60,20);

txtNumber=new TextField();

txtNumber.setBounds(100,50, 100,20);

lblResult=new Label("Result");

lblResult.setBounds(30,80,60,20);

txtResult=new TextField();

txtResult.setBounds(100,80, 100,20);

add(lblNumber);add(txtNumber);

add(lblResult);add(txtResult);

setSize(220,200);

setVisible(true);

}

public static void main(String args[])

{

new GUIApp();

}

}

**Java Applets**

* Java classes that get merged into HTML code and run inside a web browser are called as Java Applets
* Such classes must inherit from **java.applet.Applet** class
* Applet related classes are provided under **java.applet** package
* Applets do not require any constructor or entry point
* No sizing and visibility settings
* Must be public
* Java Applet class provides various methods that need to be overridden in the child class for customized applets
  + public void paint(Graphics g)
  + public void init()
    - Similar to constructor
  + public Image getImage(URL path, String filename)
  + public URL getCodeBase()
  + public URL getDocumentBase()
  + public AudioClip getAudioClip(URL path, String filename)
  + public String getParameter(String paramname)
  + public void showStatus(String s)
    - To show a message on status bar of the browser

**How to write first Applet**

* Create a class and inherit it from Applet class
* Override paint() method

import java.awt.\*;

import java.applet.\*;

public class FirstApplet extends Applet

{

public void paint(Graphics g)

{

g.drawString("Hello to Applet",10,20);

}

}

Note: painting is done on applet using java.awt.Graphics class. It provides various methods

public void drawString(String s, int x, int y)

public void drawImage(Image img, int x, int y, ImageObserver io)

public void drawImage(Image img, int x, int y,int w, int h, ImageObserver io)

**How to run an applet**

* Create an HTML file using <APPLET> tag and its attributes
  + code=”classfilename”
  + width=”x”
  + height=”y”
  + codebase=”folder of class file”
  + archive=”jar file name”

Example

<applet code="FirstApplet" width="200" height="100">

</applet>

* FirstApplet.htm

Method 1

Open this file in Web Browser

* The browser must be Java Enabled

Method 2

Open the file with **AppletViewer** tool of JDK from DOS Prompt

AppletViewer FirstApplet.htm

**Test Application**

* Write a program with a text box to input a number and print Square Root of it onto the status bar of the browser

import java.awt.\*;

import java.awt.event.\*;

import java.applet.\*;

public class GuiApplet extends Applet

{

TextField t;

Label l;

Button b;

public void init()

{

t=new TextField(10);

l=new Label("Number");

b=new Button("Square Root");

add(l);add(t);add(b);

b.addActionListener(new ActionListener()

{

public void actionPerformed(ActionEvent e)

{

double num=Double.parseDouble(t.getText());

double result=Math.pow(num,1.0/2);

showStatus("Square root is "+result);

}

});

}

}

**Placing class file and HTML files in different folders**

* Use **codebase** attribute of <applet> tag to define the folder name having the class files

<applet code="GuiApplet" width="200" height="200" codebase="classes">

</applet>

**Creating JAR file for the classes**

* Create the JAR file and define it using archive attribute of <Applet> tag

<applet code="GuiApplet" width="200" height="200" codebase="classes" archive="gui.jar">

</applet>

**Using Images into Applets**

* Use getImage() method Applet class to read an image from server
  + Image getImage(URL path, String filename)
* To get path of current website use the methods
  + URL getCodeBase()
    - Returns the path of class file
  + URL getDocumentBase()
    - Returns the path of HTML file

//Using images into applets

import java.awt.\*;

import java.applet.\*;

public class ImageApplet extends Applet

{

Image img;

public void init()

{

img=getImage(getDocumentBase(),"images/2.jpg");

}

public void paint(Graphics g)

{

g.drawImage(img,0,0,this);

}

}

**Passing parameters to the applets**

* <applet> tag provides a sub tag called as <param>
* <param> tag provides two attributes
  + Name
  + Value
* To get data send by a parameter use getParameter() method of Applet class
  + String getParameter(String pname)

Example

Pass the image name as parameter and use it to display

<applet code="ParamApplet" width="200" height="200">

<param name="photofile" value="6.jpg">

</applet>

//Using images into applets

import java.awt.\*;

import java.applet.\*;

public class ParamApplet extends Applet

{

Image img;

public void init()

{

String photo=getParameter("photofile");

img=getImage(getDocumentBase(),photo);

}

public void paint(Graphics g)

{

g.drawImage(img,0,0,this);

}

}

**What is repaint()?**

* repaint() is a method of Applet class that allows to call the paint() method again
* We the applet is repainted another internally get used called as update() to create the current applet

**Test Case**

Create a class to show the current mouse position with the mouse pointer.

<applet code="RepaintTest" width="300" height="300">

</applet>

//Using repaint method

import java.awt.\*;

import java.awt.event.\*;

import java.applet.\*;

public class RepaintTest extends Applet

{

int x,y;

public void init()

{

addMouseMotionListener(new MouseMotionAdapter()

{

public void mouseMoved(MouseEvent e)

{

x=e.getX();

y=e.getY();

repaint();

}

});

}

public void paint(Graphics g)

{

String s="Mouse is on "+x+","+y;

g.drawString(s,x,y);

}

public void update(Graphics gr)

{

int r=(int) (Math.random()\*255);

int g=(int) (Math.random()\*255);

int b=(int) (Math.random()\*255);

Color c=new Color(r,g,b);

setBackground(c);

}

}

**Working with drawings**

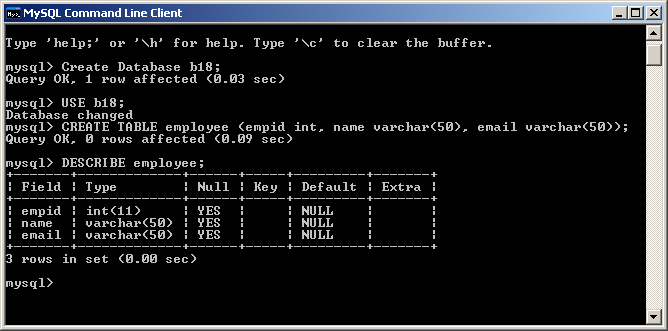
* Used to draw the drawing elements like Cirlce, Ellipse, line etc.
  + drawLine(int x1, int y1, int x2, int y2)
  + drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)
  + drawOval(int x, int y, int width, int height)
  + drawPolygon(int[] xPoints, int[] yPoints, int nPoints)
  + drawRect(int x, int y, int width, int height)
  + drawRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)
  + fillOval(int x, int y, int width, int height)

**Java Database Connectivity**

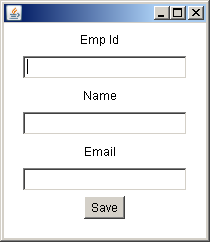
* Java is a front-end that can works with any back-end database
* First you must have RDBM Software, database and tables
  + My Sql
  + Oracle
  + MS SQL Server
  + MS Access etc.

**Using My Sql as RDBMS**

* Start MySql with password of user **root**
* Create your database
  + CREATE Database b18;
* Make this database as current database
  + USE b18;
* Create your tables
  + CREATE TABLE employee (empid int, name varchar(50), email varchar(50));
* To view the table use DESCRIBE command
  + DESCRIBE employee;



* Create a front-end screen in Java to input the data



* All classes and interfaces related with JDBC are provided under **java.sql** package
  + Class class
  + DriverManager class
  + Connection interface
  + Statement interface
  + PreparedStatement interface
  + CallableStatement interface
  + ResultSet interface

**Steps to connect with database**

1. Look for the driver class and its JAR file
   1. For MySql
      1. com.mysql.jdbc.Driver class
   2. Get the JAR file containing this class from MySql.com
2. Load the Driver into memory
   1. DriverManager.registerDriver(new com.mysql.jdbc.Driver());
3. Now provide the URL (Uniform Resource Locator) to reach the database server
   1. Connection cn=DriverManager.getConnection("jdbc:mysql://localhost:3306/b18","root",”pass");
4. Create the SQL Statement to be executed

String sql="INSERT INTO employee VALUES("+txtEmpId.getText()+",'"+txtName.getText()+"','"+txtEmail.getText()+"')";

1. Create an object and pass the reference to Statement interface to the command using Connection interface with **createStatement()** method
   1. Statement st=cn.createStatement()
2. Now execute the command depending on type of it
   1. executeUpdate()
      1. for all commands except SELECT
   2. executeQuery()
      1. for SELECT only
3. Close the connection
   1. cn.close();

Note: To show a dialog use classes from Swing (javax.swing) like JOptionPane

JOptionPane.showMessageDialog(this,"Record has been saved");

Example

try

{

DriverManager.registerDriver(new com.mysql.jdbc.Driver());

Connection cn=DriverManager.getConnection("jdbc:mysql://localhost:3306/b18","root","pass");

String sql="INSERT INTO employee VALUES("+txtEmpId.getText()+",'"+txtName.getText()+"','"+txtEmail.getText()+"')";

//System.out.println(sql);

Statement st=cn.createStatement();

st.executeUpdate(sql);

cn.close();

JOptionPane.showMessageDialog(this,"Record has been saved");

}catch(SQLException ex)

{

JOptionPane.showMessageDialog(this,ex.getMessage());

}

**Using MS Access as Database**

* To use Access as database Microsoft provides another technology called ODBC (Open Database Connectivity)
* JDBC calls the ODBC to reach the database
* Sun provides a driver called as JDBC-ODBC Bridge
  + Class file : sun.jdbc.odbc.JdbcOdbcDriver
* It is provided under rt.jar file of JRE. We do not need any classpath setting.

**Different methods to load the driver**

Method 1 : Using **Class** class

Class.forName(“classname”);

Method 2: Using DriverManager class

DriverManager.registerDriver(new constructor());

Method 3: Using System class

System.setProperty("jdbc.drivers",”classname”);

**Different ways to execute the SQL Statements**

Method 1: Using **Statement** interface

Statement st=cn.createStatement();

st.executeUpdate(sql);

Method 2: Using **PreparedStatement** interface

PreparedStatement ps=cn.prepareStatement(sql);

ps.executeUpdate();

Method 3: Using **CallableStatement** interface

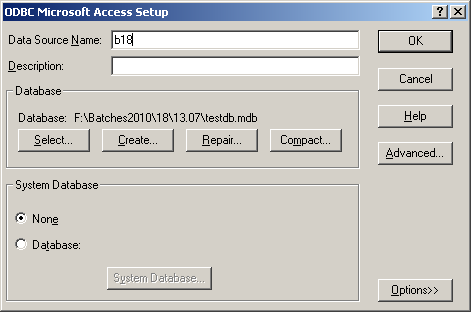
CallableStatement cs=cn.prepareCall("{call <procedurename>}");

cs.execute();

Note: JDBC-ODBC bridge needs a Data Source Name (DSN) to connect with database and hide the location of database from the program

To create DSN for a database follow the following steps

Control Panel 🡪 Administrative Tools 🡪 ODBC 🡪 Select System DSN 🡪 Add… 🡪 Select the Driver as Microsoft Access Driver (\*.mdb) for 2003 and Microsoft Access Driver (\*.mdb,\*.accdb) for 2007 and 2010 🡪 Select the Database file name and define some Data Source Name (DSN) e.g. **b18** is the data source name.



Now URL to connect with database will be

jdbc:odbc:<dsn name>

Example

jdbc:odbc:b18

**Sample Case**

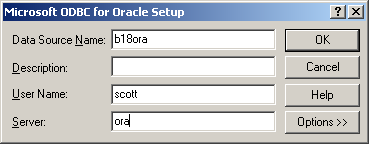
Create a table Student (rollno-N-PK, name-text, course-text) under testdb.mdb or testdb.accdb file

Create a data source name (DSN) as b18 for this database file.

Read data from keyboard and save it into database. (Use PreparedStatement and System property)

**Creating DSN for Oracle**

* Select driver as Microsoft ODBC for Oracle
* Provide the details to create the DSN



Example 1: Save Data to MS Access Direct Method

import java.util.\*;

import java.io.\*;

import java.sql.\*;

import static java.lang.System.\*;

class SaveData

{

public static void main(String args[])

{

try

{

Scanner sc=new Scanner(System.in);

BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

out.print("Enter Roll No : ");

int rollno=sc.nextInt();

out.print("Name : ");

String name=br.readLine();

out.print("Course : ");

String course=br.readLine();

String sql="INSERT INTO student VALUES("+rollno+",'"+name+"','"+course+"')";

//out.println(sql);

System.setProperty("jdbc.drivers","sun.jdbc.odbc.JdbcOdbcDriver");

Connection cn=DriverManager.getConnection("jdbc:odbc:b18");

PreparedStatement ps=cn.prepareStatement(sql);

ps.executeUpdate();

cn.close();

out.println("Records Saved");

}catch(IOException ex)

{

out.println("Error found : "+ex.getMessage());

}

catch(SQLException ex)

{

out.println("Error : "+ex.getMessage());

}

}

}

**Saving data to database using Place Holders (?)**

* While create the SQL commands don’t pass the data but pass the ? called as place holders
* Use setter method to pass data to the PreparedStatement or CallableStatement
* This method will not work with Statement
  + setXXX(columnno,data);
* Examples
  + setInt(columnno,data);
  + setDouble(columnno,data);
  + setString(columnno,data);

import java.util.\*;

import java.io.\*;

import java.sql.\*;

import static java.lang.System.\*;

class SaveDataPlaceHolders

{

public static void main(String args[])

{

try

{

Scanner sc=new Scanner(System.in);

BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

out.print("Enter Roll No : ");

int rollno=sc.nextInt();

out.print("Name : ");

String name=br.readLine();

out.print("Course : ");

String course=br.readLine();

String sql="INSERT INTO student VALUES(?,?,?)";

//out.println(sql);

System.setProperty("jdbc.drivers","sun.jdbc.odbc.JdbcOdbcDriver");

Connection cn=DriverManager.getConnection("jdbc:odbc:b18");

PreparedStatement ps=cn.prepareStatement(sql);

ps.setInt(1,rollno);

ps.setString(2,name);

ps.setString(3,course);

ps.executeUpdate();

cn.close();

out.println("Records Saved");

}catch(IOException ex)

{

out.println("Error found : "+ex.getMessage());

}

catch(SQLException ex)

{

out.println("Error : "+ex.getMessage());

}

}

}

**Reading data from database**

* Create the SELECT command
* Use ExecuteQuery() method of Statement or PreparedStatement to execute it and get the result into another interface **ResultSet**

Test Example

String sql=”Select \* from employee”;

PreparedStatement ps=cn.prepareStatement(sql);

ResultSet rs=ps.executeQuery();

* Use getXXX() methods to read data from the current row
  + String getString(column name or index number)
  + int getInt(column name or index number)
  + double getDouble(….)
* To move to different records use methods of ResultSet
  + boolean next()
  + boolean previous()
  + boolean first()
  + boolean last()

Test Case 1

Write a program to show all records inside a table “Student”

import java.sql.\*;

class ReadData

{

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

{

Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

Connection cn=DriverManager.getConnection("jdbc:odbc:b18");

String sql="Select \* from student";

PreparedStatement ps=cn.prepareStatement(sql);

ResultSet rs=ps.executeQuery();

while (rs.next())

{

System.out.println(rs.getString(1)+","+rs.getString(2)+","+rs.getString(3));

}

cn.close();

}

}

Test Case 2

Write a program to show records in GUI Format as <First> <Prev> <Next> <Last>. Make the ResultSet as scrollable and sensitive to changes in database even after issuing the select command.

Note: To make the scrollable Resultset, it provides certain constants. Constants can be of two types

1. To define the scrolling type
   1. TYPE\_FORWARD\_ONLY
   2. TYPE\_SCROLL\_SENSITIVE
      1. Changes will be available to the program
   3. TYPE\_SCROLL\_INSENSITIVE
      1. Changes will be not be available to the program
2. To define the concurrency type
   1. CONCUR\_READ\_ONLY
   2. CONCUR\_UPDATABLE

These constants get used while creating an object of Statements, CallableStatement or PreparedStatement etc.

**Assignment**

Table Name : Product (pid – N – PK, pname-Text, Price-N)

DSN Name: product

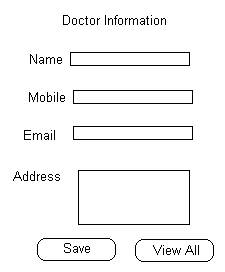
Write a program to input data from keyboard and save it into Product Table.

**Test Application**

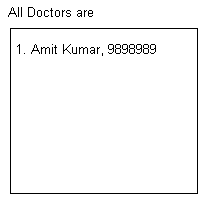
Create an application to manage doctor’s detains and the prescriptions given by doctors.

Table: Doctor

1. did – N – Auto Generated - PK
2. dname
3. mobile
4. email
5. address



When View All button is clicked show all doctors in following format



Now Add the prescription of the doctor

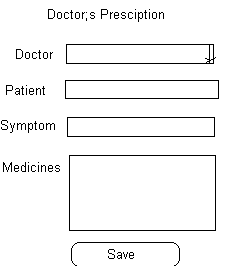


Table: Prescription

1. pid – N – Auto Generated
2. did - FK
3. patientname
4. symptom
5. medicines
6. date 🡪 current system date

Note: To get current System date use **Date** class of java.util package

import java.util.\*;

class CurrentDate

{

public static void main(String args[])

{

Date d=new Date();

System.out.println(d.toString());

}

}

Also create a Dropdown menu and Popup menu having Menu Items

1🡪 Add Doctor…

2 🡪 Add Prescription…

Types of Drivers

* Type 1: JDBC-ODBC Bridge
* Type 2: Native-API/partly Java driver
* Type 3: Net-protocol/all-Java driver
* Type 4: Native-protocol/all-Java driver

CallableStatement?

**Java Swings**

* Advance version of GUI Application to build light weight, rich look applications
* It can also be to create multiple look and feels
* It is part of Java Foundation Classes (JFC) and most of the classes starts with J
* Example
  + JFrame, JApplet, JPanel etc.
  + JButton, JTextField, JPasswordField, JTextArea
  + JScollPane, JOptionPane, JColorChoose etc.
* It allows to use images, shortcuts, hotkeys and tooltips on controls
* While adding the controls on a container get reference of the ContentPane layer
  + Container cn=getContentPane();

JLabel class

* To show text or image on the control
  + JLabel(String text)
  + JLabel(Icon img)
* All controls using an image as Icon must use a class inherited from Icon interface **ImageIcon()** class

Example

import java.awt.\*;

import javax.swing.\*;

class JLabelTest extends JFrame

{

public JLabelTest()

{

JLabel x=new JLabel(new ImageIcon("2.jpg"));

x.setToolTipText("Marry Me");

setLayout(new FlowLayout());

add(x);

setTitle("Hi");

setSize(200,200);

setVisible(true);

}

public static void main(String args[])

{

new JLabelTest();

}

}

**JScrollPane class**

* To provide scrolling on a control

**Closing a form without windowClosing**

* Define setDefaultCloseOperation() method with parameters
  + DO\_NOTHING\_ON\_CLOSE
  + HIDE\_ON\_CLOSE
  + DISPOSE\_ON\_CLOSE
  + EXIT\_ON\_CLOSE

Example

import java.awt.\*;

import javax.swing.\*;

class JScrollPaneTest extends JFrame

{

public JScrollPaneTest()

{

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

JLabel x=new JLabel(new ImageIcon("file2.jpg"));

x.setToolTipText("Marry Me");

JScrollPane jp=new JScrollPane(x);

add(jp);

setTitle("Hi");

setSize(200,200);

setVisible(true);

}

public static void main(String args[])

{

new JScrollPaneTest();

}

}

**JTextArea class**

* To select data in multiple rows and columns
  + JTextArea(int rows, int cols)
* void setText(String text)
* String getText()

import javax.swing.\*;

import java.awt.\*;

class JTextAreaTest extends JFrame

{

public JTextAreaTest()

{

JTextArea ta=new JTextArea(10,20);

JScrollPane jp=new JScrollPane(ta);

setLayout(new FlowLayout());

add(jp);

setSize(200,200);

setVisible(true);

}

public static void main(String args[])

{

new JTextAreaTest();

}

}

JButton class

* Allows to use Image, Text
* Also allows to use hot key and tooltips
  + JButton(String label)
  + JButton(Icon image)
  + JButton(String label, Icon image)

Example

import javax.swing.\*;

import java.awt.\*;

class JButtonTest extends JFrame

{

public JButtonTest()

{

JButton b1=new JButton("Save",new ImageIcon("save.gif"));

JButton b2=new JButton("Open",new ImageIcon("open.gif"));

JButton b3=new JButton("Close",new ImageIcon("close.gif"));

b1.setToolTipText("Save Data");

b2.setToolTipText("Open a file");

b3.setToolTipText("Close the File");

b1.setMnemonic('S');

b2.setMnemonic('O');

b3.setMnemonic('C');

setLayout(new FlowLayout());

add(b1);add(b2);add(b3);

setSize(200,200);

setVisible(true);

}

public static void main(String args[])

{

new JButtonTest();

}

}

**JColorChooser class**

* To select a color from a color dialog

static Color showDialog(parent, String title, Color initialColor)

Example

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.\*;

class JColorChooserTest extends JFrame implements ActionListener

{

public JColorChooserTest()

{

JButton b=new JButton("Select a color...");

b.addActionListener(this);

setLayout(new FlowLayout());

add(b);

setSize(200,200);

setVisible(true);

}

public void actionPerformed(ActionEvent e)

{

Color c=JColorChooser.showDialog(this,"Select a color",Color.white);

getContentPane().setBackground(c);

}

public static void main(String args[])

{

new JColorChooserTest();

}

}

**JPanel class**

* To create a sub container within a container
* It has all capabilities of container accept cannot be shown independently

Example

import java.awt.\*;

import javax.swing.\*;

class JPanelTest extends JFrame

{

public JPanelTest()

{

JTextField t=new JTextField();

JButton b[]=new JButton[10];

JPanel p=new JPanel();

for(int i=0;i<b.length;i++)

{

b[i]=new JButton(i+"");

p.add(b[i]);

}

p.add(new JButton("+"));

p.add(new JButton("-"));

add(t,"North");

add(p,"Center");

setResizable(false);

setSize(300,300);

setVisible(true);

}

public static void main(String args[])

{

new JPanelTest();

}

}

**Creating Toolbar**

- Use JToolBar class to create the toolbar

- Create JButton controls and add them into JToolBar

- Add JToolBar in North area of container

**Setting Icon on Frame and create full screen Frames**

Toolkit t=Toolkit.getDefaultToolkit();

Dimension d=t.getScreenSize();

Image g=t.getImage("cut.png");

setIconImage(g);

setSize(d.width,d.height);

**Creating Executable JAR files**

* Create a manifest file (.mft) to hold information of startup class file name
* Such file has one entry
  + Main-Class: <classaname>
* Now create the JAR file with extra option m

JAR cvmf filename.mft filename.jar files to merge

**Using Callable Statement**

* Using SQL Server 2005
* Create Database e.g. b18
* Create Table e.g. Student(rollno-N, name-Text, course-Text)
* Create the Procedure
  + Programmability
    - New Stored Procedure

CREATE PROCEDURE SaveStudent

@rollno int,

@name varchar(50),

@course varchar(50)

AS

BEGIN

INSERT INTO student VALUES(@rollno,@name,@course);

END

GO

* Select it and press F5 to execute it

**Using Procedures in Java**

* Create CallableStatement

import java.util.\*;

import java.sql.\*;

class UsingCallable

{

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

{

Scanner sc=new Scanner(System.in);

System.out.print("Roll No : ");

int rollno=sc.nextInt();

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

String name=sc.next();

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

String course=sc.next();

Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

Connection cn=DriverManager.getConnection("jdbc:odbc:sql");

CallableStatement cs=cn.prepareCall("{call SaveStudent(?,?,?)}");

cs.setInt(1,rollno);

cs.setString(2,name);

cs.setString(3,course);

cs.executeUpdate();

cn.close();

System.out.println("Record Saved");

}

}

**Multi Threading**

* A thread is a light weight process within a process
* It is best used to utilize the CPU resources
* Multiple activities can run automatically and concurrently
* All Java programs are single threaded by default
* One thread is always present known as **main** thread

**Life Cycle of Thread**

1. Born State
2. Ready State
3. Running State
4. Block State
5. Dead State

* Java provides **java.lang.Thread** class and **java.lang.Runnable** interface for multi threaded applications
* Thread class is used to create a thread
* Runnable interface provides a method called **run()** to provide working of thread

**Methods of Thread class**

* Thread(Runnable r)
* Thread(Runnable r, String threadname)
* String getName()
* void setName(String name)
* void setPriority(int n)
  + Min 1 Max 10 Default 5
* int getPriority()
* void start()
  + To send the thread in thread queue
* void sleep(int ms) throws InterruptedException
* void sleep(int milliseconds, int nano seconds) throws InterruptedException
* static Thread currentThread()

Example 1 : Getting the current thread information

class ThreadTest

{

public static void main(String args[])

{

Thread t=Thread.currentThread();

System.out.println(t.getName());

System.out.println(t.getPriority());

}

}

Example 2: Create a thread

//Assigning names to the threads

class MyThread implements Runnable

{

public MyThread(String name)

{

Thread t=new Thread(this,name); //born state

t.start(); // add thread to thread queue

}

public void run()

{

Thread t=Thread.currentThread();

for(int i=1;i<=10;i++)

System.out.println(t.getName()+": "+i);

}

public static void main(String args[])

{

System.out.println("Main Created");

MyThread t1=new MyThread("Pepsi");

MyThread t2=new MyThread("Coke");

MyThread t3=new MyThread("Fanta");

System.out.println("Main ends");

}

}

**Defining different jobs for different threads**

* First identify the current thread and define different job for different thread based on name of thread
* Use sleep() method to define the sleep time to automatically move control to next available thread

//Assigning names to the threads

class MyThread implements Runnable

{

public MyThread(String name)

{

Thread t=new Thread(this,name); //born state

t.start(); // add thread to thread queue

}

public void run()

{

String name=Thread.currentThread().getName();

if (name.equals("Pepsi"))

{

for(int i=1;i<=10;i++)

{

System.out.println(name+": "+i);

try

{

Thread.sleep(1000);

}catch(Exception ex)

{

}

}

}

else if (name.equals("Coke"))

{

for(char ch='A';ch<='Z';ch++)

{

System.out.println(name+" : "+ch);

try

{

Thread.sleep(600);

}catch(Exception ex)

{

}

}

}

else

{

for(int i=5;i<=100;i+=5)

{

System.out.println(name+":"+i);

try

{

Thread.sleep(1000);

}catch(Exception ex)

{

}

}

}

}

public static void main(String args[])

{

System.out.println("Main Created");

MyThread t1=new MyThread("Pepsi");

MyThread t2=new MyThread("Coke");

MyThread t3=new MyThread("Fanta");

System.out.println("Main ends");

}

}

**Application of threads in Applet**

import java.applet.\*;

import java.awt.\*;

public class ThreadApplet extends Applet implements Runnable

{

int i;

String fname;

public void init()

{

Thread t=new Thread(this);

t.start();

}

public void run()

{

for(;;)

{

fname="images/"+ (++i)+".jpg";

if(i==6) i=0;

try

{

Thread.sleep(3000);

}catch(Exception ex)

{

}

repaint();

}

}

public void paint(Graphics g)

{

Image img=getImage(getDocumentBase(),fname);

g.drawImage(img,0,0,this);

}

}

**Synchronization**

* When two or more threads try to access the same resource then there could be a deadlock
* To handle the deadlocks Java provides **synchronized** keyword
* It can be used as two ways
  + With the methods
  + As block

synchronized <return type> <function name>()

{

}

Or

<ruturn type> <function name>()

{

synchronized(this)

{

//statements

}

}

synchronized places a monitor or mutex (Mutual Exclusive Lock)

**Inter Thread Communication**

* When one thread needs the data from other thread and data is not available then the thread has to wait
* Once the data is available then we need to issue the notification
* Use methods
  + wait() throws InterruptedException
  + notify()
  + notifyAll()
* All these methods are provided by Object class

Example

//Stock.java

class Stock

{

int goods=0;

public synchronized void addStock(int i)

{

goods+=i;

System.out.println("Stock Added : "+i);

System.out.println("Current Stock : "+goods);

notify();

}

public synchronized int getStock(int j)

{

System.out.println("Item asked :"+j);

while(true)

{

if(goods>=j)

{

goods-=j;

System.out.println("Stock taken away:"+j);

System.out.println("Present stock :" +goods);

break;

}

else

{

System.out.println("Stock not enough...") ;

System.out.println("Waiting for stocks to come...");

try{

wait();

}catch(InterruptedException e){}

}

}

return goods;

}

public static void main(String args[])

{

Stock j =new Stock();

Producer p= new Producer(j);

Consumer c=new Consumer(j);

try{

Thread.sleep(10000);

p=null;

c=null;

System.out.println("Thread stopped");

}catch(InterruptedException e){}

System.exit(0);

}

}

//Consumer.java

class Consumer implements Runnable

{

Stock c;

Thread t;

Consumer(Stock c)

{

this.c=c;

t=new Thread(this,"consumer");

t.start();

}

public void run()

{

while(true)

{

try

{

t.sleep(1000);

}catch(Exception e){}

c.getStock((int)(Math.random()\*100));

}

}

}

//Producer.java

class Producer implements Runnable

{

Stock s;

Thread t;

Producer(Stock s)

{

this.s=s;

t=new Thread(this,"producer thread");

t.start();

}

public void run()

{

while(true)

{

try{

t.sleep(1000);

}catch(InterruptedException e){}

s.addStock((int)(Math.random()\*100));

}

}

}

**Daemon Thread**

* A thread that works in background is called as daemon thread
* Use check whether a thread is daemon thread or general thread use
  + boolean isDaemon() method
* To set type of thread use
  + void setDaemon(boolean value) method

**Joining the threads**

* When one thread needs some information from other thread then it should remain in memory until other thread’s work get finished
* To ensure this use join() method on such threads
* First joined thread will end in last
  + x.join()
  + y.join()
  + z.join()
* z will finish first, then y and in last x

**File Handling**

* To work with files and folders and allows to create new files in text mode and binary mode
* All related classes are provides under java.io package
  + File
  + FileReader – Read file in text mode
  + FileWriter – Write contents in text mode
  + FileInputStream – to read contents in binary mode
  + FileOutputStream – to write binary contents to the file

File class

* Use to create a folder, get list of files and sub-folder inside a folder, rename a folder and remove a folder
* Get properties of a file, remove it and rename it

File(String finame with path)

File(String foldername, String filename)

boolean exists()

returns true if the file or folder exists

boolean isFile()

boolean isDirectory()

long length() - returns length of file in bytes

long lastModified() – returns last modified data a long value. To convert into real date use Date class

String []list()

boolean mkdirs()

To create the directory

Example

//File handling

import java.io.\*;

class FileTest

{

public static void main(String args[])

{

File f=new File("d:/first.cs");

if(f.exists())

System.out.println("file exists");

else

System.out.println("file not found");

System.out.println("Size is " + f.length()+" bytes");

System.out.println("Last Modified : "+ new java.util.Date(f.lastModified()));

File f1=new File("d:/");

String s[]=f1.list();

for(String x : s)

System.out.println(x);

File f2=new File("d:/abc/pqr/xyz");

if(f2.mkdirs())

System.out.println("Folder Created");

else

System.out.println("Unable to create");

}

}

**Writing contents to a file**

* First read the contents
* Create an object of FileWriter class
* Use write() method to write the contents
* Close the file

import java.io.\*;

class SaveTextData

{

public static void main(String args[]) throws IOException

{

FileWriter fw=new FileWriter("d:/test.txt");

BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

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

String s=br.readLine();

fw.write(s);

fw.close();

}

}

Reading contents

//Reading contents of a file

import java.io.\*;

class ReadData

{

public static void main(String arg[]) throws Exception

{

FileReader f=new FileReader("d:/test.txt");

BufferedReader br=new BufferedReader(f);

String s;

while ((s=br.readLine())!=null)

{

System.out.println(s);

}

f.close();

}

}

**Saving binary data to the file**

* First convert textual data to binary data
  + Use getBytes() method of String class to convert data to byte array
    - byte [] getBytes()
* Use FileOutputStream class with write() method to save data in binary format

import java.io.\*;

class SaveTextData

{

public static void main(String args[]) throws IOException

{

FileOutputStream fw=new FileOutputStream("d:/test1.txt");

BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

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

String s=br.readLine();

byte []data=s.getBytes();

fw.write(data);

fw.close();

}

}

//Reading contents of a file

import java.io.\*;

class ReadData

{

public static void main(String arg[]) throws Exception

{

FileInputStream f=new FileInputStream("d:/test1.txt");

int n;

while ((n=f.read())!=-1)

{

System.out.print((char)n);

}

f.close();

}

}

Collections

Generics

Autoboxing and Unboxing

**Collections**

* A collection is a class that holds dynamic set of Objects
* Such classes are provided under java.util package
  + Vector
  + Hashtable
  + ArrayList
  + Enumeration
  + LinkedList
  + Stack
  + Queue

**Types of collections**

1. Legacy collection
2. Collection Framework

Legacy collection are provided with Java 1.0

1. Vector class – To hold single set of elements
2. Hashtable class – to hold key/value
3. Enumeration interface – to hold reference of the keys

Collection Framework is systemic and planned structure of classes and interface

* All such classes and interfaces are inherited from **Collection** interface
* It also has child interfaces
  + List 🡪 allows duplicate items
  + Set 🡪 allows only unique items
  + Map 🡪 allows key/value pairs
* ArrayList, LinkList, HashSet, HashMap etc. are classes from these interfaces

**Vector class**

* To hold dynamic set of objects
  + Vector()
  + void addElement(Object x)
  + Object elementAt(int index)
  + int size()

Example 1

import java.util.\*;

class VectorTest

{

public static void main(String args[])

{

Vector v=new Vector();

v.addElement(new Date());

v.addElement("Rahul");

v.addElement(new Integer(56));

v.addElement(new Integer(89));

v.addElement(new Float(45.6f));

System.out.println("Total items : "+v.size());

for(int i=0;i<v.size();i++)

System.out.println(v.elementAt(i));

}

}

**Fetching specific elements from a Vector**

* Use **instanceof** operator to compare the type of element

import java.util.\*;

class VectorTest

{

public static void main(String args[])

{

Vector v=new Vector();

v.addElement(new Date());

v.addElement("Rahul");

v.addElement(new Integer(56));

v.addElement(new Integer(89));

v.addElement(new Float(45.6f));

System.out.println("Total items : "+v.size());

for(int i=0;i<v.size();i++)

{

if(v.elementAt(i) instanceof Integer)

System.out.println(v.elementAt(i));

}

}

}

**Find the sum of the integers inside the vector**

import java.util.\*;

class VectorTest

{

public static void main(String args[])

{

Vector v=new Vector();

v.addElement(new Date());

v.addElement("Rahul");

v.addElement(new Integer(56));

v.addElement(new Integer(89));

v.addElement(new Float(45.6f));

System.out.println("Total items : "+v.size());

int sum=0;

for(int i=0;i<v.size();i++)

{

if (v.elementAt(i) instanceof Integer)

{

Integer x=(Integer) v.elementAt(i);

sum=sum+x.intValue();

}

}

System.out.println("sum is "+sum);

}

}

**Using ArrayList class**

* A similar version of Vector class in Collection Framework
  + void add(Object x)
  + Object get(int index)
  + int size()

import java.util.\*;

class VectorTest

{

public static void main(String args[])

{

ArrayList v=new ArrayList();

v.add(new Date());

v.add("Rahul");

v.add(new Integer(56));

v.add(new Integer(89));

v.add(new Float(45.6f));

System.out.println("Total items : "+v.size());

int sum=0;

for(int i=0;i<v.size();i++)

{

if (v.get(i) instanceof Integer)

{

Integer x=(Integer) v.get(i);

sum=sum+x.intValue();

}

}

System.out.println("sum is "+sum);

}

}

**Generics**

* Special collections that allows only specific kind of data
* The data type of elements is decided at the time of creation of object

import java.util.\*;

class VectorTest

{

public static void main(String args[])

{

ArrayList <Integer>v=new ArrayList<Integer>();

v.add(new Integer(56));

v.add(new Integer(89));

System.out.println("Total items : "+v.size());

int sum=0;

for(int i=0;i<v.size();i++)

{

sum=sum+v.get(i).intValue();

}

System.out.println("sum is "+sum);

}

}

**Auto boxing and Auto Unboxing**

* Automatic conversion of value type to reference type is auto boxing and reference type to value type is auto unboxing

Integer x=67; // value to reference conversion – auto boxing

int y=5+x; // reference to value conversion – auto unboxing

import java.util.\*;

class ArrayListTest

{

public static void main(String args[])

{

ArrayList <Integer>v=new ArrayList<Integer>();

v.add(56); //auto boxing

v.add(89);

System.out.println("Total items : "+v.size());

int sum=0;

for(int i=0;i<v.size();i++)

{

sum=sum+v.get(i); // auto unboxing

}

System.out.println("sum is "+sum);

}

}

**Hashtable class**

* Helps in managing key/value pairs
  + Hashtable()
  + void put(Object key, Object value)
  + Object get(Object key)
  + Enumeration keys()

**Enumeration interface**

* Interface that helps in getting details about the keys
  + boolean hasMoreElements()
    - returns true if some key is available
  + Object nextElement()
    - Returns the next key

import java.util.\*;

class HashtableTest

{

public static void main(String args[])

{

Hashtable t=new Hashtable();

t.put("BIT","Binary Digit");

t.put("JRE","Java Runtime Environment");

t.put("JVM","Java Virtual Machine");

t.put("GC","Garbage Collector");

System.out.println(t.get("JVM"));

Enumeration e=t.keys();

while (e.hasMoreElements())

{

Object key=e.nextElement();

System.out.println(key+"/"+t.get(key));

}

}

}

LinkedList class

* To managed a linked list
  + LinkedList()
  + void add(Object x)
  + void addFirst(Object x)
  + void addLast(Object x)
  + void set(int index, Object x)
    - To replace

Stack class

* To create a stack based on LIFO (Last-In-First-Out) model
  + Stack()
  + void push(Object x)
  + Object pop()
  + Object peek()

**Creating KeyStrokes on the MenuItems in Swings**

* First create a keystroke using combination of keys
* Use Virtual Key codes along with Masking keys to built the keystrock
  + KeyEvent
    - VK\_A
    - VK\_Z
    - VK\_ENTER
    - VK\_F1
    - VK\_F12
  + Event
    - ALT\_MASK
    - SHIFT\_ MASK
    - CTRL\_ MASK

Example

Ctrl+V 🡪 for vendor

Use getKeystroke() method KeyStroke class to make a keystroke

KeyStroke ks=KeyStroke.getKeyStroke(KeyEvent.VK\_V,Event.CTRL\_MASK);

To apply the keystroke on some Menu item use

setAccelerator() method

vendor.setAccelerator(ks);

**Networking or Socket Programming**

* A system to provide communication between two machines
* Communication can be of two types
  + TCP/IP based communication
    - Reliable communication
    - Connected Mode
  + UDP based communication
    - Un-reliable communication
    - Disconnected mode
* Such applications are known as client/server application which has two parts
  + Server Part
  + Client Part
* The server application must have a port number while a client must have IP Address or machine name of server along with port number to connect

**Port Number**

* A logical number allocated to some software to client/server activity
* A machine can have maximum of 65535 ports
* Some softwares have their default port numbers
  + Oracle : 1521
  + MySql : 3306
  + RMI : 1099
  + HTTP : 80
  + SMTP : 25
  + FTP: 21
  + Java Servers like Tomcat, Web Logic etc: 8080
  + Etc.

**TCP/IP based communication**

* A secured protocol that can be implanted using two classes
  + ServerSocket
  + Socket
* All classed related with networking are provided under **java.net** package
* ServerSocket class is used to create a server and wait for clients using its method
  + Socket accept()
  + ServerSocket(int port)
* Socket class is used to create relationship with server
  + Socket(String ip address or name, int port)

Write a test server and test client to check the connection

import java.net.\*;

class TestServer

{

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

{

ServerSocket s=new ServerSocket(300);

System.out.println("Server is ready....");

Socket client=s.accept(); //this method accept a **Socket** as server so we use Socket client

System.out.println("Client connected : "+client);

}

}

import java.net.\*;

import java.io.\*;

class TestClient

{

public static void main(String args[])

{

try

{

Socket server=new Socket("localhost",300);

System.out.println("Server found : "+server);

}catch(IOException ex)

{

System.out.println("Sever Not found");

}

}

}

**Testing this application**

* Go to DOS Prompt
* Open two command windows
* In First Window, run the server
* In Second window, run the client

**Methods of Socket class**

* InputStream getInputStream()
* OutputStream getOutputStream()

Write another programs to send a message from server and read it at client side

import java.net.\*;

import java.io.\*;

class HelloServer

{

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

{

ServerSocket s=new ServerSocket(300);

System.out.println("Server is ready....");

for(;;)

{

Socket client=s.accept();

PrintWriter out=new PrintWriter(client.getOutputStream(),true); //flush immediately(data send immediately so **true**)

out.println("Hi to all");

System.out.println("Client connected : "+client);

}

}

}

import java.net.\*;

import java.io.\*;

class HelloClient

{

public static void main(String args[])

{

try

{

Socket server=new Socket("localhost",300);

BufferedReader br=new BufferedReader(new InputStreamReader(server.getInputStream()));

System.out.println(br.readLine());

System.out.println("Server found : "+server);

}catch(IOException ex)

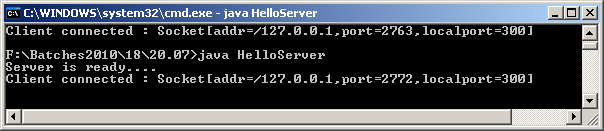
{

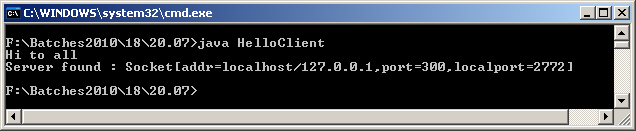
System.out.println("Sever Not found");

}

}

}





**Using UDP (User Datagram Protocol)**

* classes used
  + DatagramPacket
    - To make a data packet
  + DatagramSocket
    - To send and receive the data
  + InetAddress
    - To encapsulate the information of remote machine
* InetAddress class provides static methods to get information of other machine
  + public static InetAddress getLocalHost()
  + public static InetAddress getByName(String name)
* DatagramPacket is used to create a data packet to send and receive
  + DatagramPacket(byte []data, int size, InetAddress ia, int port)//full of data packet
  + DatagramPacket(byte []data, int size)//blank data packet
  + Use getLength() method to get size of data and getData() to get data in packet
* DatagramSocket class is used to send and receive data
  + DatagramSocket(int port)
  + void send(DatagramPacket p)
  + void receive(DatagramPacket p)
* To convert data from string to byte array use getBytes() method of String class
  + String s=”Hi”;
  + byte []data=s.getBytes();
* To covert data form byte array to String class object using constructor of String class
  + String(byte []data, int start , int end)

**ProcessBuilder class**

* To start a new process from a Java Class

import java.util.\*;

class ProcessBuilderTest

{

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

{

ProcessBuilder p=new ProcessBuilder("notepad.exe");

p.start();

}

}

**Enumerators**

* A name assigned to some value
* Use enum keyword to create the enumerators
* It creates a user defined data type

class enumTest

{

enum Rights

{Admin,General,Student,Faculty}

public static void main(String args[])

{

Rights x=Rights.Admin;

System.out.println(x);

}

}

**Java Servlets**

* Java Servlet is a class inherited from javax.servlet.GenericServlet class or javax.servlet.http.HttpServlet class
* Also known as server side applet which the HTML contents to the web browser and do not need any Java Virtual Machine at the client unlike Java Applet
* To write a servlet we must override methods of GenericServlet class or HttpServlet class
  + For GenericServlet
    - public void service(ServletRequest req, ServletResponse res) throws ServletException
  + For HttpServlet
    - public void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException
    - public void doPost(HttpServletRequest req, HttpServletResponse res) throws ServletException
* To get reference of the current browser use getWriter() method of res object
  + PrintWriter out=res.getWriter();
* To define the type of contents send to the client define contents types
  + res.setContentType("text/html");
* To run the Java Servlet, we special software called Java Web Server like
  + Tomcat
  + Web Logic
  + J2EE Server
  + Web Sphare

**Installting the Tomcat**

* Install the software at defined location and on defined port number e.g. 6789
* Define the password of admin for later use

Testing for tomcat server

* Open a web browser
  + <http://localhost:6789>

**Writing first Java Servlet**

* The class must be public
* Default Placement of files in tomcat
  + HTML/JSP
    - <tomcat>\webapps\ROOT
  + Class file
    - <tomcat>\webapps\ROOT\WEB-INF\classes

**Test Application**

Create an HTML form having a field as **num** that input a number to print a table.

<html>

<body>

<form method="POST" action="/table.xyz">

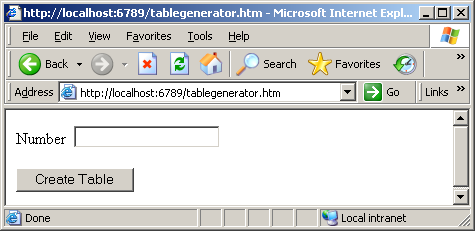
<p>Number&nbsp; <input type="text" name="num" size="20"></p>

<p><input type="submit" value="Create Table" name="B1"></p>

</form>

</body>

</html>



**Create a servlet class to receive the number and print table of that number. Place this class into some package.**

package b18;

import javax.servlet.\*;

import java.io.\*;

public class NumberTable extends GenericServlet

{

public void service(ServletRequest req, ServletResponse res) throws ServletException, IOException

{

PrintWriter out=res.getWriter();

int num=Integer.parseInt(req.getParameter("num"));

for(int i=1;i<=10;i++)

out.println(i\*num+"<br>");

}

}

To compile this class we need the JAR file containing javax.servlet package.

Add the JAR file into class path from <tomcat>\common\lib\servlet-api.jar

**Mapping the URL with the class**

* Open **web.xml** file from <tomcat>\webapps\ROOT\WEB-INF
* It is configuration file having pre-defined tags in XML format

<servlet>

<servlet-name>ts</servlet-name>

<servlet-class>b18.NumberTable</servlet-class>

</servlet>

<servlet-mapping>

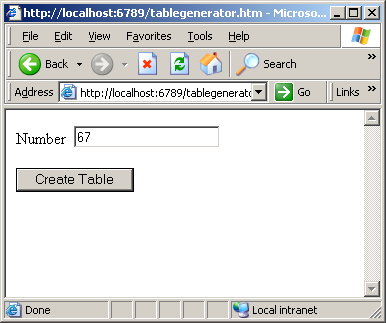
<servlet-name>ts</servlet-name>

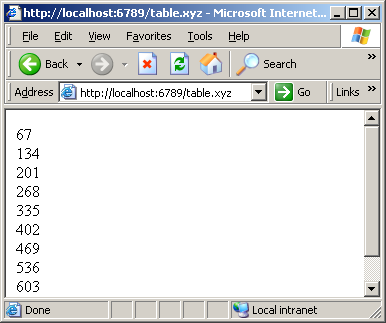
<url-pattern>/table.xyz</url-pattern>

</servlet-mapping>

Now test your application

<http://localhost:6789/tablegenerator.htm>





**RMI (Remote Method Invocation)**

* It is a client/server technology that can handle many clients from different locations and serve them by executing methods stored as server side
* Such applications use
  + RMI Protocol
  + The client must know about the IP Address and server name
  + No port number required
* Such application transfer data in secure mode using concept of marshelling and un-marshelling
* Such application contains four kind of classes
  + Interface
  + Implementation
  + Client Program
  + Server Program
* All classes and interfaces are provided under **java.rmi** and **java.rmi.server** package
* Main classes and interfaces required
  + Remote interface
  + RemoteException class
  + UnicastRemoteObject class
  + Naming class
* Additional tools
  + RMIC.EXE (RMI Compiler)
  + RMIREGISTRY.EXE

**Test Application**

Write an RMI based application to pass a string and get reverse that string.

Step 1:

First of all create an interface and inherit it from Remote interface. Define the methods to be called by the client. Every method must throw an exception RemoteException

import java.rmi.\*;

public interface Test extends Remote

{

String reverse(String s) throws RemoteException;

}

Step 2

Create an implementation class by implementing the interface and inheriting it from **UnicastRemoteObject** class.

import java.rmi.\*;

import java.rmi.server.\*;

public class TestImp extends UnicastRemoteObject implements Test

{

public TestImp() throws RemoteException

{

}

public String reverse(String s)

{

StringBuffer sb=new StringBuffer();

sb.append(s);

String temp=sb.reverse().toString();

return temp;

}

}

Step 3:

Create a server program to provide some server name to the class and send a copy of the implementation class to the client.

Use Naming.rebind() method to link the implementation class with RMIREGISTRY

//Server Program

import java.rmi.\*;

import java.net.\*;

public class Server

{

public static void main(String args[]) throws RemoteException,MalformedURLException

{

Naming.rebind("test",new TestImp());

}

}

Step 4: Create a client program to connect with server

Use rmi protocol and Naming.lookup() method to search the server and get refernce of the implementation class.

import java.rmi.\*;

public class Client

{

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

{

Test t=(Test) Naming.lookup("rmi://localhost/test");

if (t==null)

{

System.out.println("Server Not found");

}

else

{

System.out.println(t.reverse("Hi to All"));

}

}

}

Step 5:

Create Stub and Skelton for marshelling and un-marshelling the arguments

Stub is for client and Skelton is for server. In Java 5.0 + Skelton is option

Use RMIC compiler to create the stub and skelton

RMIC TestImp

**Division of files**

For Server

1. Interface
2. Implementation class
3. server program

For client side

1. Interface
2. Client class
3. Stub file

**How to run an RMI Based Application**

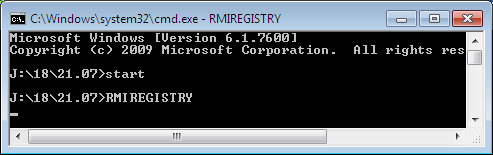
**Server Side Working**

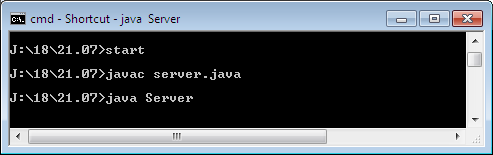
1. Start RMIREGISTRY
2. Start your server program

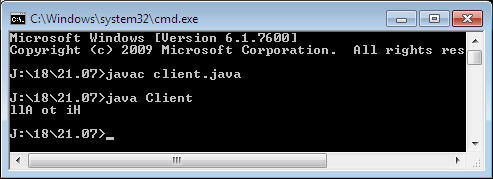
**Client Side**

Run the client program

Example:







**Running a program and display response of it**

import java.io.\*;

import java.util.\*;

class ProcessBuilderTest

{

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

{

ProcessBuilder builder = new ProcessBuilder("ipconfig");

Process javap = builder.start();

InputStreamReader tempReader = new InputStreamReader(new BufferedInputStream(javap.getInputStream()));

BufferedReader reader = new BufferedReader(tempReader);

while (true)

{

String line = reader.readLine();

if (line == null)

break;

System.out.println(line);

}

} }