JAVA NOTES

First Program:

===========

public static void main (string args[])

public ---> Visible

static ---> Memory Management

void----> It returns nothing

main()----->Default method.

strings---> Commandline arguments.

Example:

=======

class Hello

{

public static void main(String[] args)

{

System.out.println ("Hello World program");

}

}

------------------------------------------------------------------------------------

Variable:

Variable is the name of memory location.

Synatx:

DataType VariableName = Value;

Suresh------>#625378

name=#625378;

1.Instance variable

=============

class Hello

{

static String name="Suresh';

public static void main(String[] args)

{

}

}

2.Local variable

==============

class Hello

{

public static void main(String[] args)

{

String name="Suresh";

System.out.println (name);

}

}

------------------------------------------------------------------------------------

Data Types:

------------------

byte //1byte

int //4byte

short //2byte

long //8byte

double //8byte

float //4byte

boolean //no memeory allocate

char //2byte

Example:

===========

public class Demo{

public static void main(String[] args) {

// short type

short s = 20;

System.out.println("s= "+s);

// int type

int i = 20;

System.out.println("i= "+i);

// long type

long l = 20;

System.out.println("l= "+l);

// float type

float f = 20.25f;

System.out.println("f= "+f);

// double type

double d = 20.25;

System.out.println("d= "+d);

//character type

char ch = 'S';

System.out.println(ch);

}

}

Conditional Statement:

---------------------------------

Decision Making statements

**if condition :**

-----------------

if statement is used to test the condition. It checks boolean condition: true or false.

Syntax :

=======

if(Condition){

}

Example:

int time = 20;

if (time < 18) {

System.out.println("Good day.");

} else {

System.out.println("Good evening.");

}

**else if condition :**

------------------------

if-else statement also tests the condition. It executes the if block if condition is true otherwise else block is executed.

Syntax :

=======

else if(Condition){

}

Example:

=======

int time = 22;

if (time < 10) {

System.out.println("Good morning.");

} else if (time < 20) {

System.out.println("Good day.");

} else {

System.out.println("Good evening.");

}

------------------------------------------------------------------------------------

**Switch statement:**

--------------------------

switch statement executes one statement from multiple conditions

Syntax:

======

switch(DataType){

Case DataType:

Statement

break;

Case DataType:

Statement

break;

}

Example:

========

public class SwitchDemo1{

public static void main(String[] args)

{

int day = 3;

String dayName;

switch (day) {

case 1:

dayName = "Today is Monday";

break;

case 2:

dayName = "Today is Tuesday";

break;

case 3:

dayName = "Today is Wednesday";

break;

case 4:

dayName = "Today is Thursday";

break;

case 5:

dayName = "Today is Friday";

break;

case 6:

dayName = "Today is Saturday";

break;

case 7:

dayName = "Today is Sunday";

break;

default:

dayName = "Invalid day";

break;

}

System.out.println(dayName);

}

}

**Loops**

**--------------**

no. of times iterates our code or statement.

if number is fixed

**1.For**

-----------

Syntax:

======

for(initialization;condition;increment/decrement)

{

//statement

}

Example:

========

public class ForDemo1

{

public static void main(String[] args)

{

int i;

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

{

System.out.println(i);

}

}

}

**2.While**

-----------------

no. of times iterates our code or statement.

if number is not fixed.

Syntax:

======

while(condition)

{

//code for execution

}

Example:

========

public class WhileDemo1

{

public static void main(String[] args)

{

inti=1;

while(i<=10)

{

System.out.println(i);

i++;

}

}

}

**3.Do while**

----------------

no. of times iterates our code or statement.

if number is not fixed and it executes once.

Syntax:

======

do

{

//code for execution

}

while(condition);

Example:

========

public class DoWhileDemo1

{

public static void main(String[] args)

{

int i=1;

do

{

System.out.println(i);

i++;

}while(i<=10);

}

}

**Array:**

======

It is a collection of similiar data types.

**Features of Array**

==============

\*It is always indexed. Index begins from 0.

\*It is a collection of similar data types.

\*It occupies a contiguous memory location.

\*It allows to access elements randomly.

**Disadvantages:**

**===============**

\*Length is fixed .

Syntax to Declare an Array in Java

dataType[] arr; (or)

dataType []arr; (or)

dataType arr[];

Example:

==========

class Demo

{

public static void main(String[] args)

{

int[] arr = {10,20,30,40,50}; //initialization

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

{

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

}

// updating a value

arr[1] = 105;

System.out.println("element at first index: " +arr[1]);

}

}

**For-each Loop for Java Array:**

**------------------------------**

We can also print the Java array using for-each loop. The Java for-each loop prints the array elements one by one. It holds an array element in a variable, then executes the body of the loop.

The syntax:

----------

for(data\_type variable:array){

//body of the loop

}

Finding minimum value program :

-----------------------------

public static void main(String args[]){

int a[]={33,3,4,5};//declaring and initializing an array

int min=arr[0];

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

if(min>arr[i]) {

min=arr[i];

}

}

System.out.println(min);

}}

Multidimensional Array in Java:

---------------------------------

In such case, data is stored in row and column based index (also known as matrix form).

Syntax :

--------

dataType[][] arrayRefVar; (or)

dataType [][]arrayRefVar; (or)

dataType arrayRefVar[][]; (or)

dataType []arrayRefVar[];

Example:

-------

public static void main(String args[]){

//declaring and initializing 2D array

int arr[][]={{1,2,3},{2,4,5},{4,4,5}};

//printing 2D array

for(int i=0;i<3;i++){

for(int j=0;j<3;j++){

System.out.print(arr[i][j]+" ");

}

System.out.println();

}

Addition Program :

--------------------

public static void main(String args[]){

//creating two matrices

int a[][]={{1,3,4},{3,4,5}};

int b[][]={{1,3,4},{3,4,5}};

//creating another matrix to store the sum of two matrices

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

//adding and printing addition of 2 matrices

for(int i=0;i<2;i++){

for(int j=0;j<3;j++){

c[i][j]=a[i][j]+b[i][j];

System.out.print(c[i][j]+" ");

}

System.out.println();//new line

}

}

**Method:**

=========

A method is a block of code or collection of statements or a set of code grouped together to perform a certain task or operation. It is used to achieve the reusability of code. We write a method once and use it many times. We do not require to write code again and again.

Method Declaration:

==================

AccessSpecifier ReturnType MethodName (ParameterList){

//Method Body

}

Method Types:

=============

**Predefined Method:**

**-------------------**

In Java, predefined methods are the method that is already defined in the Java class libraries is known as predefined methods. It is also known as the standard library method or built-in method

**User-defined Method:**

**--------------------**

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

Example :

----------

public class Addition

{

//user defined method

public static int add(int n1, int n2) //n1 and n2 are formal parameters

{

int s;

s=n1+n2;

return s; //returning the sum

}

public static void main(String[] args)

{

int a = 19;

int b = 5;

//method calling

int c = add(a, b); //a and b are actual parameters

System.out.println("The sum of a and b is= " + c);

}

}

Java Constructors:

==================

A constructor in Java is a special method that is used to initialize objects. The constructor is called when an object of a class is created. It can be used to set initial values for object attributes.

Constructor Parameters:

-----------------------

Constructors can also take parameters, which is used to initialize attributes.

Example:

---------

class Student4{

int id;

String name;

//creating a parameterized constructor

Student4(int i,String n){

id = i;

name = n;

}

//method to display the values

void display(){System.out.println(id+" "+name);}

public static void main(String args[]){

//creating objects and passing values

Student4 s1 = new Student4(111,"Karan");

Student4 s2 = new Student4(222,"Aryan");

//calling method to display the values of object

s1.display();

s2.display();

}

}

Constructor Chaining:

----------------------

In constructor chain, a constructor is called from another constructor in the same class this process is known as constructor chaining.

Example :

----------

public class ConstructorChain

{

//default constructor

ConstructorChain()

{

this("Javatpoint");

System.out.println("Default constructor called.");

}

//parameterized constructor

ConstructorChain(String str)

{

System.out.println("Parameterized constructor called");

}

//main method

public static void main(String args[])

{

//initializes the instance of example class

ConstructorChain cc = new ConstructorChain();

}

}

Object in Java:

===============

An entity that has state and behavior is known as an object e.g., chair, bike, marker, pen, table, car, etc.

An object has three characteristics:

State : represents the data (value) of an object.

Behavior: represents the behavior (functionality) of an object such as deposit, withdraw, etc.

Identity: An object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. However, it is used internally by the JVM to identify each object uniquely.

So, an object is the instance(result) of a class.

A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.

A class in Java can contain:

Fields

Methods

Constructors

Blocks

Nested class and interface

Syntax to declare a class:

=========================

class <class\_name>{

field;

method;

}

new keyword in Java:

====================

The new keyword is used to allocate memory at runtime. All objects get memory in Heap memory area.

Ways to initialize object:

===============================

There are several ways to initialize object in Java.

1)By reference variable:

----------------------

eg:

ClassName obj=new ClassName();

2)By new Instance:

-----------------

eg:

ClassName obj=ClassName.class.newInstance();

3)By clone:

------------

eg:

ClassName obj=new ClassName();

ClassName obj2=ClassName.clone();

Types in OOPS:

=============

1)Inheritance:

--------------

Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of OOPs (Object Oriented programming system).

The syntax of Java Inheritance

class Subclass-name extends Superclass-name

{

//methods and fields

}

The extends keyword indicates that you are making a new class that derives from an existing class.

Types Of inheritance:

-----------------------

1)Single Inheritance:

-----------------

When a class inherits another class, it is known as a single inheritance.

Example:

-------

class Base

{

public void M1()

{

System.out.println( Base Class Method );

}

}

class Derived extends Base

{

public void M2()

{

System.out.printIn( Derived Class Methods );

}

}

class Test

{

public static void main(String[] args)

{

Derived d = new Derived(); // creating object

d.M1(); // print Base Class Method

d.M2(); // print Derived Class Method

}

}

Multilevel Inheritance:

-----------------------

When there is a chain of inheritance, it is known as multilevel inheritance.

Hierarchical Inheritance:

---------------------------

When two or more classes inherits a single class, it is known as hierarchical inheritance

Multiple Inheritance:

--------------------

To reduce the complexity and simplify the language, multiple inheritance is not supported in java.

Consider a scenario where A, B, and C are three classes. The C class inherits A and B classes.

If A and B classes have the same method and you call it from child class object, there will be ambiguity to call the method of A or B class.

PolyMorphism:

---------------

Poly ----> Many

Morph -----> Form

Polymorphism in Java is a concept by which we can perform a single action in different ways. Polymorphism is derived from 2 Greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So polymorphism means many forms.

There are two types of polymorphism in Java: compile-time polymorphism and runtime polymorphism. We can perform polymorphism in java by method overloading and method overriding.

Types:

------

1 ) Compile Time PolyMorphism.(Method overloading)

If a class has multiple methods having same name but different in parameters, it is known as Method Overloading

Example :

---------

public class HelloWorld {

void m() {

System.out.println("m1");

}

void m(int i) {

System.out.println("m2");

}

void m(String i) {

System.out.println("m3");

}

public static void main(String[] args){

HelloWorld obj=new HelloWorld();

obj.m(10); //it invokes at compile time , then it is called CompileTime polymorphism

}

}

2 ) Run Time Polymorphism(Method Overriding).

If subclass (child class) has the same method as declared in the parent class, it is known as method overriding in Java.

Example :

---------

public class Animal {

void walk() {

System.out.println("Walk");

}

void m() {

System.out.println("Parent");

}

void bark() {

System.out.println("Bark");

}

void smart() {

System.out.println("Smart");

}

}

public class Dog extends Animal{

void eat() {

System.out.println("Eat");

}

void m() {

System.out.println("Child");

}

public static void main(String[] args) {

Animal obj=new Dog();//upcasting

obj.m(); //it invokes at run time , then it is called RunTime polymorphism

}

}

Upcasting:

------------

Upcasting is a type of object typecasting in which a child object is typecasted to a parent class object. By using the Upcasting, we can easily access the variables and methods of the parent class to the child class.

Downcasting:

-------------

Upcasting is another type of object typecasting. In Upcasting, we assign a parent class reference object to the child class. In Java, we cannot assign a parent class reference object to the child class, but if we perform downcasting, we will not get any compile-time error. However, when we run it, it throws the "ClassCastException".

Compile Time Polymorphism:

--------------------------

Whenever an object is bound with their functionality at the compile-time, this is known as the compile-time polymorphism. At compile-time, java knows which method to call by checking the method signatures. So this is called compile-time polymorphism or static or early binding.

Runtime Polymorphism:

------------------------

Runtime polymorphism in java is also known as Dynamic Binding or Dynamic Method Dispatch. In this process, the call to an overridden method is resolved dynamically at runtime rather than at compile-time. Runtime polymorphism is achieved through Method Overriding.

Encapsulation:

--------------

Encapsulation in Java is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit. In encapsulation, the variables of a class will be hidden from other classes, and can be accessed only through the methods of their current class.

eg:

getters and setters

public class HelloWorld {//bean class

private int i=10;

public int getI() {//getter method

return i;

}

public void setI(int i) { //setter method

this.i = i;

}

}

public class Class2 {

public static void main(String[] args) {

HelloWorld obj=new HelloWorld();

System.out.println(obj.getI()+" before ");

obj.setI(20);

System.out.println(obj.getI()+" after ");

}

}

Abstraction:

------------

Abstraction is a process of hiding the implementation details and showing only functionality to the user.

Ways to achieve Abstraction

There are two ways to achieve abstraction in java

1) Abstract class (0 to 100%)

2) Interface (100%)

Abstract class:

----------------

A class which is declared as abstract is known as an abstract class. It can have abstract and non-abstract methods. It needs to be extended and its method implemented. It cannot be instantiated.

Points to Remember

------------------

An abstract class must be declared with an abstract keyword.

It can have abstract and non-abstract methods.

It cannot be instantiated.

It can have constructors and static methods also.

It can have final methods which will force the subclass not to change the body of the method.

Syntax:

-------

abstract className{}

abstract return type MethodName{}

Example:

---------

public abstract class Bike {

void brake() {

System.out.println("brake");

}

abstract void speed();

}

public class Honda extends Bike{

@Override

void speed() {

System.out.println("60Km");

}

public static void main(String[] args) {

Honda obj=new Honda();

obj.brake();

obj.speed();

}

}

public class Suzuki extends Bike{

@Override

void speed() {

System.out.println("50Km");

}

public static void main(String[] args) {

Suzuki obj=new Suzuki();

obj.brake();

obj.speed();

}

}

Interface:

-----------

An interface in Java is a blueprint of a class. It has static constants and abstract methods.

The interface in Java is a mechanism to achieve abstraction. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java.

Syntax:

-------

interface <interface\_name>{

// declare constant fields

// declare methods that abstract

// by default.

}

//interface picture

Example:

---------

public interface Mobile2 {

int i=10;

int j=20;

void ram2();

void battery2();

}

public interface Mobile extends Mobile2 {

int i=10;

int j=20;

void ram();

void battery();

}

public class Samsung implements Mobile{

@Override

public void ram() {

System.out.println("128GB");

}

@Override

public void battery() {

System.out.println("3000Mah");

}

@Override

public void ram2() {

// TODO Auto-generated method stub

}

@Override

public void battery2() {

// TODO Auto-generated method stub

}

}

public class Realme implements Mobile{

@Override

public void ram() {

System.out.println("32GB");

}

@Override

public void battery() {

System.out.println("1000Mah");

}

@Override

public void ram2() {

System.out.println("1000Mah");

}

@Override

public void battery2() {

System.out.println("1000Mah");

}

}

Marker Interface:

-----------------

An interface which has no member is known as a marker or tagged interface.

Achieve Multiple Inheritance:

-----------------------------

If a class implements multiple interfaces, or an interface extends multiple interfaces, it is known as multiple inheritance.

Difference between Abstract Class and Interface:

------------------------------------------------

1 ) An abstract class can have both abstract and non-abstract methods , whereas the interface can have only abstract methods.

2) Abstract class does not support multiple inheritance , whereas the interface supports multiple inheritances.It can provide the implementation of the interface.It can not provide the implementation of the abstract class.

3) An abstract class can have protected and abstract public methods , whereas the interface can have only have public abstract methods.

4) An abstract class can have final, static, or static final variable with any access specifier, whereas the interface can only have a public static final variable.

Collections Framework:

----------------------

The Collection in Java is a framework that provides an architecture to store and manipulate the group of objects.

Java Collections can achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation, and deletion.

Java Collection means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet).

List Interface:

----------------

List interface is the child interface of Collection interface.It can have duplicate values.

List interface is implemented by the classes ArrayList, LinkedList, Vector, and Stack.

To instantiate the List interface, we must use :

List <data-type> list1= new ArrayList();

List <data-type> list2 = new LinkedList();

List <data-type> list3 = new Vector();

List <data-type> list4 = new Stack();

ArrayList:

--------------

The ArrayList class implements the List interface. It uses a dynamic array to store the duplicate element of different data types. The ArrayList class maintains the insertion order and is non-synchronized. The elements stored in the ArrayList class can be randomly accessed. Consider the following example.

import java.util.\*;

class TestJavaCollection1{

public static void main(String args[]){

ArrayList<String> list=new ArrayList<String>();//Creating arraylist

list.add("Ravi");//Adding object in arraylist

list.add("Vijay");

list.add("Ravi");

list.add("Ajay");

//Traversing list through Iterator

Iterator itr=list.iterator();

while(itr.hasNext()){

System.out.println(itr.next());

}

}

}

LinkedList:

-----------

LinkedList implements the Collection interface. It can store the duplicate elements. It maintains the insertion order and is not synchronized. In LinkedList, the manipulation is fast.

Consider the following example.

import java.util.\*;

public class TestJavaCollection2{

public static void main(String args[]){

LinkedList<String> al=new LinkedList<String>();

al.add("Ravi");

al.add("Vijay");

al.add("Ravi");

al.add("Ajay");

Iterator<String> itr=al.iterator();

while(itr.hasNext()){

System.out.println(itr.next());

}

}

}

Vector:

-------

Vector uses a dynamic array to store the data elements. It is similar to ArrayList. However, It is synchronized and contains many methods that are not the part of Collection framework.

Consider the following example.

import java.util.\*;

public class TestJavaCollection3{

public static void main(String args[]){

Vector<String> v=new Vector<String>();

v.add("Ayush");

v.add("Amit");

v.add("Ashish");

v.add("Garima");

Iterator<String> itr=v.iterator();

while(itr.hasNext()){

System.out.println(itr.next());

}

}

}

Stack:

------

The stack is the subclass of Vector. It implements the last-in-first-out data structure.

Consider the following example.

import java.util.\*;

public class TestJavaCollection4{

public static void main(String args[]){

Stack<String> stack = new Stack<String>();

stack.push("Ayush");

stack.push("Garvit");

stack.push("Amit");

stack.push("Ashish");

stack.push("Garima");

stack.pop();

Iterator<String> itr=stack.iterator();

while(itr.hasNext()){

System.out.println(itr.next());

}

}

}

pop Method:

------------

pop() method in Java is used to pop an element from the stack. The element is popped from the top of the stack and is removed from the same

Queue Interface:

----------------

Queue interface maintains the first-in-first-out order. It can be defined as an ordered list that is used to hold the elements which are about to be processed.

PriorityQueue:

------------

The PriorityQueue class implements the Queue interface. It holds the elements or objects which are to be processed by their priorities. PriorityQueue doesn't allow null values to be stored in the queue.

Consider the following example.

import java.util.\*;

public class TestJavaCollection5{

public static void main(String args[]){

PriorityQueue<String> queue=new PriorityQueue<String>();

queue.add("Amit Sharma");

queue.add("Vijay Raj");

queue.add("JaiShankar");

queue.add("Raj");

System.out.println("head:"+queue.element());

System.out.println("head:"+queue.peek());

System.out.println("iterating the queue elements:");

Iterator itr=queue.iterator();

while(itr.hasNext()){

System.out.println(itr.next());

}

queue.remove();

queue.poll();

System.out.println("after removing two elements:");

Iterator<String> itr2=queue.iterator();

while(itr2.hasNext()){

System.out.println(itr2.next());

}

}

}

The remove() and poll() methods differ only in their behavior when the queue is empty: the remove() method throws an exception, while the poll() method returns null. The element() and peek() methods return, but do not remove, the head of the queue.

Dequeue:

----------

Deque interface extends the Queue interface. In Deque, we can remove and add the elements from both the side

Array Dequeue:

--------------

ArrayDeque class implements the Deque interface.ArrayDeque is faster than ArrayList and Stack and has no capacity restrictions.

Deque<String> deque

= new LinkedList<String>();

// We can add elements to the queue

// in various ways

// Add at the last

deque.add("Element 1 (Tail)");

// Add at the first

deque.addFirst("Element 2 (Head)");

// Add at the last

deque.addLast("Element 3 (Tail)");

// Add at the first

deque.push("Element 4 (Head)");

// Add at the last

deque.offer("Element 5 (Tail)");

// Add at the first

deque.offerFirst("Element 6 (Head)");

System.out.println(deque + "\n");

// We can remove the first element

// or the last element.

deque.removeFirst();

deque.removeLast();

System.out.println("Deque after removing "

+ "first /and last: "

+ deque);

}

}

Set Interface:

--------------

Set Interface in Java is present in java.util package. It extends the Collection interface. It represents the unordered set of elements which doesn't allow us to store the duplicate items. We can store at most one null value in Set. Set is implemented by HashSet, LinkedHashSet, and TreeSet.

Set can be instantiated as:

Set<data-type> s1 = new HashSet<data-type>();

Set<data-type> s2 = new LinkedHashSet<data-type>();

Set<data-type> s3 = new TreeSet<data-type>();

HashSet:

-------

HashSet class implements Set Interface. It represents the collection that uses a hash table for storage. Hashing is used to store the elements in the HashSet. It contains unique items.

Consider the following example.

import java.util.\*;

public class TestJavaCollection7{

public static void main(String args[]){

//Creating HashSet and adding elements

HashSet<String> set=new HashSet<String>();

set.add("Ravi");

set.add("Vijay");

set.add("Ravi");

set.add("Ajay");

//Traversing elements

Iterator<String> itr=set.iterator();

while(itr.hasNext()){

System.out.println(itr.next());

}

}

}

LinkedHashSet:

-------------

LinkedHashSet class represents the LinkedList implementation of Set Interface. It extends the HashSet class and implements Set interface. Like HashSet, It also contains unique elements. It maintains the insertion order and permits null elements.

Consider the following example.

import java.util.\*;

public class TestJavaCollection8{

public static void main(String args[]){

LinkedHashSet<String> set=new LinkedHashSet<String>();

set.add("Ravi");

set.add("Vijay");

set.add("Ravi");

set.add("Ajay");

Iterator<String> itr=set.iterator();

while(itr.hasNext()){

System.out.println(itr.next());

}

}

}

SortedSet Interface:

--------------------

SortedSet is the alternate of Set interface that provides a total ordering on its elements. The elements of the SortedSet are arranged in the increasing (ascending) order.

TreeSet:

--------

Java TreeSet class implements the Set interface that uses a tree for storage. Like HashSet, TreeSet also contains unique elements. However, the access and retrieval time of TreeSet is quite fast. The elements in TreeSet stored in ascending order.

Consider the following example:

import java.util.\*;

public class TestJavaCollection9{

public static void main(String args[]){

//Creating and adding elements

TreeSet<String> set=new TreeSet<String>();

set.add("Ravi");

set.add("Vijay");

set.add("Ravi");

set.add("Ajay");

//traversing elements

Iterator<String> itr=set.iterator();

while(itr.hasNext()){

System.out.println(itr.next());

}

}

}

Java Map Interface:

------------------

A map contains values on the basis of key, i.e. key and value pair. Each key and value pair is known as an entry. A Map contains unique keys.

A Map is useful if you have to search, update or delete elements on the basis of a key.

//Map Picture

Types same as Set.

1.HashMap

2.LinkedHashMap

3.TreeMap

import java.util.\*;

class MapExample2{

public static void main(String args[]){

Map<Integer,String> map=new HashMap<Integer,String>();

map.put(100,"Amit");

map.put(101,"Vijay");

map.put(102,"Rahul");

//Elements can traverse in any order

for(Map.Entry m:map.entrySet()){

System.out.println(m.getKey()+" "+m.getValue());

}

}

}

entrySet():

----------

entrySet()method in Java is used to create a set out of the same elements contained in the hash map. It basically returns a set view of the hash map or we can create a new set and store the map elements into them.

Exception Handling:

-------------------

The Exception Handling in Java is one of the powerful mechanism to handle the runtime errors so that the normal flow of the application can be maintained.

Exception:

----------

An exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.

Types of Java Exceptions:

-------------------------

1.Checked Exception

2.Unchecked Exception

3.Error

1) Checked Exception:

---------------------

The classes that directly inherit the Throwable class except RuntimeException and Error are known as checked exceptions. For example, IOException, SQLException, etc. Checked exceptions are checked at compile-time.

2) Unchecked Exception:

----------------------

The classes that inherit the RuntimeException are known as unchecked exceptions. For example, ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException, etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.

3) Error:

---------

Error is irrecoverable. Some example of errors are OutOfMemoryError, VirtualMachineError, AssertionError etc.

Java Exception Keywords:

------------------------

Java provides five keywords that are used to handle the exception.

1)try:

----

The "try" keyword is used to specify a block where we should place an exception code. It means we can't use try block alone. The try block must be followed by either catch or finally.

2)catch:

The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later.

finally:

--------

The "finally" block is used to execute the necessary code of the program. It is executed whether an exception is handled or not.

Example:

--------

public class TestFinallyBlock1{

public static void main(String args[]){

try {

System.out.println("Inside the try block");

//below code throws divide by zero exception

int data=25/0;

System.out.println(data);

}

//cannot handle Arithmetic type exception

//can only accept Null Pointer type exception

catch(NullPointerException e){

System.out.println(e);

}

//executes regardless of exception occured or not

finally {

System.out.println("finally block is always executed");

}

System.out.println("rest of the code...");

}

}

throw:

------

The "throw" keyword is used to throw an exception.

Example:

-------

public class TestThrow1 {

//function to check if person is eligible to vote or not

public static void validate(int age) {

if(age<18) {

//throw Arithmetic exception if not eligible to vote

throw new ArithmeticException("Person is not eligible to vote");

}

else {

System.out.println("Person is eligible to vote!!");

}

}

//main method

public static void main(String args[]){

//calling the function

validate(13);

System.out.println("rest of the code...");

}

}

throws:

------

The "throws" keyword is used to declare exceptions. It specifies that there may occur an exception in the method. It doesn't throw an exception. It is always used with method signature.

Example:

--------

class M{

void method()throws IOException{

throw new IOException("device error");

}

}

Java Custom Exception:

----------------------

In Java, we can create our own exceptions that are derived classes of the Exception class. Creating our own Exception is known as custom exception or user-defined exception.

Example:

--------

class InvalidAgeException extends Exception{

public InvalidAgeException(String str) {

super(str);

}

}

public class Keywords {

public static void main(String[] args) throws NullPointerException, InvalidAgeException{

int age=10;

if(age>18) {

System.out.println("Eligible");

}

else {

throw new InvalidAgeException("Invalid age");

}

}

}

Throw Keyword vs Throws Keyword.

--------------------------------

1.Java throw keyword is used throw an exception explicitly in the code, inside the function or the block of code, but Java throws keyword is used in the method signature to declare an exception which might be thrown by the function while the execution of the code.

2.The throw keyword is followed by an instance of Exception to be thrown, whereas The throws keyword is followed by class names of Exceptions to be thrown.

3.throw is used within the method, whereas throws is used with the method signature.

4.We are allowed to throw only one exception at a time i.e. we cannot throw multiple exceptions, whereas We can declare multiple exceptions using throws keyword that can be thrown by the method. For example, main() throws IOException, SQLException.

final vs finally vs finalize:

------------------------------

\*final is the keyword and access modifier which is used to apply restrictions on a class, method or variable.

\*finally is the block in Java Exception Handling to execute the important code whether the exception occurs or not.

\*finalize is the method in Java which is used to perform clean up processing just before object is garbage collected.

web application:

----------------

A web application is an application accessible from the web. A web application is composed of web components like Servlet, JSP, Filter, etc. and other elements such as HTML, CSS, and JavaScript. The web components typically execute in Web Server and respond to the HTTP request.

A website can be of two types:

\*Static Website

\*Dynamic Website

Static website:

-------------------

Static website is the basic type of website that is easy to create. You don't need the knowledge of web programming and database design to create a static website. Its web pages are coded in HTML.

The codes are fixed for each page so the information contained in the page does not change and it looks like a printed page.

Dynamic website:

----------------

Dynamic website is a collection of dynamic web pages whose content changes dynamically. It accesses content from a database. Therefore, when you alter or update the content of the database, the content of the website is also altered or updated.

Dynamic website uses client-side scripting or server-side scripting, or both to generate dynamic content.

//picture

JSP ----------> Presenting Logic / View Logic

Servlets -----> Processig Logic / Business Logic

JDBC ---------> DataBase Connectivity

JDBC:

---------------------------------------------------------------------------------

JDBC stands for Java Database Connectivity. JDBC is a Java API to connect and execute the query with the database. It is a part of JavaSE.

API:

---

API is the acronym for Application Programming Interface, which is a software intermediary that allows two applications to talk to each other.

Driver:

------

1) JDBC-ODBC bridge driver(Bridge driver);

-----------------------------------------

The JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. This is now discouraged because of thin driver.

//picture

Advantages:

------------

\*easy to use.

\*Database Independent Driver

Disadvantages:

--------------

\*Performance degraded because JDBC method call is converted into the ODBC function calls.

\*Its only applicable for windows.

2) Native-API driver:

----------------------

The Native API driver uses the client-side libraries of the database. The driver converts JDBC method calls into native calls of the database API. It is not written entirely in java.

//picture

Advantage:

-----------

\*performance upgraded than JDBC-ODBC bridge driver.

Disadvantage:

------------

\*Database dependent

\*Not all vendour provides Native libraries

3)Network Protocol driver(Fully Java Driver):

-------------------------------------------

The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is fully written in java.

//picture

Advantage:

-----------

\*Fully written in java.

\*no need of library

\*Multiple database access.

Disadvantages:

--------------

\*It requires another server application to install and resukt set may take longe , since the data comes through database.

4) Thin driver:

---------------

The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as thin driver. It is fully written in Java language.

//picture

Advantage:

---------

\*Better performance than all other drivers.

\*No software is required at client side or server side.

Disadvantage:

--------------

\*Drivers depend on the Database.

Steps to connect Database:

--------------------------

There are 5 steps to connect any java application with the database using JDBC. These steps are as follows:

1)Register the Driver class:

----------------------------

The forName() method of Class class is used to register the driver class. This method is used to dynamically load the driver class.

2)Create connection:

--------------------

The getConnection() method of DriverManager class is used to establish connection with the database.

3)Create statement:

-------------------

The createStatement() method of Connection interface is used to create statement. The object of statement is responsible to execute queries with the database.

4)Execute queries:

------------------

The executeQuery() method of Statement interface is used to execute queries to the database. This method returns the object of ResultSet that can be used to get all the records of a table.

5)Close connection:

------------------

By closing connection object statement and ResultSet will be closed automatically. The close() method of Connection interface is used to close the connection.

CRUD Operations Using JDBC:

---------------------------

PreRequisite:

-------------

SQL queries:

---------------

SQL is a standard language for storing, manipulating and retrieving data in databases.

Create DB:

----------

CREATE DATABASE databasename;

Create Table :

--------------

CREATE TABLE table\_name (

column1 datatype,

column2 datatype,

column3 datatype,

....

);

Drop Database;

-------------

DROP DATABASE databasename;

Drop Table:

------------

DROP TABLE tablename;

CRUD:

-----

1)Select :

----------

The SELECT statement is used to select data from a database.

The data returned is stored in a result table, called the result-set.

Syntax:

-------

SELECT column1, column2, ...

FROM table\_name;

2)INSERT:

-----------

The INSERT INTO statement is used to insert new records in a table.

Syntax:

-------

It is possible to write the INSERT INTO statement in two ways:

1. Specify both the column names and the values to be inserted:

INSERT INTO table\_name (column1, column2, column3, ...)

VALUES (value1, value2, value3, ...);

(OR)

INSERT INTO table\_name

VALUES (value1, value2, value3, ...);

3)UPDATE:

---------

The UPDATE statement is used to modify the existing records in a table.

Syntax:

---------

UPDATE table\_name

SET column1 = value1, column2 = value2, ...

WHERE condition;

4)DELETE:

----------

The DELETE statement is used to delete existing records in a table.

Syntax:

-------

DELETE FROM table\_name WHERE condition;

Prepared Statement:

==================

A prepared statement is a feature used to execute the same (or similar) SQL statements repeatedly with high efficiency.

CRUD in JDBC :

--------------

1)Insert:

----------

static final String DB\_URL = "jdbc:mysql://localhost/TUTORIALSPOINT";

static final String USER = "guest";

static final String PASS = "guest123";

public static void main(String[] args) {

// Open a connection

try(Connection conn = DriverManager.getConnection(DB\_URL, USER, PASS);

Statement stmt = conn.createStatement();

) {

// Execute a query

System.out.println("Inserting records into the table...");

String sql = "INSERT INTO Registration VALUES (100, 'Zara', 'Ali', 18)";

stmt.executeUpdate(sql);

System.out.println("Inserted records into the table...");

} catch (SQLException e) {

e.printStackTrace();

}

}

2)Fetch:

---------

static final String DB\_URL = "jdbc:mysql://localhost/TUTORIALSPOINT";

static final String USER = "guest";

static final String PASS = "guest123";

static final String QUERY = "SELECT id, first, last, age FROM Registration";

public static void main(String[] args) {

// Open a connection

try(Connection conn = DriverManager.getConnection(DB\_URL, USER, PASS);

Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery(QUERY);

) {

while(rs.next()){

//Display values

System.out.print("ID: " + rs.getInt(1));

System.out.print(", Age: " + rs.getInt(2));

System.out.print(", First: " + rs.getString(3));

System.out.println(", Last: " + rs.getString(4));

}

} catch (SQLException e) {

e.printStackTrace();

}

3)Update:

---------

static final String DB\_URL = "jdbc:mysql://localhost/TUTORIALSPOINT";

static final String USER = "guest";

static final String PASS = "guest123";

public static void main(String[] args) {

// Open a connection

try(Connection conn = DriverManager.getConnection(DB\_URL, USER, PASS);

Statement stmt = conn.createStatement();

) {

// Execute a query

System.out.println("Inserting records into the table...");

String sql = "UPDATE Registration SET AGE=23 WHERE ID=7";

stmt.executeUpdate(sql);

System.out.println("Record is Updated...");

} catch (SQLException e) {

e.printStackTrace();

}

}

4)Delete:

----------

static final String DB\_URL = "jdbc:mysql://localhost/TUTORIALSPOINT";

static final String USER = "guest";

static final String PASS = "guest123";

public static void main(String[] args) {

// Open a connection

try(Connection conn = DriverManager.getConnection(DB\_URL, USER, PASS);

Statement stmt = conn.createStatement();

) {

// Execute a query

System.out.println("Inserting records into the table...");

String sql = "DELETE FROM Registration WHERE ID=7";

stmt.executeUpdate(sql);

System.out.println("Record is Deleted...");

} catch (SQLException e) {

e.printStackTrace();

}

}

Client - Server Architecture:

--------------------------

\*Client

\*Protocol

\*Server

//picture

1)Client:

----------

In Web applications , client is fixed, which is browser.So, browser is sent a request to server and get a response from server.

We can sent request using URI (Uniform Resource Identifier).

URI:

----

URI (uniform resource identifier) identifies a resource (text document, image file, etc)

1)URL(Uniform Resource Locator):

------------------------------

A URL (Uniform Resource Locator) is a unique identifier used to locate a resource on the Internet. It is also referred to as a web address.

2)URN (Uniform Resource Name):

------------------------------

is a subset of URIs that include a name within a given space, but no location

eg:

protocol\_name://server\_ip\_address:server\_portno/App\_name/res\_path?field=value

Ip Address:

----------

An IP address is a unique address that identifies a device on the internet or a local network.

Port number:

------------

A port number is a way to identify a specific process to which an internet or other network message is to be forwarded when it arrives at a server.

query string:

-------------

It is a collection os name-value pair , which is provided along with url and which

provide some data to the server side.

2) Protocol:

-----------

Protocol is a set of rules or procedures for transmitting data between client and server.

In client - server applications , to carry request from client to server and get response from server to client we need a protocol.

HTTP:

-----

The Hypertext Transfer Protocol (HTTP) is application-level protocol for collaborative, distributed, hypermedia information systems. It is the data communication protocol used to establish communication between client and server.

why HTTP stateless?:

--------------------

Connections between the client and te server remains for the current request only.Next time it will treat as a new connection for a new request.

Methods:

---------

1.GET:

------

The GET method is used to retrieve information from the given server using a given URI. Requests using GET should only retrieve data and should have no other effect on the data.

2.HEAD:

------

Same as GET, but transfers the status line and header section only.

3.POST:

-------

A POST request is used to send data to the server, for example, customer information, file upload, etc. using HTML forms.

4.PUT:

-----

Replaces all current representations of the target resource with the uploaded content.

5.DELETE:

---------

Removes all current representations of the target resource given by a URI.

Difference between get and post:

---------------------------------

1) In case of Get request, only limited amount of data can be sent because data is sent in header. In case of post request, large amount of data can be sent because data is sent in body.

2) Get request is not secured because data is exposed in URL bar.Post request is secured because data is not exposed in URL bar.

3) Get request is more efficient and used more than Post.Post request is less efficient and used less than get.

Http codes:

-----------

1) 1xx: Informational

It means the request has been received and the process is continuing.

2) 2xx: Success

It means the action was successfully received, understood, and accepted.

3) 3xx: Redirection

It means further action must be taken in order to complete the request.

4) 4xx: Client Error

It means the request contains incorrect syntax or cannot be fulfilled.

5) 5xx: Server Error

It means the server failed to fulfill an apparently valid request.

HTTP Header:

------------

The HTTP Header contains information about the HTTP Body and the Request/Response.

Information about the body is related to the content of the Body such as the length of the content inside the body.

HTTP Body:

----------

The content/message body which is the actual data you want to fetch. The content can contain your HTML code or an image or your CSS stylesheets, JavaScript files depending on the resource for which you have made the request for.

//picture

3) Server:

---------

\* Server is a special software installed in a machine, which is used to

1. manage server side applications(web applications).

2. to take request from clients.

3. to identify the requested resources.

4. to generate response

5. to dispatch response to client.

\* eg. tomcat,GlassFish,Jboss

Servlets:

---------

\* Servlet technology is used to create a web application (resides at server side and generates a dynamic web page).

\* Servlet technology is robust and scalable because of java language and called as a server-side programming language.

Containers:

----------

\* It is used to handle dynamic pages.

\* It contains the servlet components or resources.

Deployment Descriptor:

----------------------

\* Java web applications use a deployment descriptor file to determine how URLs map to servlets, which URLs require authentication, and other information.

\* This file is named web. xml , and resides in the app's WAR under the WEB-INF/ directory.

Example:

-------

<servlet>

<servlet-name>comingsoon</servlet-name>

<servlet-class>mysite.server.ComingSoonServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>comingsoon</servlet-name>

<url-pattern>/\*</url-pattern>

</servlet-mapping>

Servlet Creation:

----------------

We can create servlet by following ways:

\* implementing Servlet interface

\* extending Generic Servlet Abstract Class.

\* extending Http Servlet Abstract Class.

Life Cycle of servlets:

======================

1.Load Servlet Class:

--------------------

Servlet container load servlet class.

2.Create Servlet Instance:

---------------------------

Servlet container create an instance of the servlet.

3.Call init():

--------------

\* Servlet contains will call init() only once.

\* init() method is used to create or load some data that will be used throughout the life of the servlet.

4.Call service():

----------------

\* The service() method is the main method to perfom the actual task.

\* It can call multiple times.

5.call destroy():

-----------------

\* When a servlet is unloaded by the servlet container, its destroy() method is called.

\* It is also called only once.

\* The servlet is unloaded while the container shutdown or reload the web app at runtime.

HttpServlet:

============

\* It is protocol dependent.

\* It has doget() and doPost().

Example:

--------

1. Request Dispatcher:

------------------

\* The RequestDispatcher interface provides the facility of dispatching the request to another resource it may be html, servlet or jsp.

\* This interface can also be used to include the content of another resource also.

Methods of RequestDispatcher interface:

--------------------------------------

The RequestDispatcher interface provides two methods. They are:

a) public void forward(ServletRequest request,ServletResponse response)throws ServletException,java.io.IOException:

Forwards a request from a servlet to another resource (servlet, JSP file, or HTML file) on the server.

b) public void include(ServletRequest request,ServletResponse response)throws ServletException,java.io.IOException:

Includes the content of a resource (servlet, JSP page, or HTML file) in the response.

//Picture

Example:

--------

2. Send Redirect:

------------------

\* The sendRedirect() method of HttpServletResponse interface can be used to redirect response to another resource, it may be servlet, jsp or html file.

\* It accepts relative as well as absolute URL.

//picture

Difference between forward() and sendRedirect() method:

------------------------------------------------------

1. The forward() method works at server side but, The sendRedirect() method works at client side.

2.forward() sends the same request and response objects to another servlet where sendRedirect() always sends a new request.

3. forward() can work within the server only, whereas sendRedirect() can be used within and outside the server.

Example:

1.Servlet Config:

-----------------

\* For each servlet ,web container creates one servlet config object.

\* Servlet Congif object will be created at the time of servlet object creation and destroyed at the time of servlet object destruction.

\* Web container hand-over the config object to th servlet as argument to init().

\* This object can be used to get configuration information from web.xml file.

Example:

--------

2. Servlet Context:

-------------------

\* For each web application. web container creates one servlet context object to hold web application level configuration information.

\* Context object will be created at the time of application deployment and destroyed atthe time of application undeployment.

\* This object can be used to get configuration information from web.xml file.

\* There is only one ServletContext object per web application.

//picture

Example:

--------

Note:

-----

if you want to share attribute to another servlet use setAtribute in context object.

Example:

Session:

-------

\* Session simply means a particular interval of time.

\* Session Tracking is a way to maintain state (data) of an user. It is also known as session management in servlet.

\* Http protocol is a stateless so we need to maintain state using session tracking techniques.

1.HttpSession:

--------------

\* Container creates a session id for each user.The container uses this id to identify the particular user.

\* An object of HttpSession can be used to perform two tasks:

1)bind objects

2)view and manipulate information about a session, such as the session identifier, creation time, and last accessed time.

//picture

Example:

--------

2.Cookie:

---------

\* Cookie is a small piece of information that is persisted between the multiple client request.

\* It stored in browser as cache.

\* It has two types:

1)Non-persistant: It is valid for single sesion only.It is removed each time when the browser is closed.

2)Persistant : It is valid for multiple session. It is not removed each time when the browser is closed.It is remove

only if user logout or signout.

//picture

Example:

--------

3. Hidden Form Field:

---------------------

\* Hidden Form Field a hidden (invisible) textfield is used for maintaining the state of an user.

\* we store the information in the hidden field and get it from another servlet.

//picture

Example:

--------

4.URL Rewriting:

----------------

\* In URL rewriting, we append a token or identifier to the URL of the next Servlet or the next resource.

\* We can send parameter name/value pairs using the following format:

eg:url?name1=value1&name2=value2&??

//picture

Example:

--------