**Day 1 : 11-01-2020**

**Phase 1 :**

Agile

Git

Java : Basic Programming

Java OOPS Concept

Exception handling

Multithreading

File Handling

Collection Framework and Data Structure

Maven

**Agile Methodology**

Application or Software

**SDLC : Software Development Life cycle**

90’s

**Waterfall Model**

1. Requirement Gathering and documentation
2. System design (look and feel)
3. Implementation or functionality using any language
4. Testing
5. Run or deploy or delivery
6. Maintenance

**Agile Methodology**

Moving fast, be flexible and response the change according to client requirements.

**Agile is concept.**

**Agile manifesto 4 :**

1. Individual and interaction -- > Process and tools
2. Working product – Comprehensive documentation
3. Customer Collaboration
4. Responding to changes

12 Agile Principles

1. Customer Satisfaction
2. Welcome changes
3. Deliver frequently
4. Work together : DevOps
5. Motivated teams
6. Face – to – Face

Etc

Implementation of Agile

1. XP
2. Lean
3. Scrum
4. Crystal

**What is Scrum :**

Agile is a set of methods and practise that focuses on iterative developments.

Scrum :

1. Product Owner : Interact with the client. It maintain the **product backlogs**. (functionality of the projects).
2. Scrum Master : The scrum master helps to team and apply the scrum to obtains business. He is only responsible for the scrum process.
3. Scrum Team : They are cross platform developers.

7 to 9 or 10 members

Sprints : 2 to 4 week

Daily they conduct scrum meeting : 10 to 15 minutes

**Spring backlogs**

**Git**

**Centralized Repository**

GitHub

Code Commit : AWS

GitLab

Azure

Google cloud

First create the Folder

Then open git bash terminal

**git init**

**git add filename.txt**

or

**git add .**

When we add the file using git add then file is present in git **staging area.**

**Staging area :** before commit the file the local repository

**git commit –m “message”**

**Java : Java is a pure object oriented and platform independent programming language.**

**The initial name of the Java is Oak. Java developed in 1991.**

**But in Nov 1995 they rename from Oak to Java.**

**It was belong to Sun Microsystem but not it is a part of Oracle.**

**Java was developed by James Gosling and Team.**

**Version of Java**

**1.0, 1.2. Java 5, 6, 7 ---🡪 Java 8, 9, 10,11,12,13, 14 Version**

**Features of Java**

1. **Java is Simple : Pointer, Memory Management, Virtual Keyword, Friend functions, destructor etc.**
2. **Pure object oriented programming language :**
3. **Platform independent programming language.**
4. **Compiler and Interpreter**
5. **Exception Handling**
6. **Multi threading**

**Open the command prompt**

**Then**

**java –version**

**javac : java compiler**

**java : java interpreter**

**class syntax**

**class className {**

**fields or variables;**

**methods;**

**main (pre-defined methods)**

**}**

In Java ClassName always follow Pascal Naming rules

1. If class contains one word then first letter of name must upper case.
2. If class contains more than one word. Then each word first letter must be upper case.

Example

Demo

Test

Employee

EmployeeDetails

ManagerInfo

To compile the Java program

**javac classname.java**

To run the program

**java className**

**Demo.java**

class Demo {

public static void main(String args[]) {

System.out.println("Welcome Java...");

System.out.print("Welcome Java...");

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

}

}

**Data Types :** Data type is a type of data which tells what type of value it can hold.

Data types mainly divided into two types.

1. Primitive data types
2. Non primitive data types or reference data types.

**Primitive data types:** This type of data type is use to store only value.

8 types

1. byte 1 byte
2. short 2
3. int 4
4. long 8 : it is use to store value without decimal point
5. float 4
6. double 8 : it is use to store the value with decimal point
7. char 2 : any single character
8. boolean 1bit : true or false.

**Data type example**

class DataTypeDemo {

public static void main(String args[]) {

int a=10;

double b = 20.20;

char c ='$';

boolean res = true;

System.out.println(a);

System.out.println(b);

System.out.println(c);

System.out.println(res);

}

}

**Type casting**

Converting one type to another is known as type casting.

2 types

1. Implicit : Internally convert
2. Explicit : We have to covert

Int family

----------------------🡪Implicit type casting -------------------🡪

byte short int long

🡨----------------------Explicit type casting -----------------------

Int family type casting

class TypeCasting {

public static void main(String args[]) {

byte a =10;

short b=a; //Implicit type casting

System.out.println(a);

System.out.println(b);

short c = 10;

// (type)variableName;

byte d = (byte)c; //Explicit type casting

System.out.println(c);

System.out.println(d);

}

}

**Int and float type casting**

----------------🡪 Implicit ---------------🡪

int float

🡨-------------Explicit ----------------------

By default every decimal number in java double consider. So double data type size is 8 types. Which we can’t store in float ie 4 byte.

class TypeCasting1 {

public static void main(String args[]) {

int a=10;

float b=a; //Implicit type casting

System.out.println(a);

System.out.println(b);

//float c = 10.10f; //Explicit type casting

float c = (float)10.10;

int d = (int)c;

System.out.println(c);

System.out.println(d);

}

}

**Operator**

1. Arithmetic Operator : +, -, \*, /, %(find the remainder)
2. Relational Operator : >, >=, <, <=, ==, !=
3. Logical operator : &&, ||, !
4. Assignment operator : =
5. Increment and decrement : ++ it increment the value by 1 and – it decrement the value by 1
6. **instanceOf : it is use to check the reference type.**

If statement

1. simple if

if(condition) {

if true block;

}

1. if else

if(condition) {

if true block

}else {

if false block

}

1. nested if : if within another if

if(condition) {

if(condition) {

**true block**

}else {

**false block**

}

}else {

if(condition) {

**true block**

}else {

**false block**

}

}

1. if else if or if ladder

if(condition1) {

Block1

}else if(codition2) {

Block2

}else if(codition3) {

Block3

}else {

Else block

}

1. switch statement : in switch statement user can decide which block you want to execute.

Syntax

switch(variableName) {

case v1:block1;

break;

case v2:block2;

break;

case v3:block3;

break;

case v4:block4;

break;

default:defaultblock

break;

}

switch, default, case and break are keywords.

varibleName must be type of int or char or String.

class SwitchStatement {

public static void main(String args[]) {

int n =4;

switch(n) {

case 1:System.out.println("1st block");

break;

case 2:System.out.println("2nd block");

break;

case 3:System.out.println("3rd block");

break;

default:System.out.println("Wrong choice");

break;

}

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

}

}

**Looping :** It is use to iterate the records one by one

Initialization : start position and end position

Conditions

Do the task

Increment or decrement

While loop

Do while loop

For loop

For each loop or enhanced loop

**class Loop {**

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

**/\*System.out.println("While loop - Entry loop ");**

**int i=1;**

**int n=10;**

**while(i<=n) {**

**System.out.println(i);**

**i++;**

**}**

**\*/**

**System.out.println("Do while - Exit Loop");**

**int i=1;**

**int n=10;**

**do {**

**System.out.println(i);**

**i++;**

**}while(i>=n);**

**}**

**}**

**Day 2 : 12-01-2020**

**For loop**

**Syntax**

**1 2 4**

**for(initialization;condition;increment/decrement) {**

**for body; 3**

**}**

For loop Example

class ForLoop {

public static void main(String args[]) {

System.out.println("For loop example");

for(int i=0;i<=10;i++) {

System.out.println(i);

}

System.out.println("Finish");

}

}

**Non primitive date type or reference data types** : It is use to store value as well as reference of another data types.

4 types

1. array
2. class (pre-defined class or user-defined class)
3. interface (pre-defined interface or user-defined interface)
4. enum (per-defined or user-defined enum)

**array:** array is user-defined data or also known as reference data type which is use to store collection of elements or value of same types.

int a;

a=10;

a=20;

syntax

datatype arrayName[];

int abc[]; // array declaration

int xyz[]={10,20,30,40,50,60};

System.out.println(xyz[0]);

**For each loop or enhanced loop**

Syntax

for(datatype variableName : arrayName) {

}

**For loop**

class ArrayDemo {

public static void main(String args[]) {

int abc[];

int xyz[]={10,20,30,40,50,60,100,130,150,200,250};

System.out.println(xyz[0]);

System.out.println(xyz[5]);

System.out.println("Size of array "+xyz.length);

System.out.println("Using For loop");

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

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

}

System.out.println("Using for each loop");

for(int n : xyz) {

System.out.println(n);

}

}

}

Another For each example

class ArrayDemo {

public static void main(String args[]) {

int abc[];

int xyz[]={10,20,30,40,50,60,100,130,150,200,250};

System.out.println(xyz[0]);

System.out.println(xyz[5]);

System.out.println("Size of array "+xyz.length);

System.out.println("Using For loop");

for(int i=2;i<8;i=i+2) {

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

}

System.out.println("Using for each loop");

for(int n : xyz) {

System.out.println(n);

}

}

}

**Creating the memory size for the array**

Syntax

datatype arrayName[]=new datatype[size];

int abc[]=new int[10];

Example

class ArrayDemo {

public static void main(String args[]) {

int abc[]=new int[10];

System.out.println("Size of array is "+abc.length);

abc[0]=100;

abc[1]=200;

int flag = 123;

System.out.println(abc[0]);

System.out.println(abc[1]);

System.out.println(abc[2]);

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

abc[i]=flag;

flag = flag +10;

}

System.out.println("All elements are ");

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

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

}

}

}

Another Example

class ArrayDemo {

public static void main(String args[]) {

int a=4; // 4 bytes

int abc[]=new int[5];

abc[0]=100;

}

}

int abc[]; one dimensional array

int abc[][]={{10,20},{30,40},{50,60}};

Two dimensional array

class ArrayDemo {

public static void main(String args[]) {

int abc[][]={{10,20},{30,40},{50,60}};

System.out.println("Display values");

/\*System.out.println(abc[0][0]);

System.out.println(abc[0][1]);

System.out.println(abc[1][0]);

System.out.println(abc[1][1]);

System.out.println(abc[2][0]);

System.out.println(abc[2][1]);\*/

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

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

System.out.print(abc[i][j]);

System.out.print(" ");

}

System.out.println();

}

}

}

Creating the memory size for two dimensional array

int abc[][]=new int[5][3];

**OOPs**

Object Oriented Programming

Procedure language

Limitation

1. They are not working on real time scenario.
2. In procedure language they given important to function rather than variable.
3. Variable are not secure then can move from one function to another function freely. (public or private keywords are not there.)
4. We can’t reusability.

**object :** any real world entity.

**State or properties have age,name,color,h,w etc**

Person

**Behaviour do/does walking, talking, sleeping,**

**teaching**

Laptop

Pen

Animal

State or properties wheel, color, price etc

Car

Behaviour start, appliedGear, moving, stop

Bank

Etc

**object is concept.**

**class :**

**class is known as blue print of the object.**

**class is also known a template of object.**

**class is a user-defined data types which help to implement or create the object.**

**array : array is use to store same type of value.**

**C/C++ structure : structure is user-defined data type which help to store different type of values.**

**class : class is user defined data type which help to store different types of value. Class can hold set function or methods.**

class Car {

int wheel;

String color;

float price;

void start() {

System.out.println("Car Start");

}

void appliedGear() {

System.out.println("Applied the Gear....");

}

void moving() {

System.out.println("Car is moving....");

}

void stop() {

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

}

}

class Test {

public static void main(String args[]) {

System.out.println("main method");

Car innova = new Car(); // heap memory

innova.start();

innova.appliedGear();

innova.moving();

innova.stop();

}

}

**Type of variable or fields**

3 types

1. Instance variable
   1. The variable which declare inside a class but outside a method including main method is known as instance variable.
   2. Instance variable hold default value according to their data types. Example int family 0, float family 0.0, char space, Boolean false, String null.
   3. Instance variable we can use directly inside all methods but the methods must be part of same class and it must be non static method.
2. Local variable
   1. The variable which declare inside a method including main method is known as local variable.
   2. Local variable doesn’t hold default value we have to initialize.
   3. The scope of local variable within that block where it declare.
3. Static variable

class Car {

int wheel;

String color;

float price;

void displayInfo() {

String temp="Car Info"; // local variable

System.out.println("Wheel "+wheel);

System.out.println("Color "+color);

System.out.println("Price "+price);

System.out.println("Temp "+temp);

}

}

class Test {

public static void main(String args[]) {

System.out.println("main method");

Car innova = new Car(); // heap memory

//innova.displayInfo();

innova.wheel = 4;

innova.color = "Gray";

innova.price = 1400000;

innova.displayInfo();

Car santro = new Car(); //heap memory

//santro.displayInfo();

santro.wheel = 4;

santro.color ="Black";

santro.price = 850000;

santro.displayInfo();

}

}

**Constructor :** It is a type of special method which help to create the object.

Pts

1. Constructor have same name as class itself.
2. Don’t provide return type for constructor not even void also.
3. Constructor no need to call explicitly. It will call automatically when we create the object.

In the life of object if we want to execute any task only one time that type of task we have to write inside a constructor.

In the life of object if we can to execute any task more than one time that type of task we have to write inside methods.

Constructor Example

class Employee {

int id;

String name;

float salary;

Employee() {

id =123;

name="Unknown";

salary = 8000;

}

Employee(int id, String name, float salary) {

this.id = id;

this.name = name;

this.salary = salary;

}

void setValue(int id, String name, float salary) {

this.id = id;

this.name = name;

this.salary = salary;

}

void display() {

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

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

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

}

}

class Test {

public static void main(String args[]) {

Employee emp1 = new Employee(); emp1.display();

Employee emp2 = new Employee(); emp2.display();

Employee emp3 = new Employee(1,"Ravi",12000); emp3.display();

Employee emp4 = new Employee();

emp4.setValue(2,"Ramesh",14000);

emp4.setValue(3,"Rajesh",16000);

emp4.display();

}

}

Object and class

4 pillar

**Encapsulation :** Binding or wrapping data (variables/fields) and code (functions/methods) in a single unit is know as Encapsulation.

class :

class Employee {

int id;

String name;

float salary;

void display() {

}

}

class Employee {

private int id;

private String name;

private float salary;

void setValue(int id, String name, float salary) { //helper methods

this.id = id;

this.name = name;

if(salary <0 ) {

this.salary = 8000;

}else {

this.salary = salary;

}

}

void display() {

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

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

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

}

}

class Test {

public static void main(String args[]) {

Employee emp1 = new Employee();

//emp1.id=100;

//emp1.name ="Ravi";

//emp1.salary = -12000;

emp1.setValue(100,"Ravi",12000);

emp1.display();

}

}

In Java all instance variable must be private and provide helper method to set the value with term and conditions.

**Inheritance :** Inheritance is use to acquire or inherits the properties or state and behaviour of old class to new class.

The gold of inheritance is use to do re-usability.

class OldClass { super class, base class or parent class

state

behaviour

}

class NewClass extends OldClass{ sub class, derived class or child class

state

behaviour

}

Simple Example of Inheritance

class A {

void dis1() {

System.out.println("dis1() method");

}

}

class B extends A{

void dis2() {

System.out.println("dis2() method");

}

}

class Test {

public static void main(String args[]) {

A obj1 = new A();

obj1.dis1();

B obj2 = new B();

obj2.dis2();

obj2.dis1();

}

}

**Types of Inheritance**

1. Single Inheritance : One super class and one sub class

class A { }

class B extends A { }

1. Multilevel Inheritance : One super class and n number of sub class connected one by one

class A {}

class B extends A{}

class C extends B {}

class D extends C {}

1. Hierarchical Inheritance : One super class and n number of sub class directly connected to super class

class A {}

class B extends A {}

class C extends A{}

class D extends A {}

1. Multiple inheritance : More than one super class and one sub class

class A {}

class B {}

class C extends A, B {} : But Java doesn’t support this type of inheritance using classes. It can support indirectly using interface.

**Oops relationship**

1. Manager Is a relationship Employee
2. Employee/Manager/Programmer Has a relationship Address

class Employee { Generics

id,name,salary primitive property

}

class Manager extends Employee{ Specific

numberOfEmp

Address add = new Address();complex property

}

class Programmer extends Employee {

projectName;

}

class Address {

city, state, pincode

}

**Day 3 : 13-01-2020**

Employee Management System

Is a relationship

Employee, Manager, Programmer, Developer, Clerk, CEO etc

Banking Application

Has relationship

Customer, Bank, Account, Transaction etc.

Bank

Customer

custId

custName

age

Account

Accno

Amount

typeofAccount

Transcation

Tid

Date

Time

typeOfTran

Online Shopping

Customer, Product, Clerk, Admin etc

Single Inheritance Example

class Employee {

private int id;

private String name;

private float salary;

void setValue(int id, String name, float salary){

this.id = id;

this.name = name;

this.salary = salary;

}

void display() {

System.out.println("Id is "+id);

System.out.println("Name is "+name);

System.out.println("Salary is "+salary);

}

}

class Manager extends Employee{

private int numberOfEmp;

void setMgrValue(int numberOfEmp){

this.numberOfEmp = numberOfEmp;

}

void displayMgr() {

System.out.println("Number of employee "+numberOfEmp);

}

}

class InheritanceDemo {

public static void main(String args[]) {

Employee emp1 = new Employee();

emp1.setValue(1,"Ravi",12000);

emp1.display();

Manager mgr1 = new Manager();

mgr1.setValue(2,"Ramesh",45000);

mgr1.setMgrValue(10);

mgr1.display();

mgr1.displayMgr();

}

}

Hierarchical Inheritance example

class Bike {

void speed() {

System.out.println("60km/hr");

}

}

class Pulsar extends Bike {

void color() {

System.out.println("Black");

}

}

class Honda extends Bike {

void color() {

System.out.println("Gray");

}

}

class InheritanceDemo {

public static void main(String args[]) {

Pulsar pu = new Pulsar();

pu.speed(); pu.color();

Honda hh = new Honda();

hh.speed(); hh.color();

}

}

**Polymorphism**

One name many forms or one concept many implementation.

2 types

Compile time polymorphism Run time polymorphism

javac java

compile interpreter

static binding dynamic binding

early binding late binding

Method Overloading Method Overriding

In C++ In C++

Function Overloading Function Overriding

Operator overloading

Java doesn’t support

Operator overloading

Compile time

Polymorphism

Method Overloading : The method have same name but different parameter list ( type of parameter list(data type) or number or parameter list must different). But return type must be same is known as Method Overloading.

**Method Overloading Example**

class Cal {

void add(int a, int b, int c) {

System.out.println(a+b+c);

}

void add(float a, float b) {

System.out.println(a+b);

}

void add(String s1, String s2) {

System.out.println(s1+s2);

}

}

class MethodOverloading {

public static void main(String args[]) {

Cal cc = new Cal();

cc.add(1,2,3);

cc.add(10,20);

cc.add("10","20");

}

}

customerDetails(String fname,String lname,int age) {

}

customerDetails(String fname,int age) {

}

Run time polymorphism

Method Overriding : The method have same name and same method signature(number of parameter, type of parameter and return type must be same).

To achieve the method overriding class must be in Inheritance.

(Minimum super class and sub class).

**Method Overriding Example**

class Bank {

void deposite() {

System.out.println("Generic Deposite Method Logic");

}

}

class Hdfc extends Bank {

void checkBalance() {

System.out.println("HDFC checkbalance method");

}

void deposite() {

System.out.println("Deposite Method Logic Provided by Hdfc");

}

}

class Sbi extends Bank {

void checkBalance() {

System.out.println("Sbi checkbalance method ");

}

}

class MethodOverriding {

public static void main(String args[]) {

Hdfc obj1 = new Hdfc();

obj1.deposite(); obj1.checkBalance();

Sbi obj2 = new Sbi();

obj2.deposite(); obj2.checkBalance();

}

}

Open terminal

**vim Demo.java**

**abstract**

**final**

**static**

abstract :

1. abstract is a keyword which we can use with method and class but not with variable.
2. Abstract method : The method without body or without curly braces or incomplete method is known as abstract method.

abstract returnType methodName(parameterList);

1. abstract class : if class contains one or more abstract method then we have to declare the class as abstract class.

abstract class className {

}

1. If any normal class extends abstract class that class must be provide the body for the all abstract methods belong to that class. That class can ignore if that class itself is a abstract class.
2. Abstract class we can’t create the object.
3. Abstract class can contains normal as well as abstract methods.

Means abstract class can contains zero or 1 or all abstract methods.

If we can to achieve partial abstraction

Hiding the internal implementation without knowing background details.

abstract class Orders {

abstract void payment();

}

class Paytm extends Orders {

void payment() {

System.out.println("Payment happend through PayTM");

}

}

class GooglePay extends Orders{

void payment() {

System.out.println("Payment happend through Google Pay");

}

}

class PhonePay extends Orders{

void payment() {

System.out.println("Payment happend through Phone Pay");

}

}

class MethodOverriding {

public static void main(String args[]) {

}

}

**final**

final keyword we can use with variable, method and class.

1. Final variable : if we want to declare the constant variable in java we have to use final keywords with variable.

final int A=10;

A=20; Error

1. Final method : if method is final we can’t override that methods but we can use that method.
2. Final class : if class is final we can’t extends that class or inherits that class.

static

static keyword we can use with variable and method but not with class

(if class is inner class then we can use static keyword but not for outer class).

1. Static method : if method is static we can call that method with the help of className

className.staticMethodName();

1. Static variable: if variable is a static we can assign the value for that variable with the help className

className.staticVariablename = value;

1. We can call static method through object also.
2. We can assign the value for static variable through object also.
3. Inside a static method we can access only static variable directly but inside non static method we can access both the type of variable ie static as well as non static.

Static Example

class Abc {

int a;

static int b;

void dis1() {

System.out.println("Non Static method");

System.out.println("a "+a);

System.out.println("b "+b);

}

static void dis2() {

System.out.println("Static method");

//System.out.println("a "+a);

System.out.println("b "+b);

}

}

class StaticDemo {

public static void main(String args[]) {

Abc.dis2();

Abc.b=10;

Abc obj1 = new Abc();

obj1.dis2();

obj1.b=20;

obj1.a=30;

obj1.dis1();

}

}

Every class contains only one static memory

Every class contains n number heap number how many object you created.

Static is like a global to all objects.

Employee

Id,name,salary -🡪 instance

MeetingTime 🡪 Static

Customer

CustId

CustName

Age

Interest 7.0 6.8 7.2 static

IFSC Code

class Abc {

final static String city=”Bangalore”;

int a;

static int b;

void dis1() {

System.out.println("a "+a);

System.out.println("b "+b);

}

}

class StaticDemo {

public static void main(String args[]) {

Abc obj1 = new Abc();

Abc obj2 = new Abc();

obj1.a=10;

obj1.b=20;

Abc.b=30;

obj2.a=40;

obj2.b=50;

Abc.b=60;

obj1.dis1(); // a = 10 b=60

obj2.dis1(); //a =40 b =60

}

}

int a=10;

void abc() {

a=20;

}

void xyz() {

printf(“%d”,a);

}

void main() {

printf(“%d”,a);

abc();

printf(“%d”,a);

a=40;

xyz();

}

**Day 4 : 14-01-2020**

**Interface : Interface is a type of reference data type which also know as 100% pure abstract class.**

**Syntax**

**interface interfaceName {**

**fields;**

**methods;**

**}**

By default all fields in interface are public static and final.

By default all methods are public and abstract.

interface Abc {

public static final int a=10;

static final int b=20;

final int c =30;

int d=40;

public abstract void dis1();

abstract void dis2();

void dis3();

}

interface Abc {

int a=10;

void dis1();

}

interface Mno {

int b=20;

void dis2();

}

interface Xyz extends Abc,Mno{

int c=30;

void dis3();

}

class Demo implements Abc,Mno {

}

Like a class one interface can extends another interface but interface can extends more than one interface but class can extends only one class.

In this example we can say using interface we can achieve multiple inheritance.

Class always implements interface. It can implements more than interface.

Which ever class implements any interface (one or more ) that class must be provide the body for all methods belong to that interface.

Access Specifiers

private

protected

default (nothing)

public

method overriding with access specifiers.

Super (class / interface ) Sub

public public

protected public

protected

default public

protected

default

private we can’t override

Interface Example

interface Abc {

int a=10;

void dis1();

}

interface Mno {

int b=20;

void dis2();

}

interface Xyz extends Abc,Mno{

int c=30;

void dis3();

}

class Demo implements Abc,Mno {

public void dis1() {

System.out.println("Abc interface method");

}

public void dis2() {

System.out.println("Mno interface method");

}

}

class InterfaceDemo {

public static void main(String args[]) {

Demo obj = new Demo();

obj.dis1();

obj.dis2();

}

}

Abstract class

1. Abstract class can contains normal as well as final variable.
2. Abstract class can contains normal as well as abstract methods ie zero abstract methods or 1 or all.
3. Normal class or abstract class can extends only one abstract class.
4. Abstract class can contains default constructor as well as we can write parameterized constructor.
5. Using abstract class we can achieve partial abstraction.

Interface

1. Interface contains only final variable.
2. Interface contains only abstract methods.
3. Normal class or abstract class can implements more than one interface.
4. Interface doesn’t contains constructor.
5. Using interface we can achieve fully abstraction.

abstract class Rbi {

abstract void tax();

void gst() {

System.out.println(“gst implements”);

}

}

interface Bank {

void withdraw()

void deposite()

void checkBalance(int accNumber)

void transaction(int accNumber, Date date);

}

class Hdfc extends Rbi implements Bank {

}

class Hsbc extends Rbi implements Bank {

}

**OOPs using Java**

packages

package is a collection of classes and interfaces. Package is like a directory or folder.

2 types

1. User-define package
2. Pre-defined or built in package.

User – defined package

Education

School College Graduation

Attendance Attendance Attendance

When we create more than one class or interface which have same name but different purpose.

Education folder

School College sub folder

Attendance.java Attendace.java

Syntax

package packageName; (packageName must be lower case)

**IDE : Integrated Development Environment**

Netbean

Eclipse (Different types )

MyEclipse

RAD

Etc

**Access Specifiers :** Access specifiers is use to provide the visibility or accessibility of variable, methods or class.

4 types

1. private : We can use private access specifiers with instance variable, static variable, non static method, static method, constructor but not with class and local variable.

Scope : Within a same class. Out side class we can’t access.

1. protected : We can use protected access specifiers with instance variable, static variable, non static method, static method, constructor but not with class and local variable.

Scope : within same package other package if it is sub class.

1. default (nothing) :We can use default (nothing) with all.

Scope : within same package. Other package can’t access.

1. public : We can use private access specifiers with instance variable, static variable, non static method, static method, constructor and class but not local variable.

Scope : within same package as well other package.

Pre – defined package.

java javax 🡪 root package

extensible

lang (language) swing

io (input output) rmi

util (utility ) net

sql (structure query language) servlet

awt(abstract window toolki) ejb

net (networking ) jms

rmi (remote method invocation)

etc etc

all pre-defined package contains set of classes, interfaces as well as other package. Other sub package again contains set of classes, interfaces and sub package.

Lang packages

Exception Handling

Multi threading

Io package file handling

Collection Framework (util package)

Data Structure

Regular Expression

Lang package

By default every java program import java package.

By default every java program extends Object class.

Without importing lang package we can use all classes and interface belong to lang package.

String

StringBuffer

Math

Object

Exception

Type of exception classes

Thread

Runnable interface

Etc

In Java String is a pre-defined class or reference data types.

Syntax

String str = “Welcome to Java Training “ Creating object literal style.

String str = new String(“Welcome to Java Training”); using new keyword.

== : it check value as well as reference code.

.equals() : it check only value it may be same memory or different memory.

**String Example**

**package** com;

**public** **class** StringDemo {

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

/\*String str1 = "Welcome to Java Training";

String str2 = new String("Welcome to Java Training");

System.out.println(str1);

System.out.println(str2);

System.out.println(str1.length());

System.out.println(str1.toLowerCase());

System.out.println(str1.toUpperCase());

System.out.println(str1.substring(5));

System.out.println(str2.substring(8, 15));//start count 0, end count 1 or size -1

System.out.println(str1.contains("Java"));

System.out.println(str1.contains("Python"));

System.out.println(str1.replace("Java", "Python"));\*/

String name1 = "Raj"; //New memory created...

String name2 = "Raj"; // refer to same memory

String name3 = **new** String("Raj"); //new memory

String name4 = **new** String("Raj"); //new memory

**if**(name1.equals(name4)) {

System.***out***.println("Equal");

}**else** {

System.***out***.println("Not equal");

}

}

}

String is known as immutable class. Immutable means can’t change.

StringBuffer : It is a type of string class which is also known as mutable string class.

Math : it class is use to do maths operations

All methods in maths is static

**Math Class Example**

**package** com;

**public** **class** MathDemo {

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

System.***out***.println(Math.***PI***);

System.***out***.println(Math.***E***);

System.***out***.println(Math.*sqrt*(9));

System.***out***.println(Math.*max*(10, 4));

}

}

By default every class in Java extends Object class.

**equals():** This method is a part of object class which help to compare to object reference value. Internally string class extends Object class and override equals methods to compare two string reference.

**toString() :** This method is a part of object class it will call automatically when we display user-defined class reference in println.

**finalize() :** This method is part of object class which internally call automatically before garbage collector(JVM).

**wait(), notify(), notifyAll() :** Method part of object use in Multi threading.

**Exception Handling**

Exception is object or memory which occurs when unexpected or abnormal conditions occurs during the execution of a program.

Java program

Compile time error Run time error

javac

Syntax error Run time

Class opening closing

Missing, statement

Semicolon missing,

Double quote missing etc

Or

Typing error

Run time error

Error Exception

Error and Exception both are pre-defined classes part of language.

**Error** : it is a type of error which generate at the run time which we can’t handle it. Example JVM crash, software or hardware issue.

**Exception :** It is a type of error which generate at the run time which we can handle it. Example Divided by Zero.

Throwable

**Exception**

Checked Exception Unchecked Exception

**RuntimeException**

SQLException ArithmeticException

IOException NumberFormatException

FileNotFoundException ArrayIndexOutOfBoundsException

NullPointerException

To handle both the type of exception

Five keywords

1. try
2. catch
3. finally
4. throw
5. throws

syntax

try {

}catch(Exception e) {

}

**Try block** : The code which makes problem one line code or multiple line code we have to keep in try block.

**Catch block** : This block will execute only if any exception generate. No exception no catch block.

**Finally** : This block it will execute 100% sure if any exception generate or not.

**equals()** : how to check user-defined object values using equals() and hashCode() methods.

**Day 5 : 15-01-2020**

**try**

**catch**

**finally**

**throw**

**throws**

**throw :** throws keyword is use to generate or raise pre-defined or user-defined exception depending upon the condition.

Syntax

throw **new Exception();**

**or**

throw **new ExceptionSubClass();**

**throw keyword example**

**package** com;

**class** AccountMinMaintain **extends** Exception {

**public** AccountMinMaintain() {

}

**public** AccountMinMaintain(String msg) {

**super**(msg); // call super class parameterized constructor

}

}

**public** **class** ThrowDemo {

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

**int** accno=123;

String name = "Ravi";

**float** amount = 700;

**float** balance = amount -250;

**try** {

//int res = 10/0;

**if**(balance<500) {

//throw new Exception();

//throw new ArithmeticException();

//throw new ArithmeticException("Maintain min 500 amount in account");

//throw new AccountMinMaintain();

**throw** **new** AccountMinMaintain("Maintaine min 500 amount in account");

}**else** {

System.***out***.println("Your balance amount is "+balance);

}

}**catch**(Exception e) {

//System.out.println("I Take Care!");

System.***out***.println(e.toString());

}

System.***out***.println("finish");

}

}

**throws :** throws keyword is use to throw the exception to caller methods.

**Syntax**

**returnType methdoName(parameterList) throws Exception, ExeptionSubClass {**

**}**

**void display() throws Exception {**

**}**

Throws Exception example

**package** com;

**public** **class** ThrowsException {

**static** **void** dis1() **throws** Exception{

//try {

**int** res = 10/0;

//}catch(Exception e) {}

System.***out***.println("Dis1 method");

}

**static** **void** dis2() **throws** Exception{

//try {

*dis1*();

//}catch(Exception e) {}

System.***out***.println("Dis2 method");

}

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

//try {

*dis2*();

//}catch(Exception e) {}

System.***out***.println("Main Method");

}

}

**Checked Exception**

UncheckedException

ArithmeticException

int res = 10/1;

int res = a/b;

ArrayIndexOutOfBoundsException

int abc[]={10,20,30,40}

abc[index];

Unchecked exception always check at the run time.

Checked Exception check twice at compile time as well as run time.

Checked Exception we can’t ignore we have to handle mandatory using try-catch block or throws.

CheckedException Example

**package** com;

**public** **class** CheckedExceptionDemo {

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

System.***out***.println("Hi");

**try** {

Thread.*sleep*(3000);

}**catch**(Exception e) {

}

System.***out***.println("Hello");

}

}

**equals() and hashCode() : both are pre-defined method belong to Object class.**

**String class internally extends Object class and override equals and hasCode() method depend upon the their requirement.**

**JavaBean :**

1. JavaBean must be public.
2. All variable must be private.
3. For every variable we have to provide setter and getter method.
4. The setter method name start with prefix set followed by variableName.
5. The getter method name start with prefix get followed by variable name.

**Normal Class :**

1. Normal class may be public or default.
2. All instance variable in normal class may be private or default.
3. If all instance variable are private then we have to provide helper method to set the value. The method name may be anything.

public class Employee {

private int id;

private String name;

public void setId(int id) {

condition.

this.id = id;

}

public int getId() {

return this.id

}

}

This class is known as Pure Encapsulation class.

100 objects

5 ---🡪 have same id, name, salary

100,Ravi,12000

100

10-🡪 have same id, same name, salary

101,Ramesh,14000

100

85--🡪 unique

85 + 1+ 1+

**Multithreading**

By default every java program depending upon the thread.

**Program :** Set of instruction to perform a specific task.

t time. 1 min, 2 min, 5 min, 10 min

**Processor** : Processor is responsible to execute the code.

**Process** : Program in execution or time taken to execute the code.

Process is heavy weighted component. Means it take more resource or more memory of your machine.

**Thread** : It is small execution of a code within a process. Thread also known as light weighted process. Means it takes less resource or less memory of your machine.

Within a process one or more thread will execute.

C and C++ by default they are process base programming language.

By default Java is thread base programming language.

To check default thread details In Java

Thread is a pre-defined class part of lang and this class contains currentThread() pre-defined static method. currentThread method return type is Thread class reference.

Thread t = Thread.currentThread();

When we display t

Thread[main,5main];

Main 🡪 name of thread

5-🡪 priority of thread ­

Main 🡪 group of thread

t.setName(“My thread”);

t.setPriority( ) //range min 1 max 10 norm 5

Thread.MAX\_PRIORITY 🡪10

Thread.MIN\_PRIORITY 🡪1

Thread.NORM\_PRIORITY🡪5

Multitasking :

We want to do more than one task at time.

We can achieve multi tasking using 2 ways

1. Process base
2. Thread base

Processor 2 min (8 min ideal

T1 t2 t3

Task1 Task2 Task3

10 min 10 min 10 min

C --🡪D set files

E -🡪D set files

F-🡪 G set files

Printing paper

1 print

2 print

3 print

In java we can create user defined Thread using 2 ways

1. Extends Thread
2. Implements Runnable interface

Extends Thread class

1. Create user defined class extends Thread class.
2. Create the Thread class reference.
3. Using thread class reference call start() method. start() is a pre-defined method which is use to ready to run (Runnable State).
4. Start() method internally call run() method. run() method is part of Thread class.
5. If we want any custom data then we have to override run() method and write the code.

Implements Runnable interface

1. Create user defined class and implements Runnable interface.
2. Runnable interface contains one method ie run(). When class implements Runnable interface we have to override run() method mandatory.
3. Then we have to create user defined class object and pass that object in Thread class constructor.
4. Then using Thread class constructor we have to call start() method.

Difference between extends Thread class and Implements Runnable interface.

When class extends Thread class that class can’t extends any other class for different purpose as well as that class become heavy class because all method of Thread class inherited.

When class implements Runnable interface that class can extends only other class for different purpose as well as can implement more than one interface for different purpose. That class become height weighted because Runnable interface contains only one method ie run() methods.

**Day 6 : 18-01-2020**

Life cycle of Thread

isAlive

join

sleep

wait()

notify()

notifyAll()

Create the

Thread --------🡪 Runnable -----------🡪Running ------------🡪Destroy

Instance

obj1 obj.start()

obj2

run i=10,j=10

t1 t1.start()

t2

**join :** This method is use to join child thread to parent thread. So parent thread wait till child thread get destroy.

Creating more than one thread in one class with same memory.

Wait()

Notify()

notifyAll()

wait() : This method is use to make the thread to wait() (pause) depending upon the condition. This method is use to suspend the thread.

notify(): This method is use to call back waited thread or resume waited thread.

It can resume any one thread.

notifyAll() : resume more than one thread. All waited thread.

These three methods belong to Object class.

Created more than one thread

1. Creating more than one thread in same memory.
2. The methods must be in synchronized.

Synchronization : It is a concept which help to block or lock the thread.

It allow use all resources for only one thread at time.

To achieve synchronization java provide

synchronized

This keyword we can apply for method or within a method we can create more then one synchronized block.

**Consumer and producer example**

**IO Package**

**Input Output**

In Java we can do Input and Output operation using Stream.

Stream : Flow of data or it is a abstraction between source and destination.

Source : Keyboard, File, Network, Database etc

Destination : Monitor(Console), File, Network, Database etc

Taking the value through keyboards and display on console.

1. DataInputStream
2. BufferedReader
3. Scanner class
4. Command line interface.

Stream

byte(1 byte) char (2 byte)

Input Output Input Output

Abstract classes.

InputStream OutputStream Reader Writer

DataInputStream DataOutputStream InputStreamReader OutputStreamWriter

FileInputStream FileOutputStream FileReader FileWriter

BufferedInputStream, BufferedOutputStream, BufferedReader,BufferedWriter

ObjectInputStream,ObjectOutputStream

PrintStream PrintWriter

Standard Input Device : Keywords

Standard output device : Monitor or console.

System.out.println(“”);

System is a pre-defined class part of lang package.

in, out, err : three static fields

javap java.packageName.classname/interface

in is InputStream class reference

out and err is PrintStream class reference

System.out if is behave like a PrintStream class reference

PrintStream ps = System.out;

ps.println(“Welcome to Java.”);

InputStream is = System.in

PrintStream reference always refer to Standard output device. Monitor

InputStream reference always refer to Standard input device. Keywords

Byte classes

Source 🡪 Keyboard DataInputStream

Destination 🡪 Consoler / Monitor 🡪 PrintStream

If we want to take primitive value through DataInputStream class we were depending upon Wrapper classes.

8 primitive types Wrapper

byte Byte

short Short

int Integer

long Long

float Float

double Double

char Character

boolean Boolean

These wrapper classes help us to convert string to primitive data types as well

as primitive to objects.

Byte wise

Source : Keywords

Destination : File :

**Day 7 : 19-01-2020**

Buffer : Buffer means temporary memory.

Using Buffer memory when we connect input as well as output operation it improve more than 1000 performance.

Store Primitive Values like id,name,salary

And Read Primitive value like id,name,salary

DataInputStream, DataOutputStream, FileInputStream and FileOutputStream etc

Object Oriented programming Language.

Id, Name, Salary, Age etc

**Object Serialization :**

Storing the object itself in external file or converting object into byte or stream format is known as Object Serialization.

**Use :**

If we want to pass the object in networking environment then object must be serialization format.

Which class object you want to do serialization that class must be implements Serializable interface.

Serializable interface is a part of io package.

This is interface is known as marker interface.

The interface which contains zero methods that interface is known as marker interface.

Serializable is a one type of marker interface.

This interface help or support to do serialization.

**Object De-Serialization :** Converting stream format data into object format is know as De-Serialization.

do {

1: Display all files in Ascending order

2. Sub Option

A Creating new file for user with some info

B Delete the file

C Search the file

D exit from sub option

3 Exit the application

switch() {

}

}

Data Structure

Collection Framework

Regular Expression

Introduction Maven tool

We want to check the properties of a file

It is file or directory.

Read mode or write mode

Size of data in file

Name of the file

Etc

File : This class is use to check the property of a file.

**Day 8 : 20-01-2020**

**Collection Framework – Database Structure**

Data Structure : It is a combination of two words data and structure.

Storing the data in proper structure format.

Linear format : array, linkedlist, vector etc

One by one : Queue and Stack

Hierarchical format : Tree

Node connected one by one :Graphs :

int a;

int abc[];

class Employee {

int id;

String name;

float salary;

}

Employee emp = new Employee();

Array object

**Employee employees[]=new Employee[10];**

employees[0]=new Employee();

employees[0].id=100;

employees[1]=new Employee();

employees[2]=new Employee();

Collection Framework – Data Structure

It contains set of classes and interfaces which help to store collection of object or elements of any types (any primitive (wrapper classes) as well user-defined class objects). It provide set of methods with help of those methods we can add, remove, search, iterate elements or object very easily.

**Util package**

Collection 🡪 interface

Extends extends extends doesn’t extends

List Set Queue Map

List : it store more than one elements. List allow duplicate. It maintain the order.

Set : It store more than one elements. It doesn’t allow duplicate. In Set some API order, unoder and sorted.

Queue : First In First Out

Map : It allow to store information in the form of key-value pairs. Where key is unique and value may be duplicate.

**List :** ArrayList, LinkedList, Vector and Stack are type of List classes.

**Set :** HashSet, LinkedHashSet, TreeSet are the implementation of Set Interface.

They are type of Set classes.

**Queue** : PriorityQueue internally implements Queue interface.

**Map :** HashMap, LinkedHashMap, TreeMap and Hashtable.

**ArrayList :**

Normal array can store only same type of values. But ArrayList can store any type of values.

Normal array fixed memory size. ArrayList dynamical memory creation. We can add or remove any elements from ArrayList easily.

**ArrayList Example**

**package** com;

**import** java.util.ArrayList;

**public** **class** ArrayListDemo {

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

ArrayList al = **new** ArrayList();

System.***out***.println("Size is "+al.size());

System.***out***.println("Empty "+al.isEmpty());

al.add(100);

al.add(10.20);

al.add("Ramesh");

al.add(30);

System.***out***.println("Size is "+al.size());

System.***out***.println("Empty "+al.isEmpty());

}

}

**ArrayList Example**

**package** com;

**import** java.util.ArrayList;

**public** **class** ArrayListDemo1 {

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

ArrayList al = **new** ArrayList();

al.add(10);

al.add(20);

al.add(30);

al.add(40);

al.add(50);

System.***out***.println(al);

System.***out***.println("Value in Zero Position "+al.get(0));

System.***out***.println("Value in 1 Position "+al.get(1));

al.add(1, 100);

System.***out***.println("Value in Zero Position "+al.get(0));

System.***out***.println("Value in 1 Position "+al.get(1));

al.remove(2); // remove the value using index.

al.remove(**new** ~~Integer~~(40));

System.***out***.println(al);

}

}

**LinkedList :** LinkedList is a type of List class as well as Queue type.

Internally LinkedList class implements List and Deque interface. Deque interface internally extends Queue interface.

Ref Value

Node

Single Linked list

Double Linked List

Circular Linked List

In Java By Default LinkedList class is a type of double Linked list class.

In ArrayList if we adding or removing more elements again and again which effect the performance.

In LinkedList we can add or remove elements with good performance.

**Vector :** It is known as legacy class. By default all methods in Vector class are synchronized.

When method are synchronized slow in performance but work is safe.

**Stack :** It is a type of List API. First In Last Out.

Push() : Add elements from top

Pop() : Retrieve elements from stack and remove

Search() : if elements is present return index position of that elements else return -1

Peek() : only retrieve top most elements.

**Stack Example**

**package** com;

**import** java.util.Stack;

**public** **class** StackDemo {

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

Stack ss = **new** Stack();

ss.push(100);

ss.push(200);

ss.push(300);

ss.push(400);

System.***out***.println(ss);

System.***out***.println("Top elements pop "+ss.pop());

System.***out***.println(ss);

System.***out***.println("Search "+ss.search(200));

System.***out***.println("Search "+ss.search(3000));

System.***out***.println("top element display "+ss.peek());

System.***out***.println(ss);

}

}

**Set :** it doesn’t allow duplicate. In Set some API maintain the order, unroder or stored.

**HashSet :**

**Example**

**package** com;

**import** java.util.HashSet;

**public** **class** HashSetDemo {

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

HashSet hs = **new** HashSet();

System.***out***.println("Size "+hs.size());

System.***out***.println("Empty "+hs.isEmpty());

hs.add(3);

hs.add(2);

hs.add(1);

hs.add(5);

hs.add(6);

hs.add(5);

System.***out***.println(hs);

System.***out***.println("Size "+hs.size());

System.***out***.println("Empty "+hs.isEmpty());

System.***out***.println("Search "+hs.contains(5));

System.***out***.println("Search "+hs.contains(7));

hs.remove(3);

System.***out***.println(hs);

hs.clear();

System.***out***.println(hs);

}

}

**LinkedHashMap :** this class internally extends HashMap. It maintain the orders.

**package** com;

**import** java.util.HashSet;

**import** java.util.LinkedHashSet;

**public** **class** LinkedHashMapDemo {

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

//HashSet hs = new HashSet();

LinkedHashSet hs = **new** LinkedHashSet();

hs.add(3);

hs.add(2);

hs.add("abc");

hs.add(5);

hs.add(6);

hs.add(5);

System.***out***.println(hs);

}

}

**TreeMap :** TreeMap internally implements SortedSet and SortedSet internally extends Set interface.

**Queue :** Queue means First In First Out.

**PriorityQueue :** First In First Out base upon the priority (lower priority).

**Map :** It is use to store the information in the form of key value pairs.

Where key is unique and value may be duplicate.

Key may be any type as well as value may be any type. It is good key must be string type because String is immutable class.

4 types

**HashMap : Display the data randomly.**

**LinkedHashMap : It maintain the order.**

**TreeMap :** TreeMap internally implements SortedMap interface and that interface extends Map interface.

**package** com;

**import** java.util.HashMap;

**import** java.util.LinkedHashMap;

**import** java.util.TreeMap;

**public** **class** LinkedHashMapDemo1 {

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

//HashMap hm = new HashMap();

//LinkedHashMap hm = new LinkedHashMap();

TreeMap hm = **new** TreeMap();

hm.put(2,"Ramesh");

hm.put(1,"Lokesh");

hm.put(4,"Suresh");

hm.put(3,"Rakesh");

hm.put(5, "Kumar");

System.***out***.println(hm);

}

}

Hashtable : By default all methods in Hashtable are synchronized.

Work is safe but slow in performance.

HashMap allow only one null key as well more than one null values.

Hashtable doesn’t allow null key as well as null value.

**package** com;

**import** java.util.HashMap;

**import** java.util.Hashtable;

**import** java.util.LinkedHashMap;

**import** java.util.TreeMap;

**public** **class** LinkedHashMapDemo1 {

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

HashMap hm = **new** HashMap();

//LinkedHashMap hm = new LinkedHashMap();

//TreeMap hm = new TreeMap();

// Hashtable hm = new Hashtable();

hm.put(2,"Ramesh");

hm.put(1,"Lokesh");

hm.put(4,"Suresh");

hm.put(3,"Rakesh");

hm.put("Ravi", "Kumar");

hm.put(**null**, "Patil");

hm.put(**null**, "Kumari");

hm.put(8, **null**);

System.***out***.println(hm);

}

}

Collection classes we store different types of Values.

**Collection Framework with generics.**

**ClassName<Type> obj = new ClassName<Type>();**

Type may be Wrapper classes (8 primitive) , String as well as all user-defined classes.

**Collection Framework with Complex objects.**

**Arrays and Collections**

**Arrays :** It is a type of class part of util package which provide set of methods which help to do operation on primitive arrays.

**Collections:** It is a type of class which help to do some operation on List type of classes.

ArrayList, LinkedList, Vector etc.

**Day 9 : 21-01-2020**

**Bubble Sort :** Bubble sort is an algorithms that compare the adjacent element and swap their position. The order can be ascending or descending.

**Selection Sort :** Selector sort is an algorithms that select the smallest/largest elements from an unsorted array in each iteration and places that elements at the beginning of unsorted list or array.

Set the first elements as minimum

Compare the minimum elements with second elements. If the second elements is smallest that minimums, then assign the second as minimum and continue the process till end.

**Insertion Sort :** Insertion sort work by inserting some key as value in the existing sorted filed. If construct the sorted array by inserting a single elements at a time.

**Sorting Example**

class SortingExample {

static void bubbleSort(int num[]) {

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

for(int j=0;j<num.length-1-i;j++) {

if(num[j]>num[j+1]) {

int temp = num[j];

num[j]=num[j+1];

num[j+1]=temp;

}

}

}

}

static void selectionSort(int num[]) {

for(int i=0;i<num.length-1;i++) {

int min = i;

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

if(num[j]<num[min]) {

min = j;

}

}

int temp = num[i];

num[i]=num[min];

num[min]=temp;

}

}

static void insertionSort(int num[]) {

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

int key = num[i];

int j=i-1;

while(j>=0 && key< num[j]) {

num[j+1]=num[j];

--j;

}

num[j+1]=key;

}

}

public static void main(String args[]) {

int num[]={3,2,6,4,1,5,7};

System.out.println("Before Sort");

for(int n:num) {

System.out.print(n+" ");

}

//bubbleSort(num);

//selectionSort(num);

insertionSort(num);

System.out.println();

System.out.println("After Sort");

for(int n:num) {

System.out.print(n+" ");

}

}

}