Class 31:-

OOPS

=====

OOPS stands for Object Oriented Programming System/Structure.

OOPS allows us to deals with real world entities using programming language.

We have following important features present in oops.

ex:

class

object

Abstraction

Encapsulation

Inheritance

Polymorphism

class

==========

A class is a blue print of an object.

A class is a logical entity.

A class is a collection of objects.

We can declare a class as follow.

syntax:

optional

|

modifier class class\_name <extends> parent\_classname

<implements> interface\_name

{

-

- //set of objects

-

}

A class will accept following modifiers.

ex:

default public

abstract final

Q) What is difference between default class and public class?

default class public class

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

To declare default class we should not To declare public class we should use

use any modifier. public modifier.

ex: ex:

class A public class A

{ {

} }

If we declare any class as default then we If we declare any class as public then we can

can access that class within the package. access that within the package and outside the

package.

Q) What is difference between abstract and final class?

abstract class final class

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

To declare abstract class we should use To declare final class we should use

abstract modifier. final modifier.

ex: ex:

abstract class A final class A

{ {

} }

Object creation is not possible. Object creation is possible.

Child creation is possible. Child creation is not possible.

Class 32:-

========

object

It is a outcome of a blue print.

It is a instance of a class.

Here instance means allocating memory for our data members.

It is a physical entity.

It is a collection of properties and behaviours.

ex:

Dog

|

|-----------------------------------------------|

Properties Behaviours

> Name > Barking

> Color > Eating

> Age > Sleeping

> Weight and etc.

> Height

and etc.

Memory space will be created when we create an object.

We can declare object as follow.

ex:

operator

|

Test t = new Test();

| | |

classname reference\_var constructor\_name

It is possible to create more then one object in a single class.

ex:

----

class Test

{

public static void main(String[] args)

{

Test t1=new Test();

Test t2=new Test();

Test t3=new Test();

System.out.println(t1.hashCode());

System.out.println(t2.hashCode());

System.out.println(t3.hashCode());

System.out.println(t1);//Test@Hexadecimal

System.out.println(t2.toString());

System.out.println(t3.toString());

}

}

Q) What is hashcode in java?

Every object JVM will create a unique identification number i.e hash code.

In order to read the hash code of an object we need to use hashCode() method.

A hashCode() method present in Object class.

Diagram: class32.1

Q) What is toString()?

Whenever we are trying to display object reference directly or indirectly toString() method

will be executed.

A toString() method is also present in Object class.

Q) What is the difference between class and object?

class object

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

To declare a class we will use class keyword. To declare object we will use new keyword.

It is a collection of objects. It is a collection of properties and behaviours.

It is a blueprint of an object. It is a outcome of a blue print.

It is a logical entity. It is a physical entity.

It can't manipulate. It can manipulate.

It does not allocate the memory. It allocates the memory.

It declares once. It declares many times.

Data Hiding

===========

Data hiding is a process of hiding object data from the outsiders.

Using private modifier we can implements data hiding concept.

The main objective of data hiding is to provide security.

ex:

---

class Account

{

private double balance=5000d;

}

class Student

{

public static void main(String[] args)

{

Account account=new Account();

System.out.println(account.balance);

}

}

Abstraction

===========

Hiding internal implementation and highlighting the set of services is called abstraction.

Using abstract classes and interfaces we can implements abstraction.

The best example of abstraction is GUI ATM machine because bank people will hide internal implementation and highlights the set of services like banking, withdrawl, mini statement and etc.

The main advantages of abstraction are.

1) It gives security because it will hide internal implementation.

2) Enhancement becomes more easy because without effecting enduser they can perform any changes

in our internal system.

3) It provides flexibility to the end user to use the system.

4) It improves maintainability of an application.

Class 33:-

Encapsulation

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

The process of encapsulating or grouping variables and it's associate methods in a single entity is called encapsulation.

Diagram: class33.1

A class is said to be encapsulated class if it supports data hiding and abstraction.

Abstraction is used to hide the data and encapsulation is used to protect the data.

In encapsulation , for every variable we need to declare setter and getter methods.

Diagram: class33.2

The main advantages of encapsulation are.

1) It gives security.

2) Enhancements becomes more easy.

3) It provides flexibility to the end user to use the system.

4) It improves maintainability of an application.

The main disadvantage of encapsulation is , it will increase the length of our code and slow down the execution process.

ex:

----

class Student

{

//current class variables

private int studId;

private String studName;

private double studFee;

//setter methods

public void setStudId(int studId)

{

this.studId=studId;

}

public void setStudName(String studName)

{

this.studName=studName;

}

public void setStudFee(double studFee)

{

this.studFee=studFee;

}

//getter methods

public int getStudId()

{

return studId;

}

public String getStudName()

{

return studName;

}

public double getStudFee()

{

return studFee;

}

}

class Test

{

public static void main(String[] args)

{

Student s=new Student();

s.setStudId(101);

s.setStudName("Alan");

s.setStudFee(1000d);

System.out.println("Student Id :"+s.getStudId());

System.out.println("Student Name :"+s.getStudName());

System.out.println("Student Fee :"+s.getStudFee());

}

}

Q) What is tightly encapsulated class?

A class is said to be tightly encapsulated class if and only if all variables of that class declared as private. Here we don't need to check these variables having setter and getter method or not.

ex:

class A

{

private int i;

private int j;

}

What is the difference between Abstraction and Encapsulation?

Abstraction Encapsulation

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

Hiding internal implementation and The process of encapsulating or grouping

highlighting the set of services is called variables and its associate methods in a

abstraction. single entity is called encapsulation.

It is used to hide the data. It is used to protect the data.

Using abstract classes and interfaces we Using access modifiers we can implements

can implements abstraction. encapsulation.

It is a process of gaining the information. It is a process of containing the information.

It solves an issue at design level. It solves an issue at implementation level.

Q)What is the difference between POJO class and Java Bean class?

POJO Java Bean

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

It can’t be serialized. It can be serialized.

Fields can have any visibility. Fields can have only private visibility.

There may or may not have 0-arg constructor. It must have 0-argument constructor.

It does not extends any other class. It can extends.

It does not implement any other interface. It can implements.

It does not use any outside annotation. It uses outside annotation.

Is-A relationship

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

Is-A relationship is also known as inheritance.

Using extends keyword we can implements Is-A Relationship.

The main objective of Is-A relationship is to provide reusability.

ex:

---

class Parent

{

public void methodOne()

{

System.out.println("Method One");

}

}

class Child extends Parent

{

public void methodTwo()

{

System.out.println("Method Two");

}

}

class Test

{

public static void main(String[] args)

{

Parent p=new Parent();

p.methodOne();

Child c=new Child();

c.methodOne();

c.methodTwo();

Parent p1=new Child();

p1.methodOne();

//Child c1=new Parent();

}

}

Conclusion

----------

Whatever parent having properties by default it comes to child. But whatever child having properties it never goes back to parent.

A parent reference can hold child object but child reference can't hold parent object.

Inheritance

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

Inheritance is a mechanism where we will derived a class in the presence of existing class.

Inheritance is a mechanism where one class will inherit the properties of another class.

The main objective of inheritance is to provide reusability.

Diagram: class33.3

We have following list of inheritance in java.

1) Single Level inheritance

2) Multi Level inheritance

3) Multiple inheritance

4) Hirarchical inheritance

5) Hybrid inheritance

1) Single Level inheritance

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

If we derived a class in the presence of one base class is called single level inheritance.

Diagram:

A (parent class/ super class/ base class)

|

|

|

B (child class / sub class / derived class)

ex:

----

class A

{

public void m1()

{

System.out.println("M1-Method");

}

}

class B extends A

{

public void m2()

{

System.out.println("M2-Method");

}

}

class Test

{

public static void main(String[] args)

{

A a=new A();

a.m1();

B b=new B();

b.m1();

b.m2();

}

}

ex:

----

class A

{

int i=10;

}

class B extends A

{

int j=20;

}

class Test

{

public static void main(String[] args)

{

A a=new A();

System.out.println(a.i);//10

B b =new B();

System.out.println(b.i+" "+b.j);//10 20

}

}

2) Multi Level inheritance

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

If a class is derived by using one base class and that class is derived from another base class is called multi level inheritance.

ex:

---

class A

{

public void m1()

{

System.out.println("M1-Method");

}

}

class B extends A

{

public void m2()

{

System.out.println("M2-Method");

}

}

class C extends B

{

public void m3()

{

System.out.println("M3-Method");

}

}

class Test

{

public static void main(String[] args)

{

A a=new A();

a.m1();

B b=new B();

b.m1();

b.m2();

C c=new C();

c.m1();

c.m2();

c.m3();

}

}

3) Multiple inheritance

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

In java, a class can't extends more then one class simultenously because java does not support multiple inheritance.

ex:

class A

{

}

class B

{

}

class C extends A,B --> invalid

{

}

Interface can extends more then one interface so we can achieve multiple inheritance concept through interfaces.

ex:

interface A

{

}

interface B

{

}

interface extends A,B --> valid

{

}

If our class does not extends any other class then it is a direct child class of Object class.

ex: diag:

class A Object

{ |

|

} A

If our class extends some other class then it is a indirect child class of Object class.

ex: diag:

class A Object

{ |

} |

class B extends A A

{ |

} |

B

Java does not support cyclic inheritance.

ex:

class A extends B

{

}

class B extends A

{

}

Q) Why java does not support multiple inheritance?

There may a chance of raising ambiguity/ increasing confusion (aspasthta badhana) problem that's why java does not support multiple inheritance.

ex:

p1.m1() p2.m1()

|-------------------------------------|

|

c.m1()

Interview Question

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

Q) What is jagged array?

Jagged array is also known as array of arrays.

It is a multi dimensional array where each row having different size.

Diagram: class33.4

ex:

import java.util.Scanner;

class Test

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

int[][] arr=new int[3][];

arr[0]=new int[3];

arr[1]=new int[2];

arr[2]=new int[4];

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

{

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

{

System.out.println("Enter the element of arr["+i+"]["+j+"] :");

arr[i][j]=sc.nextInt();

}

}

//reading

for(int[] a:arr)

{

for(int i:a)

{

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

}

//new

System.out.println();

}

}

}

Class 34:-

4) Hierarchical inheritance

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

If we derived multiple class in the presence of one base class is called hierarchical inheritance.

Diagram:

A

|

|-------------------------------|

B C

ex:

---

class A

{

public void m1()

{

System.out.println("M1-Method");

}

}

class B extends A

{

public void m2()

{

System.out.println("M2-Method");

}

}

class C extends A

{

public void m3()

{

System.out.println("M3-Method");

}

}

class Test

{

public static void main(String[] args)

{

A a=new A();

a.m1();

B b=new B();

b.m1();

b.m2();

C c=new C();

c.m1();

c.m3();

}

}

5) Hybrid inheritance

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

It is a combination of more then one inheritance.

Java does not support hybrid inheritance.

Diagram:

A

|

|-------------------------------|

B C

|-------------------------------|

|

D

Has-A relationship

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

Has-A relationship is also known as composition and aggregation.

There is no specific keyword to implements Has-A relationship but mostly we will use new operator.

The main objective of Has-A relationship is to provide reusability.

Has-A relationship will increase dependency between two components.

ex:

class Trainer

{

}

class Course

{

Trainer t=new Trainer();

-

-

}

ex:

---

class Ihub

{

public String courseName()

{

return "Full Stack Java + AWS";

}

public double courseFee()

{

return 30000d;

}

public String trainerName()

{

return "Niyaz Sir";

}

}

class Usha

{

public void getCourseDetails()

{

Ihub i=new Ihub();

System.out.println("Course Name :"+i.courseName());

System.out.println("Course Fee :"+i.courseFee());

System.out.println("Trainer Name :"+i.trainerName());

}

}

class Student

{

public static void main(String[] args)

{

Usha u=new Usha();

u.getCourseDetails();

}

}

composition

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

Without existing container object there is no chance of having contained object then the relationship between container and contained object is called composition which is strongly association.

Diagram: class34.1

aggregation

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

Without existing container object there is a chance of having contained object then the relationship between container and contained object is called aggregation which is loosely association.

Diagram: class34.2

Method overloading

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

Having same method name with different parameters/signatures in a single class is called method overloading.

All the methods present in a class are called overloaded methods.

Method overloading will reduce complexity of the programming.

Method resolution will taken care by a compiler based on reference type.

ex:

---

class MeeSeva

{

//overloaded methods

public void search(int voterId)

{

System.out.println("Details Found via voterId");

}

public void search(String houseNo)

{

System.out.println("Details Found via houseNo");

}

public void search(long aadharNo)

{

System.out.println("Details Found via aadharNo");

}

}

class Test

{

public static void main(String[] args)

{

MeeSeva ms=new MeeSeva();

ms.search(101);

ms.search("1-4-6/3");

ms.search(1234L);

}

}

Method overriding

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

Having same method name with same parameters into two different classes is called method overriding.

Methods which are present in parent class are called overridden methods.

Methods which are present in child class are called overridding methods.

Method resolution will taken care by JVM based on runtime object.

ex:

---

class Parent

{

public void property()

{

System.out.println("Cash+Gold+Land+House");

}

public void marry()

{

System.out.println("Trisha");

}

}

class Child extends Parent

{

public void marry()

{

System.out.println("Rashmika");

}

}

class Test

{

public static void main(String[] args)

{

Parent p=new Parent();

p.property(); // Cash+Gold+land+House

p.marry(); // Trisha

Child c=new Child();

c.property(); // Cash+Gold+Land+House

c.marry(); // Rashmika

Parent p1=new Child();

p1.property(); // Cash+Gold+Land+House

p1.marry(); // Rashmika

}

}

If we declare any method as final then overridding of that method is not possible.

ex:

---

class Parent

{

//overridden methods

public void property()

{

System.out.println("Cash+Gold+Land+House");

}

public final void marry()

{

System.out.println("Trisha");

}

}

class Child extends Parent

{

//overridding methods

public void marry()

{

System.out.println("Rashmika");

}

}

class Test

{

public static void main(String[] args)

{

Parent p=new Parent();

p.property(); // Cash+Gold+land+House

p.marry(); // Trisha

Child c=new Child();

c.property(); // Cash+Gold+Land+House

c.marry(); // Rashmika

Parent p1=new Child();

p1.property(); // Cash+Gold+Land+House

p1.marry(); // Rashmika

}

}

If parent don't want to share any properties to child class then we need to declare variables and methods as private.

Private methods can't be override.

ex:

class Parent

{

//overridden methods

private void property()

{

System.out.println("father property");

}

}

class Child extends Parent

{

//overridding methods

public void property()

{

System.out.println("child property");

}

}

class Test

{

public static void main(String[] args)

{

Parent p=new Parent();

p.property(); // Cash+Gold+land+House

Child c=new Child();

c.property(); // Cash+Gold+Land+House

}

}

Q) Can we overload main method in java?

Yes, we can overload main method in java but JVM always execute main method with String[] parameter only.

ex:

class Test

{

public static void main(String[] args)

{

System.out.println("String arg method");

}

public static void main(int[] iargs)

{

System.out.println("int arg method");

}

}

Method Hiding

===========

Method hiding is exactly same as method overriding with following differences.

Method Overriding Method Hiding

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

All methods present in method overriding All methods present in method hiding must be

must be non-static. static.

Method resolution will taken care by Method resolution will taken care by compiler based

JVM based on runtime object. on reference type.

It is also known as dynamic polymorphism, It is also known as static polymorphism, compile time

runtime polymorphism or late binding. polymorphism or late binding.

class Parent

{

public static void property(){

System.out.println("Cash+Gold+Land+House");

}

public static void marry(){

System.out.println("Trisha");

}

}

class Child extends Parent{

public static void marry(){

System.out.println("Rashmika");

}

}

class Test {

public static void main(String[] args)

{

Parent p=new Parent();

p.property(); // Cash+Gold+land+House

p.marry(); // Trisha

Child c=new Child();

c.property(); // Cash+Gold+Land+House

c.marry(); // Rashmika

Parent p1=new Child();

p1.property(); // Cash+Gold+Land+House

p1.marry(); // Trisha

}

}

Q) Can we override main method in java?

No, we can't override main method in java because it is static.

Class 35:-

Polymorphism

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

polymorphism has taken from Greek word.

Here poly means many and morphism means forms.

The ability to represent in different forms is called polymorphism.

The main objective of polymorphism is to provide flexibility.

Diagram: class35.1

In java we have two types of polymorphism.

1) Compile time polymorphism / Static polymorphism / Early Binding

2) Runtime polymorphism / Dynamic polymorphism / Late Binding

1) Compile time polymorphism

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

A polymorphism which exhibits(display) at compile time is called compile time polymorphism.

ex:

Method overloading

Method Hiding

2) Runtime polymorphism

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

A polymorphism which exhibits at runtime is called runtime polymorphism.

ex:

Method Overriding

Diagram: class35.2

Constructor

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

A constructor is a special method which is used to initialized an object.

Having same name as class name is called constructor.

A constructor will execute when we create an object.

A constructor does not allow any returntype.

A constructor will accept following modifiers.

ex:

default

public

private

protected

In java, constructors are divided into two types.

1) Userdefined constructor

2) Default constructor

1) Userdefined constructor

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

Constructor which is created by the programmer or user based on the application requirement is called userdefined constructor.

It is classified into two types.

i) Zero Argument constructor

ii) Parameterized constructor

i) Zero Argument constructor

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

Suppose if we are not passing any argument to userdefined constructor then that constructor is called zero argument constructor.

ex:

---

class Test

{

Test()

{

System.out.println("0-arg const");

}

public static void main(String[] args)

{

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

}

}

o/p:

main-method

ex:

---

class Test

{

public Test()

{

System.out.println("0-arg const");

}

public static void main(String[] args)

{

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

Test t=new Test();

Test t=new Test();

}

}

ex:

---

class Test

{

private Test()

{

System.out.println("0-arg const");

}

public static void main(String[] args)

{

Test t1=new Test();

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

Test t2=new Test();

}

}

o/p:

0-arg const

main-method

0-arg const

ex:

--

class Test

{

protected Test()

{

System.out.println("0-arg const");

}

public static void main(String[] args)

{

Test t1=new Test();

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

Test t2=new Test();

}

}

o/p:

0-arg const

main-method

0-arg const

ii) Parameterized constructor

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

Suppose if we are passing atleast one argument to userdefined constructor then that constructor is called parameterized constructor.

ex

---

class Employee

{

//current class variables

private int empId;

private String empName;

private double empSal;

public Employee(int empId,String empName,double empSal)

{

this.empId=empId;

this.empName=empName;

this.empSal=empSal;

}

public void getEmployeeDetails()

{

System.out.println("Employee Id :"+empId);

System.out.println("Employee Name :"+empName);

System.out.println("Employee Salary :"+empSal);

}

}

class Test

{

public static void main(String[] args)

{

Employee e=new Employee(101,"Alan Morries",1000d);

e.getEmployeeDetails();

}

}

2) Default constructor

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

It is a compiler generated constructor for every java program where we are not defining atleast

zero argument constructor.

To see the default constructor we need use below command.

ex:

javap -c Test

Diagram: class35.3

Constructor Overloading

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

Having same constructor name with different parameters or signatures in a single class is called constructor overloading.

ex:

---

class Test

{

Test()

{

System.out.println("0-arg const");

}

Test(int i)

{

System.out.println("int-arg const");

}

Test(double d)

{

System.out.println("double-arg const");

}

public static void main(String[] args)

{

Test t1=new Test();

Test t2=new Test(10);

Test t3=new Test(10.5d);

}

}

this keyword

===========

A this keyword is a java keyword which is used to refer current class object reference.

We can utility this keyword in following ways.

1) To refer current class variables

2) To refer current class methods

3) To refer current class constructors

1) To refer current class variables

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

class A

{

int i=10;

int j=20;

A(int i,int j)

{

System.out.println(this.i+" "+this.j); // 10 20

System.out.println(i+" "+j); // 100 200

}

}

class Test

{

public static void main(String[] args)

{

A a=new A(100,200);

}

}

2) To refer current class methods

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

class A

{

public void m1()

{

System.out.println("M1-Method");

this.m2();

}

public void m2()

{

System.out.println("M2-Method");

}

}

class Test

{

public static void main(String[] args)

{

A a=new A();

a.m1();

}

}

3) To refer current class constructor

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

class A

{

A()

{

System.out.println("0-arg const");

}

A(int i)

{

this();

System.out.println("int-arg const");

}

A(double d)

{

this(10);

System.out.println("double-arg const");

}

}

class Test

{

public static void main(String[] args)

{

A a=new A(10.5d);

}

}

super keyword

===========

A super keyword is a java keyword which is used to refer super class object reference.

We can utility super keyword in following ways.

1) To refer super class variables

2) To refer super class methods

3) To refer super class constructors

1) To refer super class variables

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

class A

{

int i=10;

int j=20;

}

class B extends A

{

int i=100;

int j=200;

B(int i,int j)

{

System.out.println(i+" "+j); // 1000 2000

System.out.println(this.i+" "+this.j); // 100 200

System.out.println(super.i+" "+super.j); //10 20

}

}

class Test

{

public static void main(String[] args)

{

B b=new B(1000,2000);

}

}

2) To refer super class methods

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

class A

{

public void m1()

{

System.out.println("M1-Method");

}

}

class B extends A

{

public void m2()

{

super.m1();

System.out.println("M2-Method");

}

}

class Test

{

public static void main(String[] args)

{

B b=new B();

b.m2();

}

}

API

====

API stands for Application Programming interface.

It is a base for the programmer to develop software applications.

API is a collection of packages.

In java, API is divided into three types.

1) Predefined API

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

Built-in API is called predefined API.

ex:

https://docs.oracle.com/javase/8/docs/api/

2) Userdefined API

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

API which is created by the user based on the application requirements is called

Userdefined API.

3) Third party API

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

API which is given by third party vendor is called third party API.

ex:

JAVAZOOM API

iText API

and etc.

Interview Question

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

Q) write a java program to display sub array equals to given sum?

input:

arr = 1 2 3 7 6

sum = 12

output:

2 3 7

approach1

---------

class Test

{

public static void main(String[] args)

{

int[] arr ={1,2,3,7,6};

int sum = 12;

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

{

int currentSum=0;

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

{

currentSum+=arr[j];

if(currentSum==sum)

{

for(int k=i;k<=j;k++)

{

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

}

}

}

}

}

}

approach2

----------

class Test

{

public static void main(String[] args)

{

int[] arr ={1,2,3,7,5};

int sum = 12;

int cnt=0;

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

{

int currentSum=0;

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

{

currentSum+=arr[j];

if(currentSum==sum)

{

cnt=1;

for(int k=i;k<=j;k++)

{

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

}

}

}

if(cnt==1)

{

break;

}

}

}

}

Class36:-

Interface

=========

Interface is a collection of zero or more abstraction methods.

Abstract methods are incomplete methods because they ends with semicolon and does not have any body.

ex

void m1();

It is not possible to create object for interfaces.

To write the implementation of abstract methods we will use implementation class.

It is possible to create object for implementation class because it contains method with body.

By default every abstract method is a public and abstract.

ex:

public abstract void m1();

Interface contains only constants i.e public static final.

We can declare interface as follow.

syntax:

-----

interface <interface\_name>

{

-

- //abstract methods

- //constants

-

}

If we know service requirement specification then we need to use interface.

Diagram: class36.1

ex:

----

interface A

{

//abstract method

public abstract void m1();

}

class B implements A

{

public void m1()

{

System.out.println("M1-Method");

}

}

class Test

{

public static void main(String[] args)

{

A a=new B();

a.m1();

}

}

ex:

---

interface A

{

//abstract method

public abstract void m1();

}

class Test

{

public static void main(String[] args)

{

A a=new A()

{

public void m1()

{

System.out.println("From M1 Method");

}

};

a.m1();

}

}

If interface contains four methods then we need to override all methods otherwise we will get compile time error.

ex:

---

interface A

{

public abstract void show();

public void display();

abstract void see();

void view();

}

class B implements A

{

public void show()

{

System.out.println("Show Method");

}

public void display()

{

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

}

public void see()

{

System.out.println("See Method");

}

public void view()

{

System.out.println("View Method");

}

}

class Test

{

public static void main(String[] args)

{

A a=new B();

a.show();

a.display();

a.see();

a.view();

}

}

A class can't extends more then one class simultenously.

But interface can extends more then one interface.

ex:

---

interface A

{

void m1();

}

interface B

{

void m2();

}

interface C extends A,B

{

void m3();

}

class D implements C

{

public void m1()

{

System.out.println("M1-Method");

}

public void m2()

{

System.out.println("M2-Method");

}

public void m3()

{

System.out.println("M3-Method");

}

}

class Test

{

public static void main(String[] args)

{

C c=new D();

c.m1();

c.m2();

c.m3();

}

}

A class can implements more then one interface.

ex:

---

interface Father

{

float HT=6.2f;

void height();

}

interface Mother

{

float HT=5.8f;

void height();

}

class Child implements Father,Mother

{

public void height()

{

float height=(Father.HT+Mother.HT)/2;

System.out.println("Child Height is ="+height);

}

}

class Test

{

public static void main(String[] args)

{

Child c=new Child();

c.height();

}

}

Note:

------

Interface is a collection of abstract methods, default methods and static methods.

Marker interface

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

Interface which does not have methods and constants is called marker interface.

In general, empty interface is called marker interface.

By using marker interface we will get some ability to do.

We have following list of marker interfaces.

ex:

Serializable

Cloneable

Remote and etc.

ex:

class Student implements java.io.Serializable

{

private int studId;

public void setStudId(int studId)

{

this.studId=studId;

}

public int getStudId()

{

return studId;

}

}

Abstract class

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

Abstract class is a collection of zero or more abstract methods and concrete methods.

A abstract keyword is applicable for classes and methods but not for variables.

It is not possible to create object for abstract class.

To write the implementation of abstract methods we will use sub classes.

By default every abstract method is a public and abstract.

Abstract class contains only instance variables.

syntax:

------

abstract class <class\_name>

{

-

- //abstract method

- //concrete method

- //instance variable

-

}

If we know partial implementation then we need to use abstract class.

ex:

---

abstract class Plan

{

//instance variable

protected double rate;

//abstract method

public abstract void getRate();

//concrete method

public void calculateBill(int units)

{

System.out.println("Total Units :"+units);

System.out.println("Total Bill :"+ units\*rate);

}

}

class DomesticPlan extends Plan

{

public void getRate()

{

rate=2.5d;

}

}

class CommercialPlan extends Plan

{

public void getRate()

{

rate=5.0d;

}

}

class Test

{

public static void main(String[] args)

{

DomesticPlan dp=new DomesticPlan();

dp.getRate();

dp.calculateBill(250);

CommercialPlan cp=new CommercialPlan();

cp.getRate();

cp.calculateBill(250);

}

}

Q) What is the difference between interface and abstract class?

Interface Abstract class

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

To declare interface we will use interface To declare abstract class we will use abstract

keyword. keyword.

It is a collection of abstract methods, default It is a collection of abstract methods and concrete

methods and static methods. methods.

It contains constants. It contains instance variables.

We can achieve multiple inheritance. We can't achieve multiple inheritance.

We will use implementation class to write the We will use sub class to write the logic for

logic for abstract methods. abstract methods.

If we know only specification then we need If we know partial implementation then we need to use

to use interface. abstract class.

Abstraction

===========

Hiding internal implementation and highlighting the set of services is called abstraction.

Using abstract class and interfaces we can implements abstraction.

ex:

abstract class Animal

{

//abstract method

public abstract void makeSound();

}

class Cat extends Animal

{

public void makeSound()

{

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

}

}

class Test

{

public static void main(String[] args)

{

Cat c=new Cat();

c.makeSound();

}

}

Class37:-

Package

========

A package is a collection of classes,interfaces,enums and annotations.

Enum is a special class and annotation is a special interface.

In general , a package is a collection of classes and interfaces.

A package is also known as folder or a directory.

In java packages are divided into two types.

1) Predefined packages

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

Built-In packages are called predefined packages.

ex:

java.lang

java.io

java.util

java.util.stream

java.time

and etc.

2) Userdefined packages

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

A package which is created by the user based on the application requirement is called predefined package.

To declare predefined package we need to use package keyword.

It is recommanded to declare a package name in the reverse order of url.

syntax:

-----

package <package\_name>;

ex:

package com.google.www;

ex:

----

package com.ihub.www;

import java.util.Calendar;

class Test

{

public static void main(String[] args)

{

Calendar c=Calendar.getInstance();

int h=c.get(Calendar.HOUR\_OF\_DAY);

if(h<12)

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

else if(h<16)

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

else if(h<20)

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

else

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

}

}

We can compile above program by using below command.

ex:

current directory

|

javac -d . Test.java

|

destination folder

We can run above program by using below command.

ex:

java com.ihub.www.Test

|

package name

Enum

=====

Enum is a group of named constants.

Enum concept introduced in 1.5 version.

Using enum we can create our own datatype called enumerated datatype.

When compare to old language enum, java enum is more powerful.

To declare the enum we will use enum keyword.

syntax:

------

enum type\_name

{

value1,value2,.....,valueN

}

Internal implementation of enum

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

Every enum internally implements as class concept and it extends with java.lang.Enum class.

Every enum constant is a reference variable of enum type.

ex:

enum Months final class Months extends java.lang.Enum

{ {

JAN,FEB,MAR ==> public static final Months JAN=new Months();

} public static final Months FEB=new Months();

public static final Months MAR=new Months();

}

Declaration and Usage of enum

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

enum Months

{

JAN,FEB,MAR

}

class Test

{

public static void main(String[] args)

{

Months m=Months.FEB;

System.out.println(m);//FEB

}

}

ex:

---

enum Months

{

JAN,FEB,MAR

}

class Test

{

public static void main(String[] args)

{

Months m=Months.FEB;

switch(m)

{

case JAN: System.out.println("January"); break;

case FEB: System.out.println("February"); break;

case MAR: System.out.println("March"); break;

}

}

}

java.lang.Enum class

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

The power to enum will be inherited from java.lang.Enum class.

It contains following two methods.

1) values()

---------

It is used to read set of constants from enum.

2) ordinal()

-----------

It is used to display ordinal number. (index number)

ex:

---

enum Week

{

MON,TUE,WED,THU,FRI,SAT,SUN

}

class Test

{

public static void main(String[] args)

{

Week[] w=Week.values();

for(Week w1:w)

{

System.out.println(w1+" ------------- "+w1.ordinal());

}

}

}

When compare to old language enum, java enum is more powerful because in addition to constants

we can declare variables, methods and constructors.

ex:

----

enum Cloths

{

SILK,COTTON,KHADI;

Cloths()

{

System.out.println("constructor");

}

}

class Test

{

public static void main(String[] args)

{

Cloths c=Cloths.SILK;

}

}

ex:

---

enum Cloths

{

SILK,COTTON,KHADI;

static int i=100;

public static void main(String[] args)

{

System.out.println(i);

}

}

Singleton Class

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

Singleton is one of the design pattern which allows us to create only one object for a class.

Using a class name if we call any method and that method returns same class object is called singleton class.

ex:

Calendar c = Calendar.getInstance();

LocalDate date = LocalDate.now();

LocalTime time = LocalTime.now();

To create a singleton class we required private constructor and static method.

ex:

---

class Singleton

{

static Singleton singleton=null;

private Singleton() // (private constractor)

{

}

public static Singleton getInstance() // singleton is return type and method name is getInstance

{

if(singleton==null)

{

singleton=new Singleton(); //creating object

}

return singleton;

}

}

class Test

{

public static void main(String[] args)

{

Singleton s1=Singleton.getInstance();

System.out.println(s1.hashCode());

Singleton s2=Singleton.getInstance();

System.out.println(s2.hashCode());

Singleton s3=Singleton.getInstance();

System.out.println(s3.hashCode());

}

}

Wrapper classes

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

The main objective of wrapper classes are.

1) To wrap primitive type to wrapper object and vice versa.

2) To define several utility method.

ex:

Primitive type wrapper class

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

byte Byte

short Short

int Integer

long Long

float Float

double Double

boolean Boolean

char Character

constructor

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

There are two ways to create object for wrapper classes. One will take corresponding primitive as an argument and another will take corresponding String as an argument.

ex:

wrapper class constructor

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

Byte byte or String

Short short or String

Integer int or String

Long long or String

Float float or String

Double double or String

Boolean boolean or String

Character char

ex:

---

class Test

{

public static void main(String[] args)

{

Integer i1=new Integer(10);

System.out.println(i1); //10

Integer i2=new Integer("20");

System.out.println(i2); //20

}

}

ex:

---

class Test

{

public static void main(String[] args)

{

Boolean b1=new Boolean(true);

System.out.println(b1);

Boolean b2=new Boolean("false");

System.out.println(b2);

}

}

ex:

---

class Test

{

public static void main(String[] args)

{

Character c=new Character('a');

System.out.println(c);//a

}

}

Utility methods

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

1) parseXxx()

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

It is used to convert string to primitive type.

ex:

class Test

{

public static void main(String[] args)

{

String str="10";

int i= Integer.parseInt(str);

System.out.println(i); // 10

long l= Long.parseLong(str);

System.out.println(l); // 10

float f= Float.parseFloat(str);

System.out.println(f); // 10.0

double d= Double.parseDouble(str);

System.out.println(d); // 10.0

}

}

2) toString()

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

It is used to convert wrapper object to string.

ex:

--

class Test

{

public static void main(String[] args)

{

Integer i1=new Integer(10);

String str =i1.toString();

System.out.println(str); // 10

}

}

3) xxxValue()

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

It is used to convert wrapper object to primitive type.

ex:

class Test

{

public static void main(String[] args)

{

Integer i1=new Integer(10);

byte b = i1.byteValue();

System.out.println(b);

short s = i1.shortValue();

System.out.println(s);

}

}

Q) Write a java program to perform sum of two binary numbers?

input:

1010

0101

output:

1111

ex:

---

import java.util.Scanner;

class Test

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the first binary number :");

String binary1=sc.next();

System.out.println("Enter the second binary number :");

String binary2=sc.next();

//convert binary to decimal

int a = Integer.parseInt(binary1,2);

int b = Integer.parseInt(binary2,2);

int c = a + b;

//convert decimal to binary

String result=Integer.toBinaryString(c);

System.out.println(result);

}

}

Assignment

==========

Q) Write a java program to display second highest element from given array?

input:

6 9 2 4 1 2 7

output: 7

Class 38:-

Inner classes

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

Sometimes we will declare a class inside another class such concept is called inner class.

ex:

class Outer\_class

{

class Inner\_class

{

-

-

-

}

}

Inner classes introduced as a part of event handling to remove GUI bugs.

But due to powerful features and benefits of inner classes. Programmers started to use inner classes in regular programming.

Inner class does not allow static members.

Accessing inner class data from static area of outer class

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

class Outer

{

class Inner

{

public void m1()

{

System.out.println("Inner-M1 Method");

}

}

public static void main(String[] args)

{

Outer.Inner i=new Outer().new Inner();

i.m1();

}

}

If we compile above program we will get two .class files i.e Outer.class and Outer$Inner.class.

ex:

----

class Outer

{

class Inner

{

public void m1()

{

System.out.println("Inner-M1 Method");

}

}

public static void main(String[] args)

{

new Outer().new Inner().m1();

}

}

Accessing inner class data from non-static area of outer class

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

class Outer

{

class Inner

{

public void m1()

{

System.out.println("Inner-M1 Method");

}

}

public void m2()

{

Inner i=new Inner();

i.m1();

}

public static void main(String[] args)

{

Outer o=new Outer();

o.m2();

}

}

class Outer

{

class Inner

{

public static void m1()

{

System.out.println("Inner-M1 Method");

}

}

public void m2()

{

Inner i=new Inner();

i.m1();

}

public static void main(String[] args)

{

Outer o=new Outer();

o.m2();

}

}

o/p:

C.T.E : Illegal static declaration in inner class

Types of objects in java

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

We have two types of objects in java.

1) Immutable object

2) Mutable object

1) Immutable object

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

After object creation if we perform any changes then for every change a new object is will be created such type of object is called immutable object.

ex:

String and Wrapper classes

2) Mutable object

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

After object creation if we perform any changes then all the required changes will be done in a same object such type of object is called mutable object.

ex:

StringBuffer and StringBuilder

**String**

=========

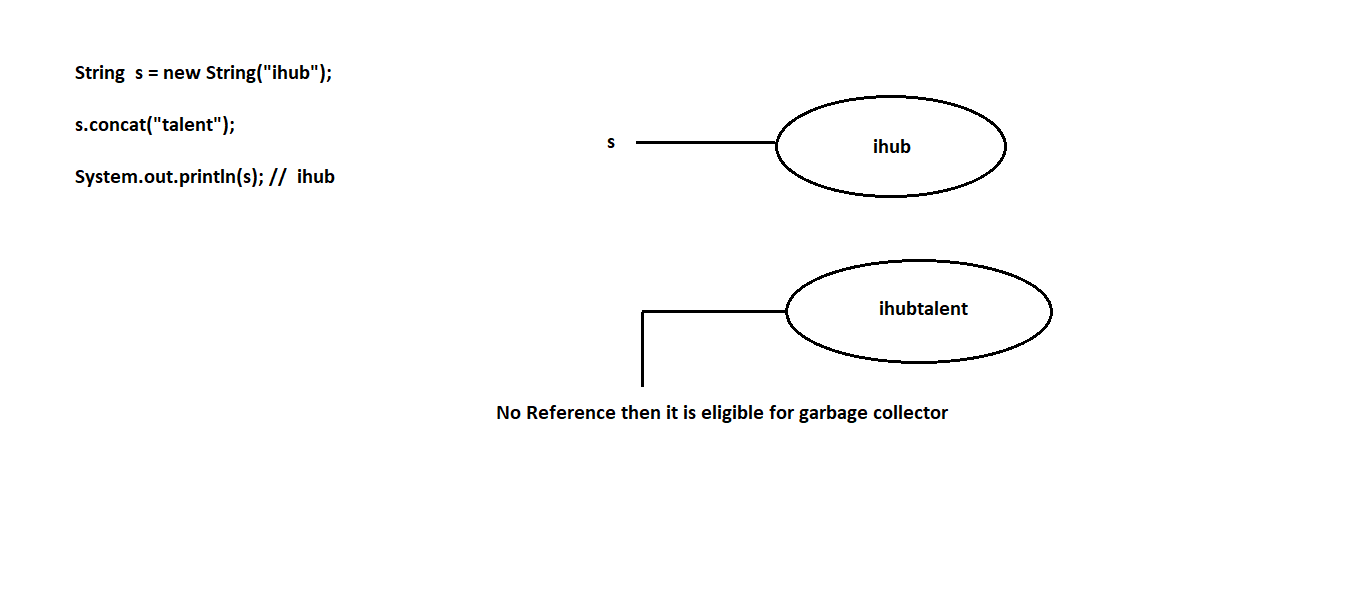
It is a collection of characters which is enclosed in a double quotation.

case1:

------

Once if we create a String object we can't perform any changes. If we perform any changes then for every change a new object will be created such behavior is called immutability of an object.

Diagram: class38.1



case2:

-----

What is the difference between == and .equals() method?

==

----

It is a comparision operator which returns boolean value either true or false.

It is used for reference comparision or address comparision.

ex:

class Test

{

public static void main(String[] args)

{

String s1=new String("ihub");

String s2=new String("ihub");

System.out.println(s1==s2); // false

}

}

.equals()

----------

It is a predefined method present in String which returns boolean value either true or false.

It is used for content comparision and it is a case sensitive.

ex:

---

class Test

{

public static void main(String[] args)

{

String s1=new String("ihub");

String s2=new String("ihub");

System.out.println(s1.equals(s2)); // true

}

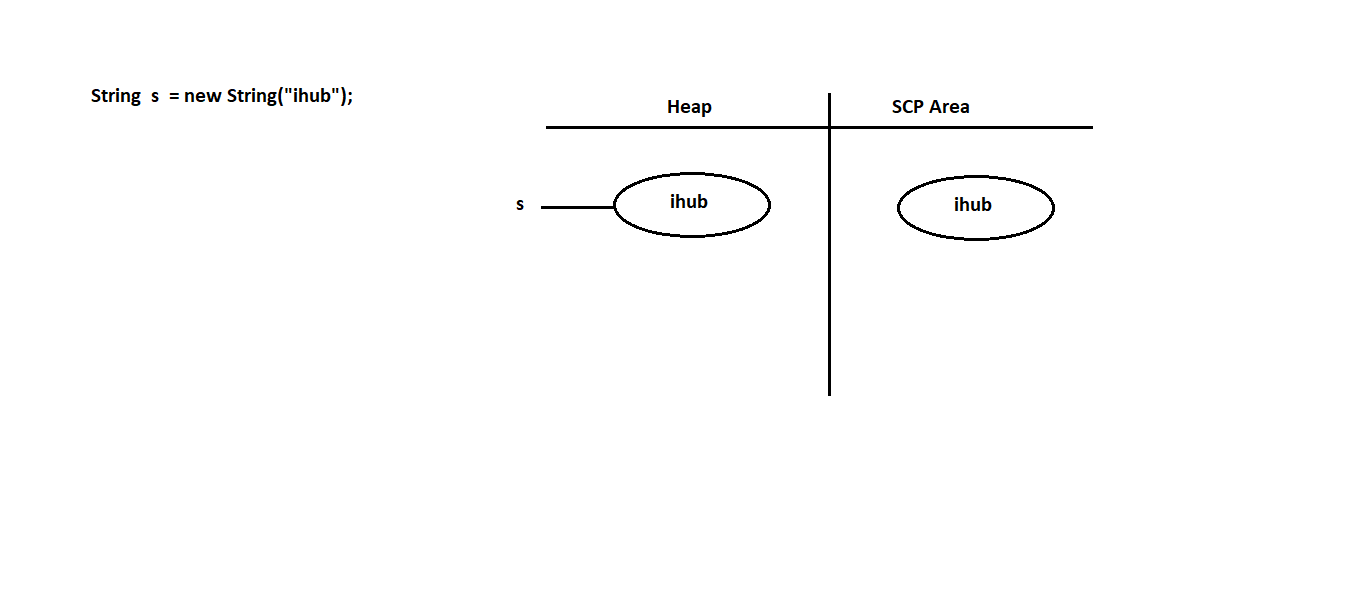
}

case3:

-----

Once if we create a String object. Two objects will be created one is on heap and another is on SCP area. But 's' always points to heap area.

Diagram: class38.2

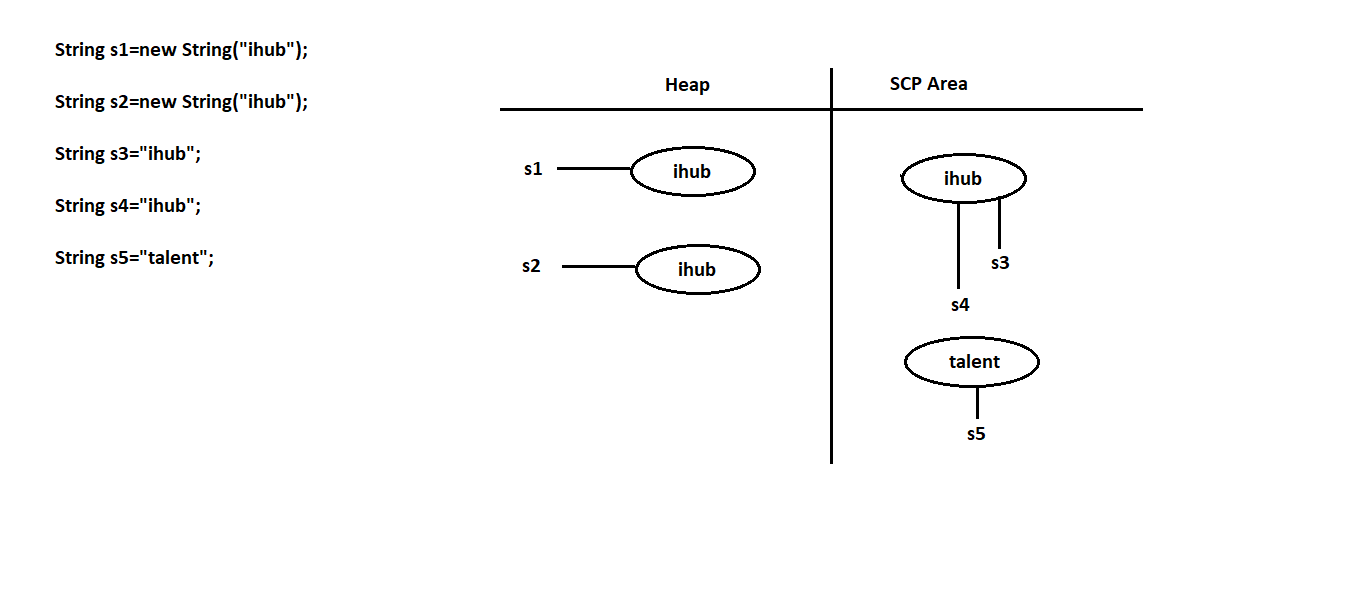


Object creation in SCP area is always optional. First JVM will check is there object is created with same content or not. If it is not created then JVM will create a new object. If it is created then JVM Won't create any new object. Hence there is no chance of having duplicate objects in SCP area.

SCP area objects do not have any object reference even though garbage collector can't access them.

SCP objects will destroy automatically when JVM shutdowns or terminated.

Diagram: class38.3

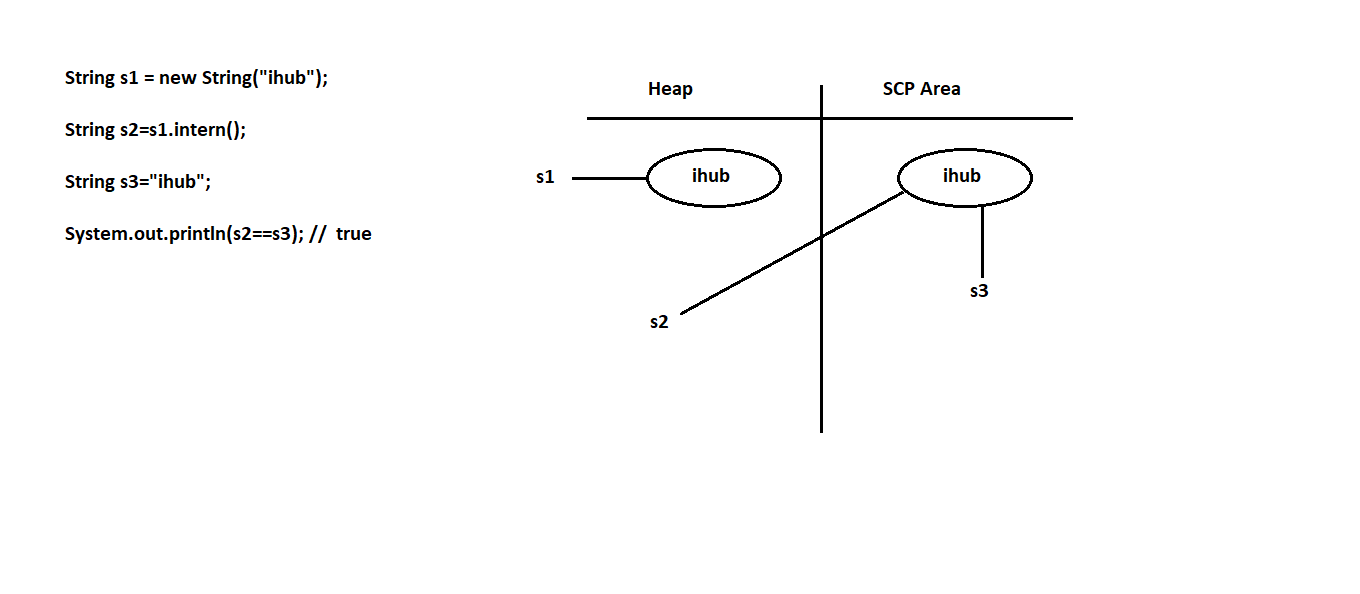


Interning of String object

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

With the help of heap object reference if we need corresponding SCP object reference then we need to use intern() method.

Diagram: class38.4



String important methods

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

Q1.) Write a java program to find out length of the string?

input:

hello

output:

5

ex:

class Test

{

public static void main(String[] args)

{

String str="hello";

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

}

}

Q2.) Write a java program to display string character by character?

input:

hello

output:

h

e

l

l

o

ex:

class Test

{

public static void main(String[] args)

{

String str="hello";

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

{

System.out.println(str.charAt(i));

}

}

}

Q3.) Write a java program to concatinate two strings?

input:

ihub

talent

output:

ihubtalent

ex:

---

class Test

{

public static void main(String[] args)

{

String str1="ihub";

String str2="talent";

System.out.println(str1.concat(str2));

}

}

Q4.) Write a java program to check both strings are equal or not?

input:

ihub

ihub

output:

Both are equals

ex:

class Test

{

public static void main(String[] args)

{

String str1="ihub";

String str2="ihub";

if(str1.equals(str2))

System.out.println("Both are equals");

else

System.out.println("Both are not equals");

}

}

Q5.) Write a java program to check both strings are equal or not?

input:

IHUB

ihub

output:

Both are equals

ex:

class Test

{

public static void main(String[] args)

{

String str1="IHUB";

String str2="ihub";

if(str1.equalsIgnoreCase(str2))

System.out.println("Both are equals");

else

System.out.println("Both are not equals");

}

}

Q6.) Write a java program to convert uppercase string to lowercase ?

input:

IHUBTALENT

output:

ihubtalent

ex:

class Test

{

public static void main(String[] args)

{

String str="IHUBTALENT";

str=str.toLowerCase();

System.out.println(str);

}

}

Q7.) Write a java program to convert lowercase string to uppercase ?

input:

ihubtalent

output:

IHUBTALENT

ex:

class Test

{

public static void main(String[] args)

{

String str="ihubtalent";

str=str.toUpperCase();

System.out.println(str);

}

}

Q8.) Write a java program to remove special characters from given string?

input:

I\_h@ub$Tale#nt1

output:

IhubTalent1

ex:

---

class Test

{

public static void main(String[] args)

{

String str="I\_h@ub$Tale#nt1";

str=str.replaceAll("[^A-Za-z0-9]","");

System.out.println(str);

}

}

Q9.) Write a java program to remove the spaces from string?

input:

I hub Tale nt

output:

IhubTalent

ex:

---

class Test

{

public static void main(String[] args)

{

String str="I hub Tale nt";

str=str.replaceAll("\\s","");

System.out.println(str);

}

}

Q10.) Write a java program to concatinate two strings?

input:

Ihub12

Talent18

output:

IhubTalent30

ex:

---

class Test

{

public static void main(String[] args)

{

String str1="Ihub12";

String str2="Talent18";

String word1=str1.replaceAll("[^A-Za-z]","");

int num1=Integer.parseInt(str1.replaceAll("[^0-9]",""));

String word2=str2.replaceAll("[^A-Za-z]","");

int num2=Integer.parseInt(str2.replaceAll("[^0-9]",""));

String word = word1 + word2;

int num = num1 + num2;

System.out.println(word+num);

}

}

Q11.) Write a java program to display the string in reverse order?

input:

hello

output:

olleh

class Test

{

public static void main(String[] args)

{

String str="hello";

for(int i=str.length()-1;i>=0;i--)

{

System.out.print(str.charAt(i));

}

}

}

Assignment

----------

Q12.) Write a java program to find out given string is palindrome or not?

input:

racar

output:

It is a palindrome string

Class 39:-

Q13.) Write a java program to display reverse of a string?

input:

hello

output:

olleh

ex:

---

class Test

{

public static void main(String[] args)

{

String str="hello";

char[] carr=str.toCharArray(); // h e l l o

//reverse variable

String rev="";

//reading reverse

for(int i=carr.length-1;i>=0;i--)

{

rev+=carr[i];

}

System.out.println(rev);

}

}

Q14.) Write a java program to check given string is palindrome or not?

input:

racar

output:

It is a palindrome string

ex:

---

class Test

{

public static void main(String[] args)

{

String str="racar";

char[] carr=str.toCharArray(); //r a c a r

//reverse variable

String rev="";

//reading reverse

for(int i=carr.length-1;i>=0;i--)

{

rev+=carr[i];

}

if(str.equals(rev))

System.out.println("It is a palindrome string");

else

System.out.println("It is not a palindrome string");

}

}

Q15.) Write a java program to display reverse of a sentence?

Input:

This is java class

output:

class java is This

ex:

class Test

{

public static void main(String[] args)

{

String str="This is java class";

String[] sarr=str.split(" "); // This is java class

//reading reverse

for(int i=sarr.length-1;i>=0;i--)

{

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

}

}

}

Q16.) Write a java program to display reverse of a string?

input:

This is java class

output:

sihT si avaj ssalc

ex:

---

class Test

{

public static void main(String[] args)

{

String str="This is java class";

String[] sarr=str.split(" "); // This is java class

//for each loop

for(String s:sarr)

{

char[] carr=s.toCharArray(); // T h i s

for(int i=carr.length-1;i>=0;i--)

{

System.out.print(carr[i]);

}

//space

System.out.print(" ");

}

}

}

Q17.) Write a java program to display the string starting with uppercase letter?

input:

This is Java class For students

output:

This Java For

ex:

class Test

{

public static void main(String[] args)

{

String str="This is Java class For students";

String[] sarr=str.split(" ");

//for each loop

for(String s:sarr)

{

char ch=s.charAt(0);

if(ch>='A' && ch<='Z')

{

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

}

}

}

}

Q18.) Write a java program to check given string is anagram or not?

input:

silent

listen

output:

It is a anagram string

import java.util.Arrays;

class Test

{

public static void main(String[] args)

{

String str1="silent";

String str2="listen";

char[] carr1=str1.toCharArray();

char[] carr2=str2.toCharArray();

Arrays.sort(carr1); // e i l n s t

Arrays.sort(carr2); // e i l n s t

boolean flag=true;

for(int i=0;i<carr1.length && i<carr2.length;i++)

{

if(carr1[i]!=carr2[i])

{

flag=false;

break;

}

}

if(flag==true)

System.out.println("It is a Anagram string");

else

System.out.println("It is not a Anagram string");

}

}

Q19.) Write a java program to perform right rotation of a given string?

input:

str = ihubtalent

cnt = 4

output:

talentihub

ex

---

class Test

{

public static void main(String[] args)

{

String str="ihubtalent";

int cnt=4;

String str1=str.substring(0,cnt);

String str2=str.substring(cnt,str.length());

str=str2+str1;

System.out.println(str);

}

}

Q20.) Write a java program to get below below output?

input:

str = Ihubtalentstudents

index = 10

word = for

output:

Ihubtalentforstudents

ex:

---

class Test

{

public static void main(String[] args)

{

String str="Ihubtalentstudents";

int index=10;

String word="for";

String word1=str.substring(0,index);

String word2=str.substring(index,str.length());

str = word1+word+word2;

System.out.println(str);

}

}

Q21.) Write a java program to display the string in a given format?

input:

XYZ

output:

XY

XZ

YX

YZ

ZX

ZY

ex:

class Test

{

public static void main(String[] args)

{

String str="XYZ";

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

{

for(int j=0;j<str.length();j++)

{

if(i!=j)

{

System.out.println(str.charAt(i)+""+str.charAt(j));

}

}

}

}

}

Assignment

===========

Q22.) Write a java program to display palindrome strings?

input:

racar is madam for java

output:

racar madam

Class 40:-

Q23.) Write a java program to display unique/distinct characters from given string?

input:

google

output:

gole

ex:

----

class Test

{

public static void main(String[] args)

{

String str="google";

String duplicates="";

String unique="";

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

{

String current=Character.toString(str.charAt(i));

if(unique.contains(current))

{

if(!duplicates.contains(current))

{

duplicates+=current;

continue;

}

}

unique+=current;

}

System.out.println(unique);

}

}

Q24.) Write a java program to display duplicate characters from given string?

input:

google

output:

og

ex:

---

class Test

{

public static void main(String[] args)

{

String str="google";

String duplicates="";

String unique="";

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

{

String current=Character.toString(str.charAt(i));

if(unique.contains(current))

{

if(!duplicates.contains(current))

{

duplicates+=current;

continue;

}

}

unique+=current;

}

System.out.println(duplicates);

}

}

Q25.) Write a java program to display most repeating character in a given string?

input:

ihubtalentinstitute

output:

t repeating for 5 times

ex:

---

class Test

{

public static void main(String[] args)

{

String str="ihubtalentinstitute";

char character=' ';

int maxCount=0;

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

{

int cnt=0;

for(int j=0;j<str.length();j++)

{

if(str.charAt(i) == str.charAt(j))

{

cnt++;

}

}

if(maxCount<cnt)

{

maxCount=cnt;

character=str.charAt(i);

}

}

System.out.println(character+" repeating for "+maxCount+" times");

}

}

Q26.) Write a java program to display the string in a given format?

input:

A1B2C3D4

output:

ABBCCCDDDD

ex:

class Test

{

public static void main(String[] args)

{

String str="A1B2C3D4";

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

{

if(Character.isAlphabetic(str.charAt(i)))

{

System.out.print(str.charAt(i));

}

else

{

int n=Character.getNumericValue(str.charAt(i));

for(int k=1;k<n;k++)

{

System.out.print(str.charAt(i-1));

}

}

}

}

}

Q27.) Write a java program to perform permutation of a given string?

input:

ABC

output:

ABC

ACB

BAC

BCA

CBA

CAB

ex:

---

class Test

{

public static void main(String[] args)

{

String str="ABC";

permutation(str.toCharArray(),0);

}

public static void permutation(char[] arr,int fi)

{

if(fi==arr.length-1)

{

System.out.println(arr);

return;

}

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

{

swapping(arr,fi,i);

permutation(arr,fi+1);

swapping(arr,fi,i);

}

}

public static void swapping(char[] arr,int fi,int i)

{

char temp=arr[fi];

arr[fi]=arr[i];

arr[i]=temp;

}

}

Q28.) Write a java program to encode the string?

input:

1106

output:

AAJF

class Test

{

public static void main(String[] args)

{

String str="1106";

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

{

int n=Character.getNumericValue(str.charAt(i));

if(n>0)

{

System.out.print((char)('A'+n-1));

}

else

{

int k=Integer.parseInt(str.substring(i-1,i+1));

System.out.print((char)('A'+k-1));

}

}

}

}

StringBuffer

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

If our content change frequently then it is never recommanded to use String because for every change a new object will be created.

To overcome this limitation Sun Micro System introduced StringBuffer concept.

In StringBuffer all the changes will reflect to single or same object only.

constructor

-----------

1) StringBuffer sb=new StringBuffer()

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

It will create empty StringBuffer object with default initial capacity of 16.

If capacity reaches to maximum capacity then new capacity will be created with below formulea.

ex:

new capacity = current\_capacity + 1 \* 2 ;

ex:

----

class Test

{

public static void main(String[] args)

{

StringBuffer sb=new StringBuffer();

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

sb.append("abcdefghijklmnop");

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

sb.append("qr");

System.out.println(sb.capacity()); // 16+1\*2=34

}

}

2) StringBuffer sb=new StringBuffer(int initialcapacity)

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

It will create StringBuffer object with specified initial capacity.

ex:

--

class Test

{

public static void main(String[] args)

{

StringBuffer sb=new StringBuffer(19);

System.out.println(sb.capacity()); // 19

}

}

3) StringBuffer sb=new StringBuffer(String s)

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

It will create StringBuffer object equivalent to String.

Here capacity will be created with below formulea.

ex:

capacity = s.length() + 16

ex:

---

class Test

{

public static void main(String[] args)

{

StringBuffer sb=new StringBuffer("ihub");

System.out.println(sb.capacity()); // 4 + 16 = 20

}

}

Q1.) Write a java program to display reverse of a string?

input:

hello

output:

olleh

ex:

---

class Test

{

public static void main(String[] args)

{

String str="hello";

StringBuffer sb=new StringBuffer(str);

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

System.out.println(rev);

}

}

Q2.) Write a java program to check given string is palindrome or not?

input:

racar

output:

It is a palindrome string

ex:

class Test

{

public static void main(String[] args)

{

String str="racar";

StringBuffer sb=new StringBuffer(str);

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

if(str.equals(rev))

System.out.println("It is a palindrome string");

else

System.out.println("It is not a palindrome string");

}

}

Assignment

===========

Q) Write a java program to count number of 2's in a given number?

input:

22

output:

6 (2,12,20,21,22)

Class : 41

========

Q3.) Write a java program to count number of 2's present in a given number?

input:

22

output:

6 (2,12,20,21,22)

ex:

---

class Test

{

public static void main(String[] args)

{

int num=22;

StringBuffer sb=new StringBuffer();

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

{

sb.append(i);

}

int cnt=0;

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

{

int n=Character.getNumericValue(sb.charAt(i));

if(n==2)

{

cnt++;

}

}

System.out.println(cnt);

}

}

Q4.) Write a java program to multiply two arrays?

input:

5 3 1

2 5

output:

13275 (531\*25)

ex:

---

class Test

{

public static void main(String[] args)

{

int[] arr1={5,3,1};

int[] arr2={2,5};

//caller method

int a =Integer.parseInt(arrayToString(arr1));

int b =Integer.parseInt(arrayToString(arr2));

System.out.println(a\*b);

}

//callie method

public static String arrayToString(int[] arr)

{

StringBuffer sb=new StringBuffer();

for(int i:arr)

{

sb.append(i);

}

return sb.toString();

}

}

Q5.) Write a java program to display the string in a given format?

input:

ABBCCCDDDD

output:

A1B2C3D4

ex:

---

class Test

{

public static void main(String[] args)

{

String str="ABBCCCDDDD";

StringBuffer sb=new StringBuffer();

int count=1;

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

{

if(i<str.length()-1 && str.charAt(i) == str.charAt(i+1))

{

count++;

}

else

{

sb.append(str.charAt(i)).append(count);

count=1;

}

}

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

}

}

StringBuilder

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

StringBuilder is exactly same as StringBuffer with following differences.

StringBuffer StringBuilder

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

Every method present in StringBuffer is No method present in StringBuilder is synchronized.

synchronized.

At a time only one thread is allowed to Multiple threads are allowed to operate on

operator on StringBuffer object. Hence it StringBuilder object. Hence it is not thread safe.

is thread safe.

Waiting time of a thread will increase There is no waiting threads relatively performance

relatively performance is low. is high.

It is introduced in 1.0v. It is introduced in 1.5v.

Note:-

-----

If our content not change frequently then it is recommanded to use String.

If our content change frequently where thread safety is required then we need to use StringBuffer.

If our content change frequently where thread safety is not required then we need to use StringBuilder.

StringTokenizer

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

StringTokenizer is a class which is present in java.util package.

It is used to tokenize the string irrespective of regular expression.

We can create StringTokenizer class object as follow.

ex:

StringTokenizer st=new StringTokenizer(String s,RegularExpression regex);

StringTokenizer class contains following five methods.

ex:

public int countTokens()

public boolean hasMoreTokens() // used to check if more words exists

public String nextToken() //given the words one by on

public boolean hasMoreElements()

public Object nextElement()

ex:

---

import java.util.StringTokenizer;

class Test

{

public static void main(String[] args)

{

StringTokenizer st=new StringTokenizer("this is java class"," ");

System.out.println(st.countTokens());//4

}

}

ex:

--

import java.util.StringTokenizer;

class Test

{

public static void main(String[] args)

{

StringTokenizer st=new StringTokenizer("this is java class"," ");

while(st.hasMoreTokens())

{

String s=st.nextToken();

System.out.println(s);

}

}

}

ex:

---

import java.util.StringTokenizer;

class Test

{

public static void main(String[] args)

{

StringTokenizer st=new StringTokenizer("this is java class"," ");

while(st.hasMoreElements())

{

String s=(String)st.nextElement();

System.out.println(s);

}

}

}

ex:

---

import java.util.StringTokenizer;

class Test

{

public static void main(String[] args)

{

StringTokenizer st=new StringTokenizer("9,99,999",",");

while(st.hasMoreElements())

{

String s=(String)st.nextElement();

System.out.println(s);

}

}

}

Q) What is garbage collection and in how many ways we can call garbage collector?

Garbage is a daemon thread which is used to destroy unused or useless objects from java.

There are two ways to call garbage collector in java.

1) System.gc()

2) Runtime.getRuntime().gc()

What is Garbage Collection ?

In java, garbage means unreferenced objects.

Garbage Collection is process of reclaiming the runtime unused memory automatically. In other words, it is a way to destroy the unused objects.

To do so, we were using free() function in C language and delete() in C++. But, in java it is performed automatically. So, java provides better memory management.

How to work Garbage Collection ?

Note: The Garbage collector of JVM collects only those objects that are created by new keyword.

So if you have created any object without new, you can use finalize method to perform cleanup processing (destroying remaining objects).

ex:

--

class Test

{

//instance variable

int i=10;

public static void main(String[] args)

{

Test t1=new Test();

System.out.println(t1.i);//10

t1=null;

//System.gc();

Runtime.getRuntime().gc();

}

public void finalize()

{

System.out.println("Garbage Collector called");

}

}

Exception Handling

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

Q) What is the difference between Exception and Error?

Exception

---------

Exception is a problem for which we can provide solution programmatically.

Exceptions raised due to syntax errors.

ex:

ArithmeticException

FileNotFoundException

IllegalArgumentException

Error

-----

Compile time error,runtime error(*Exception Handling always raises at run time*),logical error.

Error is a problem for which we can't provide solution programmatically.

Errors raised due to lack of system resources.

ex:

OutOfMemoryError

StackOverFlowError

LinkageError

As a part of java application development it is a responsibility of a programmer to provide smooth termination for every java program.

We have two types of terminations.

1) Smooth termination / Graceful termination

2) Abnormal termination

1) Smooth termination

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

During the program execution suppose if we are not getting any interruption in the middle of the program such type of termination is called smooth termination.

ex:

class Test

{

public static void main(String[] args)

{

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

}

}

2) Abnormal termination

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

During the program execution suppose if we are getting any interruption in the middle of the program such type of termination is called abnormal termination.

ex:

class Test

{

public static void main(String[] args)

{

System.out.println(10/0);

}

}

If any exception raised in our program, we must and should handle that exception otherwise our program will terminates abnormally.

Here exception will display name of the exception, description of the exception and line number of the exception.

Exception

=========

It is a unwanted, unexpected event which disturbs normal flow of a program.

Exceptions always raise at runtime so they are also known as runtime events.

The main objective of exception handling is to provide graceful termination.

In java , exception are divided into two types.

1) Predefined exceptions

2) Userdefined exceptions

1) Predefined exceptions

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

Built-In exceptions are called predefined exceptions.

It is categories into two types.

i) Checked exceptions

ii) Unchecked exceptions

Diagram; class41.1

i) Checked exceptions

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

Exceptions which are checked by the compiler at the time of compilation are called checked exceptions.

ex:

InterruptedException

EOFException

IOException

ii) Unchecked exceptions

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

Exceptions which are checked by the JVM at the time of runtime are called checked exceptions.

ex:

ArithmeticException

ClassCastException

IllegalArgumentException

Class :- 42

========

If any checked exception raised in our program we must and should handle that exception by using try and catch block.

try block

==========

It is a block which contains risky code.

A try block associate with catch block.

If any exception raise in try block then it won't be executed.

A try block is used to throw the exception to catch block.

catch block

===========

It is a block which contains error handling code.

A catch block always associate with try block.

A catch block is used to catch the exception from try block.

If there is no exception in try block then catch block won't be executed.

A catch block takes exception name as a parameter and that name must match with exception class name.

syntax:

-----

try

{

-

- //Risky Code

-

}

catch(ArithmeticException ae)

{

-

-

-

}

ex:

---

class Test

{

public static void main(String[] args)

{

try

{

System.out.println("try-block");

}

catch(Exception e)

{

System.out.println("catch-block");

}

}

}

ex:

---

class Test

{

public static void main(String[] args)

{

try

{

System.out.println(10/0);

}

catch(ArithmeticException ae)

{

System.out.println("catch-block");

}

}

}

ex:

---

class Test

{

public static void main(String[] args)

{

try

{

System.out.println("stmt1");

System.out.println(10/0);

System.out.println("stmt2");

}

catch(ArithmeticException ae)

{

System.out.println("catch-block");

}

}

}

ex:

---

class Test

{

public static void main(String[] args)

{

int i=10;

try

{

i++;

}

catch(Exception e)

{

i++;

}

System.out.println(i); // 11

}

}

A try with multiple catch blocks

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

A try block can have multiple catch blocks.

If a try block contains multiple catch blocks then order of catch block is very important it should be from child to parent but not from parent to child.

ex:

--

class Test

{

public static void main(String[] args)

{

try

{

System.out.println(10/0);

}

catch (ArithmeticException ae)

{

System.out.println("From AE");

}

catch (RuntimeException re)

{

System.out.println("From RE");

}

catch (Exception e)

{

System.out.println("From E");

}

}

}

Various methods methods to display exception details

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

Throwable class defines following methods to display exception details.

1) printStackTrace()

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

It is used to display name of the exception, description of the exception and line number of the exception.

2) toString()

-----------

It is used to display name of the exception and description of the exception.

3) getMessage()

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

It is used to display description of the exception.

ex:

----

class Test

{

public static void main(String[] args)

{

try

{

System.out.println(10/0);

}

catch(ArithmeticException ae)

{

ae.printStackTrace();

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

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

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

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

}

}

}

Q) How to handle multiple exceptions in a single catch block?

class Test

{

public static void main(String[] args)

{

try

{

System.out.println(10/0);

}

catch(IllegalArgumentException | ArithmeticException | ClassCastException e)

{

e.printStackTrace();

}

}

}

finally block

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

It is never recommanded to maintain cleanup code in try block because if any exception raised in try block then try block won't be executed.

It is never recommanded to maintain cleanup code in catch block because if there is no exception in try block then catch block won't be executed.

But we need a place where we can maintain cleanup code and it should execute irrespective of exception raised or not. Handle or not. Such block is called finally block.

syntax:

-----

try

{

-

- //Risky Code

-

}

catch(Exception e)

{

-

- //Error Handling Code

-

}

finally

{

-

- //Cleanup code

-

}

ex:

---

class Test

{

public static void main(String[] args)

{

try

{

System.out.println("try-block");

}

catch(Exception e)

{

e.printStackTrace();

}

finally

{

System.out.println("finally-block");

}

}

}

o/p:

try-block

finally-block

ex:

----

class Test

{

public static void main(String[] args)

{

try

{

System.out.println(10/0);

}

catch(Exception e)

{

e.printStackTrace();

}

finally

{

System.out.println("finally-block");

}

}

}

o/p:

java.lang.ArithmeticException: / by zero

at Test.main(Test.java:7)

finally-block

try and finally combination is valid in java.

ex:

class Test

{

public static void main(String[] args)

{

try

{

System.out.println("try-block");

}

finally

{

System.out.println("finally-block");

}

}

}

Q) What is the difference between final, finally and finalize method?

final

-------

It is a modifier which is applicable for variables, methods and classes.

If we declare any variable as final then reassignment of that variable is not possible.

If we declare any method as final then overriding of that method is not possible.

If we declare any class as final then creating child class is not possible.

finally

--------

It is a block which contains cleanup code and it will execute irrespective of exception raised or not.

finalize

----------

It is a method called by garbage collector just before destroying an object for cleanup activity.

throw statement

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

Sometimes we will create exception objects explicitly and handover to JVM manually by using throw statement.

syntax:

-------

throw new ArithmeticException("Don't divide by zero");

ex:

---

class Test

{

public static void main(String[] args)

{

System.out.println(10/0);

}

}

Here exception object is created and handover to JVM by main method.

ex:

---

class Test

{

public static void main(String[] args)

{

throw new ArithmeticException("Don't divide by zerooooo");

}

}

Here exception object is created explicitly and handover to JVM manually by using throw statement.

throws statement

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

If any checked exception raised in our program we must and should handle that exception by using try and catch block or by using throws statement.

ex:

---

class Test

{

public static void main(String[] args)

{

try

{

Thread.sleep(3000);

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

}

catch (InterruptedException ie)

{

ie.printStackTrace();

}

}

}

ex:

---

class Test

{

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

{

Thread.sleep(5000);

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

}

}

2) Userdefined exceptions

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

Exceptions which are created by the user based on the application requirement are called custom exceptions.

ex:

NoInterestInJavaException

NeedEnjoymentException

ACNotWorkingException

EligibleToVoteException

NotEligibleToVoteException

and etc.

ex

---

import java.util.Scanner;

class EligibleToVoteException extends RuntimeException

{

EligibleToVoteException(String msg)

{

super(msg);

}

}

class NotEligibleToVoteException extends RuntimeException

{

NotEligibleToVoteException(String msg)

{

super(msg);

}

}

class Test

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the age :");

int age=sc.nextInt();

if(age<18)

throw new NotEligibleToVoteException("Not Eligible to vote");

else

throw new EligibleToVoteException("Eligible to vote");

}

}

Class :- 43

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

java.io package

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

File

===

File f=new File("abc.txt");

File will check is there any abc.txt file already created or not.

If it is available it simply refers to that file.If it is not created then

it won't create any new file.

ex:

---

import java.io.\*;

class Test

{

public static void main(String[] args)

{

File f=new File("abc.txt");

System.out.println(f.exists());//false

}

}

A File object can be used to create a physical file.

ex:

import java.io.\*;

class Test

{

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

{

File f=new File("abc.txt");

System.out.println(f.exists());//false

f.createNewFile();

System.out.println(f.exists());//true

}

}

A File object can be used to create a directory also.

ex:

import java.io.\*;

class Test

{

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

{

File f=new File("bhaskar123");

System.out.println(f.exists());//false

f.mkdir();

System.out.println(f.exists());//true

}

}

Q)Write a java program to Create a "cricket123" folder and inside that folder create "abc.txt" file?

import java.io.\*;

class Test

{

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

{

File f1=new File("cricket123");

f1.mkdir();

File f2=new File("cricket123","abc.txt");

f2.createNewFile();

System.out.println("Please check the location");

}

}

FileWriter

==========

FileWriter is used to write character oriented data into a file.

constructor

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

FileWriter fw=new FileWriter(String s);

FileWriter fw=new FileWriter(File f);

ex:

FileWriter fw=new FileWriter("aaa.txt");

or

File f=new File("aaa.txt");

FileWriter fw=new FileWriter(f);

If file does not exist then FileWriter will create a physical file.

Methods

-----------

1)write(int ch)

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

It will insert single character into a file.

2)write(char[] ch)

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

It will insert array of characters into a file.

3)write(String s)

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

It will insert String into a file.

4)flush()

----------

It gives guaranttee that last character of a file is also inserted.

5)close()

-----------

It is used to close the FileWriter object.

ex:

-----

import java.io.\*;

class Test

{

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

{

FileWriter fw=new FileWriter("aaa.txt");

fw.write(98);// b

fw.write("\n");

char[] ch={'a','b','c'};

fw.write(ch);

fw.write("\n");

fw.write("bhaskar\nsolution");

fw.flush();

fw.close();

System.out.println("Please check the location");

}

}

FileReader

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

It is used to read character oriented data from a file.

constructor

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

FileReader fr=new FileReader(String s);

FileReader fr=new FileReader(File f);

ex:

FileReader fr=new FileReader("aaa.txt");

or

File f=new File("aaa.txt");

FileReader fr=new FileReader(f);

Methods

----------

1)read()

--------

It will read next character from a file and return unicode value.

If next character is not available then it will return -1.

2)read(char[] ch)

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

It will read collection of characters from a file.

3)close()

---------

It is used to close FileReader object.

ex:1

-------

import java.io.\*;

class Test

{

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

{

FileReader fr=new FileReader("aaa.txt");

int i=fr.read();

while(i!=-1)

{

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

i=fr.read();

}

fr.close();

}

}

ex:2

----------

import java.io.\*;

class Test

{

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

{

FileReader fr=new FileReader("aaa.txt");

char[] carr=new char[255];

//load the data from file to char array

fr.read(carr);

//reading the data from char array

for(char c:carr)

{

System.out.print(c);

}

fr.close();

}

}

Usage of FileWriter and FileReader is not recommanded to use

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

While inserting the data by using FileWriter ,we need to insert line

seperator(\n) which is very headache for the programmer.

While reading the data by using FileReader object ,we need to read character

by character which is not convenient to the programmer.

To overcome this limitation Sun micro system introduced BufferedWriter and BufferedReader.

BufferedWriter

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

It is used to insert character oriented data into a file.

constructor

-----------

BufferedWriter bw=new BufferedWriter(Writer w);

BufferedWriter bw=new BufferedWriter(Writer w,int buffersize);

BufferedWriter object does not communicate with files directly.

It will take the support of some writer objects.

ex:

FileWriter fw=new FileWriter("bbb.txt");

BufferedWriter bw=new BufferedWriter(fw);

or

BufferedWriter bw=new BufferedWriter(new FileWriter("bbb.txt"));

Methods

---------

1)write(int ch)

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

It will insert single character into a file.

2)write(char[] ch)

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

It will insert array of characters into a file.

3)write(String s)

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

It will insert String into a file.

4)flush()

----------

It gives guaranttee that last character of a file is also inserted.

5)close()

-----------

It is used to close the BufferedWriter object.

6)newLine()

----------

It will insert new line into a file.

ex:

import java.io.\*;

class Test

{

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

{

BufferedWriter bw=new BufferedWriter(new FileWriter("bbb.txt"));

bw.write(98);//b

bw.newLine();

char[] ch={'a','b','c'};

bw.write(ch);

bw.newLine();

bw.write("bhaskar");

bw.newLine();

bw.flush();

bw.close();

System.out.println("Please check the location");

}

}

BufferedReader

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

It is enhanced reader to read character oriented data from a file.

constructor

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

BufferedReader br=new BufferedReader(Reader r);

BufferedReader br=new BufferedReader(Reader r,int buffersize);

BufferedReader object can't communicate with files directly.IT will take

support of some reader objects.

ex:

FileReader fr=new FileReader("bbb.txt");

BufferedReader br=new BufferedReader(fr);

or

BufferedReader br=new BufferedReader(new FileReader("bbb.txt"));

The main advantage of BufferedReader over FileReader is we can read

character line by line instead of character by character.

methods

---------

1)read()

--------

It will read next character from a file and return unicode value.

If next character is not available then it will return -1.

2)read(char[] ch)

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

It will read collection of characters from a file.

3)close()

---------

It is used to close BufferedReader object.

4)nextLine()

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

It is used to read next line from the file.If next line is

not available then it will return null.

ex:

import java.io.\*;

class Test

{

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

{

BufferedReader br=new BufferedReader(new FileReader("bbb.txt"));

String line=br.readLine();

while(line!=null)

{

System.out.println(line);

line=br.readLine();

}

br.close();

}

}

PrintWriter

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

It is enhanced write to write character oriented data into a file.

constructor

-----------

PrintWriter pw=new PrintWriter(String s);

PrintWriter pw=new PrintWriter(File f);

PrintWriter pw=new PrintWriter(Writer w);

PrintWriter can communicate with files directly and it will take the support of some writer objects.

ex:

PrintWriter pw=new PrintWriter("ccc.txt");

or

PrintWriter pw=new PrintWriter(new File("ccc.txt"));

or

PrintWriter pw=new PrintWriter(new FileWriter("ccc.txt"));

The main advantage of PrintWriter over FileWriter and BufferedWriter is we can insert any type of data.

Assume if we want insert primitive values then PrintWriter is best choice.

methods

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

write(int ch)

write(char[] ch)

write(String s)

flush()

close()

writeln(int i)

writeln(float f)

writeln(double d)

writeln(String s)

writeln(char c)

writeln(boolean b)

write(int i)

write(float f)

write(double d)

write(String s)

write(char c)

write(boolean b)

ex:

------

import java.io.\*;

class Test

{

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

{

PrintWriter pw=new PrintWriter("ccc.txt");

pw.write(100);// d

pw.println(100);// 100

pw.print('a');

pw.println(true);

pw.println("hi");

pw.println(10.5d);

pw.flush();

pw.close();

System.out.println("Please check the location");

}

}

various ways to provide input values from the keyboard

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

There are various ways to provide input values from keyboard.

1)Command line argument

2)BufferedReader class

3)Console class

4)Scanner class

1)Command line argument

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

In command line argument we need to pass our inputs at runtime.

ex:

class Test

{

public static void main(String[] args)

{

String name=args[0];

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

}

}

o/p:

javac Test.java

java Test Alan

2)BufferedReader class

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

BufferedReader class present in java.io package.

BufferedReader class will take InputStreamReader object as a parameter

which is embedded with System.in.

ex:

BufferedReader br=

new BufferedReader

(new InputStreamReader(System.in));

To read input values from console we need to readLine() method.

ex:

import java.io.\*;

class Test

{

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

{

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

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

String name=br.readLine();

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

}

}

3)Console class

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

Console class present in java.io package.

We can create Console class object by using console() method of System class.

ex:

Console c=System.console();

To read inputs from console we need to use readLine() method.

ex:

import java.io.\*;

class Test

{

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

{

Console c=System.console();

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

String name=c.readLine();

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

}

}

4)Scanner class

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

Scanner class present java.util package.

We can create Scanner object class as follow.

ex:

Scanner sc=new Scanner(System.in);

We can read inputs from the console by using following methods.

ex:

next()

nextLine()

nextInt()

nextFloat()

nextDouble()

next().charAt(0);

and etc.

ex:

import java.util.\*;

class Test

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the No :");

int no=sc.nextInt();

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

String name=sc.next();

System.out.println("Enter the Fee :");

double fee=sc.nextDouble();

System.out.println(no+" "+name+" "+fee);

}

}

Class 44:-

Generics

==========

Arrays are typesafe. It means we can give guarantee that what type of elements are present in array.

If requirement is there to store String values then it is recommanded to use String[] array.

ex:

String[] sarr=new String[5];

sarr[0]="hi";

sarr[1]="hello";

sarr[2]=10;// C.T.E

At the time of retrieving the data from array we don't need to perform typecasting.

ex:

String[] sarr=new String[5];

sarr[0]="hi";

sarr[1]="hello";

-

-

String val=sarr[0];

Collections are not typesafe.We can't give guarantee that what type of elements are present in Collections.

If requirement is there to store String values then it is never recommanded to use ArrayList because we won't get any compile time error or runtime error but sometimes our program get failure.

ex:

ArrayList al=new ArrayList();

al.add("hi");

al.add("hello");

al.add(10);

At the time of retrieving the data compulsary we need to perform typecasting.

ex:

ArrayList al=new ArrayList();

al.add("hi");

al.add("hello");

al.add(10);

-

-

String s=(String)al.get(0);

To overcome this limitations Sun Micro System introduced Generics concept in 1.5v.

The main objective of generics are

1) To make Collections as typesafe.

2) To avoid typecasting problem.

java.util package

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

Q) What is the difference between Arrays and Collections?

Arrays Collections

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

It is a collection of homogeneous data It is a collection of homogeneous and hetrogeneous

elements. data elements.

Arrays are fixed in size. Collections are growable in nature.

Performance point of view arrays are Memory point of view Collections are recommanded

recommanded to use. to use.

Arrays not implemented based on data Collections are implemented based on data structure

Structure concept. concept.

Hence we can't expect any readymade concept. Hence we can expect readymade methods.

method.

It can hold primitive types and object types. It can hold only object types.

Collection Framework

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

It defines several interfaces and classes to represent group of objects in a single entity.

Collection

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

Collection is an interface which is present in java.util package.

It is a root interface for entire collection framework.

If we want to represent group of individual objects in a single entity then we need to use Collection.

Collection interface contains common methods which are applicable for entire Collection objects.

ex:

cmd> javap java.util.Collection

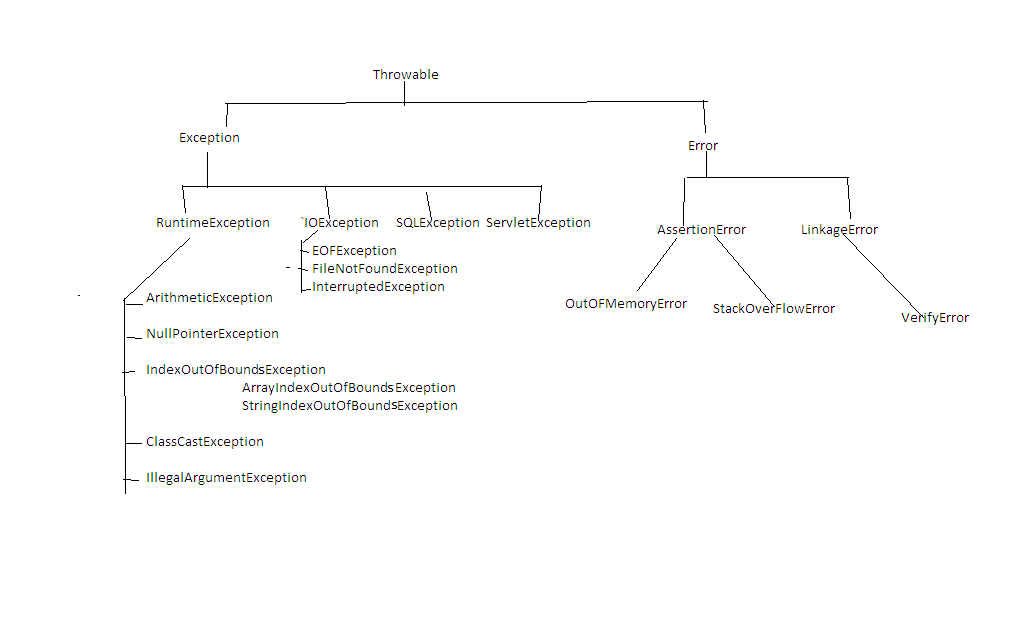
List

======

It is a child interface of Collection interface.

If we want to represent group of individual objects in a single entity where duplicate objects are allowed and order is preserved then we need to use List interface.

Diagram: class44.1



ArrayList

=========

The underlying data structure is resizable array or growable array.

Duplicate objects are allowed.

Insertion order is preserved.

Hetrogeneous objects are allowed.

Null insertion is possible.

It implements List, Serializable,Cloneable and RandomAccess interface.

ex:

---

import java.util.\*;

class Test

{

public static void main(String[] args)

{

ArrayList al=new ArrayList();

al.add("one");

al.add("two");

al.add("three");

System.out.println(al);//[one,two,three]

al.add("one");

System.out.println(al);//[one,two,three,one]

al.add(10);

System.out.println(al); //[one,two,three,one,10]

al.add(null);

System.out.println(al); //[one,two,three,one,10,null]

}

}

ex:

---

import java.util.\*;

class Test

{

public static void main(String[] args)

{

ArrayList al=new ArrayList();

al.add("one");

al.add("two");

al.add("three");

System.out.println(al);//[one,two,three]

System.out.println(al.isEmpty()); // false

System.out.println(al.contains("one")); // true

al.add(1,"gogo");

System.out.println(al); //[one,gogo,two,three]

al.remove("gogo");

System.out.println(al);//[one,two,three]

al.clear();

System.out.println(al); //[]

}

}

ex

----

import java.util.\*;

class Test

{

public static void main(String[] args)

{

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

al.add("one");

al.add("two");

al.add("three");

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

{

String s= al.get(i);

System.out.println(s);

}

}

}

ex:

---

import java.util.\*;

class Test

{

public static void main(String[] args)

{

List<String> list=new ArrayList<String>();

list.add("one");

list.add("two");

list.add("three");

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

{

String s= list.get(i);

System.out.println(s);

}

}

}

ex:

---

import java.util.\*;

class Test

{

public static void main(String[] args)

{

List<Integer> list=Arrays.asList(7,3,9,1,2,5);

System.out.println(list);

}

}

ex:

---

import java.util.\*;

class Test

{

public static void main(String[] args)

{

ArrayList<String> al1=new ArrayList<String>();

al1.add("HTML");

al1.add("CSS");

al1.add("JAVA");

al1.add("ORACLE");

al1.add("FRAMEWORK");

ArrayList<String> al2=new ArrayList<String>();

al2.add("HTML");

al2.add("CSS");

al2.add("DOTNET");

al2.add("SQLSERVER");

al2.add("FRAMEWORK");

al1.retainAll(al2);

System.out.println(al1);//[HTML, CSS, FRAMEWORK]

}

}

LinkedList

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

The underlying data structure is doubly LinkedList.

Duplicate objects are allowed.

Insertion order is preserved.

Hetrogeneous objects are allowed.

Null insertion is possible.

It implements List, Serializable,Cloneable and Deque interface.

LinkedList interface contains following methods.

ex:

public E getFirst();

public E getLast();

public E removeFirst();

public E removeLast();

public void addFirst(E);

public void addLast(E);

ex:

---

import java.util.\*;

class Test

{

public static void main(String[] args)

{

LinkedList ll=new LinkedList();

ll.add("one");

ll.add("two");

ll.add("three");

System.out.println(ll);//[one,two,three]

ll.add("one");

System.out.println(ll);//[one,two,three,one]

ll.add(10);

System.out.println(ll);//[one,two,three,one,10]

ll.add(null);

System.out.println(ll);//[one,two,three,one,10,null]

}

}

ex:

----

import java.util.\*;

class Test

{

public static void main(String[] args)

{

LinkedList ll=new LinkedList();

ll.add("one");

ll.add("two");

ll.add("three");

System.out.println(ll);//[one,two,three]

ll.addFirst("gogo");

ll.addLast("jojo");

System.out.println(ll);//[gogo,one,two,three,jojo]

System.out.println(ll.getFirst());

System.out.println(ll.getLast());

ll.removeFirst();

ll.removeLast();

System.out.println(ll);//[one,two,three]

}

}

ex:

---

import java.util.\*;

class Test

{

public static void main(String[] args)

{

LinkedList<String> ll1=new LinkedList<String>();

ll1.add("one");

ll1.add("two");

ll1.add("three");

System.out.println(ll1);//[one,two,three]

LinkedList<String> ll2=new LinkedList<String>();

ll2.add("raja");

System.out.println(ll2);//[raja]

ll2.addAll(ll1);

System.out.println(ll2);//[raja,one,two,three]

System.out.println(ll2.containsAll(ll1)); // true

ll2.removeAll(ll1);

System.out.println(ll2);//[raja]

}

}

Vector

========

The underlying data structure is resizable array or growable array.

Duplicate objects are allowed.

Insertion order is preserved.

Hetrogeneous objects are allowed.

Null insertion is possible.

It implements List, Serializable,Cloneable and RandomAccess interface.

All methods present in Vector are synchronized.

Vector class contains following methods.

ex:

addElement()

removeElementAt()

removeAllElements()

firstElement()

lastElement()

and etc.

ex:

---

import java.util.\*;

class Test

{

public static void main(String[] args)

{

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

System.out.println(v.capacity()); //10

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

{

v.addElement(i);

}

System.out.println(v); //[1,2,3,4,5,6,7,8,9,10]

System.out.println(v.firstElement());//1

System.out.println(v.lastElement());//10

v.removeElementAt(5);

System.out.println(v);//[1, 2, 3, 4, 5, 7, 8, 9, 10]

v.removeAllElements();

System.out.println(v); //[]

}

}

ex:

---

import java.util.\*;

class Test

{

public static void main(String[] args)

{

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

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

{

v.add(i);

}

System.out.println(v); //[1,2,3,4,5,6,7,8,9,10]

System.out.println(v.get(0));//1

System.out.println(v.get(v.size()-1));//10

v.remove(5);

System.out.println(v);//[1, 2, 3, 4, 5, 7, 8, 9, 10]

v.clear();

System.out.println(v); //[]

}

}

Class 45:-

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

Q) What is the difference between ArrayList and LinkedList?

ArrayList LinkedList

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

The underlying data structure is resizable The underlying data structure is doubly linkedlist.

array or growable array.

ArrayList is better for storing and accessing LinkedList is better for manipulating the data.

the data.

Memory location for ArrayList elements are Memory location for LinkedList elements are not

contigeous. contigeous.

When ArrayList is initialized, a default There is no case of default capacity in LinkedList.

capacity 10 is assigned to it.

Q) What is the difference between ArrayList and Vector?

ArrayList Vector

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

No method is synchronized. All methods are synchronized.

At a time only one thread is allowed to operate Multiple threads are allowed to operator

ArrayList object.Hence it is thread safe. Vector object.Hence it is not thread safe.

Relatively performance is low because waiting time Relatively performance is high because there

of a thread is increasing. is no waiting threads.

It is a non-legacy class. It is a legacy class.

It is introduced in 1.2v. It is introduced in 1.0v.

Q) What is the difference between Collection and Collections?

Collection Collections

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

It is a root interface for entire collection It is a utility class.

framework.

It is used to represent group of individual It defines several utility methods that are operator

objects in a single entity. on Collection.

It contains abstract methods, default methods It contains only static methods.

and static methods.

Q) Write a java program to sort the string?

input:

dog apple boy cat

output:

apple boy cat dog

import java.util.\*;

class Test

{

public static void main(String[] args)

{

String str="dog apple boy cat";

String[] sarr=str.split(" ");

List<String> list=Arrays.asList(sarr);

Collections.sort(list);

list.forEach(element -> System.out.print(element+" "));

}

}

Q) How to make ArrayList as synchronized?

import java.util.\*;

class Test

{

public static void main(String[] args)

{

List<String> list=new ArrayList<String>();

List<String> synchronizedList = Collections.synchronizedList(list);

synchronizedList.add("one");

synchronizedList.add("two");

synchronizedList.add("three");

synchronizedList.forEach(element -> System.out.print(element+" "));

}

}

Stack

=======

It is a child class of Vector class.

If we depend upon Last In First Out order then we need to use Stack.

constructor

-----------

Stack s=new Stack();

methods

--------

1) push(E)

---------

It is used to push the element to stack.

2) pop()

----------

It is used to pop toppest element from stack.

3) peek()

--------

It will return toppest element of stack.

4) isEmpty()

---------

It will check stack is empty or not.

5) search(Object o)

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

It will return offset value if element is found otherwise it will return -1.

ex:

----

import java.util.\*;

class Test

{

public static void main(String[] args)

{

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

s.push("A");

s.push("B");

s.push("C");

System.out.println(s);//[A,B,C]

s.pop();

System.out.println(s);//[A,B]

System.out.println(s.peek());// B

System.out.println(s.isEmpty()); // false

System.out.println(s.search("Z")); // -1

System.out.println(s.search("A")); // 2

}

}

Q) Write a java program to check given string is balanced or not?

input:

{[()]}

output:

It is a balanced string

import java.util.\*;

class Test

{

public static void main(String[] args)

{

String str="{[()]}";

//caller method

if(isBalanced(str))

System.out.println("It is balanced string");

else

System.out.println("It is not balanced string");

}

//callie method

public static boolean isBalanced(String str)

{

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

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

{

char ch=str.charAt(i);

if(ch=='{' || ch=='[' || ch=='(')

{

stack.push(ch);

}

else if(ch==')' && !stack.isEmpty() && stack.peek()=='(')

{

stack.pop();

}

else if(ch==']' && !stack.isEmpty() && stack.peek()=='[')

{

stack.pop();

}

else if(ch=='}' && !stack.isEmpty() && stack.peek()=='{')

{

stack.pop();

}

else

{

return false;

}

}

return stack.isEmpty();

}

}

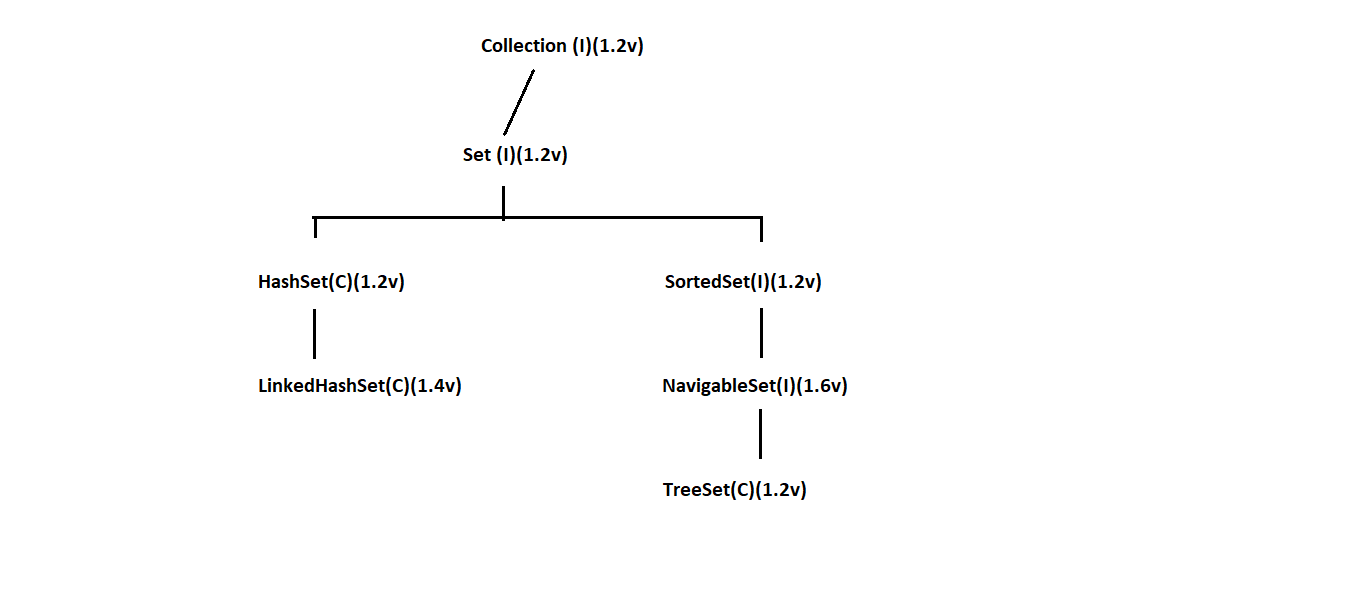
Set

======

It is a child interface of Collection interface.

If we want to represent group of individual objects in a single entity where duplicate objects are not allowed and order is not preserved then we need to use Set interface.

Diagram: class45.1



HashSet

=======

The underlying data structure is Hashtable.

Duplicate objects are not allowed.

Insertion order is not preserved because it will take hash code of an object.

Hetrogeneous objects are allowed.

Null insertion is possible.

ex:

import java.util.\*;

class Test

{

public static void main(String[] args)

{

HashSet hs=new HashSet();

hs.add("one");

hs.add("six");

hs.add("nine");

System.out.println(hs);//[nine, six, one]

hs.add("one");

System.out.println(hs);//[nine, six, one]

hs.add(10);

System.out.println(hs);//[nine, six, one, 10]

hs.add(null);

System.out.println(hs);//[null, nine, six, one, 10]

}

}

LinkedHashSet

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

It is a child class of HashSet class.

LinkedHashSet is exactly same as HashSet class with following differences.

HashSet LinkedHashSet

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

The underlying data structure is Hashtable. The underlying data structure is Hashtable and

LinkedList.

Insertion order is not preserved. Insertion order is preserved.

It is introduced in 1.2v. It is introduced in 1.4v.

ex:

import java.util.\*;

class Test

{

public static void main(String[] args)

{

LinkedHashSet lhs=new LinkedHashSet();

lhs.add("one");

lhs.add("six");

lhs.add("nine");

System.out.println(lhs);//[one, six, nine]

lhs.add("one");

System.out.println(lhs);//[one, six, nine]

lhs.add(10);

System.out.println(lhs);//[one, six, nine, 10]

lhs.add(null);

System.out.println(lhs);//[one, six, nine, 10, null]

}

}

Class 46:-

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

Q) Write a java program to display distinct elements from given array?

input:

1 2 2 3 3 3 4 4 4 4

output:

1 2 3 4

ex:

---

import java.util.\*;

class Test

{

public static void main(String[] args)

{

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

Set<Integer> set = new LinkedHashSet<Integer>();

//for each loop

for(int i:arr)

{

set.add(i);

}

set.forEach(element -> System.out.print(element+" "));

}

}

TreeSet

=======

The underlying datastructure is Balanced Tree.

Duplicate objects are not allowed.

Insertion order is not preserved because it will take sorting order.

Hetrogeneous objects are not allowed. If we try to insert hetrogeneous objects then we will get ClassCastException.

Null insertion is not possible. If we insert null then we will get NullPointerException.

ex:

---

import java.util.\*;

class Test

{

public static void main(String[] args)

{

TreeSet ts=new TreeSet();

ts.add(10);

ts.add(1);

ts.add(5);

ts.add(7);

System.out.println(ts); // [1,5,7,10]

ts.add(1);

System.out.println(ts); // [1,5,7,10]

//ts.add("one");

//System.out.println(ts); // R.E ClassCastException

//ts.add(null);

//System.out.println(ts); // R.E NullPointerException

}

}

Q) What is the difference between Comparable and Comparator interface?

Comparable

----------

Comparable is an interface which is present in java.lang package.

Comparable interface contains only one method i.e compareTo() method.

If we depend upon default natural sorting order then we need to use Comparable interface.

ex:

obj1.compareTo(obj2)

It returns -ve if obj1 comes before obj2

It returns +ve if obj1 comes after obj2

it returns 0 if both objects are same

ex:

---

class Test

{

public static void main(String[] args)

{

System.out.println("A".compareTo("Z")); // -25

System.out.println("Z".compareTo("A")); // 25

System.out.println("K".compareTo("K")); // 0

}

}

Comparator

-----------

Comparator is an interface which is present in java.util package.

Comparator interface contains following two methods i.e compare() and equals() method.

If we depend upon customized sorting order then we need to use Comparator interface.

ex:

public int compare(Object obj1,Object obj2)

It returns +ve if obj1 comes before obj2

It returns -ve if obj1 comes after obj2

it returns 0 if both objects are same

Implementation of equals() method is optional because it is present in Object class which is available to the class through inheritance.

Implementation of compare() method is mandatory.

ex:

---

import java.util.\*;

class Test

{

public static void main(String[] args)

{

TreeSet<Integer> ts=new TreeSet<Integer>(new MyComparator());

ts.add(10);

ts.add(1);

ts.add(5);

ts.add(7);

System.out.println(ts);//[10, 7, 5, 1]

}

}

class MyComparator implements Comparator

{

public int compare(Object obj1,Object obj2)

{

Integer i1=(Integer)obj1;

Integer i2=(Integer)obj2;

if(i1<i2)

return 1;

else if(i1>i2)

return -1;

else

return 0;

}

}

ex:

---

import java.util.\*;

class Test

{

public static void main(String[] args)

{

TreeSet<Integer> ts=new TreeSet<Integer>(new MyComparator());

ts.add(10);

ts.add(1);

ts.add(5);

ts.add(7);

System.out.println(ts);//[1, 5, 7, 10]

}

}

class MyComparator implements Comparator

{

public int compare(Object obj1,Object obj2)

{

Integer i1=(Integer)obj1;

Integer i2=(Integer)obj2;

if(i1<i2)

return -1;

else if(i1>i2)

return 1;

else

return 0;

}

}

Q) Write a java program to compare two dates?

import java.time.\*;

class Test

{

public static void main(String[] args)

{

LocalDate date1=LocalDate.now();

LocalDate date2=LocalDate.of(2024,10,15);

if(date1.compareTo(date2)>0)

System.out.println("date1 is biggest");

else if(date1.compareTo(date2)<0)

System.out.println("date2 is biggest");

else

System.out.println("Both are same");

}

}

Map

====

It is not a child interface of Collection interface.

If we want to represent group of individual objects in key and value pair then we need to use

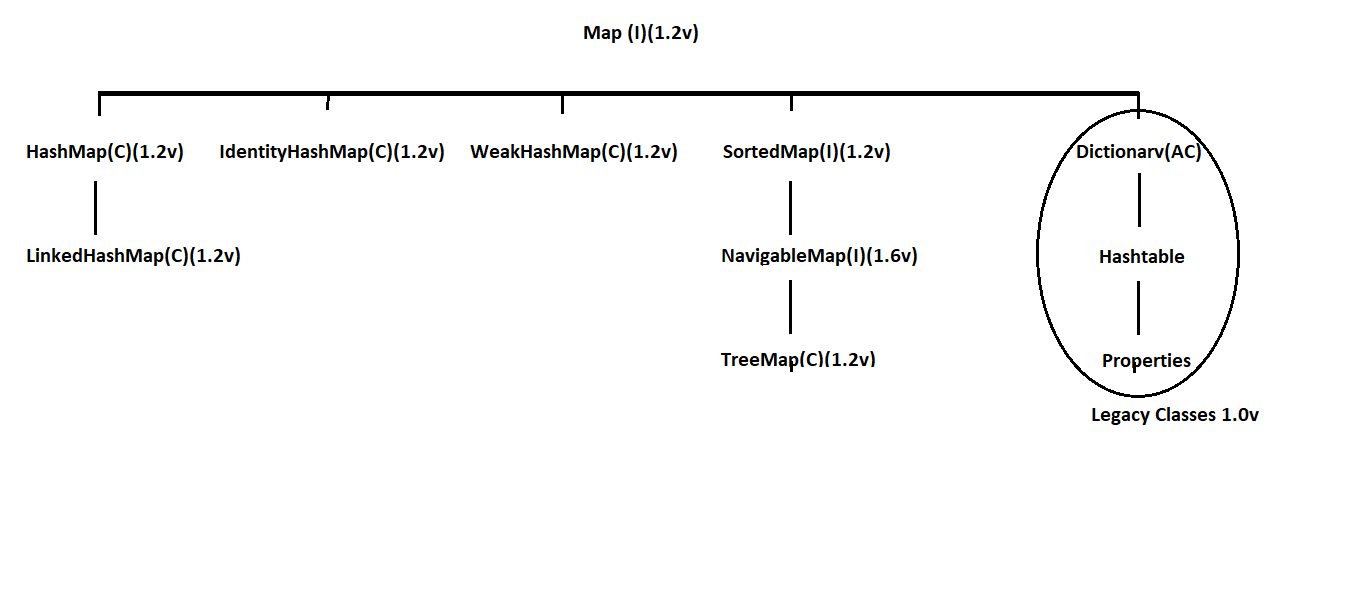
Map interface.

Key and value both must be objects.

Key can't be duplicate but value can be duplicate.

Each key and value pair is called single/one entry.

Diagram: class46.1



HashMap

=========

The underlying data structure is Hashtable.

Duplicate key is not allowed but value can be duplicate.

Insertion order is not preserved because it will take hash code of a key.

Hetrogeneous objects are allowed for both key and value.

Null insertion is possible for both key and value.

ex:

--

import java.util.\*;

class Test

{

public static void main(String[] args)

{

HashMap hm=new HashMap();

hm.put("one","raja");

hm.put("six","jose");

hm.put("nine","brook");

hm.put("five","alan");

System.out.println(hm);//{nine=brook, six=jose, one=raja, five=alan}

hm.put("one","gogo");

System.out.println(hm);//{nine=brook, six=jose, one=gogo, five=alan}

hm.put(10,100);

System.out.println(hm);//{nine=brook, six=jose, one=gogo, 10=100, five=alan}

hm.put(null,null);

System.out.println(hm);//{null=null, nine=brook, six=jose, one=gogo, 10=100, five=alan}

}

}

LinkedHashMap

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

LinkedHashMap is a child class of HashMap class.

LinkedHashMap is exactly same as HashMap class with following differences.

HashMap LinkedHashMap

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

The underlying data structure is Hashtable. The underlying data structure is Hashtable and

LinkedList.

Insertion order is not preserved. Insertion order is preserved.

It is introduced in 1.2v. It is introduced in 1.4v.

import java.util.\*;

class Test

{

public static void main(String[] args)

{

LinkedHashMap lhm=new LinkedHashMap();

lhm.put("one","raja");

lhm.put("six","jose");

lhm.put("nine","brook");

lhm.put("five","alan");

System.out.println(lhm);//{one=raja, six=jose, nine=brook, five="alan"}

lhm.put("one","gogo");

System.out.println(lhm);//{one=gogo, six=jose, nine=brook, five="alan"}

lhm.put(10,100);

System.out.println(lhm);//{one=gogo, six=jose, nine=brook, five="alan", 10=100}

lhm.put(null,null);

System.out.println(lhm);//{one=gogo, six=jose, nine=brook, five="alan", 10=100, null=null}

}

}

ex:

---

import java.util.\*;

class Test

{

public static void main(String[] args)

{

Map<String,String> map=new LinkedHashMap<String,String>();

map.put("one","raja");

map.put("two","ravi");

map.put("three","ramana");

Set s=map.keySet();

System.out.println(s);

Collection c=map.values();

System.out.println(c);

Set s1=map.entrySet();

System.out.println(s1);

}

}

Class 47:-

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

TreeMap

========

The underlying data structure is RED BLACK TREE.

Duplicate key is not allowed but value can be duplicate.

Insertion order is not preseved because it takes sorting order of key.

If we depend upon default natural sorting order then key must be homogeneous and Comparable.

If we depend upon customized sorting order then key must be hetrogeneous and Non-Comparable.

Key can't be null but value can be null.

ex:

---

import java.util.\*;

class Test

{

public static void main(String[] args)

{

TreeMap<Integer,String> tm=new TreeMap<Integer,String>();

tm.put(10,"ten");

tm.put(1,"one");

tm.put(5,"five");

tm.put(7,"seven");

System.out.println(tm); // {1=one, 5=five, 7=seven, 10=ten}

tm.put(1,"gogo");

System.out.println(tm);//{1=gogo, 5=five, 7=seven, 10=ten}

tm.put(4,null);

System.out.println(tm); //{1=gogo, 4=null, 5=five, 7=seven, 10=ten}

tm.put(null,"four");

System.out.println(tm); //R.E NullPointerException

}

}

Hashtable

=========

The underlying data structure is Hashtable.

Duplicate key is not allowed but value can be duplicate.

Insertion order is not preserved because it takes descending order of key.

Hetrogeneous objects are allowed for both key and value.

Null insertion is not possible for both key and value.

ex:

----

import java.util.\*;

class Test

{

public static void main(String[] args)

{

Hashtable ht=new Hashtable();

ht.put(1,"one");

ht.put(10,"ten");

ht.put(5,"five");

ht.put(3,"three");

System.out.println(ht);//{10=ten, 5=five, 3=three, 1=one}

ht.put(1,"gogo");

System.out.println(ht);//{10=ten, 5=five, 3=three, 1=gogo}

ht.put("four",4);

System.out.println(ht);//{10=ten, 5=five, four=4, 3=three, 1=gogo}

//ht.put(6,null);

//System.out.println(ht); //R.E NullPointerException

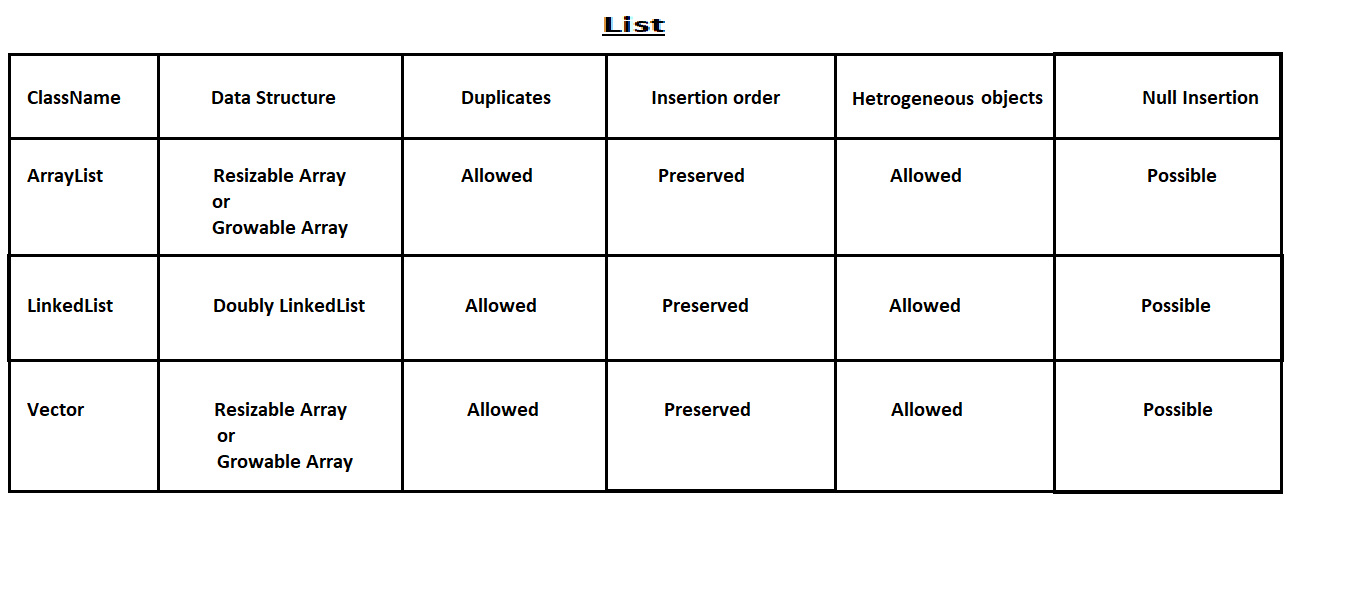
//ht.put(null,"six");

//System.out.println(ht); // R.E NullPointerException

}

}

Diagram: class47.1



Q) Write a java program to display number of occurance of a given string?

input:

This is is java class class

output:

This=1 is=2 java=1 class=2

ex:

import java.util.\*;

class Test

{

public static void main(String[] args)

{

String str="This is is java class class";

String[] sarr=str.split(" ");

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

for(String s:sarr)

{

if(map.get(s)!=null)

{

map.put(s,map.get(s)+1);

}

else

{

map.put(s,1);

}

}

map.forEach((key,value)-> System.out.print(key+"="+value+" "));

}

}

Q) Write a java program to display number of occurance of a given string?

input:

java

output:

j=1 a=2 v=1

ex:

import java.util.\*;

class Test

{

public static void main(String[] args)

{

String str="java";

char[] carr=str.toCharArray();

Map<Character,Integer> map=new LinkedHashMap<Character,Integer>();

for(char c:carr)

{

if(map.get(c)!=null)

{

map.put(c,map.get(c)+1);

}

else

{

map.put(c,1);

}

}

map.forEach((key,value)-> System.out.print(key+"="+value+" "));

}

}

Q) Write a java program to display lucky number from given array?

input:

1 1 2 3 3 3 4 4

output:

3

ex:

---

import java.util.\*;

class Test

{

public static void main(String[] args)

{

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

System.out.println(luckyNumber(arr));

}

//callie method

public static int luckyNumber(int[] arr)

{

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

for(int i:arr)

{

if(map.containsKey(i))

{

map.put(i,map.get(i)+1);

}

else

{

map.put(i,1);

}

}

int x=0;

int max=-1;

for(Map.Entry<Integer,Integer> entry:map.entrySet())

{

if(entry.getKey() == entry.getValue())

{

x = entry.getKey();

max = Math.max(x,max);

}

}

return max;

}

}

Types of Cursors in java

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

Cursors are used to read the objects one by one from Collections.

We have three types of cursors.

1) Enumeration

2) Iterator

3) ListIterator

1) Enumeration

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

It is used to read the objects one by one from legacy Collection objects.

We can create Enumeration object as follow.

ex:

Enumeration e=v.elements();

Enumeration interface contains two methods.

ex:

public boolean hasMoreElements();

public Object nextElement();

ex:

---

import java.util.\*;

class Test

{

public static void main(String[] args)

{

Vector v=new Vector();

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

{

v.add(i);

}

System.out.println(v); //[1,2,3,4,5,6,7,8,9,10]

Enumeration e=v.elements();

while(e.hasMoreElements())

{

Integer i=(Integer)e.nextElement();

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

}

}

}

Limitations with Enumeration

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

Enumeration is used to read objects one by one from legacy Collections objects only.Hence it is a not a universal cursor.

Using Enumeration we can perform read operation but not remove operation.

To overcome this limitation Sun Micro System introduced Iterator.

2) Iterator

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

Iterator is used to read objects one by one from any Collection object.Hence it is a universal cursor.

Using Iterator we can perform read and remove operations.

We can create Iterator object as follow.

ex:

Iterator itr=al.iterator();

Iterator interface contains following three methods.

ex:

public boolean hasNext();

public Object next();

public void remove();

ex:

---

import java.util.\*;

class Test

{

public static void main(String[] args)

{

ArrayList al=new ArrayList();

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

{

al.add(i);

}

System.out.println(al);//[1,2,3,4,5,6,7,8,9,10]

Iterator itr=al.iterator();

while(itr.hasNext())

{

Integer i=(Integer)itr.next();

if(i%2==0)

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

else

itr.remove();

}

System.out.println(al);//[2,4,6,8,10]

}

}

Limitations with Iterator

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

Using Enumeration and Iterator we can read objects only in forward direction but not in backward direction.Hence they are not bi-directional cursors.

Using Iterator we can perform read and remove operations but not adding and replacement of new object.

To overcome this limitation Sun Micro System introduced ListIterator.

3) ListIterator

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

It is a child interface of Iterator interface.

ListIterator is used to read objects only from List Collection objects.

Using Listiterator we can perform read , remove, adding and replacement of new objects.

We can create ListIterator object as follow.

ex:

ListIterator litr=al.listIterator();

ListIterator interface contains following methods.

ex:

public boolean hasNext()

public Object next()

public void remove()

public boolean hasPrevious()

public Object previous()

public int nextIndex()

public int previousIndex()

public void set(Object o)

public void add(Object o)

ex:

---

import java.util.\*;

class Test

{

public static void main(String[] args)

{

ArrayList al=new ArrayList();

al.add("bala");

al.add("venki");

al.add("chiru");

al.add("nag");

ListIterator litr=al.listIterator();

while(litr.hasNext())

{

String s=(String)litr.next();

System.out.println(s);

}

}

}

ex:

---

import java.util.\*;

class Test

{

public static void main(String[] args)

{

ArrayList al=new ArrayList();

al.add("bala");

al.add("venki");

al.add("chiru");

al.add("nag");

ListIterator litr=al.listIterator();

while(litr.hasNext())

{

String s=(String)litr.next();

if(s.equals("bala"))

{

litr.remove();

}

}

System.out.println(al); //[venki,chiru,nag]

}

}

ex:

---

import java.util.\*;

class Test

{

public static void main(String[] args)

{

ArrayList al=new ArrayList();

al.add("bala");

al.add("venki");

al.add("chiru");

al.add("nag");

ListIterator litr=al.listIterator();

while(litr.hasNext())

{

String s=(String)litr.next();

if(s.equals("bala"))

{

litr.add("charan");

}

}

System.out.println(al); //[bala, charan, venki, chiru, nag]

}

}

ex:

---

import java.util.\*;

class Test

{

public static void main(String[] args)

{

ArrayList al=new ArrayList();

al.add("bala");

al.add("venki");

al.add("chiru");

al.add("nag");

ListIterator litr=al.listIterator();

while(litr.hasNext())

{

String s=(String)litr.next();

if(s.equals("bala"))

{

litr.set("charan");

}

}

System.out.println(al); //[charan, venki, chiru, nag]

}

}

Diagram: class47.2

Class 48:-

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

Multithreading

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

Q) What is the difference between Thread and Process?

Thread

------

It is a leight weight sub process.

We can run multiple threads concurently.

One thread can communicate with another thread.

Process

-------

It is a collection of threads.

We can run multiple process concurently.

One process can't communicate with another process.

Multitasking

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

Executing several task simultenously such concept is called multitasking.

We have two types of multitasking.

1) Thread based multitasking

2) Process based multitasking

1) Thread based multitasking

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

Executing several task simultenously where each task is a same part of a program.

It is best suitable for programmatic level.

2) Process based multitasking

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

Executing several task simultenously where each task is a independent process.

It is best suitable for OS level.

MultiThreading

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

Executing several threads simultenously such concept is called multithreading.

In multithreading only 10% of work should be done by a programmer and 90% of work will be done by JAVA API.

The main important application area of multithreading are.

1) To implements multi media graphics.

2) To develop video games.

3) To develop animations.

Ways to start a thread in java

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

There are two ways to start a thread in java.

1) By extending Thread class

2) By implementing Runnable interface

1) By extending Thread class

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

class MyThread extends Thread

{

public void run()

{

for(int i=1;i<=5;i++)

{

System.out.println("Child-Thread");

}

}

}

class Test

{

public static void main(String[] args)

{

//instantitate a thread

MyThread t=new MyThread();

//start a thread

t.start();

for(int i=1;i<=5;i++)

{

System.out.println("Parent-Thread");

}

}

}

case1: Thread Schedular

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

If multiple threads are waiting for execution which thread has to be executed will decided by thread schedular.

What algorithm, behaviour, mechanism used by thread schedular is depends upon JVM vendor.

Hence we can't expect any execution order or exact output in multithreading.

case2: Difference between t.start() method and t.run() method

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

If we invoke t.start() method then a new thread will be created which is responsible to execute run() method automatically.

class MyThread extends Thread

{

public void run()

{

for(int i=1;i<=5;i++)

{

System.out.println("Child-Thread");

}

}

}

class Test

{

public static void main(String[] args)

{

//instantitate a thread

MyThread t=new MyThread();

//start a thread

t.start();

for(int i=1;i<=5;i++)

{

System.out.println("Parent-Thread");

}

}

}

If we invoke t.run() method then no new thread will be created but run() method will execute just like normal method.

ex:

class MyThread extends Thread

{

public void run()

{

for(int i=1;i<=5;i++)

{

System.out.println("Child-Thread");

}

}

}

class Test

{

public static void main(String[] args)

{

//instantitate a thread

MyThread t=new MyThread();

//no new thread

t.run();

for(int i=1;i<=5;i++)

{

System.out.println("Parent-Thread");

}

}

}

case3: If we won't override run() method

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

If we won't override run() method then Thread class run() method will execute automatically.

Thread class run() method is empty implementation.Hence we won't get any output from child thread.

ex:

---

class MyThread extends Thread

{

}

class Test

{

public static void main(String[] args)

{

//instantitate a thread

MyThread t=new MyThread();

//start a thread

t.start();

for(int i=1;i<=5;i++)

{

System.out.println("Parent-Thread");

}

}

}

case4: If we overload run() method

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

If we overload run() method then Thread class start() method always execute run() method with zero arguments only.

ex:

---

class MyThread extends Thread

{

public void run(int i)

{

System.out.println("int-arg method");

}

public void run()

{

System.out.println("0-arg method");

}

}

class Test

{

public static void main(String[] args)

{

//instantitate a thread

MyThread t=new MyThread();

//start a thread

t.start();

for(int i=1;i<=5;i++)

{

System.out.println("Parent-Thread");

}

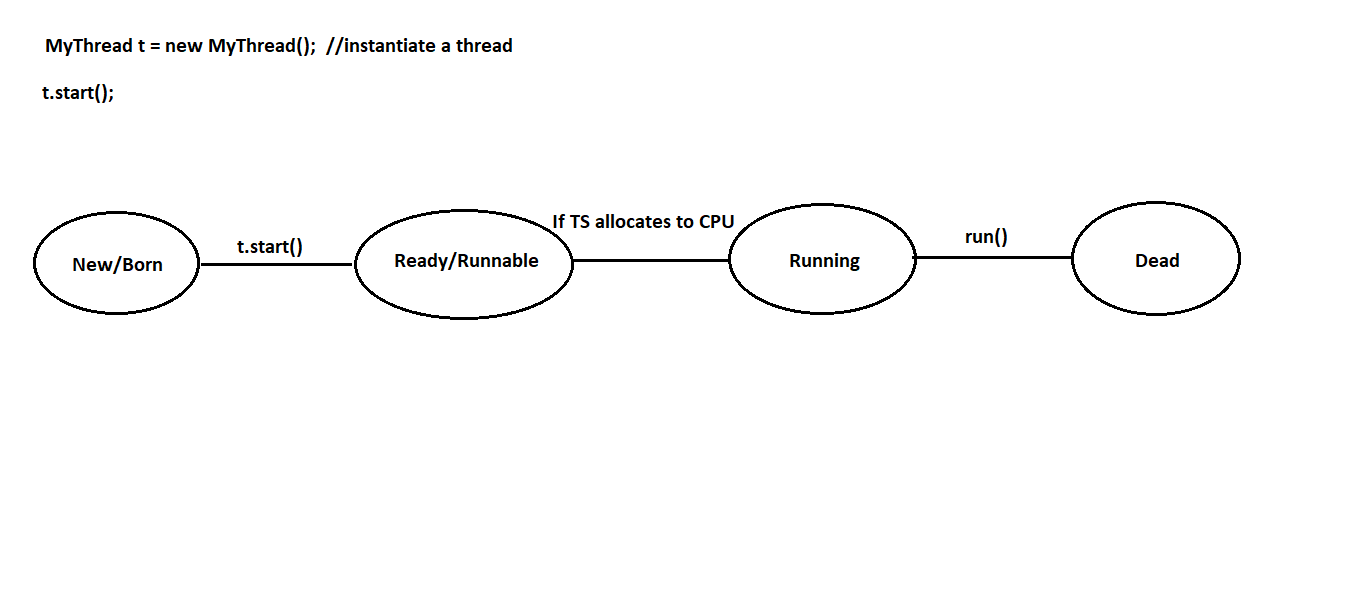
}

}

case 5: Life cycle of a thread

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

Diagram: class48.1



Once if we create thread then our thread will be in new or born state.

Once if we call t.start() method our thread goes to ready/runnable state.

If thread schedular allocates to CPU then our thread enters to running state.

Once the run() method execution is completed then our thread goes to dead state.

2) By implementing Runnable interface

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

class MyRunnable implements Runnable

{

public void run()

{

for(int i=1;i<=5;i++)

{

System.out.println("Child-Thread");

}

}

}

class Test

{

public static void main(String[] args)

{

MyRunnable r=new MyRunnable();

Thread t=new Thread(r); // r is a targatable interface

t.start();

for(int i=1;i<=5;i++)

{

System.out.println("Parent-Thread");

}

}

}

Setting and getting name of a thread

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

In java, every thread has a name. Explicitly provided by the programmer or automatically generated by the JVM.

We have following methods to set and get name of a thread.

ex:

public final void setName(String name);

public final String getName();

ex:

---

class MyThread extends Thread

{

}

class Test

{

public static void main(String[] args)

{

System.out.println(Thread.currentThread().getName());// main

MyThread t=new MyThread();

System.out.println(t.getName());//Thread-0

Thread.currentThread().setName("Parent-Thread");

System.out.println(Thread.currentThread().getName());//Parent-Thread

t.setName("Child-Thread");

System.out.println(t.getName());//Child-Thread

}

}

Thread priority

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

In java, every thread has a priority explicitly provided by the programmer and automatically generated by JVM.

The valid range of thread priority is 1 to 10. Where 1 is a least priority and 10 is a highest priority.

If we take more then 10 priority then we will get IllegalArgumentException.

Thread class defines following standard constants as thread priority.

ex:

Thread.MAX\_PRIORITY - 10

Thread.NORM\_PRIORITY - 5

Thread.MIN\_PRIORITY - 1

We don't have such constants like LOW\_PRIORITY and HIGH\_PRIORITY.

A thread which is having highest priority will be executed first.

If multiple threads having same priority then we can't expect any execution order.

Thread schedular uses thread priorities while allocating to CPU.

We have following methods to set and get thread priority.

ex:

public final void setPriority(int priority)

public final int getPriority()

ex

---

class MyThread extends Thread

{

}

class Test

{

public static void main(String[] args)

{

System.out.println(Thread.currentThread().getPriority());// 5

MyThread t=new MyThread();

System.out.println(t.getPriority());//5

Thread.currentThread().setPriority(10);

System.out.println(Thread.currentThread().getPriority());//10

t.setPriority(4);

System.out.println(t.getPriority());//4

//t.setPriority(11);//R.E IllegalArgumentException

}

}

Daemon Thread

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

Daemon thread is a service provider thread which provides services to user threads.

Life of daemon thread is depends upon user threads because when user threads died then daemon thread will die automatically.

There are many daemon thread are running internally like Garbage collector , finalizer and etc.

We can start a daemon thread by using setDeamon(true) method.

To check a thread is a daemon or not we will use isDaemon() method.

ex:

---

class MyThread extends Thread

{

public void run()

{

for(int i=1;i<=5;i++)

{

System.out.println(Thread.currentThread().isDaemon());

System.out.println("Child-Thread");

}

}

}

class Test

{

public static void main(String[] args)

{

MyThread t=new MyThread();

t.setDaemon(true);

t.start();

for(int i=1;i<=5;i++)

{

System.out.println("Parent-Thread");

}

}

}

Assignment

===========

class Employee

{

private int empId;

private String empName;

private double empSal;

//parameterized constructor

//setter and getter methods

}

interface IEmployee

{

public List<Employee> getAllEmployees();

}

class EmployeeImpl implements IEmployee

{

public List<Employee> getAllEmployees()

{

List<Employee> list=new ArrayList<Employee>();

list.add(new Employee(101,'raja',1000d));

list.add(new Employee(102,'ravi',2000d));

list.add(new Employee(103,'ramana',3000d));

return list;

}

}

Class 49:-

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

Various ways to prevent a thread from execution

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

There are three ways to prevent(stop) a thread from execution.

1) yield()

2) join()

3) sleep()

1) yield()

-----------

It pause the current execution thread and gives the chance to other threads having same

priority.

If there is no waiting threads or low priority threads then same thread will continue

it's execution.

If multiple waiting threads having same priority then we can't expect any execution order.

ex:

public static native void yield();

Diagram: class49.1

class MyThread extends Thread

{

public void run()

{

for(int i=1;i<=5;i++)

{

Thread.currentThread().yield();

System.out.println("Child-Thread");

}

}

}

class Test

{

public static void main(String[] args)

{

MyThread t=new MyThread();

t.start();

for(int i=1;i<=5;i++)

{

System.out.println("Parent-Thread");

}

}

}

2) join()

----------

If a thread wants to wait untill the completion of some other threads then we need to

join().

A join() method throws one checked exception called InterruptedException so we must

and should handle that exception by using try and catch block or by using throws stmt.

ex:

public final void join()throws InterruptedException

public final void join(long ms)throws InterruptedException

public final void join(long ms,int ns)throws InterruptedException

Diagram: class49.2

class MyThread extends Thread

{

public void run()

{

for(int i=1;i<=5;i++)

{

System.out.println("Child-Thread");

}

}

}

class Test

{

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

{

MyThread t=new MyThread();

t.start();

t.join();

for(int i=1;i<=5;i++)

{

System.out.println("Parent-Thread");

}

}

}

3) sleep()

-----------

If a thread don't want to perform any operation on perticular amount of time then we

need to use sleep() method.

A sleep() method throws one checked exception so we must and should handle that exception

by using try and catch block or by using throws statement.

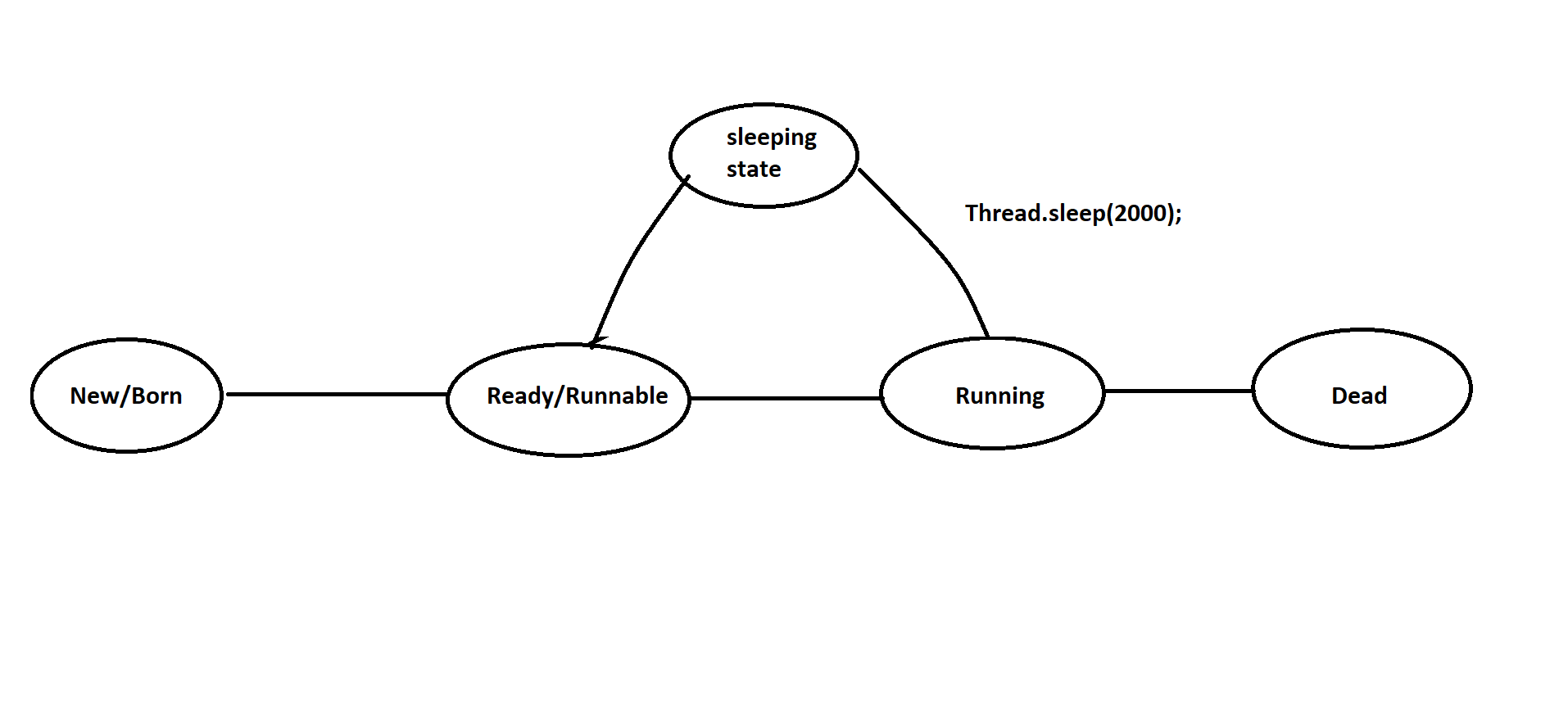
ex:

public static native void sleep()throws InterruptedException

public static native void sleep(long ms)throws InterruptedException

public static native void sleep(long ms,int ns)throws InterruptedException

Diagram: class49.3



ex:

---

class MyThread extends Thread

{

public void run()

{

for(int i=1;i<=5;i++)

{

System.out.println("Child-Thread");

try

{

Thread.sleep(2000);

}

catch (InterruptedException ie)

{

ie.printStackTrace();

}

}

}

}

class Test

{

public static void main(String[] args)

{

MyThread t=new MyThread();

t.start();

for(int i=1;i<=5;i++)

{

System.out.println("Parent-Thread");

}

}

}

Problem without synchronization

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

If there is no sychronization then we will face following problems.

1) Data inconsistency

2) Thread interference

ex:

---

class Table

{

void printTable(int n)

{

for(int i=1;i<=5;i++)

{

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

try

{

Thread.sleep(2000);

}

catch (InterruptedException ie)

{

ie.printStackTrace();

}

}

}

}

class MyThread1 extends Thread

{

Table t;

MyThread1(Table t)

{

this.t=t;

}

public void run()

{

t.printTable(5);

}

}

class MyThread2 extends Thread

{

Table t;

MyThread2(Table t)

{

this.t=t;

}

public void run()

{

t.printTable(10);

}

}

class Test

{

public static void main(String[] args)

{

Table obj=new Table();

MyThread1 t1=new MyThread1(obj);

MyThread2 t2=new MyThread2(obj);

t1.start();

t2.start();

}

}

synchronization

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

A synchronized keyword is applicable for methods and blocks.

A synchronization is allowed one thread to execute given object. Hence we achieve thread safety.

The main advantage of synchronization is we solve data inconsistence problem.

The main disadvantage of synchronization is ,it will increase waiting time of a thread which reduce the performance of the system.

If there is no specific requirement then it is never recommanded to use synchronization concept.

synchronization internally uses lock mechanism.

Whenever a thread wants to access object , first it has to acquire lock of an object and thread will release the lock when it completes it's task.

When a thread wants to execute synchronized method.It automatically gets the lock of an object.

When one thread is executing synchronized method then other threads are not allowed to execute other synchronized methods in a same object concurently.But other threads are allowed to execute non-synchronized method concurently.

ex:

class Table

{

synchronized void printTable(int n)

{

for(int i=1;i<=5;i++)

{

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

try

{

Thread.sleep(2000);

}

catch (InterruptedException ie)

{

ie.printStackTrace();

}

}

}

}

class MyThread1 extends Thread

{

Table t;

MyThread1(Table t)

{

this.t=t;

}

public void run()

{

t.printTable(5);

}

}

class MyThread2 extends Thread

{

Table t;

MyThread2(Table t)

{

this.t=t;

}

public void run()

{

t.printTable(10);

}

}

class Test

{

public static void main(String[] args)

{

Table obj=new Table();

MyThread1 t1=new MyThread1(obj);

MyThread2 t2=new MyThread2(obj);

t1.start();

t2.start();

}

}

synchronized block

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

If we want to perform synchronization on specific resource of a program then we need to use

synchronization.

ex:

If we have 100 lines of code and if we want to perform synchronization only for

10 lines then we need to use synchronized block.

If we keep all the logic in synchronized block then it will act as a synchronized method.

ex:

class Table

{

void printTable(int n)

{

synchronized(this)

{

for(int i=1;i<=5;i++)

{

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

try

{

Thread.sleep(2000);

}

catch (InterruptedException ie)

{

ie.printStackTrace();

}

}

}//sync

}

}

class MyThread1 extends Thread

{

Table t;

MyThread1(Table t)

{

this.t=t;

}

public void run()

{

t.printTable(5);

}

}

class MyThread2 extends Thread

{

Table t;

MyThread2(Table t)

{

this.t=t;

}

public void run()

{

t.printTable(10);

}

}

class Test

{

public static void main(String[] args)

{

Table obj=new Table();

MyThread1 t1=new MyThread1(obj);

MyThread2 t2=new MyThread2(obj);

t1.start();

t2.start();

}

}

3)Static synchronization

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

In static synchronization the lock will be on class but not on object.

If we declare any static method as synchronized then it is called static synchronization method.

ex:

class Table

{

static synchronized void printTable(int n)

{

for(int i=1;i<=5;i++)

{

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

try

{

Thread.sleep(2000);

}

catch (InterruptedException ie)

{

ie.printStackTrace();

}

}

}

}

class MyThread1 extends Thread

{

public void run()

{

Table.printTable(5);

}

}

class MyThread2 extends Thread

{

public void run()

{

Table.printTable(10);

}

}

class Test

{

public static void main(String[] args)

{

MyThread1 t1=new MyThread1();

MyThread2 t2=new MyThread2();

t1.start();

t2.start();

}

}

.Inter-Thread Communication

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

Two threads can communicate with one another by using wait(),notify() and notifyAll() method.

The Thread which is expecting updations it has to wait() method and the thread which is performing updations it has to call notify() method.

wait(),notify() and notifyAll() method present in Object class but not in Thread class.

To call wait(),notify() and notifyAll() method our current thread must be in a synchronized area otherwise we will get IllegalMonitorStateException.

Once a thread calls wait() method on a given object ,1st it will release the lock of that object immediately and entered into waiting state.

Once a thread calls notify() and notifyAll() method on a given object.It will release the lock of that object but not immediately.

Except wait(),notify() and notifyAll() method ,there is no such concept where lock release can happen.

ex:

class MyThread extends Thread

{

int total=0;

public void run()

{

synchronized(this)

{

System.out.println("Child Thread started calculation");

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

{

total=total+i;

}

System.out.println("Child thread giving notification");

this.notify();

}

}

}

class Test

{

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

{

MyThread t=new MyThread();

t.start();

synchronized(t)

{

System.out.println("Main Thread waiting for updating");

t.wait();

System.out.println("Main -Thread got notification ");

System.out.println(t.total);

}

}

}

DeadLock in java

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

DeadLock will occur in a suitation when one thread is waiting to access

object lock which is acquired by another thread and that thread is waiting

to access object lock which is acquired by first thread.

Here both the threads are waiting release the thread but no body will

release such situation is called DeadLock.

ex:

class Test

{

public static void main(String[] args)

{

final String res1="hi";

final String res2="bye";

Thread t1=new Thread()

{

public void run()

{

synchronized(res1)

{

System.out.println("Thread1: Locking Resource 1");

synchronized(res2)

{

System.out.println("Thread1: Locking Resource2");

}

}

}

};

Thread t2=new Thread()

{

public void run()

{

synchronized(res2)

{

System.out.println("Thread2: Locking Resource 2");

synchronized(res1)

{

System.out.println("Thread1: Locking Resource 1");

}

}

}

};

t1.start();

t2.start();

}

}

Drawbacks of multithreading

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

1)DeadLock

2)Thread Starvation

Class 50:-

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

Java 8 Features

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

Functional Interface

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

Interface which contains only one abstract method is called functional interface.

It can have any number of default methods and static methods.

It is also known as SAM interface or Single Abstract Method interface.

The main objective of functional interface is to achieve functional programming.

ex:

a = f1()

{

}

f1(f2(){})

{

}

@FunctionalInterface annotation is used to declare functional interface and it is optional.

ex:

---

@FunctionalInterface

interface A

{

public abstract void m1();

}

class B implements A

{

public void m1()

{

System.out.println("M1-Method");

}

}

class Test

{

public static void main(String[] args)

{

A a=new B();

a.m1();

}

}

ex:

---

@FunctionalInterface

interface A

{

public abstract void m1();

}

class Test

{

public static void main(String[] args)

{

A a=new A()

{

public void m1()

{

System.out.println("From M1 Method");

}

};

a.m1();

}

}

Lamda Expression

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

Lamda expression introduced in Java 8.

It is used to concise the code.

We can use lamda expression when we have functional interface.

Lamda expression consider as method.

The main objective of lamda expression is to achieve functional programming.

Lamda expression does not allow name, returntype and modifier.

ex:

Java method

-----------

public void m1()

{

System.out.println("M1 Method");

}

Lamda Expression

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

()->

{

System.out.println("M1 Method");

};

ex:

---

@FunctionalInterface

interface A

{

public abstract void m1();

}

class Test

{

public static void main(String[] args)

{

A a=()->

{

System.out.println("M1 Method");

};

a.m1();

}

}

ex:

---

@FunctionalInterface

interface A

{

public abstract void m1(int i,int j);

}

class Test

{

public static void main(String[] args)

{

A a=(int i,int j)->

{

System.out.println(10+20);

};

a.m1(10,20);

}

}

ex:

---

@FunctionalInterface

interface A

{

public abstract int m1(int i,int j);

}

class Test

{

public static void main(String[] args)

{

A a=(int i,int j)->

{

return i+j;

};

System.out.println(a.m1(100,200));

}

}

default methods in interface

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

Java provides facility to declare default methods in interface.

If we declare any method in interface and tagged with default keyword is called default method.

It is a non-abstract method.

It can be override.

ex:

---

interface A

{

//abstract method

public abstract void m1();

//default method

default void m2()

{

System.out.println("M2 Method");

}

}

class B implements A

{

public void m1()

{

System.out.println("M1 Method");

}

}

class Test

{

public static void main(String[] args)

{

A a=new B();

a.m1();

a.m2();

}

}

ex:

--

interface A

{

//abstract method

public abstract void m1();

//default method

default void m2()

{

System.out.println("M2 Method");

}

}

class B implements A

{

public void m1()

{

System.out.println("M1 Method");

}

public void m2()

{

System.out.println("M2 Method Override");

}

}

class Test

{

public static void main(String[] args)

{

A a=new B();

a.m1();

a.m2();

}

}

Using default methods of an interface we can achieve multiple inheritance in java.

ex:

---

interface Right

{

default void m1()

{

System.out.println("Right-M1 Method");

}

}

interface Left

{

default void m1()

{

System.out.println("Left-M1 Method");

}

}

class Middle implements Right,Left

{

public void m1()

{

System.out.println("Middle-M1 Method");

}

}

class Test

{

public static void main(String[] args)

{

Middle m=new Middle();

m.m1();

}

}

ex:

--

interface Right

{

default void m1()

{

System.out.println("Right-M1 Method");

}

}

interface Left

{

default void m1()

{

System.out.println("Left-M1 Method");

}

}

class Middle implements Right,Left

{

public void m1()

{

Right.super.m1();

}

}

class Test

{

public static void main(String[] args)

{

Middle m=new Middle();

m.m1();

}

}

ex:

---

interface Right

{

default void m1()

{

System.out.println("Right-M1 Method");

}

}

interface Left

{

default void m1()

{

System.out.println("Left-M1 Method");

}

}

class Middle implements Right,Left

{

public void m1()

{

Left.super.m1();

}

}

class Test

{

public static void main(String[] args)

{

Middle m=new Middle();

m.m1();

}

}

static methods in interface

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

Java provides facility to declare static methods in interface.

If we declare any method in interface and tagged with static keyword is called static method.

It is a non-abstract method.

It can't be override.

ex:

---

interface A

{

static void m1()

{

System.out.println("M1 Method");

}

}

class Test

{

public static void main(String[] args)

{

A.m1();

}

}

Stream API

===========

Stream API introduced in Java 8.

It is used to perform bulk operations on Collections.

If we want to process the objects from Collections we need to use Stream API.

It is present in java.util.stream package.

ex:

---

import java.util.\*;

import java.util.stream.\*;

class Test

{

public static void main(String[] args)

{

List<Integer> list=Arrays.asList(7,8,1,2,5,9,4);

List<Integer> newList=list.stream().filter(i->i%2==0).collect(Collectors.toList());

System.out.println(newList);

}

}

ex:

---

import java.util.\*;

import java.util.stream.\*;

class Test

{

public static void main(String[] args)

{

List<Integer> list=Arrays.asList(7,8,1,2,5,9,4);

long count=list.stream().filter(i->i%2!=0).count();

System.out.println(count);

}

}

ex:

---

import java.util.\*;

import java.util.stream.\*;

class Test

{

public static void main(String[] args)

{

List<Integer> list=Arrays.asList(7,8,1,2,5,9,4);

List<Integer> newList=list.stream().sorted().collect(Collectors.toList());

System.out.println(newList);

}

}

ex:

---

import java.util.\*;

import java.util.stream.\*;

class Test

{

public static void main(String[] args)

{

List<Integer> list=Arrays.asList(7,8,1,2,5,9,4);

List<Integer> newList=list.stream().sorted(Comparator.reverseOrder()).collect(Collectors.toList());

System.out.println(newList);

}

}

ex:

---

import java.util.\*;

import java.util.stream.\*;

class Test

{

public static void main(String[] args)

{

List<Integer> list=Arrays.asList(7,8,1,2,5,9,4);

long minimum=list.stream().min((i1,i2)->i1.compareTo(i2)).get();

System.out.println(minimum);

}

}

ex:

---

import java.util.\*;

import java.util.stream.\*;

class Test

{

public static void main(String[] args)

{

List<Integer> list=Arrays.asList(7,8,1,2,5,9,4);

long maximum=list.stream().max((i1,i2)->i1.compareTo(i2)).get();

System.out.println(maximum);

}

}

ex:

---

import java.util.\*;

import java.util.stream.\*;

class Test

{

public static void main(String[] args)

{

List<Integer> list=Arrays.asList(7,7,8,4,1,2,5,5,9,4);

List<Integer> newList=list.stream().distinct().collect(Collectors.toList());

System.out.println(newList);

}

}

forEach() method

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

A forEach() method introduced in Java 8.

It is used to iterate the objects from Collections.

ex:

---

import java.util.\*;

class Test

{

public static void main(String[] args)

{

List<Integer> list=Arrays.asList(7,8,4,1,2,5,9);

list.forEach(element -> System.out.print(element+" "));

}

}

ex:

---

import java.util.\*;

class Test

{

public static void main(String[] args)

{

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

map.put(1,"one");

map.put(2,"two");

map.put(3,"three");

map.forEach((key,value)-> System.out.print(key+"="+value+" "));

}

}

Method Reference(::)

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

Method reference introduced in java 8.

Method reference is used to refer method of functional interface.

Method reference is a special type of lambda expression.

ex:

---

import java.util.\*;

class Test

{

public static void main(String[] args)

{

List<Integer> list=Arrays.asList(7,8,4,1,2,5,9);

list.forEach(System.out::println);

}

}

Interview Question

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

Q) Write a java program to display employee information based on sorting order of employee id?

import java.util.\*;

import java.util.stream.\*;

class Employee

{

private int empId;

private String empName;

private double empSal;

//parameterized constructor

public Employee(int empId,String empName,double empSal)

{

this.empId=empId;

this.empName=empName;

this.empSal=empSal;

}

//getter methods

public int getEmpId()

{

return empId;

}

public String getEmpName()

{

return empName;

}

public double getEmpSal()

{

return empSal;

}

}

class Test

{

public static void main(String[] args)

{

List<Employee> list=new ArrayList<Employee>();

list.add(new Employee(104,"Lisa",4000d));

list.add(new Employee(101,"Alan",1000d));

list.add(new Employee(102,"Ben",2000d));

list.add(new Employee(103,"Jessi",3000d));

List<Employee> newList=list.stream().sorted(Comparator.comparingInt(Employee::getEmpId)).collect(Collectors.toList());

newList.forEach(employee -> System.out.println(employee.getEmpId()+" "+employee.getEmpName()+" "+employee.getEmpSal()));

}

}

Q) Write a java program to display employee information based on sorting order of employee name?

import java.util.\*;

import java.util.stream.\*;

class Employee

{

private int empId;

private String empName;

private double empSal;

//parameterized constructor

public Employee(int empId,String empName,double empSal)

{

this.empId=empId;

this.empName=empName;

this.empSal=empSal;

}

//getter methods

public int getEmpId()

{

return empId;

}

public String getEmpName()

{

return empName;

}

public double getEmpSal()

{

return empSal;

}

}

class Test

{

public static void main(String[] args)

{

List<Employee> list=new ArrayList<Employee>();

list.add(new Employee(104,"Lisa",4000d));

list.add(new Employee(101,"Alan",1000d));

list.add(new Employee(102,"Ben",2000d));

list.add(new Employee(103,"Jessi",3000d));

List<Employee> newList=list.stream().sorted(Comparator.comparing(Employee::getEmpName)).collect(Collectors.toList());

newList.forEach(employee -> System.out.println(employee.getEmpId()+" "+employee.getEmpName()+" "+employee.getEmpSal()));

}

}

Assignment

==========

Q) Write a java program to display duplicate and unique elements from given array?

input:

1 1 2 3 3 4 5 5 6

output:

duplicate elements : 1 3 5

unique elements : 2 4 6