04/01/2018-Task

* **Can abstract class have constructors in Java?**

Yes, abstract class has constructors in java. We can access those constructors by instantiating subclasses of the abstract class.

* **Can abstract class implements interface in Java? Do they require to implement all methods?**

Yes, abstract class can implements interface in java. It is not necessary to implements all the methods in the interface. But the subclass of abstract class should give implementation for all the methods.

* **Can abstract class be final in Java?**

No, abstract class won’t be final. Marking the class "final" means that no other class can extend it and hence abstract classes cannot be marked final.

* **Can abstract class have static methods in Java?**

We cannot have abstract static method like this

abstract class AbstractClassExample

{

abstract static void method();

}

But we can have,

Abstract class AbstractClassExample

{

Static void method(){

}

}

* **Can you create instance of abstract class?**

No, we can’t create instance for abstract class.

* **Is it necessary for abstract class to have abstract method?**

No, it is not necessary we can create abstract class without abstract methods.

* **Difference between abstract class and interface in Java?**

|  |  |
| --- | --- |
| INTERFACE | ABSTRACT CLASS |
| 1.It supports multiple Inheritence | 1.It doesn’t supports multiple inheritance concept |
| 2.Interface doesn’t contain data member, constructor | 2. It contains data member and contructors |
| 3.It contains only abstract methods | 3.It contains both abstract and non abstract methods |
| 4.It does not have access specifiers by default everything is public | 4.It has specific access specifiers |
| 5.member of interface cannot be static | 5.only complete memberof an abstract can be static |

* **When do you favor abstract class over interface?**

(i) By using abstract class, We can share code among several closely related classes. but using interface we dont know which class implements interface. an unrelated classes would implement our interface.

(ii) In abstract class we can declare non-static or non-final fields but we can't declare a non static or non-final field in interface.

(iii) classes that extend abstract class can have many common methods or fields.

* **What is abstract method in Java?**

Methods which doesnt have implementation are known as abstract methods in java Syntax for abstract method:

public void display();

* **Can abstract class contains main method in Java ?**

Abstract class can have main method in java. since it is a static method it doesnt need an object to call

**Example:**

public abstract class Sample

{

public static void main(String args[])

{

System.out.println("Abstract Class main method : ");

}

}

* **what is static block in java?**

**Static blocks are called satic initialization blocks. A static initialization block is a normal block of code enclosed in braces {} preceded by static keyword. thos block of code will run when the jvm loads the class file (ie) thse blocks will run before the main function** .A class can have multiple static blocks which will execute with the same sequence as it is written inside the class.

Example for single static block:

class Demo{

static int num;

static String mystr;

static{

num = 10;

mystr = "priya";

}

public static void main(String args[])

{

System.out.println("Value of num: "+num);

System.out.println("Value of mystr: "+mystr);

}

}

output:

Value of num: 10

Value of mystr:priya

Example for multi static block:

class Demo1{

static int num;

static String mystr;

//First Static block

static{

System.out.println("Static Block 1");

num = 10;

mystr = "Block1";

}

//Second static block

static{

System.out.println("Static Block 2");

num = 20;

mystr = "Block2";

}

public static void main(String args[])

{

System.out.println("Value of num: "+num);

System.out.println("Value of mystr: "+mystr);

}

}

output:

Static Block 1

Static Block 2

Value of num: 20

Value of mystr:Block2

* **What is the need of static block?**

(i)we donot need to create object for calling the static blocks.we can initialize the static variables it will be executed when the class file is loaded.

(ii)we can access static fields of the class before we run the constructor.because to execute constructor block we need to do instantiation.

* **Can we overload static methods in java?**

yes, we can overload static methods in java. but we cannot override static methods in java.

Example for overloading static methods:

public class Test {

public static void display() {

System.out.println("display method is called");

}

public static void display(int a) {

System.out.println("display(int a) method is called ");

}

public static void main(String args[])

{

Test.display();

Test.display(10);

}

}

output:

display method is called

display(int a) method is called

* **Can we call super class static methods from sub class?**

we can call super class static method from subclass using both parent class name and child class name.

Example

public class SuperDemo{

public static void show(){

System.out.println("Super class show() method called");

}

}

public class SubDemo extends SuperDemo{

public void print(){

System.out.println("Sub class print() method called");

}

public static void main(String args[]){

SuperDemo.show();

SubDemo.show();

}

}

output:

Super class show() method called

Super class show() method called

* **What is the difference between final and static keywords?**

**Final:**

1)When we apply "final" keyword to a variable,the value of that variable remains constant. (or) Once we declare a variable as final.the value of that variable cannot be changed.

2)It is useful when a variable value does not change during the life time of a program

Static

1)when we apply "static" keyword to a variable ,it means it belongs to class.

2)When we apply "static" keyword to a method,it means the method can be accessed without creating any instance of the class

* **Write a note on covariant return type with example code.**

The covariant return type specifies that the return type may vary in the same direction as the subclass.

Example Code:

class A

{

A get()

{return this;}

}

class B1 extends A

{

B1 get()

{return this;}

void message()

{

System.out.println("welcome to covariant return type");

}

public static void main(String args[]){

new B1().get().message();

}

}

* **Write a note on Enum with example code.**

(i) The enum keyword is used to declare an enumeration, a distinct type that consists of a set of named constants called the enumerator list.

(ii) Usually it is best to define an enum directly within a namespace so that all classes in the namespace can access it with equal convenience. However, an enum can also be nested within a class or struct. 1

By default, the first enumerator has the value 0, and the value of each successive enumerator is increased by 1.

Example code :

enum Color

{

RED, GREEN, BLUE;

}

public class Test

{

public static void main(String[] args)

{

Color c1 = Color.RED;

System.out.println(c1);

}

}

* Write a note on use of super keyword and super() method.

Super Keyword:

The super keyword in java is a reference variable which is used to refer immediate parent class object. Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

Usage of super Keyword:

1.Super can be used to refer immediate parent class instance variable.

We can use super keyword to access the data member or field of parent class. It is used if parent class and child class have same fields.

2.Super can be used to invoke immediate parent class method.

The super keyword can also be used to invoke parent class method. It should be used if subclass contains the same method as parent class. In other words, it is used if method is overridden.

Super() Method

Super() is used to call immediate parent. Super() can be used with instance members i.e., instance variables and instance methods. Super() can be used within constructor to call the constructor of parent class.

* **Write a code to implement abstraction using interface.**

public interface MyInterface

{

public void method1();

public void method2();

}

public class A implements MyInterface {

public void method1()

{

System.out.println("implementation of method1");

}

public void method2()

{

System.out.println("implementation of method2");

}

public static void main(String arg[])

{

MyInterface obj = new A();

obj.method1();

obj.method2();

}

}

* **Write a Java program to sort a numeric array and a string array.**

import java.util.Arrays;

public class Sorting {

public static void main(String[] args){

int[] array1 = {56,2,76,34,7,6};

String[] array2 = {"orange","apple","mango"};

System.out.println("Original numeric array : "+Arrays.toString(array1));

Arrays.sort(array1);

System.out.println("Sorted numeric array : "+Arrays.toString(array1));

System.out.println("Original string array : "+Arrays.toString(array2));

Arrays.sort(array2);

System.out.println("Sorted string array : "+Arrays.toString(my\_array2));

}

}

output:

Original numeric array : [56,2,76,34,7,6]

Sorted numeric array : [2,6,7,34,56,76]

Original string array : [orange,apple,mango]

Sorted string array : [apple,mango,orange]

* **Write a Java program to sum values of an array.**

public class SumOfArrays {

public static void main(String[] args) {

int array[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

int sum = 0;

for (int i : array)

sum += i;

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

}

}

output:

The sum is 55

* **Write a Java program to remove a specific element from an array.**

import java.util.Scanner;

public class Delete

{

public static void main(String[] args)

{

int n, x, flag = 1, loc = 0;

Scanner s = new Scanner(System.in);

System.out.print("Enter no. of elements you want in array:");

n = s.nextInt();

int a[] = new int[n];

System.out.println("Enter all the elements:");

for (int i = 0; i < n; i++)

{

a[i] = s.nextInt();

}

System.out.print("Enter the element you want to delete:");

x = s.nextInt();

for (int i = 0; i < n; i++)

{

if(a[i] == x)

{

flag =1;

loc = i;

break;

}

else

{

flag = 0;

}

}

if(flag == 1)

{

for(int i = loc+1; i < n; i++)

{

a[i-1] = a[i];

}

System.out.print("After Deleting:");

for (int i = 0; i < n-2; i++)

{

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

}

System.out.print(a[n-2]);

}

else

{

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

}

}

}

* **Write a Java program to reverse an array of integer values.**

import java.util.Arrays;

public class Reverse {

public static void main(String[] args){

int[] array1 = {12,13,14,18};

System.out.println("Original array : "+Arrays.toString(array1));

for(int i = 0; i < array1.length / 2; i++)

{

int temp = array1[i];

array1[i] = array1[array1.length - i - 1];

array1[array1.length - i - 1] = temp;

}

System.out.println("Reverse array : "+Arrays.toString(array1));

}

}

output:

Original array : [12,13,14,18]

Reverse array : [18,14,13,12]

* **Write a Java program to find the duplicate values of an array of integer values.**

import java.util.Arrays;

public class Duplicate {

public static void main(String[] args)

{

int[] array = {1, 2, 5, 5, 6, 6, 7, 2};

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

{

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

{

if ((array[i] == array[j]) && (i != j))

{

System.out.println("Duplicate Element : "array[j]);

}

}

}

}

}

output:

Duplicate Element : 2

Duplicate Element : 5

Duplicate Element : 6

﻿