DAY-33

-------

CONSTRUCTOR OVERLOADING:

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

Defining more than one constructor in a class by changing the :

1. number of parameter

2. type of parameter

3. order of parameter

EXAMPLE:

--------

class Employee

{

private String name;

private int id;

private float salary;

public Employee(String name)

{

this.name = name;

}

public Employee(String name,int id)

{

this.name = name;

this.id = id;

}

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

{

this.name = name;

this.id = id;

this.salary = salary;

}

public String getName()

{

return name;

}

public int getId()

{

return id;

}

public float getSalary()

{

return salary;

}

}

class Demo

{

public static void main(String[] args)

{

Employee e1 = new Employee("ramu");

System.out.println(e1.getName());

Employee e2 = new Employee("ramu",124);

System.out.println(e2.getName());

System.out.println(e2.getId());

Employee e3 = new Employee("ramu",124,50.45f);

System.out.println(e3.getName());

System.out.println(e3.getId());

System.out.println(e3.getSalary());

}

}

OUTPUT:

--------

ramu

ramu

124

ramu

124

50.45

NOTE: Constructor overloading is used to extend the funcyionality of the constructor .It exhibits virtual polymorphism.

this() constructor call:

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

this --> this keyword is used which will internally points to the currently executing object.

this() --> this() method is a constructor call within the same class.

Calling the constructor within another constructor of a same class can be achived using this()

EXAMPLE:

--------

class A

{

A()

{

System.out.println("inside default constructor");

}

A(int x)

{

this();

System.out.println("inside parametrized constructor");

}

}

class Demo1

{

public static void main(String[] args)

{

A a = new A(18);

}

}

OUTPUT:

-------

inside default constructor

inside parametrized constructor

EXAMPLE:2

----------

class A

{

A()

{

this("ramu");

System.out.println("inside default constructor");

}

A(int x)

{

this();

System.out.println("inside parametrized constructor");

}

A(String s)

{

System.out.println("inside String parametrized constructor");

}

}

class Demo1

{

public static void main(String[] args)

{

A a = new A(18);

}

}

OUTPUT:

-------

inside String parametrized constructor

inside default constructor

inside parametrized constructor

Whenever this() method is used we must fallow some rules:

RULE:1

-------

-->this() should be used as the first statement in the user defined constructor if we place it any where else we will get compilation error.

EXAMPLE:

--------

class A

{

A()

{

System.out.println("inside default constructor");

//this(); --> compilation error

}

}

class Demo1

{

public static void main(String[] args)

{

A a = new A();

}

}

OUTPUT:

-------

call to this must be first statement in constructor

this();// --> compilation error

RULE:2

------

this() constructor call can not be recurrsive in nature.

EXAMPLE:

--------

class A

{

A()

{

this();

System.out.println("inside default constructor");

}

}

class Demo1

{

public static void main(String[] args)

{

A a = new A();

}

}

OUTPUT:

-------

error: recursive constructor invocation A()

RULE:3

------

this() can be used or placed only within the constructor but not in any method.

EXAMPLE:3

---------

class A

{

A()

{

System.out.println("inside default constructor");

}

void A()

{

this();

System.out.println("inside default constructor");

}

}

class Demo1

{

public static void main(String[] args)

{

A a = new A();

}

}

OUTPUT:

-------

error: call to this must be first statement in constructor this();

this() and super()

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

NOTE: Inside the constructor the first statement should be either this() or super(). If the user is not using either super() or this() then compiler

by default will add super() to the constructor.

EXAMPLE:

---------

class A

{

A()

{

super() --> added by the compiler

System.out.println("inside default constructor");

}

}

NOTE: super() and this() can not be placed in the same constructor.

EXAMPLE:

---------

class A

{

A()

{

super();

this();

System.out.println("inside default constructor");

}

}

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

TYPES OF METHODS OR BLOCKS

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

The types of methods or blocks are :

1. instance methods and instance block

2. static methods and static blocks

EXAMPLE:

-------

class Demo

{

int a = 10;

int b = 20; // instance variables OR object variables.

void fun() // instance methods

{

int c;

int c=a+b;

}

{

system.out.println("hello world"); // instance block

}

}

NOTE: Instance blocks are anonymous(nameless) blocks which are executed during the object creation before the constructor call

EXAMPLE:

--------

class A

{

A()

{

System.out.println("inside the default constructor");

}

A(int x)

{

System.out.println("inside the parametrized constructor");

}

{

System.out.println("inside the instance block");

}

}

class Demo2

{

public static void main(String []args)

{

A a =new A();

}

}

OUTPUT:

-------

inside the instance block

inside the default constructor

NOTE: A class can have any number of instance blocks the order of execution is always from top-bottom.

EXAMPLE:

---------

class A

{

{

System.out.println("inside the first instance block");

}

{

System.out.println("inside the second instance block");

}

{

System.out.println("inside the third instance block");

}

}

class Demo2

{

public static void main(String []args)

{

A a =new A();

}

}

OUTPUT:

-------

inside the first instance block

inside the second instance block

inside the third instance block

NOTE: The order of the memeory allocation is 1. instance variables 2. instance blocks(non-static blocks) 3. constructor

EXAMPLE:

--------

class A

{

//int a=10;

A()

{

System.out.println("inside the constructor");

}

{

System.out.println("inside the instance block");

}

float f = 12.14f;

void fun()

{

System.out.println("inside the instance method");

}

}

class Demo2

{

public static void main(String []args)

{

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

A a =new A();

}

}

OUTPUT:

------

inside the main method

inside the instance block

inside the constructor

NOTE: Unlike constructors instance blocks are used to initialize the instance variables

EXAMPLE:

---------

class A

{

int age;

{

age=16;

System.out.println("the age is"+age);

}

}

class Demo2

{

public static void main(String []args)

{

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

A a =new A();

}

}

OUTPUT:

-------

inside the main method

the age is 16

NOTE: Instance blocks are used to provide the logic which is common for all the objects. If the logic is specific for different

objects then we can use this code in the constructor

EXAMPLE:

--------

class Student

{

int register\_number;

{

System.out.println("WELCOME TO STUDY ONLINE");

}

Student()

{

register\_number=20;

System.out.println("your registration number is :" +register\_number);

}

Student(int x)

{

register\_number = x;

System.out.println("your registration number is :" +register\_number);

}

}

class Demo2

{

public static void main(String []args)

{

//System.out.println("inside the main method");

Student s1 =new Student();

Student s2 =new Student(21);

}

}

OUTPUT:

-------

WELCOME TO STUDY ONLINE

your registration number is :20

WELCOME TO STUDY ONLINE

your registration number is :21

NOTE: The memory for instance variables and methods and blocks are always given in the heap memory segment.