Constructors

Table Of Contents

Ways Of Initializing The Object

```
There are 3 ways to intilize the object.
---------------------------------------
1. using the refrence variable
2. using the methods
3. using the constructors
```

▼ using the reference variable

```
package _8waysOfInitializingTheObject;

public class _1initilizeObjectUsingReferenceVariableEx1
{
    // instance variables
    String Name;
    int age;
    int phone;
    public static void main(String[] args)
    {
        _1initilizeObjectUsingReferenceVariableEx1 io = new _1initilizeObjectUsingReferenceVariableEx1();

        //accessing objects using reference variable(io --> reference variable)
        io.Name="Nandan";
        io.age = 22;
        io.phone = 1234567;
        System.out.println(io.Name+"\n"+io.age+"\n"+io.phone);
    }
}
//output
Nandan
22
1234567

Process finished with exit code 0
```

▼ using the method

```
package _8waysOfInitializingTheObject;

import java.util.Scanner;

public class _2initializeObjectUsingMethodEx2
{
    public static void main(String[] args)
```

```
    {
        data rv = new data();

        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the name: ");
        String studentName = scanner.next();

        System.out.print("Enter the age: ");
        int studentAge = scanner.nextInt();


        rv.setName(studentName);
        rv.setAge(studentAge);

        System.out.println("Name: "+rv.getName());
        System.out.println("Age: "+rv.getAge());

    }
}
class data
{
    String name;
    int age;

    //initializing object using method
    public void setName(String sname)
    {
        name = sname;
    }
    public void  setAge(int sage)
    {
        age = sage;
    }
    String getName()
    {
        return name;
    }
    int getAge()
    {
        return age;
    }
}
//output
Enter the name: Nandan
Enter the age: 22
Name: Nandan
Age: 22
```

▼ using the constructors

```
package _8waysOfInitializingTheObject;

import java.util.Scanner;

public class _3initializeObjectUsingMethodEx3
{
    //you can use constructor here also but i used in separate class because this main class length is so big
    public static void main(String[] args)
    {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the name: ");
        String studentName = scanner.next();

        System.out.print("Enter the age: ");
        int studentAge = scanner.nextInt();

        System.out.print("Enter the phone: ");
        int studentPhone = scanner.nextInt();

        //calling constructor(This is Parameterized constructor)
        constructorEx cs = new constructorEx(studentName,studentAge,studentPhone);

    }
}
class constructorEx
{
    //instance variable
    String name;
    int age;
    int phone;

    //initializing object using constructor
    constructorEx(String sname, int sage, int sphone)
    {
```

```
        name = sname;
        age = sage;
        phone = sphone;
        System.out.println("Name: "+name);
        System.out.println("Age: "+age);
        System.out.println("Phone: "+phone);
    }
}
//output
Enter the name: Nandan
Enter the age: 22
Enter the phone: 1234567
Name: Nandan
Age: 22
Phone: 1234567
```

Introduction To Constructors

- It is a special type of method which is used to initialize the object.
- Every time an object is created using the new() keyword, at least one constructor is called.

Rules Of Constructor

1. Constructor name must be the same as its class name
2. A Constructor must have no explicit return type
3. A Java constructor cannot be abstract, static, final, and synchronized
4. The access modifier allowed for the Constructor method is public, private, protected, default.

Types Of Constructor

- constructor is classified into 2 types based on the use of parameter

1. default constructor

- In a class if constructor is not present during the compilation. The compiler will add default constructor.
- The default constructor is no argument or zero parameterized in nature.
- If user has provided the constructor within a class then during compilation compiler will add a super() method to the body of the Constructor.
- NOTE: Whenever compiler adds the default constructor the access modifier of the constructor will be same as that of class.

```
package Constructor;
class data
{
    String Name = "Nandan";
    String usn = "1";
    //default constructor
    data()
    {
        System.out.println("This is default constructor");
        this.Name = Name;
        this.usn = usn;
    }

    void display()
    {
        System.out.println("Name is: "+Name);
        System.out.println("USN is: "+usn);
    }
}

public class _1_defaultConstructor
{
    public static void main(String[] args)
    {
        /**
         * @Note
         * The movement when we create an object to object constructor is called
         * when ever we create an object to a class immediately constructor gets called
         */
        data data  = new data();
        data.display();
```

```
    }
}
//output
This is default constructor
Name is: Nandan
USN is: 1
```

2. parameterized constructor

- A constructor that takes parameter and the same parameter type should be used when we are creating an object using "new()"

```
package Constructor;

import java.util.Scanner;

class Data
{
    Data(String Name, String Usn)
    {
        System.out.println("This is parameterized constructor");
        System.out.println("Name is: "+Name);
        System.out.println("USN is: "+Usn);
    }
}



public class _2_parameterizedConstructor
{
    public static void main(String[] args)
    {
        Scanner info = new Scanner(System.in);
        System.out.print("Enter Name: ");
        String name = info.next();
        System.out.print("Enter USN: ");
        String usn = info.next();

        /**@Note - when we use parameterized constructor we need to pass the same datatype argument during object creation
         *
         */
        Data data = new Data(name,usn);
    }
}
//output
Enter Name: Nandan
Enter USN: 1
This is parameterized constructor
Name is: Nandan
USN is: 1
```

The Shadow Problem & Solution

- **Shadow Problem:** shadow problem occurs when the instance variable and local variable are of same name at that time the compiler gives full preference for local variable.

▼ Shadow Problem

```
package _10shadowProblemAndSolution;

import java.util.Scanner;

public class _1shadowProblemEx1
{
    public static void main(String[] args)
    {
        shadowProblemEx ex = new shadowProblemEx();
        Scanner scanner= new Scanner(System.in);
        System.out.print("Enter the name: ");
        String s_name = scanner.next();
        System.out.print("Enter the id: ");
        int s_id = scanner.nextInt();
        ex.setData(s_id,s_name);
        System.out.println("The Id is: "+ex.getStudentId());
        System.out.println("The Name: "+ex.getStudentName());
    }
}
class shadowProblemEx
{
    /*
```

```
        shadow problem occurs when the instance variable and local variable are of same name
        at that time the compiler gives full preference for local variable
     */
    private int studentId;
    private String studentName;
    public void setData(int studentId, String studentName)
    {
        //shadow problem causing here
        studentId = studentId;
        studentName= studentName;
    }
    int getStudentId()
    {
        return studentId;
    }
    String getStudentName()
    {
        return studentName;
    }
}
//output
Enter the name: Nandan
Enter the id: 41
The Id is: 0
The Name: null
```

- **Shadow Problem Solution: use of "this" keyword.** It always points the currently executing object and also brings the difference b/w local and instance variable.

▼ Shadow Problem Solution Using **"this"** keyword

```
package Constructor;

import java.util.Scanner;

/**
 * @Note
 * this-keyword:
 *  - it is used to resolve the shadow problem(instance variable and local variable are of same names)
 *  - this-keyword always points the current execution object
 *  - at that time the compiler gives full preference for local variable
 */

class thisKeywordExample
{
    private String name;
    private int id;

    public String getName()
    {
        return name;
    }

    public void setName(String name)
    {
        //shadow problem solution
        this.name = name;
    }

    public int getId()
    {
        return id;
    }

    public void setId(int id)
    {
        //shadow problem solution
        this.id = id;
    }
}

public class _4_theShadowProblemSolution_Using_this_keyword_Example
{
    public static void main(String[] args)
    {
        Scanner input = new Scanner(System.in);
        String name;
        int id;

        System.out.print("Enter your name: ");
        name = input.next();

        System.out.print("Enter your id: ");
```

```
        id= input.nextInt();

        thisKeywordExample example = new thisKeywordExample();
        example.setName(name);
        example.setId(id);

        System.out.println("Name is: "+example.getName());
        System.out.println("Id is: "+example.getId());
    }
}
//output
Enter your name: Nandan
Enter your id: 12
Name is: Nandan
Id is: 12
```

▼ **"this"** keyword usages

```
package Constructor;

import java.util.Scanner;

class thisUsageExample
{
    private String name;
    private int id;

    thisUsageExample(String Name,int Id)
    {
        this.name = Name;
        this.id = Id;
    }
}

public class _5_this_keyword_usage_Example
{
    public static void main(String[] args)
    {
        Scanner input = new Scanner(System.in);
        String name;
        int id;

        System.out.print("Enter your name: ");
        name = input.next();

        System.out.print("Enter your id: ");
        id= input.nextInt();

        new thisUsageExample(name,id);
        System.out.println("Name is: "+name);
        System.out.println("Id is: "+id);
    }
}
//output
Enter your name: Nandan
Enter your id: 12
Name is: Nandan
Id is: 12
```

Difference Between Constructor And Method

Constructor & Method

Aa Constructor	Method
1. Constructor should be same as class name	1. method name can be anything.
2. They don't have return type	2. return type is compulsory
3. Constructor calling is implicit	3. method calling is explicit
4. Constructor will be called only one during the object creation	4. method can be called n number times
5. access modifier permitted are : public, private, protected, default	5. we can also use static, final etc.
6. Constructors can not be inherited	6. methods can be inherited

Constructor Overloading

- Defining more than one constructor in a class by adding the parameter, changing the order of parameter, and by adding different parameter to the constructor.

    1. number of parameter
    2. type of parameter
    3. order of parameter

▼ **Code for clear understanding**

```
package Constructor;

class constructorOverloading
{
    private String a;
    constructorOverloading()
    {
        System.out.println("This is default constructor");
        System.out.println();
    }

    constructorOverloading(String a)
    {

        System.out.println(a);
        System.out.println();
    }

    constructorOverloading(String aa,int bb)
    {
        System.out.println("aa: "+aa);
        System.out.println("bb: "+bb);
        System.out.println();
    }

    constructorOverloading(int bbb,String aaa)
    {
        System.out.println("aaa: "+aaa);
        System.out.println("bbb: "+bbb);
        System.out.println();
    }
}


public class _3_constructorOverloading
{
    public static void main(String[] args)
    {
        System.out.println("Constructor Overloading\n");

        /**
         * @Note
         * Constructors can be overloaded in 3 ways
         * 1. OVERLOADING: number of parameter
         * 2. OVERLOADING: type of parameter
         * 3. OVERLOADING: order of parameter
         */

        //to read the default constructor
        new constructorOverloading();
        //for parameterized constructor with order string type
        new constructorOverloading("OVERLOADING: number of parameter");

        //for parameterized constructor with order string type then int type
        new constructorOverloading("OVERLOADING: type of parameter",10);

        //for parameterized constructor with order int type then string type
        new constructorOverloading(100,"OVERLOADING: order of parameter");
    }
}
//output
Constructor Overloading

This is default constructor

OVERLOADING: number of parameter

aa: OVERLOADING: type of parameter
bb: 10

aaa: OVERLOADING: order of parameter
bbb: 100
```

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

Difference of this And this()

this - its a keyword

- this - keyword is used to points to the currently executing object.
- this keyword solves the **shadow problem** by differentiating the local variable and global variable

▼ "this" keyword

```
package Constructor;

import java.util.Scanner;

class thisUsageExample
{
    private String name;
    private int id;

    thisUsageExample(String Name,int Id)
    {
        this.name = Name;
        this.id = Id;
    }
}

public class _5_this_keyword_usage_Example
{
    public static void main(String[] args)
    {
        Scanner input = new Scanner(System.in);
        String name;
        int id;

        System.out.print("Enter your name: ");
        name = input.next();

        System.out.print("Enter your id: ");
        id= input.nextInt();

        new thisUsageExample(name,id);
        System.out.println("Name is: "+name);
        System.out.println("Id is: "+id);
    }
}
//output
Enter your name: Nandan
Enter your id: 12
Name is: Nandan
Id is: 12
```

this() - its a method

- this() - method is used to call constructor with in another constructor with in the same class.

▼ "this()" method

```
package Constructor;

import java.util.Scanner;

class thisMethodExample
{
    private String name;
    private int id;
    private String email;
    private int phone;

    thisMethodExample()
    {
        System.out.println("this() method is used to call one constructor from another constructor");
    }

    thisMethodExample(String name, int id)
    {
        this(); //this method must be first statement while calling from another constructor
        this.name=name;
```

```
        this.id=id;
    }

    thisMethodExample(String name, int id, String email, int phone)
    {
        this(name,id); //in this method() we can call parameterized constructor with respective datatype
        this.email=email;
        this.phone=phone;
        System.out.println("Name: "+name);
        System.out.println("Id: "+id);
        System.out.println("Email: "+email);
        System.out.println("Phone: "+phone);

    }
}

public class _6_this_method_Example
{
    public static void main(String[] args)
    {
        new thisMethodExample("Nandan",12,"gnnandan7@gmail.com",123456789);
    }
}
//output
this() method is used to call one constructor from another constructor
Name: Nandan
Id: 12
Email: gnnandan7@gmail.com
Phone: 123456789
```

Rule of this() method

1. this() should be used as the **first statement** in the constructor if we place it any where else we will get compilation error.
2. this() constructor call should not be recursive in nature, that means this() method should not be used to call the same constructor.
3. this() can be used or placed only within the constructor to call another constructor but not in any method

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.

```
package Constructor;

class superMethodExample
{
    private String name;
    private int id;
    private String email;
    private int phone;

    superMethodExample()
    {
        System.out.println("super() method is used to call parent class constructor from child class constructor in Inheritance concep
    }

    superMethodExample(String name, int id)
    {
        this(); //this method must be first statement while calling from another constructor
        this.name=name;
        this.id=id;
    }

    superMethodExample(String name, int id, String email, int phone)
    {
        //this() calls the constructs of same class with matching parameters
        this(name, id); //in this method() we can call parameterized constructor with respective datatype
        this.email=email;
        this.phone=phone;
        System.out.println("Name: "+name);
        System.out.println("Id: "+id);
        System.out.println("Email: "+email);
        System.out.println("Phone: "+phone);
    }
}

class childClass extends superMethodExample
{
    //calling parent class constructor
    childClass()
    {
```

```
        //super() calls parent class constructor of matching parameters
        super("Nandan",12,"gnnandan7@gmail.com",123456789);
    }
}

public class _7_superMethod_Example
{
    public static void main(String[] args)
    {
//        using child class instance we are calling default constructor of child class
        new childClass();
    }
}
//output
super() method is used to call parent class constructor from child class constructor in Inheritance concept
Name: Nandan
Id: 12
Email: gnnandan7@gmail.com
Phone: 123456789
```

- **this - Keyword** : used to solve the shadow problem by differentiating local variable and global variable
- **this() - Method** : it should be first statement in the constructor and it should not call the same constructor where you used
- **super() - Method** : it also used to call the constructor and it should be the first statement within the constructor

NOTE - We should not use this() and super() in the same constructor because if we use any of the other will become second statement so we shouldn't use both to call the constructor.