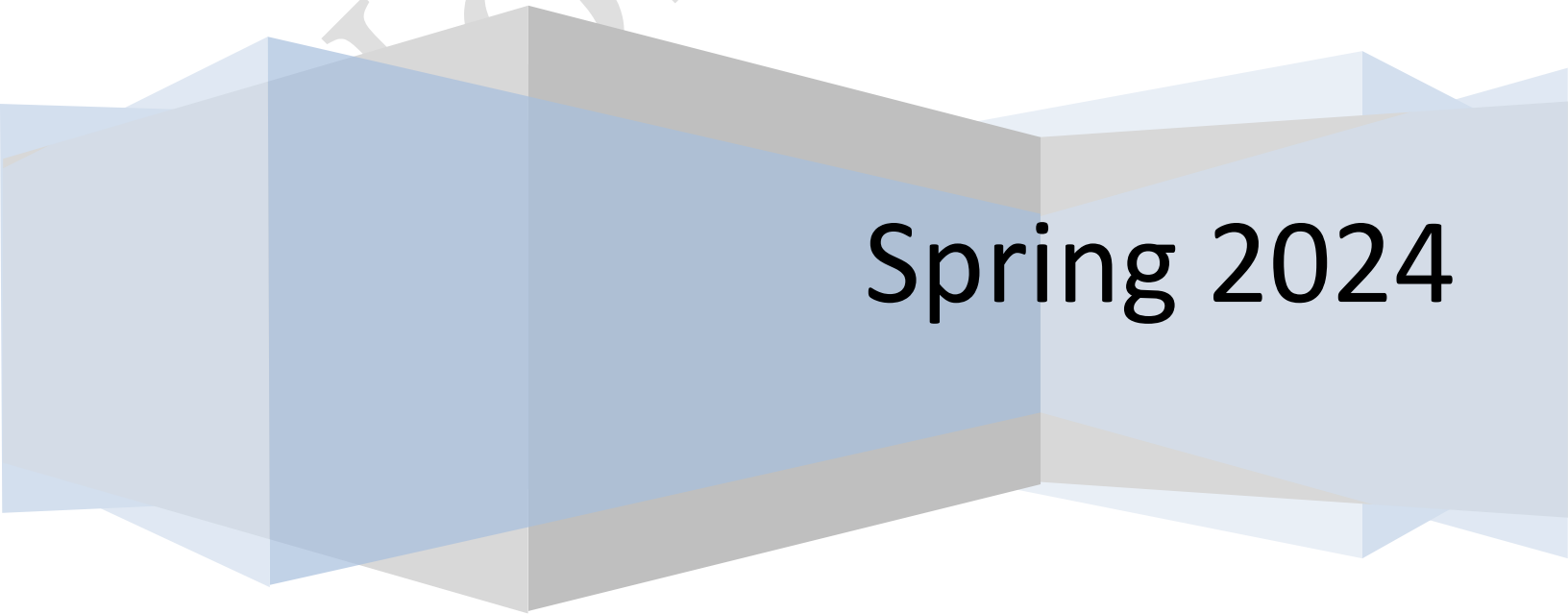
Northern University, Nowshera

Constructors in Java class

Week # 08 - Lecture 15- 16



Spring 2024

Learning Objectives:

- **CONSTRUCTORS**
  a. default constructor
  b. no argument constructor
  c. parameterized constructor
  d. constructor overloading
- **THIS KEYWORD**
  a. this as current class instance variable
  b. this to invoke current class method
  c. this to invoke current class constructor
  d. this as an argument in the method call
  e. this as argument in the constructor call
  f. this to return the current class instance from the method

1. Constructors in Java:

In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling constructor, memory for the object is allocated in the memory. 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.

Note: It is called constructor because it constructs the values at the time of object creation. It is not necessary to write a constructor for a class. It is because java compiler creates a default constructor if your class doesn't have any.

1.1. Need of Constructor?

Think of a Box. If we talk about a box class then it will have some class variables (say length, breadth, and height). But when it comes to creating its object (i.e Box will now exist in computer's memory), then can a box be there with no value defined for its dimensions. The answer is no.
So constructors are used to assign values to the class variables at the time of object creation, either explicitly done by the programmer or by Java itself (default constructor).

1.2. When is a Constructor called?

Each time an object is created using **new()** keyword at least one constructor (it could be default constructor) is invoked to assign initial values to the **data members** of the same class.

1.3. Rules for writing 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

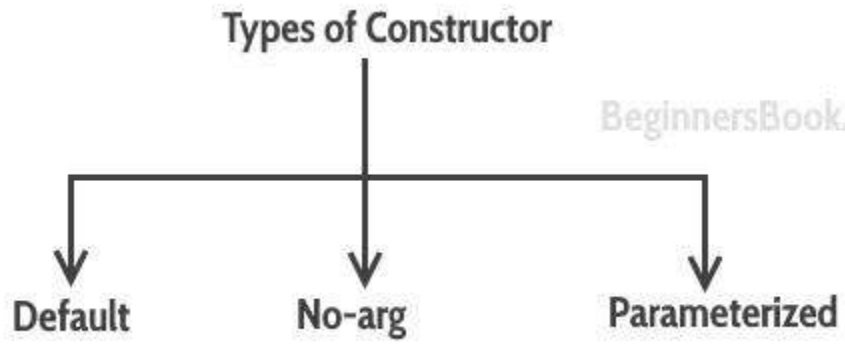
A constructor is invoked at the time of object or instance creation. For Example:

```
class Student
{
  .......
  // A Constructor
  Student() {}
  .......
}

// We can create an object of the above class using the below statement.
//This statement calls above constructor.
Student obj = new Student();
```

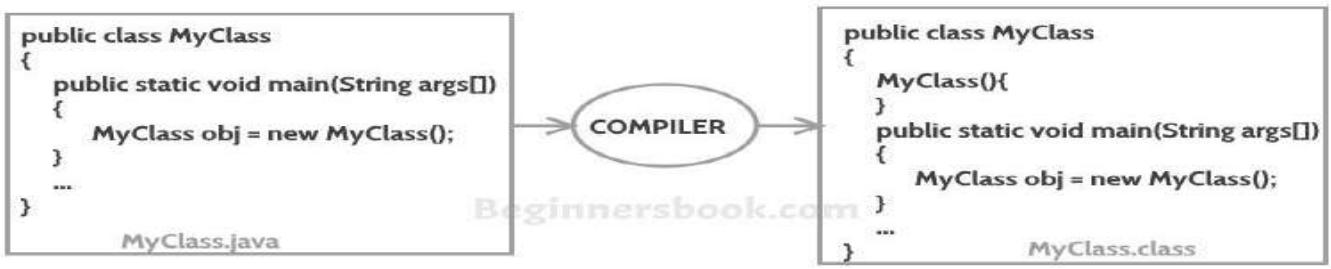
1.4. Types of constructor

There are three types of constructors: Default, No-arg constructor and Parameterized.



1.4.1. Default constructor

If you do not implement any constructor in your class, Java compiler inserts a default constructor into your code on your behalf. This constructor is known as default constructor. You would not find it in your source code (the java file) as it would be inserted into the code during compilation and exists in .class file. This process is shown in the diagram below:



The following example shows illustration of default constructor.

```
1. // Java Program to illustrate calling a no-argument constructor
2. import java.io.*;
3. class Student
4. {
5.   private int regNum;
6.   private String name;
7.   public void show()
8.   {
9.       System.out.println(regnum+ " \t" + name);
10.  }
11.}
12.class Main
13.{
14.  public static void main (String[] args)
15.  {
16.          // this would invoke default constructor.
17.          Student s1 = new Student();
18.          // Default constructor provides the default
```

```
19.          // values to the object like 0, null
20.           s1.show();
21.  }
22.}
```

Output :

```
0    null
```

If you implement any constructor then you no longer receive a default constructor from Java compiler.

1.4.2. no-arg constructor:

Constructor with no arguments is known as **no-arg constructor**. The signature is same as default constructor; however body can have any code unlike default constructor where the body of the constructor is empty.

Although you may see some people claim that that default and no-arg constructor is same but in fact they are not, even if you write **public Student() { }** in your class Student it cannot be called default constructor since you have written the code of it.

```
public class Student {

    private int regNum;
    private String name;
    public Student()
    {
        System.out.println("This is no argument Constructor");
    }
    public void show()
    {
        System.out.println(regNum+ " \t" + name);
    }
}
class Main
{
    public static void main (String[] args)
    {
        // this would invoke no argument constructor.
        Student s1 = new Student();
        // Default constructor provides the default
        // values to the object like 0, null
```

```
            s1.show();
        }
    }
```

Output :

```
This is no argument Constructor
0 null
```

1.4.3. Parameterized Constructor:

A constructor that has parameters is known as parameterized constructor. If we want to initialize fields of the class with your own values, then we use a parameterized constructor.

Example #1: parameterized constructor

```
// Java Program to illustrate calling a no-argument constructor
import java.io.*;
class Student
{
```

```
            private int regNum;
            private String name;
            public Student(String nm, int id)
            {
                 name = nm;       regNum = id;
            }
        public void show()
        {
        System.out.println("Student Registration # "+regNum+ " and name " + name);
        }
    }
    class Main
    {
                public static void main (String[] args)
                {
                        // this would invoke the parameterized constructor.
                        Student s1 = new Student("ali", 123);
```

```
                        s1.show();
                }
    }
```

Output :

```
Student Registration 123 and name ali
```

Example #2: parameterized constructor

In this example we have a parameterized constructor with two parameters id and name. While creating the objects obj1 and obj2 I have passed two arguments so that this constructor gets invoked after creation of obj1 and obj2.

```
public class Employee {

   private int empId;
   private String empName;

   //parameterized constructor with two parameters
   public Employee(int id, String name){
       empId = id;
       empName = name;
   }
   public void info(){
        System.out.println("Id: "+empId+" Name: "+empName);
   }

   public static void main(String args[]){
        Employee obj1 = new Employee(10245,"Asad");
        Employee obj2 = new Employee(92232,"Adeel");
        obj1.info();
        obj2.info();
   }
```

```
}
```

Output:

```
Id: 10245 Name: Asad
Id: 92232 Name: Adeel
```

Example3: parameterized constructor

In this example, we have two constructors, a default constructor and a parameterized constructor. When we do not pass any parameter while creating the object using new keyword then default constructor is invoked, however when you pass a parameter then parameterized constructor that matches with the passed parameters list gets invoked.

```
class Example2
{
	private int var;
	//default constructor
	public Example2()
	{
		var = 10;
	}
	//parameterized constructor
	public Example2(int num)
	{
		var = num;
	}
	public int getValue()
	{
		 return var;
	}
	public static void main(String args[])
	{
		 Example2 obj = new Example2();
		 Example2 obj2 = new Example2(100);
```

```
                System.out.println("var is: "+obj.getValue());
                System.out.println("var is: "+obj2.getValue());
        }
}
```

Output:

```
var is: 10
var is: 100
```

What if you implement only parameterized constructor in class?

```
class Example3
{
        private int var;
        public Example3(int num)
        {
                var=num;
        }
        public int getValue()
        {
                 return var;
        }
        public static void main(String args[])
        {
                 Example3 myobj = new Example3();
                 System.out.println("value of var is: "+myobj.getValue());
        }
}
```

Output: It will throw a compilation error. The reason is, the statement Example3 myobj = new Example3() is invoking a default constructor which we don't have in our program. When you don't implement any constructor in your class, compiler inserts the default constructor into

your code, however when you implement any constructor (in above example I have implemented parameterized constructor with int parameter), then you don't receive the default constructor by compiler into your code.

If we remove the parameterized constructor from the above code then the program would run fine, because then compiler would insert the default constructor into your code.

```
class Example3
{
	private int var;

	public int getValue()
	{
		return var;
	}
	public static void main(String args[])
	{
		Example3 myobj = new Example3();
		System.out.println("value of var is: "+myobj.getValue());
	}
}
```

Output:

```
value of var is: 0
```

1.5. Constructor Overloading

Constructor overloading in Java is a technique of having more than one constructor with different parameter lists. They are arranged in a way that each constructor performs a different task. They are differentiated by the compiler by the number of parameters in the list and their types. Compiler differentiates constructors on the basis of numbers of parameters, types of the parameters and order of the parameters.

Example1: Constructor overloading

```
// Java Program to illustrate constructor overloading

class Student
```

```
{
    // constructor with one argument
    public Student (String name)
    {
        System.out.println("Constructor with one argument - String : " + name);
    }
    // constructor with two arguments
    public Student (String name, int age)
    {
        System.out.println("Constructor with two arguments : String and Integer:"
+ name + " "+ age);
    }
    // Constructor with one argument but with different type than previous.
    public Student (int id)
    {
        System.out.println("Constructor with one argument : " +
                "int : " + id);
    }
}
class Main
{
    public static void main(String[] args)
    {
        // Creating the objects of the class named 's1' by passing different arguments
        // Invoke the constructor with one argument of type 'String'.
        Student s1 = new Student("Sikandar");

        // Invoke the constructor with two arguments
        Student s2 = new Student("Danish", 26);

        // Invoke the constructor with one argument of type 'int'.
        Student s3 = new Student (3256);
    }
}
```

Output:

```
Constructor with one argument - String : Sikandar

Constructor with two arguments - String and Integer : Danish 26

Constructor with one argument - int : 3256
```

Example2: Constructor overloading

```
class Student
{
    private int id;
    private String name;
    private int age;
    //creating two arg constructor
    public Student(int i,String n)
    {
       id = i;
       name = n;
```

```
    }
    //creating three arg constructor
    public Student(int i,String n,int a)
    {
        id = i;
        name = n;
        age=a;
    }
    public void display()
    {
        System.out.println(id+" "+name+" "+age);
    }
    public static void main(String args[])
    {
        Student s1 = new Student(111,"Ali");
        Student s2 = new Student(222,"Asad",25);
        s1.display();
        s2.display();
    }
}
```

Output:

```
111 Ali 0
222 Asad 25
```

2. this keyword in java

There can be a lot of usage of java this keyword. In java, this is a reference variable that refers to the current object.

Here is given the 6 usage of java this keyword.

1. this can be used to refer current class instance variable.

2. this can be used to invoke current class method (implicitly)

3. this() can be used to invoke current class constructor.

4. this can be passed as an argument in the method call.

5. this can be passed as argument in the constructor call.

6. this can be used to return the current class instance from the method.

Suggestion: If you are beginner to java, lookup only three usage of this keyword.

2.1 this: to refer current class instance variable

The this keyword can be used to refer current class instance variable. If there is ambiguity between the instance variables and parameters, this keyword resolves the problem of ambiguity.

Understanding the problem without this keyword

Let's understand the problem if we don't use this keyword by the example given below:

```
class Student {
    int rollno;
    String name;
    float fee;

    Student(int rollno, String name, float fee) {
        rollno = rollno;
        name = name;
        fee = fee;
    }

    void display() {
        System.out.println(rollno + " " + name + " " + fee);
    }
}

class TestThis1 {
    public static void main(String args[]) {
        Student s1 = new Student(111, "Ali", 5000f);
        Student s2 = new Student(112, "Asad", 6000f);
        s1.display();
        s2.display();
    }
```

```
}
```

Output:

```
0 null 0.0
0 null 0.0
```

In the above example, parameters (formal arguments) and instance variables are same. So, we are using this keyword to distinguish local variable and instance variable.

Solution of the above problem by this keyword

```
class Student {
    int rollno;
    String name;
    float fee;

    Student(int rollno, String name, float fee) {
        this.rollno = rollno;
        this.name = name;
        this.fee = fee;
    }

    void display() {
        System.out.println(rollno + " " + name + " " + fee);
    }
}

class TestThis2 {
    public static void main(String args[]) {
        Student s1 = new Student(111, "Ali", 5000f);
    Student s2 = new Student(112, "Asad", 6000f);
        s1.display();
        s2.display();
    }
}
```

Output:

```
111 Ali 5000
112 Asaf 6000
```

If local variables(formal arguments) and instance variables are different, there is no need to use this keyword like in the following program:

Program where this keyword is not required

```
class Student {
    int rollno;
    String name;
    float fee;
    Student(int r, String n, float f) {
        rollno = r;
        name = n;
        fee = f;
    }

    void display() {
        System.out.println(rollno + " " + name + " " + fee);
    }
}

class TestThis3 {
    public static void main(String args[]) {
        Student s1 = new Student(111, "Ali", 5000f);
        Student s2 = new Student(112, "Asad", 6000f);
        s1.display();
        s2.display();
    }
}
```

Output:

```
111 Ali 5000
112 Asad 6000
```

It is better approach to use meaningful names for variables. So we use same name for instance variables and parameters in real time, and always use this keyword.

2.2. this: to invoke current class method

You may invoke the method of the current class by using the this keyword. If you don't use the this keyword, compiler automatically adds this keyword while invoking the method. Let's see the example

User code	Compiler Generated code

`class A {`
`  void m() {`
`  System.out.println("hello m");`
`  }`
 	
`  void n() {`
`  System.out.println("hello n");`
`  m(); //same as this.m()`
`  }`
	`class A {`
	`  void m() {`
	`  System.out.println("hello m");`
	`  }`
	
	`  void n() {`
	`  System.out.println("hello n");`
	`  this.m();`
	`  }`

```
}
class TestThis4
{
    public static void main(String args[])
    {
        A a=new A();
        a.n();
    }
}
```

```
}
class TestThis4
{
    public static void main(String args[])
    {
        A a=new A();
        a.n();
    }
}
```

Output:

```
hello n
hello m
```

2.3. this: to invoke current class constructor

The this() constructor call can be used to invoke the current class constructor. It is used to reuse the constructor. In other words, it is used for constructor chaining.

Calling default constructor from parameterized constructor:

```
class A {
    private int x;
    public A() {
        System.out.println("hello a");
    }

    public A(int x) {
        this();
        System.out.println(x);
    }
}

class TestThis5 {
    public static void main(String args[]) {
        A a = new A(10);
    }
}
```

Output:

```
hello a
10
```

Calling parameterized constructor from default constructor:

```
class A {
    public A() {
        this(5);
        System.out.println("hello a");
    }

    public A(int x) {
        System.out.println(x);
    }
}

class TestThis6 {
    public static void main(String args[]) {
        A a = new A();
    }
}
```

Output:

```
5
hello a
```

Real usage of this() constructor call

The this() constructor call should be used to reuse the constructor from the constructor. It maintains the chain between the constructors i.e. it is used for constructor chaining. Let's see the example given below that displays the actual use of this keyword.

```
class Student {
    private int rollno;
    private String name, course;
    private float fee;

    public Student(int rollno, String name, String course) {
        this.rollno = rollno;
        this.name = name;
        this.course = course;
    }

    public Student(int rollno, String name, String course, float fee) {
        this(rollno, name, course);//reusing constructor  this.fee=fee;
    }

    public void display() {
```

```
        System.out.println(rollno + " " + name + " " + course + " " + fee);
    }
}

class TestThis7 {
    public static void main(String args[]) {
        Student s1 = new Student(111, "ali", "java");
        Student s2 = new Student(112, "asad", "java", 6000f);
        s1.display();
        s2.display();
    }
}
```

Output:

```
111 ali java null
112 asad java 6000
```

2. 4. this: to pass as an argument in the method

The this keyword can also be passed as an argument in the method. It is mainly used in the event handling. Let's see the example:

```
class S2 {
    void m(S2 obj) {
        System.out.println("method is invoked");
    }

    void p() {
        m(this);
    }

    public static void main(String args[]) {
        S2 s1 = new S2();
        s1.p();
    }
}
```

Output:

```
method is invoked
```

Application of this that can be passed as an argument:

In event handling (or) in a situation where we have to provide reference of a class to another one. It is used to reuse one object in many methods.

2. 5. this: to pass as argument in the constructor call

We can pass the this keyword in the constructor also. It is useful if we have to use one object in multiple classes. Let's see the example:

```
class B {
    A4 obj;

    B(A4 obj)
    {
        this.obj = obj;
    }

    void display()
    {
        System.out.println(obj.data);//using data member of A4 class
    }
}
class A4
{
        int data = 10;
        A4()
        {
            B b = new B(this);
            b.display();
        }
        public static void main (String args[])
        {
            A4 a = new A4();
        }
    }
```

Output:

```
Output:10
```

2. 6 this keyword can be used to return current class instance

We can return this keyword as an statement from the method. In such case, return type of the method must be the class type (non-primitive). Let's see the example:

Syntax of this that can be returned as a statement

```
return_type method_name()
{
    return this;
}
```

Example of this keyword that you return as a statement from the method

```
class A {
    A getA() {
        return this;
    }

    void msg() {
        System.out.println("Hello java");
    }
}

class Test1 {
    public static void main(String args[]) {
        A a = new A();
        a.getA().msg();
        //
    }
}
```

Output:

```
Hello java
```

Proving this keyword

Let's prove that this keyword refers to the current class instance variable. In this program, we are printing the reference variable and this, output of both variables are same.

```
class A5 {
    void m() {
        System.out.println(this);//prints same reference ID
    }

    public static void main(String args[]) {
        A5 obj = new A5();
        System.out.println(obj); //prints the reference ID  obj.m();
    }
}
```

Output:

```
A5@22b3ea59
```

```
A5@22b3ea59
```


Assignment #7

READ THE FOLLOWING INSTRUCTIONS CAREFULLY:

1. *Attempt the Assignment after reading the lecture notes, watching the videos lectures and doing self-learning of the topic from web.*
2. *Submit your assignment via email /Google class room to respective teacher by the end of this week, dated 19 April 2020 before : 11:59 PM*
3. *Your Assignment should be a single file either pdf or MS Word (handwritten scanned document) and must follow the naming format, your Name, Reg#, Section, subject and assignment number, i.e. AliAhmed_2018-Arid-0001-CS2A_OOP_Asgn4*

Q#1: Analyze and Complete missing code in following example:

```
class Company {
    String domainName;
    // missing code goes here
    public static void main(String[] args) {

        Company defaultObj = new Company();
        Company Obj = new Company("BIIT.edu.pk");
```

```
        defaultObj.printName ();
        Obj.printName ();

        defaultObj.changeName ("UAAR.edu.pk");
        defaultObj.printName ():
    }
```

Q#2: Write a program to print the name, age and GPA of students by creating a Student class. If no name is passed while creating an object of Student class, then the name should be "Unknown", age should be zero and GPA should be 0.0, otherwise the name should be equal to the String value passed while creating object of Student class. You should create at least three objects. ***Note: Use this keyword to set values of class***

Q#3: Rewrite program (Question #2) in a way that the values should entered by end user. Write an input function (instead parameterized constructor) that will ask the user to input name, age and GPA and set them to instance variables/attributes of student class. Write main program to create an object of class student and then set and display their values using functions.