Constructors

Introduction to Programming I

Lecture 2

Introduction to Constructors

This lecture and its associated materials have been produced by Mr. Abhinav Dahal (M.Sc., Linnaeus University) of iAcademy for the purposes of lecturing on the above described subject and the material should be viewed in this context. The work does not constitute professional advice and no warranties are made regarding the information presented. The Author and iAcademy do not accept any liability for the consequences of any action taken as a result of the work or any recommendations made or inferred. Permission to use any of these materials must be first granted by iAcademy.

Agenda

- Week 2 Lecture Coverage
  - Introduction to Constructors
  - Need for Constructor
  - Declaration & Definition of Constructors
  - Difference between Constructors & methods
  - Types of Constructors
  - The *this* keyword
  - Constructor Overloadding
  - Calling a Constructor Inside Another Constructor

Review of Previous Week

- Concept of Objects
- Introducing Classes
- About Java
- Introduction to BlueJ
- Java Character Set
- Tokens
- Identifiers

Let’s get started with Lecture 2

An Introduction to **Constructors**

Introduction - Constructors

- Object-oriented programming (OOP) is a particular conceptual approach to designing programs, and Java is a programming language that eases the way to applying that approach.
- The most important OOP features: ***Data Abstraction, Encapsulation, Data-hiding, Inheritance,*** and ***Polymorphism*** are not only implemented but also tied together by the single most important Java enhancement, ***a class.***

Introduction - Constructors

- But definition of a class only creates a data type.
- The objects of a class type have to be created and initialized separately, something that is done by a constructor.

Constructor

- A constructor is a member method of a class that is called for initializing when an object is created of that class.
- It has the same name as that of the class and its primary job is to initialize the object to a legal initial value for the class.
- If a class has a constructor, each object of that class will be initialized before any use is made of the object.

Constructor

- Consider the following class having a constructor:

```
class Student{
    private int rollNo;
    private float marks;
    public Student(){
        rollNo=0;
        marks=0.0
    }
}
```

constructor

Need for Constructor

- Constructors have one purpose in life: to create an instance of a class.
- This can be called creating an object :

  Student s1 = new Student();
- The purpose of other methods, by contrast, is much more general, a method's basic function is to execute Java code.

Need for Constructor

Declaration and Definition

- A constructor is a special method of a class with the same name as that of its class.
- A constructor is defined like other member functions of a class.
- In the previous example, the Student constructor has been defined as a *public* member function.

Constructor - Declaration and Definition

- We can even define a constructor under *private* or *protected* sections.
- A constructor also obeys the usual access rules of a class.
- That mean, a *private* or *protected* constructor is not available to the non-member functions.
- In other words, with a *private* or *protected* constructor, you cannot create an object of the same class in a non-member function, however, this is allowed in the member functions.

Difference Between Constructors & Methods

Parameter	Constructors	Methods
Purpose	Creates an instance of a class	Groups Java statements
Return type	Has no return type, not even void	void or a valid return type
Name	Same name as the class()	Any name except the class()
Execution	Called at the time of object creation	Called when a function call for the specific method is encountered. Method-calls are to be specified by the programmer.

Types of Constructors

- The constructor functions in Java can be of two types: One which can receive parameters **(parameterized)** and second, which cannot receive parameter **(non-parameterized)**.

Parameterized constructor

```
Student studentObj = new Student (“Ram Sharma”,20);
```

Non-parameterized constructor

```
Student studentObj = new Student();
```

Any Questions?

The *this* Keyword

- As soon as you define a class, the member functions are created and placed in the memory space *only once*.
- That is, only one copy of member functions is maintained that is shared by all the objects of the class.
- Only space for data members is allocated separately for each object.

The *this* Keyword

- This has an associated problem. If only one instance of a member function exists, how does it come to know which object's data member is to be manipulated?
- For example, if *member-function3* is capable of changing the value of *data-member2* and we want to change the value of *data-member2* of *object1* how would the *member-function3* come to know which object's *data-member2* is to be changed?

The *this* Keyword

- The answer to this problem is ***this*** keyword.
- When a member function is called, it is automatically passed an implicit argument that is a reference to the object that invoked the function.
- This reference is called ***this.***
- That is, if *object1* is invoking *memeber-function3,* then an implicit argument is passed to *member-function3* that points to *object1* i.e. ***this*** pointer now points to *object1.*

Constructor Overloading

- Just like any other function, the constructor of a class may also be overloaded so that even with different number and types of initial values, an object may still be initialized,
- For example, consider the following code fragment that shows a legal class definition, which has overloaded constructors.

Constructor Overloading – Example

```
Class ConsOverLoad{
    int a, b;
    float c;
    public ConsOverLoad(){
        a=0;
        b=0;
        c=0;
    }
    public ConsOverLoad(int x){
        a=x;
        b=x;
        c=0;
    }
    public ConsOverLoad(int x, int y, float z){
        a=x;
        b=y;
        c=z;
    }
    public void ConsOverLoadTest(){
      ConsOverLoad val1=new ConsOverLoad();
      ConsOverLoad val2=new ConsOverLoad(2);
       ConsOverLoad val3=new ConsOverLoad(2,10,7.5F);
    }
}
```

Constructor Overloading

- The previous example had three constructors with the same name as that of the class but these constructors are different from one another in terms of their ***signature*** i.e., the number and type of parameters differ from one another.
- Hence, these are called ***overloaded constructors.***

Calling a Constructor Inside Another Constructor

- In the previous example, the class ConsOverLoad has three overloaded constructors.

1. Takes no arguments
2. Takes one argument
3. Takes three arguments

Calling a Constructor Inside Another Constructor

- If we want to create an object by passing two arguments, we need to write a constructor with two arguments:

```
public ConsOverLoad(int x, int y){
    a=x;
    b=y;
    c=0;
}
```

Calling a Constructor Inside Another Constructor

- Alternatively, we may call a constructor with more parameters, giving default values for the missing parameters.
- For this, we need to use keyword ***this*** to call other constructors in the same class as shown below:

```
public ConsOverLoad(int x, int y){
        this(x, y, 0);
    }
```

This will invoke the third constructor that takes three arguments.

Calling a Constructor Inside Another Constructor

- So, you can use **this**(...) to call another constructor of same class from within a constructor.
- This call should be the first statement within a constructor.


ANY
QUESTIONS
?
www.rekruitin.com
ReKruiTIn.com

Summary: Week 2 Lecture

- Introduction to Constructors
- Need for Constructor
- Declaration & Definition of Constructors
- Difference between Constructors & methods
- Types of Constructors
- The *this* keyword
- Constructor Overloadding
- Calling a Constructor Inside Another Constructor

What to Expect: Week 2 Lab

- Discussion about the topics covered in the lecture.
- Provide practical questions based on constructor so it is vital that you go through the lecture slide!!.


Thank you