

Week - 03

CONSTRUCTORS

A constructor is a special kind of method that determines how an object is initialized when it's created.

Rules for creating Java constructor

There are basically three rules defined for the constructor.

1. Constructor name must be same as its class name
2. Constructor must have no explicit return type. i.e., Constructors do not have a return type not even void
3. Constructors are invoked using the new operator when an object is created. Constructors play the role of initializing objects.

Types of Java constructors

There are two types of constructors:

1. Default constructor (no-arg constructor)
2. Parameterized constructor

Default Constructor (No-argument constructor)

```
class Box
{
double width;
double height;
double depth;

    // This is the constructor for Box.
    Box()
    {
        System.out.println("Constructing Box");
        width = 10;
        height = 10;
        depth = 10;
    }

    // compute and return volume
    double volume()
    {
        return width * height * depth;
    }
}
```

Parameterized Constructors

Box() constructor in the preceding example does initialize a Box object, it is not very useful—all boxes have the same dimensions. What is needed is a way to construct Box objects of various dimensions. The easy solution is to add parameters to the constructor.

```
/* Here, Box uses a parameterized constructor to initialize the dimensions of a box. */
class Box
{
double width; double height; double depth;
// This is the constructor for Box.
    Box(double w, double h, double d)
    {
        width = w;
        height = h;
        depth = d;
    }
// compute and return volume
    double volume()
    {
        return width * height * depth;
    }
}

class BoxDemo7
{
    public static void main(String args[])
    {
// declare, allocate, and initialize Box objects
        Box mybox1 = new Box(10, 20, 15);
        Box mybox2 = new Box(3, 6, 9);
        double vol;
// get volume of first box
        vol = mybox1.volume();
        System.out.println("Volume is " + vol);
// get volume of second box
        vol = mybox2.volume();
        System.out.println("Volume is " + vol);
    }
}
```

DESTRUCTOR


- In Java, when we create an object of the class it occupies some space in the memory (heap). If we do not delete these objects, it remains in the memory and occupies unnecessary space that is not upright from the aspect of programming. To resolve this problem, we use the **destructor**.
- The **destructor** is the opposite of the constructor. The constructor is used to initialize objects while the destructor is used to delete or destroy the object that releases the resource occupied by the object.
- Remember that there is no concept of destructor in Java. In place of the destructor, Java provides the garbage collector that works the same as the destructor.

Advantages of Destructor

- It releases the resources occupied by the object.
- No explicit call is required, it is automatically invoked at the end of the program execution.
- It does not accept any parameter and cannot be overloaded.

ACCESS MODIFIERS [VISIBILITY CONTROL]

Access Modifier Access Location	Public	Protected	Friendly	Private
Same class	Yes	Yes	Yes	Yes
Subclass in same package	Yes	Yes	Yes	No
Other classes in same package	Yes	Yes	Yes	No
Subclasses in other package	Yes	Yes	No	No
Non subclasses in other package	Yes	No	No	No



Default Access Modifier(friendly) - No keyword:

- Default access modifier means we do not explicitly declare an access modifier for a class, field, method etc.
- A variable or method declared without any access control modifier is available to any other class in the same package. The default modifier cannot be used for methods, fields in an interface.

Private Access Modifier - private:

- Methods, Variables and Constructors that are declared private can only be accessed within the declared class itself.
- Private access modifier is the most restrictive access level. Class and interfaces cannot be private.

- Variables that are declared private can be accessed outside the class if public getter methods are present in the class.
- Using the private modifier is the main way that an object encapsulates itself and hide data from the outside world.

Public Access Modifier - public:

- A class, method, constructor, interface etc declared public can be accessed from any other class. Therefore fields, methods, blocks declared inside a public class can be accessed from any class belonging to the Java Universe.
- However, if the public class we are trying to access is in a different package, then the public class still need to be imported.
- Because of class inheritance, all public methods and variables of a class are inherited by its subclasses.

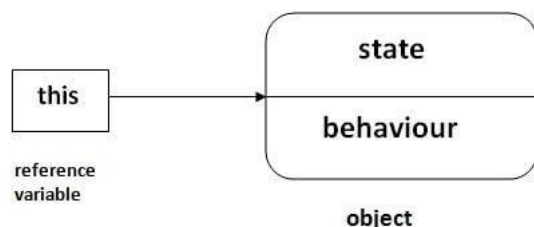
Protected Access Modifier - protected:

- Variables, methods and constructors which are declared protected in a superclass can be accessed only by the subclasses in other package or any class within the package of the protected members' class.
- The protected access modifier cannot be applied to class and interfaces. Methods, fields can be declared protected, however methods and fields in a interface cannot be declared protected.
- Protected access gives the subclass a chance to use the helper method or variable, while preventing a nonrelated class from trying to use it.

this KEYWORD

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

this keyword refers to the current object in a method or constructor.



this can also be used to:

- Invoke current class constructor
- Invoke current class method
- Return the current class object
- Pass an argument in the method call
- Pass an argument in the constructor call

AUTOBOXING AND UNBOXING

- The automatic conversion of primitive data types into its equivalent Wrapper type is known as boxing and opposite operation is known as unboxing. This is the new feature of Java5. So java programmer doesn't need to write the conversion code.
- No need of conversion between primitives and Wrappers manually so less coding is required.

Primitive type	Wrapper class
boolean	Boolean
byte	Byte
char	Character
float	Float
int	Integer
long	Long
short	Short
double	Double

Simple Example of Autoboxing in java:

```
class BoxingExample1{
    public static void main(String args[]){
        int a=50;
        Integer a2=new Integer(a);//Boxing
        Integer a3=5;//Boxing
        System.out.println(a2+" "+a3);
    }
}
```

Simple Example of Unboxing in java:

The automatic conversion of wrapper class type into corresponding primitive type, is known as Unboxing. Let's see the example of unboxing:

```
class UnboxingExample1{
    public static void main(String args[]){
        Integer i=new Integer(50);
        int a=i;
        System.out.println(a);
    }
}
```

OPERATORS IN JAVA

An operator is a symbol that tells the computer to perform certain mathematical or logical manipulation. Operators are used in the program to manipulate data and variables. They usually form a part of the mathematical or logical expression.

Java operators can be divided into following categories:

1. Arithmetic Operators
2. Relational Operators
3. Bitwise Operators
4. Logical Operators
5. Assignment Operators
6. conditional operator

Arithmetic Operators

The basic arithmetic operations in Java Programming are addition, subtraction, multiplication, and division.

Operator	Description	Example
+ (Addition)	Adds two operands	$5 + 10 = 15$
- (Subtraction)	Subtract second operands from first. Also used to Concatenate two strings	$10 - 5 = 5$
* (Multiplication)	Multiplies values on either side of the operator.	$10 * 5 = 50$
/ (Division)	Divides left-hand operand by right-hand operand.	$10 / 5 = 2$
% (Modulus)	Divides left-hand operand by right-hand operand and returns remainder.	$5 \% 2 = 1$
++ (Increment)	Increases the value of operand by 1.	2++ gives 3
-- (Decrement)	Decreases the value of operand by 1.	3-- gives 2

Relational Operators

- Relational Operators are used to checking relation between two variables or numbers.
- Relational Operators are Binary Operators.
- Relational Operators returns “Boolean” value. i.e., it will return true or false.
- Most of the relational operators are used in “If statement” and inside Looping statement in order to check truthiness or falseness of condition.

Operators	Descriptions	Examples
== (equal to)	This operator checks the value of two operands, if both are equal , then it returns true otherwise false.	(2 == 3) is not true.
!= (not equal to)	This operator checks the value of two operands, if both are not equal , then it returns true otherwise false.	(4 != 5) is true.
> (greater than)	This operator checks the value of two operands, if the left side of the operator is greater , then it returns true otherwise false.	(5 > 56) is not true.
< (less than)	This operator checks the value of two operands if the left side of the operator is less , then it returns true otherwise false.	(2 < 5) is true.
>= (greater than or equal to)	This operator checks the value of two operands if the left side of the operator is greater or equal , then it returns true otherwise false.	(12 >= 45) is not true.
<= (less than or equal to)	This operator checks the value of two operands if the left side of the operator is less or equal , then it returns true otherwise false.	(43 <= 43) is true.

The Bitwise Operators

A	B	A & B	A B	A ^ B
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Java defines several bitwise operators, which can be applied to the integer types, long, int, short, char, and byte.

The Assignment Operator

The assignment operator is the single equal sign, =. The assignment operator works in Java much as it does in any other computer language. It has this general form:

```
var = expression;
```

```
int x, y, z;
```

```
x = y = z = 100; // set x, y, and z to 100
```

This fragment sets the variables x, y, and z to 100 using a single statement. This works because the = is an operator that yields the value of the right-hand expression. Thus, the value of z = 100 is 100, which is then assigned to y, which in turn is assigned to x. Using a "chain of assignment" is an easy way to set a group of variables to a common value.

The ? Operator or [Conditional Operator] or [Ternary operator]

Java includes a special ternary (three-way) operator that can replace certain types of "if then-else" statements. This operator is the ?, and it works in Java much like it does in C and C++. It can seem somewhat confusing at first, but the ? can be used very effectively

```
expression1 ? expression2 : expression3
```

Here, expression1 can be any expression that evaluates to a boolean value. If expression1 is true, then expression2 is evaluated; otherwise, expression3 is evaluated.

The result of the ? operation is that of the expression evaluated. Both expression2 and expression3 are required to return the same type, which can't be void.

Special Operators

Java supports special operators such as instanceof operator and member selection operator (.) or dot operator

- instanceof Operator

The java instanceof operator is used to test whether the object is an instance of the specified type (class or subclass or interface) i.e. object reference variables

```
(Object reference variable) instanceof (class/interface type)
```

Ex: person instanceof Student

is true if the object person belongs to the class student; otherwise it's false.

- Dot operator

The dot operator (.) is used to access the instance variables and methods of class objects

Ex: person1.age // reference to the variable age

Person1.salary() // reference to the method salary()

JAVA EXPRESSIONS

Every expression consists of at least one operator and an operand. Operand can be either a literal, variable or a method invocation.

```
int a = 10; //Assignment expression
```

```
System.out.println("Value = "+x;);
```

```
int result = a + 10; //Assignment exp
```

```
if(val1 <= val2) //Boolean expression
```

```
b = a++; //Assignment exp
```

Expressions are made bold and italic in the above examples

Startertutorials.com

Evaluation of expressions

It is common for an expression to have more than one operator. For example, consider the below example:

$$(20 * 5) + (10 / 2) - (3 * 10)$$

Expression evaluation in Java is based upon the following concepts:

- Operator precedence
- Associativity rules

Operator precedence

All the operators in Java are divided into several groups and are assigned a precedence level. The operator precedence chart for the operators in Java is shown below:

Highest Precedence	Operators
	++ (postfix), -- (postfix)
	++ (prefix), -- (prefix), ~, !, +(unary), -(unary), (type-cast)
	*, /, %
	+, -
	>>, >>>, <<
	>, >=, <, <=, instanceof
	==, !=
	&
	^
	&&
	?:
Lowest Precedence	=, op=

Startertutorials.com

Now let's consider the following expression:

$$10 - 2 * 5$$

One will evaluate the above expression normally as, $10-2$ gives 8 and then $8*5$ gives 40. But Java evaluates the above expression differently. Based on the operator precedence chart shown above, $*$ has higher precedence than $+$. So, $2*5$ is evaluated first which gives 10 and then $10 - 10$ is evaluated which gives 0.

What if the expression contains two or more operators from the same group? Such ambiguities are solved using the associativity rules.

Associativity Rules

When an expression contains operators from the same group, associativity rules are applied to determine which operation should be performed first. The associativity rules of Java are shown below:

Operator Group	Associativity	Type of Operation
! ~ ++ -- + -	right-to-left	unary
* / %	left-to-right	multiplicative
+ -	left-to-right	additive
<< >> >>>	left-to-right	bitwise
< <= > >=	left-to-right	relational
== !=	left-to-right	relational
&	left-to-right	bitwise
^	left-to-right	bitwise
	left-to-right	bitwise
&&	left-to-right	boolean
	left-to-right	boolean
?:	right-to-left	conditional
= += -= *= /= %= &=	right-to-left	assignment
^= = <<= >>= >>>=	right-to-left	assignment
,	left-to-right	comma

Startertutorials.com

Now, let's consider the following expression:

$$10-6+2$$

In the above expression, the operators $+$ and $-$ both belong to the same group in the operator precedence chart. So, we have to check the associativity rules for evaluating the above expression. Associativity rule for $+$ and $-$ group is left-to-right i.e, evaluate the expression from left to right. So, $10-6$ is evaluated to 4 and then $4+2$ is evaluated to 6.

Use of parenthesis in expressions

Let's look at our original expression example:

$$(20 * 5) + (10 / 2) - (3 * 10)$$

You might think that, what is the need of parenthesis (and) in the above expression. The reason I had included them is, parenthesis have the highest priority (precedence) over all the operators in Java.

So, in the above expression, $(20*5)$ is evaluated to 100, $(10/2)$ is evaluated to 5 and $(3*10)$ is evaluated to 30. Now, our intermediate expression looks like:

$$100 + 5 - 30$$

Now, we can apply the associativity rules and evaluate the expression. The final answer for the above expression is 75.

```
public class ExpressionDemo
{
    public static void main(String args[])
    {
        int a = (20*5)+(10/2)-(3 * 10);
        System.out.println("value of a is="+ a);
    }
}
```