Object Oriented Programing 1

Lecture 5: Type Casting, Operators in Java, Static and Initializer Block and User input/output

By

Elubu Joseph

MSci.IS

Email: josebulinda@gmail.com

or

jose@kumiuniversity.ac.ug

Agenda

1. Type Casting
2. Operators in Java,
3. Static and Initializer Block,
4. User input/output

Type Casting in Java

Type Casting in Java

Type Casting is a process of changing/converting one data type value to another type. In Java, we can convert one data type of value to another type.

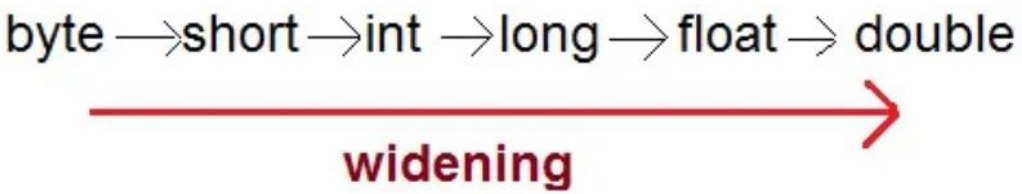
Example :

```
int x = 11;
byte y = (byte)x;
```

Classification of Type casting

In Java, type casting is classified into two types,

1. Widening Casting(Implicit)



2. Narrowing Casting(Explicitly done)

Widening or Automatic type conversion

Automatic Type casting take place when;-

1. the two types are compatible
2. the target type is larger than the source type **E.g.**

```
public class TypeCastPro{
  public static void main(String[] args){
  int i = 450;
  long l = i;
  float f = l; //no explicit type casting required
  System.out.println("Int ="+i);
  System.out.println("Long = "+l);
  System.out.println("Float = "+f);
} }
```

Outtput

```
Int =450
Long = 450
Float = 450.0
```

Narrowing or Explicit type conversion

When assigning a larger type value to a variable of smaller type, you need to perform explicit type casting. If you don't perform casting then compiler reports compile time error.

Example

```
public class TypeCastPro1{
  public static void main(String[] args){
    double d = 106.06;
    long l = (long)d; //explicit type casting required
    int i = (int)l; //explicit type casting required
    System.out.println("Double = "+d);
    System.out.println("Long = "+l);
    System.out.println("Int = "+i);
} }
```

Output

```
Double = 106.06
Long = 106
Int = 106
```

Conversion of int and double into a byte

Here, we are converting int and double type to byte type by using **explicit type casting.**

```
class TypeCastPro2 {
  public static void main(String args[]){
    Byte b;
    int i = 300;
    double d = 400.150;
    b = (byte)i;
    System.out.println(" Conversion of int to byte: i = " + i + " b = " +
b);
    System.out.println("================");
    b = (byte) d; System.out.println("Conversion of double to
byte: d = " + d + " b= " + b);
} }
```

Conversion of int and double into a byte

Output

```
Conversion of int to byte: i = 300 b = 44
==========================
Conversion of double to byte: d = 400.15 b= -112
```

Operators in Java

Operators in Java

Operator is a symbol which tells the compiler to perform some operation on operands. **Operands** are values on which operators operate to achieve some results.

Sometimes we need to perform arithmetic operations then we use plus (+) operator for addition, multiply(*) for multiplication etc. Operators are always essential part of any programming language.

Operators in Java +

Categories of Operators

Java operators can be divided into following categories:

1. Arithmetic operators
2. Relational operators
3. Logical operators
4. Bitwise operators
5. Assignment operators
6. Conditional operators
7. Instanceof opearator

1. Arithmetic operators+

are used to perform arithmetic operations like: addition, subtraction, division etc which help solve mathematical expressions. The below table contains Arithmetic operators.

Operator	**Description**
+	adds two operands
-	subtract second operands from first
*	multiply two operand
/	divide numerator by denumerator
%	remainder of division
++	Increment operator increases integer value by one
--	Decrement operator decreases integer value by one

Arithmetic operators++

Sample Program:

Lets create an example to understand arithmetic operators and their operations.

```
class Arithmetic{
public static void main(String as[]){
int a, b, c; a=21; b=2;
c=a+b;
System.out.println(a+"+"+b +" = "+c);
c=a-b;
System.out.println(a+"-"+b +" = "+c);
c=a*b;
System.out.println(a+"*"+b +" = "+c);
```

Arithmetic operators+++

Sample Program

```
c=a/b;
System.out.println(a+"/"+b +" = "+c);
c=a%b;
System.out.println(a+"%"+b +" = "+c);
a=++a;
System.out.println("Increment Operator a=++a = "+a);
a=--a;
System.out.println("decrement Operator a=--a "+a);
}
}
```

Arithmetic operators++++

Out put

```
===== Arithmetic Operator Results =====
21-2 = 19
21+2 = 23
21*2 = 42
21%2 = 1
21/2 = 10
Increment Operator a=++a: = 22
decrement Operator a=--a: = 21
```

2. Relational operators

Relational operators are used to compare operands or values from the left one on the right of the operator. It can be use to test whether two values are equal or not equal or less than or greater than etc.

Operator	Description	Example
==	Check if two operand are equal	3 == 3 True
!=	Check if two operand are not equal.	3 != 3 False
>	Check if operand on the left is greater than operand on the right	3 > 3 False
<	Check operand on the left is smaller than right operand	3 < 3 False
>=	check left operand is greater than or equal to right operand	3 >= 3 True
<=	Check if operand on left is smaller than or equal to right operand	3 <= 3 True

2. Relational operators+

Example 2

```
class Relationalops{
public static void main(String as[]){
int a, b;
a=70; b=60;
System.out.println("a == b = " + (a == b) );
System.out.println("a != b = " + (a != b) );
System.out.println("a > b = " + (a > b) );
System.out.println("a < b = " + (a < b) );
System.out.println("b >= a = " + (b >= a) );
System.out.println("b <= a = " + (b <= a) ); }
}
```

Output will generate true and false value

Output

Suppose a = 70 and b = 60, then

```
a == b = false
a != b = true
a > b = true
a < b = false
b >= a = false
b <= a = true
```

3. Logical operators

Logical Operators are used to check conditional expression. For example, we can use logical operators in if statement to evaluate conditional based expression. We can use them into loop as well to evaluate a condition. Java supports following 3 logical operator.

Suppose we have two variables whose values are: **a=true** and **b=false**.

Operator	Description	Example
&&	Logical AND	(a && b) is false
||	Logical OR	(a || b) is true
!	Logical NOT	(!a) is false

3. Logical operators+

Example:

Operators return either true or false value in the following example.

```
class LogicalOps{
public static void main(String as[]) {
    boolean a = true, b = false;
    System.out.println("a && b = " + (a&&b));
    System.out.println("a || b = " + (a||b) );
    System.out.println("!(a && b) = " + !(a && b));
}
}
```

3. Logical operators+

Example:

Operators return either true or false value in the following example.

Output

```
a && b = false
a || b = true
!(a && b) = true
```

4. Bitwise operators

Bitwise operators perform manipulations of data at **bit level**. These operators also perform **shifting of bits** from right to left.

Operator	Description
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
<<	left shift
>>	right shift

Note! Bitwise operators are not applied to float or double. Be sure to learn about data types in next lectures.

Interpreting Bitwise **&**, **|** and **^** operations

The table on the next slide shows the input terminals at **‘a’** and **‘b’** the output terminal is at each of the arguments given by each operator above, such as **a** & **b**, **a** | **b** and **a** ^ **b**.

1. For **a & b** The output is "true" when both inputs are "true." Otherwise, the output is "false.“
2. For **a | b** The output is "true" when either or both of the inputs are "true." Otherwise, the output is "false.“
3. For **a ^ b** The output is "true" when either of the inputs are “False.“, Otherwise, the output is "false.“

Now lets see truth table for bitwise &, | and ^

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

For **a & b** The output is "true" when both inputs are "true." Otherwise, the output is "false.“

a	b	a & b
0	0	0
0	1	0
1	0	0
1	1	1

For **a | b** The output is "true" when either or both of the inputs are "true.“

Otherwise, the output is "false.“

a	b	a | b
0	0	0
0	1	1
1	0	1
1	1	1

For **a ^ b** The output is "true" when either of the inputs are “False.“, Otherwise, the output is "false.“

a	b	a ^ b
0	0	0
0	1	1
1	0	1
1	1	0

Bitwise operators Left-Shift << and Right-Shift >>

Operator	Description
<<	left shift
>>	right shift

The bitwise **shift** operator, shifts the bit value. The left operand specifies the value to be shifted and the right operand specifies the number of positions that the bits in the value have to be shifted. Both operands have the same precedence

Example

Left-Shift symbol << and Right-shift>> Symbol

```
a = 0001000
b = 2
a << b = 0100000
a >> b = 0000010
```

Examples of left and right-shifts operations

a	b	a<<b	a>>b
0001000	2	0100000	0000010
00001000	2	00100000	00000010
00010000	2	01000000	00000100

Example of Bitwise Operator Program

```
class BitwiseOps{
public static void main(String as[]){
int a = 100; int b = 50;
int c = 0; c = a & b;
System.out.println("a & b = " +(a & b) );
System.out.println("a & b = " + c );
c = a | b;
System.out.println("a | b = " + c );
c = a ^ b;
System.out.println("a ^ b = " + c );
c = ~a;
System.out.println("~a = " + c );
c = a << 2;
System.out.println("a << 2 = " + c );
c = a >> 2;
System.out.println("a >>2 = " + c );
c = a >>> 2;
System.out.println("a >>> 2 = " + c );
} }
```

Output

```
===== Bitwise Operator Results =====
a & b = 32
a | b = 118
a ^ b = 86
~a = -101
a << 2 = 400
a >>2 = 25
a >>> 2 = 25
```

5. Assignment Operators

are used to assign a value to a variable. They are also used combined with arithmetic operators to perform arithmetic operations and then assign the result to the variable. Java surports the following assignment operators

Operator	Description	Example
=	Assigns values from right side operands to left side operand	a = b
+=	Adds right operand to the left operand and assign the result to left	a+=b is same as a=a+b
-=	Subtracts right operand from the left operand and assign the result to left operand	a-=b is same as a=a-b
*=	Mutiply left operand with the right operand and assign the result to left operand	a*=b is same as a=a*b
/=	Divides left operand with the right operand and assign the result to left operand	a/=b is same as a=a/b
%=	Calculate modulus using two operands and assign the result to left operand	a%=b is same as a=a%b

Using Assignment Operators

Example:

In the example below, all assignment operators have right to left associativity.

```
class AssignmentOps{
public static void main(String as[]){
        int a = 30; int b = 10; int c = 0; c = a + b;
        System.out.println("c = a + b = " + c );
        c += a ;
        System.out.println("c += a = " + c ); c -= a ;
        System.out.println("c -= a = " + c ); c *= a ;
        System.out.println("c *= a = " + c ); a = 20;
        c = 25; c /= a ; System.out.println("c /= a = " + c );
        a = 20; c = 25; c %= a ;  System.out.println("c %= a = " + c );
        c <<= 2 ;
        System.out.println("c <<= 2 = " + c );
        c >>= 2 ; System.out.println("c >>= 2 = " + c );
        c >>= 2 ; System.out.println("c >>= 2 = " + c );
        c &= a ; System.out.println("c &= a = " + c );
        c ^= a ; System.out.println("c ^= a = " + c );
        c |= a ; System.out.println("c |= a = " + c );
} }
```

6. Conditional operator

Conditional operators are used to evaluate a condition that's applied to one or two Boolean expressions. It is also known as Ternary operator.

1.Ternary Operator ? : Operator

It is actually the if condition that we use in decision making, but using conditional operator, we turn the if condition statement into a short and simple operator.

The syntax of a conditional operator is :

```
Expression 1 ? expression 2: expression 3
```

Explanation

1. The question mark **"?"** in the syntax represents the **if** part.
2. The first expression (expression 1) generally returns either true or false, based on which it is decided whether (expression 2) will be executed or (expression 3)
3. If (expression 1) returns true then the expression on the left side of **" : "** i.e (expression 2) is executed.
4. If (expression 1) returns false then the expression on the right side of **" : "** i.e (expression 3) is executed.

Example of a program

```
class ConditionalOps{
public static void main(String as[]){
int k,  a=20, l=10;
     k = (l == 5) ? 100: 40;
     System.out.println( "False therefore K: " + k );

     k = (a > l) ? 100: 40;
     System.out.println( "True, therefore K: " + k );
}
}
```

Using ternary Operator (?:)

Output

```
====== Conditional Operator Results ======
False, therefore K : 40
True, therefore K : 100
```

7. instanceOf operator

is a java **keyword** and used to test whether the given **reference belongs to provided type** or not. Type can be a class or interface. **It returns either true or false**.

Example:

Here, we created a string reference variable that stores “KUMU”. Since it stores string value so we test it using isinstance operator to check whether it belongs to string class or not. See the below example.

Sample program

```
class instanceofOps{
public static void main(String as[]){
String a = “KUMU";
boolean b = a
instanceof String;
System.out.println("It's  " + b + " " + a + " is a
String"  );
}
}
```

Output

```
===== instanceof Operator Results =====
It's  true KUMU is a string
```

Dealing with static and Initializer Blocks

Static Blocks in Java

Static block is a set of statements which are executed by the JVM before the main method. If we want to perform any task at the time of class loading, we can define that task inside the static block. This task will be executed at the time of class loading.

In a class, any number of a static block can be defined, and this static blocks will be executed from top to bottom first then later main() method statements are executed.

Note! We use **static** keyword help us manage memory.

Static Blocks+

Syntax:

```
static {
// statements….

}
```

Vs

```
pubic static void main(){
// statements….

}
```

Note! Static block can be written either before or after the main method but still JVM executes its statements before the statements in the main() method.

Note! 3 static blocks in this program

```
class StaticPro {
static {
     System.out.println("I am the static block above main()");
}
     public static void main(String as[]) {
     System.out.println("I am the main() method executed last");
}
static {
     System.out.println("I am the first static block below main()
");
}
static {
System.out.println("I am the second static block Below main");
System.out.println("********************");
} }
```

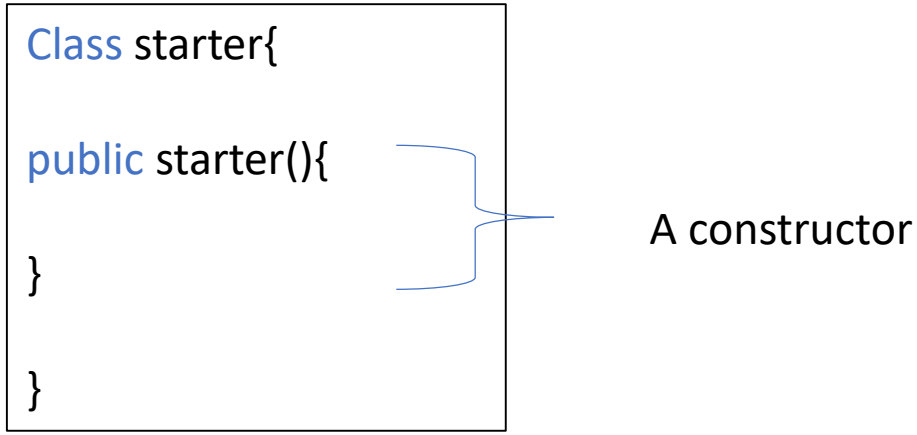
Static block - Output

Static blocks executes in order of their sequence. First static block will execute first through up to the last static block then the main() method statements will be executed.

```
run:
I am the static block above main()
I am the first static block below main()
==========================
I am the second static block below main()
==========================
I am the main() method executed last although I second in hirachy
```

Initializer Block in Java

In Java, the initializer Block is used to initialize **instance data members**. The initializer block is executed whenever an object is created. The Initializer block is copied into Java compiler and then to every constructor. **A constructor** is a method whose name is the same as the name of the class**. E.g.**



Like static blocks, the initialization block is also executed before the code in the constructor.

Sample code

Initializer block

```
class InitializerPro{
   {
   System.out.println("===Welcome to initializer block===");
   System.out.println("Initializer block executes before the constructor");
   }
      public InitializerPro(){
      System.out.println("I am the constructor running here");
      }
      public static void main(String as[]){
            InitializerPro obj = new InitializerPro();
            System.out.println("I am the main() method");
      }
}
```

Output

```
=== Welcome to initializer block ===
Initializer block executes before the constructor block
I am the constructor Runing here
I am the main() method
```

Note that the Initializer block is executed before the constructor.

Example using static and initializer block

```
class two{
      static {
      System.out.println("I am the parant Static block");
      }
      {
      System.out.println("I am the parent initializer block");
      }
      two() {
      System.out.println("I am the parent constructor");
      }
}
```

Example using static and initializer block

```
public class one extends two {
    static {
        System.out.println(“I am the Child satic block");
    }
    one() {
        System.out.println(“I am the  constructor of child class");
    }
    {
     System.out.println(“I am the child initialization block");
    }
    public static void main(String[] args){
        new one();

System.out.println("inside main");
    }
}
```

Output

```
I am the parant Static block
I am the Child Static block
I am the parent initializer block
I am the parent constructor
I am the child initializer block
I am the  constructor of child class
I am the child class main() method
```

User Input/Output

User Input/Output

We shall deal with the most back input and out functions(Scanner() and print or println()), which will help us build some meaningful programs. However, we will look at detailed java input stream and output stream at a later stage.

All along, we have been using System.out.println() method which we said is an output stream that helps us to output some information on the screen. Its now time to introduce ourselves to the user input stream to allow us build programs that users will be able to interact with by means of input.

User Input (Scanner)

The Scanner class is used to get user input, and it is found in the java.util package.

To use the Scanner class, create an object of the class and use any of the available methods found in the Scanner class documentation. In our example, we will use the nextLine() method, which is used to read Strings:

Example

User Input sample program

```
import java.util.Scanner;  // Import the Scanner class

class inputPro {

  public static void main(String[] args) {
    String name;       // variable declaration
    Scanner ob = new Scanner(System.in);  // Create a Scanner object

    System.out.println(“What is your name?");

    name = ob.nextLine();  // Read user input

    System.out.println(“Welcome " + name);  // Output user input

  }

}
```

Output

```
run:
What is your Surname?
```

Waiting for user input

```
What is your Surname?
James Mark
```

User entered names

```
What is your Surname?
James Mark
Welcome James Mark
BUILD SUCCESSFUL (total time: 1 minute 51 seconds)
```

After entering names,
user pressed Enter Key

Input Types

In the example above, we used the `nextLine()` method, which is used to read Strings. To read other types, look at the table below:

Method	Description
`nextBoolean()`	Reads a `boolean` value from the user
`nextByte()`	Reads a `byte` value from the user
`nextDouble()`	Reads a `double` value from the user
`nextFloat()`	Reads a `float` value from the user
`nextInt()`	Reads a `int` value from the user
`nextLine()`	Reads a `String` value from the user
`nextLong()`	Reads a `long` value from the user
`nextShort()`	Reads a `short` value from the user

In the example below, we use different methods to read data of various types:

Example

```
import java.util.Scanner;

class inputPro1 {
  public static void main(String[] args) {
    Scanner myObj = new Scanner(System.in);

    System.out.println("Enter name, age and salary");
    // String input
    String name = myObj.nextLine();

    // Numerical input
    int age = myObj.nextInt();
    double salary = myObj.nextDouble();

    // Output input by user
    System.out.println("Name: " + name);
    System.out.println("Age: " + age);
    System.out.println("Salary: " + salary);
  } }
```

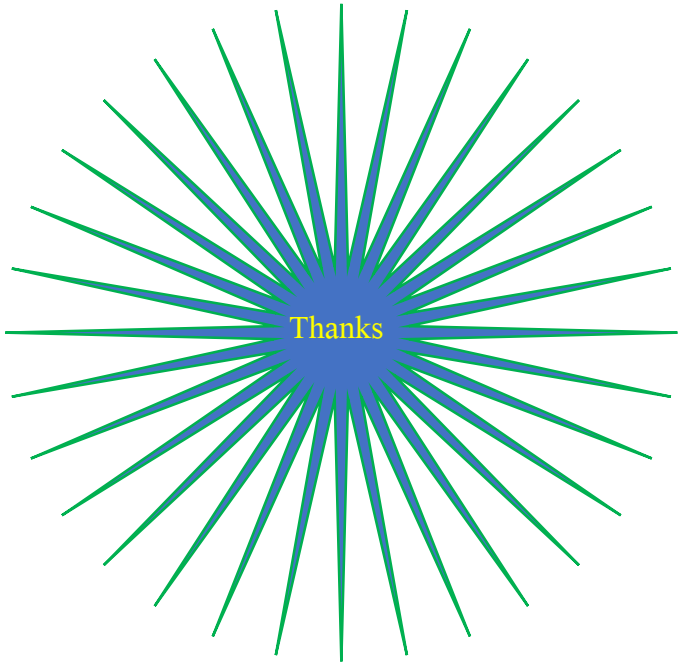
Review Questions

1. Explain the difference between Static Block and Initializer Block
2. With example, discuss any three types of operators you know.

Summary

1. Type Casting
2. Operators in Java,
3. Static and Initializer Block,
4. User input/output

Thank you for Listening



Reference

Leahy, P. (2019, July 12). *Discover how conditional operators are used in Java*. ThoughtCo. Retrieved April 8, 2022, from https://www.thoughtco.com/conditional-operator-2034056

Variable in Java. Studytonight.com. (n.d.). Retrieved April 1, 2022, from https://www.studytonight.com/java/variable.php

What is IDE or Integrated Development Environments? Veracode. (n.d.). Retrieved March 24, 2022, from https://www.veracode.com/security/integrated-development-environment

Java type casting. (n.d.). Retrieved April 8, 2022, from https://www.w3schools.com/java/java_type_casting.asp

Type casting in Java. Studytonight.com. (n.d.). Retrieved April 8, 2022, from https://www.studytonight.com/java/type-casting-in-java.php