## Java Type Casting

Type casting is when you assign a value of one primitive data type to another type.

In Java, there are two types of casting:

* ****Widening Casting**** (automatically) - converting a smaller type to a larger type size  
  byte -> short -> char -> int -> long -> float -> double
* ****Narrowing Casting**** (manually) - converting a larger type to a smaller size type  
  double -> float -> long -> int -> char -> short -> byte

## Widening Casting

Widening casting is done automatically when passing a smaller size type to a larger size type:

### Example

public class MyClass {

public static void main(String[] args) {

int myInt = 9;

double myDouble = myInt; // Automatic casting: int to double

System.out.println(myInt); // Outputs 9

System.out.println(myDouble); // Outputs 9.0

}}

## Narrowing Casting

Narrowing casting must be done manually by placing the type in parentheses in front of the value:

### Example

public class MyClass {

public static void main(String[] args) {

double myDouble = 9.78;

int myInt = (int) myDouble; // Manual casting: double to int

System.out.println(myDouble); // Outputs 9.78

System.out.println(myInt); // Outputs 9

}}

## Widening Type Casting

In ****Widening Type Casting****, Java automatically converts one data type to another data type.

### **Example: Converting int to double**

class Main {

public static void main(String[] args) {

// create int type variable

int num = 10;

System.out.println("The integer value: " + num);

// convert into double type

double data = num;

System.out.println("The double value: " + data);

}

}

****Output****

The integer value: 10

The double value: 10.0

In the above example, we are assigning the int type variable named num to a double type variable named data.

Here, the Java first converts the int type data into the double type. And then assign it to the double variable.

In the case of ****Widening Type Casting****, the lower data type (having smaller size) is converted into the higher data type (having larger size). Hence there is no loss in data. This is why this type of conversion happens automatically.

****Note****: This is also known as ****Implicit Type Casting****.

## Narrowing Type Casting

In ****Narrowing Type Casting****, we manually convert one data type into another using the parenthesis.

### **Example: Converting double into an int**

class Main {

public static void main(String[] args) {

// create double type variable

double num = 10.99;

System.out.println("The double value: " + num);

// convert into int type

int data = (int)num;

System.out.println("The integer value: " + data);

}

}

****Output****

The double value: 10.99

The integer value: 10

In the above example, we are assigning the double type variable named num to an int type variable named data.

Notice the line,

int data = (int)num;

Here, the int keyword inside the parenthesis indicates that that the num variable is converted into the int type.

In the case of ****Narrowing Type Casting****, the higher data types (having larger size) are converted into lower data types (having smaller size). Hence there is the loss of data. This is why this type of conversion does not happen automatically.

****Note****: This is also known as ****Explicit Type Casting****.

Let's see some of the examples of other type conversions in Java.

## Example 1: Type conversion from int to String

class Main {

public static void main(String[] args) {

// create int type variable

int num = 10;

System.out.println("The integer value is: " + num);

// converts int to string type

String data = String.valueOf(num);

System.out.println("The string value is: " + data);

}

}

****Output****

The integer value is: 10

The string value is: 10

In the above program, notice the line

String data = String.valueOf(num);

Here, we have used the valueOf() method of the [Java String class](https://www.programiz.com/java-programming/string" \o "Java String) to convert the int type variable into a string.

## Example 2: Type conversion from String to int

class Main {

public static void main(String[] args) {

// create string type variable

String data = "10";

System.out.println("The string value is: " + data);

// convert string variable to int

int num = Integer.parseInt(data);

System.out.println("The integer value is: " + num);

}

}

****Output****

The string value is: 10

The integer value is: 10

In the above example, notice the line

int num = Integer.parseInt(data);

Here, we have used the parseInt() method of the Java Integer class to convert a string type variable into an int variable.

****Note****: If the string variable cannot be converted into the integer variable then an exception named NumberFormatException occurs.

## Java Operators

Operators are used to perform operations on variables and values.

In the example below, we use the + ****operator**** to add together two values:

### Example

int x = 100 + 50;

Although the + operator is often used to add together two values, like in the example above, it can also be used to add together a variable and a value, or a variable and another variable:

### Example

int sum1 = 100 + 50; // 150 (100 + 50)int sum2 = sum1 + 250; // 400 (150 + 250)int sum3 = sum2 + sum2; // 800 (400 + 400)

Java divides the operators into the following groups:

* Arithmetic operators
* Assignment operators
* Comparison operators
* Logical operators
* Bitwise operators

## Arithmetic Operators

Arithmetic operators are used to perform common mathematical operations.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Operator** | **Name** | **Description** | **Example** | **Try it** |
| + | Addition | Adds together two values | x + y | [Try it »](https://www.w3schools.com/java/showjava.asp?filename=demo_oper_add" \t "https://www.w3schools.com/java/_blank) |
| - | Subtraction | Subtracts one value from another | x - y | [Try it »](https://www.w3schools.com/java/showjava.asp?filename=demo_oper_sub" \t "https://www.w3schools.com/java/_blank) |
| \* | Multiplication | Multiplies two values | x \* y | [Try it »](https://www.w3schools.com/java/showjava.asp?filename=demo_oper_mult" \t "https://www.w3schools.com/java/_blank) |
| / | Division | Divides one value by another | x / y | [Try it »](https://www.w3schools.com/java/showjava.asp?filename=demo_oper_div" \t "https://www.w3schools.com/java/_blank) |
| % | Modulus | Returns the division remainder | x % y | [Try it »](https://www.w3schools.com/java/showjava.asp?filename=demo_oper_mod" \t "https://www.w3schools.com/java/_blank) |
| ++ | Increment | Increases the value of a variable by 1 | ++x | [Try it »](https://www.w3schools.com/java/showjava.asp?filename=demo_oper_inc" \t "https://www.w3schools.com/java/_blank) |
| -- | Decrement | Decreases the value of a variable by 1 | --x | [Try it »](https://www.w3schools.com/java/showjava.asp?filename=demo_oper_dec" \t "https://www.w3schools.com/java/_blank) |

## Java Assignment Operators

Assignment operators are used to assign values to variables.

In the example below, we use the ****assignment**** operator (=) to assign the value ****10**** to a variable called ****x****:

### Example

int x = 10;

The ****addition assignment**** operator (+=) adds a value to a variable:

### Example

int x = 10;

x += 5;

A list of all assignment operators:

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Example** | **Same As** | **Try it** |
| = | x = 5 | x = 5 | [Try it »](https://www.w3schools.com/java/showjava.asp?filename=demo_oper_ass1" \t "https://www.w3schools.com/java/_blank) |
| += | x += 3 | x = x + 3 | [Try it »](https://www.w3schools.com/java/showjava.asp?filename=demo_oper_ass2" \t "https://www.w3schools.com/java/_blank) |
| -= | x -= 3 | x = x - 3 | [Try it »](https://www.w3schools.com/java/showjava.asp?filename=demo_oper_ass3" \t "https://www.w3schools.com/java/_blank) |
| \*= | x \*= 3 | x = x \* 3 | [Try it »](https://www.w3schools.com/java/showjava.asp?filename=demo_oper_ass4" \t "https://www.w3schools.com/java/_blank) |
| /= | x /= 3 | x = x / 3 | [Try it »](https://www.w3schools.com/java/showjava.asp?filename=demo_oper_ass5" \t "https://www.w3schools.com/java/_blank) |
| %= | x %= 3 | x = x % 3 | [Try it »](https://www.w3schools.com/java/showjava.asp?filename=demo_oper_ass6" \t "https://www.w3schools.com/java/_blank) |
| &= | x &= 3 | x = x & 3 | [Try it »](https://www.w3schools.com/java/showjava.asp?filename=demo_oper_ass7" \t "https://www.w3schools.com/java/_blank) |
| |= | x |= 3 | x = x | 3 | [Try it »](https://www.w3schools.com/java/showjava.asp?filename=demo_oper_ass8" \t "https://www.w3schools.com/java/_blank) |
| ^= | x ^= 3 | x = x ^ 3 | [Try it »](https://www.w3schools.com/java/showjava.asp?filename=demo_oper_ass9" \t "https://www.w3schools.com/java/_blank) |
| >>= | x >>= 3 | x = x >> 3 | [Try it »](https://www.w3schools.com/java/showjava.asp?filename=demo_oper_ass10" \t "https://www.w3schools.com/java/_blank) |
| <<= | x <<= 3 | x = x << 3 | [Try it »](https://www.w3schools.com/java/showjava.asp?filename=demo_oper_ass11" \t "https://www.w3schools.com/java/_blank) |

## Java Comparison Operators

Comparison operators are used to compare two values:

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Name** | **Example** | **Try it** |
| == | Equal to | x == y | [Try it »](https://www.w3schools.com/java/showjava.asp?filename=demo_oper_compare1" \t "https://www.w3schools.com/java/_blank) |
| != | Not equal | x != y | [Try it »](https://www.w3schools.com/java/showjava.asp?filename=demo_oper_compare2" \t "https://www.w3schools.com/java/_blank) |
| > | Greater than | x > y | [Try it »](https://www.w3schools.com/java/showjava.asp?filename=demo_oper_compare3" \t "https://www.w3schools.com/java/_blank) |
| < | Less than | x < y | [Try it »](https://www.w3schools.com/java/showjava.asp?filename=demo_oper_compare4" \t "https://www.w3schools.com/java/_blank) |
| >= | Greater than or equal to | x >= y | [Try it »](https://www.w3schools.com/java/showjava.asp?filename=demo_oper_compare5" \t "https://www.w3schools.com/java/_blank) |
| <= | Less than or equal to | x <= y | [Try it »](https://www.w3schools.com/java/showjava.asp?filename=demo_oper_compare6" \t "https://www.w3schools.com/java/_blank) |

## Java Logical Operators

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Operator** | **Name** | **Description** | **Example** |  |
| && | Logical and | Returns true if both statements are true | x < 5 &&  x < 10 |  |
| || | Logical or | Returns true if one of the statements is true | x < 5 || x < 4 |  |
| ! | Logical not | Reverse the result, returns false if the result is true | !(x < 5 && x < 10) |  |

Logical operators are used to determine the logic between variables or values:

## Java Bitwise Operators

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Operator** | **Description** | **Example** | **Same as** | **Result** | **Decimal** |
| & | AND - Sets each bit to 1 if both bits are 1 | 5 & 1 | 0101 & 0001 | 0001 | 1 |
| | | OR - Sets each bit to 1 if any of the two bits is 1 | 5 | 1 | 0101 | 0001 | 0101 | 5 |
| ~ | NOT - Inverts all the bits | ~ 5 | ~0101 | 1010 | 10 |
| ^ | XOR - Sets each bit to 1 if only one of the two bits is 1 | 5 ^ 1 | 0101 ^ 0001 | 0100 | 4 |
| << | Zero-fill left shift - Shift left by pushing zeroes in from the right and letting the leftmost bits fall off | 9 << 1 | 1001 << 1 | 0010 | 2 |
| >> | Signed right shift - Shift right by pushing copies of the leftmost bit in from the left and letting the rightmost bits fall off | 9 >> 1 | 1001 >> 1 | 1100 | 12 |
| >>> | Zero-fill right shift - Shift right by pushing zeroes in from the left and letting the rightmost bits fall off |  |  |  |  |

Bitwise operators are used to perform binary logic with the bits of an integer or long integer.