Type Conversion and Type Casting

What is Type conversion in Java?

The process of converting a value from one data type to another is known as **type conversion in Java.**

Type conversion is also known as type casting in java or simply ‘casting’.

If two data types are compatible with each other, Java will perform such conversion automatically or implicitly for you.

We can easily convert a primitive data type into another primitive data type by using type casting.

Similarly,it is also possible to convert a non-primitive data type (referenced data type) into another non-primitive data type by using type casting.

But we cannot convert a primitive data type into an advanced (referenced) data type by using type casting. For this case, we will have to use methods of Java **Wrapper classes**.

Types of Casting in Java

Two types of casting are possible in Java are as follows:

1. Implicit type casting (also known as automatic type conversion)
2. Explicit type casting

Implicit Type Casting in Java

Automatic conversion (casting) done by Java compiler internally is called implicit conversion or implicit type casting in java.

Implicit casting is performed to convert a lower data type into a higher data type. It is also known as **automatic type promotion in Java**.

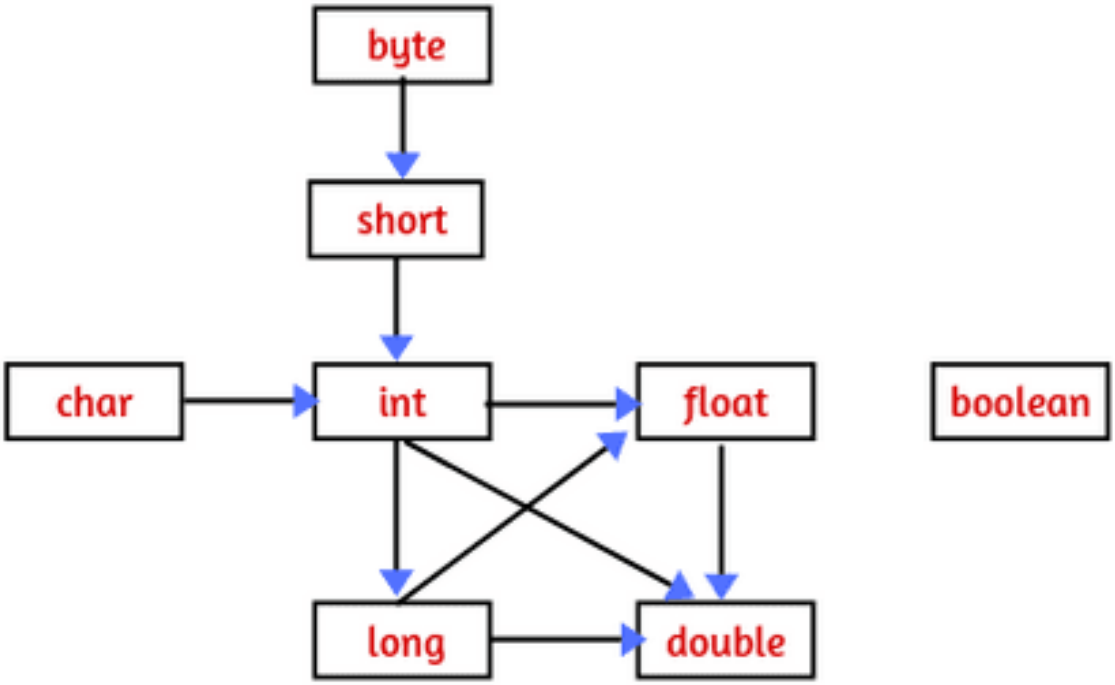
```
int x = 20;
long y = x; // Automatic conversion
byte z = x; // Type mismatch: cannot convert from int to byte.
```

Automatic Type Conversion in Java with Example

When we perform arithmetic or mathematical operations with operands of different types, Java compiler performs an implicit conversion internally.

In other words, Java compiler automatically converts ‘lower’ type to ‘higher’ type before the operation proceeds and gives the result of higher type.

Lower types are those types that consume less memory and store less number of digits in value. Higher types use more memory and store more number of digits.



Automatic Type Promotion Rules in Expression

1. If byte, short, and int are used in a mathematical expression, Java always converts the result into an int.

2. If a single long is used in the expression, the whole expression is converted to long.

3. If a float operand is used in an expression, the whole expression is converted to float.

4. If any operand is double, the result is promoted to double.

5. Boolean values cannot be converted to another type.

6. Conversion from float to int causes truncation of the fractional part which represents the loss of precision. Java does not allow this.

7. Conversion from double to float causes rounding of digits that may cause some of the value’s precision to be lost.

8. Conversion from long to int is also not possible. It causes the dropping of the excess higher order bits.

```
package typeConversion;
public class AutoTypeConversion
{
   int x = 20;
   double y = 40.5;
   long p = 30;
   float q = 10.60f;
void sum()
{
    int a = x + y; // (1) // Error; cannot convert from double to int.
    double b = x + y;

   System.out.println("Sum of two numbers: " +a);
   System.out.println("Sum of two numbers: " +b);
 }
void sub()
{
    long c = p - q;  // (2) // Error; cannot convert from float to long.
    float d = p - q;
    System.out.println("Subtraction of two numbers: " +c);
    System.out.println("Subtraction of two numbers: " +d);
  }
public static void main(String[] args)
{
   AutoTypeConversion obj = new AutoTypeConversion();
    obj.sum(); // Calling sum method.
    obj.sub(); // Calling sub method.
  }
}
```

```
package typeConversion;
public class PromoteTest
{
  public static void main(String[] args)
  {
    byte b = 42;
    char c = 'a';
    short s = 1024;
    int i = 50000;
```

```
    float f = 5.67f;
    double d = .1234;

// Expression:
   double result = (f * b) + (i / c) - (d * s);

//Result after all the promotions are done.
   System.out.println("result = " + result);
  }
 }
```

When takes place Automatic type conversion in Java?

An automatic type conversion takes place in Java if these two conditions are met:

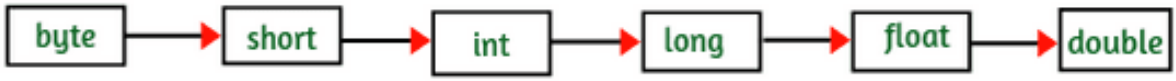
- The data types are compatible with each other.
- The destination type is bigger than the source type.

When
these two conditions are met, a widening conversion will take place.
The process of converting lower data type into higher data type is
called **widening in java**.

Widening is safe because there is no loss of data or accuracy or precision. For example, the int type is always large enough to hold all bytes value. Therefore, no explicit casting is needed.

The widening conversion (i.e. automatic type conversion) is possible in Java when the numeric type (including integer and floating-point type) is compatible with each other.

But no widening conversion is supported from numeric type to char or boolean because the numeric type is incompatible with char or boolean. Also char and boolean are incompatible with each other.



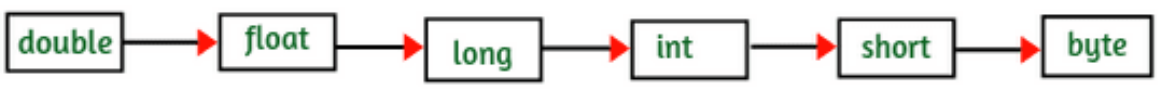
```
package typeConversion;
public class WideningConversionEx
{
 public static void main(String[] args)
 {
   int a = 50;
   double d = 100;

    long l = a; // Automatic type conversion
    int i = l; // Type mismatch: cannot convert from long to int.


     float f = l; // Automatic type conversion
     int i1 = f; // Type mismatch: cannot convert from float to int.
     float f1 = d; // Type mismatch.
     double d1 = f;

   System.out.println("Int value "+a);
   System.out.println("Long value "+l);
   System.out.println("Float value "+f);
   System.out.println("Double value " +d1);
  }
}
```

Explicit Type casting (Narrowing conversion) in Java

create a conversion between two incompatible types. This kind of conversion is also known as **narrowing conversion in java**.

The conversion of a higher data type into a lower data type is called narrowing conversion.

Since this type of conversion is performed by the programmer, not by the compiler automatically, therefore, it is also called **explicit type casting in java**. It is done to convert from a higher data type to a lower data type.




```
int x;
double y = 9.99;
```

```
x = (int)y;   // It will compile in Java and the resulting value will simply be 9.

double d = 100.9;
long l = (long)d; // Explicit type casting.

int x;
float y = 27.6f;
x = (int)(y + 0.5);

int a = 66;
char ch = (char)a; // ch stores 'B'.
```

```
package typeConversion;
public class ExplicitTest
{
   double d = 100.04;
  void conversion()
  {
// explicit type casting
    long l = (long)d;
    int i = (int)l;
  System.out.println("Double value "+d); // fractional part lost.
  System.out.println("Long value "+l); // fractional part lost.
  System.out.println("Int value "+i);
}
public static void main(String[] args)
{
   ExplicitTest obj = new ExplicitTest();
    obj.conversion();
  }
}
```

```
package typeConversion;
public class ExplicitTest
{
  public static void main(String[] args)
  {
    byte b;
    int i = 257;
    double d = 323.142;
   System.out.println("Conversion of int to byte.");
   b = (byte) i;
   System.out.println("i = " + i + " b = " + b);
   System.out.println("Conversion of double to int.");

   i = (int) d;
   System.out.println("d = " + d + " b = " + i);
   System.out.println("Conversion of double to byte.");

   b = (byte) d;
   System.out.println("d = " + d + " b = " + b);
  }
}
```

Disadvantage of Explicit Type casting in Java

There are the following disadvantages of using type casting in Java.

- When you will use type casting in Java, you can lose some information or data.
- Accuracy can be lost while using type casting.
- When a double is cast to an int, the fractional part of a double is discarded which causes the loss of fractional part of data.

Difference between Implicit casting and Explicit casting in Java

The differences between implicit casting and explicit casting in Java are as follows:

1.Implicit type casting is done internally by java compiler whereas, explicit type casting is done by the programmer. Java compiler does not perform it automatically.

2. In explicit casting, cast operator is needed whereas, no need for any operator needs in the case of implicit type casting.

3.If we perform explicit type casting in a program, we can lose information or data but in the case of implicit type casting, there is no loss of data.

4. Accuracy is not maintained in explicit type casting whereas, there is no issue of accuracy in implicit type conversion.

5. Implicit type conversion is safe but explicit type casting is not safe.