# Casting

**To Cast:** To convert the datatype of an expression ie cast it into another datatype.

**Automatic Type Conversion:** This is when the compiler automatically converts one data type to another.

Example

```
char ch;
ch = 65;        /* 65 (an integer) is automatically cast into  the
                   character with Unicode value 65 */
System.out.println (ch);  //prints A;  A has the Unicode value 65
```

Automatic conversion will only take place when there is no loss of data. If a conversion results in the loss of precision, as in an int value converted to a short, then the compiler will issue an error message unless an explicit cast is made.

Expressions can promote (convert) to a wider type (one with more memory; more bits) without an explicit cast. For example:

```
int i = 12;
long j;   // Literals are int types so require L suffix
j = i;    // OK; automatically converts int to long
```

**Explicit Type Conversion**: This is when the programmer forces a conversion.

**Syntax** To convert type1 data into type2 data, put the type1 keyword in brackets in front of the type2 data.

For example, to convert floating point data to integer data:

```
int i ;
float f= 5.6F;
i = (int) f;        // Cast float as int (truncates the decimals)
System.out.println(i);    //prints 5
```

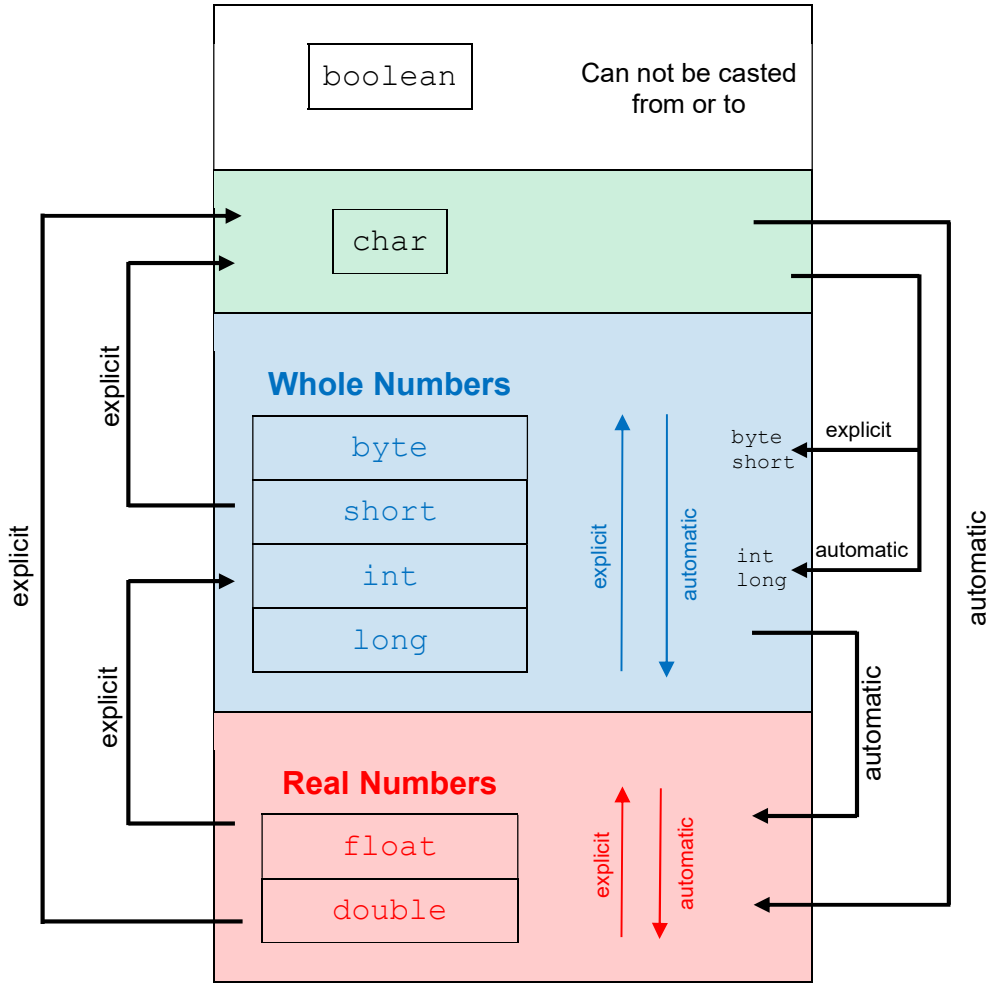
However, you can not assign a value to a more narrow type without an explicit cast:

```
int  i=12;
long j = 123L;
i = j;         // Error in assigning long to int
i=(int)j;     // OK; converts long to int
```

**Summary**

A data type with lower precision (fewer bits) can be converted to a type of higher precision without explicit casting. To convert a higher precision type to a lower precision, however, an explicit cast is required, or the compiler will flag an error.

Note that when you cast a value of a wider type down to a narrower type, such as an int value to a byte variable, the upper bytes will be truncated. That is, the lowest order byte in the int value will be copied to the byte value. Data may be lost.



### Primitive Type Conversion Table

Below is a table that indicates to which of the other primitive types you can cast a given primitive data type. The symbol **E** indicates that an explicit cast is required since the precision is decreasing. The symbol **A** indicates that the precision is increasing so an automatic cast occurs without the need for an explicit cast. **N** indicates that the conversion is not allowed.

| | int | long | float | double | char | short | byte | boolean |
|---|---|---|---|---|---|---|---|---|
| **int** | - | A | A* | A | E | E | E | N |
| **long** | E | - | A* | A* | E | E | E | N |
| **float** | E | E | - | A | E | E | E | N |
| **double** | E | E | E | - | E | E | E | N |
| **char** | A | A | A | A | - | E | E | N |
| **byte** | A | A | A | A | E | A | - | N |
| **short** | A | A | A | A | E | - | E | N |
| **boolean** | N | N | N | N | N | N | N | - |

The * asterisk indicates that the least significant digits may be lost in the conversion even though the target type allows for bigger numbers. For example, a large value in an int type value that uses all 32 bits will lose some of the lower bits when converted to float since the exponent uses 8 bits of the 32 provided for float values.

### Mixed Types in an Expression

If an expression holds a **mix** of types, the lower precision or narrower value operand is converted to a higher precision or wider type. This result then must be cast if it goes to a lower precision type:

```
float x,y=3;
int j,i=3;
x= i*y;        // OK since i will be promoted to float
j= i*y;        // Error since result is a float value
j= (int)(i*y) // OK
```

The process of converting a value to a wider or higher precision integer or floating point type is called "numeric promotion". The following rules apply for promotion in an expression of two operands, as in x+i:

- If either operand is of type double, the other is converted to double.
- Otherwise, if either operand is of type float, the other is converted to float.
- Otherwise, if either operand is of type long, the other is converted to long.
- Otherwise, both operands are converted to type int.

Note that char data can be cast to other integer values but may result in unexpected values since it is unsigned while the other integer types are signed.