Documentation

The Java™ Tutorials

Trail: Learning the Java Language
Lesson: Numbers and Strings
Section: Numbers

The Java Tutorials have been written for JDK 8. Examples and practices described in this page don't take advantage of improvements introduced in later releases and might use technology no longer available.
See Java Language Changes for a summary of updated language features in Java SE 9 and subsequent releases.
See JDK Release Notes for information about new features, enhancements, and removed or deprecated options for all JDK releases.

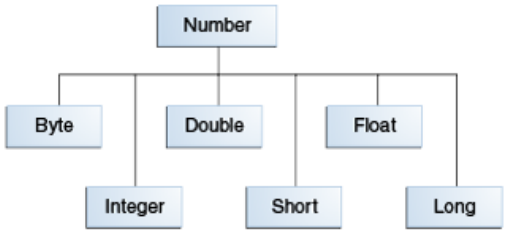
The Numbers Classes

When working with numbers, most of the time you use the primitive types in your code. For example:

```
int i = 500;
float gpa = 3.65f;
byte mask = 0x7f;
```

There are, however, reasons to use objects in place of primitives, and the Java platform provides *wrapper* classes for each of the primitive data types. These classes "wrap" the primitive in an object. Often, the wrapping is done by the compiler—if you use a primitive where an object is expected, the compiler *boxes* the primitive in its wrapper class for you. Similarly, if you use a number object when a primitive is expected, the compiler *unboxes* the object for you. For more information, see Autoboxing and Unboxing

All of the numeric wrapper classes are subclasses of the abstract class `Number`:



Note: There are four other subclasses of `Number` that are not discussed here. `BigDecimal` and `BigInteger` are used for high-precision calculations. `AtomicInteger` and `AtomicLong` are used for multi-threaded applications.

There are three reasons that you might use a `Number` object rather than a primitive:

1. As an argument of a method that expects an object (often used when manipulating collections of numbers).
2. To use constants defined by the class, such as `MIN_VALUE` and `MAX_VALUE`, that provide the upper and lower bounds of the data type.
3. To use class methods for converting values to and from other primitive types, for converting to and from strings, and for converting between number systems (decimal, octal, hexadecimal, binary).

The following table lists the instance methods that all the subclasses of the `Number` class implement.

Methods Implemented by all Subclasses of Number

Method	Description

`byte byteValue()`
`short shortValue()`
`int intValue()`
`long longValue()`
`float floatValue()`
`double doubleValue()`
	Converts the value of this `Number` object to the primitive data type returned.

`int compareTo(Byte anotherByte)`
`int compareTo(Double anotherDouble)`
`int compareTo(Float anotherFloat)`
`int compareTo(Integer`
	Compares this `Number` object to the argument.

`anotherInteger)`
`int compareTo(Long anotherLong)`
`int compareTo(Short anotherShort)`	

`boolean equals(Object obj)`	Determines whether this number object is equal to the argument. The methods return `true` if the argument is not `null` and is an object of the same type and with the same numeric value. There are some extra requirements for `Double` and `Float` objects that are described in the Java API documentation.

Each `Number` class contains other methods that are useful for converting numbers to and from strings and for converting between number systems. The following table lists these methods in the `Integer` class. Methods for the other `Number` subclasses are similar:

Conversion Methods, `Integer` Class

Method	Description
`static Integer decode(String s)`	Decodes a string into an integer. Can accept string representations of decimal, octal, or hexadecimal numbers as input.
`static int parseInt(String s)`	Returns an integer (decimal only).
`static int parseInt(String s, int radix)`	Returns an integer, given a string representation of decimal, binary, octal, or hexadecimal (`radix` equals 10, 2, 8, or 16 respectively) numbers as input.
`String toString()`	Returns a `String` object representing the value of this `Integer`.
`static String toString(int i)`	Returns a `String` object representing the specified integer.
`static Integer valueOf(int i)`	Returns an `Integer` object holding the value of the specified primitive.
`static Integer valueOf(String s)`	Returns an `Integer` object holding the value of the specified string representation.
`static Integer valueOf(String s, int radix)`	Returns an `Integer` object holding the integer value of the specified string representation, parsed with the value of radix. For example, if s = "333" and radix = 8, the method returns the base-ten integer equivalent of the octal number 333.

Previous page: Numbers
Next page: Formatting Numeric Print Output