Numbers using Primitive Types

and Reference Class Types

**Primitive Data Types:**

That value is stored as in in the main memory

Issues of overflow and precision

The Java statement, int num; , stores 0 in memory

**Reference Data Types:**

The reference/the location where it is in memory is stored in main

Issue of memory

The Java statement, Integer num; , stores null in memory in its memory location

Read: <http://www.drdobbs.com/jvm/a-modern-primitive-discussion/232601450>

**Automatic Conversion between Primitive Types and Wrapper Class Types**

A data type is a set of values and a set of operations defined on those values. The primitive data types that you have been using are supplemented in Java by extensive libraries of reference types that are tailored for a large variety of applications. The behavior of a data type in an application programming interface (API).

A primitive type can be automatically converted to an object using a wrapper class, and vice versa, depending on the context, using the API (the library). The wrapper classes provide constructors, constants, and conversion methods for manipulating various data types.

\*\*\* the wrapper classes do NOT have no-args constructors\*\*\*

\*\*\* all instances of the wrapper classes are immutable\*\*\*

**Converting Strings into numbers:**

Note: the parseDataType(\*) and the valueOf(\*) methods, where \* can be just the String, or the String and specifying the numeric counting system

* Binary = 2
* Octal = 8
* Decimal = 10
* Hexadecimal = 16

**Converting numbers into Strings:**

* The argument is converted and returned as a String instance. If the number is negative, the sign will be preserved.

int a = 1234;

int b = -1234;

String str1 = Integer.toString(a);

String str2 = Integer.toString(b);

System.out.println("String str1 = " + str1);

System.out.println("String str2 = " + str2);

* This method uses instance of Integer class to invoke it’s toString() method.

int a = 1234;

int b = -1234;

String str1 = Integer.toString(a);

String str2 = Integer.toString(b);

System.out.println("String str1 = " + str1);

System.out.println("String str2 = " + str2);

int d = 1234;

Integer obj = new Integer(d);

String str4 = obj.toString();

System.out.println("String str4 = " + str4);

* Using the DecimalFormat.format() method, you can specify the number of decimal places and comma separator for readability.

int e = 12345;

DecimalFormat df = new DecimalFormat("#,###");

String Str5 = df.format(e);

System.out.println(Str5);

* Converting with a specific numbering system.

int i = 255;

String binaryString = Integer.toBinaryString(i);

System.out.println(binaryString);

**Boxing**: converting a primitive value to a wrapper object

**Unboxing**: converting a wrapper object to a primitive type

**Autoboxing** & **Autounboxing** a feature added to JDK 5: The compiler will automatically box a primitive value that appears in a context requiring an object and will unbox an object that appears in a context requiring a primitive value

int n1 = 100;

Integer n2 = new Integer(100);

Using autoboxing: Integer n2 = 100;

**BigInteger and BigDecimal Classes**

BigInteger allows very large integers

BigDecimal allows high-precision floating-point values – with no limit to the precision

* Could have issues with the divide method if it cannot be terminated

Both classes have a better precision using the String argument than a number argument

* Methods: add, subtract, multiply, divide, and remainder