Lesson 2.1

Data Types and Type Casting

By the end of this lesson you will learn:

- Different Data Types in Java
- Memory Size, Value Range and Default Value of different Data Types in Java
- Wrapper classes of Different Data Types
- Type Casting and Its types
- Type Casting of Primitive Data Types

In order to code in Java, you must have sound knowledge on different data types. It will help you to choose data types for attributes efficiently.

Different Datatypes

The classification of different data types in Java has been illustrated in the following diagram:

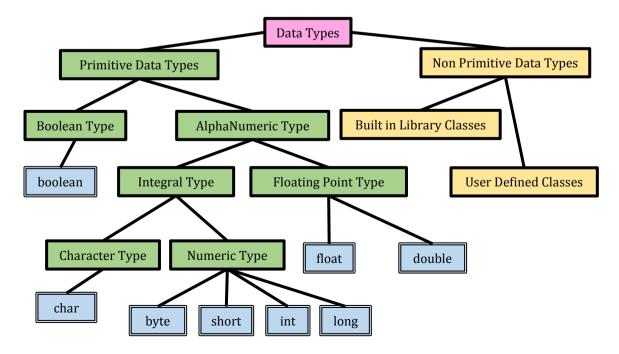


Figure: Data Type Classification Tree

We can see from the classification tree that data types in Java can be divided into two categories. One is Primitive Data Type and the other is Non Primitive Data type. The Non-Primitive data type can be divided into two categories. They are: the built-in library classes of Java and the user defined classes. So, there are numerous not primitive data types in Java. Some widely used built in library classes includes String, System, Random, Scanner etc. All these built-in classes can be found in different built in packages of Java. The user defined classes are the classes that we will be writing throughout the whole semester.

The primitive data types can be divided into two categories. One is Boolean type and the other is AlphaNumeric type. Under the Boolean type there is only one data type named **boolean**. Under the AlphaNumeric type there are two different categories of data type. One is Integral type and the other is Floating Point type. There are two data types under the Floating Point type. They are **float** and **double**. The Integral type can again be divided into two categories. One is Character type and the other is Numeric type. There is only one type named **char** under the Character type and there are four types under the Numeric type. These four types are: **byte**, **short**, **int** and **long**. So, we find eight different primitive data types in java.

Memory Size, Value Range and Default Value of different Data Types

The following table shows the memory size, minimum value, maximum value and default value for each of the eight primitive datatypes.

Table: Memory Size, Minimum Value, Maximum Value and Default Value of Primitive Datatypes

Data Type	Memory Size	Minimum Value	Maximum Value	Default Value	
boolean	1 Byte	-	-	false	
byte	1 Byte	-27	27 -1	0	
short	2 Bytes	-2 15	215 -1	0	
int	4 Bytes	-2 ³¹	231 - 1	0	
long	8 Bytes	-2 ⁶³	2 ⁶³ -1	0L	
char	2 Bytes	'\u0000'	'\uFFFF'	'\u0000'	
float	4 Dryton	-3.4 x 10 ³⁸	-1.4 x 10 ⁻⁴⁵	0.0F	
	4 Bytes	1.4 x 10 ⁻⁴⁵	3.4×10^{38}	0.01	
double	O Dryto	-1.79 x 10 ³⁰⁸	-4.9 x 10 ⁻³²⁴	0.0	
double	8 Byte	4.9 x 10 ⁻³²⁴	1.79 x 10 ³⁰⁸	0.0	

As we know, 1Byte = 8bits, a variable of boolean datatype holds 8 bits of memory. The boolean datatype only has two values: *true* and *false*. So, among the 8 bits, only 1 bit is used to represent its value. Rest of the 7 bits remain idle.

Variable Declaration and Initialization Memory Representation boolean bn1 = true: boolean bn2 = false;

A byte variable also occupies 1Byte (8 bits) of memory. Byte is a numerical datatype and represents signed number. Minimum value of byte is -128 and maximum value is 127. It is not necessary to memorize these value ranges, as we can calculate them using some simple formula.

If, the memory size in bits = n, The minimum value of a numeric datatype = -2^{n-1} ; The maximum value of a numeric datatype = 2^{n-1} - 1; So, for byte data type, n = 8 So, the minimum value of byte datatype = -2^{8-1} = -2^7 = -128; And, the maximum value of byte datatype = 2^{8-1} - 1 = 2^7 - 1 = 127;

Variable Declaration and Initialization **Memory Representation** byte b1 = -128; byte b2 = 127:

Similarly, the three other numeric datatypes short, int and long represents signed numbers and occupies 2Bytes, 4Bytes and 8Bytes of memory (16bits, 32bits and 64bits) respectively. The value range calculations and memory representations are just as like as byte.

A variable of char datatype holds 2Bytes hence 16bits of memory. These values are represented using unsigned hexadecimal numbers. The minimum value is ' $\u0000$ ' and maximum value is ' $\u0000$ '. There are in total 2^{16} values for a char datatype.

Variable Declaration and **Memory Representation** Initialization char $c1 = '\u0000'$; char $c2 = '\setminus uFFFF'$;

The float and double datatypes are of 4Bytes and 8Bytes (32bits and 64bits) respectively. Their value ranges cannot be calculated using the formulas mentioned above. So, unfortunately we have to know these ranges.

The following code illustrates the default values and value ranges for each of the primitive datatypes:

```
public class DataTypeDemo
{
      boolean b1:
      char c1. c2:
      float f1, f2, f3, f4;
      double d1, d2, d3, d4;
      byte bt1, bt2;
      short s1. s2:
      int i1. i2:
      long 11, 12;
      public static void main(String args[])
            DataTypeDemo dtd = new DataTypeDemo();
            System.out.println("Default Value of boolean: "+dtd.b1);
            System.out.println("Default Value of char: "+dtd.c1);
            dtd.c1 = '\u0000';
            System.out.println("Lowest Range of char: "+dtd.c1);
            dtd.c2 = '\uFFFF';
            System.out.println("Highest Range of char: "+dtd.c2);
            System.out.println("Default Value of float: "+dtd.f1);
            System.out.println("Lowest Range of Negetive float: "+dtd.f1);
            System.out.println("Highest Range of Negetive float: "+dtd.f2);
            System.out.println("Lowest Range of Positive float: "+dtd.f3);
            System.out.println("Highest Range of Positive float: "+dtd.f4);
            System.out.println("Default Value of byte: "+dtd.bt1);
            dtd.bt1 = -128;
            System.out.println("Lowest Range of byte: "+dtd.bt1);
            dtd.bt2 = 127;
            System.out.println("Highest Range of byte: "+dtd.bt2);
            //the next two lines will give error
            //dtd.bt1 = -129;
            //dtd.bt2 = 128;
      }
}
// An extended version of the code with double, short, int and long will be discussed in class.
```

So, till now we have discussed about primitive datatypes. What about non-primitive datatypes? The memory size of non-primitive datatypes depends on various factors. For example the memory of *String* depends on its length. We all know that String is a collection of characters. Suppose we have two String variables:

```
String str1 = "Shakib";
String str2 = "Mashrafee";
```

The variable str1 has 6 characters. Each character holds 2Bytes of memory. So, the str1 variable will hold 6 x 2Bytes = 12Bytes of memory. Similarly, the str2 variable has 9 characters and it holds 18Bytes of memory.

Wrapper classes of Different Data Type

For each of the primitive datatypes there is a java library class that allows us to use primitive data types as objects. These classes are called Wrapper Class. The following table shows the wrapper classes for each of the primitive datatypes:

Data Type	Wrapper Class
boolean	Boolean
byte	Byte
short	Short
int	Integer
long	Long
char	Character
float	Float

Table: Wrapper Classes

These classes also hold necessary methods of performing different operations regarding their respective primitive datatypes. For example: if we want to convert a String into an integer or into a double, the Integer class has a method <code>int parseInt(String)</code> and the Double class has a method <code>double parseDouble(String)</code> for the conversion. Both of these methods take a String value in their parameter, converts them and returns the converted value. Again, a code example will be shown in class.

Double

double

Type Casting and Its Types

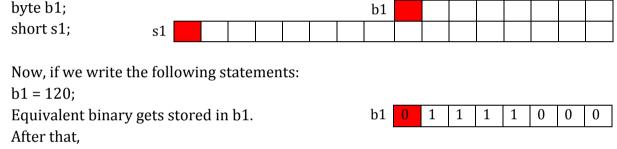
Type Casting is the process of converting the value of a primitive data type to another primitive data type. Example: Converting an integer value to a double value and vice

versa, converting a char value to an integer and vice versa etc. There are two types of type castings:

- i. Implicit Type Casting.
- ii. Explicit Type Casting.

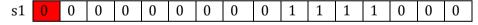
Implicit Type Casting: Converting the value of a smaller primitive data type to a larger primitive data type is known as Implicit Type Casting. It happens automatically. We do not need to mention it in our codes. There is no possibility of value loss during implicit type casting. Example: from byte to short.

Let's, declare one byte type variable, one short type variable and draw the memory representation for them:



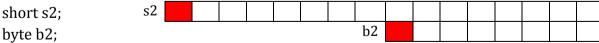
s1 = b1; // we do not need to mention anything.

Sign bit of b1 gets copied into the sign bit of s1. Rest of the bits of b1 gets copied into their respective positions in s1. Rest of the bits in s1 is filled with 0. Equivalent decimal is still 120. So, there is no value loss.



Explicit Type Casting: Converting the value of a larger primitive data type to a smaller primitive data type is known as Explicit Type Casting. It does not happen automatically. We need to mention it in our codes. There is a possibility of value loss during explicit type casting. For example: from short to byte.

Let's, declare one short type variable, one byte type variable and draw the memory representation for them:



Now, if we write the following statements:

s2 = 130;

Equivalent binary gets stored in s2.

s2	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0
----	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

After that,

b2 = (byte) s2; // we need to mention the destination datatype while assigning.

The bits of b2 gets filled up by the respective bits of s2. Equivalent decimal is -126. So, there is a value loss.

b1 $\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$

Type Casting of Primitive Data Types

Whether a conversion is implicit or explicit, it depends on the memory size and/or the value loss during the conversion process. The following table shows which conversions are implicit type casting and which conversions are explicit type casting:

Table: Type Castings

Data Type	Type Casting	Data Type	Type Casting		
byte to short		short to byte			
short to int		int to short			
int to long		long to int			
byte to int		int to byte	Evaliait		
byte to long	Implicit	long to byte			
short to long		long to short	Explicit		
float to double		double to float			
int to double		double to int			
long to float		float to long			
short to char	Explicit	char to short			

We will be discussing a code to demonstrate all these castings in class.

Practice

- ✓ Different types of data types along with their size, minimum value, maximum value.
- ✓ Wrapper classes of different types.
- ✓ Different types of type castings.