

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:

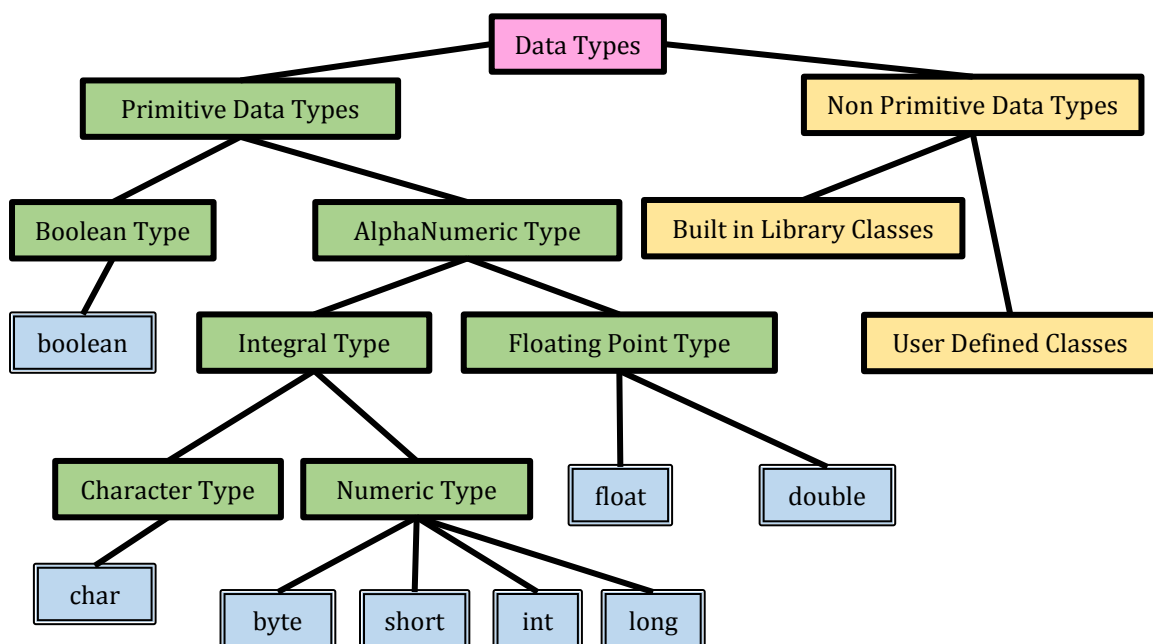


Figure: Data Type Classification Tree

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
<i>boolean</i>	1 Byte	-	-	false
<i>byte</i>	1 Byte	-2^7	$2^7 - 1$	0
<i>short</i>	2 Bytes	-2^{15}	$2^{15} - 1$	0
<i>int</i>	4 Bytes	-2^{31}	$2^{31} - 1$	0
<i>long</i>	8 Bytes	-2^{63}	$2^{63} - 1$	0L
<i>char</i>	2 Bytes	'\u0000'	'\uFFFF'	'\u0000'
<i>float</i>	4 Bytes	-3.4×10^{38}	-1.4×10^{-45}	0.0F
		1.4×10^{-45}	3.4×10^{38}	
<i>double</i>	8 Byte	-1.79×10^{308}	-4.9×10^{-324}	0.0
		4.9×10^{-324}	1.79×10^{308}	

																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																					</
--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	----

The minimum value of a numeric datatype = -2^{n-1} ;

$$n = 8$$

And, the maximum value of byte datatype = $2^{8-1} - 1 = 2^7 - 1 = 127$;

[illegible][illegible]

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:

[illegible]

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 $6 \times 2\text{Bytes} = 12\text{Bytes}$ 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:

Table: Wrapper Classes

Data Type	Wrapper Class
<i>boolean</i>	Boolean
<i>byte</i>	Byte
<i>short</i>	Short
<i>int</i>	Integer
<i>long</i>	Long
<i>char</i>	Character
<i>float</i>	Float
<i>double</i>	Double

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 *int parseInt(String)* and the Double class has a method *double parseDouble(String)* 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.

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:

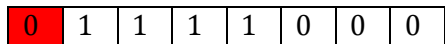
```
byte b1;
short s1;
```



Now, if we write the following statements:

```
b1 = 120;
```

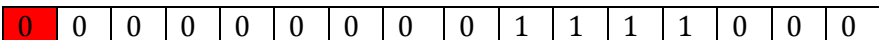
Equivalent binary gets stored in b1.



After that,

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

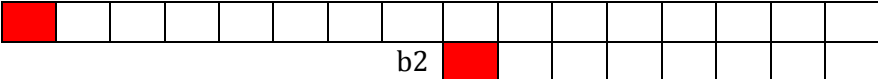
Sign bit of b1 gets copied into the sign bit of s1. Rest of the bits of b1 gets copied into their respective positions in s1. Rest of the bits in s1 is filled with 0. Equivalent decimal is still 120. So, there is no value loss.



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:

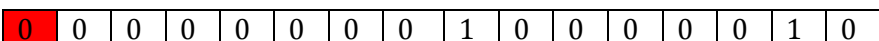
```
short s2;
byte b2;
```



Now, if we write the following statements:

```
s2 = 130;
```

Equivalent binary gets stored in s2.



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

1	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

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
<i>byte to short</i>	Implicit	<i>short to byte</i>	Explicit
<i>short to int</i>		<i>int to short</i>	
<i>int to long</i>		<i>long to int</i>	
<i>byte to int</i>		<i>int to byte</i>	
<i>byte to long</i>		<i>long to byte</i>	
<i>short to long</i>		<i>long to short</i>	
<i>float to double</i>		<i>double to float</i>	
<i>int to double</i>		<i>double to int</i>	
<i>long to float</i>		<i>float to long</i>	
<i>short to char</i>	Explicit	<i>char to short</i>	

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.