**CSCI 2302**

**Object-Oriented Thinking Chapter**

**Wrapper Classes, BigInteger & BigDecimal Notes**

**And**

**Integer, Double, BigInteger & BigDecimal Lab**

Contents

[Wrapper Classes 1](#_Toc36580233)

[valueOf method 2](#_Toc36580234)

[parse method 2](#_Toc36580235)

[difference between 3](#_Toc36580236)

[autoboxing and autounboxing 3](#_Toc36580237)

[BigInteger and BigDecimal reference data types 3](#_Toc36580238)

Intro: Owing to performance considerations, primitive data type values are not objects in Java. Because of the overhead of processing objects, the language’s performance would be adversely affected if primitive data type values were treated as objects. However, many Java methods require the use of objects as arguments. Java offers a convenient way to incorporate, or wrap, a primitive data type value into an object (e.g., wrapping an int into an Integer object, wrapping a double into a Double object, and wrapping a char into a Character object). By using a wrapper class, you can process primitive data type values as objects. Java provides Boolean, Character, Double, Float, Byte, Short, Integer, and Long wrapper classes in the java.lang package for primitive data types. The Boolean class wraps a Boolean value true or false. This lab uses Integer and Double as examples to introduce the numeric wrapper classes.

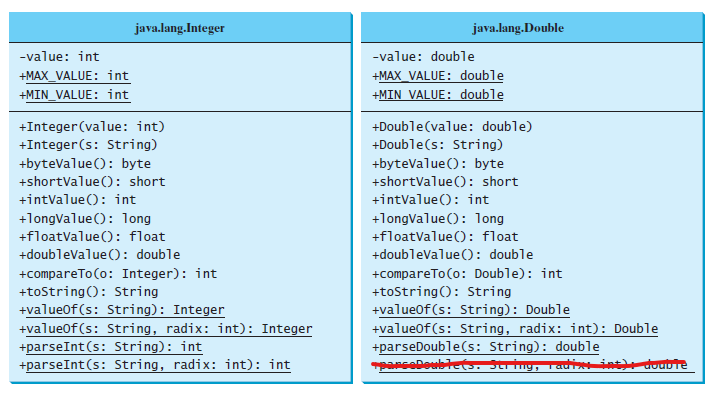
Learning Goals**:**  Toexamine Java’s language use of primitive data types and reference (objects/ADTs) data types. To understand why Java uses primitive data types. We will apply the concepts of turning (wrapping) primitive data types into object data types.

Notes**:**

## Wrapper Classes

The primitive data types have matching reference data type, these reference data types are called wrapper classes because the class will take a primitive data type and “wrap” it into the reference data type. Below are the Integer and Double reference/wrapper classes

FYI: The parseDouble(s: String, radix: int): double method does not exist.



It is nice to be able use the ­MAX\_VALUE and the MIN\_VALUE for looking min and max values in a set when you cannot assign those values to the first value in that set.

### valueOf method

Each numeric wrapper class has a useful static method, valueOf(String s), that instantiates a new object with the numeric value of s.

For example, if you are wanting an Integer from the user, then you can read in that value as a String and utilize the valueOf method to instantiate an Integer with that value:

String userNumberAsString = input.next();

Integer userNumberAsInteger = Integer.valueOf(userNumberAsString);

### parse method

Each numeric wrapper class has a useful static method, parse*DataType*(String, s) and parse*DataType* (String s, int radix), that returns a primitive data type with the numeric value of s. The argument, radix, stands for the numeric numbering system; Binary (radix = 2), Decimal (radix = 10), Hexadecimal (radix = 16), and Octal (radix = 8). But that doesn’t mean you can’t use that primitive value and instantiate a reference data type from it.

For example, if you are reading in data from an older format version, like Octal, you can read in that value then format it into an int.

int number = Integer.parseInt(input.next(), 8);

### difference between

What is the difference between these two methods?

The valueOf method can take an integer or a String as a parameter and returns a reference data type; and when the given String is invalid, it provides an error. This method can also take in a character as a parameter but the output will be its corresponding Unicode value. This method will always cache values in the range -128 to 127, inclusive, and may cache other values outside of this range.

The parse*DataType* can **only** take a String for the argument and returns primitive data type. It will produce an error if the String is not of numeric numbers.

### autoboxing and autounboxing

Before we cover the next point, I’d like for you to remember a couple of things:

* You can assign a double from an int: double number = 202;
  + called widening a type
* You have to cast a double to become an int: int number = (int)123.456;
  + Called narrowing a type

When you convert a primitive data type to a reference data type (*to* a wrapper object) it is called boxing.

When you convert a reference data type to a primitive data type (*from* a wrapper object) it is called unboxing.

Since this is something that is required often, Java’s compiler will do this for you; thus autoboxing and autounboxing.

## BigInteger and BigDecimal reference data types

The BigInteger reference data type can hold very, VERY, large whole numbers; an integer of any size.

BigInteger is immutable, meaning any modification to that number creates a new object.

The constructor for this class **only** takes in a String for the argument, think about that for a second, why? If BigInteger can hold a number that is larger than 9,223,372,036,854,775,807 which is the largest value the primitive data types can hold, how else would you get that VERY large number to be assigned to the BigInteger.

BigInteger bigNumber =

new BigInteger(“123456789987654321123456789987654321”);

The BigDecimal reference data type can hold a floating-point number (a number that contains a decimal – a fractional number) with high-precision.

BigDecimal is immutable, meaning any modification to that number creates a new object.

The constructor can take a String, a Double, or a double, but you should only use the String to create the BigDecimal. Think about it, it you are creating this highly accurate variable, then it has to start off with a correct value, the Double and the double have accuracy issues.

There is no limit to the precision of a BigDecimal object.

If the divide method cannot terminate, an AritheticException will be thrown.

BigDecimal bigNumberWithAFraction =

new BigDecimal(“123.32165498778945612631230”);

Since the both of these classes creates a variable that is a reference data type, you cannot use the primitive math operators (+, -, \*, /, %) on it, you have to use the methods add, subtract, multiply, divide, and remainder to perform the arithmetic operations. You also cannot use the primitive relation operators (==, !=, <, <=, >, >=), you have to use the compareTo method.

To use these methods, you will have to use the math class, meaning you have to include:

import java.math.\*;

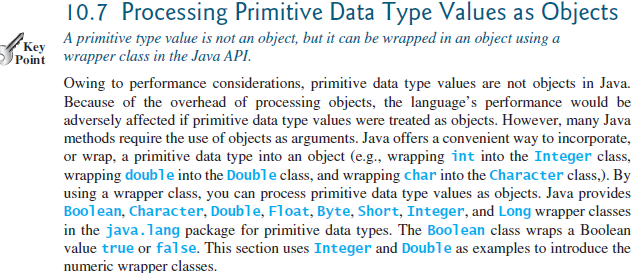
at the beginning of the program to use these methods. (<https://docs.oracle.com/javase/8/docs/api/java/lang/Math.html>)

Task:Complete the steps outlined below in a file named YourMySFAusername\_I\_D\_BI\_BD.java.

**Integer, Double, BigInteger & BigDecimal Lab**

For this lab, you will be reading from: idbb.txt and the keyboard. Set up your Scanners appropriately.

1. Display to the screen why Java uses primitive data types (OOT Chapter, section 7).

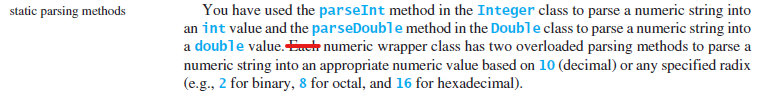


(Remember that API stands for Application Program Interface, aka the Java library which contains the predefined classes and interfaces for developing Java programs)

1. Display to the screen why the wrapper classes do not have no-arg constructors (OOT Chapter, section 7).



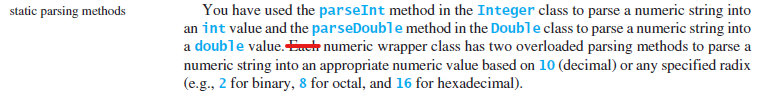
1. Read in, from the file, a value as a String and parse into an Integer variable. (Recall that parsing only takes String data types.)



(only the Integer class)

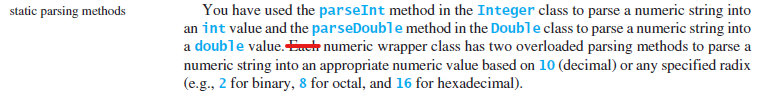


1. Read in, from the file, a value as a String and parse it into a Double variable. (Recall that parsing only takes String data types.)



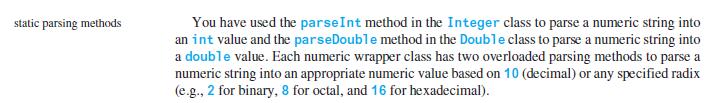


1. Read in, from the file, the next four values as String and parse into Integer variables, the format to read in: String radix. (Recall that parsing only takes String data types and only parsing takes the numeric system.)

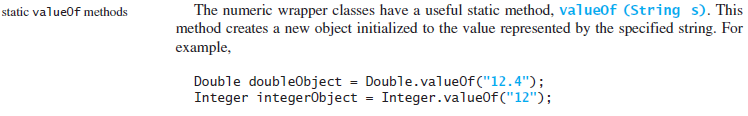




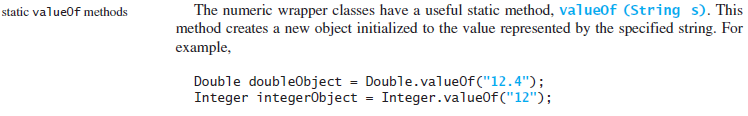
1. Explain (print to the screen) what the radix argument is.



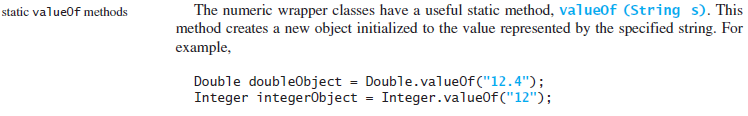
1. Declare and create an Integer data type using the valueOf method with the argument of 2302.



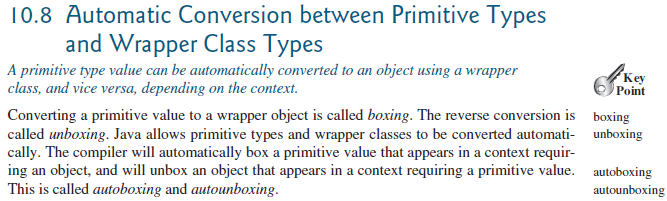
1. Declare and create an Integer data type using the valueOf method with the argument of lumberjacks.



1. Declare and create a Double data type using the valueOf method with the argument of 2302.001.

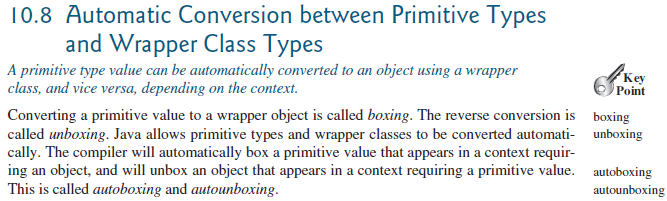


1. Declare and create an Integer variable by using Java’s autoboxing and the value 2302.





1. Declare and create a Double variable by using Java’s autoboxing and the value 2302.123.





1. Read in, from the file, the next line as a String, declare and create a BigInteger variable using that String.





1. Read in, from the file, the next line as an int, declare and create a BigInteger variable using that int.



1. Explain (print to the screen) the difference between the variables from #12 & #13 – if there is not a difference state that.
2. Read in, from the file, the next line as a String, declare and create a BigDecimal variable using that String.



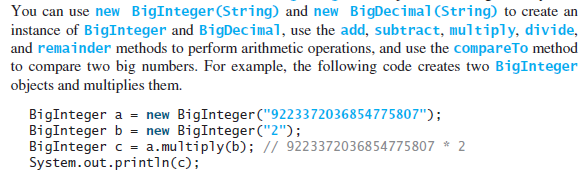


1. Read in, from the file, the next line as a double, declare and create a BigDecimal variable using that double.



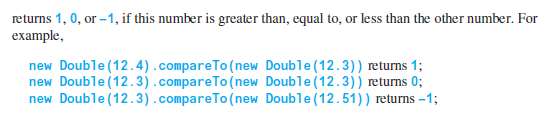


1. Explain (print to the screen) the difference between the variables from #15 & #16 – if there is not a difference state that.
2. Divide the variables from #15 & #16 and print the result to the screen.



1. Compare the variables from #15 & #16 using the compareTo method and print the result to the screen.





Submit: Submit your YourMySFAusername\_I\_D\_BI\_BD.java file in the Dropbox in Brightspace by D2L.