303.2.1
Wrapper Class, Characters Class, Mathematical Class, and Strings Class

Learning Objective

- In this presentation, we will learn how to solve practical problems programmatically. Through the example problems, we will discuss Java Wrapper classes, auto boxing, unboxing, and other related subjects.

- We will explore and learn math operators and the built-in methods available in Java Math Class, as well as how to use them to perform mathematical equations. By the end of this module, you will have a good understanding of how to work with math operators and methods in Java.

- We will demonstrate String Manipulation by using String methods such as String built-in methods.

- There are a few practice labs and practice assignments available. Learners can use office hours to complete labs and assignments.

Outline

Overview of Java Autoboxing and Unboxing

In this section, we are going to learn about Java autoboxing and unboxing. Java introduced the concept of autoboxing and unboxing in Java 5. Using this concept, we can interchangeably convert the primitive data types into their respective Wrapper Classes.

At the end of this lesson, learners will be able to understand and demonstrate the Wrapper Classes.

Topic 1: Java Wrapper Class

- The built-in primitive types are not objects, and they do not have methods. To bridge the gap between the built-in types and objects, Java defines **Wrapper Classes** for each of the primitive types.
- The wrapper classes are objects encapsulating primitive Java types.
  - **Autoboxing:** primitive values are automatically converted to their matching wrapper class objects.
  - **Unboxing**: converting an object of a wrapper type to its corresponding primitive value.

The Java Wrapper Classes are shown on the right. We will learn more about them later.

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

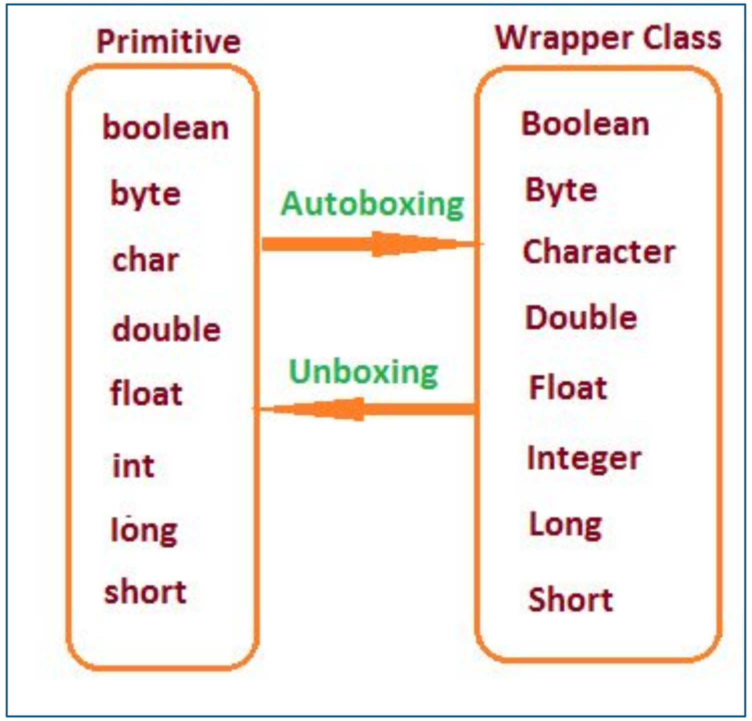
Wrapper Class Examples

```
byte b = 100;
Byte bb = 100;
short s = 100;
Short ss = 100;
int i = 100;  // primitive data type
Integer ii = 100;  // literal method
Integer iii = new Integer(1000);  // new operator
long l = 100l;
Long ll = 100l;
float f = 100.0f;
Float ff = 21.24f;
double d = 546.32;
Double dd = 545.255;
char ch = 'a';
Character cha = 'a';
boolean bo = true;
Boolean boo = true;
```

Example - AutoBoxing and UnBoxing

```
 //--------AutoBoxing Examples---
int a = 20;
Integer convertintoInteger = a;

char c = 'A';
Character convertintoChar = c;

double d = 565.23;
Double convertintoDouble = d ;
```

```
// ---- UnBoxing Example------
Integer i = 56;
int ii = i;

Character c = 'a';
char ch = c;

Double d = 2563.32;
double dd = d;
```



Questions?

Lesson 2

Overview of Java Character Class

In this section, we will discuss the Character Class in Java, which wraps the valu of primitive data type **char** into its **object**.

By the end of this lesson, learner will be able to describe and demonstrate the Characters in Java.

Topic 1: **The Character Class**

- Java provides a wrapper class named **Character,** which is an option for the variable's data type. The Character class object can hold a **single character** value just like a primitive **char** data type.
- The Character class offers a number of useful methods for manipulating characters. All of the attributes, methods, and constructors of the Character class are specified by the Unicode Data file, which is maintained by the ***Unicode Consortium.***
- There are two ways to create a Character class in Java:
  - ➢ **Character literal:** Character can be created by assigning a Character literal to a Character variable.

```
Character letter = 'g';
Character num = '7';
```

Please go to our **wiki** document to learn more about **Unicode**.

→ **Using the new operator:** JVM will create a new character object in normal (non-pool).

→ Heap memory.

```
Character letter = new Character( 'g' );
Character num = new Character( '7' );
```

Please go to our **wiki** document to learn more about **Character** Class.

Example - Character and Char Data Type

In the below example, we are using both a Char primitive data type and a Wrapper class primitive data type.

```
public class characterclassDemo{
   public static void main(String args[]) {
       // Wrapper character class
        Character letter = new Character( 'g' );
       Character num = new Character( '7' );
       System.out.println(letter);
       System.out.println(num);

      // primitive char data type
       char letter_ch = 'g';
       char num_ch = '7';
       System.out.println(letter_ch);
       System.out.println(num_ch);
   }
}
```

```
Output
g
7
g
7
```

Both will give you the same result, but Character class has more built-in methods for character manipulation.

Topic 2: Popular methods in the Character Class

The Character class also provides several methods useful when manipulating, inspecting, or dealing with single-character data. Some popular methods of Character class are as follows:

Method	Description
isDigit(ch)	Returns true if the specified character is a digit.
isLetter(ch)	Returns true if the specified character is a letter.
isLetterOfDigit(ch)	Returns true if the specified character is a letter or a digit.
isLowerCase(ch)	Returns true if the specified character is a lowercase letter.
isUpperCase(ch)	Returns true if the specified character is an uppercase letter.
toLowerCase(ch)	Returns the lowercase of the specified character.
toUpperCase(ch)	Returns the uppercase of the specified character.
valueOf() **valueOf(String s),** **valueOf(String s, int radix)**	The valueOf method returns the relevant Number Object holding the value of the argument passed. The argument can be a primitive data type, string, etc.

TEKsystems Own change

Example - Methods in the Character Class

```java
public class CharacterClassDemo {
   public static void main(String[] args) {
       Character letter = 'A';
       System.out.println(letter);
       Character asciNumber = 97;
       System.out.println(asciNumber);
       Character uninumber = '\u0031';
       Character uninumber2 = '\u0032';
       System.out.println(uninumber);
       System.out.println(uninumber2);
       System.out.println("======compareTo========");
       Character Obj1 = '2';
       Character Obj2 = '2';
       int result =    Obj1.compareTo(Obj2);
       System.out.println(result);
       System.out.println("----equals() -------");
       boolean  result2 =    Obj1.equals(Obj2);
       System.out.println(result2);
       System.out.println("----isletter() -------");
       // isletter method determines wheather the specified char value is letter
       System.out.println(Obj1.isLetter(Obj2));
       System.out.println("----isDiggit() -------");
       // isDiggit() determin whether the specified char value is a digit
       System.out.println( Obj2.isDigit(Obj1));
       Obj1.valueOf('A');   // Obj1 = 'A'
       System.out.println("----compareTo-------");
       Character a = 'B';
       Character aa = 'B';
       System.out.println(a == aa);
       int rs =  a.compareTo(aa);
       System.out.println(rs);
   }}
```

This example demonstrates how to use, declare, and initialize data type as character type using Character Class.

Topic 3: Character Class - Escape Sequence

- You cannot have a String literal break across lines. How do you include a line break? **Solution**: An **escape sequence** is a two-character sequence that represents a single special character.
- An escape sequence is a character preceded by a backslash(\), which gives a different meaning to the compiler. The following table shows the escape sequences in Java:

Escape Sequence	Description
\t	Inserts a tab in the text at this point.
\b	Inserts a backspace in the text at this point.
\n	Inserts a new line in the text at this point
\r	Inserts a carrier return in the text at this point
\f	Inserts a form feed in the text at this point
\’	Inserts a single quote character in the text at this point
\”	Inserts a double quote character in the text at this point
\\	Inserts a backslash in the text at this point

TEKsystems
Own change

Example: Escape Sequence

Create a class named **escapedemo** and write a below code in that class.

```
public class escapedemo {
    public static void main(String[] args) {
    System.out.print("Hello \nWelcome to"); //using \n
    System.out.println(" The \"Per Scholas\" Training institute."); //using \"
    System.out.println("This is a \'Java Developer\' Cohort."); //using \'
    System.out.println("My java file is in: projects\\src\\java"); //using \\
  }
}
```

Output

```
Hello
Welcome to The "Per Scholas" Training Institute.

This is a 'Java Developer' Cohort.
My java file is in: projects\src\java
```

TEKsystems Own change

Lesson 3

String literal or Data type

In this lesson, with the help of examples, we will explore Java Strings, how to create them, and discuss the various methods of the String class.

By the end of this lesson, learners will be able to describe, explore, and demonstrate the Strings.

Topic 1: Overview of String Class

- ❑ **String** is a sequence of characters. For example, "Hello" is a string of five characters.
- ❑ **String** is immutable. An Immutable Object means that the state of the object cannot change after its creation. **"String is immutable,"** which means that you cannot change the object itself, but you can change the reference to the object.
  - ➢ There are methods for inserting and replacing characters of a String, but these methods return a brand new String object and <u>do not modify</u> the String on which the method was called.

[Please go to our **wiki** document to learn more **String** Class].

TEKsystems Own change

Topic 2: Initializing and Declaring Strings

There are two ways to create Strings in Java:

- **String literal:** Strings can be created by assigning a String literal to a String variable:
  - `String s1 = “welcome”;`
  - `String s2 = “welcome”; //it doesn’t create a new instance.`

  Each time we create a string literal, the JVM checks the **"string constant pool"** first. If the string already exists in the pool, a reference to the pooled instance is returned. If the string does not exist in the pool, a new string instance is created and placed in the pool.

- Using the **new** operator:
  - `String s = new String(“welcome"); //creates two objects and one reference variable.`

  In such a case, JVM will create a new string object in normal **(non-pool) heap memory**, and the literal “welcome" will be placed in the string constant pool. The variables will refer to the object on the heap (non-pool). The String class defines multiple constructors.

To understand the concept of **Java String pool** please go to the **wiki** document.

Topic 3: Popular String Methods

Java String class provides various methods to perform different operations on strings. We will look into some of the commonly used string operations.

```
// Finding String size
int length()   // returns the length of the String
boolean isEmpty()  // same as str.length() == 0
boolean isBlank() contains only white spaces (JDK 11)


// String Comparison
boolean equals(String another)
boolean equalsIgnoreCase(String another)
compareTo(String another)
valueOf(type arg)
boolean startsWith(String another)
boolean endsWith(String another
// Finding a Character in a String
int indexOf()
int lastIndexOf()
boolean contains()
```

```
// Extracting a char or substring
char charAt(int idx)
String substring()

//Converting Strings - Strings are immutable
String toLowerCase()
String toUpperCase()
String concat(String another)
String trim() // creates a new String removing white
spaces from front and back
String strip()// strips the leading and trailing white
spaces (Unicode aware) (JDK 11)

// Text Processing and Regular Expression (JDK 4)
boolean matches(String regex)
String replace(char old, char new)
String replaceAll(String regex, String replacement)
String[] split(String regex)  // Split the String using
regex as delimiter, return a String array
```

Hands-On Lab: String methods

You can find this LAB- 303.2.1-String Class methods on Canvas, under the Guided Lab section. If you have any technical questions while performing the lab activity, ask your instructors for assistance.

Practice Assignment for String Class

After you learn about String class and its method, you can complete the assignment: 303.2.1- Practice Assignment - Strings, which you can find on Canvas under the **Assignment** section.

Use your office hours to complete this assignment. If you have any technical questions while performing the lab, ask your instructors for assistance.

Note: This assignment does not count toward the final grade

Topic 4: Reading a **String** From the Console

When you are developing console applications using Java, you must read input from users through the console. To read a string from the console as input in Java applications, you can use the **Scanner** class along with the **System.in**.

Go to LAB - 303.2.2-Reading a String From the Console on Canvas under the **Guided Lab section**. If you have any technical questions while performing the lab, ask your instructors for assistance.

Please view our **wiki** document to learn more about **Reading a String From the Console.**

Topic 5a: Conversion From **String** to **Number**

We can use **parseInt()** and **valueOf()** methods for conversion.

Example:

```
public class Stringconversion {
   public static void main(String[] args) {
   String intString = "1";

   int intValue = Integer.parseInt(intString); // parseInt method convert string into primitive data type

   Integer IntWrapper = Integer.valueOf(intString); // valueOf(): convert string into non primitive data type

   String doubleString = "1.0";
   double doubleValue = Double.parseDouble(doubleString);
   System.out.println(doubleValue);
   Float fwrapper = Float.valueOf(doubleString );
   System.out.println(fwrapper);
}
}
```

Notice that we have used class names ***Integer*** and ***Double*** to invoke the parsing methods. ParseInt and parseDouble are static methods of their respective classes.

Topic 5(b): Conversion From **Number** to **String**

A **toString()** is an in-built method in Java that returns the value given to it in a *string* format. Hence, any object that this method is applied to will be returned as a *string object*. The toString() method in Java has two implementations:

The first implementation is when it is called a method of an object instance. See the example below:

```
public static void main( String args[] ) {
        //Creating an integer of value 10
        Integer number=10;
 // Calling the toString() method as a function of the
Integer variable
        System.out.println( number.toString() );
    }
```

The second implementation is when you call the member method of the relevant class by passing the value as an *argument*. See the example below:

```
public static void main( String args[] ) {
        // The method is called on datatype Double
        // It is passed the double value as an argument
        System.out.println(Double.toString(11.0));
        // Implementing this on other datatypes
        //Integer
        System.out.println(Integer.toString(12));
        // Long
        System.out.println(Long.toString(123213123));
        // Booleam
        System.out.println(Boolean.toString(false));
    }
```

Topic 6a : Advanced String Manipulation - **StringBuffer**

- StringBuffer is a peer class of **String** that provides much of the functionality of strings.
- StringBuffer class creates objects that hold a mutable sequence of characters.
- Substrings are inserted anywhere in the StringBuffer.
- StringBuffer class extends (or inherits from) **Object** class.
- The **String** represents fixed-length, immutable character sequences, while **StringBuffer** represents growable and writable character sequences.
- A **String** is immutable. If you try to alter its value, another object gets created. A **StringBuffer** is mutable so it can change its value.

StringBuffer - Constructors

- **new StringBuffer()** → It creates an empty String buffer with the initial capacity of 16.

- **new StringBuffer(int capacity)** → It creates an empty String buffer with the specified capacity as length.
  - StringBuffer(100) → Room for 100 characters.

- **new StringBuffer(String str)** → It creates a String buffer with the specified string.
  - New StringBuffer("Per Scholas") → Room for given string and 16 additional characters.

Example One - **StringBuffer**

```
public static void main(String[] args) {
         StringBuffer str = new StringBuffer("Java Full Stack Developer");
         int len = str.length();
         System.out.println("Length : " + len);
         int cap = str.capacity();
         System.out.println("Capacity : " + cap);
// ----- append()--------------------
         str.append("and Software Eng "); // appends a string in the previously defined string.
         System.out.println(str);
         str.append("Java is my favourite language ");
         str.append("but i love python");
         System.out.println(str);
// ----------- reverse()----------------------
       StringBuffer stringName = new StringBuffer("Welcome to Per Scholas");
        System.out.println("Original String: " + stringName);
        stringName.reverse();
        System.out.println("Reversed String: " + stringName);
     }
```

Example Two - **StringBuffer**

```
public class StringbufferExampleTwo {

     public static void main(String[ ] args) {
     // ------ insert()----------------
     StringBuffer s = new StringBuffer("Java");
    s.insert(4, "language");
    System.out.println(s);
    // --------- replace()---------
    StringBuffer sT = new StringBuffer("Java");
    sT.replace(0, 2, "Hello");
    System.out.println(sT);

    StringBuffer str = new StringBuffer("Welcome to Java Fullstack ");
    System.out.println("Original string: " + str);
    System.out.println("Substring with start index: " + str.substring(5));
    System.out.println("Substring with start and end index: " + str.substring(5, 10));
    }
}
```

Output

```
Javalanguage
Hellova
Original string: Welcome to Java Fullstack
Substring with start index: me to Java Fullstack
Substring with start and end index: me to
```

Topic 7: Advanced String Manipulation - **StringJoiner**

- StringJoiner is used to construct a sequence of characters separated by a delimiter, and optionally starts with a supplied prefix and ends with a supplied suffix.

- Constructors:
  - New StringJoiner(“,”).
    - Creates StringJoiner that is comma-delimited (e.g., Mike, Bairon, and Erik).

  - new StringJoiner(“,”, “[“, “]”).
    - Creates StringJoiner that is comma-delimited, with each string surrounded by brackets (e.g., [Mike], [Bairon], [Erik]).

Example One - **StringJoiner**

```
import java.util.StringJoiner;
public class StringjoinnerDemoOne {

     public static void main(String[] args) {
      StringJoiner joinNames = new StringJoiner(","); // passing comma(,) as delimiter
      // ----Adding values to StringJoiner  ----
             joinNames.add("Java");
             joinNames.add("Python");
             joinNames.add("C Sharp");
             joinNames.add("Javascript");

             System.out.println(joinNames);
     }
}
```

Console Output:

```
Java,Python,C Sharp,Javascript
```

Example Two - **StringJoiner**

```
import java.util.StringJoiner;
public class StringjoinnerDemotwo {
     public static void main(String[] args) {
//---- passing comma(,) and square-brackets as delimiter  ----
StringJoiner joinData = new StringJoiner(",", "[", "]");
      // ---Adding values to StringJoiner  ---
          joinData.add("Java");
          joinData.add("Python");
          joinData.add("C Sharp");
          joinData.add("JavaScript");
          System.out.println(joinData);
     }
}
```

Console Output:

```
[Java,Python,C Sharp,JavaScript]
```

Example Three - **StringJoiner**

```
import java.util.StringJoiner;
public class StringjoinnerDemoOne {
     public static void main(String[] args) {
     StringJoiner joinNames = new StringJoiner(",", "[", "]");   /* passing comma(,) and square-brackets as delimiter */
  // ----- Adding values to StringJoiner---
         joinNames.add("New York");
         joinNames.add("New Jersey");
  //----  Creating StringJoiner with :(colon) delimiter
    StringJoiner joinNames2 = new StringJoiner(":", "[", "]");  /* passing colon(:) and square-brackets as delimiter  */
  // ---Adding values to StringJoiner-----
          joinNames2.add("Dallas");
         joinNames2.add("Chicago");
  // ---- Merging two StringJoiner  ----
         StringJoiner merge = joinNames.merge(joinNames2);
         System.out.println(merge);
     }
}
```

Console Output:

```
[New York,New Jersey,Dallas:Chicago]
```

Formatting Output on the Console

Java has several ways to print/display an output on the Console. We will explore some of them.

Topic 1: Formatting Output on Console

There are multiple ways of formatting Output in Java. Some of them are old-school and borrowed directly from old classics (such as printf from C) while others are more in the spirit of object-oriented programming (OOP).

Frequently used formatting ways:

- System.out.format().
- System.out.printf().
- String.format().

Topic 2: The **printf** and **format** Methods

- Earlier, you saw the use of the **print** and **println** methods for printing strings to standard output **(System.out)**. The Java programming language has other methods that allow you to exercise much more control over your print output when numbers are included. Those methods include:
    - System.out.**printf**(format, String... arguments);
    - System.out.**format**(format, String... arguments);

- You can use **`format`** or **`printf`** anywhere in your code that previously used `print` or `println`.

- Both methods are identical and behave similarly.

  Please view our **wiki** document to learn more about output formatting on console.

TEKsystems
Own change

Examples One: **format()** Method Specifier

```
int i = 1024;
byte b = 127;
double d = 1.232, tiny = d / 1000000.0;
boolean bool = true;
System.out.format(" This is an integer: %d and this is a byte: %d.\r\n", i, b);
System.out.format(" This is a double: %.4f and this is tiny: %.4e.\r\n", d, tiny);
System.out.format(" And this is a String: %s", "This is a string.\r\n");
```

The result would be:

```
This is an integer: 1024 and this is a byte: 127.
This is a double: 1.2320 and this is tiny: 1.2320e-06.
And this is a String: This is a string.
```

The Format Method Specifier expects at least one argument – the format string.

- It can accept any number of additional arguments, and of any type.
- The additional arguments are matched, in order, to the format specifiers found within the format string.
- The format specifiers must match the provided arguments; otherwise, an IllegalFormatException is thrown.

Examples Two: **format()** Method Specifier

This program shows some of the formatting that you can perform with `format`.

The output is shown within the double quotes in the embedded comment:

```
import java.util.Calendar;
import java.util.Locale;
public class TestFormat {
    public static void main(String[] args) {
      long n = 461012;
      System.out.format("%d %n", n);      //  -->  "461012"
      System.out.format("%08d%n", n);    //  -->  "00461012"
      System.out.format("%+8d%n", n);    //  -->  " +461012"
      System.out.format("%,8d%n", n);    // -->  " 461,012"
      System.out.format("%+,8d%n%n", n); //  -->  "+461,012"

      double pi = Math.PI;

      System.out.format("%f %n", pi);       // -->  "3.141593"
      System.out.format("%.3f%n", pi);     // -->  "3.142"
      System.out.format("%10.3f%n", pi);   // -->  "     3.142"
      System.out.format("%-10.3f%n", pi);  // -->  "3.142"

      Calendar c = Calendar.getInstance();
      System.out.format("%tB %te, %tY%n", c, c, c); // -->  "May 29, 2006"
      System.out.format("%tl:%tM %tp%n", c, c, c);  // -->  "2:34 am"
      System.out.format("%tD%n", c);    // -->  "05/29/06"
    }
}
```

Example One - **printf** Method

If we have multiple printf() statements without a newline specifier:

```
System.out.printf("Hello, %s!", "Michael Scott");
System.out.printf("Hello, %s!", "Jim");
System.out.printf("Hello, %s!", "Dwight");
```

The result would be:

```
 Hello, Michael Scott!Hello, Jim!Hello, Dwight!
```

However, if we include the newline character:

```
System.out.printf("Hello, %s!%n", "Michael Scott");
System.out.printf("Hello, %s!%n", "Jim");
System.out.printf("Hello, %s!%n", "Dwight");
```

The result would be:

```
Hello, Michael Scott!
Hello, Jim!
Hello, Dwight!
```

Example Two - **printf** Method

```java
public static void main(String[] args) {
        System.out.println("------------------------");
        String data = "Hello World!";
        System.out.printf("Printing a string: %s\n", data);
        System.out.printf("Printing a string in uppercase: %S\n", data);
        int x = 100;
        System.out.printf("Printing a decimal integer: %d\n", x);
        int hours = 100;
        double days = 100/24.0; // used 24.0 to avoid integer division
        System.out.print("days: ");
        System.out.printf("%.3f",days);
        // -------- Specifying Width --------
        System.out.printf("%-10.2f%n", 18.0); // left aligned: -
        System.out.printf("%10.2f%n", 20.0); // right aligned
        System.out.printf("%10.3f", 10.2); // no text
        System.out.printf("%n"); // only %n
        System.out.printf("%10.2f%5d%n", 15.7,3); // no text and %n
        System.out.printf("%10.2f%d%n", 15.7,3); // no space before 3
        System.out.printf("%10.2f", 15.7);
        System.out.printf("%n%10.2f%n%5d%n", 11.3,8);
        // --- mix different data types --------------
        System.out.printf("%s is %d years old","Jane", 23);     }
}
```

Example Three- **printf** Method

Scenario: Suppose you want to print a text with the birth date of a person. The following is an example of such a text:

```
January 16, 1970  is Mike's birth day. Let's go and celebrate.
January 11, 1971  is John's birth day. Let's go and celebrate.
January 16, 1972  is Jane's birth day. Let's go and celebrate.
January 16, 1973  is Kite's birth day. Let's go and celebrate.
```

The above text contains fixed text and formatted text. In order to reuse the format and fill in the person's name and the birthday, we can write a template as follows:

```
<month> <day>,  <year>  is <name>'s  birth  day. Let's go and celebrate.
```

- The date of birth is enclosed in angle brackets, as is the name of the person. We can call them placeholders. Later on, we can provide real value for the placeholders.
- To use the template in Formatter class, we convert a placeholder to a format specifier. The template becomes a format string.
- A format specifier starts with a percent sign %.
- We can rewrite the template string, which can be used with the Formatter class as follows.

```
%1$tB %1$td,  %1$tY is %2$s's  birth  day. Let's go and celebrate.
```

In this format string, %1$tB, %1$td, %1$tY, and %2$s are format specifiers.

Example Three - printf Method (continued)

- The following code shows how to use the above format string to print formatted text.
- In the code, we created a LocalDate, which stores the birthday for Mike.
- The local date value and "Mike" become the input values for the format string.

```
import java.time.LocalDate;
import java.time.Month;
public class birthdayExample {
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        LocalDate dob = LocalDate.of(1971, Month.MAY, 16);
        System.out.printf(  "%1$tB %1$td,  %1$tY is %2$s's birth day. Let's go and celebrate.", dob,
"Mike");
    }
}
```

Console Output:

```
May 16, 1971 is Mike's birthday. Let's go and celebrate!
```

Topic 3: String.format() method

- ❑ String format method is used to format the string in the specified format. This method can be used as String output format by passing multiple arguments because the second parameter is the variable-arguments(Var-Args).
- ❑ The format function returns a String. The main advantage of `String.format()` over `System.out.printf()`and `System.out.format()` is its return type - it returns a `String.`

Java String.format() method Syntax:

```
String.format(String format, Object... args)
```

Java String Format Specifiers

%c – Character,
%d – Integer
%s – String
%f – Floating number
%h – hash code of a value

Note: `String.format()` is useful if you need to produce a simple formatted `String` for some purposes (e.g., used in method `toString()`). For complex string, use `StringBuffer/StringBuilder` with a `Formatter`. If you simply need to send a simple formatted string to the console, use `System.out.printf()`.

Please view our **wiki** document to learn more about **output formatting on console using String.format() method**.

Example - **String.format()**

```
public class Main {
    // The main function
    public static void main(String args[]) {

        // The objects to be used during formatting
        String s = "Per Scholas";
        float n = 74.47f;

        // The result formatted String objects
        String str1 = String.format("My Company name is %s", s),
                str2 = String.format("My age is %.8f", n);

        // Printing the resultant formatted Strings
        System.out.println(str1 + " " + str2);
    }
 }
```

Output:
My Company name is Per Scholas. My age is 74.47000122

Explanation:

In this example, we have obtained ***str1*** and ***str2***. They are String objects that are outputs of the format method. In the first ***format()*** call (the result of which is stored in String str1), we place the second argument String **"s"** at the position of the specifier ***"%s"*** in the first argument String literal.

In the second format() call (upon str2), we place the float *n* in the argument String at the position of the specifier ***"%f".***

Topic 4: Formatting Output on Console using - **DecimalFormat() class**

We often need to format numbers, such as by taking two decimal places for a number or show only the integer part of a number. Java provides a **DecimalFormat** class, which can help you to format numbers and use your specified pattern as quickly as possible.

DecimalFormat() class is actually designed to format any number in Java, be it ***integer, float or double***.

```
import java.text.DecimalFormat;
public class decimaldemo{
   public static void main(String[] args) {
      String pattern = "####,####.##";
      double number = 123456789.123;

      DecimalFormat numberFormat = new
DecimalFormat(pattern);

      System.out.println(number);

      System.out.println(numberFormat.format(number));
   }
}
```

Output

It will print the following result.

```
1.23456789123E8
1,2345,6789.12
```

Practice Problem: Monetary Units

Write a program that lets the user enter decimal dollars and cent and outputs the monetary equivalent in single dollars, quarters, dimes, nickels, and pennies.

- For example:

Input:	3.87
Output:	3 dollars
	3 quarters
	1 dime
	0 nickels
	2 pennies

Practice Assignment - 4

This assessment will be administered through HackerRank. Please complete the following Java practices assignment by opening the links below.

- java Datatypes
- java Int to String
- java Currency Formatter

If you have technical questions while performing the practice assignment, ask your instructors for assistance.

Note: It is NOT a mandatory assignment. This assignment does not count toward the final grade

Summary

- **print() -** prints string inside the quotes.

- **println() -** prints string inside the quotes similar to the print() method. Then, the cursor moves to the beginning of the next line.

- **printf() -** provides string formatting (similar to printf in C/C++ programming).

Lesson 5

Overview of Math Class

The Math Class provides commonly used mathematical functions such as square root, exponential(exp), logarithm(log), and trigonometric functions like sin, cosine, tan etc. The class provides several static methods for the mathematical functions. The basic math functions of the Java Math class will be covered in this sections.

By the end of this lesson, learners should be able to:

describe and demonstrate how to declare and initialize Math Class in Java.

TEKsystems Own change

Topic 1: The **Math** Class

- Java provides many useful methods in the Math class for performing common mathematical functions.
- **Math** class Methods Categories include:
  - Min, max, abs, and random methods.
  - Trigonometric methods.
  - Exponential methods.
  - Rounding methods.

- We cannot create objects of the Java Math class because all variables and methods of **Math** class are 'static.'
- **Math** class contains two ***public static final variables*** *also called* ***Constant***. Those are,
  - pi.
  - e.

Example - Math Class ***Constant***

The constants **e** and **π** are defined in the **Math** class. By convention, names of constants are all uppercase: • **Math.E** and **Math.PI**

Create a **MathDemo** class and write the code below in that class.

```
public class MathDemo{
   public static void main(String[] args) {
       System.out.println(Math.PI);
       System.out.println(Math.E);
   }
}
```

Output:

```
3.141592653589793
2.718281828459045
```

Min(), Max(), and Abs() methods

The following are common methods for Math class. All methods are static.

- max(a, b)and min(a, b).
  - Returns the maximum or minimum of two parameters.
- abs(a).
  - Returns the absolute value of the parameter.

Examples:

Math.max(2, 3)	returns 3
Math.max(2.5, 3)	returns 3.0
Math.min(2.5, 3.6)	returns 2.5
Math.abs(-2)	returns 2
Math.abs(-2.1)	returns 2.1

Example - Min(), Max(), and Abs() methods

```
public static void main(String[] args) {
   int a = 10, b = -20;
   System.out.println(Math.abs(a)); // 10
   System.out.println(Math.abs(b)); // 20
   System.out.println(Math.max(a, b)); // 10
   System.out.println(Math.min(a, b)); // -20
}
```

Output:

```
10
20
10
-20
```

Exponential and Logarithmic Methods

- exp(double a).

  Returns the base of natural log (e) to the power of argument.

- log(double a).

  Return the natural logarithm of `a`.

- log10(double a).

  Return the 10-based logarithm of `a`.

- pow(double a, double b).

  Return `a` raised to the power of `b`.

- sqrt(double a).

  Return the square root of `a`.

Examples:

Math.exp(1) →	returns 2.71
Math.log(2.71) →	returns 1.0
Math.pow(2, 3) →	returns 8.0
Math.pow(3, 2) →	returns 9.0
Math.pow(3.5, 2.5) →	returns 22.91765
Math.sqrt(4) →	returns 2.0
Math.sqrt(10.5) →	returns 3.24

TEKsystems
Own change

Example - Exponential Methods

```
public static void main(String[] args)
    {
      double x = 28;
       double y = 4;

        // return the square root of y
        System.out.println("Square root of y is: " + Math.sqrt(y));
         //returns 28 power of 4 i.e. 28*28*28*28
        System.out.println("Power of x and y is: " + Math.pow(x, y));
         // return the logarithm of given value
        System.out.println("Logarithm of x is: " + Math.log(x));
        System.out.println("Logarithm of y is: " + Math.log(y));
         // return the logarithm of given value when base is 10
        System.out.println("log10 of x is: " + Math.log10(x));
        System.out.println("log10 of y is: " + Math.log10(y));

        // return the log of x + 1
        System.out.println("log1p of x is: " +Math.log1p(x));

        // return a power of 2
        System.out.println("exp of x is: " +Math.exp(x));
    }
```

Output:-

```
Square root of y is: 2.0
Power of x and y is: 614656.0
Logarithm of x is: 3.332204510175204
Logarithm of y is: 1.3862943611198906
log10 of x is: 1.4471580313422192
log10 of y is: 0.6020599913279624
log1p of x is: 3.367295829986474
exp of x is: 1.446257064291475E12
```

Rounding Methods

The Math Class offers multiple methods to round off a number such as round(), ceil(), floor(), rint().

- double ceil(double x).
  - x rounded up to its nearest integer. This integer is returned as a double value.
- double floor(double x).
  - x is rounded down to its nearest integer. This integer is returned as a double value.
- double rint(double x).
  - x is rounded to its nearest integer. If x is equally close to two integers, the even one is returned as a double.
- int round(float x).
  - Return (int)Math.floor(x+0.5).
- long round(double x).
  - Return (long)Math.floor(x+0.5).

Rounding Methods - Example One

```java
public static void main(String[] args) {
  double d = 83.67;
  System.out.println(Math.ceil(d));  // return double & rounded up to its nearest integer
  System.out.println(Math.floor(d)); // return double and rounded down to its nearest integer
  System.out.println(Math.rint(d)); // return double but find the closest integers for these double numbers
  //Math.round() It is used to round of the decimal numbers to the nearest value.
  System.out.println(Math.round(d));  // 84;
  double a = 1.878;
  System.out.println(Math.round(a));  // 2
  // value equals to 5 after decimal
  double b = 1.5;
  System.out.println(Math.round(b));  // 2
  // value less than 5 after decimal
  double c = 1.34;
  System.out.println(Math.round(c));  // 1
}
```

The Random Method

- Math.random().

  The random method of the Math class generates a random number in the range [0.0, 1.0]

  (0.0 <= Math.random() < 1.0)

  includes 0.0

  excludes 1.0

  - Examples:

    (int) (Math.random()*10) //Returns a random integer between 0 and 9.

    50 + (int) (Math.random()*50) //Returns a random integer between 50 and 99.

  - In general:

    a + Math.random() * b //Returns a random number between a and a+b, excluding a+b.

Example One: The Random Method

Java program to generate integer numbers between 1 to 100.

```
public static void main(String[] args) {

     // generate integer number b/w 1 to 100
     int n1 = (int) (Math.random()*100);
     int n2 = (int) (Math.random()*100);

     // display generated integer numbers
     System.out.println(n1);
     System.out.println(n2);
 }
```

Output will be integer numbers between 1 to 100.

Example Two: The Random Method

Java program to generate number between 100 to 1000.

```
public static void main(String[] args) {

     // declare max and min
     int min = 100;
     int max = 1000;

     // calculate range
     int range = (max-min) + 1;

     // generate floating-point number b/w 100 to 1000
     double d1 = Math.random()*range + min;

     // generate integer number b/w 100 to 1000
     int n1 = (int)(Math.random()*range + min);

     // display result
     System.out.println(d1);
     System.out.println(n1);
   }
```

Output will integer numbers between 100 to 1000.

Trigonometric Methods

The Math class contains methods for trigonometric operations such as cos(), sin(), tan(), tanh(), cosh(), atan(), etc.

Please go to our **wiki** document to learn more about trigonometric operations.

Summary

Wrapper classes provide a way to use primitive data types (int, boolean, etc.) as objects. We explored wrapper classes in Java, as well as the mechanism of autoboxing and unboxing.

Character is one of the most basic units of data in Java. It is a primitive data type that we can convert into the objects of its respective wrapper class, called the Character Class.

The String class is immutable; once it is created, a String object cannot be changed. If there is a necessity to make modifications to Strings of characters, the class String includes methods for examining individual characters of the sequence for comparing strings, searching strings, extracting substrings, and creating a copy of a string with all characters translated into uppercase or lowercase characters. Case mapping is based on the Unicode Standard.

References

https://linuxhint.com/rounding-numbers-java/

https://www.baeldung.com/java-wrapper-classes

https://www.scaler.com/topics/java-string-format/

Questions?

End of Module