2.5

Methods that return values:

- We have been discussing methods that don't return values so far.

ex:
```
public class Calculator {
    public static void main (String[] args) {
        int num1 = 4;
        int num2 = 5;
        calculateSum(num1, num2);
    }
    public static void calculateSum (int a, int b) {
        System.out.println(a + b);
    }
}
```

Note that we are not storing the result of this method in a variable like int result = calculateSum(num1, num2). We cannot do this since calculateSum method does not return a value (void).

- We can write methods that can return a value.
  - Method signature will have a return data type like int, double, String.
  - Inside the method, it will also have a return keyword with a value that should have the same data type as the signature.

ex:
```
public class Calculator {
    public static void main (String[] args) {
        int num1 = 4;
        int num2 = 5;
        int result = calculateSum(num1 + num2)
    }
    public static int calculateSum (int a, int b) {
        return (a+b);
    }
}
```

Note that we are storing the value that we get from calculateSum method to result variable

Note that the method signature says that the method will return an int value

This is the return keyword. It returns the sum of a and b.

- It will give you an error if the method was returning a double but the method signature was expecting an int.

ex:
```
public static int calculateSum () {
    return (5.0)
}
```

int a ← expects int a

↳ returns a double value

- However, it will not give you an error if the method was returning an int but method signature was expecting a double

ex:
```
public static double calculateSum () {
    return (5)
}
```

You can store int to a double but not store double to int.

- You can also not store a string to a double, string to int etc.

- Example of storing values returned from a method to a variable.

```
public static class Calculator {
    public static void main(String[] args) {
        int x = 5;
        int y = 10;
        int result = calculateSum(x,y);
    }
}
```

Note that result is an int variable (This would still work if the variable was double since we can store int values to double variable)

//will be replaced by 15 and stored in result.

we are able to store what gets returned from the method to result variable

```
public static int calculateSum(int num1, int num2) {
    return (num1 + num2);
}
}
```

method signature returns int value

we are returning int value.

This code will not work if we were returning a double value because result variable in the main method has a data type of int. we cannot store a double value (from calculateSum method) to result variable (int).

Accessors / Getters:

- Accessors / Getters are methods that we can use to access a value in an object.
- These methods have a <u>non-void</u> return type since they return a value.
- These methods also takes no arguments. It is solely to access value in an object.

```
ex: public class DataStore {
        public static void main(String[] args) {
            Secret s1 = new Secret();
            System.out.println(s1.getSecret());
        }
    }
    public class Secrets{
        String password;

        Secrets() {
            password = "abcde";
        }
        public String getSecret() {
            return password;
        }
    }
```

create a new object s1 and initialize with constructor (no parameter)

accessing a value of s1 object from getter method.

Constructor with no parameters.

non-void return type

no parameters.

This is how getter methods are written

2.6

Strings:

Strings are Java objects of the String class that hold sequence of characters (a, b, c, &, etc).
That is why String data type starts with a capital letter unlike int, double etc.
(String: class types; int, double: primitive types)

- You can assign a null keyword to string variable
  eg: String name = null; ← keywords don't need quotations (" ").

- Ways to use a String variable
  String s1 = new String("hello"); ← This is creating s1 object from class String. We discussed earlier that String is a class type.

  OR

  String s2 = "hello"; ← This is an easier way but it is doing the same thing.

- Concatenation
  Use + operator to join multiple strings
  ex: String s1 = "hello";
  String s2 = "world";
  System.out.println(s1 + " " + s2); ← using + operator to join strings.

Primitive values (int, boolean, etc) will be converted to a string when you add them with a String.
ex: String message = "12" + 4 + 3;
System.out.println(message); ← This will print 1243. It converts 4 and 3 to String since "12" is a string.

Backslash is used for special characters (" , \ )
Use case:

- Using a quotation mark in a string.
  ex: "Here is a quotation mark \" " ← \" prints the quotation mark which we wanted instead of giving an error.

- Printing a backslash.
  ex: "Here is a backslash \\" ← This prints: Here is a backslash \.
  we need a backslash to print a backslash.

- Print a string in multiple lines
  ex: "Line 1 \n Line 2" ← This prints: Line 1
  Line 2.
  \n makes sure that the string after \n goes to the next line.

2.7 String Methods

- length: returns the length of a string

ex: String name = "Hello World";
System.out.println(name.length()); ← Prints 11

- Substring(from, to): Returns substring from index up to (but not including) the to index

String name: " H e l l o W o r l d "
Index: 0 1 2 3 4 5 6 7 8 9 10
(Index starts with 0)

System.out.println(name.substring(0, 3)); ← Prints Hel
System.out.println(name.substring(4, 8)); ← Prints o Wo

- subString(from): Sets the starting point of the string to the end of the string

String name: " H e l l o "
System.out.println(name.substring(1)); ← Prints ello

- indexOf(substring): Searches for the substring and returns the (first) index of where it is.
Returns -1 if it doesn't exist.

String name = "Hello"
System.out.println(name.indexOf("lo"); ← Prints 3
System.out.println(name.indexOf("l"); ← Prints 2.
System.out.println(name.indexOf("ali"); ← Prints -1

- CompareTo: compares two strings character by character. ← Method is case sensitive H is different from h
  - If they are equal, it returns 0
  - If the first string is alphabetically ordered before the second string, it returns a negative number.

ex: String s1: "Hello";
String s2: "Hello!";
System.out.println(s1.compareTo(s2)); ← prints a negative number.

  - If the first string is alphabetically ordered after the second string, it returns a positive number

ex: String s1: "Hello";
String s2: "Hello!";
System.out.println(s2.compareTo(s1)); ← prints a positive number.

↓ Not super important to know.

Note: The actual number it returns does not matter, but it is the distance in the first letter that is different.

ex: String s1: "Apollo";
String s2: "Hello";
System.out.println(s2.compareTo(s1)); ← Prints (7) ← Since H is 7 letters ahead of A

- Equals: The Equals method compares two strings character and returns true or false. ← method is case sensitive. H is different from h.

ex: String s1: "Hello";
String s2: "World";
System.out.println(s1.equals(s2)); ← Prints false

2.8

Wrapper Classes: Integer and Double

- For every primitive type in Java, there is a built-in object type called a wrapper class.
  - * wrapper class for int is called Integer
  - * wrapper class for double is called Double

→ Note that wrapper classes start with capital letters. This is because these are of class type just like Strings.
→ This is just like writing a class (e.g Student class). Java wrote these built-in classes so we can make use of the methods they wrote for us.

→ Storing a value using Integer / Double wrapper classes

```
Integer i = new Integer(2);   ← just like creating a new Integer object using constructor that takes 1 int parameter
Double d = new Double(3.5);   ← just like creating a new Double object.
```

OR

```
Integer i = 2;
Double d = 3.5;
```
} Newer and simpler way of writing. It still creates a new object.

→ Using wrapper methods:

As mentioned earlier, we can use built-in methods that are created for us for these wrapper classes.

```
ex: public static void main(String[] args) {
        System.out.println(Integer.MIN_VALUE);   Prints: -2147483648
        System.out.println(Integer.MAX_VALUE);   Prints: 2147483647
    }
```

There are more methods like these that we can use. We will cover in later chapters.

← If you print INTEGER.MIN_VALUE - 1, it would print 2147483647 (called underflow)

← If you print INTEGER.MAX_VALUE + 1, it would print -2147483648 (called overflow)

} There is a whole theory behind it since ints are represented in binaries. There is no need to go in detail.

→ Autoboxing and unboxing:

Autoboxing: automatic conversion of primitive data type (ex: int) to their corresponding object wrapper classes (e.g: Integer)

```
ex: Integer i = 2;
```
Integer: storing to a wrapper class. 2: primitive int value

Unboxing: automatic conversion of wrapper class (ex: Double) to the primitive type (ex: double)

```
ex: Double d = 2.5;   ← autoboxing example
    double n = i      ← unboxing (back to primitive)
```

2.9

Math Class

→ This is another example of built-in Math class that Java created for us so we can use the methods in Math class.

Note: These are static methods so we can call them directly using "Math.methodName()" without creating an object.
ex: Math.random()

→ Some examples of Math class methods.

- int abs(int) : returns absolute value of an int (e.g. Math.abs(-4) returns 4)
- double abs(double) : returns absolute value of a double (e.g. Math.abs(-4.0) returns 4.0)
- double pow(double, double) : returns the value of first parameter raised to the power of second parameter (Math.pow(2,3) returns 8.0)
- double sqrt(double) : returns a positive squareroot of a double value (Math.sqrt(9) returns 3.0)
- double random() : returns a double value greater than or equal to 0.0 and less than 1.0 (not including 1.0) (Math.random() returns 0.543... ← or any random value.)