

Module 1-5

Inputs and Outputs

- Methods
- Command Line

Objectives

- Be able to use `System.in/System.out` to perform console I/O in a program
- Be able to correctly parse input from the input to primitive data types
- Be able to check for string equality
- Be able to split a string apart using known split character
- Be able to explain the process of a command line application (Take input, calculate data, give output)
- Be able to run their command line apps in their IDE

Arrays and For Loop - review

Array: cities

index	value
0	Columbus
1	Cleveland
2	Cincinnati
3	Pittsburgh
4	Detroit

Loops in Java



For Loop - review

- `for (int i = 0 ; i < cities.length ; i++) {`
 - `String cityName = cities[i];`
 - `}`
1. <http://dashboard.tehelevator.com/te-explanations/loops/for.html>

Methods

Methods

- Methods are **related** (hint: {...}) statements that complete a specific task or set of tasks.
- Methods can be called from different places in the code.
- When called, inputs can be provided to a method.
- Methods can also return a value to its caller.

Methods: General Syntax

Here is the general syntax:

```
<access Modifier> <return type> <name of the method> (... params...) {  
    // method code.  
}
```

- The return type can be one of the data types (boolean, int, float, etc.) we have seen so far.
- If the return type is “void” it means nothing is returned by the method.

Methods: Example

Here is a specific example of a non-void method:

```
public class MyClass {  
  
    public int addTwoNumbers(int a, int b) {  
        return a+b;  
    }  
}
```

The method `addTwoNumbers` is a method of the `MyClass` class.

The method accepts 2 parameters as input. More specifically, it expects 2 integers

The method has a return value of `int`, so there needs to be a return statement that returns an integer.

Methods: Example

Here is a specific example of a void method:

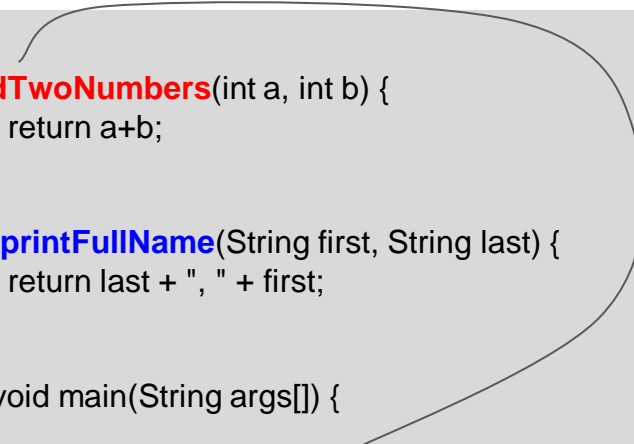
```
public class MyClass {  
    public void addTwoNumbers(int a, int b) {  
        System.out.println(a+b);  
    }  
}
```

This method is void, thus has no return statement.

Methods: Calling A Method

Methods can be called from other methods.

```
public class MyClass {  
    public int addTwoNumbers(int a, int b) {  
        return a+b;  
    }  
  
    public String printFullName(String first, String last) {  
        return last + ", " + first;  
    }  
  
    public static void main(String args[]) {  
  
        int result = addTwoNumbers(3,4);  
        System.out.println(result);  
        // result will be equal to 7.  
  
        String fullName = printFullName("Minnie", "Mouse");  
        System.out.println(fullName);  
        // result will be equal to "Mouse, Minnie"  
  
    }  
}
```



In here, we call the method **printFullName** from **callingFunction**, providing all needed parameters and saving the result into result.

Methods: Calling A Method

Once a method has been defined, it can be called from somewhere else.

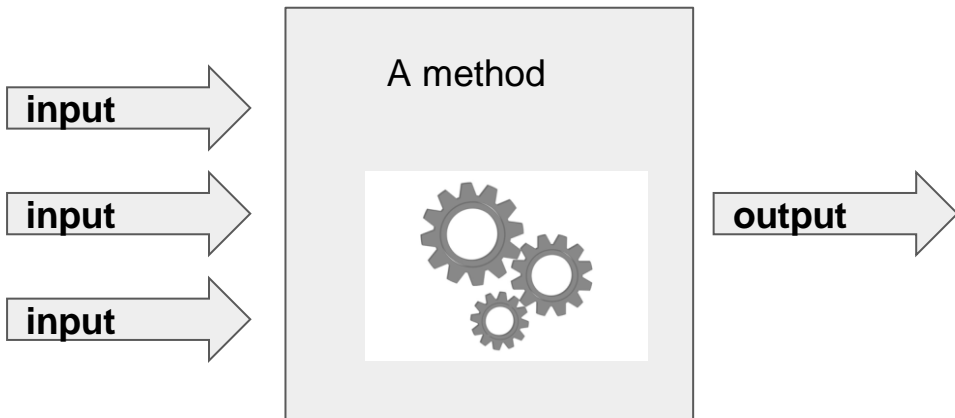
```
public class MyClass {  
    public int addTwoNumbers(int a, int b) {  
        return a+b;  
    }  
  
    public void callingMethod ( ) {  
  
        int result = addTwoNumbers(3,4);  
        System.out.println(result);  
        // result will be equal to 7.  
    }  
}
```

addTwoNumbers takes 2 inputs, an integer a and an integer b. These are known as **parameters**.

When we call **addTwoNumbers**, we must provide the exact inputs specified (in this case 2 integers).

Methods: Example

Methods are Java's versions of functions. You can think of this as a process that could potentially take several inputs and use it to generate output.



Command Line Input / Output

Getting Input from the Command Line

- All programming languages must have the ability to read in data (input)
- Examples of input: a file, data being transmitted from a network, or **data typed in by the user.**

System.in and System.out

- Refer to standard input and output streams.
- System.in refers to the keyboard
- System.out refers to the console (monitor/terminal)
- To read from the keyboard, we need to create a Scanner object
Scanner input = new Scanner(System.in);

Using the Scanner Object

```
import java.util.Scanner;

public class InputReader {

    public static void main(String[] args) {

        Scanner userInput = new Scanner(System.in);

        System.out.print("Please enter your name: ");
        String name = userInput.nextLine();

        System.out.print("Please enter your height: ");
        String heightInput = userInput.nextLine();
        int height = Integer.parseInt(heightInput);

        System.out.println("Your name is: " + name + ".");
        System.out.println("Your height is: " + height +
            " inches.");
    }
}
```

To use the scanner object, we must import in the correct class.

Create an object of type scanner

The input is read and stored into a String called name.

The input is read and stored into a String called heightInput.

heightInput is converted into an int using the **Integer Wrapper Class**.

Parsing Strings

- What data type is the input from command line?
 - Always Strings!
- How do we “convert” this to numbers, dates, times, decimals, etc.?
 - Parse the string to convert it to a different data type

Java:

```
Integer.parseInt(String s);  
Long.parseLong(String s);  
Double.parseDouble(String s);  
Boolean.parseBoolean(String s); // Boolean s is a lower case s
```

Wrapper Classes

- Up until now, we have seen most of the primitive data types, to name a few: **int**, **boolean**, **char**, **long**, **float**...
- You have also seen some non-primitive types: **Strings** and **Arrays**
- You might have noticed that non-primitive types seem to have extra functionality that can be invoked with the dot operator, for example: **(myArray.length)**.
- All the primitive data types have more powerful non-primitive equivalents, these are called **wrapper classes**. You have seen an example of this.

```
int height = Integer.parseInt(heightInput);
```

* albeit this example uses a static method of the wrapper class (more on this at a later date)

Wrapper Classes

Primitive	Wrapper	Example of Use
int	Integer	Integer myNumber = 3;
double	Double	Double myDouble = 3.1;

Declaring a variable using the Wrapper class gives you a little bit more flexibility. For example, you are able to run certain utility methods by using the dot operator.

```
Integer myNumber = 3;  
String myStringNumber = myNumber.toString();
```

In the above example we have used a Wrapper class, and then a method of that class (toString()) to convert the value to a String. In general, if you know type conversions will be involved, Wrapper classes might be a good idea.

String equality

- Code example
- Difference between `==` and `.equals()` method
 - `==` can only be used with primitive data types
 - `.equals` should be used on reference types (Class objects)

```
String s1 = new String("HELLO");  
String s2 = new String("HELLO");  
System.out.println(s1 == s2);           // false  
System.out.println(s1.equals(s2));      // true
```

Reading In Multiple Items

```
import java.util.Scanner;

public class InputReader {

    public static void main(String[] args) {

        Scanner userInput = new Scanner(System.in);
        System.out.print("Please enter several names: ");
        String lineInput = userInput.nextLine();

        String [] inputArray = lineInput.split(" ");

        for (int i=0; i < inputArray.length; i++) {

            System.out.println(inputArray[i]);
        }
    }
}
```

This is one possible way to handle input for more than one item.

- When prompted a user enters each item separated by a space.
- The split method separates out each time using the spaces, and puts all of the items into an array!

Reading In Multiple Items

```
1 import java.util.Scanner;
2
3 public class InputReader {
4
5     public static void main(String[] args) {
6
7         Scanner userInput = new Scanner(System.in);
8         System.out.print("Please enter several objects: ");
9         String lineInput = userInput.nextLine();
10
11         String [] inputArray = lineInput.split(" ");
12
13         for (int i=0; i < inputArray.length; i++) {
14             System.out.println(inputArray[i]);
15         }
16     }
17 }
```

Console x

<terminated> InputReader [Java Application] C:\Program Files\Java\jre1.8.0_211\bin\javaw.exe (Sep

Please enter several objects: Ford GM Chrysler Toyota Honda Nissan BMW

Ford

GM

Chrysler

Toyota

Honda

Nissan

BMW

The user entered each car brand separated by a space

The whole input is “split” and repackaged as an array

Objectives

- Be able to use System.in/System.out to perform console I/O in a program

```
import java.util.Scanner;

public class InputReader {

    public static void main(String[] args) {

        Scanner userInput = new Scanner(System.in);

        System.out.print("Please enter your name: ");
        String name = userInput.nextLine();

        System.out.println("Your name is: " + name + ".");

    }
}
```


Objectives

- Be able to use System.in/System.out to perform console I/O in a program
- Be able to correctly parse input from the input to primitive data types

```
import java.util.Scanner;

public class InputReader {

    public static void main(String[] args) {

        Scanner userInput = new Scanner(System.in);

        System.out.print("Please enter your height: ");
        String heightInput = userInput.nextLine();
        int height = Integer.parseInt(heightInput);

        System.out.println("Your height is: " + height +
                           " inches.");
    }
}
```

Objectives

- Be able to use System.in/System.out to perform console I/O in a program
- Be able to correctly parse input from the input to primitive data types
- Be able to check for string equality



```
If (name == "Bob") {
```

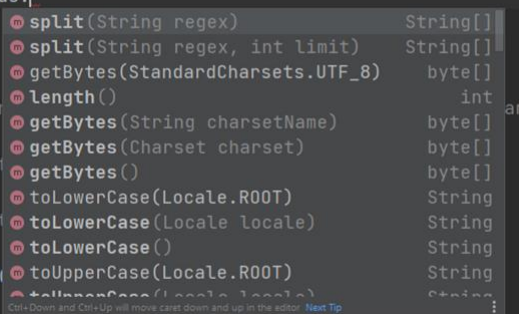
```
If (name.equals( "Bob" )) {
```

Objectives

- Be able to use System.in/System.out to perform console I/O in a program
- Be able to correctly parse input from the input to primitive data types
- Be able to check for string equality
- Be able to split a string apart using known split character

```
String words = "These are my words";
String[] wordArray = words.
}

/*
2. Given an array of ints,
the last element are equal.
sameFirstLast([1, 2, 3]) →
sameFirstLast([1, 2, 3, 1])
sameFirstLast([1, 2, 1]) →
*/
public boolean sameFirstLast(
    return false;
}
```



```
String words = "These are my
words";
String[] wordArray =
words.split(" ");
String words = "These are my
words";
String[] wordArray =
words.split("\\s");
```

Objectives

- Be able to use `System.in/System.out` to perform console I/O in a program
- Be able to correctly parse input from the input to primitive data types
- Be able to check for string equality
- Be able to split a string apart using known split character
- Be able to explain the process of a command line application (Take input, calculate data, give output)



Input Process Output

Objectives

- Be able to use System.in/System.out to perform console I/O in a program
- Be able to correctly parse input from the input to primitive data types
- Be able to check for string equality
- Be able to split a string apart using known split character
- Be able to explain the process of a command line application (Take input, calculate data, give output)
- Be able to run their command line apps in their IDE



Command Line Programs (Second Deck)

Module 01 : 05

Today's Objectives

1. Loops Continued
2. Troubleshooting and Debugging
3. System.in
4. Parsing Strings
5. System.out
6. Command Line Arguments

Increment / Decrement Shorthand

i++ → use the current value of i, and then $i = i + 1$;

++i → $i = i + 1$, then use the new value of i

i-- → use the current value of i, then $i = i - 1$;

--i → $i = i - 1$, then use the new value of i

For readability, the general standard is to only use these increment / decrement shorthand in the for() of a loop, or on lines by themselves.

for (; ; i++) ← OK

i++;

← OK

~~**x = 120 + --y - 5;**~~ ← Discouraged

Assignment Shorthand

x += y → $x = x + y$;

x -= y → $x = x - y$;

x *= y → $x = x * y$;

x /= y → $x = x / y$;

Command Line Programming

```
Dark Tunnel                               Score: 0           Moves: 6
Foot Bridge
You are standing on a crude but sturdy wooden foot bridge crossing a deep
ravine. The path runs north and south from here.

>go south
Great Cavern
This is the center of the great cavern, carved out of the limestone.
Stalactites and stalagmites of many sizes are everywhere. The room glows with
dim light provided by phosphorescent moss, and weird shadows move all around
you. A narrow path winds southwest among the stalagmites, and another leads
northeast.

>go southwest
Shallow Ford
You are at the southern edge of a great cavern. To the south across a shallow
ford is a dark tunnel which looks like it was once enlarged and smoothed. To
the north a narrow path winds among stalagmites. Dim light illuminates the
cavern.

>go south
You have moved into a dark place.
It is pitch black. You are likely to be eaten by a grue.

>
```

System.in

Java **System.out** and **System.in** refer to the standard input and output *streams*. The *System.in* captures characters a user types into the console, and *System.out* outputs characters to the console to be read by the user.

While *System.out* provides easy to use methods, like `println()`, *System.in* is more difficult to manage. The `Scanner` wrapper class can be used to make it easier.

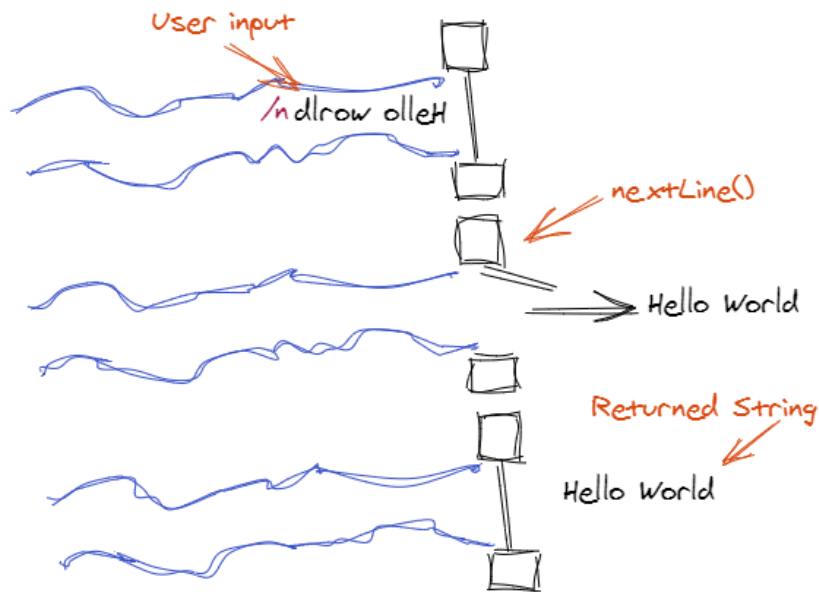
```
import java.util.Scanner;

Scanner in = new Scanner(System.in);
String userInput = in.nextLine();
```

Scanner nextLine()

`scanner.nextLine()` gets text from input stream up until a newline (the user presses Enter). The text is returned as a `String` and the newline is digarded.

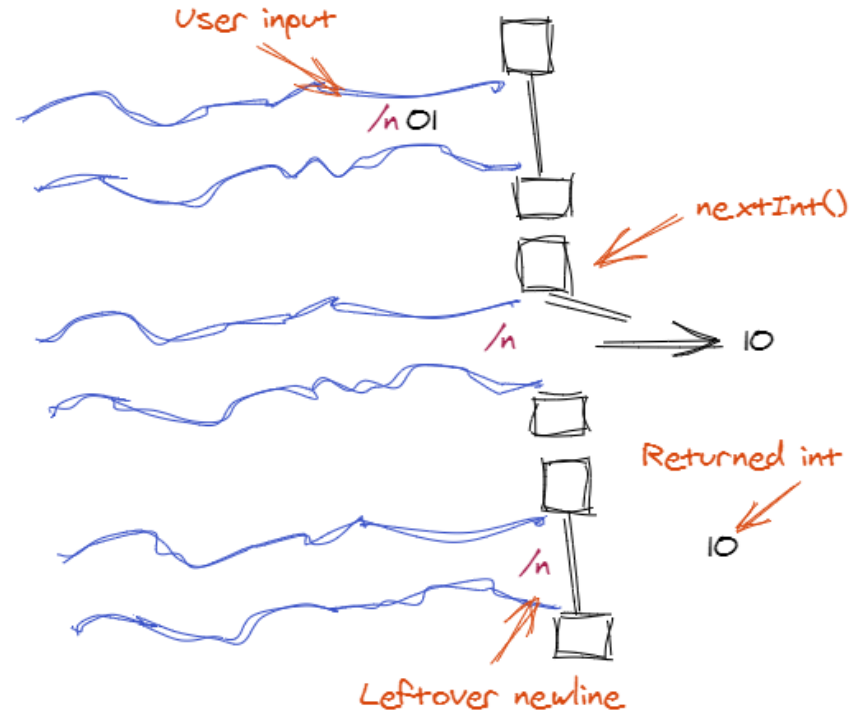
```
String userInput = scanner.nextLine();
```



Scanner nextInt()

Scanner also provides methods to get the next numeric values from the stream. It gets the next characters that can be converted to the numeric data type and leaves the rest.

A newline character is not a numeric type, so it will be left on the stream. So after using methods like `nextInt()`, `nextLine()` must be called to “clean up” the newline.



```
int userNumber = scanner.nextInt();  
scanner.nextLine();
```

Parsing Strings

When a data is converted between like data types, it is called **casting**. **Parsing** is the conversion of data between *unlike* data types. For example: String → int

Parse methods that can parse a string to that data type are available for each of the basic data types using their wrapper class. The String must contain characters that are valid for the data type it is being parsed into.

Data Type	Wrapper	Parse Method
int	Integer	Integer.parseInt(string)
long	Long	Long.parseLong(string)
double	Double	Double.parseDouble(string)
boolean	Boolean	Boolean.parseBoolean(string)

System.out

println(string) → prints the string to the console and adds a newline

print(string) → prints the string to console and does not add a newline

println(format, data) → prints the data to the console using the provided format.

System.out.printf("Your item is %-10s", "book"); → adds book so that it always takes up 10 spaces

%-10s

% - Starts a formatter

-10 - 10 sets the size to 10, - adds any padding on the left.

s - defines the data type being formatted as a String

- s – formats strings
- d – formats decimal integers
- f – formats the floating-point numbers
- t – formats date/time values

System.out.printf("Total Cost \$%4.2f", 4.2507); → formats 4.2507 to a total of 4 characters and 2 decimal places

%4.2f

% - Starts a formatter


4.2 - sets the total size to 4 characters and .2 sets it to 2 decimal places

f - defines the data type being formatted as a floating point number

Command Line Arguments

Arguments passed on the command line to an application are populated into `String[] args` array of the `main()` method.

`$ git 0commit 1-m 2"my comment"`



```
public static void main( String[] args ) {  
    args[ 0 ] == "commit"  
    args[ 1 ] == "-m"  
    args[ 2 ] == "my comment"  
}
```