Command Line Programs

You Made It! Happy Friday!

Weekly Calendar

MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
✗	✗	✗	✗	

Today's Objectives

- Using System.in/System.out/Console.ReadLine() to perform console I/O in a program.
- Parsing input from the input stream to primitive data types.
- Check for string equality.
- Convert a String list with a known separator character into an array.
- Command line application process: Take input, calculate data, give output.
- Running command line apps in your IDE

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>** (... arguments…) {
    // 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: Examples

Methods can accept input parameters but are not required to.

```
public void printHello() {
     System.out.println("Hello, World!")
}
```

Methods: Examples

No input parameters

Methods can accept input parameters but are not required to.

```
public void printHello() {
    System.out.println("Hello, World!")
}
```

Methods: Examples

No input parameters

Methods can accept input parameters but are not required to.

```
public void printHello() {
     System.out.println("Hello, World!")
}
```

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

Two input parameters of type `int`.

Methods: Examples

Methods can return a value but are not required to.

```
public void printHello() {
     System.out.println("Hello, World!")
}
```

Methods: Examples

No return value (`void` return type specified). MUST NOT return a value.

```
public void printHello() {
    System.out.println("Hello, World!")
}
```

Methods can return a value but are not required to.

Methods: Examples

No return value (`void` return type specified). MUST NOT return a value.

Methods can return a value but are not required to.

```
public void printHello() {
    System.out.println("Hello, World!")
}
```

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

Returns an `int` value. If the return type is not `void`, the method MUST return a value of the specified type.

Method Signature

Methods have a signature, which is made up of:

- Name (should be descriptive)

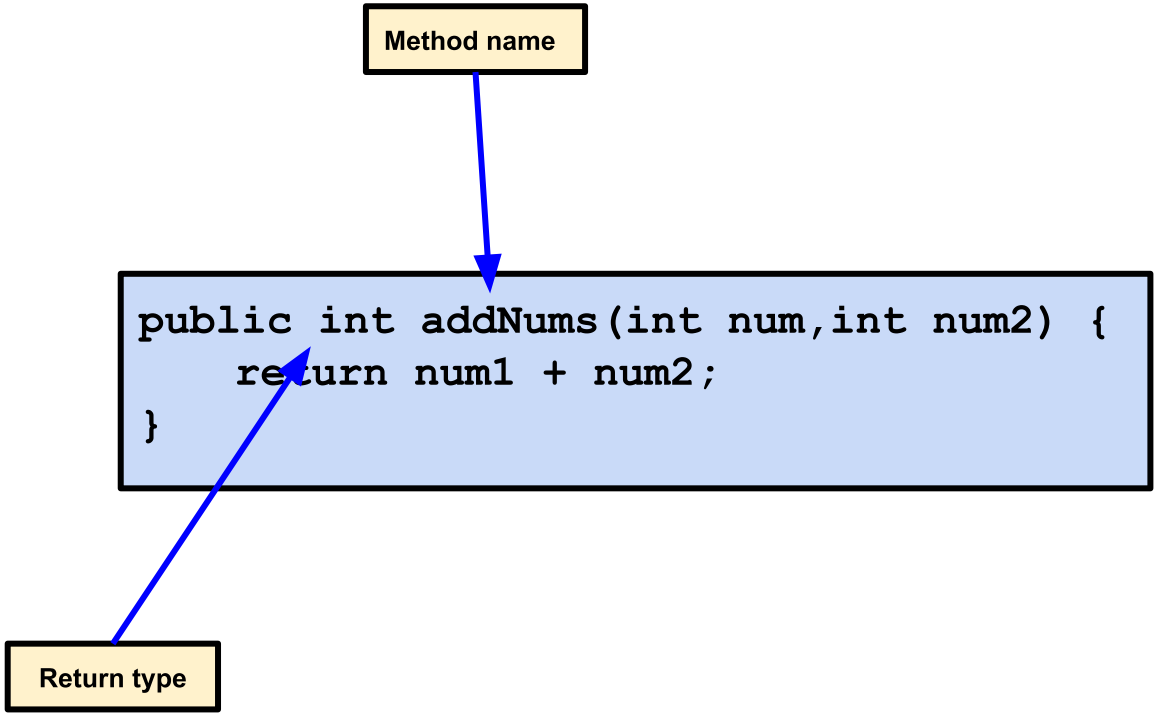
Method name

```
public int addNums(int num,int num2) {
    return num1 + num2;
}
```

Method Signature

Methods have a signature, which is made up of:

- Name (should be descriptive)
- Return Type (e.g. int, long, double, float, boolean, …)

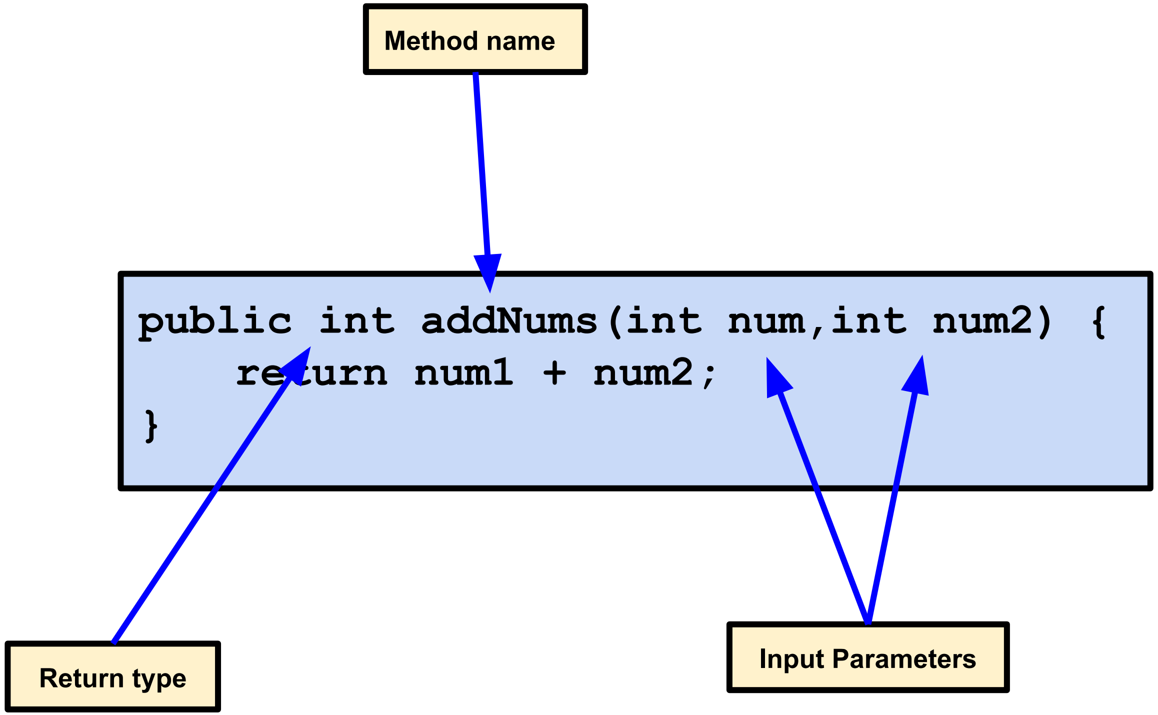


```
public int addNums(int num,int num2) {
    return num1 + num2;
}
```

Method Signature

Methods have a signature, which is made up of:

- Name (should be descriptive)
- Return Type (e.g. int, long, double, float, boolean, …)
- Input Parameters



```
public int addNums(int num,int num2) {
    return num1 + num2;
}
```

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 void callingFunction (String args[]) {

              int result = addTwoNumbers(3,4);
              System.out.println(result);
              // result will be equal to 7.

              String fullName = printFullName("Andy", "Chong");
              System.out.println(fullName);
              // result will be equal to "Chong, Andy"
       }
}
```

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 void callingFunction (String args[]) {

                int result = addTwoNumbers(3,4);
                System.out.println(result);
                // result will be equal to 7.

                String fullName = printFullName("Andy", "Chong");
                System.out.println(fullName);
                // result will be equal to "Chong, Andy"
        }
}
```

`callingFunction` makes a call to **`printFullName`** providing all needed parameters and saving the returned value into **`result`**

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 String printFullName(String first, String last) {
              return last + ", " + first;
       }

       public void callingFunction (String args[]) {

              int result = addTwoNumbers(3,4);
              System.out.println(result);
              // result will be equal to 7.

              String fullName = printFullName("Andy", "Chong");
              System.out.println(fullName);
              // result will be equal to "Chong, Andy"
       }
}
```

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 String printFullName(String first, String last) {
              return last + ", " + first;
       }

       public void callingFunction (String args[]) {

              int result = addTwoNumbers(3,4);
              System.out.println(result);
              // result will be equal to 7.

              String fullName = printFullName("Andy", "Chong");
              System.out.println(fullName);
              // result will be equal to "Chong, Andy"
       }
}
```

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

Calling a Method

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

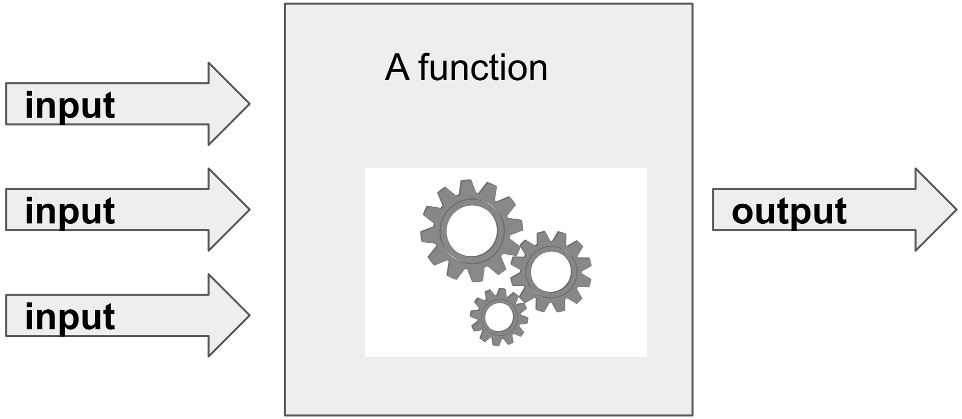
```
public class MyClass {
       public int addTwoNumbers(int a, int b) {
              return a+b;
       }

       public String printFullName(String first, String last) {
              return last + ", " + first;
       }

       public void callingFunction (String args[]) {

              int result = addTwoNumbers(3,4);
              System.out.println(result);
              // result will be equal to 7.

              String fullName = printFullName("Andy", "Chong");
              System.out.println(fullName);
              // result will be equal to "Chong, Andy"
       }
}
```

`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**.

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 + " cm's.");
      }
}
```

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 + " cm's.");
      }
}
```

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

Using the Scanner Object

```java
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 + " cm's.");
      }
}
```

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

Create an object of type `Scanner`

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 + " cm's.");
      }
}
```

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`**.

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 + " cm's.");
      }
}
```

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`.**

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 + " cm's.");
      }
}
```

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**.

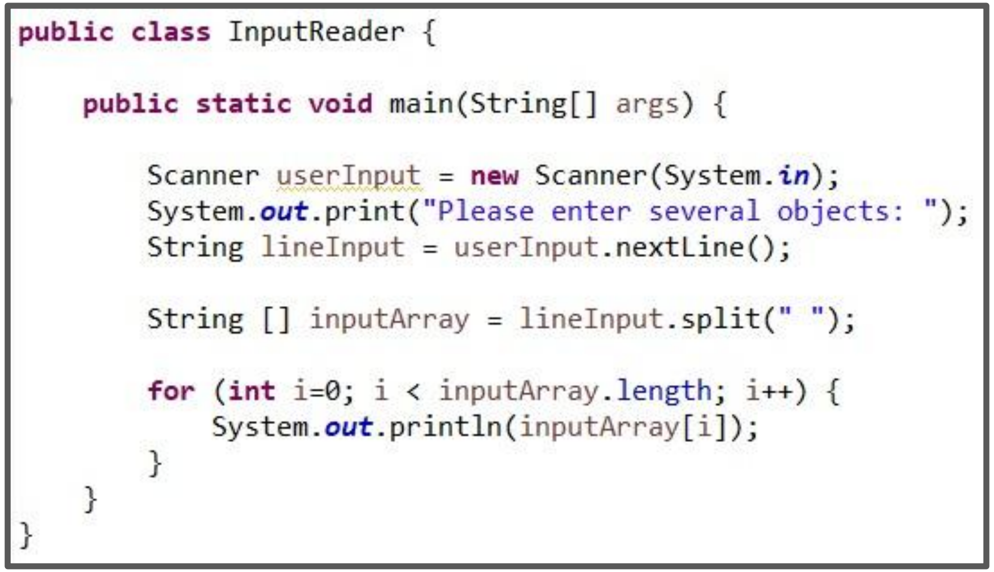
Reading In Multiple Items

```
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 objects: ");
          String lineInput = userInput.nextLine();

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

          for (int i=0; i < inputArray.length; i++) {
               System.out.println(inputArray[i]);
          }
     }
}
```

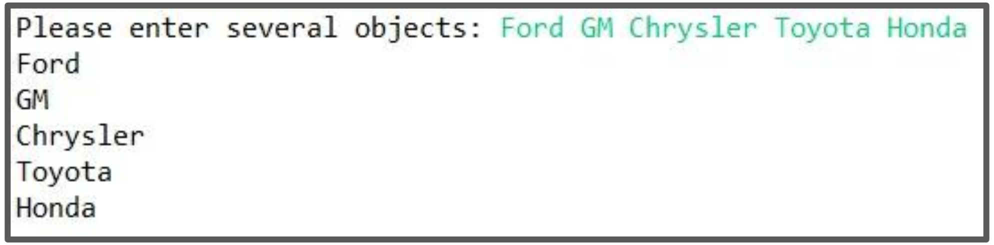
This is one possible way to handle input for more than one item.

- 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

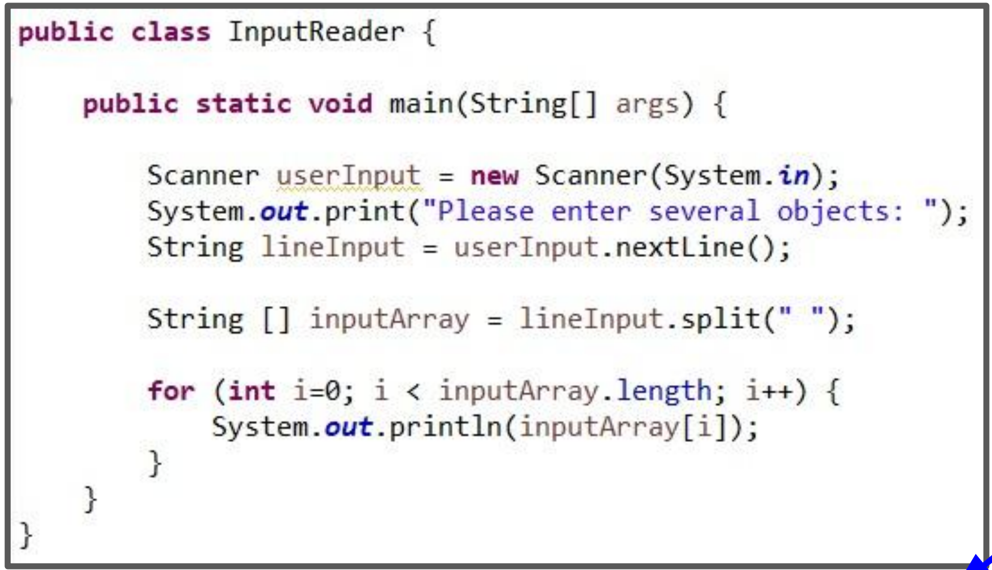


```
public class InputReader {

    public static void main(String[] args) {

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

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

        for (int i=0; i < inputArray.length; i++) {
            System.out.println(inputArray[i]);
        }
    }
}
```

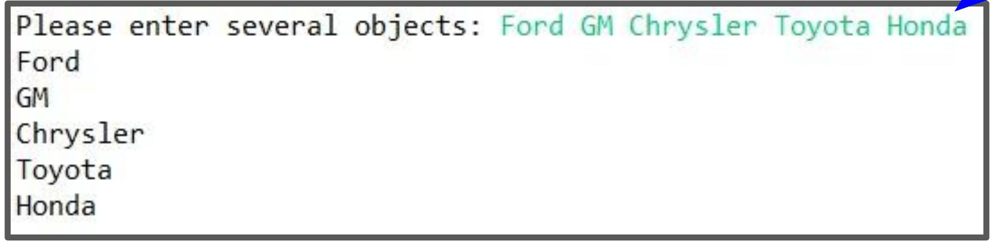


```
Please enter several objects: Ford GM Chrysler Toyota Honda
Ford
GM
Chrysler
Toyota
Honda
```

Reading In Multiple Items



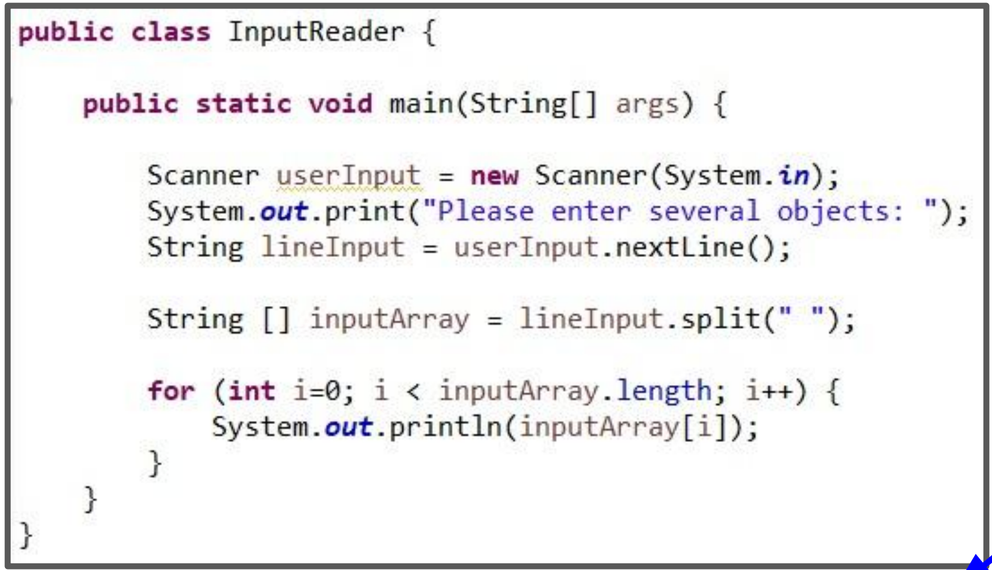
```
public class InputReader {

    public static void main(String[] args) {

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

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

        for (int i=0; i < inputArray.length; i++) {
            System.out.println(inputArray[i]);
        }
    }
}
```



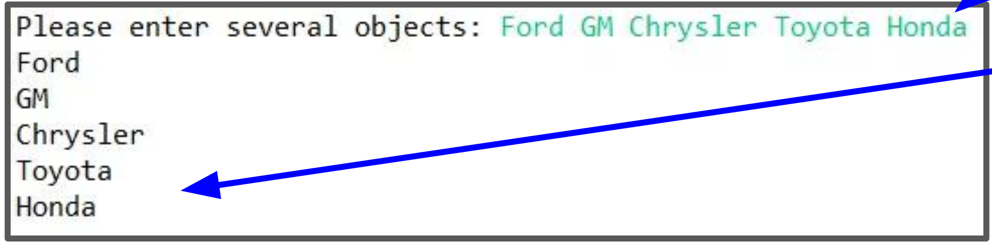
```
Please enter several objects: Ford GM Chrysler Toyota Honda
Ford
GM
Chrysler
Toyota
Honda
```

The user entered each car brand separated by a space

Reading In Multiple Items



```java
public class InputReader {

    public static void main(String[] args) {

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

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

        for (int i=0; i < inputArray.length; i++) {
            System.out.println(inputArray[i]);
        }
    }
}
```



```
Please enter several objects: Ford GM Chrysler Toyota Honda
Ford
GM
Chrysler
Toyota
Honda
```

The user entered each car brand separated by a space

The whole input is **`split`** and repackaged as an array

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.