

# Chapter 7: Objects, Methods and Properties

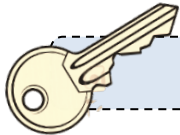
## Learning Outcomes:

- ✓ Instantiate objects from a class and call non-static methods from instantiated objects in a program
- ✓ Implement and use object properties
- ✓ Implement well-defined functions and methods in a consistent manner



## Prerequisite Knowledge:

- ✓ Complete Chapter 6
- ✓ Be able to identify the importance of the 'main' method in a program
- ✓ Be confident in writing algorithms, and using selection and iteration structures in a program



Objects

Public/Private

Void

Argument/Parameter

Global/Local

Keywords

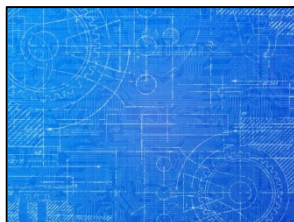
## 7.1 The Theory: Non-static Methods and Objects

All of the programs authored until now have been written entirely into a single method, the 'main' method. Although these 'single method programs' work as expected, they are limited. They are limited in design, and they are limited in functionality. To build programs that make use of good design, and to include greater flexibility, programs need to be broken down into individual classes and their respective methods. Initially, this approach may seem more complex, but in reality it is much easier, especially as programs become larger and naturally more complex.

### Classes and Objects

In Chapter 4, static and non-static methods (from the Java API) were discussed in detail. In summary, the chapter explained how static methods can be executed directly from a class, whereas non-static methods, before they can be used, must first be instantiated into an object. This particular chapter will focus on how a programmer can create (and use) non-static methods to simplify programs; but first, the relationship between classes and objects must be examined more closely. The best way in which to imagine the relationship between a class and an object is to envision a class as a 'blueprint'. This so-called blueprint can then be used to create many objects; in Java, these objects are created from the class by using the following syntax `[MyClassName myObjectName = new MyClassName();]`. The `[new]` keyword is used to create a reference to a new object from a specified class; it is that simple!

```
MyTransformerClass transformerObject1 = new MyTransformerClass();  
MyTransformerClass transformerObject2 = new MyTransformerClass();  
MyTransformerClass transformerObject3 = new MyTransformerClass();
```



MyTransformerClass



transformerObject1



transformerObject2



transformerObject3

## Objects and Non-Static Methods

```
public class MyTransformerClass{  
    public void transform(){  
        //the code  
    }  
    public void untransform (){  
        //the code  
    }  
}
```

Once a class has been instantiated into an object, its internal methods can be called (executing the code contained by the methods). For example, to execute the [transform()] or [untransform()] methods from the MyTransformerClass, an object must first be created. To call a method, simply refer to the method's name via the instantiated object. The syntax for this would look similar to this: [myObject.runMyMethod();].

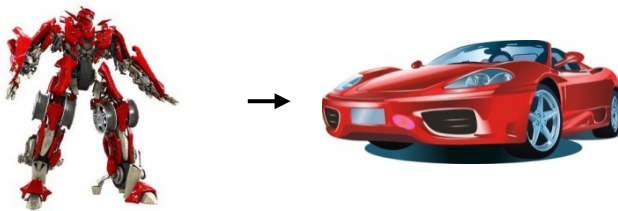
The code below demonstrates how an object can be instantiated, as well as how the object's respective methods can be called. To be specific, the code creates three objects from the MyTransformerClass class, before calling the [transform()] method that is available inside of each object. Remember that the [transform()] method now belongs to each instantiated object, not the MyTransformerClass class. For example, if the code contained by a method manipulates an object's state, only the state of that specific object (the object that called the method) would be affected, not all of the objects created from the class. Finally, note that the keyword [static] is used in the declaration of the 'main' method; 'static' means that the method belongs to the class and not to an instantiated object – in other words, the method can be used without the need to instantiate an object first. This characteristic makes the 'main' method an ideal place to begin any program!

```
public static void main(String[] args){  
    MyTransformerClass transformerObject1 = new MyTransformerClass();  
    transformerObject1.transform();  
  
    MyTransformerClass transformerObject2 = new MyTransformerClass();  
    transformerObject2.transform();  
  
    MyTransformerClass transformerObject3 = new MyTransformerClass();  
    transformerObject3.transform();  
}
```

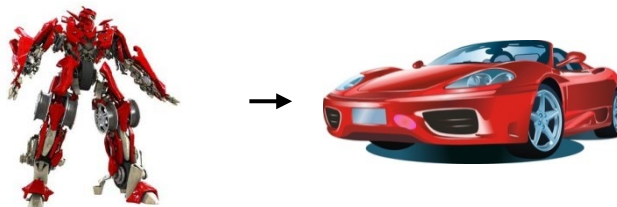
Object name

Class name

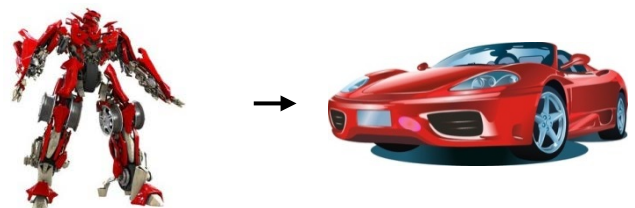
Method name



transformerObject1.transform();



transformerObject2.transform();




transformerObject3.transform();

## 7.2 Practical: Using Objects and their Methods

### Objects and Non-Static Methods in Practice

The example below demonstrates how multiple objects can be created from the same class; in this case, `objectX` and `objectY` are generated from the `MyFirstObject` class. Both objects are used to call the `[myFirstMethod()]` and `[mySecondMethod()]` methods. As the `main` method is `static` (meaning that it does not have to be instantiated first) and the first method to be called by any program, it makes for an ideal place to create objects from, as well as to call their respective methods. For reference purposes, note that static methods can be called directly in a program too by referring to the class name and then the method name, i.e. `[myClassName.myStaticMethodName();]`.

Try the code

```
public class MyFirstObject{  
    public static void main(String[] args){  
        MyFirstObject objectX = new MyFirstObject();  
        MyFirstObject objectY = new MyFirstObject();  
        objectX.myFirstMethod();  
        objectX.mySecondMethod();  
        objectY.myFirstMethod();  
        objectY.mySecondMethod();  
    }  
  
    public void myFirstMethod(){  
        System.out.println("I am method 1");  
    }  
  
    public void mySecondMethod(){  
        System.out.println("I am method 2");  
    }  
}
```

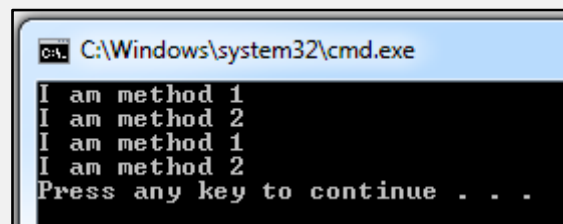
**MyFirstObject Class**  
  
+ static main method  
  
(Static means that the method belongs to the class and not an object)

**ObjectX and ObjectY**  
  
+ myFirstMethod method  
+ mySecondMethod method  
  
(Non-static means that the method must be called from an instantiated object)



### Step by step

1. Type out the above code
2. Save the program as `MyFirstObject.java`
3. Compile the program
4. Run the `MyFirstObject.class` file
5. Check the results of the program



```
C:\Windows\system32\cmd.exe  
I am method 1  
I am method 2  
I am method 1  
I am method 2  
Press any key to continue . . .
```

### Activity 7.1

Write a small utility program that will demonstrate the principles of methods and objects. The class should be named `MyTransformerClass`. The class should contain two methods, `[transform()]` and `[untransform()]`, both of which should contain a `[println()]` method that will print appropriate text to the command-line.

From the class, two objects should be created called `'bumbleBee'` and `'optimusPrime'`. Both objects, once instantiated, should be used to call the two methods contained in the class.

## 7.3 The Theory: Properties and Scope

### Object Properties and Class Properties

Every object has state; for example, a light bulb's state would either be on or off, or a dog's state might be 'Marley, Labrador, golden, 30 kg.' In programming, an object's state can be captured with object properties – variables that are declared outside of any method and thus belonging to an instantiated object. The example below demonstrates how object properties can be referred to in a program; note that the properties (myNum1, myNum2 and total) are available in both instantiated objects *x* and *y*. If a third object was instantiated, it too would have the same properties.

```
public class MyP{
    int myNum1 = 25;    //global variables
    int myNum2 = 15;    //object properties
    int total;
    static int oc = 0;  //object counter (class)

    public static void main(String[] args){
        MyP x = new MyP();
        MyP y = new MyP();
        x.add();
        x.showAnswer();
        y.add();
        y.showAnswer();
        System.out.println("Objects:" + MyP.oc);
    }

    public void add(){ //add method
        total = myNum1 + myNum2;
        MyP.oc = MyP.oc + 1;
    }

    public void showAnswer(){ //show method
        System.out.println("Sum: " + total);
    }
}
```

Earlier in the chapter, it was determined that methods declared with the keyword `[static]` will result in a method belonging to a class and not to an object. This is the same for properties too; for example, `[static int oc = 0]`, declared outside of any method, would result in a variable belonging to the class and not an instantiated object. Although there is no limit to the number of objects that can be created from a class, a class can only exist once in a program, thus a static class property can only exist once too. Static variables are very useful, especially when counting how many objects have been generated from a particular class!



### Programming Tips!

Variables that are declared inside of a method are referred to as 'local'. This is because the scope of the variable is contained within the method where it was declared; for example, if `[int x;]` was declared in method A, it cannot be seen or referred to by method B.

On the other hand, a variable that is declared outside of any method is referred to as 'global'; in difference a global variable can be referred to by any method within that class/object. For example, if `[int x;]` was declared outside of any method, it can be referred to by both method A and method B.

```
public void methodA(){
    int x;
    x = 5;
}
public void methodB(){
    x = x + 5; //error
    // x cannot be found
}
```

```
int x;
public void methodA(){
    x = 5; //x found
}
public void methodB(){
    x = x + 5; //x found
}
```

## 7.4 Practical: Objects, Methods and Properties

The program below creates two objects from the `ObjectsAndProperties` class. Both objects then call the `[getName()]` and `[sayName()]` methods; these methods ask the user to name the new object, before printing the new name to the command-line. The static method `[myStaticObjectCounter()]` is used to print the value of the static variable `oc` to the command-line; the `oc` variable keeps count of the number of objects created from the class. In Java, an object property can be referred to similarly to that of a method, by using the full stop, i.e. `[myObjectName.myPropertyName;]`. In the example below, this line of code `[objectX.objectName.equals("A")]` is used to determine whether an object property (in particular the `objectName` property) has the value 'A'; note that because the object property is of type string, it inherits all of the usable `String` class methods (such as `[equals()]`) too!

Try the code



```
import java.io.*;
public class ObjectsAndProperties{

    String objectName = null;    //object properties/global variables
    static int oc = 0;           //object counter (static variable)

    public static void main(String[] args){
        ObjectsAndProperties objectX = new ObjectsAndProperties();
        ObjectsAndProperties objectY = new ObjectsAndProperties();
        objectX.getName();
        objectY.getName();
        objectX.sayName();
        objectY.sayName();
        if(objectX.objectName.equals("A") || objectY.objectName.equals("A")){
            System.out.println("\n ***! An object has the name A !***!");
        }
        ObjectsAndProperties.myStaticObjectCounter(); //calls a static method
                                                       //from the class
    }

    public void getName(){ //asks the user to name the object
        BufferedReader x = new BufferedReader(new InputStreamReader(System.in));
        System.out.println("Enter an object name:");
        try{
            objectName = x.readLine();
        }catch(IOException e){
            System.out.println("Error: " + e);
        }
        ObjectsAndProperties.oc = ObjectsAndProperties.oc + 1; //adds 1 to oc
    }

    public void sayName(){ //prints the object name
        System.out.println("\n-----OBJ NAME-----\n " + objectName);
    }

    static void myStaticObjectCounter(){ //prints the number of objects
        System.out.println("\n\n-----OBJ COUNT-----");
        System.out.println("Objects:" + ObjectsAndProperties.oc);
    }

}
```



### Step by step

1. Type out the above code
2. Save the program as `ObjectsAndProperties.java`
3. Compile the program
4. Run the `ObjectsAndProperties.class` file
5. Check the results of the program

```
C:\Windows\system32\cmd.exe
Enter an object name:
Joseph
Enter an object name:
A
-----OBJ NAME-----
Joseph
-----OBJ NAME-----
A
***! An object has the name A !***!
-----OBJ COUNT-----
Objects:2
Press any key to continue . . .
```



## The Currency Converter

To complete the currency converter application, skills from previous chapters will need to be revisited too, including the `[readLine()]` method, parsing and selection. This chapter has begun to delve into the world of object-orientated programming (OOP); at first this may seem complex, so if necessary revise this chapter several times before continuing to the next!



### Activity 7.2a



**The Currency Converter** – write a program that can be used to convert global currencies into GBP, as well as being able to convert GBP into other global currencies. Create a `currencyConverter` object from a class that has two methods:

`[convertToGBP()]` and `[convertToForeign()]`.

Method main should contain an IF statement that enables a user to choose between the two methods. The user should be able to enter a decimal value and select a global currency from a list; depending on the method, the program should either convert a foreign currency amount to GBP or convert GBP to a foreign currency amount. The program should have the following object properties:

- final double USD = 1.6117; //US Dollar
- final double EUR = 1.2216; //Euro
- final double AUD = 1.5578; //Australian Dollar
- final double CAD = 1.5985; //Canadian Dollar
- final double SWF = 1.4681; //Swiss Franc
- final double HKD = 1.2507; //Hong Kong Dollar
- final double JAY = 1.3173; //Japanese Yen
- final double NZD = 1.9793; //New Zealand Dollar

'Final' marks the variable as a constant; in other words, its value is restricted from change. The convention for constant names is to use entirely upper-case letters.

```
C:\Windows\system32\cmd.exe
---->Choose an option:
<A> Convert to GBP <Great British Pounds>
<B> Convert to FCA <Foreign Currency Amount>
A
---->Enter a foreign currency amount:
3443.43
---->Please choose a currency:
<USD> United States Dollar
<EUR> Euro
<AUD> Australian Dollar
<CAD> Canadian Dollar
<SWF> Swiss Franc
<HKD> Hong Kong Dollar
<JAY> Japanese Yen
<NZD> New Zealand Dollar
USD
----->CONVERSION<-----
3443.43 USD to GBP = 2,136.52
Press any key to continue . . .
```

### Activity 7.2b



**The Currency Converter Extended** – amend the main method of the program so that the application continues to loop until the user chooses to exit.

Add a 3% commission charge for all conversions; the charge is 3% of the original starting amount.



### Programming Tips!

Formatting (as used in Chapter 6) can be used with numbers too; first, however, the class must be imported, which can be achieved by adding the following line of code to the top of a program `[import java.text.*;]`. The example below shows how a number can be formatted using the decimal formatter:

```
DecimalFormat myF = new
DecimalFormat("#,###.##");
String output = myF.format(10000.23);
System.out.println(output);
```

```
C:\Windows\system32\cmd.exe
10,000.23
Press any key to continue . . .
```



### Programming Tips!

To convert GBP to another currency, multiply GBP amount by the conversion rate; for example, GBP \* AUD. To convert from a foreign currency to GBP, divide the currency amount by the conversion rate; for example, FCA / AUD.

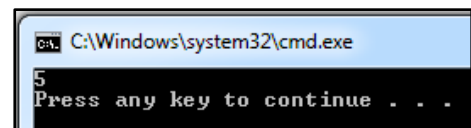
## 7.5 The Theory: Passing Values and Functions

### Arguments and Parameters

Smart programmers consider the required variables (for the method to run correctly) in advance. For example, a method that calculates the area of a circle would require the radius value first, before executing any code. In Java, 'parameters' can be specified that demand values before a method will execute – similar to asking for ingredients before following a recipe. When values are passed to a method that has parameters, these values are referred to as 'arguments'. This has been seen before; for example, the `[println()]` method requires a string argument, i.e. 'hello world'.

The example below holds a method called `[myAddition(int num1, int num2)]` that requires two integer parameters before the method can be called; note that the data type of a parameter must be specified, as well as the name. When the method is called `[x.myAddition(2,3)]`, two integer arguments are passed to the method; the method then adds them together, before printing them back to the command-line. Although the example below passes two integer values, these can just as easily be replaced with two integer variables! Importantly, arguments are passed in order of left to right; i.e. the first value will be placed into `num1`, and the second will be placed into `num2`!

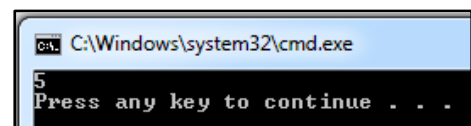
```
public class MyMaths{  
    public static void main(String[] args){  
        MyMaths x = new MyMaths();  
        x.myAddition(2,3); //passed arguments  
    }  
    public void myAddition(int num1, int num2){  
        System.out.println(num1 + num2);  
    }  
}
```



### Functions

A function is an alternative name given to a method that returns a value; functions have been utilised in earlier chapters. For example, `[readLine()]` is a function that returns a string value. If a method does not return a value, it is marked with the keyword `[void]`. However, if a method does return a value, the keyword `[void]` is replaced with the data type of the proposed return value instead. The returned value can then be stored into a variable for later use, in this case `[int myNum]`. To return a value the keyword `[return]` is used; a function can only return one value, although it may return an array (see Chapter 8).


```
public class MyMaths{  
    public static void main(String[] args){  
        MyMaths x = new MyMaths();  
        int myNum = x.myAddition(2,3);  
        System.out.println(myNum);  
    }  
    public int myAddition(int num1, int num2){  
        int sum = num1 + num2;  
        return sum; //return integer answer  
    }  
}
```



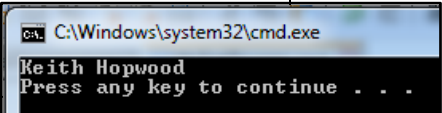
## 7.6 Practical: Writing Good Methods and Functions

### Using Functions in a Program

The function below concatenates a first name value and a surname value together, before returning a full name value. For the function to work, it requires two parameters: `firstName` and `surname`. Without the parameters the function will not execute. Java is known as 'pass by copy', which means a copy of a variable's value is passed to a method/function, not the variable itself. In this case, a copy of the values stored in the variables `fName` and `lName` is given to the function, and they are given in order of left to right.


Try the code 

```
public class MyNameApp{  
    public static void main(String[] args){  
        MyNameApp nameObject = new MyNameApp(); //new object  
        String fName = "Keith";  
        String lName = "Hopwood";  
        String name = nameObject.myName(fName, lName);  
        System.out.println(name);  
    }  
  
    public String myName(String firstName, String surname){  
        String fullName = firstName + " " + surname;  
        return fullName;  
    }  
}
```

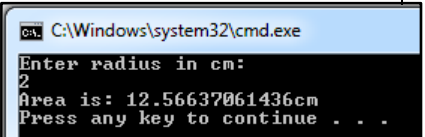


```
C:\Windows\system32\cmd.exe  
Keith Hopwood  
Press any key to continue . . .
```

Below is another example of a function; this time the function calculates the area of circle, based on a radius value that is entered by the user, before returning the calculated area. The returned value is stored in the `myArea` variable, before being printed to the command-line.

Try the code 

```
import java.io.*;  
public class MyAreaFunctions{  
    public static void main(String[] args){  
        String input = null;  
        BufferedReader x = new BufferedReader(new InputStreamReader(System.in));  
        MyAreaFunctions myAreaObject = new MyAreaFunctions(); //new object  
        System.out.println("Enter radius in cm:");  
        try{  
            input = x.readLine();  
        }catch(IOException e){  
            System.out.println("Error: " + e);  
        }  
        double userValue = Double.parseDouble(input);  
        double myArea = myAreaObject.myCircleArea(userValue); //call function  
        System.out.println("Area is: " + myArea + "cm");  
    }  
  
    public double myCircleArea(double radius){  
        double area = 3.14159265359 * radius * radius;  
        return area;  
    }  
}
```



```
C:\Windows\system32\cmd.exe  
Enter radius in cm:  
2  
Area is: 12.56637061436cm  
Press any key to continue . . .
```

### Activity 7.3

Write a program that calculates the area of a triangle, square, rectangle and trapezium, based on dimensions given by a user. Each shape should have its own function that returns the calculated area. The user should be able to choose which shape to calculate the area for, from a given list.



## The Wage Calculator

The wage calculator application features complex requirements that may be initially difficult to code; thus it is advisable to take some time to write some pseudocode prior to attempting the task. Pseudocode is a design principle, and it is remarkably easy to follow; simply write out the steps in English, no coding talk, just functionality and calculations! A little forethought is likely to reduce the overall time spent coding. To help with this task, one of the functions has already been written in advance. There is also some pre-written pseudocode to help with the calculating of overtime.



```
public class MyWageApp{
    public static void main(String[] args){
        MyWageApp myWageCalc = new MyWageApp();
        System.out.println("-->Welcome to the Wage App \n");
        //finish code
    }
    public double nationalInsurance(double remainingGross){
        double ni = remainingGross * 0.11;
        return ni; // NI function - returns NI Amount
    }
}
```



### Programming Tips!

Calculating the overtime might be difficult; the following pseudocode (English-like algorithm code) may help:

```
if (hours <= 40)
    gross = hours * hourlyRate
else if (hours <= 50)
    rate1 = 40 * hourlyRate
    rate2 = (hours-40) * (hourlyRate * 1.5)
    gross = rate1 + rate2
else if (hours > 50)
    rate1 = 40 * hourlyRate
    rate2 = 10 * (hourlyRate * 1.5)
    rate3 = (hours-50) * (hourlyRate * 2)
    gross = rate1 + rate2 + rate3
end if
```



### Programming Tips!

Although a function can only return one value, multiple return statements can be used in an IF statement to have different return possibilities. See below:

```
if(x > 10 ){
    return true;
} else if(x > 0) {
    return false;
}
```

### Activity 7.4



**The Wage Calculator** – write a program that can be used to calculate the weekly wage (including overtime) earned by employees. The program should ask the user to input the number of hours worked, as well as to select their hourly rate; there are three hourly rates to choose from: they are administrator (£7.43 per hour), programmer (£16.98 per hour) and team leader (£21.10 per hour).

Overtime is calculated based on the following information: any employee who accumulates more than 40 hours per week earns an overtime bonus. The bonus is 'time and a half' for greater than 40 to 50 hours and 'double time' for any hours greater than 50.

The program needs to summarise the gross and net income, as well as calculate and summarise the tax (25% of gross) and national insurance (11% of remaining gross after income tax) deductions.

The program should have the following methods/functions and parameters:

- hourlyRateSelection()
- hoursWorked()
- calculateGross(double hours, double hourlyRate)
- incomeTax(double gross)
- nationalInsurance(double remainingGross)
- summary(double gross, double net, double incomeTax, double nationalInsurance, double hours, double hourlyRate)

## 7.7 Practical: Public Vs Private

### Using Public and Private

The terms 'public' and 'private' refer to the visibility of a class, method or variable. Visibility is important as it determines how a class, or parts of a class, may be used. For example, the [private] keyword can be used to restrict the use of a method or variable inside of a class to only that class; this behaviour can help protect the integrity of a class' inner workings. In contrast, the [public] keyword can be used to make the visibility of something completely open; in other words, accessible to all, including other classes. The example below demonstrates how visibility can be used between classes; note that anything set to 'private' cannot be referenced to in the `TestPilot` class, only items set to 'public' can be used.



```
public class MyValidation{
    private int myVar;
    public void myMethod{
        //some code
    }
    private void myMethod2{
        //some code
    }
}
```

```
public class TestPilot {
    public static void main(String[] args){
        MyValidation obj = new MyValidation();
        obj.myVar = 4;        //error
        obj.myMethod();       //works great!
        obj.myMethod2();      //error
    }
}
```

Visibility becomes more important as applications begin to grow in size, and naturally become more ambiguous. Simply considering whether class methods/properties are accessible to other classes takes little deliberation, but in the long run may help to reduce unforeseen problems.

### Activity 7.4



Write out the complete code for both classes featured on the next page (the `TestPilot` and `MyValidation` class) and compile them; remember to keep the file names the same as the class names. Run the program and check the outputted results. Next, remove the letter 'g' from the x variable, change the value of variable y to 6 and change the value of variable z to 'hi'. Run the program again to compare the differences. Finally, change all the functions in the `MyValidation` class to private, recompile the program and run the program again; observe the error that is displayed when trying to access something which is set to private!

## Using Multiple Classes

All of the activities so far have been written into a single class, but in reality large programs span across many classes. The example below uses two classes; the `TestPilot` class only holds the `main` method, which contains the code to instantiate and manipulate an object generated from the `MyValidation` class. The `MyValidation` class holds many methods that can be used in a program to validate input.

Try the code

```
public class MyValidation{  
    public boolean isNumeric(String str){ //does str contain only numbers?  
        try{  
            double d = Double.parseDouble(str);  
        }  
        catch(NumberFormatException nfe){ //catches number format errors  
            return false; //on error return false  
        }  
        return true; //otherwise return true  
    }  
  
    public boolean isBetween1and10(int num){ //is num between 1 and 10?  
        if(num >=1 && num <=10){  
            return true; //if yes return true  
        }  
        return false; //otherwise return false  
    }  
  
    public boolean isEmpty(String str){ //is str empty?  
        str = str.trim(); //removes any character spaces  
        if(str.equals("")){  
            return true; //if empty return true  
        }  
        return false; //otherwise return false  
    }  
}
```



```
C:\Windows\system32\cmd.exe  
false  
false  
true  
Press any key to continue . . .
```

```
public class TestPilot{  
    public static void main(String[] args){  
        MyValidation testRun = new MyValidation();  
        String x = "34534g";  
        Boolean result1 = testRun.isNumeric(x);  
        System.out.println(result1);  
  
        int y = 11;  
        Boolean result2 = testRun.isBetween1and10(y);  
        System.out.println(result2);  
  
        String z = " ";  
        Boolean result3 = testRun.isEmpty(z);  
        System.out.println(result3);  
    }  
}
```



### Programming Tips!

Once the `MyValidation` class is up and running, it can be used on any future projects that require some validation testing of a user's input. For example, use the `isNumeric` function to check whether a string is entirely made up of numbers, before parsing it; this will prevent a possible run-time error! Genius!