

## CMIS 141 Hands-on Lab Week 2

### Overview

This document provides a series of exercises to assist the student becoming comfortable coding in Java. The topics covered in this lab include reading from standard input/output, applying Java style guide best practices, developing selection statements including if/else and switch statements, and analyzing Java code to identify and correct various forms of programming errors.

It is assumed the JDK 8 (or higher) programming environment is properly installed and the associated readings for this week have been completed.

### Submission requirements

Hands-on labs are designed for you to complete each week as a form of self-assessment. You do not need to submit your lab work for grading. However; you should post questions in the weekly questions area if you have any trouble or questions related to completing the lab. These labs will help prepare you for the graded assignments, therefore; it is highly recommended you successfully complete these exercises.

### Objectives

The following objectives will be covered by successfully completing each exercise:

1. Read primitive data types from standard Input/output using the `BufferedReader` and `Scanner` classes.
2. Apply Java style guide best practices while developing Java code.
3. Design, develop and test if and if/else statements in Java.
4. Design, develop and test switch statements in Java
5. Analyze Java code to identify and correct common program errors.

### Using Java Classes and Methods

We haven't provided too many details about Java classes and methods within those classes. However; you have been using classes and methods since the beginning of this course. For example, `main()` is a method in most of the classes we have created to date. Methods can be thought of as small chunks of code providing some specific functionality.

The good news is Java has thousands of existing classes that contain some extremely useful methods. The `BufferedReader` and `Scanner` classes are two examples of this existing functionality. Instead of having to create our own class that will read data from standard input we can use the existing ones. In most cases, we should use the existing classes as opposed to creating our own. As you continue to study Java, you will learn to how to find these classes and take advantage of their utility.

The following are some critical concepts upon how to use Java classes:

1. **Constructing an instance of a class.** Most Java classes are constructed using this syntax:

```
Classname ReferenceName = new Classname(Arguments);
```

Arguments are parameters that are used to better define the constructed class. Some classes don't require any arguments, others require multiple arguments.

For example to construct an instance of a Scanner class with reference scannerIn that uses System.in as a constructor argument, we use this syntax:

```
Scanner scannerIn = new Scanner(System.in);
```

To construct an instance of a InputStreamReader class with reference isReader that uses System.in as a constructor argument, we use this syntax:

```
InputStreamReader isReader = new InputStreamReader(System.in);
```

You should focus on the pattern of constructing instances of classes as you will use this approach to construct most of your classes.

2. **Using methods of a class.** Methods can take input arguments and often return values. Most Java class methods are invoked using this syntax:

```
ReferenceName.Methodname(Arguments);
```

Arguments are parameters that are used to better define the methods. Some methods don't require any arguments, other require multiple arguments. Also, some methods do not return any values, others return specific types.

The following, shows a method call of a Scanner reference name:

```
int favInt = scannerIn.nextInt();
```

In this example, an int is returned from the nextInt() method call. No arguments are used for the nextInt() method.

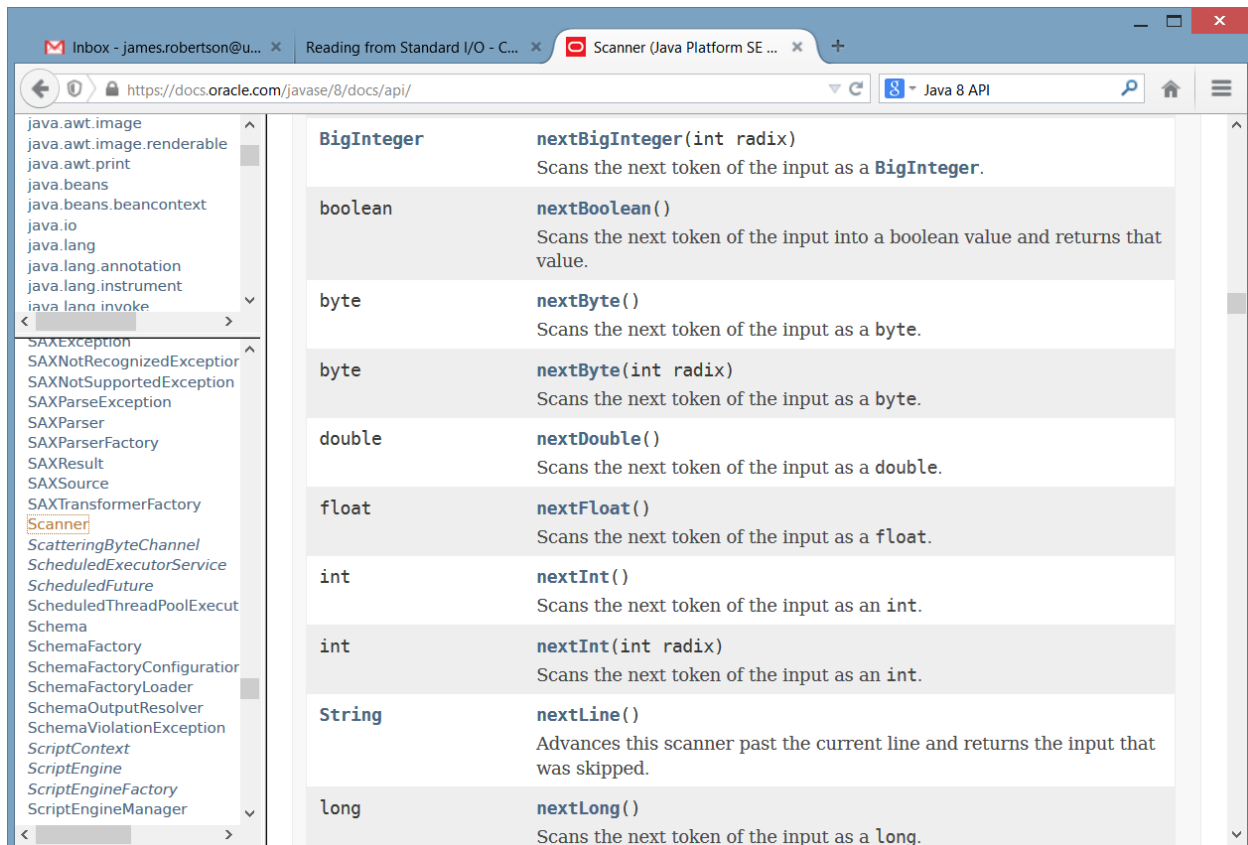
The following is a more complicated example but the pattern of use is the same. In this case, the argument is stdin.readLine(), the method is parseInt(), the reference classname is Integer and the value returned is an int named favInt.

```
int favInt = Integer.parseInt(stdin.readLine());
```

3. **Finding classes and methods.** The Java Application Programming Interface (API) is the place to locate and find classes and their associated methods. The API is massive and will take years to master. However; you should know how to find the API and some classes and associated methods. The current URL for the Java 8 API is found here:

<https://docs.oracle.com/javase/8/docs/api/>

The API is divided into packages and classes. Packages are groups of related classes. If you click on the Scanner class, the constructors and methods are revealed. Notice, the methods provide details on the return type and input argument, if any is needed.



You should search the Java API for classes and methods you are using to become comfortable with other options that are available. The document can be overwhelming, so focus on the few classes and methods we are using at first. In time, you will be quite comfortable at traversing and using the many classes and associated methods in the Java API.

### Exercise 1 – Read primitive data types from standard Input/output using the **BufferedReader** and **Scanner** classes

The process of interacting with a computer program often involves reading information entered by a user through a keyboard. This exercise provides two different approaches for using Java to accept data entered from a keyboard. Entering data into a program from a keyboard is referred to as using Standard Input.

The following set of steps provides code for entering data using both the **BufferedReader** and **Scanner** classes. The **BufferedReader** class is part of the Java IO package. The **Scanner** class is part of the Java Util package.

- a. Open your favorite text editor.



b. Type (or copy and paste) the following Java code into your text editor

```
// Import each required Java class
import java.io.IOException;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.util.Scanner;

/**
 * Standard I/O Demo
 */
public class StandardIODemo
{
    public static void main(String[] args) throws IOException
    {
        // Variables to hold values
        int favInt = 0;
        double favDouble = 0.0;
        int secondfavInt = 0;
        double secondfavDouble = 0.0;
        boolean myBoolean = false;
        short myShort = 0;

        // Define a InputStreamReader based on Standard Input (System.in)
        InputStreamReader isReader = new InputStreamReader(System.in);
        // Send the InputStreamReader to a BufferedReader
        BufferedReader stdin = new BufferedReader(isReader);

        // Prompt the user to enter an int
        System.out.println("Enter your favorite integer:");
        // The readLine() method reads everything entered
        // However the Integer.parseInt() method converts
        // the value to an int
        favInt = Integer.parseInt(stdin.readLine());
    }
}
```

```

// Prompt the user to enter an double
System.out.println("Enter your favorite double number:");
// Read the line and convert to a double
favDouble = Double.parseDouble(stdin.readLine());

// Print the results to verify your data
System.out.println("Your favorite int is: " + favInt);
System.out.println("Your favorite double is: " + favDouble);

// Use the Scanner class to perform the same functionality
Scanner scannerIn = new Scanner(System.in);

// Prompt the user to enter another int
System.out.println("Enter your second favorite integer:");
// the nextInt() method scans the next int value
secondfavInt = scannerIn.nextInt();

// Prompt the user to enter another double
System.out.println("Enter your second favorite double number:");
// the nextDouble() method scans the next int value
secondfavDouble = scannerIn.nextDouble();

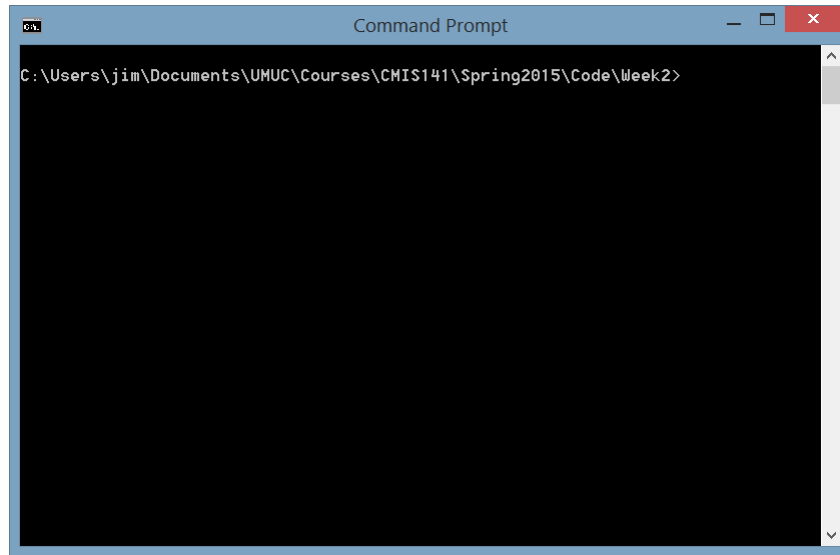
// Prompt the user to enter a boolean
System.out.println("Enter your favorite boolean value:");
// the nextBoolean() method scans the next boolean value
myBoolean = scannerIn.nextBoolean();

// Prompt the user to enter a short
System.out.println("Enter your favorite short value:");
// the nextShort() method scans the next short value
myShort = scannerIn.nextShort();

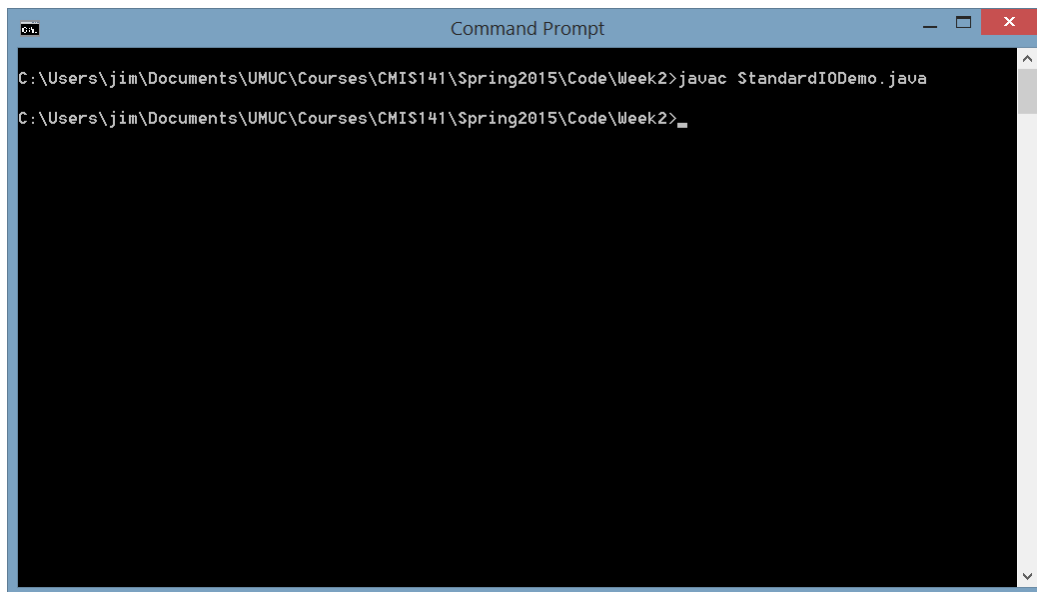
// Print the results to verify your data
System.out.println("Your second favorite int is: " + secondfavInt);
System.out.println("Your second favorite double is: " +
secondfavDouble);
System.out.println("Your favorite boolean is: " + myBoolean);
System.out.println("Your favorite short is: " + myShort);
}
}

```

- c. Save the file as "StandardIODemo.java" in a location of your choice.
- d. Use the cd command to change to the folder (directory) location of your StandardIODemo.java file.



- e. To compile the StandardIODemo.java file, type `javac StandardIODemo.java` at the command prompt.

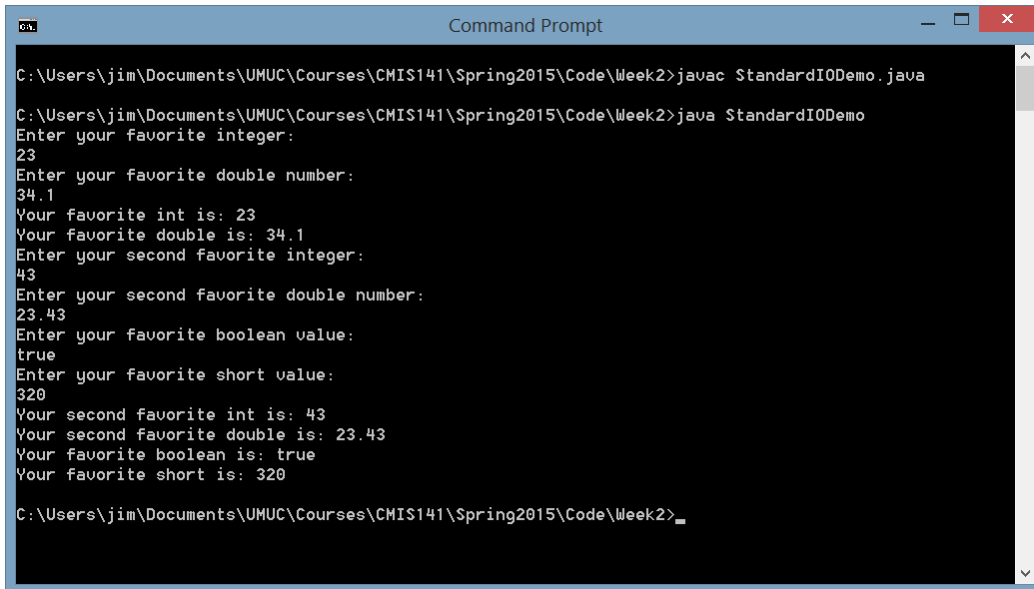


- f. To run the StandardIODemo.class file, type `java StandardIODemo` at the command prompt. When prompted, enter six values including int, double, int, double, boolean and short. For example, the following values could be entered:

23  
34.1  
43  
23.43

true  
320

The output will look similar to this:



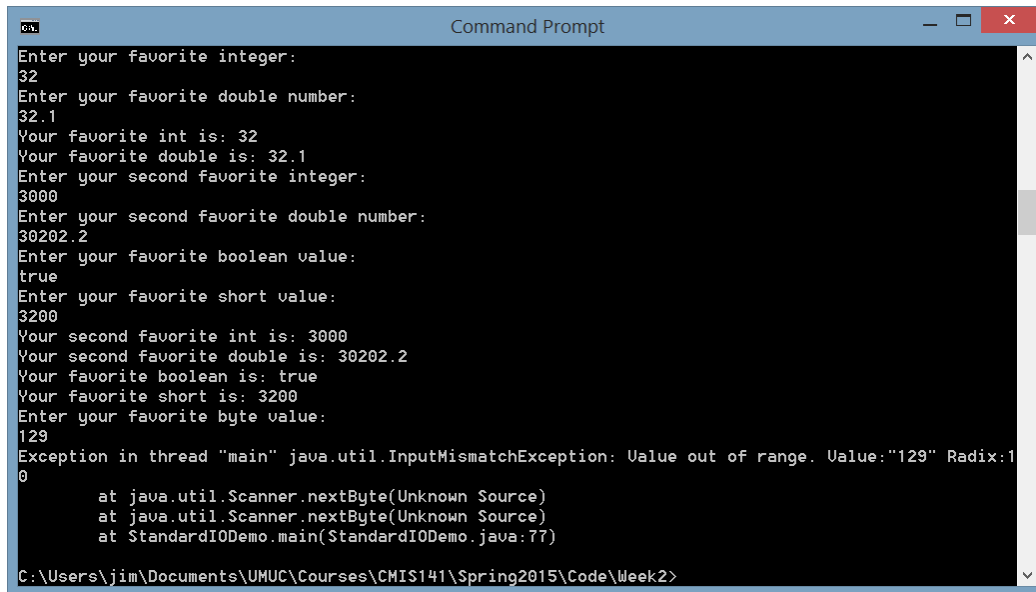
```
C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code\Week2>javac StandardIODemo.java
C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code\Week2>java StandardIODemo
Enter your favorite integer:
23
Enter your favorite double number:
34.1
Your favorite int is: 23
Your favorite double is: 34.1
Enter your second favorite integer:
43
Enter your second favorite double number:
23.43
Enter your favorite boolean value:
true
Enter your favorite short value:
320
Your second favorite int is: 43
Your second favorite double is: 23.43
Your favorite boolean is: true
Your favorite short is: 320
C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code\Week2>
```

- g. Experiment by adding the following code that prompts the user to input a byte named myByte and then prints the value entered by the user.

```
// Prompt the user to enter a byte
System.out.println("Enter your favorite byte value:");
// the nextByte() method scans the next byte value
byte myByte = scannerIn.nextByte();
System.out.println("Your favorite byte is: " + myByte);
```

As you analyze and experiment with the code, note the following:

1. scannerIn.nextByte(), scannerIn.nextInt(), scannerIn.nextBoolean() and other methods listed in the Scanner class API are used to store the specific type of data. For example, nextInt() looks for an int, nextBoolean() looks for a boolean and nextByte() looks for a byte.
2. If you attempt to input a value that is not in the range of the data type, you will receive a Runtime error. For example, since Bytes range in value from -128 to 127, if you attempt to enter 129 an error will result.



```
Command Prompt
Enter your favorite integer:
32
Enter your favorite double number:
32.1
Your favorite int is: 32
Your favorite double is: 32.1
Enter your second favorite integer:
3000
Enter your second favorite double number:
30202.2
Enter your favorite boolean value:
true
Enter your favorite short value:
3200
Your second favorite int is: 3000
Your second favorite double is: 30202.2
Your favorite boolean is: true
Your favorite short is: 3200
Enter your favorite byte value:
129
Exception in thread "main" java.util.InputMismatchException: Value out of range. Value:"129" Radix:10
    at java.util.Scanner.nextByte(Unknown Source)
    at java.util.Scanner.nextByte(Unknown Source)
    at StandardIODemo.main(StandardIODemo.java:77)
C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code\Week2>
```

Now it is your turn. Try the following exercise:

Create a Java class named `MyStandardIO` using your favorite text editor. Be sure you name the file `"MyStandardIO.java"`. Add code to the file in the `main()` method that will provide functionality to prompt the user to enter a user's age, address latitude and longitude, the approximate number of stars in our galaxy, the maximum value Java can store as a short, and a true or false response to the question "Are you stressed out?". The data types declared for the variables used should be byte, float, float, long, short and boolean, respectively for each of the user responses. After all data has been captured from Standard input, use the `System.out.println()` method to echo the variable name and value to Standard output.

## Exercise 2 – Apply Java style guide best practices while developing Java code

After reading the Java Style Guide documents in the course content area for this week, you should be comfortable with the different type of in-line code documentation including header, class, method, block, line, and end-of-brace comments. You should also be comfortable with applying the guidelines for naming variables, classes and methods, using import statements, and how to use braces.

As a beginning programmer in Java, some of the style guide details found in the google style document may not be obvious since many of the materials have not been covered yet. However; the key guidelines to follow for this course include:

1. Use header comments in all Java file to include filename, author, date and brief purpose of the program.
2. Use in-line comments to describe major functionality of the code.
3. Use meaningful variable names and user prompts.

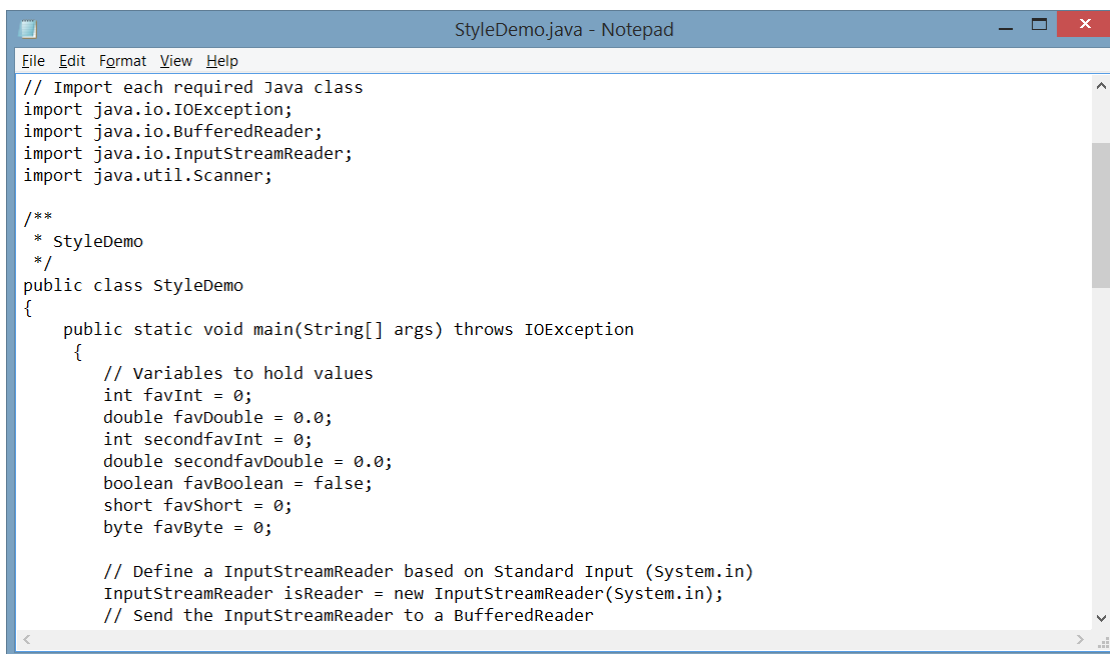


4. Class names are written in UpperCamelCase
5. Variable names are written in lowerCamelCase
6. Constant names are in written in All Capitals (e.g. MYCONSTANT)
7. Braces should follow K&R style where there is no line break before the opening brace, there is a line break after the opening brace and there is line break before the closing brace. Here is an example:

```
public class MyStandardIO {  
    public static void main(String[] args) {  
    } // End of main  
} // End class
```

For this exercise, we will open up a demo from a previous exercise and apply the proper formatting, documentation and recommended style.

- a. Open your favorite text editor and load the StandardIODemo.java file
- b. Change the name of the file and class to StyleDemo.java



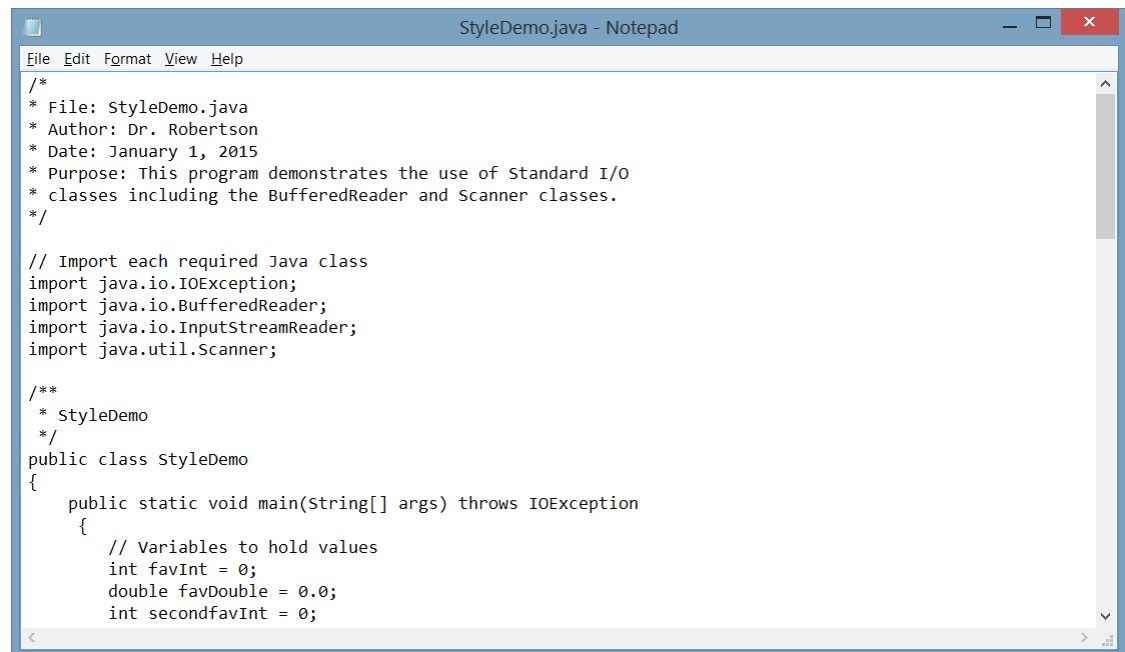
```
StyleDemo.java - Notepad  
File Edit Format View Help  
// Import each required Java class  
import java.io.IOException;  
import java.io.BufferedReader;  
import java.io.InputStreamReader;  
import java.util.Scanner;  
  
/**  
 * StyleDemo  
 */  
public class StyleDemo  
{  
    public static void main(String[] args) throws IOException  
    {  
        // Variables to hold values  
        int favInt = 0;  
        double favDouble = 0.0;  
        int secondfavInt = 0;  
        double secondfavDouble = 0.0;  
        boolean favBoolean = false;  
        short favShort = 0;  
        byte favByte = 0;  
  
        // Define a InputStreamReader based on Standard Input (System.in)  
        InputStreamReader isReader = new InputStreamReader(System.in);  
        // Send the InputStreamReader to a BufferedReader
```

- c. We can use the 7 key guidelines above as a checklist to apply the desired style to the code. First we make sure we have header comments to include filename, author, date and brief purpose of the program. Header documentation is added to very top of the file

The following header comments will be added to the file:

```
/*
 * File: StyleDemo.java
 * Author: Dr. Robertson
 * Date: January 1, 2015
 * Purpose: This program demonstrates the use of Standard I/O
 * classes including the BufferedReader and Scanner classes.
 */
```

The top of the file will now look similar to this:



```
StyleDemo.java - Notepad
File Edit Format View Help
/*
 * File: StyleDemo.java
 * Author: Dr. Robertson
 * Date: January 1, 2015
 * Purpose: This program demonstrates the use of Standard I/O
 * classes including the BufferedReader and Scanner classes.
 */

// Import each required Java class
import java.io.IOException;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.util.Scanner;

/**
 * StyleDemo
 */
public class StyleDemo
{
    public static void main(String[] args) throws IOException
    {
        // Variables to hold values
        int favInt = 0;
        double favDouble = 0.0;
        int secondfavInt = 0;
    }
}
```

- d. Next, we will include in-line comments describing major functionality of the code. Although, it may be considered somewhat subjective as to what major functionality is, **all Java code should have some in-line comments**. When adding comments, you should describe transitions and other parts of code that may be difficult to understand. Fortunately, we have been adding in-line comments all along as we developed code. Notice, we added comments for variable definitions, user prompts, output statements and lines of code where some explanation might be in order. For example, explaining how the `readline()` and `parseInt()` methods may make sense for an introductory programming course. Adding and refining in-line comments may look similar to the following:

```
// Variables to hold values
int favInt = 0;
double favDouble = 0.0;
int secondfavInt = 0;
```

```

double secondfavDouble = 0.0;
boolean myBoolean = false;
short myShort = 0;

    // Define a InputStreamReader based on Standard Input
    InputStreamReader isReader = new InputStreamReader(System.in);
// Send the InputStreamReader to a BufferedReader
    BufferedReader stdin = new BufferedReader(isReader);

// Prompt the user to enter a byte
System.out.println("Enter your favorite byte value:");
// the nextByte() method scans the next byte value
    byte myByte = scannerIn.nextByte();
    System.out.println("Your favorite byte is: " + myByte);

// ...

```

- e. Next, make sure meaningful variable names and prompts are used. Meaningful can be described as “does the variable name seem appropriate for what purpose the variable is serving?” As we look at the variable names, they are used to enter a favorite number or value. So names such as favoriteInt or favInt do seem appropriate. So the following variable definitions make sense at this point:

```

// Variables to hold values
int favInt = 0;
double favDouble = 0.0;
int secondfavInt = 0;
double secondfavDouble = 0.0;
boolean favBoolean = false;
short favShort = 0;
byte favByte = 0;

```

Similarly, prompts should provide details specifying what the user should enter. So prompts such as “Enter your favorite integer:” and “Enter your favorite short value:” make sense.

Modify the code to make sure all defined variables match the assignments in the code. For example:

```

favShort = scannerIn.nextShort();
favByte = scannerIn.nextByte();

```

- f. We will now check our variable, class and constant names to be sure they are written in the proper case. Variables names should be written in lowerCamelCase, class names should be written in UpperCamelCase and Constants should be written in ALLCAPS.

Our class name is StyleDemo which is in UpperCamelCase. Our variable names are mostly in lowerCamelCase but we have to modify secondFavInt and secondFavDouble changing them to secondFavInt and secondFavDouble to be compliant:

```
int favInt = 0;
double favDouble = 0.0;
int secondFavInt = 0;
double secondFavDouble = 0.0;
boolean favBoolean = false;
short favShort = 0;
byte favByte = 0;
```

After making the lowerCamelCase changes to the definitions be sure to change the assignment and print statements:

```
secondFavInt = scannerIn.nextInt();
secondFavDouble = scannerIn.nextDouble();
System.out.println("Your second favorite int is: " +
secondFavInt);
System.out.println("Your second favorite double is: " +
secondFavDouble);
```

We don't have any constants in this example, but if we did, we would name using all capitals. (e.g. final double PI = 3.1416;)

- g. Finally, we check that are braces follow K&R style where there is no line break before the opening brace, there is a line break after the opening brace and there is line break before the closing brace. Since the braces after the class and main methods have a return, we will need to change the code to meet this requirement. So instead of this code:

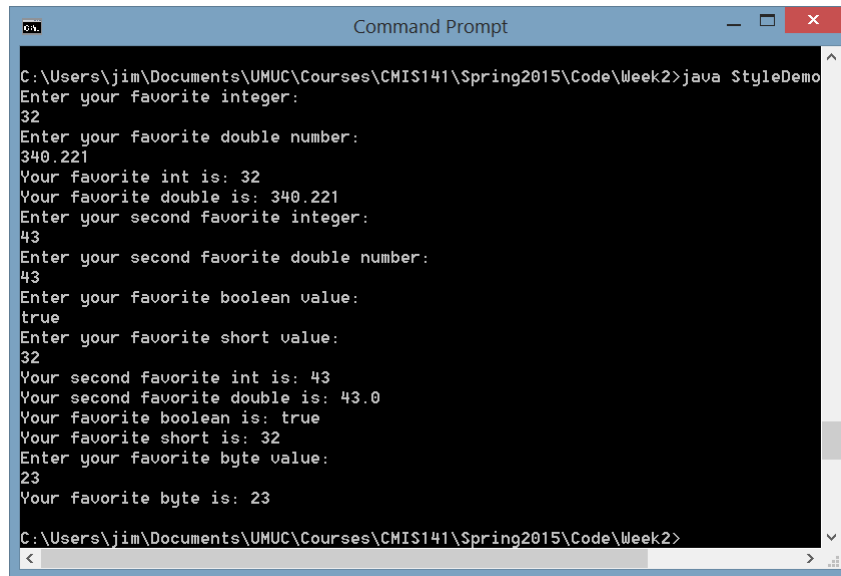
```
public class StyleDemo
{
    public static void main(String[] args) throws IOException
    {
```

We will have this code:

```
public class StyleDemo {
    public static void main(String[] args) throws IOException {
```

Notice the starting brace was moved to the end of the line as opposed to a new line.

- h. After all of the style changes have been made, be sure you compile and test your code, removing any possible errors introduced by the changes.



```
C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code\Week2>java StyleDemo
Enter your favorite integer:
32
Enter your favorite double number:
340.221
Your favorite int is: 32
Your favorite double is: 340.221
Enter your second favorite integer:
43
Enter your second favorite double number:
43
Enter your favorite boolean value:
true
Enter your favorite short value:
32
Your second favorite int is: 43
Your second favorite double is: 43.0
Your favorite boolean is: true
Your favorite short is: 32
Enter your favorite byte value:
23
Your favorite byte is: 23
C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code\Week2>
```

Now it is your turn. Try the following exercise:

Load up the OperatorsDemo.java file from week 1, rename the file MyStyleDemo.java and apply the recommend style guide enhancements. Be sure your modified code compiles and runs without issue.

### Exercise 3 - Design, develop and test if and if/else statements in Java.

Selection statements are needed in any programming language to provide alternative paths in processing. Selection statements allow us to test a condition and make decisions based on that test condition. For example, anyone who is greater than or equal to the age of 18 may have the right to vote in our elections. A simple selection statement for this scenario might look like this:

```
int age = 19;
boolean canVote = false;
if (age >= 18 ) {
    canVote = true;
}
```

Selection statements can become quite complex depending upon the scenario or context of the application. For example, if we aren't concerned with leap years or the rules associated with how to determine a leap year, the following selection statements could be used to determine the number of days in the month based on the integer value of the month.

```
// Variables to hold values
int month = 1;
int numDays = 30;

// Selection statement to determine number
```

```

// of days in month
if (month == 1) {
    numDays = 31;
}
else if (month == 2) {
    numDays = 28;
}
else if (month == 3) {
    numDays = 31;
}
else if (month == 4) {
    numDays = 30;
}
else if (month == 5){
    numDays = 31;
}
else if (month == 6) {
    numDays = 30;
}
else if (month == 7) {
    numDays = 31;
}
else if (month == 8) {
    numDays = 31;
}
else if (month == 9) {
    numDays = 30;
}
else if (month == 10) {
    numDays = 31;
}
else if (month == 11) {
    numDays = 30;
}
else if (month == 12) {
    numDays = 31;
}
}

```

Although the above code will work, it can be simplified using the || operator to group the similar months:

```

// Variables to hold values
int month = 3;
int numDays = 0;

// Selection statement to determine number
// of days in month
if ((month == 1) ||
    (month == 3) ||
    (month == 5) ||

```

```

        (month == 7) ||
        (month == 8) ||
        (month == 10) ||
        (month == 12) ) {
            numDays = 31;
        }
    else if (month == 2) {
        numDays = 28;
    }
    else if ((month == 4) ||
        (month == 6) ||
        (month == 9) ||
        (month == 11) ) {
        numDays = 30;
    }
    else {
        System.out.println("Month is not a value between 1 and
12");
        // Exit the application for invalid Months
        System.exit(0);
    }
}

```

In this code, there is an error check in the final else statement that provides an output and call the `System.exit()` method. This call exits the program if the number entered is not a valid integer between 1 and 12. Although a bit abrupt, this is one approach to exit the application if data or other parameters are not correct. We will learn more user-friendly approaches to resolve data issues later in the semester.

For this exercise, we will write a simple program that prompts a user for the month of the year and then responds by printing the number of days in that month (assuming a non-leap year). As programs become more complex, notice you can reuse code from other examples to solve problems. For example, the `Scanner` class was used to gather integer input from standard input in the `StandardIODemo.java` file. This same approach can be applied to help solve this problem.

- a. Open your favorite text editor and type or copy and paste the following code.

```

/*
 * File: SelectionIfDemo.java
 * Author: Your name here
 * Date: January 1, 2015
 * Purpose: This program demonstrates the use of if/else selection
 * statements.
 */

public class SelectionIfDemo {
    public static void main(String[] args) {

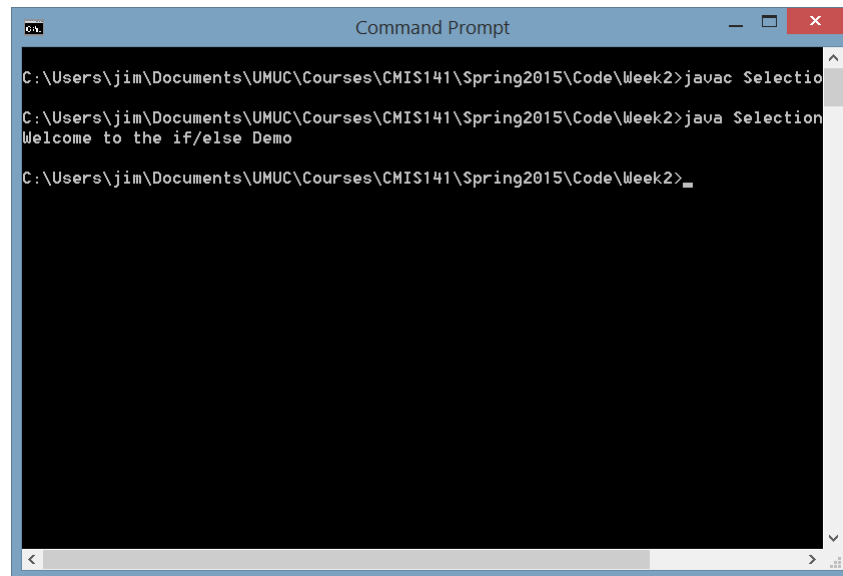
        // Display a Welcome note
        System.out.println("Welcome to the if/else Demo");
    }
}

```

```
}  
}
```

Notice, we are applying the Java style guide to all of our code now. Applying the style guide while you code is easier than applying the guidelines later.

- b. Save the file as SelectionIfDemo.java. Make sure it compiles and runs without issue.



```
Command Prompt  
C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code\Week2>javac SelectionIfDemo.java  
C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code\Week2>java SelectionIfDemo  
Welcome to the if/else Demo  
C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code\Week2>
```

- c. Add code to allow the input of an integer month. This will include defining the variables and importing the Scanner class. And displaying the month entered to standard output.

```
/*  
 * File: SelectionIfDemo.java  
 * Author: Dr. Robertson  
 * Date: January 1, 2015  
 * Purpose: This program demonstrates the use of if/else selection  
 * statements.  
 */  
  
// Import statements  
import java.util.Scanner;  
  
public class SelectionIfDemo {  
    public static void main(String[] args) {  
  
        // Display a Welcome note  
        System.out.println("Welcome to the if/else Demo");  
  
        // Variables to hold values  
        int month = 0;
```



```

        // Use the Scanner class to input data
        Scanner scannerIn = new Scanner(System.in);

        System.out.println("Enter the Month (1-12):");
        // the nextInt() method scans the next int value
        month = scannerIn.nextInt();

        // Verify the month was entered
        System.out.println("The following month was entered " + month);
    }
}

```

Compiling and running results in the following output:

```

C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code\Week2>javac SelectionIfDemo.java
C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code\Week2>java SelectionIfDemo
Welcome to the if/else Demo
Enter the Month (1-12):
4
The following month was entered 4
C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code\Week2>

```

- d. Next, add the selection statement if/else logic. This is where the numDays variable is defined and all of the logic to determine the number of days based on the input month is added.

This results in the following complete code:

```

/*
 * File: SelectionIfDemo.java
 * Author: Dr. Robertson
 * Date: January 1, 2015
 * Purpose: This program demonstrates the use of if/else selection
 * statements.
 */

// Import statements
import java.util.Scanner;

public class SelectionIfDemo {
    public static void main(String[] args) {

```

```

// Display a Welcome note
    System.out.println("Welcome to the if/else Demo");

// Variables to hold values
int month = 0;
int numDays = 0;

// Use the Scanner class to input data
    Scanner scannerIn = new Scanner(System.in);

System.out.println("Enter the Month (1-12):");
// the nextInt() method scans the next int value
    month = scannerIn.nextInt();

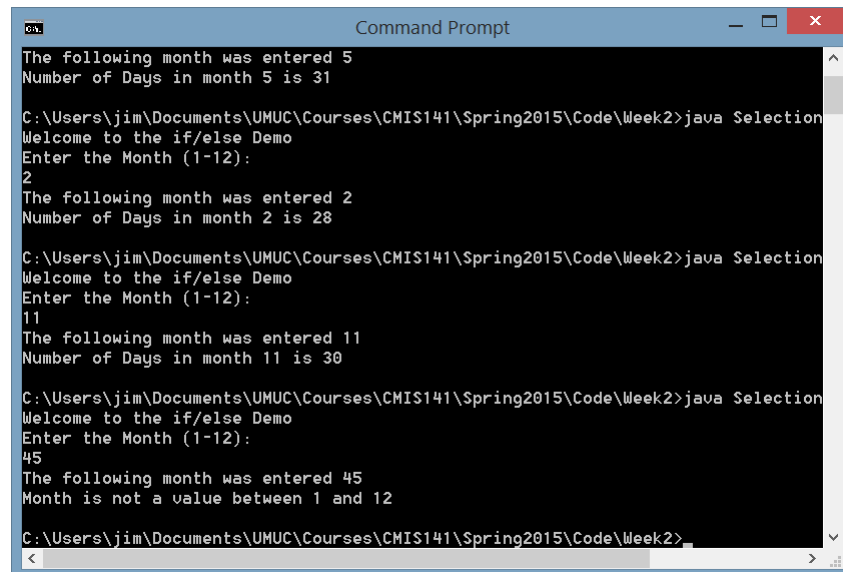
// Verify the month was entered
System.out.println("The following month was entered " + month);

// Selection statement to determine number
// of days in month
if ((month == 1) ||
    (month == 3) ||
    (month == 5) ||
    (month == 7) ||
    (month == 8) ||
    (month == 10) ||
    (month == 12) ) {
    numDays = 31;
}
else if (month == 2) {
    numDays = 28;
}
else if ((month == 4) ||
    (month == 6) ||
    (month == 9) ||
    (month == 11) ) {
    numDays = 30;
}
else {
    System.out.println("Month is not a value between 1 and
12");
    // Exit the application for invalid Months
    System.exit(0);
}

// Output the number of days in the selected month
System.out.println("Number of Days in month " + month +
    " is " + numDays);
}
}

```

- e. Finally, compile and run the program. Enter multiple test cases as input. Be sure to verify the code works for all possible valid month input values. You should also test for numbers outside of the range of valid months to make sure the application exits and provides the expected output message.



```
Command Prompt

The following month was entered 5
Number of Days in month 5 is 31

C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code\Week2>java Selection
Welcome to the if/else Demo
Enter the Month (1-12):
2
The following month was entered 2
Number of Days in month 2 is 28

C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code\Week2>java Selection
Welcome to the if/else Demo
Enter the Month (1-12):
11
The following month was entered 11
Number of Days in month 11 is 30

C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code\Week2>java Selection
Welcome to the if/else Demo
Enter the Month (1-12):
45
The following month was entered 45
Month is not a value between 1 and 12

C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code\Week2>
```

Now it is your turn. Try the following exercise:

Create a file named `MySelectionIfDemo.java`. Create an application that will return the total tax due based on an individual's income. The following logic should be applied. If the income is less than \$20,000 the tax is calculated by  $\text{income} * 0.10$ . If the income is less than \$40,000 the tax is calculated by  $\text{income} * 0.12$ . If the income is less than \$60,000 the tax is calculated by  $\text{income} * 0.14$ . For all other incomes greater than or equal to \$60,000 the tax is calculated by  $\text{income} * 0.15$ . If an income of less than 0 is entered the program should exit with a friendly message informing the user income must be greater than 0. After calculating the tax, the tax along with the user's income should be printed to Standard output. Be sure your modified code compiles and runs without issue. Test your application by making sure each possible input range yields the correct output, including values less than 0.

#### Exercise 4 – Design, develop and test switch statements in Java

The switch statement provides similar functionality to the if/else statement. When using the switch statement consider the following information:

1. Multiple cases can be used to provide many different processing paths.
2. The break statement is optional but if not used, additional case statements will be processed, until a break statement is found. Be careful as this might not be the results you desired. In most cases break statements should be used unless the logic or design of your application dictates otherwise.

For this exercise, we will create an application that used switch statements to display an event in a calendar based on a month entered by the user. For example, if the user enters 3 (for March), the event displayed could be “Celebrate St. Patrick’s Day!”.

- a. Open your favorite text editor and type or copy and paste the following code.

```
/*
 * File: SelectionSwitchDemo.java
 * Author: Your name here
 * Date: January 1, 2015
 * Purpose: This program demonstrates the use of switch selection
 * statements.
 */

public class SelectionSwitchDemo {
    public static void main(String[] args) {

        // Display a Welcome note
        System.out.println("Welcome to the Switch Demo");

    }
}
```

- b. Add code to allow the input of an integer month. This will include defining the variables and importing the Scanner class.

```
/*
 * File: SelectionSwitchDemo.java
 * Author: Dr. Robertson
 * Date: January 1, 2015
 * Purpose: This program demonstrates the use of switch selection
 * statements.
 */

// Import statements
import java.util.Scanner;

public class SelectionSwitchDemo {
    public static void main(String[] args) {

        // Display a Welcome note
        System.out.println("Welcome to the Switch Demo");

        // Variables to hold values
        int month = 0;
        int numDays = 0;

        // Use the Scanner class to input data
        Scanner scannerIn = new Scanner(System.in);
```

```

System.out.println("Enter the Month (1-12):");
// the nextInt() method scans the next int value
    month = scannerIn.nextInt();

// Verify the month was entered
System.out.println("The following month was entered " + month);

```

c. Add switch statements displaying a small list of monthly events based on the month entered.

```

// Selection statement to determine Holiday Greetings
    switch (month) {
        case 1:
            // Print out January Events
            System.out.println("Happy New Year!");
            System.out.println("Celebrate Martin Luther King's
            Birthday!");
            break;
        case 2:
            // Print out February Events
            System.out.println("Happy Valentine's Day!");
            System.out.println("Happy President's Day!");
            break;
        case 3:
            // Print out March Events
            System.out.println("Enjoy St. Patrick's Day!");
            break;
        case 4:
            // Print out April Events
            System.out.println("Thomas Jefferson was born in April!");
            break;
        case 5:
            // Print out May Events
            System.out.println("Enjoy Memorial Day!");
            System.out.println("Don't forget to Celebrate with Mom!");
            break;
        case 6:
            // Print out June Events
            System.out.println("Happy Father's Day!");
            break;
        case 7:
            // Print out July Events
            System.out.println("Happy 4th of July!");
            break;
        case 8:
            // Print out August Events
            System.out.println("Barack Obama was born in August!");
            break;
        case 9:
            // Print out September Events
            System.out.println("Celebrate Labor Day!");

```

```

        break;
    case 10:
        // Print out October Events
        System.out.println("Happy Halloween!");
        break;
    case 11:
        // Print out November Events
        System.out.println("Happy Thanksgiving Day!");
        System.out.println("Remember to Thank a Vet!");
        break;
    case 12:
        // Print out December Events
        System.out.println("Enjoy those Holidays with Family!");
        break;
    default:
        System.out.println("Month is not a value between 1 and
12");
        // Exit the application for invalid Months
        System.exit(0);
}

```

Feel free to add monthly events and activities of your choice.

- d. Finally, compile and run the program. Enter multiple test cases as input. Be sure to verify the code works for all possible valid month input values. You should also test for numbers outside of the range of valid months to make sure the application exits and provides the expected output message.

```

C:\Users\jim\Documents\UMUC\Courses\CHIS141\Spring2015\Code\Week2>java Selection
Welcome to the Switch Demo
Enter the Month (1-12):
1
The following month was entered 1
Happy New Year!
Celebrate Martin Luther King's Birthday!
Thank you for entering the month of 1

C:\Users\jim\Documents\UMUC\Courses\CHIS141\Spring2015\Code\Week2>java Selection
Welcome to the Switch Demo
Enter the Month (1-12):
2
The following month was entered 2
Happy Valentine's Day!
Happy President's Day!
Thank you for entering the month of 2

C:\Users\jim\Documents\UMUC\Courses\CHIS141\Spring2015\Code\Week2>

```

As always, experiment with code by additional functionality. Also, be sure to remove some of the break statements to explore this functionality.

Now it is your turn. Try the following exercise:

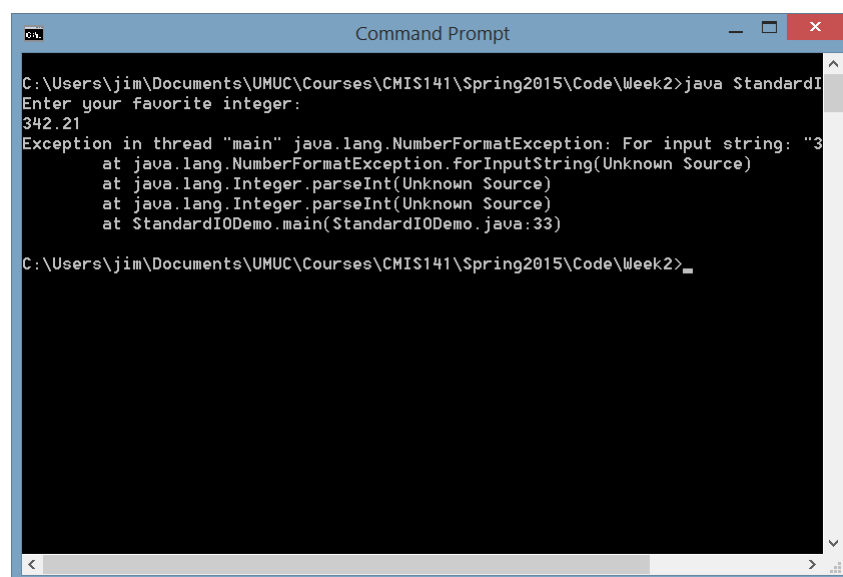
Create a file named `RolePlayingGame.java`. In this text-based role playing game, a user will enter a letter from the alphabet representing the direction of movement for a player in the game. You should determine and print out a message to standard output based on one of four different directional movements entered by the user including 'N', 'E', 'S' and 'W', representing North, East, South and West, respectively. Be creative in the output of your role playing game. Test your application by making sure each possible input range yields the correct output, including values that are not a specified valid move. (Hint: use the `scannerIn.next().charAt(0)` to read a character from standard input.)

### Exercise 5 - Analyze Java code to identify and correct common program errors

Programming errors prevent a program from functioning as expected. Common programming errors include compile, run-time, logic and specification. You have most likely already run into the first three.

Compile errors occur at compile time and stop the compile process before bytecodes are generated. Compile errors must be fixed before the program can be run. Typical compile errors include forgetting to include the semi-colon (;) at the end of Java statement, using a reserved word for a variable or forgetting to add the closing brace or parenthesis in a Java class. These are usually fairly easy to correct.

Run-time errors occur when you attempt to run the compiled bytecodes. Examples, include entering a data type that was not expected, or the location of a resource was not available. For example, if we entered a double or float value when an int was expected, a run-time error would occur. This is demonstrated while running the `StandardIODemo` class. If we enter a decimal number when the program expected an int, a run-time error results. Run-time usually provide the line number and other information to help determine the issue.



```
C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code\Week2>java StandardI
Enter your favorite integer:
342.21
Exception in thread "main" java.lang.NumberFormatException: For input string: "3
    at java.lang.NumberFormatException.forInputString(Unknown Source)
    at java.lang.Integer.parseInt(Unknown Source)
    at java.lang.Integer.parseInt(Unknown Source)
    at StandardIODemo.main(StandardIODemo.java:33)

C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code\Week2>
```

Logic errors are more challenging to identify as they usually don't evoke an error from the application. In most cases the application runs. However; the output may not be the expected or desired result. For example, if the program needed to provide the product of two numbers and the programmer mistakenly coded the sum of two numbers, the program would run, but the results would not be correct. Detailed and comprehensive testing is the best way to catch logic errors.

Specification errors result from a requirement that was not correctly understood or documented. The program does not produce the desired outcome due to miscommunication during the requirements gathering phase. It could also result from a programmer modifying or enhancing the code based on what he/she thinks the customer wants. Regardless, of how the miscommunication occurs, the customer ends up with a product that does not work as they expected.

In this exercise we will demonstrate run-time and logic errors. Let us create a Java class named `ErrorsDemo.java` that allows a user to submit data for a simple survey that collects age, average exam scores and if they are married.

- a. Open your favorite text editor and type or copy and paste the following code.

```
/*
 * File: ErrorsDemo.java
 * Author: Your name here
 * Date: January 1, 2015
 * Purpose: This program demonstrates the generation and
 * mitigation of common programming errors
 */

// Import each required Java class
import java.util.Scanner;

public class ErrorsDemo {
    public static void main(String[] args) {

        // Variables to hold values
        byte age = 0;
        double averageScore = 81.9;
        boolean isMarried = false;

        // Use the Scanner class to read System.in
        Scanner scannerIn = new Scanner(System.in);

        // Prompt the user to enter another int
        System.out.println("Enter your age:");
        // the nextByte() method scans the next int value
        age = scannerIn.nextByte();

        // Prompt the user to enter another double
        System.out.println("Enter the average exam score:");
        // the nextDouble() method scans the next int value
        averageScore = scannerIn.nextDouble();
    }
}
```



```

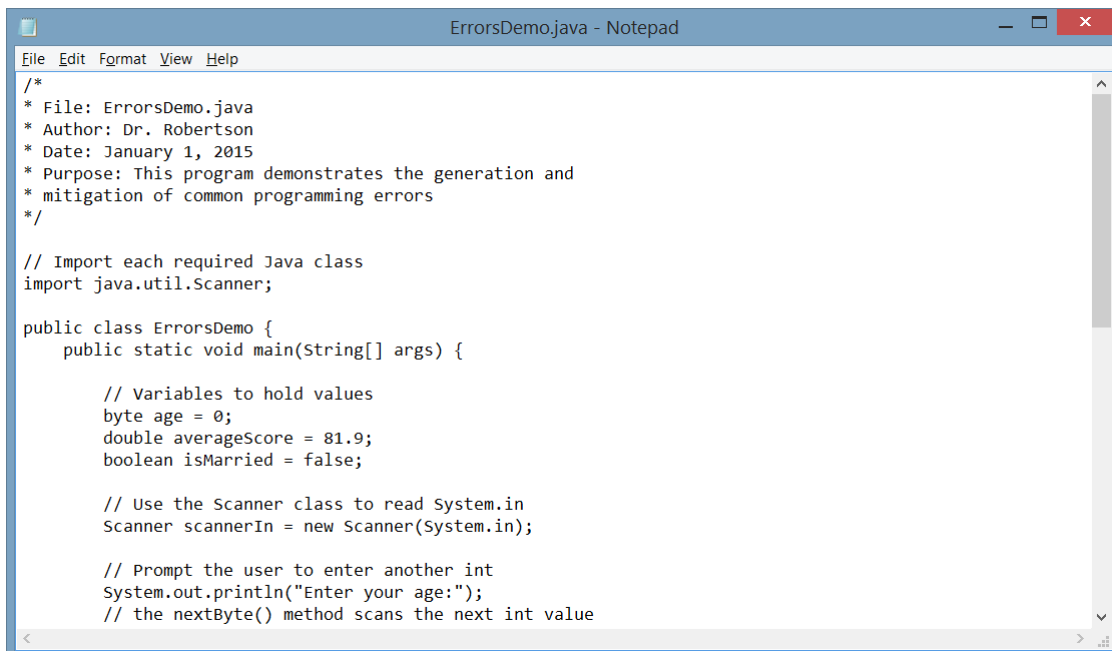
// Prompt the user to enter another double
System.out.println("I am married (true or false):");
// the nextBoolean() method scans the next boolean value
isMarried = scannerIn.nextBoolean();

// Print the results to verify your data
System.out.println("Thank you for responding to our survey.");
System.out.println("Your results are: ");
System.out.println("Your age is: " + age);
System.out.println("The average Score is: " + averageScore);
// Logic to print Marriage status
if (isMarried) {
    System.out.println("I am married.");
}
else {
    System.out.println("I am currently not married.");
}

}
}

```

b. Save your file as ErrorsDemo.java.



```

ErrorsDemo.java - Notepad
File Edit Format View Help
/*
 * File: ErrorsDemo.java
 * Author: Dr. Robertson
 * Date: January 1, 2015
 * Purpose: This program demonstrates the generation and
 * mitigation of common programming errors
 */

// Import each required Java class
import java.util.Scanner;

public class ErrorsDemo {
    public static void main(String[] args) {

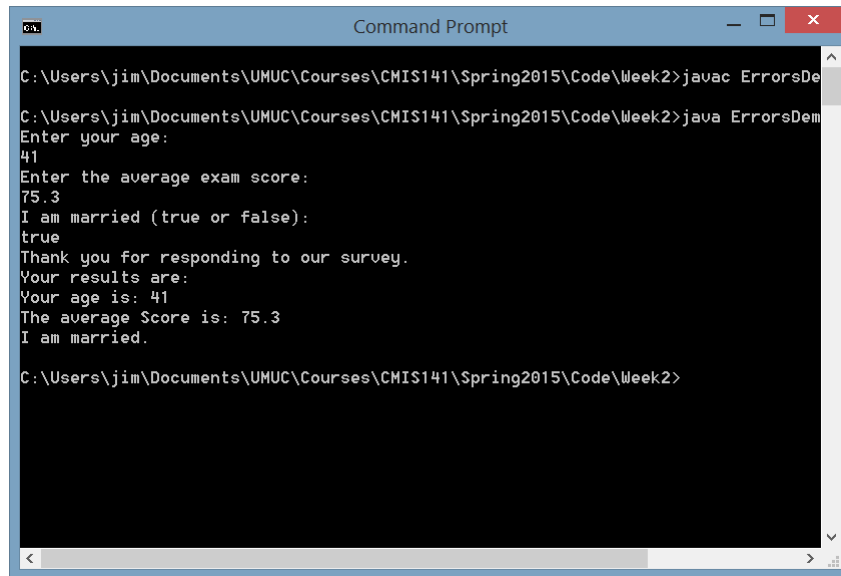
        // Variables to hold values
        byte age = 0;
        double averageScore = 81.9;
        boolean isMarried = false;

        // Use the Scanner class to read System.in
        Scanner scannerIn = new Scanner(System.in);

        // Prompt the user to enter another int
        System.out.println("Enter your age:");
        // the nextByte() method scans the next int value
    }
}

```

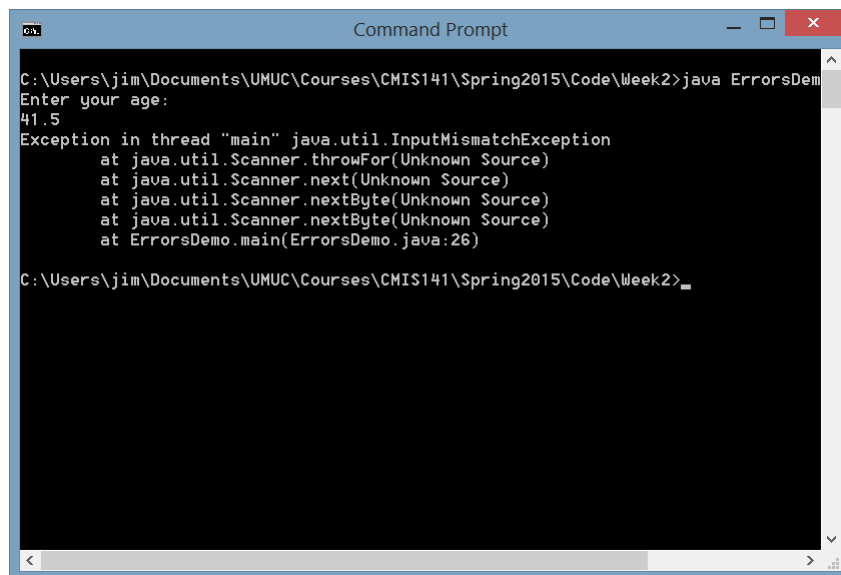
c. Compile and run your program entering values for age, averageScore and isMarried when prompted.



```
Command Prompt
C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code\Week2>javac ErrorsDemo.java
C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code\Week2>java ErrorsDemo
Enter your age:
41
Enter the average exam score:
75.3
I am married (true or false):
true
Thank you for responding to our survey.
Your results are:
Your age is: 41
The average Score is: 75.3
I am married.
C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code\Week2>
```

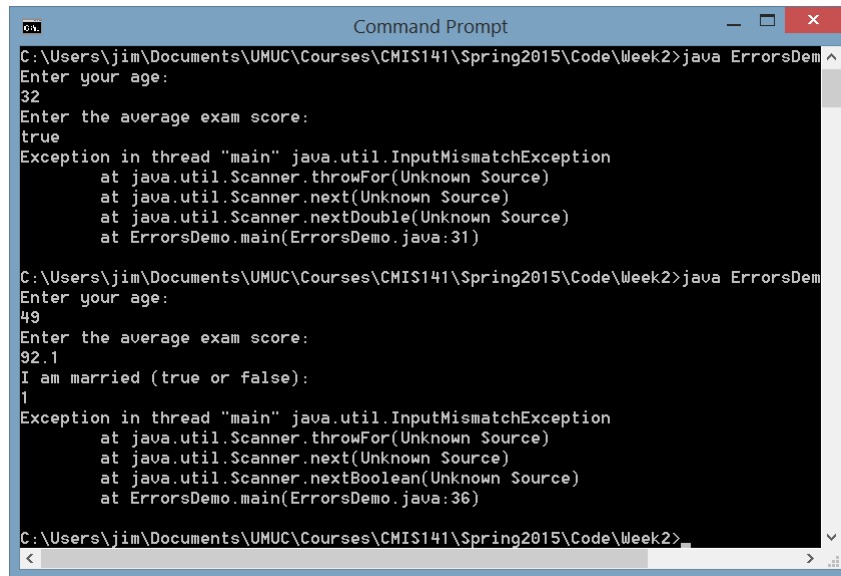
- d. Next, without changing the code, let us generate some run-time errors by entering incorrect data types when prompted for our data. In this case we will enter 41.5 for the age.

Notice the program exits immediately upon detecting the run-time error. The error message is fairly descriptive in that the program was expecting a byte (whole number between -128 and 127) and received a decimal value of 41.5



```
Command Prompt
C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code\Week2>java ErrorsDemo
Enter your age:
41.5
Exception in thread "main" java.util.InputMismatchException
    at java.util.Scanner.throwFor(Unknown Source)
    at java.util.Scanner.next(Unknown Source)
    at java.util.Scanner.nextByte(Unknown Source)
    at java.util.Scanner.nextByte(Unknown Source)
    at ErrorsDemo.main(ErrorsDemo.java:26)
C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code\Week2>
```

- e. Similarly, if we enter an incorrect data type for the averageScore and isMarried variables, errors run-time errors will be generated.



```
C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code\Week2>java ErrorsDem
Enter your age:
32
Enter the average exam score:
true
Exception in thread "main" java.util.InputMismatchException
    at java.util.Scanner.throwFor(Unknown Source)
    at java.util.Scanner.next(Unknown Source)
    at java.util.Scanner.nextDouble(Unknown Source)
    at ErrorsDemo.main(ErrorsDemo.java:31)

C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code\Week2>java ErrorsDem
Enter your age:
49
Enter the average exam score:
92.1
I am married (true or false):
1
Exception in thread "main" java.util.InputMismatchException
    at java.util.Scanner.throwFor(Unknown Source)
    at java.util.Scanner.next(Unknown Source)
    at java.util.Scanner.nextBoolean(Unknown Source)
    at ErrorsDemo.main(ErrorsDemo.java:36)

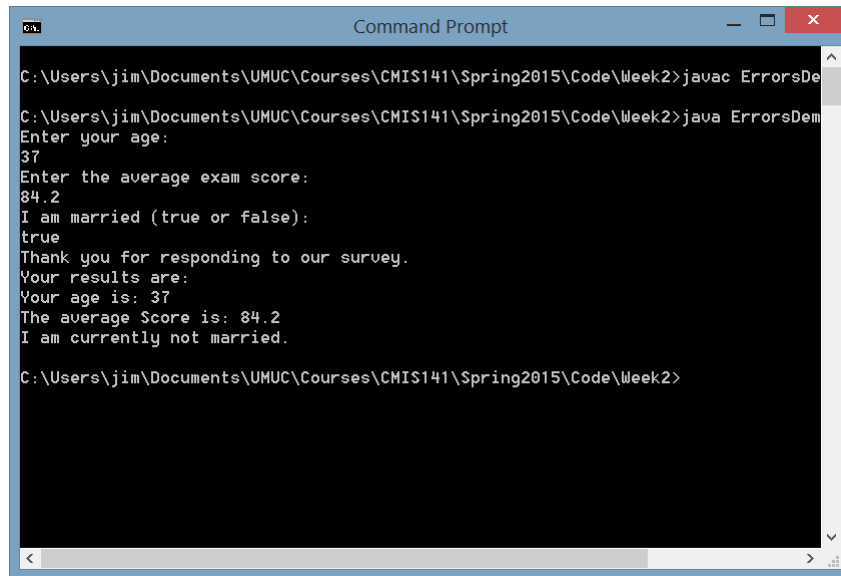
C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code\Week2>
```

You should experiment by running this code and attempt to generate run-time errors. Notice, if you attempt to enter an integer value (e.g. 92) for the average score, no error is generated. This is because the integer is just treated like 92.0 and no loss of data occurs.

- f. Finally, to generate a logic error, let us change the logic in the marriage print statement slightly.

```
// Logic to print Marriage status
if (!isMarried) {
    System.out.println("I am married.");
}
else {
    System.out.println("I am currently not married.");
}
```

Notice the first line was changed to add the Not (!) operator. This code will compile and run but the answer is not expected or correct. Notice the logic error is causing the output of the marriage status to be incorrect.



```
C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code\Week2>javac ErrorsDe
C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code\Week2>java ErrorsDem
Enter your age:
37
Enter the average exam score:
84.2
I am married (true or false):
true
Thank you for responding to our survey.
Your results are:
Your age is: 37
The average Score is: 84.2
I am currently not married.

C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code\Week2>
```

Testing your code is the best way to detect run-time and logic errors. Comprehensive testing includes testing all possible branches of selection statements and testing with a variety of data values.

Now it is your turn. Try the following exercise:

Open up your RolePlayingGame.java file you created in the last exercise. Compile and run the example. Experiment by entering invalid data types when prompted. For example, enter numbers, special characters and other values. Did you generate a run-time error? Why or why not? Try to introduce a logic error. For example, remove the break statements or change the logic based on the data entry.