CMIS 141 Hands-on Lab
Week 7

Overview

The week we continue our study of Java by providing information and examples on classes to work with String data and the Java 8 Date-Time API, and how to use command line arguments. In this lab, we will work with the StringBuffer, and StringBuilder, use the java.time package and pass command line arguments to an application.The java.time package is new in Java 8 so you must be using the JDK 8 or above to demonstration this functionality.

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:

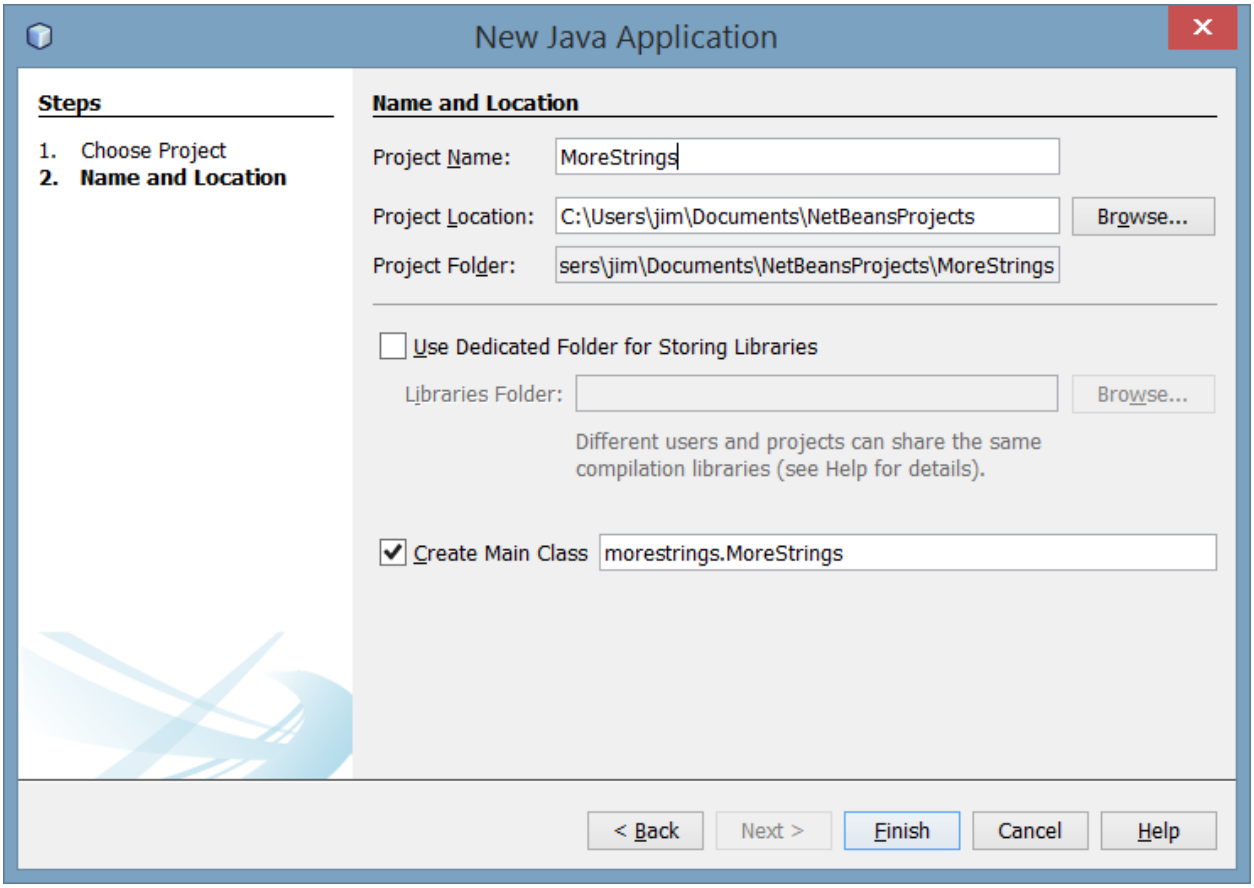
1. Compare, contrast and use the *String, StringBuffer and StringBuilder classes*
2. Use classes in the java.time package
3. Use command line arguments in Java

Exercise 1 – Compare, contrast and use the *String, StringBuffer and StringBuilder classes*

As mentioned earlier in the semester when Strings were introduced, Strings are immutable. There are additional classes including the StringBuffer and StringBuilder classes that can be used when you want mutable Strings. The String class should typically be used unless you envision significant changes of the String. The StringBuffer class methods are thread safer and although not as efficient as the StringBuilder may be a good choice if you are using threads and concurrent processing.

In this exercise, we will use the String, StringBuffer and StringBuilder classes to construct several objects and then use select methods from each. Visiting the Java 8 API to compare the methods and constructors of each of these class is recommended. Since we are now comfortable using an IDE, we will use the IDE for all remaining exercises this semester.

a. Launch your IDE and create a new project named MoreStrings.

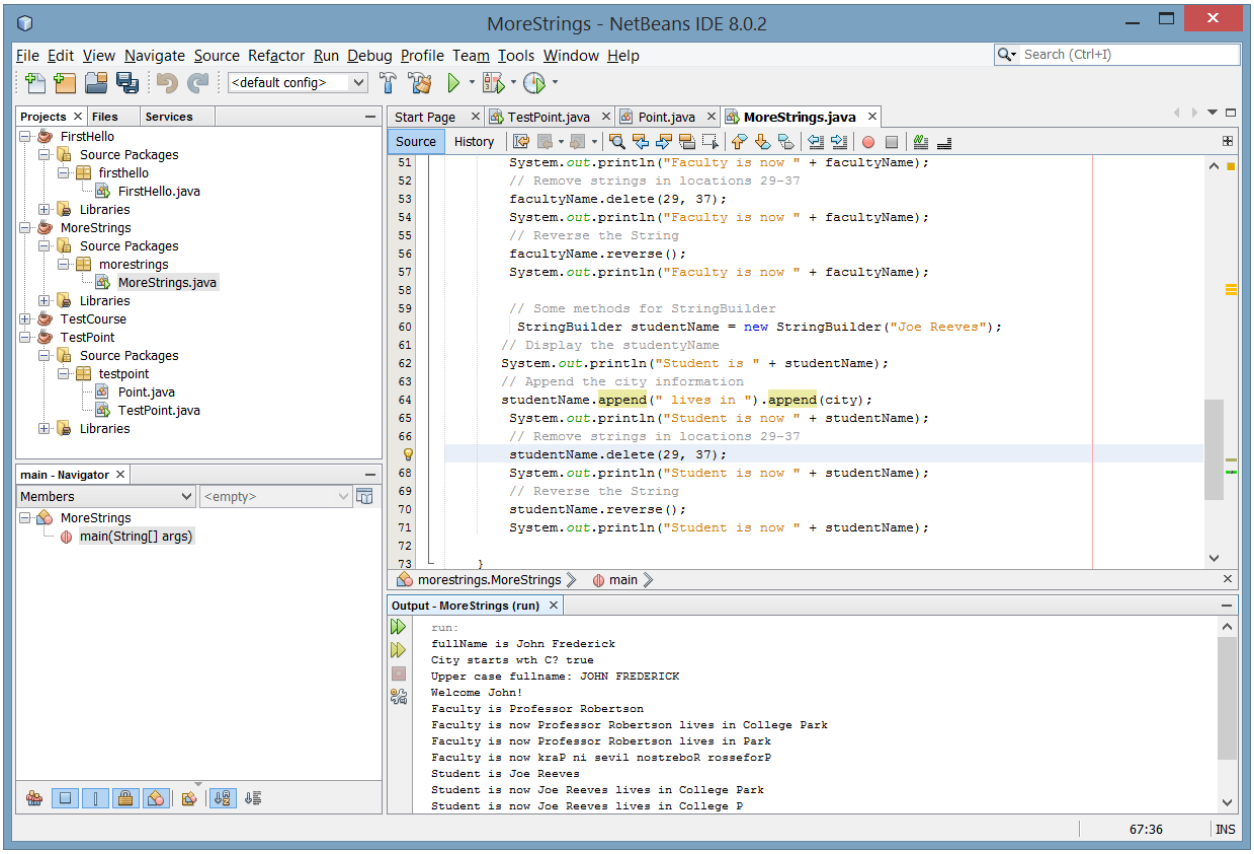


b. Type or copy and paste the following java code into the MoreStrings.java file in your IDE.

```
package morestrings;

/*
* File: MoreString.java
* Author: Dr. Robertson
* Date: January 1, 2015
* Purpose: This program constructs String,
* StringBuffer and StringBuilder objects
* and uses several methods
*/

public class MoreStrings {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        // Create Strings
        String firstName = new String ("John");
        String lastName = new String("Frederick");
        String city = new String ("College Park");
        // Use String methods
```

```
        // Concatenate first and lastname - add space
        String fullName = firstName.concat(" ").concat(lastName);
        System.out.println("fullName is " + fullName);
        // Check if city starts with C
        System.out.println("City starts with C? " + city.startsWith("C"));
        // Display UpperCase
        System.out.println("Upper case fullname: " + fullName.toUpperCase());
        // Comparison returns true
        if (firstName.equals("John")){
            System.out.println("Welcome John!");
        }
        // Comparison returns false
        if(lastName.equals(firstName)) {
            System.out.println("Your first and last name are the same");
        }

        // Add some StringBuffers
        StringBuffer facultyName = new StringBuffer("Professor Robertson");
        // Display the facultyName
        System.out.println("Faculty is " + facultyName);
        // Append the city information
        facultyName.append(" lives in ").append(city);
         System.out.println("Faculty is now " + facultyName);
         // Remove strings in locations 29-37
         facultyName.delete(29, 37);
         System.out.println("Faculty is now " + facultyName);
         // Reverse the String
         facultyName.reverse();
         System.out.println("Faculty is now " + facultyName);

         // Some methods for StringBuilder
          StringBuilder studentName = new StringBuilder("Joe Reeves");
        // Display the studentyName
        System.out.println("Student is " + studentName);
        // Append the city information
        studentName.append(" lives in ").append(city);
         System.out.println("Student is now " + studentName);
         // Remove strings in locations 29-37
         studentName.delete(29, 37);
         System.out.println("Student is now " + studentName);
         // Reverse the String
         studentName.reverse();
         System.out.println("Student is now " + studentName);

    }

}
```

c. Compile and run the code by clicking on the green arrow in the IDE.



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

1. String, StringBuffer and StringBuilder provide similar functionality for working with Strings.
2. StringBuffer and StringBuilder methods emphasis changing the String. Notice the reverse() and append() methods.

```
studentName.append(" lives in ").append(city);
studentName.reverse();
```

3. The methods for StringBuffer and StringBuilder are almost identical in terms of name and functionality.

As always you should experiment with the code by constructing additional String, StringBuffer and StringBuilder objects and use multiple methods in each class. Use the Java 8 API to look up methods functionality to better understand how each method could be applied.

Now it is your turn. Try the following exercise:

Using your IDE, create a project named MyMoreStrings. Construct at least 5 different Strings, StringBuffer and StringBuilder objects, using arrays. For each of the objects created, call and print the results of at least 5 different methods. Hint: Using a loop will allow you to efficiently cycle through each object in the array.

Exercise 2 – Use classes in the java.time package

New to Java 8 is a Date and Time package containing classes based on the calendar system defined in ISO-8601 standard. Similar to the Strings class, most of the methods in the Date-Time API are immutable. A new object must be constructed to store altered values of an immutable object.

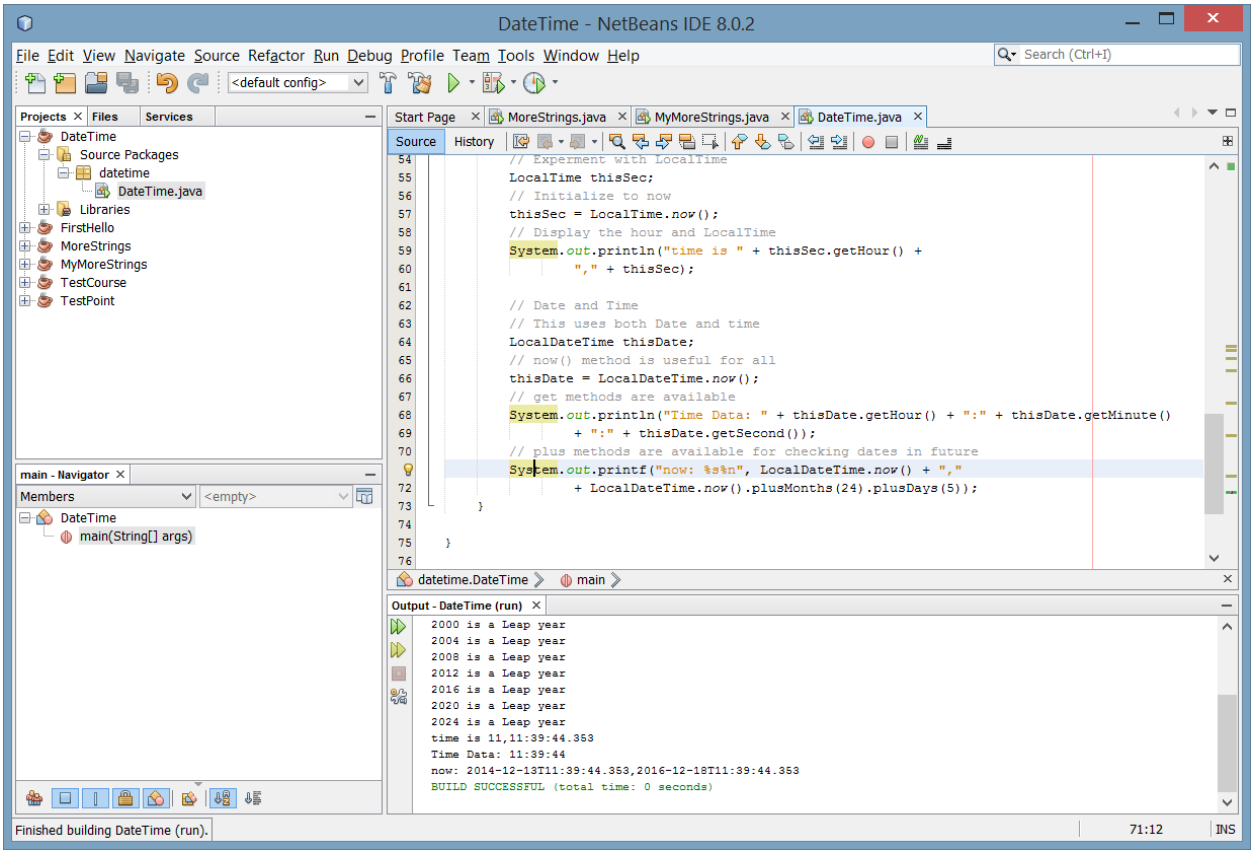
The Date-Time API in Java is complex and comprehensive covering all time zones, temporal adjustments and the ability to create your own calendar. This exercise will demonstrate some of the simpler and more common classes and methods in the java.time package.

a. Launch your IDE and create a new project named DateTime.
b. Type or copy and paste the following java code into the DateTime.java file in your IDE.

```
package datetime;

import java.time.DayOfWeek;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.LocalTime;
import java.time.Year;

/*
 * File: DateTime.java
 * Author: Dr. Robertson
 * Date: January 1, 2015
 * Purpose: This program demonstrates
 * the use of the classes in the
 * java.time package
 */
public class DateTime {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // Use LocalDate Class
        // now() uses current local date
        LocalDate date = LocalDate.now();

        // get Year, Julian day of year
        // and Day of Month
        int year = date.getYear();
        int yearDay = date.getDayOfYear();
        int monthDay = date.getDayOfMonth();

        // Print results
```

```
        System.out.println("Year is " + year);
        System.out.println("Julian day is " + yearDay);
        System.out.println("Day of month is " + monthDay);

        // Use plus method
        System.out.printf("%s%n", DayOfWeek.MONDAY.plus(4));

        // Use of method and loop
        // to determine leap years
        for (int i = 1990; i < 2025; i++) {
            boolean isLeap = Year.of(i).isLeap();
            if (isLeap) {
                System.out.println(i + " is a Leap year");
            }
        }
        // Experment with LocalTime
        LocalTime thisSec;
        // Initialize to now
        thisSec = LocalTime.now();
        // Display the hour and LocalTime
        System.out.println("time is " + thisSec.getHour() +
                "," + thisSec);

        // Date and Time
        // This uses both Date and time
        LocalDateTime thisDate;
        // now() method is useful for all
        thisDate = LocalDateTime.now();
        // get methods are available
        System.out.println("Time Data: " + thisDate.getHour() + ":" +
thisDate.getMinute()
                + ":" + thisDate.getSecond());
        // plus methods are available for checking dates in future
        System.out.printf("now: %s%n", LocalDateTime.now() + ","
                + LocalDateTime.now().plusMonths(24).plusDays(5));
    }

}
```

c. Compile and run the code by clicking on the green arrow in the IDE.



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

1. The now() method is available and useful for most Date-Time API classes. You use the now() method to establish the current time.

```
LocalDate date = LocalDate.now();
LocalTime thisSec;
thisSec = LocalTime.now();
LocalDateTime thisDate;
thisDate = LocalDateTime.now();
```

2. Using get methods in all the classes provides access to specific fields such as year, month, day, Julian day, day of week, hours, minutes, and seconds. Exploring the Java API is encouraged to make use of the many get methods in each of the classes.

```
int year = date.getYear();
int yearDay = date.getDayOfYear();
int monthDay = date.getDayOfMonth();
```

3. Using the plus methods allows access to future dates. You can “daisy-chain” the methods to increment multiple values. For example, this code adds 24 months and 5 days to the current date.

```
LocalDateTime.now().plusMonths(24).plusDays(5))
```

Now it is your turn. Try the following exercise:

Using your IDE, create a project named TimeDiff. Create an application that determines how long a loop or sequence of code takes to execute. Hint: take Instant snap shots of time before the code sequence and after the sequence to determine the difference. Use the Duration class to calculate the time difference in nanoseconds and then convert to seconds.

Exercise 3 – Use command line arguments in Java

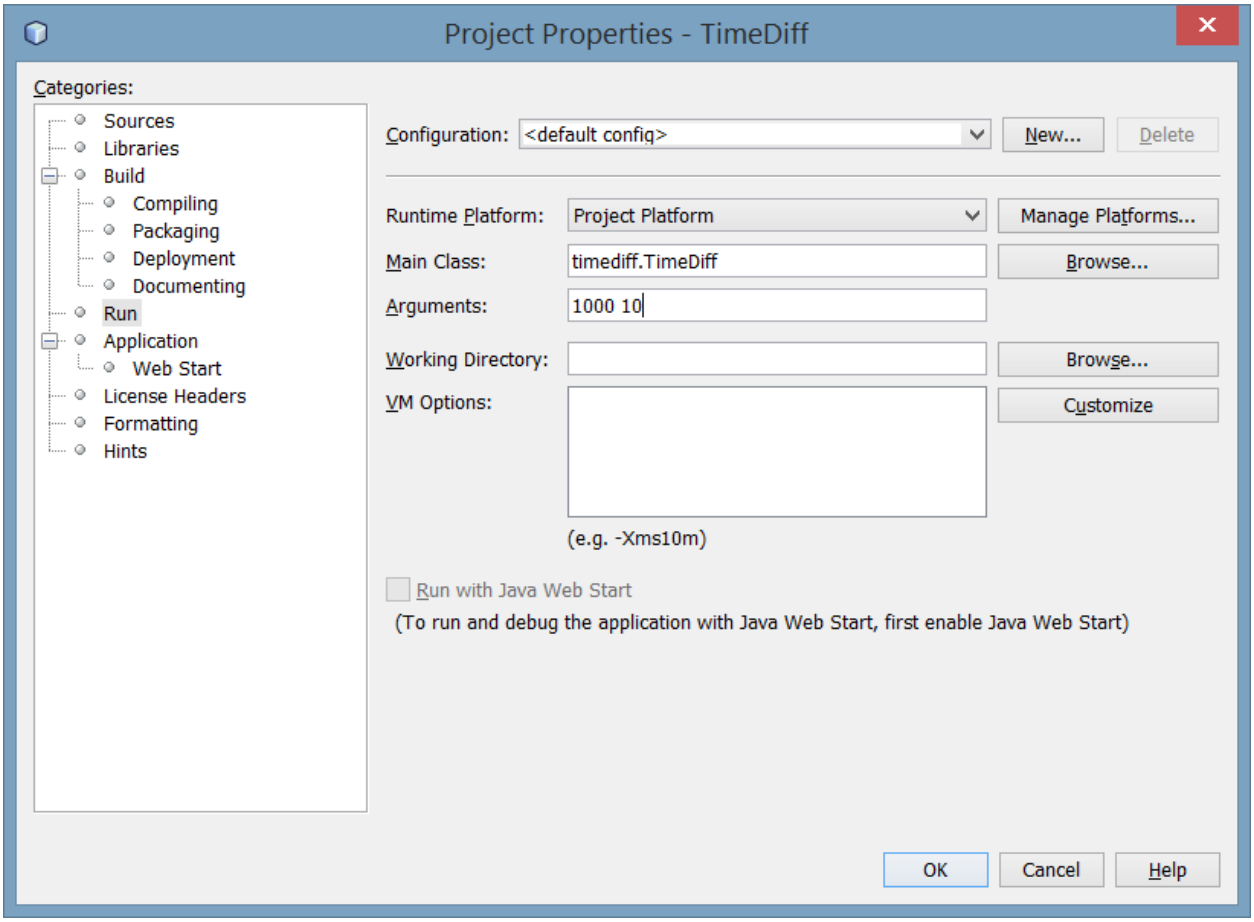
If you need to run programs over and over with different parameters, command line arguments are useful. Command line arguments allow you to run a program and send input parameters into the program. Imagine needing to load different filenames but you don't want to recompile each time. You can could send the filename into the program using command line arguments:

java RunCommands Filename1

In this example, the Java program named RunCommands uses Filename1 as an input parameter.

Command line arguments can be used directly at the command line when you launch from your DOS (or command prompt). You can also enter command line arguments in Netbeans so you don't have to open a Command prompt.

In Netbeans, you can set-up Command Line arguments by selecting
Run->Set Project Configuration->Custom.
Then add your parameters in the Arguments text field. In the screen shot below I added two command line arguments – 1000 and 10.



If we entered the following code to the main method, we could print the arguments entered at the command prompt.

```
for (int i=0;i<args.length;i++){
      System.out.println("index: Command line argument is " + i + ":"
      +args[i]);
  }
```

This would result in this output for the parameters above.

```
index: Command line argument is 0:1000
index: Command line argument is 1:10
```

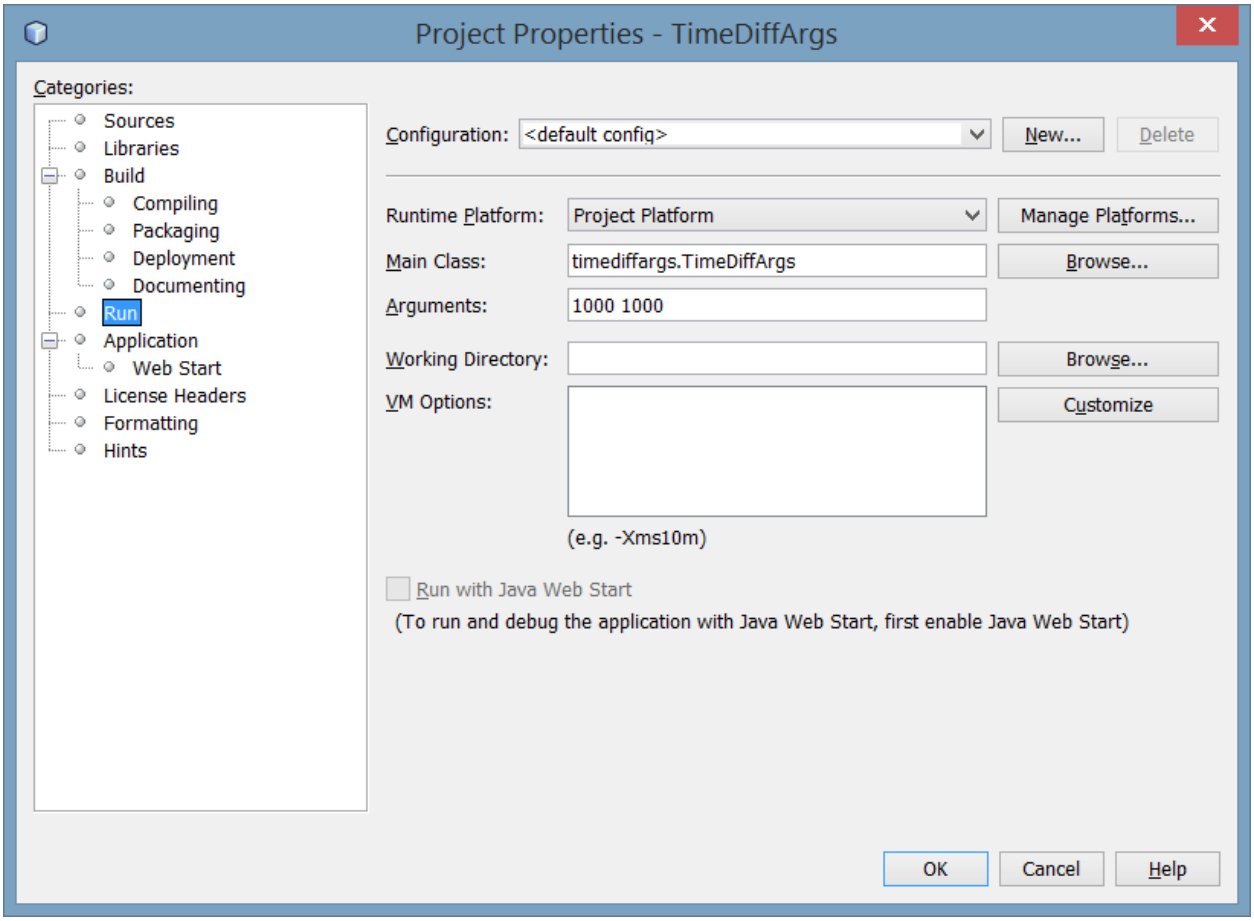
The command line arguments can then be used in the code to allow the code to be run over and over again without recompiling. For example we could send in the loop stop values as command arguments and run the TimeDiff project using those parameters.

To demonstrate this, complete the following exercise:

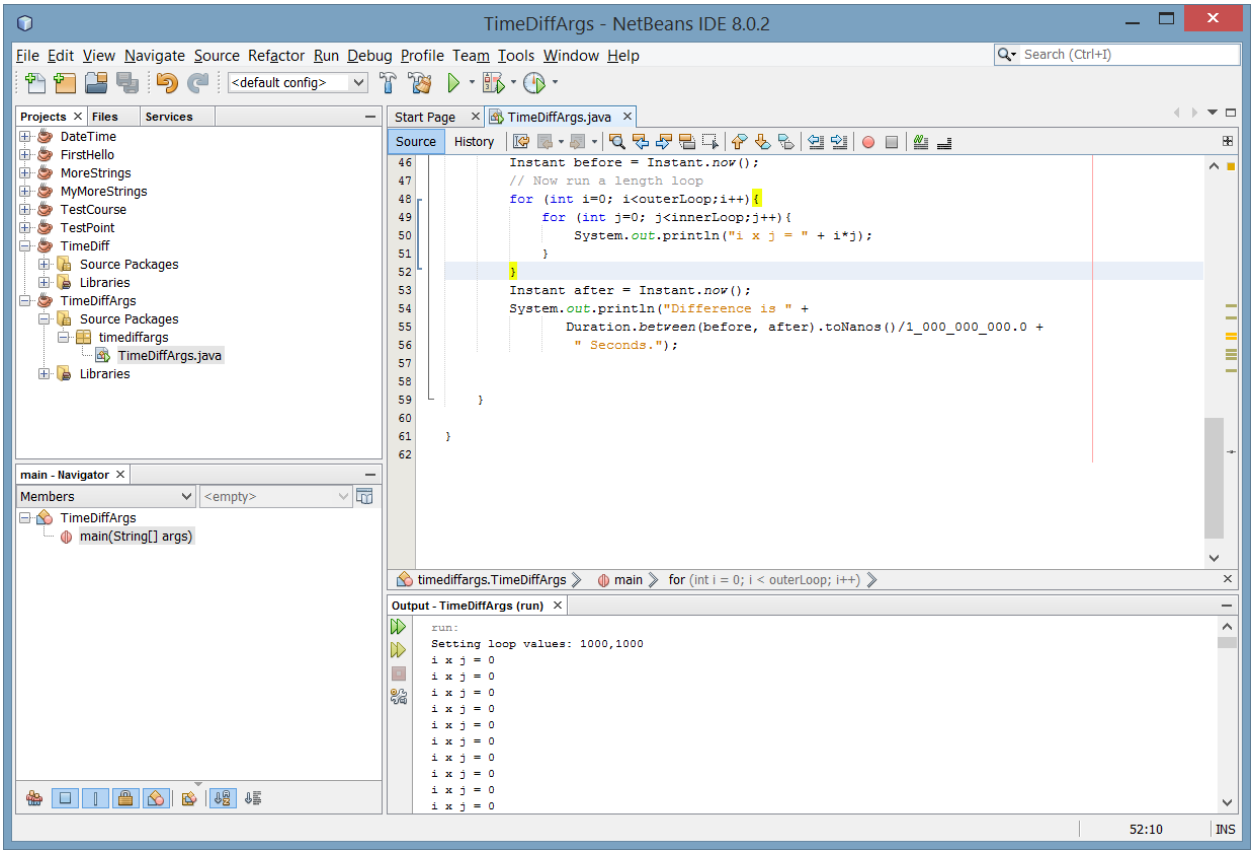
a. Launch your IDE and create a new project named TimeDiffArgs.
b. Type or copy and paste the following java code into the TimeDiffArgs.java file in your IDE.

```
package timediffargs;

import java.time.Duration;
import java.time.Instant;

/*
* File: TimeDiffArgs.java
* Author: Dr. Robertson
* Date: January 1, 2015
* Purpose: This program demonstrates
* sending in command line input
* parameters
*/
public class TimeDiffArgs {

    /**
     * @param args the command line arguments
     */
     /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {

        // Default values
        int outerLoop = 100;
        int innerLoop = 10;
        // Check to make sure we have command line argument
        if (args.length == 2){
            outerLoop = Integer.parseInt(args[0]);
            innerLoop = Integer.parseInt(args[1]);
            System.out.println("Setting loop values: " +
                    args[0] + "," + args[1]);
        }
        else {
            System.out.println("Application requires 2 command arguments");
            System.out.println("e.g. java TimeDiffArgs 1000 10");
            System.exit(0);
        }

        // Snap an instance
        Instant before = Instant.now();
        // Now run a length loop
        for (int i=0; i<outerLoop;i++){
            for (int j=0; j<innerLoop;j++){
                System.out.println("i x j = " + i*j);
            }
        }
        Instant after = Instant.now();
        System.out.println("Difference is " +
               Duration.between(before, after).toNanos()/1_000_000_000.0 +
                " Seconds.");


    }

}
```

c. Be sure to configure command line arguments for your outer and inner loop stop values in the project by selecting the run-> Set Project Configuration-> Customize. And then add your two integer values to the Arguments text field.



d. Compile and run the code by clicking on the green arrow in the IDE.



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

1. Command line arguments are stored in the args String array. Note this is a String array. If you are entering numeric values, you need to convert to the appropriate number type by using the wrapper classes. (e.g. Integer, Float, Double …) The code checks to see that 2 values have been entered. If the args.length is not equal to 2 then the application will provide a message to the user specifying the program usage and exiting the program.

```
// Default values
        int outerLoop = 100;
        int innerLoop = 10;
        // Check to make sure we have command line argument
        if (args.length == 2){
            outerLoop = Integer.parseInt(args[0]);
            innerLoop = Integer.parseInt(args[1]);
            System.out.println("Setting loop values: " +
                    args[0] + "," + args[1]);
        }
        else {
            System.out.println("Application requires 2 command arguments");
            System.out.println("e.g. java TimeDiffArgs 1000 10");
            System.exit(0);
        }
```

2. The values input into from the command line arguments are used in the code to control the loops.

```
for (int i=0; i<outerLoop;i++){
       for (int j=0; j<innerLoop;j++){
            System.out.println("i x j = " + i*j);
        }
}
```

Now it is your turn. Try the following exercise:

Using your IDE, create a project named GenRandom. The application will generate a number of Random integers based on input from the user. The input from the user should come from command line arguments. Exit the program with a user-friendly message if a command line argument is not entered.