CMIS 141 Hands-on Lab
Week 6

Overview

The week we continue our study of classes and objects and start to use an Integrated Development Environment (IDE). In this lab, we will focus on more of Java's language features and class design. An IDE will be used to help with compiling and building our programs. IDE's assist in programming by providing an easy-to-use environment for testing debugging code.

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 and contrast *class* and *local* variables
2. Use the *this* reference
3. Create overloaded constructors
4. Use an IDE to compile and run an application consisting of more than one class

Exercise 1 – Compare and contrast class and local variables

Variable scope refers to where a variable is accessible within a class. If a variable is defined within a method it is considered a local variable and is only accessible within that method. If a variable is defined outside of a method, it is a class variable and is accessible anywhere within the class.

Class variables can be declared as static. As mentioned before static variables means there is only one instance of the variable. It will be shared by all objects constructed from the class. The following example creates a class that has both class and local variables. It also provides static class variables and demonstrates the accessibility of each.

a. Open your favorite text editor and type (or copy and paste) the following Java:

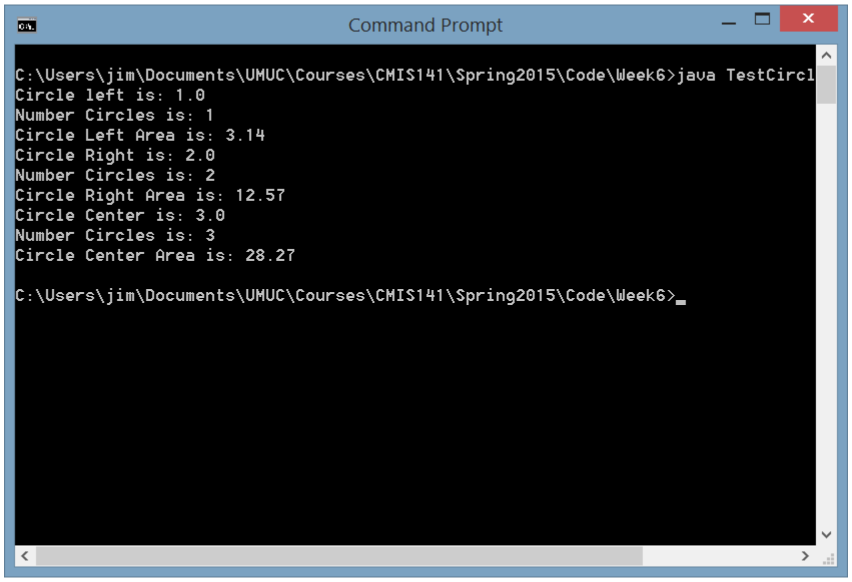
```
/*
* File: CircleScopeDemo.java
* Author: Dr. Robertson
* Date: January 1, 2015
* Purpose: This program demonstrates the scope
* of class and local variables
*/

public class CircleScopeDemo {
 // Define a static class variable
 // Hold number of objects
   private static int numCircles = 0;
 // Define class variable
   private double radius = 1.0;

  // Constructors
  // Default constructor
   public CircleScopeDemo() {
        this.radius = 1.0;
        numCircles++;
  }
  public CircleScopeDemo (double r) {
      radius = r;
        numCircles++;
  }

  // Area method
  public double getArea() {
     return Math.PI*Math.pow(radius,2);
  }

  // getter method
  public double getRadius() {
     return radius;
  }

  public static int getNumCircles() {
    return numCircles;
  }

}
```

b. Save the file as “CircleScopeDemo.java” in a location of your choice.
c. To compile the file, type javac CircleScopeDemo.java at the command prompt.
d. You will need a test class to construct several CircleScopeDemo objects. Open up your favorite text editor and type the following code:

```
/*
* File: TestCircleScopeDemo.java
* Author: Dr. Robertson
* Date: January 1, 2015
* Purpose: This program constructs instances
* of the CircleScopeDemo
*/

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

      // Construct a default circle
        CircleScopeDemo circleLeft = new CircleScopeDemo();

      // Call the get values
```

```
        System.out.println("Circle left is: " +
                  circleLeft.getRadius());
          System.out.println("Number Circles is: " +
                 CircleScopeDemo.getNumCircles());
        System.out.println("Circle Left Area is: " +
                 String.format("%.2f",circleLeft.getArea()));

        // Construct another Circle
        CircleScopeDemo circleRight = new CircleScopeDemo(2.0);
        System.out.println("Circle Right is: " +
                circleRight.getRadius());
          System.out.println("Number Circles is: " +
                CircleScopeDemo.getNumCircles());
        System.out.println("Circle Right Area is: " +
                String.format("%.2f",circleRight.getArea()));

        // Construct another Circle
        CircleScopeDemo circleCenter = new CircleScopeDemo(3.0);
        System.out.println("Circle Center is: " +
                 circleCenter.getRadius());
          System.out.println("Number Circles is: " +
                 CircleScopeDemo.getNumCircles());
        System.out.println("Circle Center Area is: " +
                 String.format("%.2f",circleCenter.getArea()));

      }
}
```

e. Save the file as TestCircleScopeDemo.java in the same location where you saved the CircleScopeDemo.java file. Compile and execute the program to observe the following output:



```
C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code\Week6>java TestCircl
Circle left is: 1.0
Number Circles is: 1
Circle Left Area is: 3.14
Circle Right is: 2.0
Number Circles is: 2
Circle Right Area is: 12.57
Circle Center is: 3.0
Number Circles is: 3
Circle Center Area is: 28.27

C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code\Week6>_
```

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

1. The CircleScopeDemo class contains two private class variables. numCircles is static and radius is not. This makes sense as radius will be unique to each instance of the CircleScopeDemo class created and should not be a shared variable. However; the numCircles will be shared and should be declared as static. This way, as we create each new instance, the numCircles counter is increased and we always know how many circles we have.

```
        private static int numCircles = 0;
        private double radius = 1.0;
```

2. The CircleScopeDemo class contains two different constructors. The default no argument constructor sets the radius to 1.0 and then increments the numCircles static class variable. The second constructor allows a user to enter the specific radius of the circle as an argument. This argument is then used to set the radius for the instance and then the numCircles static class variable is incremented.

```
       public CircleScopeDemo() {
          this.radius = 1.0;
          numCircles++;
       }
       public CircleScopeDemo (double r) {
          radius = r;
          numCircles++;
       }
```

3. A local variable named area is used in the getArea() method of the CircleScopeDemo class. It is local since it is only accessible within the getArea() method. If you attempt to access it, or print it from any location in the class, a compile error would result.

```
    public double getArea() {
      double area = Math.PI*Math.pow(radius,2);
      return area;
    }
```

4. The CircleScopeDemo class has two getter methods. One returns the value of the radius instance variable whereas the other returns the value of the numCircles static class variable. Notice the getNumCircles() method is declared as static. This is required to be able to return the static variable without a compile error.

```
  public double getRadius() {
       return radius;
    }

    public static int getNumCircles() {
      return numCircles;
  }
```

5. In the TestCircleScopeDemo we construct 3 instances of the CircleScopeDemo class. Each time we construct an instance, one of the constructors is called and the static numCircles class variable in incremented. The getArea(), getRadius() and getNumCircles() methods are called each time after construction of the instance to demonstrate the numCircles static class variable is properly incremented and the methods return the expected results.

```
CircleScopeDemo circleLeft = new CircleScopeDemo();

      // Call the get values
      System.out.println("Circle left is: " +
                circleLeft.getRadius());
        System.out.println("Number Circles is: " +
               CircleScopeDemo.getNumCircles());
      System.out.println("Circle Left Area is: " +
               String.format("%.2f",circleLeft.getArea()));
```

You should experiment with the code by constructing additional CircleScopeDemo instances. Note the behavior of the methods and variables.

Now it is your turn. Try the following exercise:

Create a Java class named SquareDemo using your favorite text editor. Be sure you name the file “SquareDemo.java”. You will also need a TestSquareDemo.java for constructing the SquareDemo objects. Add code to the Square demo to construct a square based on the length of its sides. Provide a default no-argument constructor that sets the side length to 1.0. Provide a static class variable for keeping track of how many SquareDemo objects have been constructed. Create methods for calculating the area of the square and perimeter of the square. Create getter and setter methods for all class variables. Test your program by constructing at least 5 SquareDemo objects of different sizes and display the results from each method call.

Exercise 2 – Use the *this* reference

this is a keyword in Java used as a reference for an object to refer to itself. For example, within an object, you can call the object members including fields, constructors and methods. If we defined a variable named radius, using `this.radius` within the class refers to the radius data field. Similarly, using this.getRadius(), or this() would refer to the classes getRadius() method and the default constructor, respectively. Using the `this` reference can provide some code shortcuts and reduce some coding that might be needed.

In the following code example, we will use the `this` reference to simplify the CircleScopeDemo from the previous exercise.

a. Open your favorite text editor and type (or copy and paste) the following Java:

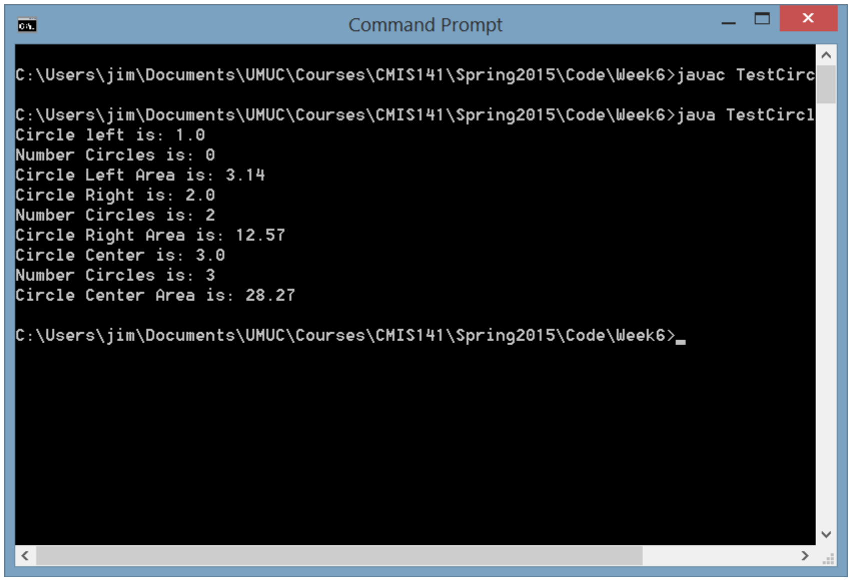
```
/*
* File: CircleWithThis.java
* Author: Dr. Robertson
* Date: January 1, 2015
* Purpose: This program demonstrates the
* use of the this reference
*/

public class CircleWithThis {
 // Define a static class variable
 // Hold number of objects
   private static int numCircles = 0;
 // Define class variable
   private double radius = 1.0;

  // Constructors
  // Default constructor
   public CircleWithThis() {
        this(1.0);
  }
  public CircleWithThis (double r) {
      this.radius = r;
        numCircles++;
  }

  // Area method
  public double getArea() {
     double area = Math.PI*Math.pow(radius,2);
     return area;
  }

  // getter method
  public double getRadius() {
     return this.radius;
  }

  public static int getNumCircles() {
    return numCircles;
  }

}
```

b. Save the file as "CircleWithThis.java" in a location of your choice.
c. To compile the file, type javac CircleWithThis.java at the command prompt.
d. You will need a test class to construct several CircleWithThis objects. Open up your favorite text editor and type the following code:

```
/*
* File: TestCircleWithThis.java
* Author: Dr. Robertson
* Date: January 1, 2015
* Purpose: This program constructs instances
* of the CircleWithThis
*/
```

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

      // Construct a default circle
        CircleWithThis circleLeft = new CircleWithThis();

      // Call the get values
      System.out.println("Circle left is: " +
                 circleLeft.getRadius());
        System.out.println("Number Circles is: " +
                CircleScopeDemo.getNumCircles());
      System.out.println("Circle Left Area is: " +
                String.format("%.2f",circleLeft.getArea()));

      // Construct another Circle
      CircleWithThis circleRight = new CircleWithThis(2.0);
      System.out.println("Circle Right is: " +
               circleRight.getRadius());
        System.out.println("Number Circles is: " +
               CircleWithThis.getNumCircles());
      System.out.println("Circle Right Area is: " +
               String.format("%.2f",circleRight.getArea()));

      // Construct another Circle
      CircleWithThis circleCenter = new CircleWithThis(3.0);
      System.out.println("Circle Center is: " +
                circleCenter.getRadius());
        System.out.println("Number Circles is: " +
                CircleWithThis.getNumCircles());
      System.out.println("Circle Center Area is: " +
                String.format("%.2f",circleCenter.getArea()));

    }
}
```

e. Save the file as TestCircleWithThis.java in the same location where you saved the CircleWithThis.java file. Compile and execute the program to observe the following output:



```
C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code\Week6>javac TestCirc

C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code\Week6>java TestCircl
Circle left is: 1.0
Number Circles is: 0
Circle Left Area is: 3.14
Circle Right is: 2.0
Number Circles is: 2
Circle Right Area is: 12.57
Circle Center is: 3.0
Number Circles is: 3
Circle Center Area is: 28.27

C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code\Week6>_
```

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

1. The CircleScopeDemo and the CircleWithThis classes provide the same functionality. However; the CircleWithThis has a smaller number of lines of code. When we use the this(1.0) constructor call we didn't have to increment the numCircles data field in two locations. Since the default constructor calls the argument with the radius as the input parameter, the numCircles data field will properly incremented.

```
// Default constructor
   public CircleWithThis() {
        this(1.0);
  }
  public CircleWithThis (double r) {
      this.radius = r;
        numCircles++;
  }
```

2. The `this` reference emphasizes we are referring to the members of the CircleWithThis class. This comes in handy in getter and setter methods as it more apparent which data field is being returned or assigned.

```
// getter method
  public double getRadius() {
     return this.radius;
  }
```

If we had added a setter method, the code would look similar to this:

```
// Setter method
  public setRadius(double radius) {
     this.radius = radius;
  }
```

When we use this.radius the code easily differentiates the method input parameter name (radius) from the value it is assigned (this.radius)

3. Finally, notice the outputs from running the TestCircleWithThis and the TestCircleScopeDemo classes are identical.

Now it is your turn. Try the following exercise:

Convert your SquareDemo and TestSquareDemo files you created in the last exercise to take advantage of the `this` reference.

Exercise 3 – Create Overloaded constructors

We have already created some overloaded constructors when you created the default no argument constructor along with another constructor that used one input parameter:

```
// Default constructor
   public CircleWithThis() {
        this(1.0);
  }
  public CircleWithThis (double r) {
      this.radius = r;
        numCircles++;
  }
```

Adding additional constructors is possible and may enhance your class design. For example, if we wanted to construct a Course class, providing maximum students and course name as input parameters we could use the default constructor, and additional constructors as follows:

```
// Default constructor
public Course() {
       // this must be called first
        this(34,"New Course");
  }

  // Full parameterized Constructor
  public Course (int numStudents, String courseName) {
      this.numStudents = numStudents;
        this.courseName = courseName;
        numCourses++;
  }
```

```
        // Partial parameterized Constructor
        public Course (int numStudents) {
              // Number of Students is defined
            this(numStudents,"New Course");
              this.numStudents = numStudents;
        }

        // Partial parameterized Constructor
        public Course (String courseName) {
              this(34,courseName);
              this.courseName = courseName;
        }
```

Although this example is more complicated, the approach is the same as the previous example. We have a default no argument constructor which calls the fully parameterized constructor. Inside the no argument constructor, default values are assigned to the constructor call:

```
        this(34,"New Course");
```

Anytime you use the this() constructor call, it must appear as the first line in the constructor. If we attempted to place a non-comment line prior to the fully parameterized call, we would receive a compile error:

error: call to this must be first statement in constructor

The remaining constructors are partial in that, one parameter is sent as a parameter and the other is assigned inside the constructor itself:

```
        public Course (int numStudents) {
              // Number of Students is defined
            this(numStudents,"New Course");
              this.numStudents = numStudents;
        }
```

In the following code example, we will construct a Course class that has a default no argument constructor a fully parameterized constructor and two partial parameterized constructors.

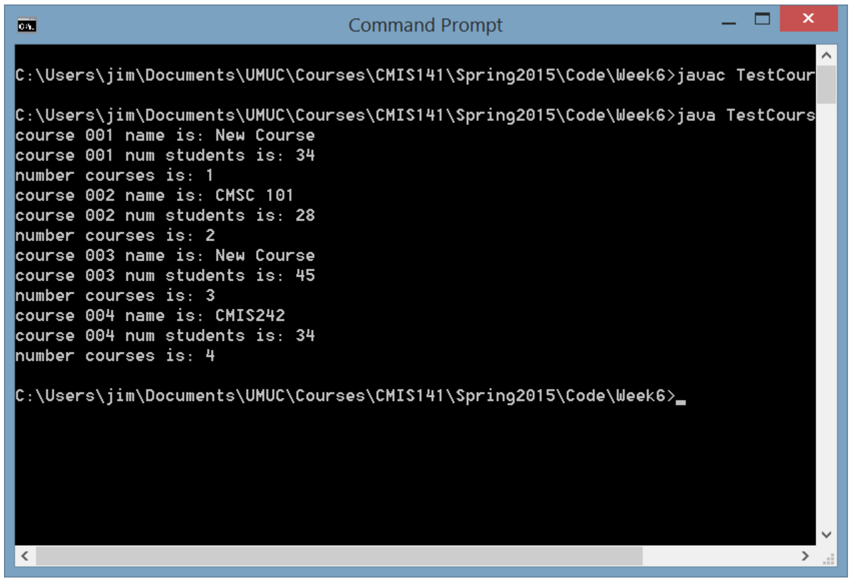
a. Open your favorite text editor and type (or copy and paste) the following Java:

```
/*
* File: Course.java
* Author: Dr. Robertson
* Date: January 1, 2015
* Purpose: This program demonstrates the
* use overloaded constructors
* for a course class in Java
*/

public class Course {
 // Define a static class variable
 // Hold number of objects
   private static int numCourses = 0;
 // Define class variables
   private int numStudents = 34;
   private String courseName = new String("New Course");

  // Constructors
  // Default constructor
   public Course() {
       // this must be called first
        this(34,"New Course");
  }
  // Full parameterized Constructor
  public Course (int numStudents, String courseName) {
   this.numStudents = numStudents;
        this.courseName = courseName;
        numCourses++;
  }
  // Partial parameterized Constructor
  public Course (int numStudents) {
        // Number of Students is defined
   this(numStudents,"New Course");
        this.numStudents = numStudents;
  }

  // Partial parameterized Constructor
  public Course (String courseName) {
        // Number of Students is defined
   this(34,courseName);
        this.courseName = courseName;
  }


  // getter method
  public String getCourseName() {
     return this.courseName;
  }
  public int getNumStudents() {
     return this.numStudents;
  }

  public static int getNumCourses() {
    return numCourses;
  }

}
```

b. Save the file as “Course.java” in a location of your choice.
c. To compile the file, type javac Course.java at the command prompt.
d. You will need a test class to construct several Course objects. Open up your favorite text editor and type the following code:

```
/*
* File: TestCourse.java
* Author: Dr. Robertson
* Date: January 1, 2015
* Purpose: This program constructs instances
* of the Course class
*/

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

   // Construct a default Course
        Course course001 = new Course();

   // Call the get values
   System.out.println("course 001 name is: " +
                course001.getCourseName());
        System.out.println("course 001 num students is: " +
               course001.getNumStudents());
   System.out.println("number courses is: " +
                Course.getNumCourses());

   // Construct a Specific course
        Course course002 = new Course(28,"CMSC 101");

   // Call the get values
   System.out.println("course 002 name is: " +
                course002.getCourseName());
        System.out.println("course 002 num students is: " +
               course002.getNumStudents());
   System.out.println("number courses is: " +
                Course.getNumCourses());

   // Construct a Specific course
       // Using partial constructor
        Course course003 = new Course(45);

   // Call the get values
   System.out.println("course 003 name is: " +
                course003.getCourseName());
        System.out.println("course 003 num students is: " +
               course003.getNumStudents());
   System.out.println("number courses is: " +
                Course.getNumCourses());

   // Construct a Specific course
       // Using partial constructor
        Course course004 = new Course("CMIS242");

   // Call the get values
   System.out.println("course 004 name is: " +
                course004.getCourseName());
        System.out.println("course 004 num students is: " +
               course004.getNumStudents());
   System.out.println("number courses is: " +
                Course.getNumCourses());

    }

}
```

f. Save the file as TestCourse.java in the same location where you saved the Course.java file. Compile and execute the program to observe the following output:



```
C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code\Week6>javac TestCour

C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code\Week6>java TestCours
course 001 name is: New Course
course 001 num students is: 34
number courses is: 1
course 002 name is: CMSC 101
course 002 num students is: 28
number courses is: 2
course 003 name is: New Course
course 003 num students is: 45
number courses is: 3
course 004 name is: CMIS242
course 004 num students is: 34
number courses is: 4

C:\Users\jim\Documents\UMUC\Courses\CMIS141\Spring2015\Code\Week6>_
```

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

1. The partial and no argument constructors use the values provided inside the constructor to assign the missing value(s). Be sure to experiment with the code to set different default values.
2. Try to generate a compile error by introducing a non-comment line prior to the this() call in the no argument or partial parameterized constructor. For example, consider this code for generating a compile error:

```
// Partial parameterized Constructor
  public Course (String courseName) {
        // Number of Students is defined
        String myCourseName = "SDEV300";
      this(34,mycourseName);
        this.courseName = courseName;
  }
```

Now it is your turn. Try the following exercise:

Create a Circle2D and a TestCircle2D class. The Circle2D will contain data fields for the x and y center positions and the radius of the circle. These fields should be of double data type. The default values of these fields should be 0.0, 0.0 and 1.0, respectively. Define a static int data field to keep track of the total number of circles. Constructors should include a default no argument constructor, a fully parameterized constructor (using x-center, y-center and radius parameters) and 2 additional partial constructors using appropriate parameters. Provide getter and setter methods for the x-center, y-center and radius data fields. In addition, provide a method to calculate and return the area of the circle and a method to calculate and return the circumference of the circle. The TestCircle2D class should be used to test at least 5 different Circle2D instances**. Hint: Try to use arrays and loops to minimize the lines of code.**

Exercise 4 – Use an IDE to compile and run an application consisting of more than one class

An IDE can be used to provide a powerful graphical user interface making creating and debugging of Java code much simpler. Most professional programmers will use an IDE to develop their code. In this course, we introduce an IDE and provide a few of the useful features. It is not expected you will learn all of the bells and whistles of the IDE for this course. However; you will become comfortable with compiling multiple classes, formatting code, using packages, automatically importing packages and debugging. You will pick up more features as you continue to use an IDE.

Popular professional IDE's include Netbeans, Eclipse and others. The following provides general instructions on how to use Netbeans for compiling, running and debugging Java code. Be sure to read the manufacturer instructions on how to install the software on your system.

The instructions on how to install Netbeans are currently found here:

https://netbeans.org/community/releases/80/install.html

A brief summary of the installation process is:
1. Download Netbeans Java SE for your platform (Windows, Mac, Linux)

2. Install by double clicking on the downloaded file.

3. Accept permissions following installation Wizard instructions.

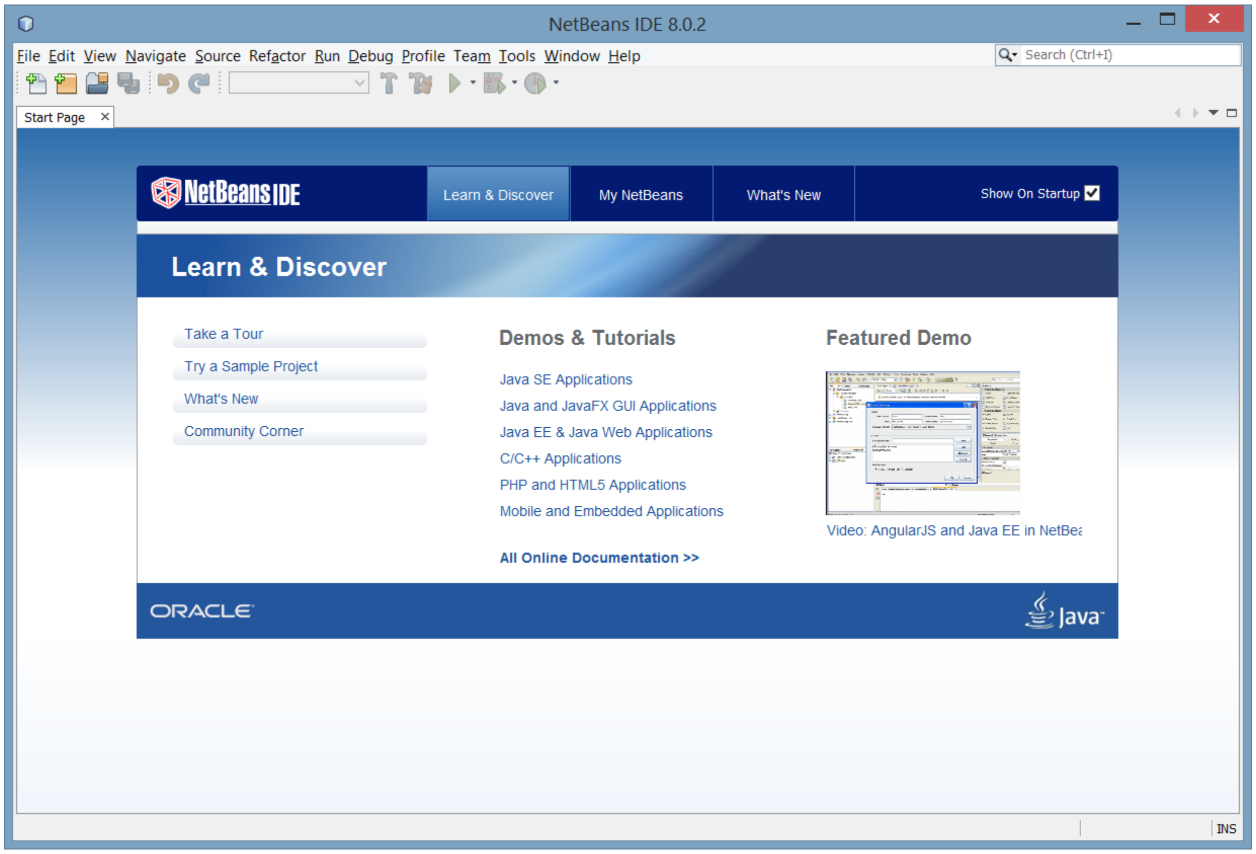
4. During the installation you will need to accept licenses. Using the default directories will work in most cases.

5. The installation process takes 5-10 minutes install on most modern systems.

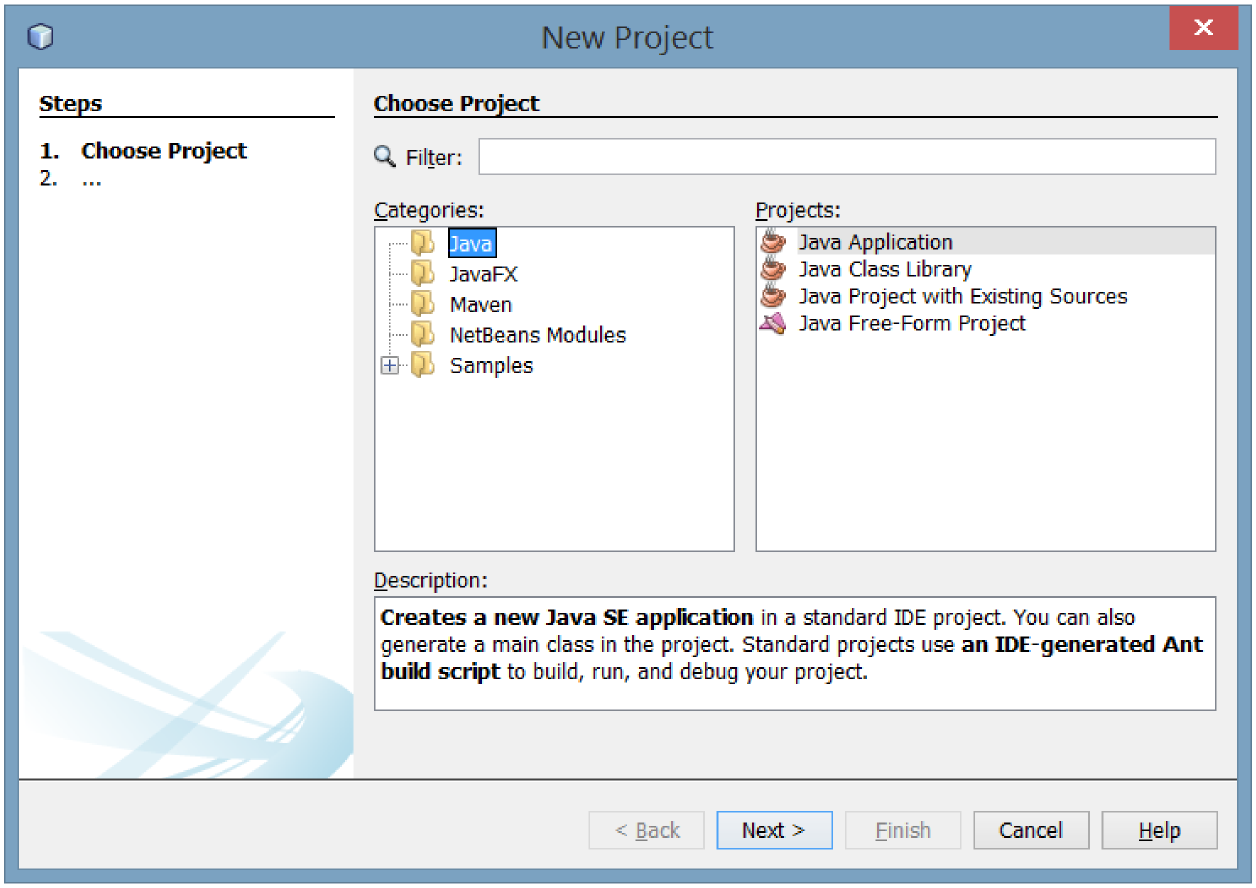
You only need the Java SE installation options. PHP/C and Java EE are not required for this course.

You are welcome to install another IDE but be aware the notes in the classroom provide Netbeans specific screen shots and directions. If you use another IDE, you will need to work through the user's guide and other online materials to prepare you for completing assignments.

a. After you have successfully downloaded and installed Netbeans, launch Netbeans by double clicking the icon on your desktop.


NetBeans IDE 8.0.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
Start Page
NetBeans IDE
Learn & Discover
My NetBeans
What's New
Show On Startup
Learn & Discover
Take a Tour
Try a Sample Project
What's New
Community Corner
Demos & Tutorials
Java SE Applications
Java and JavaFX GUI Applications
Java EE & Java Web Applications
C/C++ Applications
PHP and HTML5 Applications
Mobile and Embedded Applications
All Online Documentation >>
Featured Demo
Video: AngularJS and Java EE in NetBea
ORACLE
Java
INS

b. To create a new project, select File -> New Project from the menu.


New Project

Steps

1. **Choose Project**
2. ...

Choose Project

Filter:

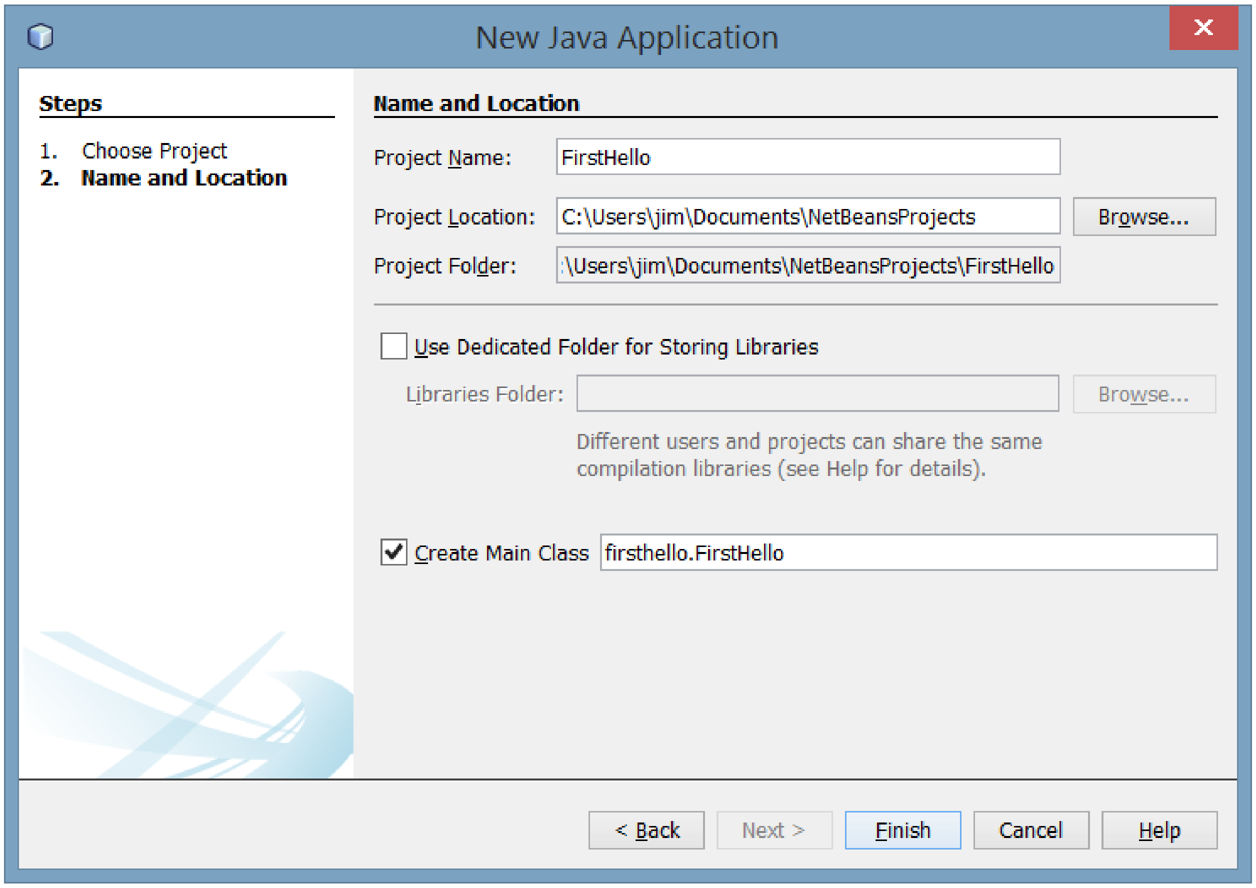
Categories:
- Java
- JavaFX
- Maven
- NetBeans Modules
- Samples

Projects:
- Java Application
- Java Class Library
- Java Project with Existing Sources
- Java Free-Form Project

Description:

Creates a new Java SE application in a standard IDE project. You can also generate a main class in the project. Standard projects use **an IDE-generated Ant build script** to build, run, and debug your project.

c. We will create a Java application. So highlight the Java Application and then click Next.
d. Name the project FirstHello. (Using the default folder locations is acceptable)


New Java Application

Steps

1. Choose Project
2. **Name and Location**

Name and Location

Project Name: FirstHello

Project Location: C:\Users\jim\Documents\NetBeansProjects

Project Folder: :\Users\jim\Documents\NetBeansProjects\FirstHello

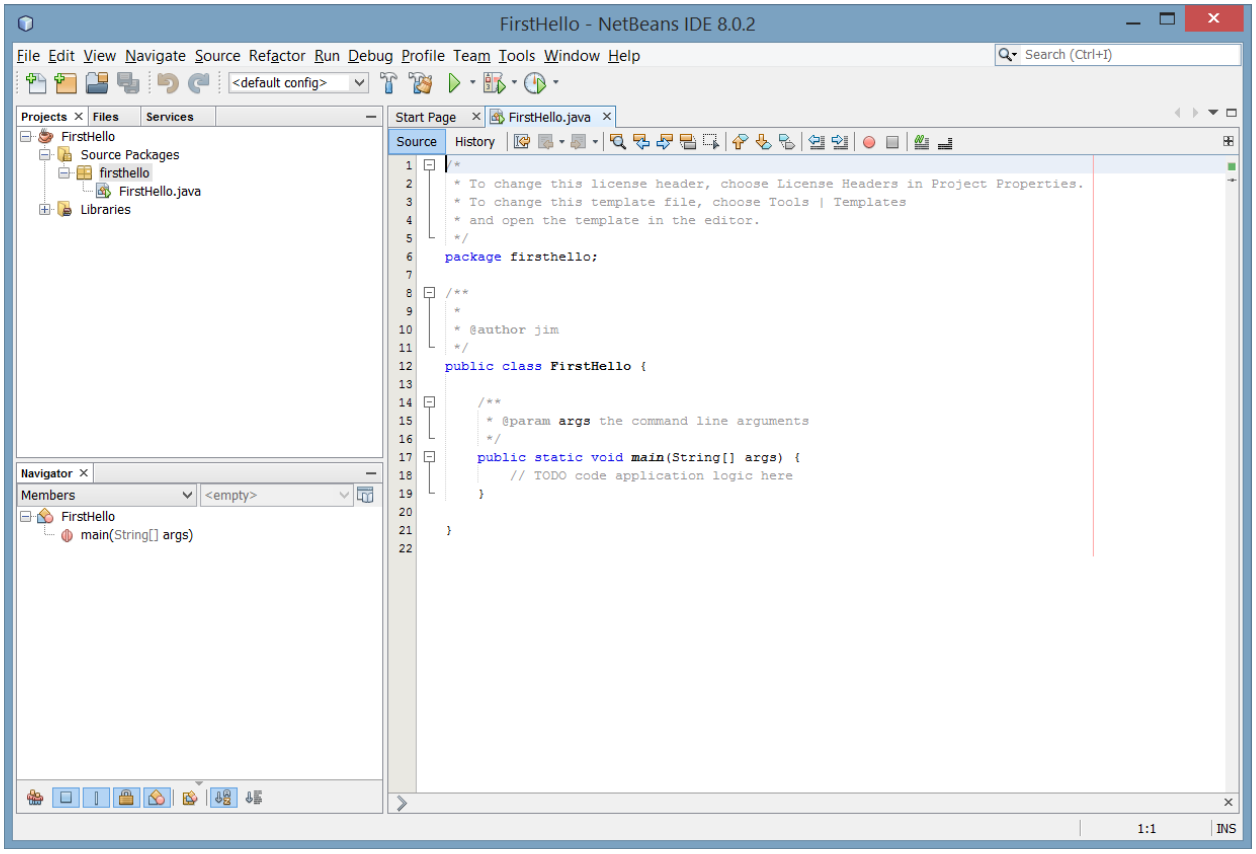
☐ Use Dedicated Folder for Storing Libraries

Libraries Folder:

Different users and projects can share the same compilation libraries (see Help for details).

☑ Create Main Class firsthello.FirstHello

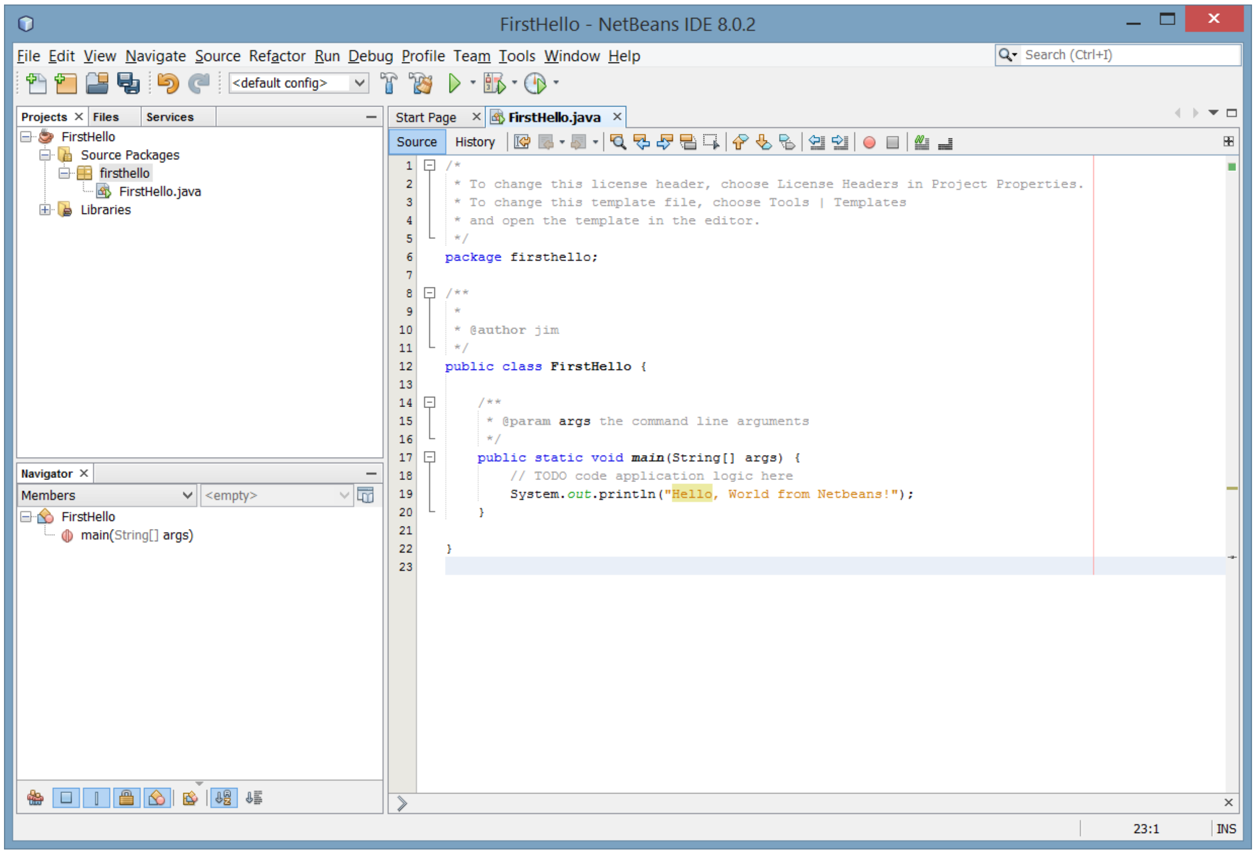
e. Click Finish.



```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package firsthello;

/**
 *
 * @author jim
 */
public class FirstHello {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
    }

}
```

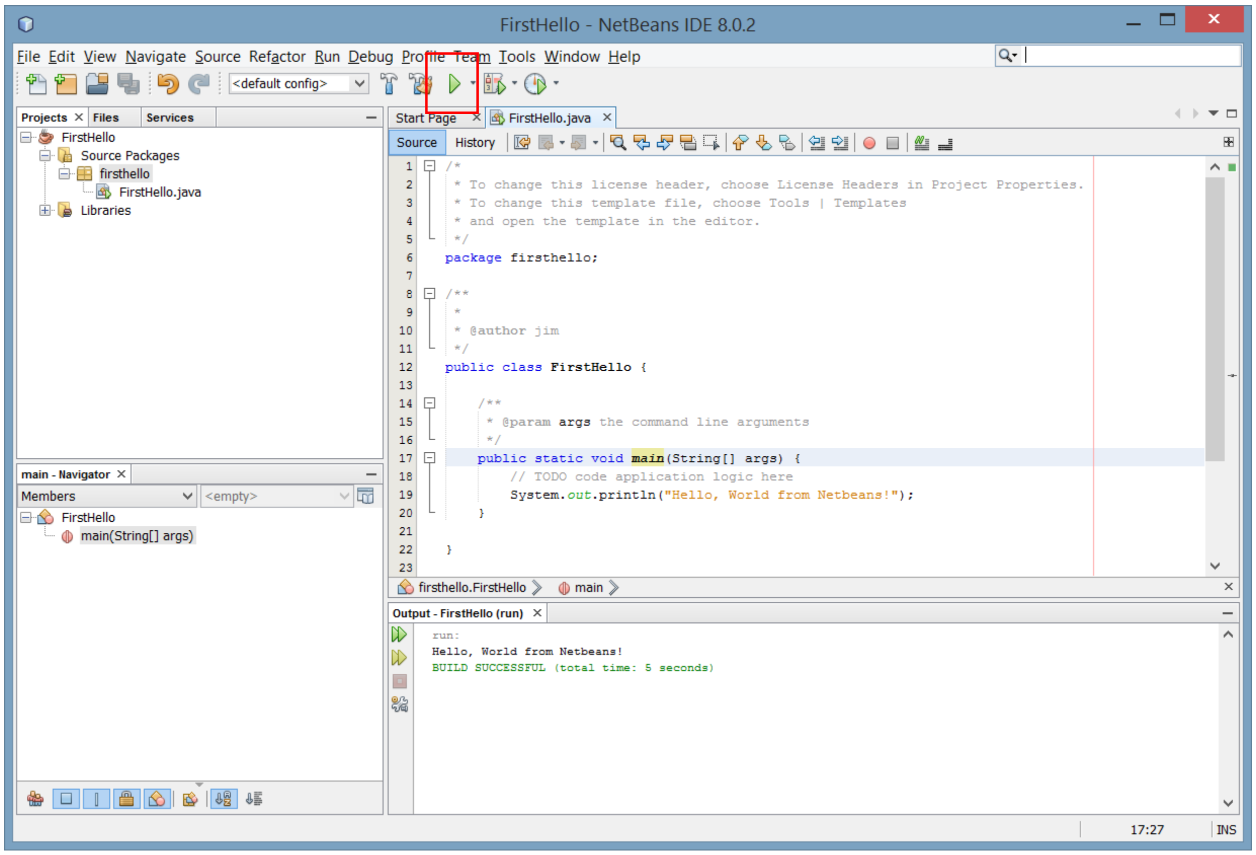
f. The wizard will automatically create a file named FirstHello that includes the main() method. Also, notice the FirstHello, is placed in the firstHello package. Packages are used to group similar classes.

g. Add one line of code to the main() method to print a Hello World greeting.



```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package firsthello;

/**
 *
 * @author jim
 */
public class FirstHello {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        System.out.println("Hello, World from Netbeans!");
    }

}
```

h. To compile and execute the code, click the green arrow near the top of the page.



```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package firsthello;

/**
 *
 * @author jim
 */
public class FirstHello {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        System.out.println("Hello, World from Netbeans!");
    }

}
```

```
run:
Hello, World from Netbeans!
BUILD SUCCESSFUL (total time: 5 seconds)
```

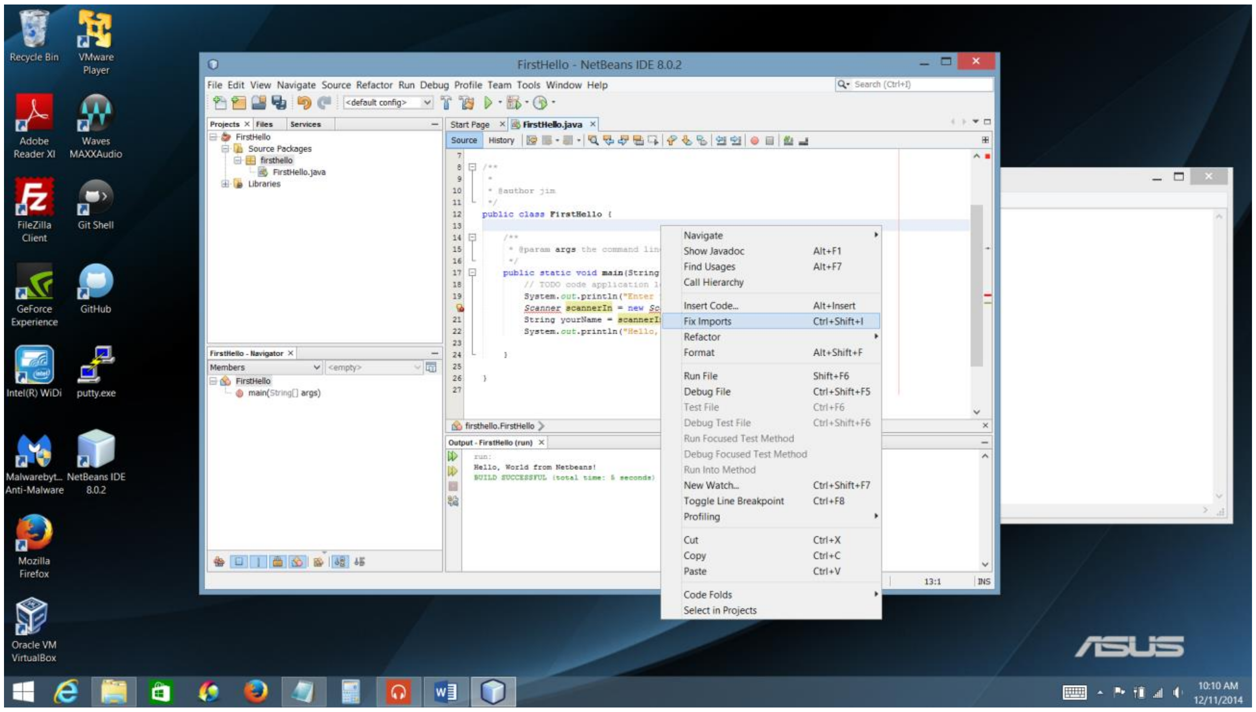
The IDE will compile and run the code and display the results in the output window near the bottom of the screen. Take note of the Projects listed on the left side of the IDE. As you create additional projects, the new projects will be added to the listing. The source packages list your package structure along with any files you have added.

i. Add additional code to request the user enter their name. Modify the output to include the user's name.

```
System.out.println("Enter your name: ");
Scanner scannerIn = new Scanner(System.in);
String yourName = scannerIn.next();
System.out.println("Hello, " + yourName + " from Netbeans!");
```

As you enter these lines in Netbeans, notice the IDE begins to help you the coding by providing logical code to continue your coding. Most professional IDE's have this feature. If you start a quote ("), the IDE, may add the end quote for you (""). The IDE also, looks up possible methods and fields for your statements. For example, when I typed in "System." , the IDE provided available options from the Java API that would work with System. This is referred to as intellisense and can save you significant time when creating projects from scratch.

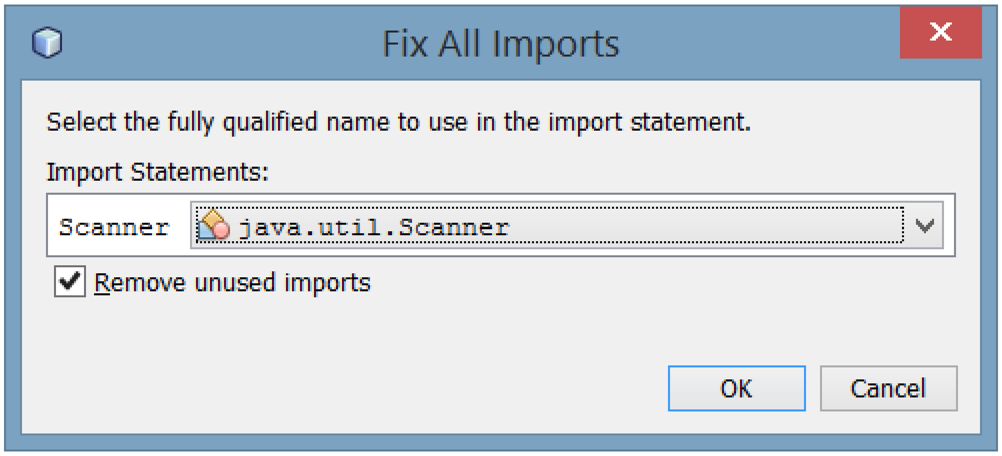
Another feature in Netbeans that you should start using right away is the "Fix imports" feature. After you entered the `Scanner scannerIn = new Scanner(System.in)` line of code, you probably noticed a yellow bulb with a red circle appear on the left side of the code for that line. The IDE is telling you there is an issue with this line of code. The problem with the code is we haven't imported the Scanner class yet. The IDE will quickly fix this for us. Click your right mouse anywhere in the coding window and select "Fix Imports".



```
/**
 *
 * @author jim
 */
public class FirstHello {

    /**
     * @param args the command lin
     */
    public static void main(String
        // TODO code application l
        System.out.println("Enter
        Scanner scannerIn = new Sc
        String yourName = scannerI
        System.out.println("Hello,

    }

}
```

```
run:
Hello, World from Netbeans!
BUILD SUCCESSFUL (total time: 5 seconds)
```

In most cases, the IDE will do an excellent job suggesting which class is needed to import. Select the scanner class and click OK.



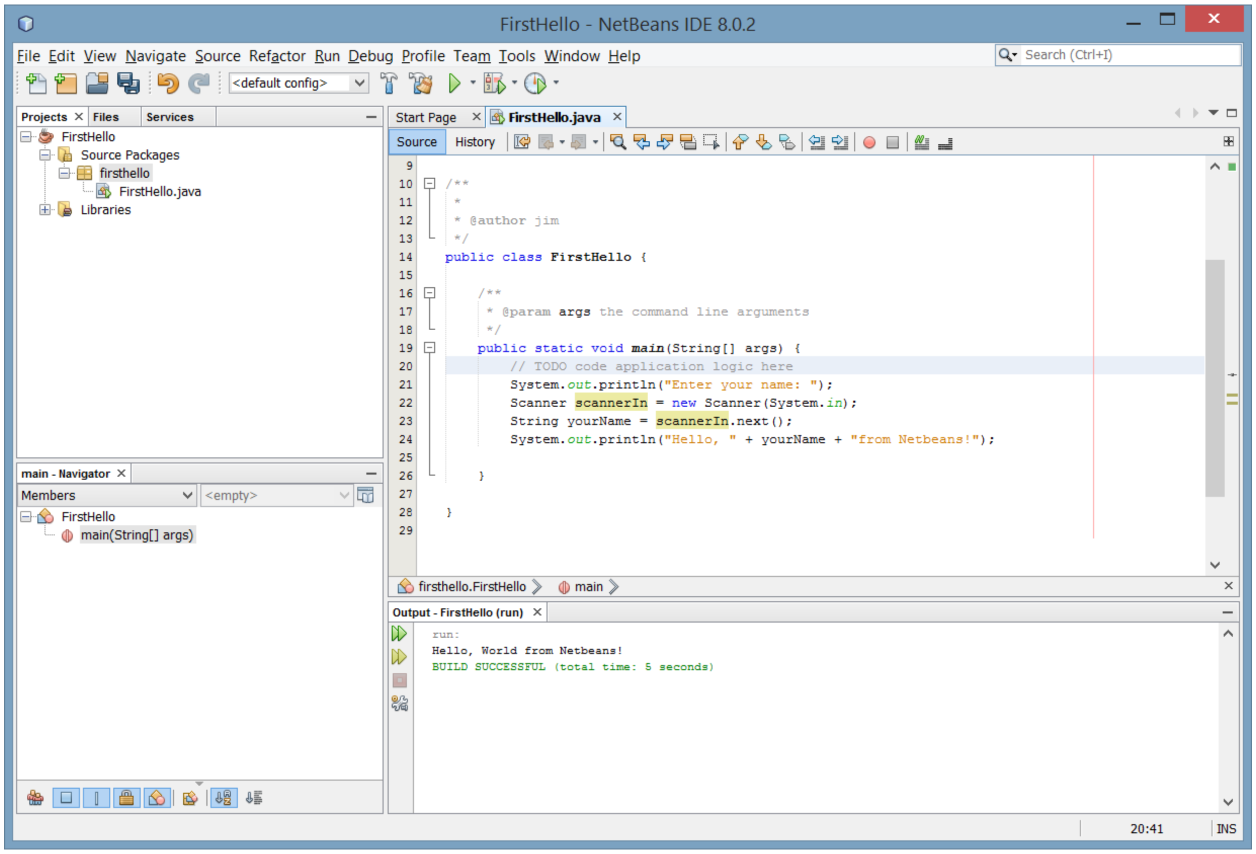
Select the fully qualified name to use in the import statement.

Import Statements:

Scanner java.util.Scanner

[✓] Remove unused imports

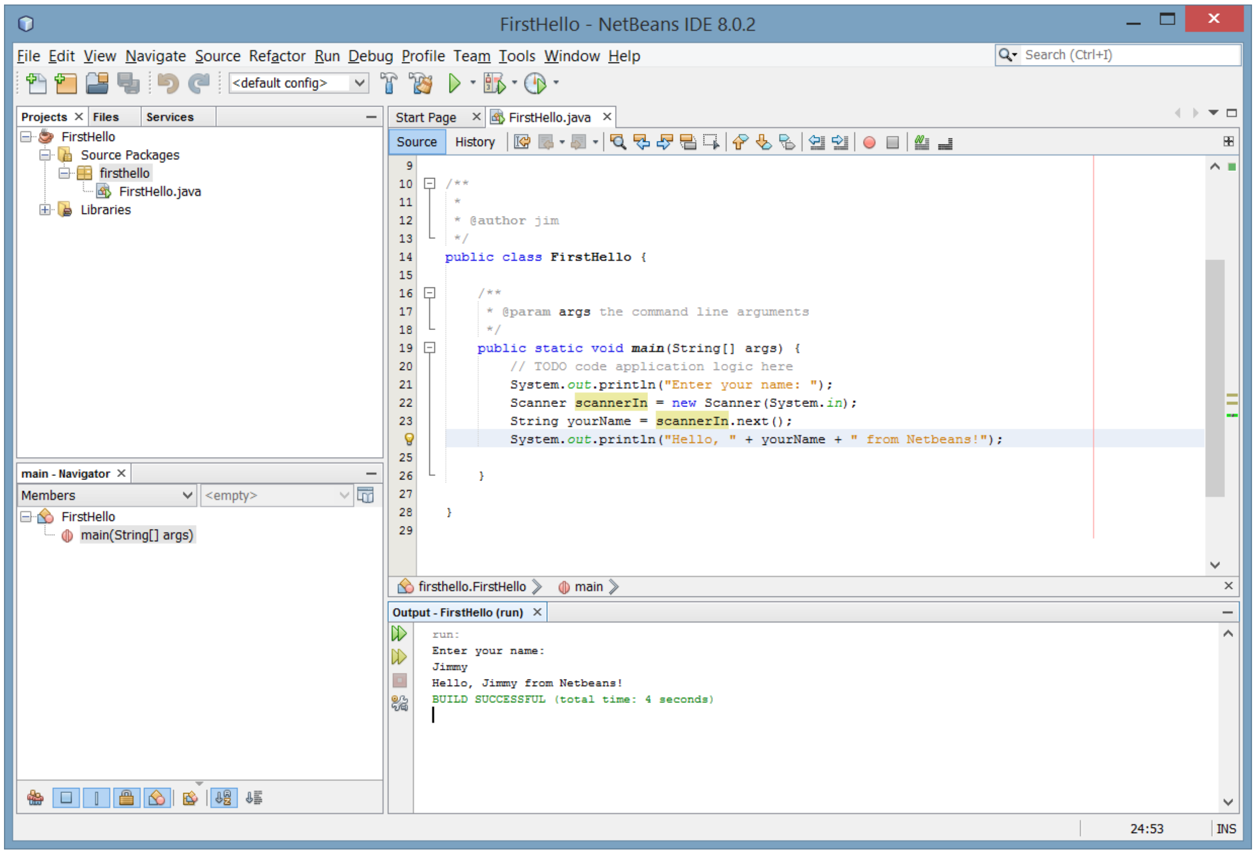
Notice the yellow bulb has disappeared.



```
/**
 *
 * @author jim
 */
public class FirstHello {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        System.out.println("Enter your name: ");
        Scanner scannerIn = new Scanner(System.in);
        String yourName = scannerIn.next();
        System.out.println("Hello, " + yourName + "from Netbeans!");

    }

}
```

```
run:
Hello, World from Netbeans!
BUILD SUCCESSFUL (total time: 5 seconds)
```

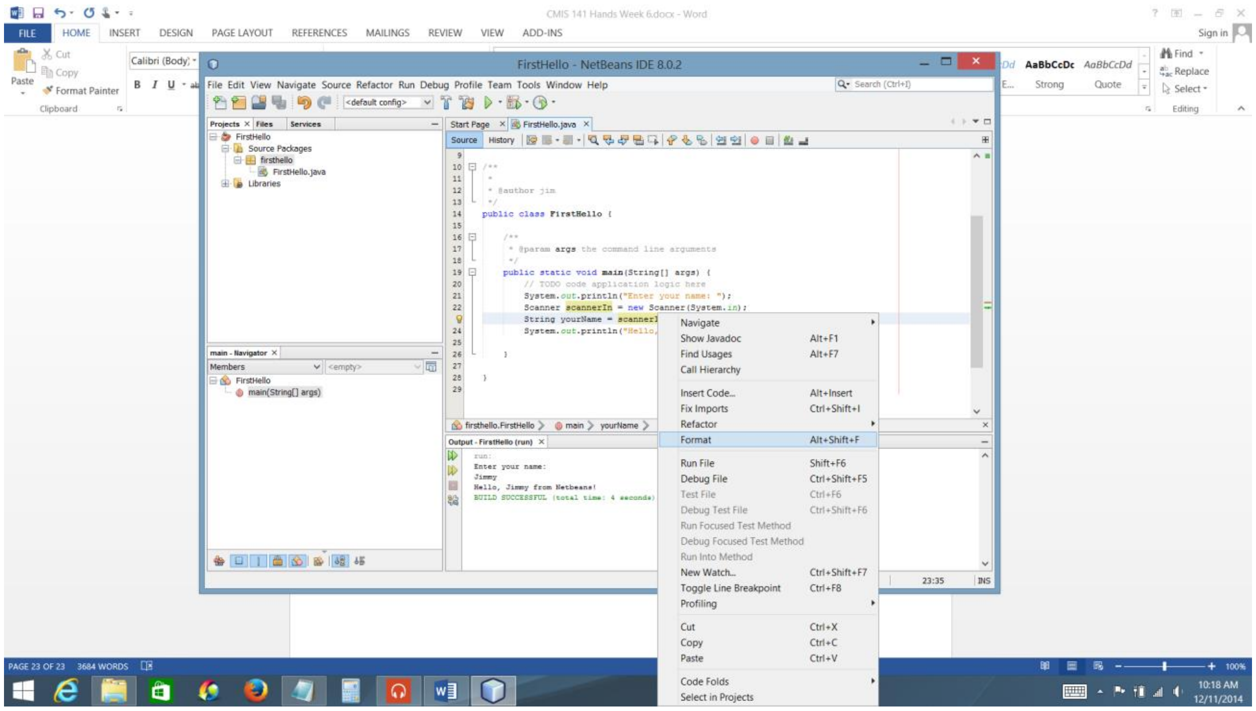
j. To compile and run the code, click the green arrow. When prompted, enter your firstname.



```
/**
 *
 * @author jim
 */
public class FirstHello {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        System.out.println("Enter your name: ");
        Scanner scannerIn = new Scanner(System.in);
        String yourName = scannerIn.next();
        System.out.println("Hello, " + yourName + " from Netbeans!");

    }

}
```

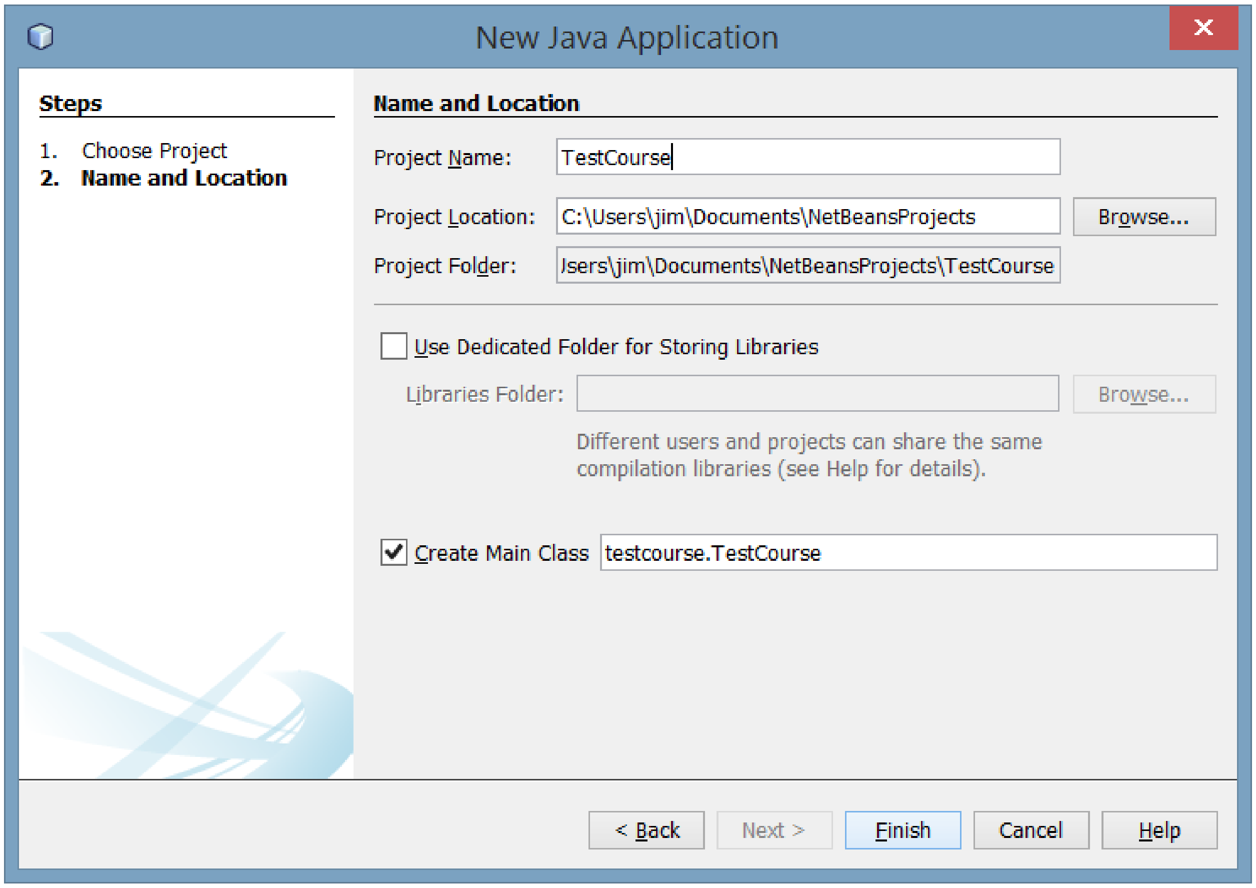
```
run:
Enter your name:
Jimmy
Hello, Jimmy from Netbeans!
BUILD SUCCESSFUL (total time: 4 seconds)
```

k. Let us explore one more feature in the IDE. TO format code, click on the right mouse and select format.



The Format feature cleans up indenting issues and other formatting issues with your Java code.

l. Let us create another project to demonstrate how to compile and run multiple files within the IDE. Use the File -> New Project menu to create a new project name TestCourse.



Project Name: TestCourse

Project Location: C:\Users\jim\Documents\NetBeansProjects

Project Folder: Jsers\jim\Documents\NetBeansProjects\TestCourse

☐ Use Dedicated Folder for Storing Libraries

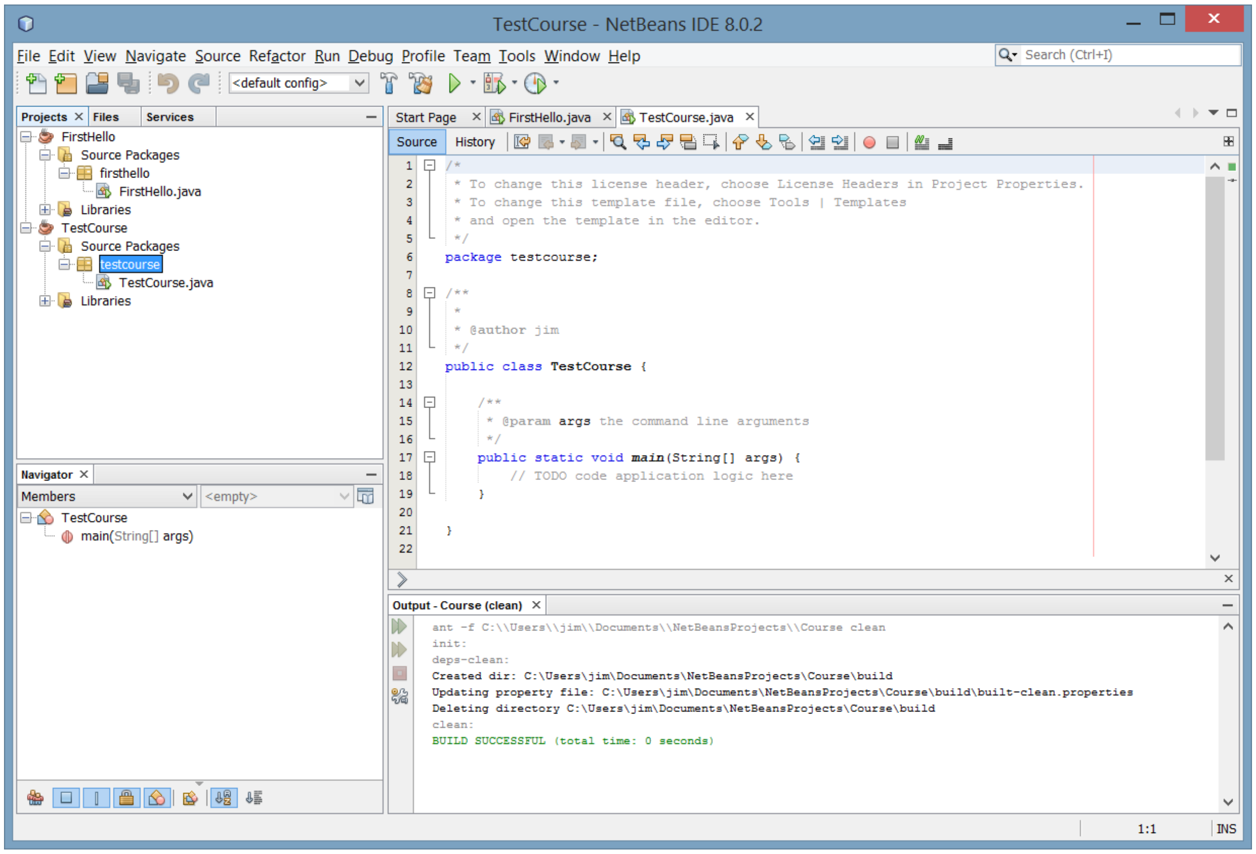
Libraries Folder:

Different users and projects can share the same compilation libraries (see Help for details).

☑ Create Main Class testcourse.TestCourse

Click Finish to continue.

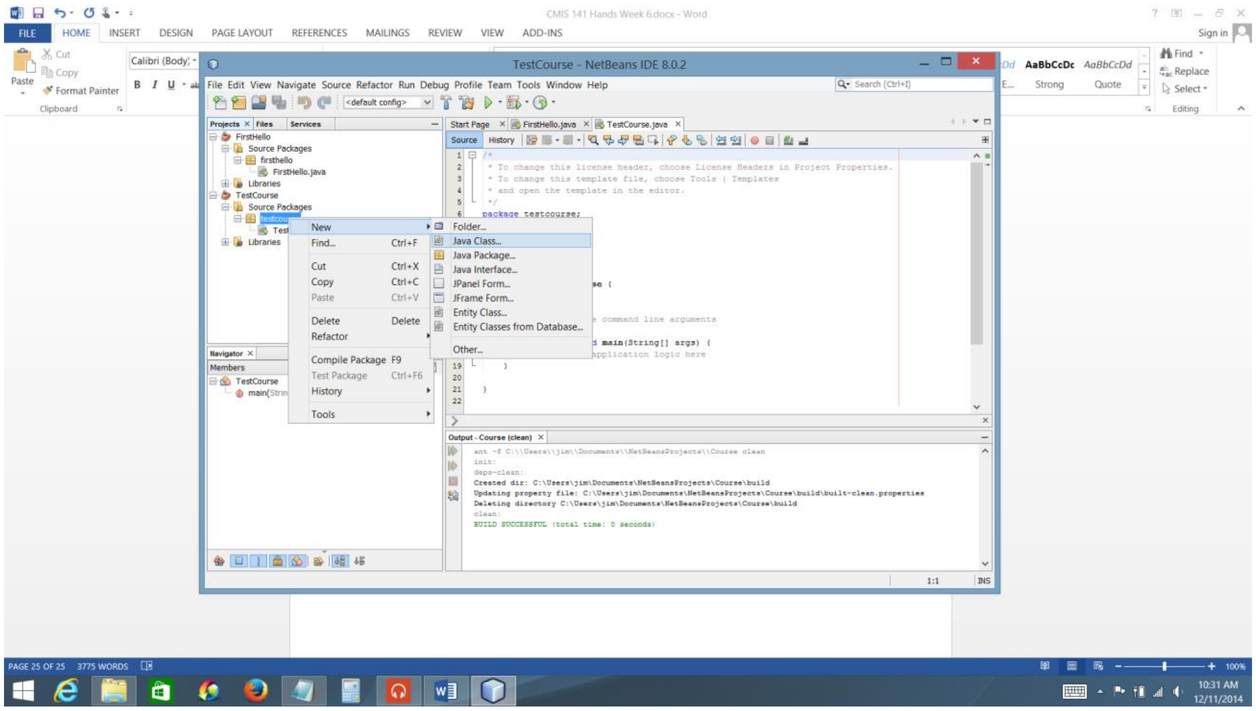
m. The TestCourse project and associated java file will appear.



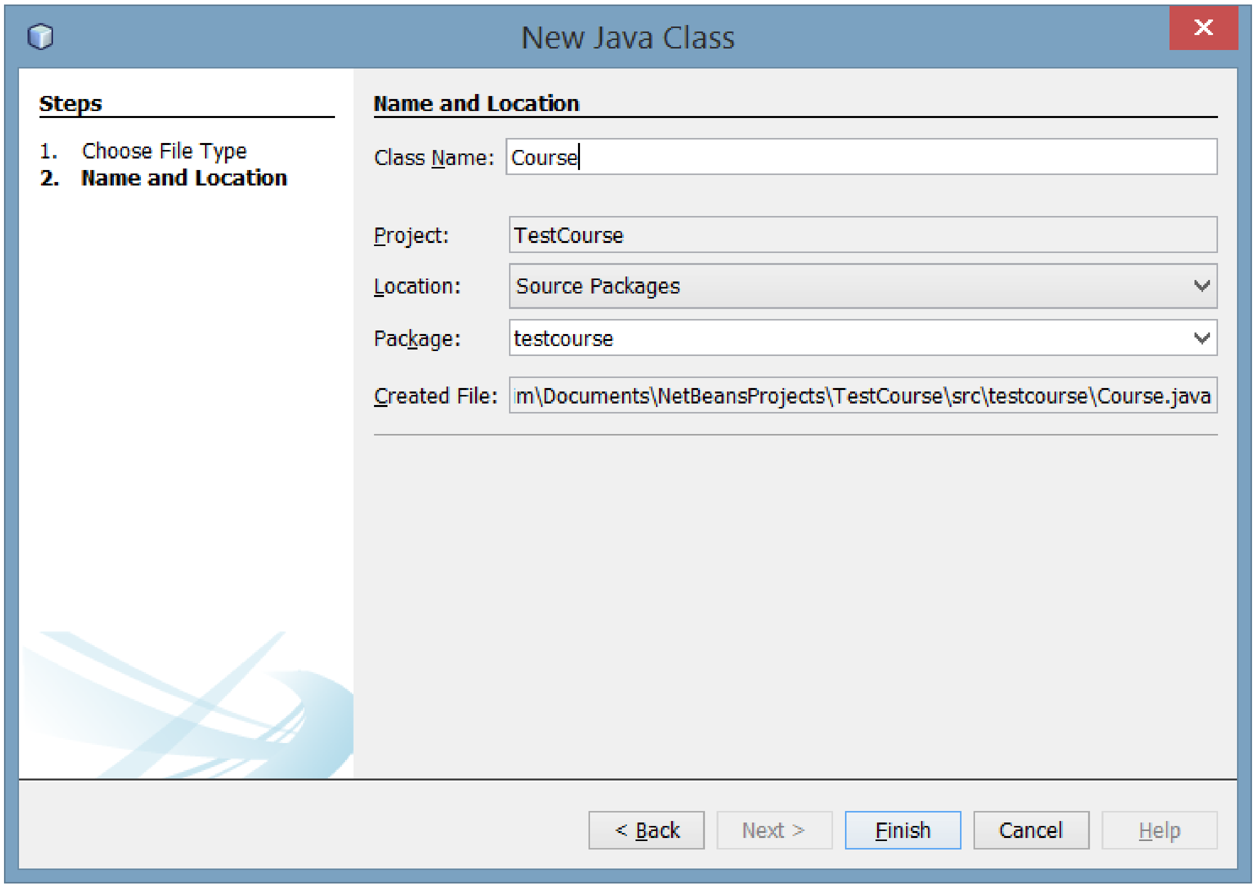
```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package testcourse;

/**
 *
 * @author jim
 */
public class TestCourse {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
    }

}
```

```
ant -f C:\\Users\\jim\\Documents\\NetBeansProjects\\Course clean
init:
deps-clean:
Created dir: C:\Users\jim\Documents\NetBeansProjects\Course\build
Updating property file: C:\Users\jim\Documents\NetBeansProjects\Course\build\built-clean.properties
Deleting directory C:\Users\jim\Documents\NetBeansProjects\Course\build
clean:
BUILD SUCCESSFUL (total time: 0 seconds)
```

n. To add the Course.java file, select the testCourse package on the left side of the page, then right mouse click and select New -> Java Class.



o. Name the class Course.



Class Name: Course

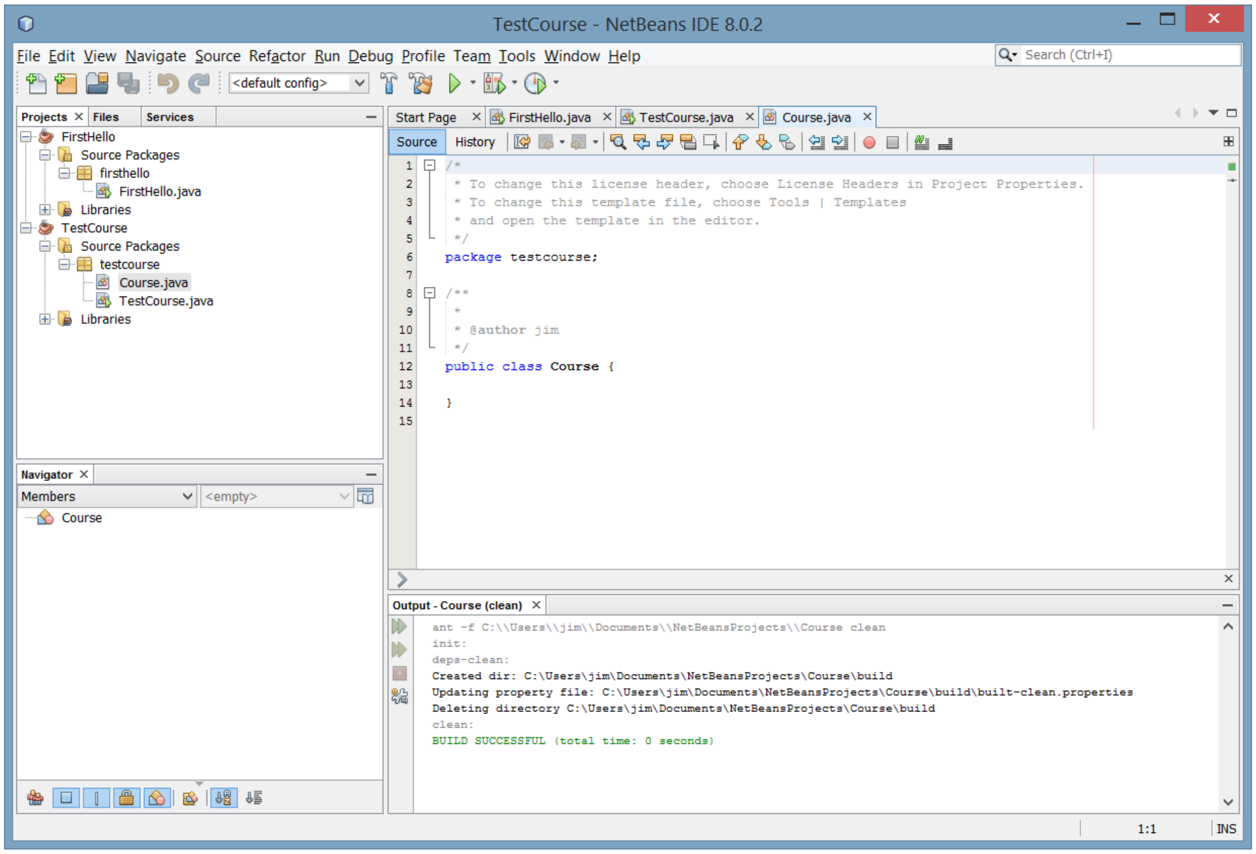
Project: TestCourse

Location: Source Packages

Package: testcourse

Created File: m\Documents\NetBeansProjects\TestCourse\src\testcourse\Course.java

p. Click Finish to edit the Course.java file

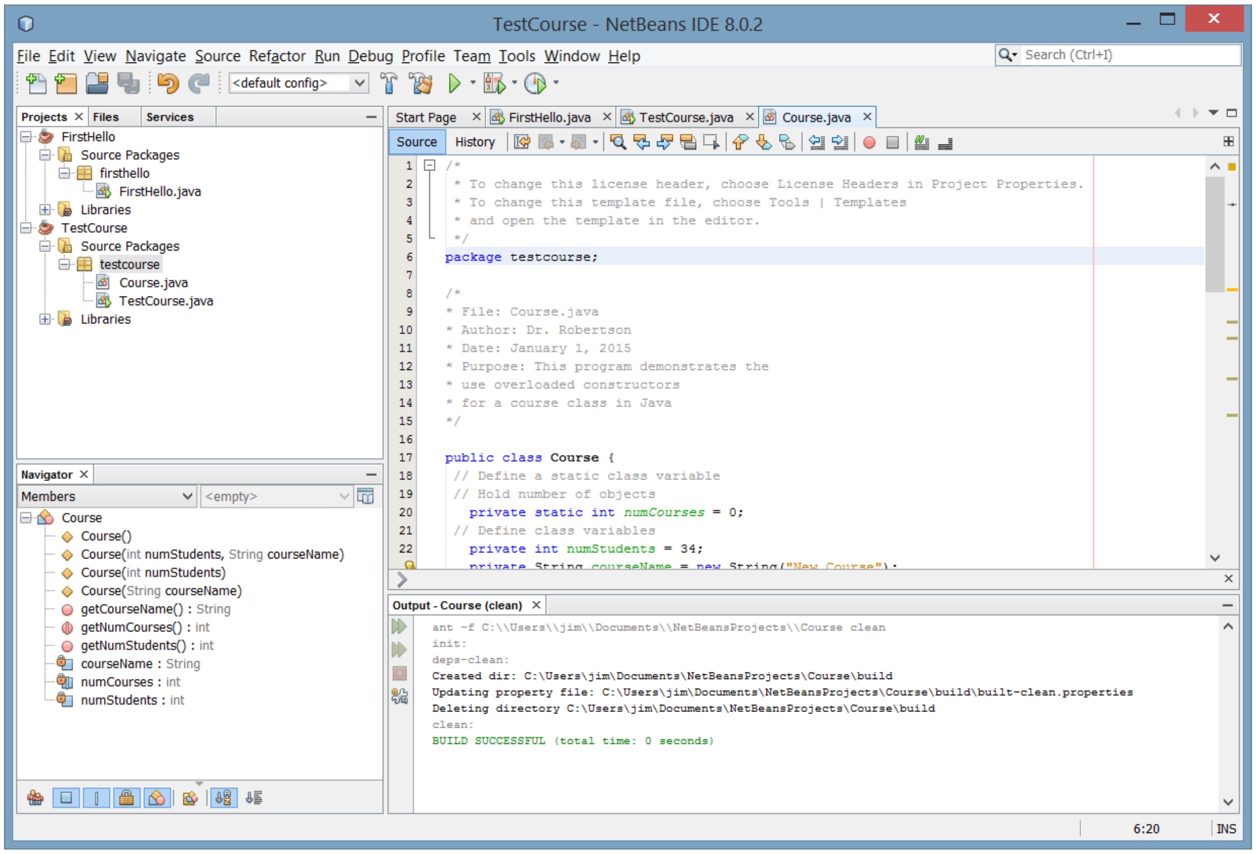


```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package testcourse;

/**
 *
 * @author jim
 */
public class Course {

}
```

```
ant -f C:\\Users\\jim\\Documents\\NetBeansProjects\\Course clean
init:
deps-clean:
Created dir: C:\Users\jim\Documents\NetBeansProjects\Course\build
Updating property file: C:\Users\jim\Documents\NetBeansProjects\Course\build\built-clean.properties
Deleting directory C:\Users\jim\Documents\NetBeansProjects\Course\build
clean:
BUILD SUCCESSFUL (total time: 0 seconds)
```

q. Copy and paste the following Java code into the Course.java file below the package name replacing everything except the package definition.

```
/*
* File: Course.java
* Author: Dr. Robertson
* Date: January 1, 2015
* Purpose: This program demonstrates the
* use overloaded constructors
* for a course class in Java
*/

public class Course {
 // Define a static class variable
 // Hold number of objects
   private static int numCourses = 0;
 // Define class variables
   private int numStudents = 34;
   private String courseName = new String("New Course");

  // Constructors
  // Default constructor
   public Course() {
       // this must be called first
        this(34,"New Course");
  }
  // Full parameterized Constructor
  public Course (int numStudents, String courseName) {
   this.numStudents = numStudents;
        this.courseName = courseName;
        numCourses++;
  }
  // Partial parameterized Constructor
  public Course (int numStudents) {
        // Number of Students is defined
   this(numStudents,"New Course");
        this.numStudents = numStudents;
  }

  // Partial parameterized Constructor
  public Course (String courseName) {
        // Number of Students is defined
   this(34,"SDEV300");
        this.courseName = courseName;
  }


  // getter method
  public String getCourseName() {
     return this.courseName;
  }
  public int getNumStudents() {
     return this.numStudents;
  }

  public static int getNumCourses() {
    return numCourses;
  }

}
```



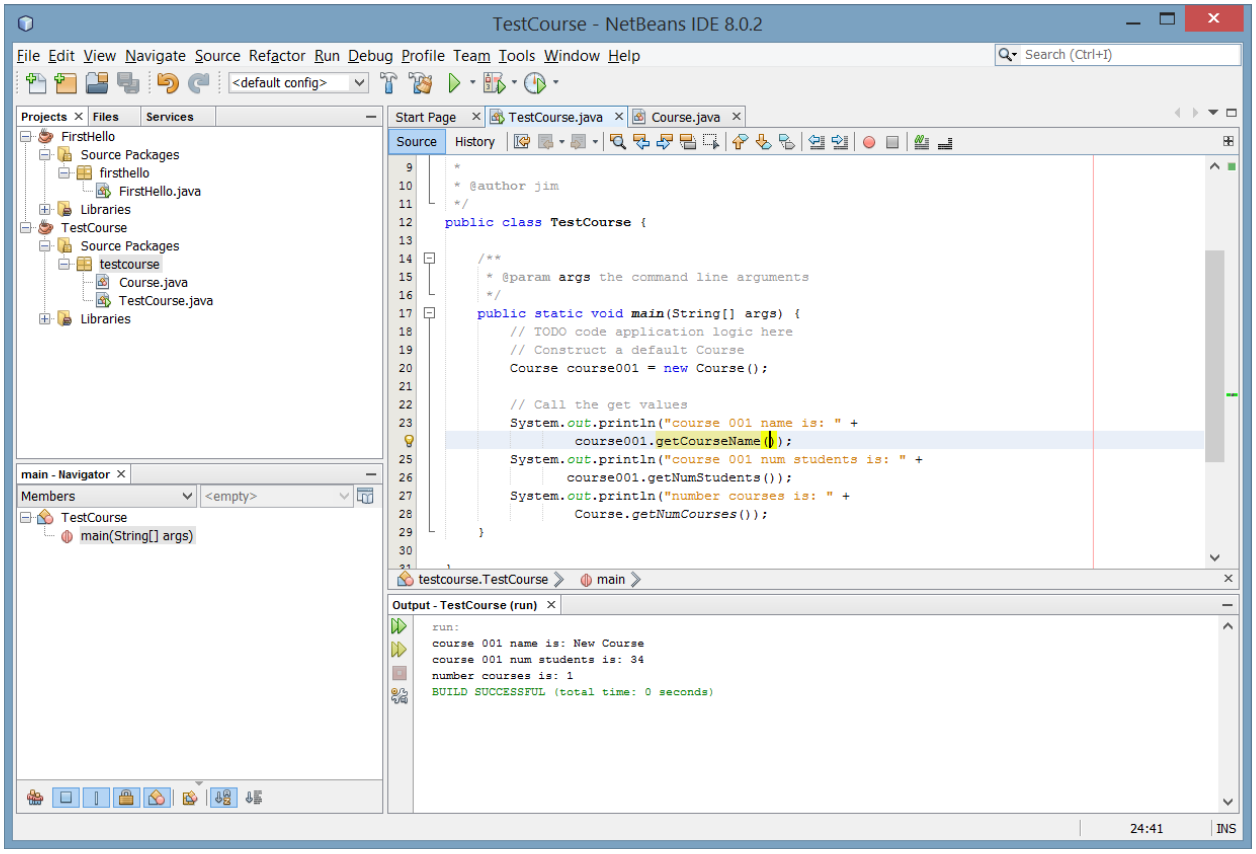
```java
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package testcourse;

/*
* File: Course.java
* Author: Dr. Robertson
* Date: January 1, 2015
* Purpose: This program demonstrates the
* use overloaded constructors
* for a course class in Java
*/

public class Course {
 // Define a static class variable
 // Hold number of objects
   private static int numCourses = 0;
 // Define class variables
   private int numStudents = 34;
   private String courseName = new String("New Course");
```

```
ant -f C:\\Users\\jim\\Documents\\NetBeansProjects\\Course clean
init:
deps-clean:
Created dir: C:\Users\jim\Documents\NetBeansProjects\Course\build
Updating property file: C:\Users\jim\Documents\NetBeansProjects\Course\build\built-clean.properties
Deleting directory C:\Users\jim\Documents\NetBeansProjects\Course\build
clean:
BUILD SUCCESSFUL (total time: 0 seconds)
```

r. Modify the TestCourse.java file to include the following code in the main() method:

```java
// Construct a default Course
       Course course001 = new Course();

    // Call the get values
    System.out.println("course 001 name is: " +
              course001.getCourseName());
      System.out.println("course 001 num students is: " +
             course001.getNumStudents());
    System.out.println("number courses is: " +
              Course.getNumCourses());
```

s. Compile and run the TestCourse by clicking the green arrow.



```
 *
 * @author jim
 */
public class TestCourse {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        // Construct a default Course
        Course course001 = new Course();

        // Call the get values
        System.out.println("course 001 name is: " +
                course001.getCourseName());
        System.out.println("course 001 num students is: " +
               course001.getNumStudents());
        System.out.println("number courses is: " +
                Course.getNumCourses());
    }

```

```
run:
course 001 name is: New Course
course 001 num students is: 34
number courses is: 1
BUILD SUCCESSFUL (total time: 0 seconds)
```

t. You can go back and forth between your projects by clicking on the project you want to edit in the left Projects area.

You should experiment with the Netbeans environment to become comfortable compiling, running, Fixing imports and Formatting code. Be sure to use try creating projects that include more than one class.

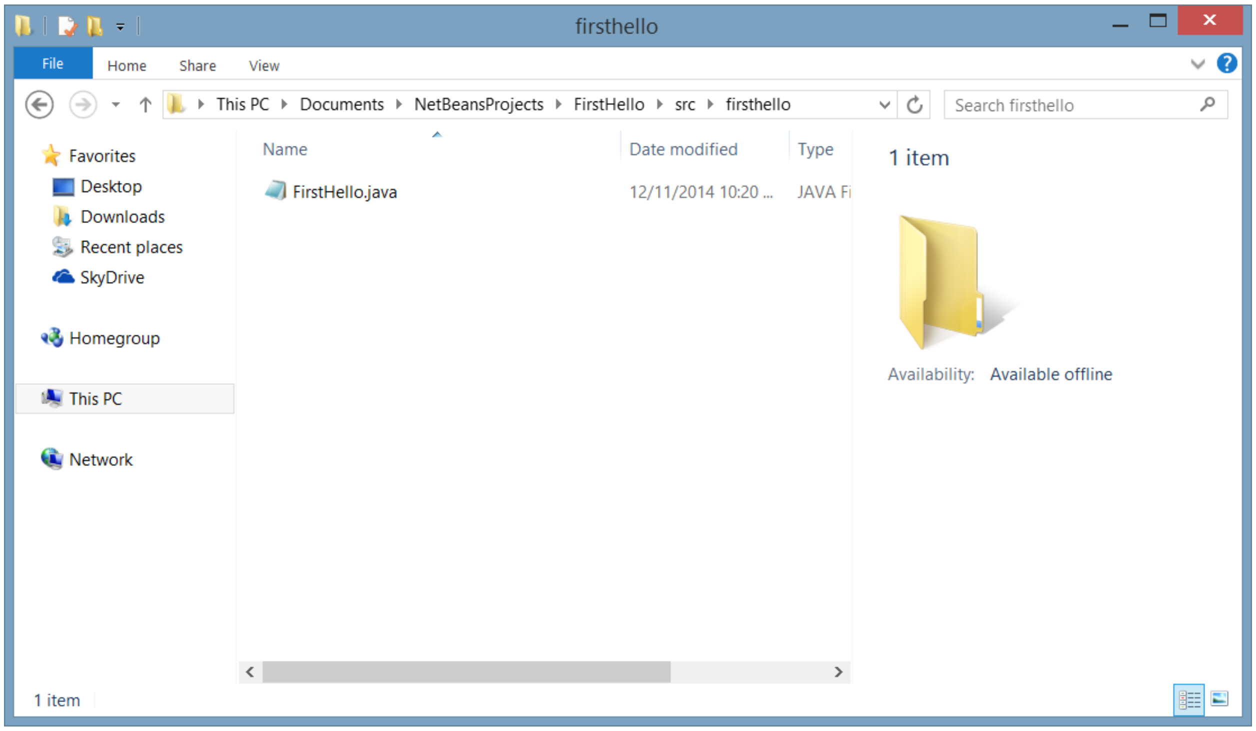
Now it is your turn. Try the following exercise:

Create a Point and a TestPoint class using your installed IDE (e.g, Netbeans). The Point will contain private data fields for x, y positions for each point. Define a static int data field to keep track of the total number of points. Constructors should include a default no argument constructor, a fully parameterized constructor. Provide a method to calculate the distance between two points. The TestPoint class should be used to test at least 3 different Point instances**.**

Finally, you should explore your desktop to determine where the IDE saves your files. Netbeans places you source files in the package folder under the src in your Netbeans projects folders. The location of your Netbeans projects may vary but it is usually in your [home]/Documents folder. The [home] folder is your default folder location for saving files. For a windows machine, this is most likely your account name under the Users folder.

If my Project was named FirstHello, and my package was firstHello, the source files are found in this location:

C:\Users\jim\Documents\NetBeansProjects\FirstHello\src\firsthello



Name	Date modified	Type
FirstHello.java	12/11/2014 10:20 ...	JAVA Fi

Please work to locate your Java source files as you will need to know this to be able to upload your Java source files for your graded projects.