Project 1: Points and Rectangles

In completing this project you will use the *BlueJ* IDE to write a definition for a class that models a rectangle. You will also have the opportunity to learn about Java's graphics capabilities and gain more experience with the static `Math` methods.

This project begins with instructions on how to prepare the *BlueJ* IDE for a new project. The main body of the project then falls into three parts. In Part A you will write the definition for the `APRectangle` class. Parts B and C are optional; they are provided for students who like more of a challenge. In Part B you will write code to draw a rectangle in a graphics window. In Part C you will explore how the rectangle can be rotated.

Preparing BlueJ

1) Start the *BlueJ* IDE. Choose **New Project…** from the **Project** menu. Your teacher may have set up a folder into which all students' Java source code files are to be saved. If so and if this folder does not appear in the *New Project* dialog, then navigate to it in the dialog's tree control in the usual way so that its name appears in the "Look In" selector (see Figure 1).

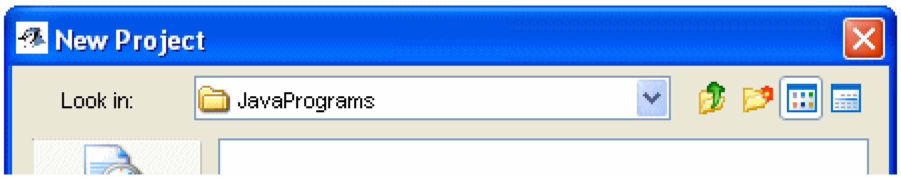


Figure 1: The New Project Dialog.

2) If this is your first project with *BlueJ*, you should create your own personal source code folder that will contain your various project files. Your teacher may assign you your own personal folder name. Click the "Create New Folder" icon — the second icon to the right of the "Look In" selector. A new folder icon appears in the body of the dialog (see Figure 2).

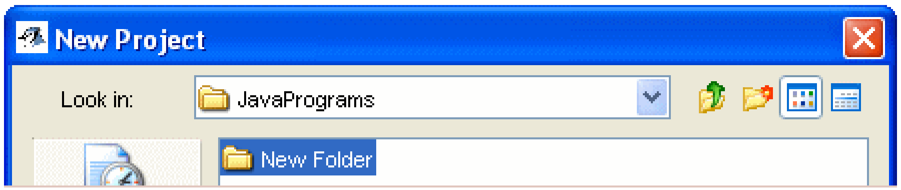


Figure 2: Creating Your Personal Source Code Folder.

3) Click the "New Folder" label once, change it to your personal folder name, and press the **Enter** key. Double-click the folder icon to the left of your new personal folder's name to transfer that name into the "Look In" selector. Enter the new project's name, "Project1" (without the quote marks), into the *File name* box near the bottom of the *New Project* dialog, and click the **Create** button. The *New Project* dialog goes away and the new project's name appears in the main *BlueJ* application window's title bar (see Figure 3).

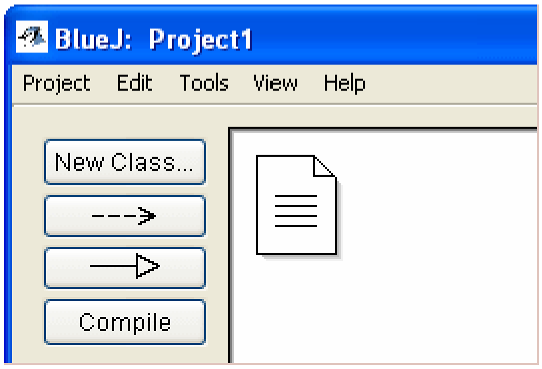


Figure 3: Project1 — Ready To Go!

BlueJ is now set up for you to begin programming.

Part A: Building a Rectangle Class

This is a required part of this project. In this part, you write class definitions for `APPoint` and `APRectangle` and work with the `main` method. Work through each of the numbered items in order.

1) Click the **New Class...** button. Enter "MainClass" (without the quote marks) into the *Class Name* box of the *Create New Class* dialog, leave the *Class Type* selection as "Class", and click the **Ok** button. A colored rectangular icon representing the newly-created `MainClass` class appears in the *BlueJ* window (see Figure 4).

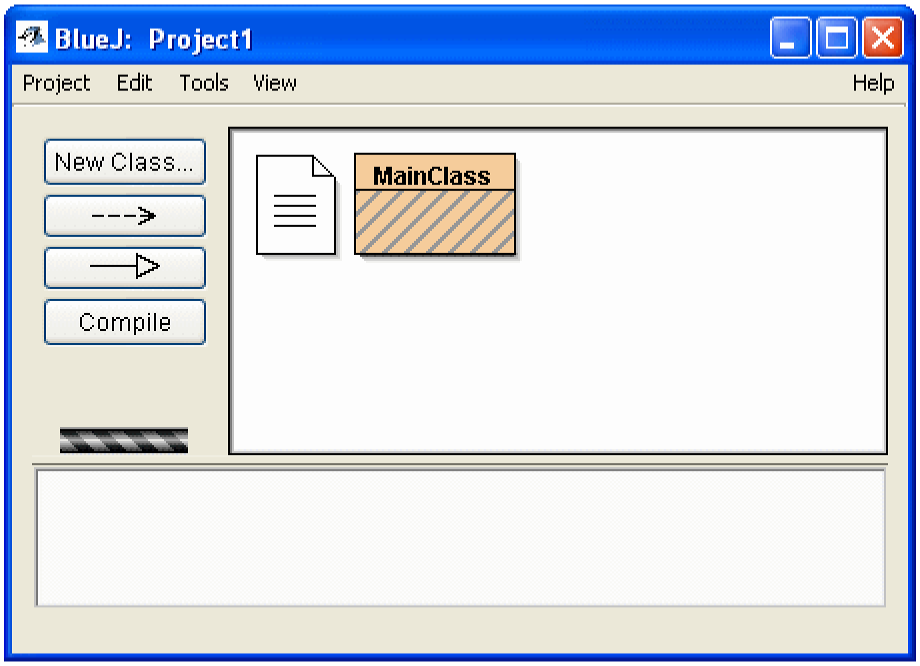
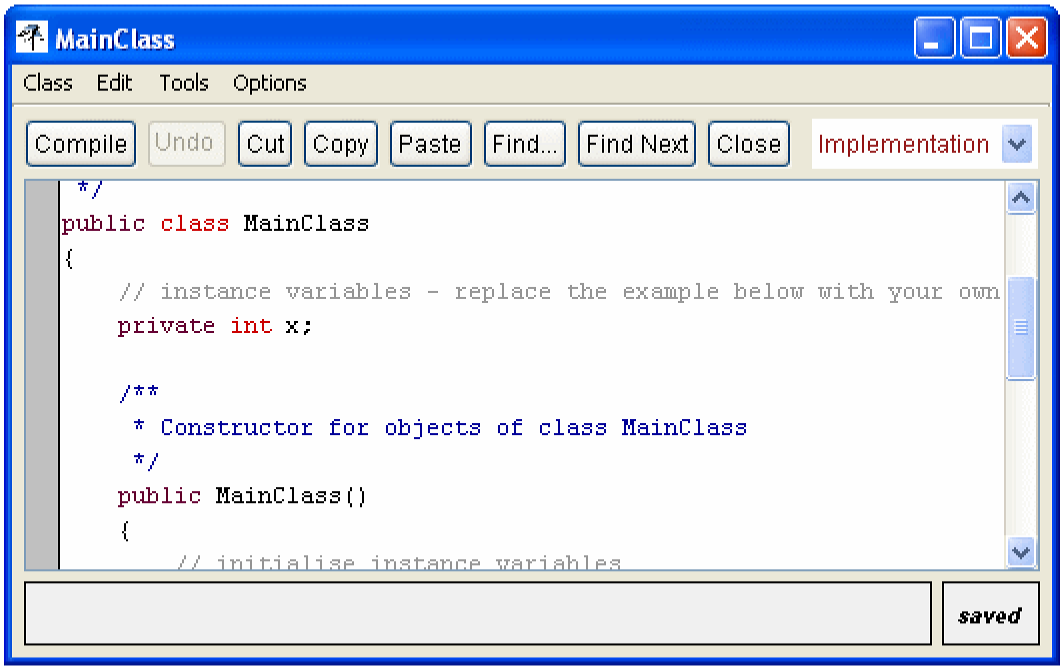


Figure 4: The MainClass icon.

2) Right-click the `MainClass` icon.[1] On the resulting context menu, choose the **Open Editor** item. An editor window containing sample code for the `MainClass` class appears (see Figure 5).



```
 */
public class MainClass
{
    // instance variables - replace the example below with your own
    private int x;

    /**
     * Constructor for objects of class MainClass
     */
    public MainClass()
    {
        // initialise instance variables
```

Figure 5: The MainClass Editor Window.

Edit the sample code in the editor window until it reads as shown in Figure 6:

```
public class MainClass
{
    /**
     * Constructor for objects of class MainClass
     */
    public MainClass()
    {
    }

    /**
     * The main method
     */
    public static void main(String[] args)
    {
        System.out.println( "Done!" );
    }
}
```

Figure 6: Initial MainClass Definition.

[1] Macintosh users should read "right-click" as *ctrl-click*.

Click the **Close** button on the editor window. The changes you have made to the code are automatically saved.

3) Click the **Compile** button on the main *BlueJ* window. After a short pause, the `MainClass` icon should lose its striping, thereby indicating that the class has compiled successfully. If not, the editor window will automatically reopen and an error message will be displayed in the section below the code area. Correct the problem identified by the error message, close the editor window, and try to compile the class again.

4) Once the class has compiled successfully, open the *BlueJ* Terminal by choosing the **Show Terminal** item on the **View** menu. Right-click the `MainClass` icon and then choose the signature — `void main(String[] args)` — of the `main` method on the context menu. Click the **Ok** button on the resulting *Method Call* dialog. The dialog goes away and the exclamation "Done!" appears on the *BlueJ* Terminal.

5) Create a new class called "APPoint" (see Note 1 on the last page of this document). A colored rectangular icon representing the newly-created `APPoint` class appears in the *BlueJ* window. Right-click the `APPoint` icon, choose **Open Editor**, and change the `APPoint` class definition to the following (or copy-and-paste the definition from the exercise on page 4 of the "A Point Class" section of the online course, and then delete the `move` method):

```
public class APPoint
{
    private double myX;
    private double myY;

    public APPoint( double x, double y )
    {
        myX = x;
        myY = y;
    }

    public double getX()
    {
        return myX;
    }

    public void setX( double x )
    {
        myX = x;
    }

    public double getY()
    {
        return myY;
    }

    public void setY( double y )
    {
        myY = y;
    }
}
```

Compile the `APPoint` class (either by clicking the **Compile** button on the `APPoint` class editor window or by right-clicking the `APPoint` class icon in the main *BlueJ* window and choosing **Compile**). Then close the `APPoint` class editor window. To test that the `APPoint` class works, reopen the definition of

`MainClass` (by right-clicking its icon in the main *BlueJ* window and choosing **Open Editor**), insert this definition of the static method `printAPPoint` before the definition of `main`:

```
public static String printAPPoint( APPoint p )
{
    return "(" + p.getX() + "," + p.getY() + ")";
}
```

and change the definition of `main` to:

```
public static void main( String[] args )
{
    APPoint p = new APPoint( 1.0, 2.0 );
    System.out.println( "p is " + printAPPoint( p ) );
    System.out.println( "Done!" );
}
```

Compile and execute the program (see Note 2 on the last page of this document). The *BlueJ* Terminal should display the message

```
p is (1.0,2.0)
Done!
```

If an error message is displayed instead, read the message carefully and make a corresponding fix to your program. Do not proceed to the next step until you have successfully executed your program.

[Notice incidentally that a dashed arrow has appeared in the main *BlueJ* window leading from the `MainClass` icon to the `APPoint` icon. This is a "uses" arrow. It provides a visual reminder to us that the `MainClass` class "uses" the `APPoint` class.]

6) Create a new class called "APRectangle" (see Note 1 on the last page of this document) and edit the `APRectangle` source file so that it contains the following class definition:

```
public class APRectangle
{
    private APPoint    myTopLeft;
    private double     myWidth;
    private double     myHeight;

    public APRectangle( APPoint topLeft, double width, double height )
    {
        myTopLeft = topLeft;
        myWidth = width;
        myHeight = height;
    }
}
```

Add accessor instance methods for the three instance variables, and then click the **Compile** button on the `APRectangle` class editor window to compile your code and check for errors.

Reopen the definition of `MainClass` and insert a definition of the static method `printAPRectangle` after the definition of `printAPPoint`. This method should be defined in such a way that, if it is applied to the `APRectangle` object whose top left corner is the `APPoint` object with coordinate (-5.0,3.6), whose width is 7.5, and whose height is 6.3, then the following string is returned:

```
"[APRectangle (-5.0,3.6) 7.5,6.3]"
```

To achieve this, you will probably find it useful to call upon the `printAPPoint` static method as well as all three of the `APRectangle` class's accessor instance methods. Once you have completed this definition, modify the `main` method so that, when you compile and execute your program, it tests that the program is working correctly.

7) Add each of the following instance methods to the `APRectangle` class. In each case, modify the `main` method to test your definition. You may find it useful to use the *BlueJ* debugger while performing these tests. [For information on using the debugger, refer to the file *Using the BlueJ IDE* that may be downloaded from the online course.]

    a) Add modifier methods for the three instance variables.

    b) Add a `getTopRight` method that returns the `APPoint` object that represents the point at the top right corner of the rectangle. Hint: the `return` statement should be of the form:

    ```
    return new APPoint( … );
    ```

    In the optional parts of this project, you will draw `APRectangle` objects in a window whose x-coordinates increase *to the right*. So the `APPoint` object at the top right corner of an `APRectangle` object should have an x-coordinate that is *greater* than that of the `APPoint` object at its top left corner.

    c) Add `getBottomLeft` and `getBottomRight` methods that return the `APPoint` objects at the bottom left and bottom right corners, respectively. In the optional parts of this project, you will draw `APRectangle` objects in a window whose y-coordinates increase *downward*. So the `APPoint` objects at the bottom two corners of an `APRectangle` object should have y-coordinates that are *greater* than that of the `APPoint` object at its top left corner.

    d) Add an `area` method that returns the area of the rectangle.

    e) Add a `shrink` method that takes a single argument, a `double d`, and that shrinks the rectangle to `d`% of its current size, that is,

    ```
    myWidth *= (d / 100.0);
    myHeight *= (d / 100.0);
    ```

This concludes Part A of this project.

Part B: Drawing the Rectangle

This part of the project is optional. In this part, you construct a stand-alone Java application that will draw your rectangle.

1) Create a new class called "APCanvas" (see Note 1 on the last page of this document). In the `APCanvas` class editor window, replace the class definition with the following, making sure to include the two `import` statements:

```
import java.awt.*;
import javax.swing.*;

public class APCanvas extends JPanel
{
    public APCanvas()
    {
    }

    private void paintMe( Graphics g )
    {
    }

    public void paintComponent( Graphics g )
    {
        super.paintComponent( g );
        paintMe( g );
    }
}
```

(The `import` statements let the Java compiler know where to look for the `Graphics` and `JPanel` classes. The use of `import` statements and the use of the keyword `extends` that appears in the first line of the class definition are explained later in the online course.)

Compile this class by clicking the editor window's **Compile** button.

2) In the `MainClass` class editor window, insert these two `import` statements at the start of the file:

```
import java.awt.*;
import javax.swing.*;
```

Then change the definition of the `main` method so that it reads as follows:

```
public static void main( String[] args )
{
    JFrame frame = new JFrame( "AP Java Test " );
    frame.getContentPane().add( new APCanvas() );
    frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
    frame.pack();
    frame.setSize( 200,224 );
    frame.setLocation( 100, 100 );
    frame.setVisible( true );
}
```

(The reason why we set the height of the `JFrame` to be 24 more than its width is to provide room for the window's title bar.)

Compile and execute your program (see Note 2 on the last page of this document). When you are successful, running this program will produce a blank window — possibly hidden behind other windows — with the caption *AP Java Test* (see Figure 7).

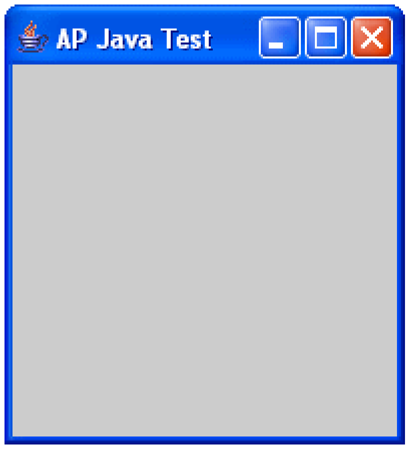


Figure 7: The Stand-alone Drawing Application.

3) Close the blank *AP Java Test* window by clicking the **X** button in its top right corner. Then modify the `paintMe` method of the `APCanvas` class so that the method body consists of this statement:

```
g.drawLine( 0, 0, 200, 200 );
```

Save the modified `APCanvas` class and then compile and execute the program. This time, the *AP Java Test* window that appears has a single solid line drawn from top left to bottom right.

The instance `g` of the `Graphics` class represents a 2-dimensional Cartesian plane with coordinate (0,0) in the top left corner, and with x-coordinates increasing *to the right* and y-coordinates increasing *downward*. The `drawLine` method of the `Graphics` class draws a solid line across this plane from the point specified by the first two arguments to the point specified by the last two arguments. Experiment with `drawLine` until you are thoroughly familiar with its effect.

4) At the start of the `APRectangle` source file, add these two import statements:

```
import java.awt.*;
import javax.swing.*;
```

Then add a new instance method with the signature `draw( Graphics g )` to the `APRectangle` class that draws the rectangle by calling the `drawLine` method of `g` four times:

```
public void draw( Graphics g )
{
    APPoint topLeft = myTopLeft;
    APPoint topRight = getTopRight();
    APPoint bottomLeft = getBottomLeft();
    APPoint bottomRight = getBottomRight();

    g.drawLine( (int)topLeft.getX(), (int)topLeft.getY(),
                (int)topRight.getX(), (int)topRight.getY() );

    // . . .
}
```

Note how the variables `topRight`, `bottomLeft`, and `bottomRight` capture the positions of the vertices before drawing takes place. This avoids having to call the corresponding methods repeatedly, which would make the program less efficient. Notice also that we cast all the coordinates to `int`s before passing them as arguments to the `drawLine` method. We must do this because that method expects its arguments to be `int`s.

Test your code by compiling and running your program. You could, for example, change the body of the `paintMe` method of the `APCanvas` class so that it reads like this:

```
APPoint p = new APPoint( 20.0, 60.0 );
APRectangle r = new APRectangle( p, 65.0, 45.0 );
r.draw( g );
```

This completes Part B of this project.

Part C: Rotating the Rectangle

This part of the project is also optional. In this part, you use `Math` methods to rotate the rectangle.

1) To the class definition for `APRectangle`, add an instance variable `myAngle` of type `double`. In the constructor, set `myAngle` to zero, and provide accessor and modifier instance methods for `myAngle`. The instance variable `myAngle` represents a counterclockwise rotation, measured in radians (see Note 3 on the last page of this document), with the center of rotation at the top left corner of the `APRectangle` instance (see Figure 8).

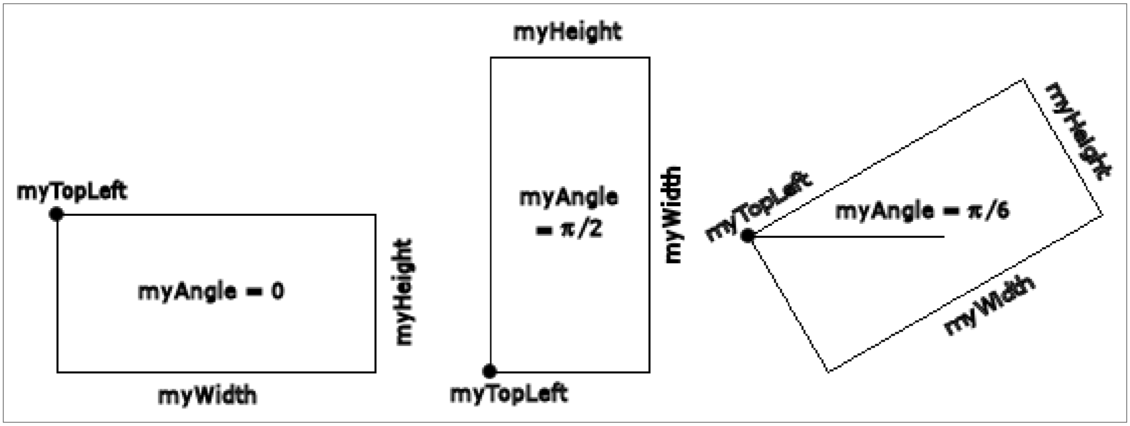


Figure 8: Rotating the Rectangle.

2) To calculate the `APPoint` representing the top right corner we may use the Java methods `Math.sin` and `Math.cos` (see Figure 9) as follows:

```
public APPoint getTopRight()
{
    double x = myWidth * Math.cos( myAngle );
    double y = myWidth * Math.sin( myAngle );

    return new APPoint( myTopLeft.getX() + x, myTopLeft.getY() - y );
}
```

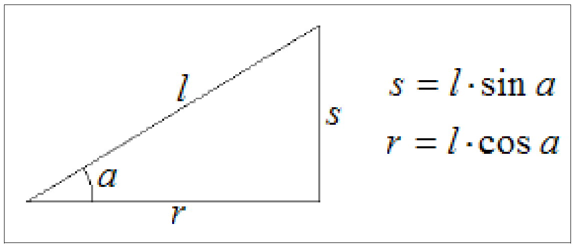


Figure 9: Using *sin* and *cos*.

In the above code, we subtract the value of `y` from the y-coordinate of the `APRectangle` object's top left corner because the top right corner is rotated upward and we are drawing our rectangles in a window whose y-coordinates increase downward.

Once we have the position of the top right corner, we can locate the bottom right corner (see Figure 10) as follows:

```
public APPoint getBottomRight()
{
    APPoint t = getTopRight();

    double x = myHeight * Math.sin( myAngle );
    double y = myHeight * Math.cos( myAngle );

    return new APPoint( t.getX() + x, t.getY() + y );
}
```

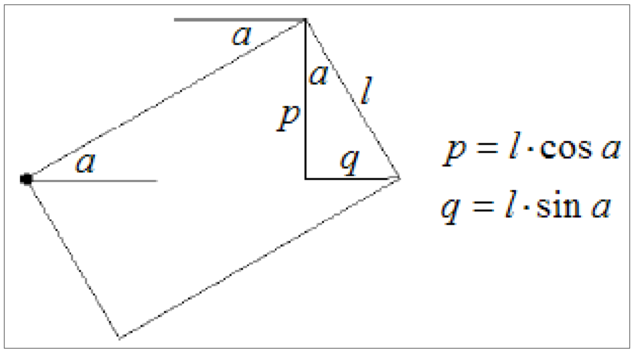


Figure 10: Finding the Bottom Right Corner.

Modify the class definition for `APRectangle` to include these methods, and rewrite the method `getBottomLeft` to take account of the rectangle's rotation.

Rewrite the `paintMe` method of the `APCanvas` class to test that your rectangle draws correctly at a variety of rotations. You may use the built-in Java constant `Math.PI` as follows:

```
APPoint p = new APPoint( 100.0, 100.0 );
APRectangle r = new APRectangle( p, 50.0, 25.0 );
r.setAngle( Math.PI / 6 );
r.draw( g );
```

3) Using a `for` loop and the `shrink` method of the `APRectangle` class, try to reproduce the display shown in Figure 11.

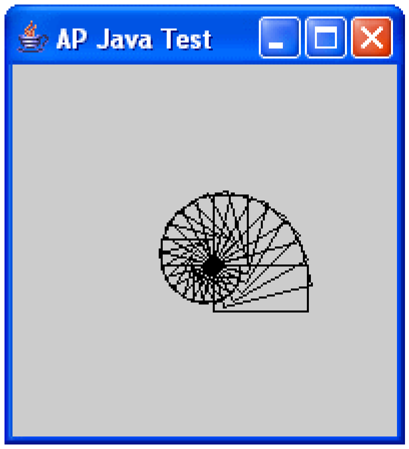


Figure 11: A Seashell!

This completes Part C of this project.

Notes

1) To create a class, follow the procedure used to create the `MainClass` class. That is, click the **New Class...** button. Enter the name of the class into the *Class Name* box of the *Create New Class* dialog (the name you use must contain no spaces or punctuation), leave the *Class Type* selection as "Class", and click the **Ok** button.

2) To compile and execute the program, click the **Compile** button on the main *BlueJ* window. If any class icon retains its striping, right-click the icon and choose **Compile**. If the *BlueJ* Terminal is not open, choose **Show Terminal** on the **View** menu. Finally, right-click the `MainClass` class icon, choose the signature — `void main(String[] args)` — of the `main` method, and then click the **Ok** button on the *Method Call* dialog.

3) All Java trigonometric functions use angles measured in radians. Measured in degrees, a full turn is 360, whereas in radians a full turn is 2π, or about 6.28. That is, 360° = 2π. So, for example, 180° = π, 90° = π/2, 30° = π/6, and so on.