The purpose of this assignment is to draw and manipulate a simple robot on the screen. The code you create could be used in a much larger game/simulation.

You are required to write a complete class and to use several provided classes to build a BlueJ project. The full mark for the assignment is 10, but there are Bonus/Challenge extensions that are worth up to 4 extra marks.

The assignment is to be submitted via ETA before **noon** on **October 15**. Late assignments will be accepted until noon on October 18, but a 20% penalty will be assessed (and no bonus marks for late assignments). Assignments submitted later will *not* be graded.

The class you will write is called `Robot`. It has fields, two constructors, and methods described below. Two of the fields of `Robot` are objects of the `Circle` and `Square` classes respectively. These are classes from the BlueJ `shapes` project accompanying Chapter 1 in the text.

WARNING: You must not modify the classes in the `shapes` project. You will hand in only your `Robot.java` file.

Your class **must** include properly formatted JavaDoc comments. Be sure to test the generation of your documentation.

If you have not yet done so, carefully read the Course Ethics section on the course web page. Note especially the remarks concerning **plagiarism**:

Do not *expect that small changes in a program (such as altering comments, changing variable names, or interchanging statements) will avoid detection. If you cannot do the work yourself, it is extremely unlikely that you will succeed in disguising someone else's work.*

Class Robot

A `Robot` object has a name, a head (circle), a body (square), an x-position and a y-position. The head circle just touches the center of the top side of the body square, and the x- and y-positions refer to the top left-corner of the body square.

Fields

`name` A `String`, for example `"Robbie"` or `"C3PO"`.

`head` A reference to a `Circle` object.

`body` A reference to a `Square` object.

`xPos, yPos` Two `int` values that hold the x- and y-positions of the top-left corner of the robot's body square.

Constructors

There are two constructors for the `Robot` class. The first constructor takes no parameters and does the following: sets the robot's name to `"Marvin"`; creates a `Circle` object referenced by `head`, and changes the circle's diameter to 40 and color to `"green"`; creates a `Square` object referenced by `body`, and changes the square's color to `"blue"` and its size to 50; sets `xPos` to 100 and `yPos` to 80, and moves the square so that its top-left corner is at the screen location $(x, y) = (100, 80)$; moves the circle so that it sits on top of the square exactly as described earlier; makes both the circle and the square visible.

The second constructor takes *five* parameters: the first is a `String` indicating the robot's name, the second is a `String` indicating the color of the head, the third is a `String` indicating the color of the body, and the fourth and fifth are `int` values indicating the initial x- and y-positions of the robot, respectively (again, these refer to the top-left corner of the body square). This constructor does everything the first constructor does, except that the robot's name, colors, and initial positions are set using the parameters.

Note: For the second constructor, if the x- and y-positions passed via the last two parameters would cause *any part* of the robot's body to be off-screen, use the default location $(x, y) = (100, 80)$. You can make use of the `moveTo` method described below.

Methods

The `Robot` class has five methods. The first, called `getName`, is an accessor method that simply returns the `String` holding the name of the robot. The method `moveHorizontal` takes a single `int` parameter (can be positive or negative) and moves the robot horizontally by that amount. Similarly, the method `moveVertical` takes a single `int` parameter and moves the robot vertically by that amount. (Remember that the y-position increases as you move *downward*.) The methods `moveHorizontal` and `moveVertical` do not perform any checks on the input — the move may place the robot off-screen, but that's fine. The method `moveTo` takes two `int` parameters indicating new x- and y-positions for the robot, and returns a `boolean` value. If the new positions would place any part of the robot's body off-screen, don't move the robot, print out a warning message, and return `false`; otherwise, move to the specified location, and return `true`. The last method is `slowMoveHorizontal`. This method takes a single `int` parameter indicating a horizontal distance (positive or negative), and then moves the robot horizontally *one pixel at a time* until either the full distance has been covered *or* the robot hits the right or left wall, whichever happens first. If the robot hits the wall, it should stop when it is exactly flush with the wall.

Note: In your `slowMoveHorizontal` method, *do not* call the `slowMoveHorizontal` methods in the `Circle` and `Square` classes.

Bonus/Challenge Problems

There are two directions you may pursue for extra credit. You may choose to work on only one or a combination of both. If you wish to submit work for bonus marks you should make a new project called `RobotPlus`, starting again from the `shapes` project and containing a new class that you should name `RobotPlus`. Once again, you must not modify the other classes of the `shapes` project.

Bonus marks will be awarded on the basis of both the quality of your implementation (including documentation) and on its creativity. Remember that your class **must** include properly formatted JavaDoc comments.

The first challenge is to enhance the appearance of a `Robot` object. This can be done by adding, for example, arms, legs, facial features, a hat... This will require both that you add new fields and that you modify the constructors and methods of the `Robot` class. Note that you may use the `Triangle` class from the `shapes` project to express some features. It is expected that the two constructors described above will again be available, but you may add new constructors with appropriate parameters. You *must* also ensure that *all* of the methods from the `Robot` class described above are implemented in `RobotPlus`.

The second challenge is to enhance the movements of the robot. There are two suggested methods and you may try others (be sure to fully document all of your methods!). The first method `slowDiagonal` takes takes two parameters, an `int` called `distance` and a `String` called `direction`. The valid values for `direction` are `"NE"`, `"SE"`, `"NW"` and `"SW"`. The method should move the robot slowly in the indicated direction on the canvas by a diagonal distance of `distance` pixels but leave the robot at the wall if the distance would take it off-screen. The second suggested method is called `diagonalBounce`. It takes the same parameters as `slowDiagonal`, but this time if the robot hits the wall it should reflect from the wall (for example hitting the wall in a `"SE"` direction means the robot will continue in the `"SW"` direction).

Submitting Your Assignment

You should place your results in a zipped file called `usrnmA1.zip` (containing only `Robot.java` or both `Robot.java` and `RobotPlus.java`) and submit it using ETA.

Don't forget to include proper headers and Javadoc-style comments.