CS 273 Laboratory 9: Methods and Classes

This lab gives you practice creating a simple Java class.

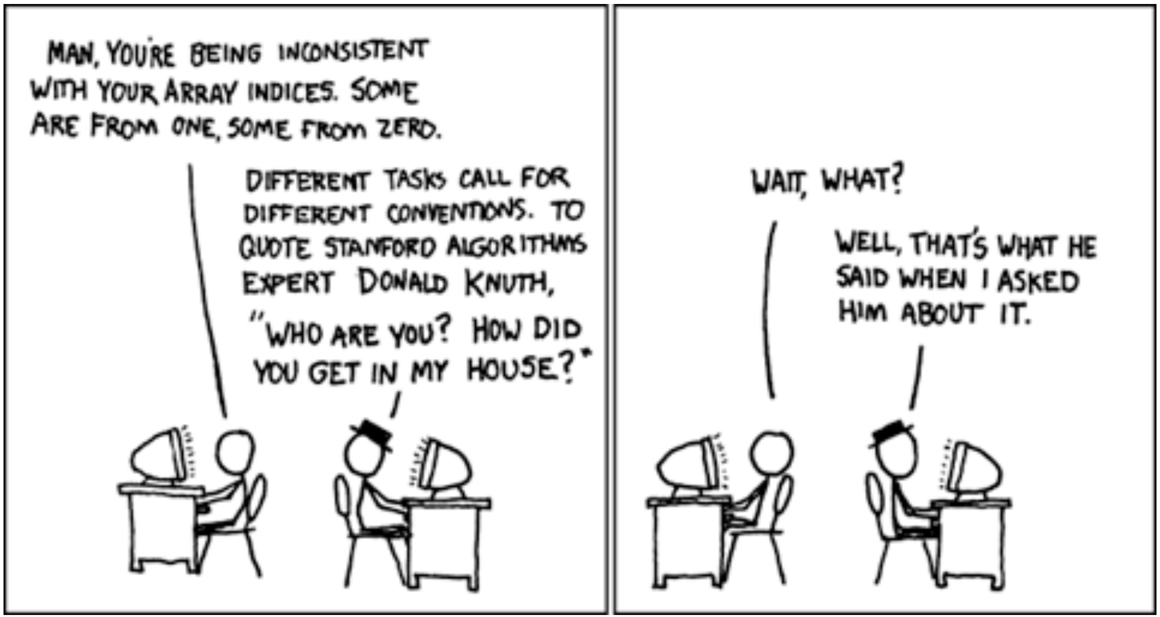
Preliminaries

In this laboratory, you will create a Java class, `Die`, that models a six-sided die. Each part of this lab deals with two classes: a `Run`*XX* class, which is provided, and a `Die` class, which you will create. In each successive part, the Run*XX* class makes calls to additional `Die` methods. You will need to implement these methods as the lab progresses.

> Note: you should not modify any of the `Run`*XX*`.java` files for this lab. You should only modify `Die.java`. Even when BlueJ tells you that there is an error in one of the `Run`*XX*`.java` files, it is really reporting an incompatibility with your `Die.java` file; the problem is in `Die.java`.

There are several RunXX classes. In Part 1 and Part 2, use `RunAB` class, in Part 3, use `RunC` class, in Part 4, use `RunD` class, and in Part 5, use `RunE` class.

Create a folder named `lab9` on the `cs273` area of your network drive. Go to the course website and download the software for Lab9 (lab9.zip). Unzip the software into the `lab9` folder you just created.


MAN, YOU'RE BEING INCONSISTENT WITH YOUR ARRAY INDICES. SOME ARE FROM ONE, SOME FROM ZERO.
DIFFERENT TASKS CALL FOR DIFFERENT CONVENTIONS. TO QUOTE STANFORD ALGORITHMS EXPERT DONALD KNUTH, "WHO ARE YOU? HOW DID YOU GET IN MY HOUSE?"
WAIT, WHAT?
WELL, THAT'S WHAT HE SAID WHEN I ASKED HIM ABOUT IT.

www.xkcd.com

Laboratory

You will be implementing the Die class in this lab. The following chart provides an overview of the attributes (instance variables) and actions (methods) of the Die class.

Die						
	Associated Checkpoint	Public?	Name	Type		
Attributes	Checkpoint 1	no	xCoord	int		
		no	yCoord	int		
		no	currentValue	int		
	Checkpoint 5	no	size	int		
	Associated Checkpoint	Public?	Name	Return Type	Input Parameters	
					name	type
Actions	Checkpoint 1	no	reRoll	void		
		yes	paint	void	g	Graphics
		yes	roll	void	g	Graphics
		yes	Die (constructor)	N/A	xPos yPos	int int
	Checkpoint 2	no	drawSpot	void	g xPos yPos	Graphics int int
	Checkpoint 3	yes	value	int		
		yes	toString	String		
	Checkpoint 4	yes	equals	Boolean	other	Die
	Checkpoint 5	yes	setSize	void	newSize	int

Part 1: Create a simple Die class

1. Open the project.
2. Modify the file `Die.java` (which is presently an empty class definition) so that the class `Die` contains the following internal state, as `private` instance variables.
    - x-coordinate to represent upper left hand coordinate of die (when drawn on a Graphics object)
    - y-coordinate to represent upper left hand coordinate of die (when drawn on a Graphics object)
    - currentValue that tells which face of the die is showing (This should always be in the range 1 through 6.)
3. Create a `public` constructor that takes two parameters: x-position and y-position. The constructor has two steps. First, it should initialize the instance variables using the two parameter values. Second, the constructor should invoke the `reRoll` method to initialize the `currentValue` instance variable. You will implement the `reRoll` method in the next step.
4. Create a `private` method, `reRoll`, which simply updates the current value of the

die with a new (probably different) "random" value between 1 and 6, inclusive. You may find the `Math.random()` method useful for generating "random" numbers.

Hint: If you multiply a random number generated by `Math.random()` by 6, this new number will be in the range [0, 6.0).

Background Information: Observe the five other classes in your BlueJ Project: RunAB, RunC, RunD, RunE, and RunAbstract. Each of these classes, already implemented by Dr. Vegdahl, does the following:

- Creates a `Die` object by calling your `Die` constructor.
- Whenever the user presses the `Roll` button, it invokes the `Die` object's `roll` method.
- Whenever it needs to repaint itself, it invokes the `Die` object's `paint` method, and displays a textual message that tells how many times the die has been rolled.

At this point, if you were to compile your entire BlueJ project, you would have a number of compilation errors in the various Run classes. These errors are due to the fact that you haven't finished writing enough code yet to make your `Die` class compatible with these other classes. Do not change the Run classes; keep programming.

5. Create a `public` method, `paint`, which draws the die onto a `Graphics` object. The method should not return anything. It takes a single parameter: a `Graphics` object. The paint method should have the following behavior:
   - It must draw a single *white* 50 x 50 pixel square, with a *black* border. This is the die. Suppose the input parameter to the method is called `g`, then the method may invoke drawing methods on g by using the syntax `g.fillOval(....arguments here....);`.
   - The die should be drawn such that the x-coordinate and y-coordinate instance variables dictate the upper-left corner of the drawn die.
   - The `currentValue` should be printed as *black* text somewhere inside the die. To do so, `currentValue` must be converted to a String.
   - To perform error-checking, the paint method must check that the `Graphics` object is not `null` before attempting any drawing on the `Graphics` object.
6. Create a `public` method, `roll`. It should not return anything and it takes a single `Graphics` object as a parameter. The `roll` method must call the `reRoll` method to set `currentValue` to a number between 1 and 6, inclusive. After calling the `reRoll` method, it should call the `paint` method so that the die is drawn to the screen.

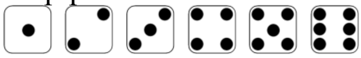
   Design Note: There is a separate `roll` and `reRoll` method because one may want to call `reRoll` in several places inside the `Die` class, but outsiders may only `roll` the die so that the change is reflected on the screen.

7. It may be necessary to add one or more `import` directives to the top of your `Die.java` file before it will compile.
8. Compile the RunAB class, right-click on RunAB, and select `void main(String[] args)` to execute your code.

checkpoint 1 (25 points): Show your lab instructor or assistant the executing program.

Part 2: Modify the paint method in the Die class so that it draws a die with spots (i.e., pips)

1. Create a `private` method, `drawSpot`, that takes three parameters: an x-coordinate, a y-coordinate, and a Graphics object. It should draw one filled circle with diameter of 10 pixels, centered at the x-coordinate and y-coordinate. **IMPORTANT:** It is essential that you implement this such that drawSpot expects the x,y coordinate passed to the method to be specified *with respect to the top-left corner of the die* rather than with respect to the upper left corner of the graphics object defined by the canvas. If this instruction is confusing to you, ask now. Incorrect code will have to be rewritten.
2. Modify the `paint` method so that it draws a die with filled black spots rather than a single digit. For example, if `currentValue` is 2, your paint method should display 2 dots. You will likely have to treat each of the 6 possible die-values as a special case. The switch statement may be useful here. The `paint` method should call (invoke) the `drawSpot` method to draw one spot on the die.

Note: The pips must not touch the edges of the die and must be drawn in traditional dice format: .

checkpoint 2 (20 points): Show your lab instructor or assistant the executing program.

Part 3: Add a method to get the value of the die, and to convert the die into a character-string

Background Information: The RunC class does more than in the previous steps. In addition to the previous behavior:

- it creates a second `Die` object (in a different location than the first) and rolls both when the `Roll` button is pressed
- it prints the minimum two-die sum it has seen so far
- it prints the maximum two-die sum it has seen so far

Due to its increased functionality, it is necessary to implement two additional methods in the `Die` class.

1. Create a `public` method, `value`, that takes no parameters and returns the value of `currentValue` of a `Die` (a number between 1 and 6) object. For example, if

`currentValue` is assigned to 4, then the value returned should be 4.

2. Create a `public` method, `toString`, that takes no parameters and returns a `String` object. The `String` object returned should be a `String` containing a single character, where the character corresponds to the `currentValue` of a `Die` object. For example, if `currentValue` is assigned to 4, then the `String` returned should be "4".

After the `value` and `toString` methods are created correctly, the RunC class should properly print out minimum and maximum values of dice-pairs and the current values of the dice.

checkpoint 3 (20 points): Show your lab instructor or assistant the executing program.

If you have completed all above checkpoints, you have a grade of D (65) for this lab.

Part 4: Add a method to tell whether two dice are equal to one another

Background Information: The RunD class does yet one additional thing: it prints a textual message that tells whether the current dice denote "doubles", that is, if the two die have the same value. It determines if they are equal by invoking the `equals` operation, as in:

```
   if (myDie.equals(myDie2)) ...
```

Java's default behavior for the equals method is "are they the same object?": in other words, are they referenced by the same location in memory. Because two separate `Die` objects are used, the default `equals` method will always return false. It is your job to create an `equals` method that returns true if the dice are showing the same value.

1. Write a `public` method, `equals`, that takes a `Die` object as its only parameter. It should return a `boolean` value that tells whether the face-value, or `currentValues,` of "this" `Die` and the parameter `Die` are equal.

checkpoint 4 (15 points): Show your lab instructor or assistant the executing program.

If you have completed all above checkpoints, you have a grade of B- (80) for this lab.

Part 5: Add a method that allows you to change the size of a die

Background Information: The RunE class implements the same behavior as before. In addition, it changes the size of the second die. It does this by invoking the `setSize` method, as in:

```
      myDie2.setSize(100);
```

or:

```
    myDie2.setSize(40);
```

The RunE class will set the size of the second die as follows:

- If a quoted string denoting a valid integer between 10 and 250 is typed between the "{" and "}" in the text field of the BlueJ: Method Call dialog box, the specified size will be used for the die. Thus, if { "123" } were specified, then the size of the second die would be set to 123.
- Otherwise, the 40 or 100 will be randomly selected for the size of the second die.

Your Die class should work reasonably for sizes of both 40 and 100—and whatever other sizes between 10 and 250 that your instructor/assistant chooses.

1. Because `Die` objects may now have different sizes, you need to store the size of a die as part of its state. Thus, you should add a `private` instance variable to your `Die` class that gives its size in pixels.
2. Add a `public` method, `setSize`, that takes one `int` parameter, and sets the size of the `Die` to the value specified by the parameter. It should not return a value.
3. Modify the `paint` method so that it draws the die as an NxN-pixel square where N is the value of the new instance variable. The sizes and positions of the spots on the `Die` should be adjusted accordingly.
4. The initial size of the die should be 50 before `setSize` is ever called. This means that the `Die` constructor should include the following line:

```
   this.setSize(50);
```

checkpoint 5 (20 points): Show your lab instructor or assistant the executing program.

If you have completed all above checkpoints, you have a grade of A (100) for this lab.

Part 6: Draw the die in three dimensions

1. Modify the `paint` method such that dice are drawn in three dimensions. The "up" face of the die should be mostly showing; a couple of the other faces should also be showing, but in 3-dimensional perspective. This should model an actual die, where the 1 and 6 are on opposite faces of the die (and similarly for 2 and 5, and for 3 and 4). Thus 1 and 6 should never be drawn on adjacent faces.

checkpoint 6 (20 points): Show your lab instructor or assistant the executing program.

If you have completed all above checkpoints, you have a grade of A+ (120) for this lab.

Part 7: finish up

Close all windows. Log off. You're done.