Chapter 10

Mutable Objects

10.1 Exercises

Exercise 10.1 The `java.awt` package contains a sub-package and class for representing a geometric line segment: `java.awt.geom.Line2D.Double` with these attributes, all of type `double`:

- `x1` and `y1`, representing the x and y coordinates of one endpoint of the line segment
- `x2` and `y2`, representing the x and y coordinates of the other endpoint of the line segment

In order to distinguish it from the `Double` wrapper class that you saw in Chapter 9, do both of these `import`s:

```java
import java.awt.geom.Line2D;
import java.awt.geom.Line2D.Double;
```

You can then create a new line with code like this:

```java
Line2D.Double segment = new Line2D.Double(3.7, 4.5, 8.2, 1.6);
```

Write the following methods:

`length(Line2D.Double segment)`
: returns the length of the line segment as a `double`. *Hint*: use the Pythagorean theorem

`stringOf(Line2D.Double segment)`
: returns a `String` representing the line segment. For example, if the segment has endpoints (1.2, 3.4) and (7.8, 9.5), the return value would be `"(1.2, 3.4) - (7.8, 9.5)"`

Write the following `void` methods which alter the fields of their parameters. They do not return a new value:

`flip(Line2D.Double segment)`
: switches the `y1` and `y2` coordinates of the given line segment. For example, if the segment had endpoints (1.2, 3.4) and (7.8, 9.5), after the call the segment would have endpoints (1.2, 9.5) and (7.8, 3.4)

`reflectXAxis(Line2D.Double segment)`
: Reflects the line segment across the x-axis by making the y coordinates the negative of their current values. For example, the segment (1.2, 3.4) to (7.8, 9.5) would become (1.2, -3.4) to (7.8, -9.5)

`reflectYAxis(Line2D.Double segment)`
: Reflects the line segment across the y-axis by making the x coordinates the negative of their current values. For example, the segment (1.2, 3.4) to (7.8, 9.5) would become (-1.2, 3.4) to (-7.8, 9.5)

Now, write a `main` method that prompts the user for the coordinates of a line segment's endpoints and then prints, properly labeled, the segment length and the result of calling the preceding three methods. Here is what the output might look like:

```
Enter x and y coordinates for start point of line: 1.2 3.4
Enter x and y coordinates for end point of line: 5.6 7.8
Length of line segment (1.2, 3.4) - (5.6, 7.8) is 6.22
Flipped segment: (1.2, 7.8) - (5.6, 3.4)
Reflected along x-axis: (1.2, -3.4) - (5.6, -7.8)
Reflected along y-axis: (-1.2, 3.4) - (-5.6, 7.8)
```

Exercise 10.2 As you may have noticed in the preceding exercise, being able to change an object's mutable fields meant that you had to keep re-creating the original line segment before calling each of the flip/reflect methods.

In this exercise, you'll rewrite those methods to return a brand new `Line2D.Double` segment, leaving the original object untouched:

`flip(Line2D.Double segment)`
: returns a new `Line2D.Double` with the `y1` and `y2` coordinates of the given line segment reversed. For example, if the segment had endpoints (1.2, 3.4) and (7.8, 9.5), the method would return a new line segment with endpoints (1.2, 9.5) and (7.8, 3.4)

`reflectXAxis(Line2D.Double segment)`
: returns a new `Line2D.Double` that is the result of reflecting the line segment across the x-axis. It does this by returning a new line segment whose y coordinates are the negative of the original segment's values. For example, when given the segment (1.2, 3.4) to (7.8, 9.5), the return value would be a new segment (1.2, -3.4) to (7.8, -9.5)

`reflectYAxis(Line2D.Double segment)`
: returns a new `Line2D.Double` that is the result of reflecting the line segment across the y-axis. It does this by returning a new line segment whose x coordinates are the negative of the original segment's values. For example, when given the segment (1.2, 3.4) to (7.8, 9.5), the return value would be a new segment (-1.2, 3.4) to (-7.8, 9.5)

The output will look exactly like the preceding exercise's output, but you might find that the code is much more compact.

Exercise 10.3 Some data compression schemes use a technique called *run length encoding* to save space. For example, in the string `"abbbcdddeffff"` can be encoded as `"a3bc3de4f"`, where a digit tells how many times to repeat the following character.

Write a method named `compress` that takes a a non-compressed `String` as its parameter and returns a run length encoded version of the string. Don't run length encode any letters repeated less than three times (a non-repeated letter would require two bytes: one for the length and one for the character; a letter repeated twice would not save any space.)

Write method named `expand` that takes a run length encoded `String` as its parameter and returns the non-compressed String.

These two methods should follow this pattern:

1. Create an empty `StringBuilder` object to hold the result.
2. Iterate through the input `String`, updating the `StringBuilder` object as you go.
3. Convert the `StringBuilder` to a `String` and return it.

You can also create a `StringBuilder` object with the content of the original `String` and update it in place, but that makes the algorithms significantly more difficult.

Finally, write a `main` method that repeatedly asks the user for a string until they enter the empty string. Use that `String` to create a new `StringBuilder` object. Compress the data, print it out, expand it, and print that result. Presume that the repeat count will never be more than one digit; that is, no character is repeated more than nine times.

The following methods in the `Character` class are useful for manipulating characters:

- `Character.isDigit` returns `true` if its `char` argument is a digit, `false` otherwise.
- `Character.digit` takes a character and a number base and returns the corresponding integer value. For example, `Character.digit('8', 10)` returns the integer value 8.
- `Character.forDigit` takes an integer and a number base and returns the corresponding character. For example, `Character.forDigit(7, 10)` returns the `char` value `'7'`.

Here is an example of what the program might look like:

```
Type a string or press ENTER to quit: aBBBcDDDDeFFFFFg
Compressed string: a3Bc4De5Fg
Expanded string: aBBBcDDDDeFFFFFg

Type a string or press ENTER to quit: no repeated info
Compressed string: no repeated info
Expanded string: no repeated info

Type a string or press ENTER to quit:
```