COLBY COLLEGE
Department of Mathematics
and Computer Science

CS115 **Allen B. Downey**
Spring 1998

Assignment 9: Objects

Due: Wednesday 14 April

System objects

`System` is the name of the built-in class that contains methods and objects used to get input from the keyboard, print text on the screen, and do file input/output (I/0).

`System.out` is the name of the object we have used to print text. When you invoke `print` and `println`, you invoke them on the object named `System.out`.

Interestingly, you can print `System.out`:

```
System.out.println (System.out);
```

The output is:

```
java.io.PrintStream@80cc0e5
```

As usual, when Java prints an object, it prints the type of the object, which is `PrintStream`, the package in which that type is defined, `java.io`, and a unique identifier for the object. On my machine the identifier is `80cc0e5`, but if you run the same code, you will probably get something different.

There is also an object named `System.in` that has type `BufferedInputStream`. `System.in` makes it possible to get input from the keyboard. Unfortunately, it does not make it easy to get input from the keyboard. First, you have to use `System.in` to create a new `InputStreamBuffer`.

```
    InputStreamReader in = new InputStreamReader (System.in);
```

Then you use `in` to create a new `BufferedReader`:

```
    BufferedReader keyboard = new BufferedReader (in);
```

The point of all this manipulation is that there is a method you can invoke on a `BufferedReader`, called `readLine`, that gets input from the keyboard and converts it into a String. For example:

```
    String s = keyboard.readLine ();
    System.out.println (s);
```

reads a line from the keyboard and prints the result.

There is only one problem. There are things that can go wrong when you invoke `readLine`, and they might cause an IOException. There is a rule in Java that if a method might cause an exception, it should say so. The syntax looks like this:

```
public static void main (String[] args) throws IOException {
```

This indicates that `main` might "throw" an IOException. You can think of throwing an exception as similar to throwing a tantrum.

1. Start with a nice clean copy of the palindrome program. If you didn't get it working, use my solutions to patch it up (they are in notes22.txt). Add a line at the beginning of the file that says `import java.io.*;`. That means we should import all the classes in the `java.io` package.

2. Type in the code on the previous page and play around with it. What error message do you get if you forget to declare that `main` throws an exception? What if you forget the `import` statement? What happens if you type weird keys into `readLine`? Is there anything you can type to cause an IOException?

3. Write code that generates the following output:

```
Type a word or phrase.
I will check and see if it is a palindrome.

> banana

You typed: banana
banana is not a palindrome.
```

4. Put your code in a loop so that it prompts the user to type a word over and over, and it checks each word for palindromity.

5. Add a condition to the loop so that if the user types the word "stop," the program terminates. Remember that you cannot use the `==` operator to compare Strings.

Using objects as parameters

1. Start with a clean copy of the Snowperson assignment. You can use my solutions if you want or you can use your own.

   `http://rocky.colby.edu/cs115/solutions/hw4/`

   If you use your solution, you might want to take a minute to review it, make sure you remember how it works, and clean it up a little.

2. Add a few lines of code to the program and experiment with Rectangle and Point objects a bit. Try out one or two of the examples from the book or Quiz 12. Try creating an object variable (but no object) and then access one of the instance variables. What happens? Look at the output from the exception handler and see if you can find the line of the program that caused the error.

3. Modify the Snowperson code so that every method that receives a bounding box as an argument gets a single Rectangle object instead of four integers. For example, the declaration of `paintSnowperson` would look like

    ```
    public void paintSnowperson (Graphics g, Rectangle bb)
    ```

4. Along with your modified code, turn in a paragraph that compares the two versions, discussing things like readability, debuggability, code length and encapsulation.

    In the Java library, they had the option of using Rectangles to pass parameters to methods like drawRect and drawOval, but they chose to pass four integers instead. In fact, there are only a couple of methods in the whole Java library that use Rectangles at all. What do you think of this decision? Would you prefer a version of drawOval that takes a single Rectangle as an argument? What if you could have both?

Using objects as return values

1. You are probably getting sick of the factorial method by now, but just for practice, write an iterative version of `factorial`. Put it in a new project named Big.

2. Print a table of the integers from 0 to 30 along with their factorials. At some point around 15, you will probably see that the answers are not right any more. Why not?

3. Go to the Sun web page that contains all the Java documentation; there is a link to it from the class web page. Click on the `java.math` package and then click on the `BigInteger` class. Print the documentation of the BigInteger class. Read it.

    BigIntegers are built-in objects that can represent arbitrarily-big integers. There is no upper bound except the limitations of memory size and processing speed.

4. There are several ways to create a new BigInteger, but for some bizarre reason, providing an integer as an argument is not one of them. On the other hand, we can convert an integer to a String and then use a String to create a BigInteger:

    ```
    public static BigInteger convertToBig (int n) {
      String s = Integer.toString (n);
      BigInteger big = new BigInteger (s);
      return big;
    }
    ```

    Type in that code and try out a few simple cases like creating a BigInteger and printing it. Notice that `println` knows how to print BigIntegers! Don't forget to add `import java.math.BigInteger` to the beginning of your program.

5. Unfortunately, because BigIntegers are not primitive types, we cannot use the usual math operators on them. Instead we have to use object methods like `add`. In order to add two BigIntegers, you have to invoke `add` on one of the objects and pass the other as an argument. For example:

    ```
    BigInteger small = convertToBig (17);
    BigInteger big = convertToBig (1700000);
    BigInteger total = small.add (big);
    ```

Try out some of the other methods, like `multiply` and `pow`.

6. Convert `factorial` so that it performs its calculation using BigIntegers, and then returns the BigInteger as a result. You can leave the parameter alone—it will still be an integer.

7. Try printing the table again with your modified factorial function. Is it correct up to 30? How high can you make it go? On my machine, I calculated the factorial of all the numbers from 0 to 999, but it took over 2 minutes. The last number, 999!, has 2565 digits.