Homework 4

Due: in final review class (tentatively May 10)

1. A JScrollPane adds optional scrollbars to another component. It is used to add scroll bars to JTextArea, JTextPane (formatted text), or JTable. For example:

```
/** a multi-line text area where you can type text */
private void initComponents() {
    JTextArea textarea = new JTextArea(6, 40); // size: (rows,columns)
    // automatically wrap text at word boundaries
    textarea.setLineWrap(true);
    textarea.setWrapStyleWord(true);
    // add scroll bars that automatically appear when needed
    JScrollPane pane = new JScrollPane(textarea);
    pane.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED);
    this.add(pane);
    pack();
}
```

1.1 What design pattern does JScrollPane use?

1.2 Draw a class diagram that shows how JScrollPane implements the pattern. Note that, according to the pattern, JScrollPane can wrap *any* `Component`.

1.3 What is the advantage of providing scroll bars in this way, rather than having each class provide scroll bars itself, e.g. JTextArea, JTextPane, JTable, and JList could each have "built-in" scroll bars?

2. In the Java I/O classes, InputStream (and its subclasses) read an input source as **bytes**. The principle methods are read() one byte, read(byte[] b), and read(byte[] b, int offset, int length).

A Reader, such as InputStreamReader reads bytes from an InputStream and provides the data as *characters*. Converting bytes to characters is not simple, since a character may be 1 to 4 bytes long (using UTF-8 encoding). The principle methods are read() one character, read(char[] c), and read(char[] c, int offset, int length).

In other words, a Reader *converts* a byte-oriented interface to a character-oriented interface.

```
InputStream in = new FileInputStream( "filename" );
Reader reader = new InputStreamReader( in );
char c = (char) reader.read(); // read one char
```

2.1 What design pattern does Reader use? Hint: it *converts* one interface to another interface.

2.2 Draw a class diagram of the relationship between Reader, InputStreamReader, and InputStream.

Note: only use Reader objects for *character* input. If you use a Reader to read *binary* data, such as an image or audio file, the results will be corrupted.

3. BufferedReader is a Reader that *wraps* another Reader and adds the ability to read input one "line" at a time. BufferedReader gets the characters from the wrapped Reader object. Continuing the above example:

```
BufferedReader bufRead = new BufferedReader( reader );
String line = bufRead.readLine( );
// readLine() removes the newline character from data. It returns null if no more input.
```

What design pattern does BufferedReader use? Justify your answer.

4. Swing has some examples of components that use the Model-View-Controller (MVC) pattern. The controller is incorporated in another component, so there are only "M" and "V" as separate objects.

4.1 Give an example of a pair of Swing classes that use the MV(C) pattern, and explain how they use the pattern.

4.2 In the MVC pattern, the model needs a way to notify the view when the model data changes so the view can update itself. In your Swing example, how can the model notify the view?

4.3 What is the advantage of separating the model and view into separate classes? Why not have just one class that handles everything?

5. The usual way of starting a Swing UI is like this:

```
public class MyFrame extends JFrame
    public MyFrame() {
        initComponents();  // initialize GUI components
    }
    . . .
    public static void main(String [] args) {
        SwingUtilities.invokeLater( new Runnable() {
            public void run() {
                new MyFrame().setvisible( true );
            }
        } );
    }
```

Rewrite the main method to use a *lambda expression* and be more compact.

Lambda Programming

The following problems refer a Student class with this structure.

There is example code for these problems so you can verify your answer. It requires Java 8 to run. The code is at:

https://bitbucket.org/skeoop/hw4

To get a List of students, use:

```
StudentFactory.getInstance().getStudents()
```

Student
-name: String -id: String -sex: char {'M' or 'F'} -birthday: LocalDate
getName(): String getId(): String getSex(): char getBirthday(): LocalDate

6. Write a Compator<Student> that orders students by birthday, so the oldest student comes first. Write this as a lambda expression.

7. Write a java.util.function.Predicate named **male** that returns true if a Student (the parameter) is male.

7.1 Write this as an anonymous class.

7.2 Write this as a lambda expression.

7.3 Write a lambda named **female** that is true for female students.

7.4 Write a Predicate named **`bymonth`** as a lambda that returns true if the Student parameter has birthday in April.

8. Write a Consumer named **`print`** that accepts a Student as parameter and prints to System.out:

Mr. Foo Bar, born mm/dd/yyyy or *Ms. Foo Bar, born mm/dd/yyyy* depending on sex.

(use the student's actual name and birthday, of course).

9.1 Finally, write a static method that accepts a List, a Predicate, and a Consumer and applies the predicate and consumer to each element in the list:

public static <T> filterAndPerform(List<T> list, Predicate<T> filter, Consumer<T> consume)

9.2 How would you use this method to print all students born in April?

9.3 How would you use this method to print all *male* students born in April, using just one line of code? You should not modify the filterAndPerform method – just change the parameter values.

10. The method filterAndPerform can be replaced by a single line of code so we don't really need the method at all. Complete the blanks.

```
List<Student> students = StudentFactory.getInstance().getStudents();
students.stream().filter( ____________ ).forEach( _____________ );
```

Reference: *The Java 8 Tutorials* on "Aggregate Operations".

11.1 The **`bymonth`** lambda is clumsy: we have to "hard code" the number of the month we want. Write an **`IntFunction`** that creates a lambda for any month we want. The month is specified as the **`IntFunction`**'s parameter, such as `formonth(4)`.

```
import java.util.function.IntFunction;
// formonth takes a month (1-12) as parameter and returns a
// Predicate that is true for students born in that month.
IntFunction< Predicate<Student> > formonth = __________?____________
```

In other words, `formonth(4)` returns a Predicate<Student> that is true for students born in April (month 4), just like `bymonth`.

11.2 Use `formonth`, `female` (problem 7.3), and `print` (problem 8) to print all female students born in August (month 8). Use just one line of code.