American University of Armenia
CS 120 Intro to OOP
Spring 2019

Homework Assignment 5, Part 2

In this part of the homework you will improve your Game of Life implementation to handle errors properly and to be able to list and select patterns from a set of patterns stored as a text file specified either locally or on the web. There are four parts: adding exceptions; reading the pattern data into a new object of type `PatternStore`; storing and manipulating the data once read; and updating the program interface to use the new functionality.

1. **(10 points)** The current implementation does not cope well when input data is malformed or missing. To start you should create a custom `PatternFormatException` exception class (and define two constructors for it) to embody the errors we get.

   Your first task is to adapt `Pattern` to throw this custom exception with specific messages exemplified here:

   ```
   Input:  Valid string, e.g.  "Glider:Richard K. Guy:20:20:1:1:001 101 011"
   Exception message:  <no exception thrown>

   Input:  ""
   Exception message:  "Please specify a pattern."

   Input:  "SomeRandomString"
   Exception message:  "Invalid pattern format:  Incorrect number of fields in pattern
   (found 1)."

   Input:  "Glider:Richard Guy:20:20:1:"
   Exception message:  "Invalid pattern format:  Incorrect number of fields in pattern
   (found 5)."

   Input:  "Glider:Richard Guy:a:b:1:1:010 001 111"
   Exception message:  "Invalid pattern format:  Could not interpret the width field as
   a number ('a' given)."

   Input:  "Glider:Richard Guy:20:20:one:1:010 001 111"
   Exception message:  "Invalid pattern format:  Could not interpret the startX field
   as a number ('one' given)."

   Input:  "Glider:Richard Guy:20:20:1:1:010 0a1 111"
   Exception message:  "Invalid pattern format:  Malformed pattern '010 0a1 111'."
   ```

   Modify `Pattern` so that the constructor for the class as well as the method `initialise` will throw a `PatternFormatException` with these inputs. To test, you may find it a good idea to catch the exceptions in `GameOfLife` and print the message. Note how the exception message reflects the incorrect parts of the input.

2. **(10 points)** Information is stored as a sequence of binary digits. However, it is usually the case that the data we want to store has a higher abstract meaning than that—we might

want to store ints or Strings for example. In this case we use an *encoding* to transform our data to binary and back again.

One of the most common encodings is ASCII which specifies how to store characters as binary using one byte per character. Another common standard is UTF-8 which can encode a much bigger range of characters by using one or more bytes.

Your data sources will use ASCII, with one pattern per line in the format you've been using so far. For example:

```
Glider:Richard K. Guy:8:8:1:1:001 101 011
Blinkers:Horton Conway:8:8:0:0:01000000 01001110 01000000 00000000 00000100 11100100 00000100 00000000
Octogon:Goodman/Taber:8:8:0:0:00011000 00100100 01000010 10000001 10000001 01000010 00100100 00011000
Block+Boat+Beehive:Unknown:8:8:0:0:11000000 11000100 00001010 00001010 00000100 11000000 10100000 01000000
```

Java offers the Java I/O Standard library to handle reading and writing data. An `InputStream` represents a source of binary data such as a file on disk or a sensor. An `OutputStream` represents a place to write binary data such as a file on disk or a client on the network. Streams provide you with an interface that lets you read and write `byte`s.

Textual data is so common that the standard library provides special support for this. A `Reader` represents a source of character data and a `Writer` represents a sink for character data. These provide you with an interface that lets you read and write `char`s. When you create a `Reader` or a `Writer` an *encoding* is used to interpret the raw bytes correctly. Java will use a default encoding suitable for your platform but you can also specify it yourself.

Take a look at the documentation for the `Reader` class in the Java online documentation. Pay particular attention to the methods for reading characters. For example, the method `int read(char[] cbuf)` describes a method that reads data into a `char` array. What's missing is any way to get a whole line. This is best done using a `BufferedReader`, which adds a method `String readLine()` to read a single line in. This class "is a" `Reader` but unusually it also "has a" `Reader`. You need to know how to get at a file line-by-line:

```java
Reader r = new FileReader("path/to/your/file");
BufferedReader b = new BufferedReader(r);
String line = b.readLine();
while (line != null) {
  // Do whatever you need to do with line
  line = b.readLine();
}
```

A more concise way of writing this would be:

```java
Reader r = new FileReader("path/to/your/file");
BufferedReader b = new BufferedReader(r);
String line;
while ( (line = b.readLine()) != null) {
  // Do whatever you need to do with line
}
```

See if you can work out what's going on there and explain in your report why we can replace the former with the latter.

As you will have guessed, `FileReader` is a specialised `Reader` for reading files on the your computer. You will also need to read files over the web given a URL. This is the same, except you have to get a different type of `Reader`:

```java
URL destination = new URL("https://www.whatever.com/yourfile.txt");
URLConnection conn = destination.openConnection();
```

```
Reader r = new InputStreamReader(conn.getInputStream());
BufferedReader b = new BufferedReader(r);
// Rest as above
```

Create a new class `PatternStore` using this template:

```
import java.io.*;
import java.net.*;
import java.util.*;

public class PatternStore {

  public PatternStore(String source) throws IOException {
    if (source.startsWith("http://") || source.startsWith("https://")) {
      loadFromURL(source);
    }
    else {
      loadFromDisk(source);
    }
  }

  public PatternStore(Reader source) throws IOException {
    load(source);
  }

  private void load(Reader r) throws IOException {
    // TODO: read each line from the reader and print it to the screen
  }


  private void loadFromURL(String url) throws IOException {
    // TODO: Create a Reader for the URL and then call load on it
  }

  private void loadFromDisk(String filename) throws IOException {
    // TODO: Create a Reader for the file and then call load on it
  }

  public static void main(String args[]) {
    PatternStore p = new PatternStore(args[0]);
  }
}
```

Complete the TODO sections.

Test your class for a simple text file and for a URL. For the URL you should be able to pass it the argument “https://bit.ly/2V7uEc3” and see:

```
phi:Life lexicon:11:10:3:3:01110 10001 10001 01110
Glider:Richard K. Guy:8:8:1:1:001 101 011
aircraft carrier:Life lexicon:8:7:2:2:1100 1001 0011
```

3. **(10 points)** Once you’re happy that you’re reading in the data, you need to store it. We will rely on (partially filled) arrays.

Add the following state to `PatternStore`. `patterns` is a partially filled array of all patterns in the store (order unspecified); `MAX_NUMBER_PATTERNS` is the maximum potential number of patterns; `numberUsed` is the actual number of patterns in the store.

```java
public static final int MAX_NUMBER_PATTERNS = 1000;

private Pattern[] patterns = new Pattern[MAX_NUMBER_PATTERNS];
private int numberUsed = 0;
```

Adapt your `load` method to read the data into this new structure. If a line in the file is malformed you should print out the offending line as a warning, but just continue (i.e. not kill the program).

Make sure you only ever make one `Pattern` object for a given pattern.

4. **(10 points)** Copy the following into your `PatternStore`

```java
public Pattern[] getPatterns() {
  // TODO: Get an array of all patterns
}

public String[] getPatternAuthors() {
  // TODO: Get a sorted array of all pattern authors in the store
}

public String[] getPatternNames() {
  // TODO: Get a sorted array of all pattern names in the store
}
```

Implement and test the methods above. Make sure that your implementations do not result in any privacy leaks.

Hint: Check out the documentation for the `Arrays` class in the Java online documentation to find an easy way to sort an array of `Strings`. In your report, explain which method and with which parameters (number and types) you can use for this task.

5. **(10 points)** Your final task is to update the `GameOfLife` so it takes a path or URL to a `PatternStore` when started:

```
> java GameOfLife https://bit.ly/2FJERFh
```

You do not need to support any other form of running the game (so `--array` etc is no longer required). Instead you start the game, it loads all patterns and you can then enter:

- `l` to list all the patterns with a number associated to each
- `p X` to start playing pattern number `X`
- `f` to move forward one generation once play has begun.

If the pattern fits inside 64 bits, you should use `PackedWorld`, otherwise `ArrayWorld`. This all goes inside the `play` method in `GameOfLife`, which is mostly done for you here:

```java
public void play() throws IOException {

  String response = "";
  BufferedReader in = new BufferedReader(new InputStreamReader(System.in));

  System.out.println("Please select a pattern to play (l to list):");
  while (!response.equals("q")) {
    response = in.readLine();
    System.out.println(response);
    if (response.equals("f")) {
      if (world == null) {
        System.out.println("Please select a pattern to play (l to list):");
      }
      else {
        world.nextGeneration();
        print();
      }
    }
    else if (response.equals("l")) {
      Pattern[] names = store.getPatterns();
      int i = 0;
      for (Pattern p : names) {
        System.out.println(i + " " + p.getName() + "  (" + p.getAuthor() + ")");
        i++;
      }
    }
    else if (response.startsWith("p")) {
      Pattern[] names = store.getPatterns();
      // TODO: Extract the integer after the p in response
      // TODO: Get the associated pattern
      // TODO: Initialise world using PackedWorld or ArrayWorld based
      // on pattern world size
      print();
    }
  }
}

public static void main(String args[]) throws IOException {
  if (args.length != 1) {
    System.out.println("Usage: java GameOfLife <path/url to store>");
    return;
  }

  try {
    PatternStore ps = new PatternStore(args[0]);
    GameOfLife gol = new GameOfLife(ps);
    gol.play();
  }
  catch (IOException ioe) {
    System.out.println("Failed to load pattern store");
  }
}
```

Note that `System.in` is converted into a `BufferedReader` so we can read strings from the command line, just like with files or URLs.

Give `GameOfLife` a member variable `store` that holds a reference to the `PatternStore` you create (and pass to the constructor in `main` above).

Complete the TODO section in `play()`.

Add a new constructor to `World` that takes a `Pattern` object, and corresponding constructors in `ArrayWorld` and `PackedWorld`.

You should now be able to run:

```
> java GameOfLife https://bit.ly/2FJERFh
```

and play any of the 400+ patterns.