Jacquard Loom - A1
Due: Sep 29, 2022 at 11:59PM

1 Jacquard Loom - A1 Specification

1.1 Available Resources

- Lecture slides
- Other sections of the provided code
- me
- The textbook
- Java documentation for List/ArrayList on the Oracle website

1.2 Assignment Instructions

- This is an *individual* assignment.
- Make sure to read through all of the specifications so your submission is complete.
- Follow all the submission steps in the Setup document by the assignment deadline.

1.3 Assignment Link

The skeleton code for the assignment is available at `https://classroom.github.com/a/LKfl9xWn`.

1.4 Looms

Looms are machines that have allowed humans to quickly weave fabric and tapestries for thousands of years. A basic loom has one set of strings held under tension, and another set of perpendicular strings which are woven, row by row, through the first set. In this way, a tapestry is built one row at a time. Once it's finished, the pattern that it displays is based on the over/under pattern of the woven thread through the tensioned thread.

The first human-built machine that could run sequences of simple instructions was a fabric-weaving loom in around 1805. In that machine, a sequence of punched cards decided if the thread being pulled across the loom (the weft) went above or below the line of tensioned threads (the warp). With different color threads for the warp and the weft, this allowed the cards to dictate which color was visible for any given stitch in the cloth.

A card gave directions for a single row of the cloth, so a sequence of cards dictated a potentially complicated pattern in a cloth! If you replace the sequence of cards with a different sequence and run the loom, you will get a different pattern. Computer programs work the same way, although their effects change electronic memory rather than changing physical thread "memory".

1.5 Implementation

The goal of this assignment is to practice writing Java code that uses OOP design ideas, and to use List operations in order to simulate a Jacquard loom.

Rather than labs, where we define the functionality of a data structure, we will use the built-in Java data structures to help solve larger problems for assignments.

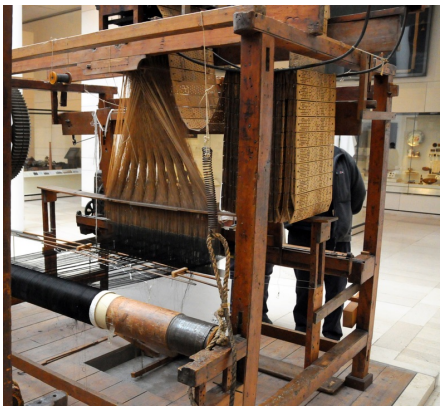
For this assignment, use the class `ArrayList` as a data structure to hold all the information that you need to keep track of. `ArrayList` has the same `List` interface that we expect from `SLList`. Example usage:

```
List<Color> row = new ArrayList<>();
row.add(Color.red);
Color c = row.get(0)
```

1.6 Class Interactions

There are several classes required for a tapestry to be produced:

- The **Card** class describes a single punchcard as a list of **Hole** objects.
- The **Loom** class keeps track of the thread colors installed on the loom, and the part of the tapestry that has already been woven.
- The **Weaver** class behaves like a human technician might – it builds a List of cards from a pattern file, and runs a row of thread on a loom, based on where in the List of cards it has reached.
- Lastly, the **Studio** holds a weaver and a loom, and provides the GUI for us to run the program.



Loom with punchcards.

Studio results.

2 Class Implementations

There are `// TODO` markings throughout the different classes that are your responsibility to complete. Right now, methods return `null` if they return anything, but you should replace those returns with proper functionality.

The classes are discussed in the order that you should implement them. Each class has a `main` method for simple unit and integration testing. The **Studio** class has the `main` method that should be run to launch the GUI and visualize results.

2.1 Card

A Card is used to store a list of Hole objects. It has simple `get` and `set` methods, and a `punch` method that converts a String of x's and spaces into a List of Hole objects. Read through the method comments to understand better how each method should behave.

2.2 Loom

A Loom keeps track of the two thread colors in use, and the tapestry that has been woven thus far. The warp color is set when the loom is created, as those threads cannot be changed. The weft can technically be changed out, though we won't change it mid-weave in this assignment.

Note the type of the tapestry: `List<List<Color>>`. This means the information is stored as a list of lists of colors. A list of colors defines one row of colors, so a list of lists defines a list of rows. This is a standard way of storing two-dimensional information. Read through the tests in `main` to see code examples for getting a specific row or specific color from a row with a list.

There are two interesting methods in **Loom**: `run` takes a Card and adds a list of Color to the tapestry on the loom. The list should be built by reading through the Card's list of holes, and adding Colors to the row. Once the row is complete, it can be added to the list of rows (aka list of list of colors). `getTapestry` does two things: it resets the tapestry on the loom, and it returns the removed tapestry.

2.3 Weaver

A Weaver performs the tasks that a loom-operating human would: it can build a list of cards based on a pattern sheet, it can hold a deck of cards and keep track of where in the deck it currently is, and it can run a loom. The weaver keeps track of its current place in the deck with an iterator.

`weaveRow` attempts to feed a card to the supplied loom. (When a loom is run, it updates its tapestry so the weaver just needs to run the loom with the appropriate card.) If the weaver has no cards left, the `weaveRow` call returns **false**.

`buildCards` uses the supplied file path to open and scan through a pattern file for a text description of the card deck. It should create a new list of cards, and then create new cards, punch them based on the pattern file, and store them in the list. If you need a refresher on Scanner objects to read text files, refer back to CS1 or CS2 assignments that dealt with file input.

2.4 Studio

Once all of the classes are running their unit tests correctly, you should be ready to run the `Studio` class. Pick a weft color and load a pattern file (from the `patterns` directory) before weaving a tapestry. If all goes well, you will see your tapestry printed out, row by row!

If something is broken, Java will crash and conveniently give you the line of code that something broke on. At that point, it's time to start the debugging process – considering what value you expected, what value was found instead, and what part of code was responsible for the inconsistency.

2.5 Patterns

I have provided a few pattern files in the `patterns` subdirectory. A valid pattern file has only `x`'s and spaces on a line, and every line has the same length. **The `x` dictates a stitch where the card is punched and**

a warp color shows on top, the <space> means there is no punch and the weft color shows up on top. Each row of the pattern file corresponds to one row on the tapestry.

In addition to the supplied patterns, create your own pattern file in `pattern/newPattern.txt`. It should be at least a 10x10 pattern, with whatever design you want. Make sure that it displays correctly with your loom!

3 Double Check:

- Have you implemented all the `TODO` methods?
- Does the Studio weave and display your pattern correctly?
- Have you written a new pattern in `newPattern.txt`?
- Have you committed/pushed your code from **all** the files you modified?

4 Grading

Card class punches holes: +10%

Card class gets size and holes: +10%

Loom class gets and sets weft: +10%

Loom class runs and returns tapestry: +20%

Weaver builds cards: +20%

Weaver gets and sets cards: +10%

Weaver weaves: +10%

newPattern supplied: +10%