COSC 102 Spring 2015

Stack Applications Lab 10

Due date: **Submit on Moodle before 11:55 p.m. the day after your lab**

The objectives of this lab are

1. to work on two applications of stacks and
2. to write an `Iterator`.

In this week's lab, you will work with the `StackInterface` and its `Vector` based implementation, the `VectorStack`, which are provided by our textbook

First you will work on a couple of static functions to solve two simple problems. Then you had navigation functionality to a web-browser to permit going back and forth between webpages.

Setup

Download the `zip` file from the course website to get the starter files.

Feel free to use Eclipse if you want. Read and replicate last week instructions for the files in each folder of the downloaded `zip` file. We are here to help, let us know if you are having problem setting up the environment.

Balancing Parentheses

In the file `StackDriver.java` your task is to complete the `checkBalance` method using the traditional `Stack` methods declared in `StackInterface.java`.

In class we have seen an algorithm to determine whether an arithmetic expression is balanced with respect to parentheses. For example,

```
(w * [x + y] / z - [p / {r - q}])
```

is balanced, but the expression

```
(w * [x + y) / z - [p / {r - q}])
```

is not.

The helper method `checkBalance` takes a `String` expression and returns a `boolean` value indicating whether the expression is balanced. Complete its body to include a loop that processes each character of the expression and handles parentheses with stack operations to check the parentheses correspondences. You might find a `switch` statement useful: notice in `SwitchDemo2` how it can have multiple `case` labels.

Use the helper method `match` to recognize matching parentheses types.

Implementing and Using an Iterator

You will implement and use an `Iterator` object so you will need

```
import java.util.Iterator;
```

in the classes concerned.

In `StackDriver.java` uncomment the last line of `testStackOperations()`, i.e.,

```
print(myStack);
```

This line calls the helper method `print` that uses a generic `VectorStack`. Note the generic in the function declaration: the generic type is "parameterized" with the first `<T>` of the method declaration.

```
public static <T> void print(VectorStack<T> s)
```

With this `<T>`, the generic type `<T>` essentially becomes a variable that can be used in place of a type for the scope of the function. This additional generic does not change the way the function is called, and the value of `<T>` is inferred from the parameter.

The goal is to have this helper method prints the content of any `VectorStack` parameter **without changing the object state**. In our present call we want at the console

```
Joe
Jane
Jill
Jess
```

without changing the state of `myStack`. Calling it again should produce the same result.

To implement such functionality

- `VectorStack` needs to be modified to implement `Iterable<T>` with the method returning an `Iterator<T>` object implemented as an inner class. Refer to the `LinkedListWithIterator` class we studied in class and that is posted on our Moodle course.
- The method body of `print` needs to be completed: the iterator obtained from `VectorStack` and used to traverse the stack content. The top element should be printed first.

Once it works for `myStack` create in `testStackOperations` a new `VectorStack` that stores `Integer` references and call `print` passing it as an arguments. It should work similarly.

Before starting the next part, show the instructor your parentheses and iterator implementations.

Web Browser

In this part, you add the navigation capability provided by the back and forward buttons to a web browser program. Consider the following two extra class files.

- `Browser.java`. The web application program with the `main` method that opens a browser window, after some GUI initialization. It contains the code to retrieve pages from the web and to respond to window events, such as button presses and hyperlink clicks. The `Browser` class has also an instance of a `BrowserHistory` object to help navigating among the webpages previously visited.
- `BrowserHistory.java`. **Your code**: the class you implement. It contains methods called from `Browser` to add the navigation functionality to the browser window.

To start the program select the `class Browser` to run its `main` method. It takes no arguments. Running the program opens a browser window containing a display pane and a toolbar. The display pane and address bar are initially empty, and the `Back` and `Forward` buttons are disabled.

- Type a complete URL (including its protocol, for example, `http://` is necessary). Try Colgate cs webpage `http://cs.colgate.edu`
- Hit enter or press the `Go` button to load the page in the display pane.
- Click on hyperlinks in the displayed page to load the next target page in the display.
- Java has built-in support for HTML 3.2, and GUI controls are used to do the web-page rendering: the display is primitive.

The behavior you add is such that when the user clicks the **`Back`** or **`Forward`** button, a new URL corresponding to previously-visited pages (just like those buttons in a traditional web browser) is loaded in the browser.

Notice that once a URL is loaded, the **`Source`** checkbox can be toggled to see the raw text sent to the browser. HTML uses tags and part of its validation is matching opening and closing tags...

Back and Forward Buttons

To implement the functionality of a web-browser `Back` and `Forward` buttons you have to complete the file `BrowserHistory.java`. Don't forget to read the comments at the top of the file.

Functionality

When a user enters a URL in the address bar for the first time, nothing should happen. After that, anytime a new URL is accessed (whether it is from the address bar or a hyperlink), the previous URL should be recorded. When the user clicks the `Back` button that URL is reloaded.

A user should be able to click the `Back` button multiple times, visiting previously loaded pages in order.

The `Forward` button is only enabled after a user has clicked the `Back` button. The page displayed before the user clicks the `Back` button is recorded for forward traversal, so that the `Forward` button loads that page. Multiple `Back` visits should permit multiple `Forward` visits.

The `Forward` button should be disabled when the user clicks on a hyperlink or enters an address through the address bar. This new URL essentially creates a new forward path, deleting the old sequence of webpages. Visiting a new URL does not disable the `Back` button; in fact, previous pages are retained and the current page is added. The `Forward` button should also be disabled when the user has reached the end of the forward sequence.

The `Back` button should be disabled when there are no pages to reload, and enabled if there are any. In particular, the `Back` button should be enabled when the user visits a second URL after opening the browser window.

Methods

The functionality should be implemented using the `back()`, `forward()`, and `hyperlink()` methods in `BrowserHistory.java`.

A pointer to the browser window is contained in the private instance variable `browser` of BrowserHistory.java. Use it to call instance methods contained in the class
`Browser`. In particular you should use the following methods of the `Browser` object:

- `setBack(boolean b)`: sets the enabled state of the `Back` button in the browser window to `b` (`true` means enable; `false` means disable)
- `setFwd(boolean b)`: sets the enabled state of the `Forward` button in the browser window to `b`.

You may find the following method useful (but you don't have necessarily to use it; it depends of your algorithm):

- `String getURL()`: returns the URL currently displayed in the browser. In particular, this means:
  - in `back()` and `forward()`, it is the one displayed at the time the user clicked the button (the "old" page) and
  - in `hyperlink()`, the URL already fetched and just about to be displayed is returned.

Submit

Demonstrate your programs during the lab.

Submit on Moodle the following files

- `StackDriver.java`,
- `VectorStack.java` and
- `BrowserHistory.java`.