Browsr: A (Very Incomplete) Web Browser

Software-ontwerp
2020-2021

Iteration 2

Contents

1 Introduction

For the course *Software-ontwerp*, you will design and develop *Browsr*, a very incomplete web browser. The main challenge will be the user interface layer, which you will design from scratch. In Section 2, we explain how the project is organized, discuss the quality requirements for the software you will design and develop, and describe how we evaluate the solutions. In Section 3, we explain the problem domain of the application. The use cases are discussed in Section 4. Finally, we specify some implementation constraints in Section 5.

2 General Information

In this section, we explain how the project is organized, what is expected of the software you will develop and the deliverables you will hand in.

2.1 Team Work

For this project, you will work in groups of four. Each group is assigned an advisor from the educational staff. If you have any questions regarding the project, you can contact your advisor and schedule a meeting. When you come to the meeting, you are expected to prepare specific questions and have sufficient design documentation available. *If the design documentation is not of sufficient quality, the corresponding question will not be answered.* It is your own responsibility to organize meetings with your advisor and we advise to do this regularly. Experience from previous years shows that groups that regularly meet with their advisors produce a higher quality design. If there are problems within the group, you should immediately notify your advisor. Do not wait until right before the deadline or the exam!

To ensure that every team member practices all topics of the course, a number of roles are assigned by the team itself to the different members at the start of each iteration (or shortly thereafter in case of the first iteration). A team member that is assigned a certain role will give the presentation or demo corresponding to that role at the end of the iteration. That team member is *not supposed to do all of the work concerning his task*! He must, however, take a coordinating role in that activity (dividing the work, sending reminders about tasks to be done, make sure everything comes together, etc.), and be able to answer most questions on that topic during the evaluation. The following roles will be assigned round-robin:

Design Coordinator The design coordinator coordinates making the design of your software.

Testing Coordinator The testing coordinator coordinates the planning, designing, and writing of the tests for the software.

Domain Coordinator The domain coordinator coordinates the maintenance of the domain model.

As already mentioned, the goal of these roles is to make every team member participate in all aspects of the development of your system. **During each presentation or demo, every team member must be able to explain the**

used domain model, the design of the system, and the functioning of your test suite.

2.2 Iterations

The project is divided into 3 iterations. In the first iteration, you will implement the base functionality of the software. In subsequent iterations, new functionality will be added and/or existing functionality will be changed.

2.3 The Software

The focus of this course is on the *quality* (maintainability, extensibility, stability, readability,...) of the software you write. We expect you to use the development process and the techniques that are taught in this course. One of the most important concepts are the General Responsibility Assignment Software Principles (GRASP). These allow you to talk and reason about an object oriented design. **You should be able to explain all your design decisions in terms of GRASP**.

You are required to provide class and method documentation as taught in previous courses (e.g. the OGP course), as appropriate. When designing and implementing your system, you should use a *defensive programming style*. This means that the *client* of the public interface of a class cannot bring the objects of that class, or objects of connected classes, into an inconsistent state.

Unless explicitly stated in the assignment, you do not have to take into account persistent storage, security, multi-threading, and networking. If you have doubts about other non-functional concerns, please ask your advisor.

2.4 Testing

All functionality of the software should be tested. **For each use case, there should be a dedicated scenario test class.** For each use case flow, there should be at least one test method that tests the flow. Make sure you group your test code per step in the use case flow, indicating the step in comments (e.g. `// Step 4b`). Scenario tests should not only cover success scenarios, but also negative scenarios, i.e., whether illegal input is handled defensively and exceptions are thrown as documented. You determine to which extent you use unit testing. The testing coordinator briefly motivates the choice during the evaluation of the iteration.

Tests should have good coverage, i.e. a testing strategy that leaves large portions of a software system untested is of low value. If your IDE's test coverage functionality reports that only 60% of your code is covered by tests, this indicates there may be a serious problem with (the execution of) your testing strategy. However, be careful when drawing conclusions from both reported high coverage and reported low coverage (and understand why you should be careful). The testing coordinator is expected to briefly report the results of running the test suite with coverage tracking during the evaluation of the iteration.

2.5 UML Tools

There are many tools available to create UML diagrams depicting your design. You are free to use any of these as long as it produces correct UML. One of these UML tools is Visual Paradigm. You can find a link to a website where you can download the program and find the license key. However, we recommend UMLet for its simplicity.

2.6 What You Should Hand In

Exactly *one* person of the team hands in a ZIP-archive via Toledo. The archive contains the items below and follows the structure defined below. **Make sure that you use the prescribed directory names.**

- directory `groupXX` (where XX is your group number (e.g. 01, 12, ...))
  - `doc`: a folder containing, for each API you define in your system, a subfolder with the Javadoc for that API. For example, if you implemented a library of GUI components, include a subfolder with the Javadoc for this library's API.
  - `diagrams`: a folder containing UML diagrams that describe your design (at least one structural overview of your entire design, and sufficient detailed structural and behavioural diagrams to illustrate every use case)
  - `src`: your system's source code
  - `system.jar`: your system's compiled executable
  - `design.pdf` (optional): a document that clarifies your design and the main decisions; this document can be a prose text or a slide deck, or any other form that you deem appropriate.

When including your source code into the archive, make sure to *not include files from your version control system.* Make sure you choose relevant file names for your analysis and design diagrams (e.g. `SSDsomeOperation.png`). You do **not** have to include the project file of your UML tool, only the exported diagrams. We should be able to start your system by executing the JAR file with the following command: `java -jar system.jar`.

Needless to say, the general rule that anything submitted by a student or group of students must have been authored exclusively by that student or group of students, and that accepting help from third parties constitutes exam fraud, applies here.

2.6.1 Late Submission Policy

If the zip file is submitted N minutes late, with $0 \leq N \leq 240$, the score for all team members is scaled by $(240 - N)/240$ for that iteration. For example, if your solution is submitted 30 minutes late, the score is scaled by 87.5%. So the maximum score for an iteration for which you can earn 4 points is reduced to 3.5. If the zip file is submitted more than 4 hours late, the score for all team members is 0 for that iteration.

2.6.2 When Toledo Fails

If the Toledo website is down – and *only* if Toledo is down – at the time of the deadline, submit your solution by e-mailing the ZIP-archive to your advisor. The timestamp of the departmental e-mail server counts as your submission time.

2.7 Evaluation

After iteration 1, and again after iteration 2, there will be an intermediate evaluation of your solution. An intermediate evaluation lasts 35 minutes and consists of: a presentation about the design and the testing approach, accompanied by a demo of the system and the test suite.

The intermediate evaluation of an iteration will cover only the part of the software that was developed during that iteration. Before the final exam, the *entire* project will be evaluated. It is your own responsibility to process the feedback, and discuss the results with your advisor.

The evaluation of an iteration is planned in the week after that iteration. Immediately after the evaluation is done, you mail the PDF file of your presentation to Prof. Bart Jacobs & Tom Holvoet and to your advisor.

2.7.1 Presentation Of The Current Iteration

The main part of the presentation should cover the design. The motivation of your design decisions *must* be explained in terms of GRASP principles. Use the appropriate design diagrams to illustrate how the most important parts of your software work. Your presentation should cover the following elements. Note that these are not necessarily all separate sections in the presentation.

1. A discussion of the high level design of the software (use GRASP patterns). Give a rationale for all the important design decisions your team has made.

2. A more detailed discussion of the parts that you think are the most interesting in terms of design (use GRASP patterns). Again we expect a rationale here for the important design decisions.

3. A discussion of the testing approach used in the current iteration.

4. An overview of the project management. Give an approximation of how many hours each team member worked. Use the following categories: group work, individual work, and study (excluding the classes and exercise sessions). In addition, insert a slide that describes the roles of the team members of the current iteration, and the roles for the next iteration. Note that these slides do not have to be presented, but we need the information.

Your presentation should not consist of slides filled with text, but of slides with clear design diagrams and keywords or a few short sentences. The goal of giving a presentation is to communicate a message, not to write a novel. All design diagrams should be *clearly readable* and use the correct UML notation. It is therefore typically a bad idea to create a single class diagram with all information. Instead, you can for example use an overview class diagram with only the most

important classes, and use more detailed class diagrams to document specific parts of the system. Similarly, use appropriate interaction diagrams to illustrate the working of the most important (or complex) parts of the system.

2.8 Peer/Self-assessment

In order for you to critically reflect upon the contribution of each team member, you are asked to perform a peer/self-assessment within your team. For each team member (including yourself) and for each of the criteria below, you must give a score on the following scale: *poor/lacking/adequate/good/excellent*. The criteria to be used are:

- Design skills (use of GRASP and DESIGN patterns, ...)
- Coding skills (correctness, defensive programming, documentation,...)
- Testing skills (approach, test suite, coverage, ...)
- Collaboration (teamwork, communication, commitment)

In addition to the scores themselves, we expect you to briefly explain for each of the criteria why you have given these particular scores to each of the team members. The total length of your evaluation should not exceed 1 page.

Please be fair and to the point. Your team members will not have access to your evaluation report. If the reports reveal significant problems, the project advisor may discuss these issues with you and/or your team. Please note that your score for this course will be based on the quality of the work that has been delivered, and not on how you are rated by your other team members.

Submit your peer/self-assessment by e-mail to both Prof. Bart Jacobs & Tom Holvoet and your project advisor, using the following subject: **[SWOP] peer-/self-assessment of group $groupnumber$ by $firstname$ $lastname$**.

2.9 Deadlines

- The deadline for handing in the ZIP-archive on Toledo is **23 April, 2021, 3:30pm**.
- The deadline for submitting your peer/self-assessment is **25 April, 2021, 6pm**, by e-mail to both your project advisor and Prof. Bart Jacobs & Tom Holvoet.

3 Browsr

In this iteration, you will extend Browsr with support for `form` *elements,* `input type="text"` *elements,* `input type="submit"` *elements, a Bookmarks bar, and two dialog screens: the Save As dialog screen and the Add Bookmark dialog screen.*

Develop Browsr, a very incomplete web browser. It only needs to support the following HTML elements: `table`, `tr`, `td`, `a`, `form`, and `input`. It only needs to support four attributes: the `a` element's `href` attribute, the `input` element's `type` and `name` attributes, and the `form` element's `action` attribute. It only needs to

support the values `text` and `submit` for the `input` element's `type` attribute. It need not support the `name` attribute on `input` elements that specify value `submit` for the `type` attribute.

Specifically, a Browsr document is a *content span*. A content span is either text, or an `a` element (a hyperlink) containing text, or a `table` element, or a `form` element, or an `input` element. A `table` element contains a sequence of zero or more `tr` elements (table rows). A `tr` element contains a sequence of zero or more `td` elements (table cells with table data). A `td` element contains a content span. A `form` element contains a content span, but it does not directly or indirectly contain nested `form` elements. An `input` element does not have an end tag, so it does not have any content. An `input type="text"` element allows the user to enter text. An `input type="submit"` element is shown as a button containing the text "Submit".

This means table cells can contain nested tables. It also means Browsr does not support hyperlinks interspersed with text.

You are not allowed to use any existing HTML parsing or generic parsing libraries or tools.

To load HTML documents from URLs, just use `java.net.URL.openStream`. You are not allowed to use any other URL loading libraries. Your system can load URLs synchronously; i.e. it is okay if your system freezes while the page is loading.

Rules for rendering (i.e. displaying) a document:

- Hyperlinks must be shown in blue and underlined
- Your system need not break lines of text that are too long to fit on one line
- Tables must be layed out as follows:
  - The width of a column equals the maximum of the widths of the cells in that column
  - The height of a row equals the maximum of the heights of the cells in that row
  - Within a cell, the contents are left/top justified
- Within the document as a whole, the contents are left/top justified
- The width of a text input field is fixed (i.e. independent of the amount of text entered by the user). It shows the text entered by the user inside a box. You need not support the case where the text entered by the user does not fit inside the text box.

Your system shall show a single window whose contents consist of an address bar, a Bookmarks bar, and a document area. The document area shows the HTML document loaded from the URL entered into the address bar. Clicking a hyperlink causes the system to navigate to the URL obtained by composing (using the two-argument `java.net.URL` constructor) the existing URL and the value of the `href` attribute of the hyperlink.

The Bookmarks bar shows the current bookmarks, side by side, as hyperlinks. If the user clicks a bookmark, it is dealt with just like any other hyperlink. The

initial list of bookmarks is unspecified. The user can add a bookmark by pressing Ctrl+D. This shows the Add Bookmark dialog screen, with a text input field for the name of the bookmark, preceded by a label "Name", and a text input field for the URL of the bookmark, preceded by a label "URL", with the current URL pre-filled, and two buttons: an "Add Bookmark" button and a "Cancel" button. Clicking the Add Bookmark button adds the bookmark to the end of the list of bookmarks and closes the dialog screen; clicking the Cancel button simply closes the dialog screen.

The user can save the current document as a `.html` file by pressing Ctrl+S. This shows the Save As dialog screen, with a text input field for the filename, preceded by a label "File name", and two buttons: a "Save" button and a "Cancel" button. Clicking the "Save" button creates a file with the given name and writes the currently shown HTML document to it, and closes the dialog screen. Clicking the "Cancel" button simply closes the dialog screen.

A dialog screen covers the entire CanvasWindow client area. That is, while a dialog screen is shown, the regular contents of the CanvasWindow client area (i.e. the Address bar, the Bookmark bar, and the document area) are not visible. While a dialog screen is shown, pressing Ctrl+D or Ctrl+S does not have any effect. After a dialog screen is closed, the original state of the CanvasWindow client area is restored, including the state of the text input fields in the document area. That is, any text entered by the user into any text input fields before they opened the dialog screen will still be present after they close the dialog screen.

Clicking the address bar, a text input field corresponding to an `input type="text"` element, or a text input field in a dialog screen causes the text input field to receive keyboard focus. When the text input field has keyboard focus, you shall show an insertion point (i.e. a "text cursor"). Typing characters causes the characters to be inserted at the insertion point; the Delete, Backspace, Left, Right, Home, End, Shift-Left, Shift-Right, Shift-Home, and Shift-End keys shall behave as usual. Typing characters while text is selected replaces the selected text. When clicking the text input field, initially all text is selected. Selected text shall be shown on a blue background. You need not support scrolling in a text input field or in the document area.

Hitting the Enter key or clicking outside the address bar causes the address bar to lose focus and causes the system to navigate to the URL given by the contents of the address bar. Hitting the Escape key causes the text that was present before the address bar gained keyboard focus to be restored, and causes the address bar to lose focus.

Clicking a Submit button causes the system to navigate to the URL obtained by composing the current URL with the URL specified in the enclosing `form` element's `action` attribute, followed by a question mark and the URL-encoded names (as can be obtained using class `java.net.URLEncoder` with the UTF-8 character encoding) and values of the `input type="text"` elements contained within the `form` element. If a Submit button is not contained within a form, clicking it does nothing.

Consider the example document in Fig. 1, loaded from URL

`https://people.cs.kuleuven.be/~bart.jacobs/swop/browsrformtest.html`

If the user enters the characters `'s a` into the first text input field, and the

```
<form action="browsrformactiontest.php">
  <table>
    <tr><td>List words from the Woordenlijst Nederlandse Taal
    <tr><td>
      <table>
        <tr>
          <td>Starts with:
          <td><input type="text" name="starts_with">
        <tr>
          <td>Max. results:
          <td><input type="text" name="max_nb_results">
      </table>
    <tr><td><input type="submit">
  </table>
</form>
```

Figure 1: An example Browsr document

characters 10 into the second text input field, and then clicks the Submit button, the system shall navigate to the URL

```
https://people.cs.kuleuven.be/~bart.jacobs/swop/
    browsrformactiontest.php?starts_with=%27s+a&max_nb_results=10
```

In case of an error parsing the URL, loading the document, or parsing the document, an error document shall be shown in the document area. When Browsr starts, a welcome document shall be shown in the document area.

The visual representation of Submit buttons and of the buttons in dialog screens shall be such that users shall clearly recognize them as being buttons, by (for example) showing their label within a rounded rectangle, against a darker background color. Also, the action corresponding to a button shall occur only when the mouse button is released after being pressed while the mouse cursor was over the button. While the mouse button is down, the button shall be shown differently to indicate that the button is being pressed. (We leave unspecified what happens if the mouse cursor leaves the button while the mouse button is still pressed.)

Figure 2 shows the domain model of Browsr.

Clearly, at this level of abstraction, the problem domain is quite simple. The main challenge of this assignment, then, lies not in the design of the domain layer of your system, but in the design of the user interface layer.

4 Use Cases

Figure 3 shows the use case diagram for Browsr. The remainder of this section describes the use cases in detail.

Notes:

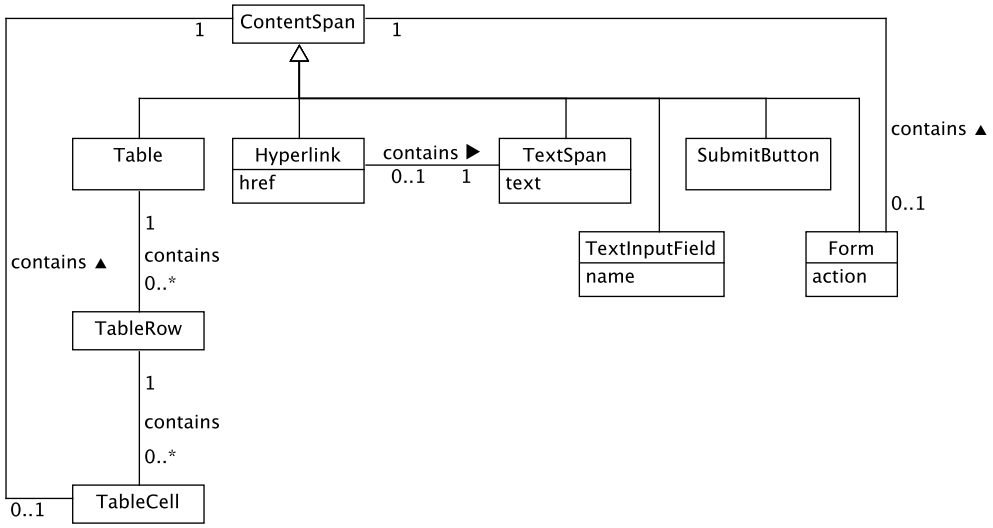

Figure 2: Domain model

- A system's requirements can be specified at various levels of abstraction with respect to the nature of the interface between the system and its environment (e.g. a human user). Often the specification abstracts over the nature of the interface to focus on the aspects that are most relevant to the usefulness of the system. However, in this assignment, since the main challenge concerns the design of the code that implements the user interface, we specify the interaction with the user in unusually specific detail.
- While the tool you develop should be functional, the user interface need not be of the level of "finish" that would be expected of a commercial product. For example:
  - The text cursor need not blink.
  - You need not support scrolling the window if the information does not fit into the window.
  - You need not (in this iteration) provide a menu, Save functionality, printing functionality, etc.

4.1 Use Case: Activate Hyperlink

Main Success Scenario:

1. The user clicks a hyperlink in the document area.
2. The system composes the hyperlink's `href` attribute value with the referring document's URL to obtain the full URL for the document to

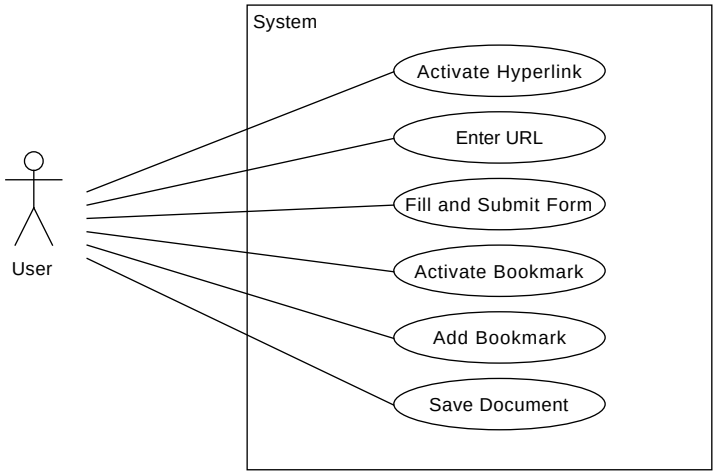

Figure 3: Use case diagram for Browsr.

be loaded. It loads the document and shows it in the document area. It also updates the Address bar to show the document's full URL.

Extensions:

2a. The URL is malformed, loading the document fails, or parsing the document fails.

1. The system shows an error document in the document area.

4.2 Use Case: Enter URL

Main Success Scenario:

1. The user clicks the Address bar.
2. The system indicates that the Address bar has focus, and shows the entire contents of the Address bar as selected.
3. The user edits the contents of the Address bar by pressing Backspace, Delete, Left, Right, Home, End, Shift-Left, Shift-Right, Shift-Home, and Shift-End to move the insertion point and select text, and character keys to replace the selection with the entered character or insert the character at the insertion point.
4. The system shows the updated contents as they are being edited.
5. The user presses Enter or clicks outside the Address bar to finish editing the contents.
6. The system loads the document and shows it in the document area.

Extensions:

5a. The user presses Escape to cancel editing the contents of the Address bar.

1. The contents are changed back to the value that was active when the Address bar received focus.

6a. The URL is malformed, loading the document fails, or parsing the document fails.

1. The system shows an error document in the document area.

4.3 Use Case: Fill and Submit Form

Precondition: The current document defines at least one form.

Main Success Scenario:

1. The user clicks a text input field in the document area.
2. The system indicates that the text input field has focus, and shows the entire contents of the text input field as selected.
3. The user edits the contents of the text input field by pressing Backspace, Delete, Left, Right, Home, End, Shift-Left, Shift-Right, Shift-Home, and Shift-End to move the insertion point and select text, and character keys to replace the selection with the entered character or insert the character at the insertion point.
4. The system shows the updated contents as they are being edited.

   The user repeats Steps 1–4 until they are done filling the form.

5. The user clicks the form's Submit button.
6. The system loads the document at the URL obtained by composing the current document's URL with the URL specified in the `form` element's `action` attribute, followed by a question mark and the names and values of the form's text input fields, and shows it in the document area. It shows the new document's URL in the Address bar.

Extensions:

6a. The URL is malformed, loading the document fails, or parsing the document fails.

1. The system shows an error document in the document area.

4.4 Use Case: Activate Bookmark

Main Success Scenario:

1. The user clicks a bookmark in the Bookmark bar.
2. The system loads the document at the bookmark's URL and shows it in the document area. It also updates the Address bar to show the document's URL.

Extensions:

2a. The URL is malformed, loading the document fails, or parsing the document fails.

1. The system shows an error document in the document area.

4.5 Use Case: Add Bookmark

Main Success Scenario:

1. The user presses Ctrl+D.
2. The system shows a dialog screen titled "Add Bookmark" that covers the CanvasWindow client area. The dialog screen contains a text input field for the name of the new bookmark, preceded by the label "Name", and a text input field for the URL of the new bookmark, preceded by the label "URL". It also contains two buttons, labelled "Add Bookmark" and "Cancel", respectively.
3. The user clicks one of the text input fields.
4. The system indicates that the text input field has focus, and shows the entire contents of the text input field as selected.
5. The user edits the contents of the text input field by pressing Backspace, Delete, Left, Right, Home, End, Shift-Left, Shift-Right, Shift-Home, and Shift-End to move the insertion point and select text, and character keys to replace the selection with the entered character or insert the character at the insertion point.
6. The system shows the updated contents as they are being edited.

   The user repeats Steps 3–6 until they are done filling the dialog screen.

7. The user clicks the Add Bookmark button.
8. The system adds the bookmark to the list of bookmarks shown in the Bookmark bar and closes the dialog screen.

Extensions:

7a. The user clicks the Cancel button.

1. The system closes the dialog screen without adding a new bookmark to the list of bookmarks.

4.6 Use Case: Save Document

Main Success Scenario:

1. The user presses Ctrl+S.
2. The system shows a dialog screen titled "Save As" that covers the CanvasWindow client area. The dialog screen contains a text input field for the name of the `.html` file to be created, preceded by the label "File name". It also contains two buttons, labelled "Save" and "Cancel", respectively.

3. The user clicks the text input field.
4. The system indicates that the text input field has focus, and shows the entire contents of the text input field as selected.
5. The user edits the contents of the text input field by pressing Backspace, Delete, Left, Right, Home, End, Shift-Left, Shift-Right, Shift-Home, and Shift-End to move the insertion point and select text, and character keys to replace the selection with the entered character or insert the character at the insertion point.
6. The system shows the updated contents as they are being edited.

   The user repeats Steps 3–6 until they are done filling the dialog screen.

7. The user clicks the Save button.
8. The system creates a file with the specified name, writes the current document into it, and closes the dialog screen.

Extensions:

7a. The user clicks the Cancel button.

1. The system closes the dialog screen without creating a file.

5 Implementation

The main design challenge is to design the presentation layer of your system. You will have to develop your own GUI framework. You have to develop your system in Java, but you are not allowed to use an existing Java GUI framework, such as AWT, Swing, or SWT. You have to use the CanvasWindow class attached. You are not allowed to modify it. You are allowed to use the AWT elements (such as `FontMetrics` and `Graphics`) that are necessary to use this class, but you cannot use the AWT or Swing component hierarchies. Good luck!

The SWOP Team members