CS 520
Homework 1
Refactoring a Static Website Generator

Due: **Tuesday, September 29, 2026, 9:00 AM EDT**. Students may work together on this assignment, but each student must submit their own assignment. If you work with others, you must clearly state the names of the students you worked with. Your responses to questions must be your own, but there is no penalty for working together. You are allowed to use AI tools to assist with this assignment (in fact, it is expected).

Late assignments will not be accepted without **prior** permission.

This assignment takes time. Do not wait until the last minute to start the assignment.

Overview and goal

The primary goal of this assignment is to become familiar with AI tools used to edit code and integrate them into a code-editing environment. The assignment accomplishes this in the context of working with an existing non-trivial code base that has several design and testability flaws. This mirrors a common setting in practice where you will join a team that already has a code base to work with and that code base is not in the best shape but still needs to be extended and maintained. You will work to improve the design of the program without changing its behavior (known as refactoring). You will not fix all issues but will make some improvements to the code base. You will spend most of the time reading existing code and looking for design problems, and using AI to navigate and improve the code. For changes you make (whether on your own or with the use of AI), you will justify your changes.

Acknowledgement

This assignment builds on an assignment originally designed by Christian Kästner, Claire Le Goues, and colleagues at Carnegie Mellon University.

Getting started

Your first step is to set up your tools. You will need a GitHub account, and IDE, git, maven, and other tools installed on your machine, and access to an LLM. We will not tell you which tools to use (though we'll make suggestions) or how to install them. Figuring this out is part of your assignment. We suggest you install VS Code (`https://code.visualstudio.com/`) as your IDE. If you chose to use AI (which we recommend you do), you can use any AI coding tool you want. GitHub, VS Code, Maven, etc., are all available for free, and plenty of free AI coding tools exist that can be used to complete this assignment. However, some AI coding tools offer better integration with coding IDEs (such as VS Code) for paid versions. GitHub Copilot, which has integration with VS Code, is free for students. Though we cannot provide support for getting the free Copilot, you can follow these instructions to get it:
`https://docs.github.com/en/copilot/how-tos/copilot-on-github/set-up-copilot/enable-copilot/set-up-for-students`

Log into your GitHub account (a free account is more than sufficient) and create a new private repository called `UMACS520F26HW1`. **Make sure the repository is private!**

Your repository MUST remain private both during the semester and afterwards, in perpetuity. Making your repository public is a violation of the UMass Academic Honesty policy, which can lead to a failing grade for the course. This policy is enforceable even after the semester ends and after you graduate. You may delete the repository after the end of the semester, or you may keep it private, but you should never make it public. We will be running scripts to periodically check and ensure all repositories remain private.

Share the repository with `UMassCS520` by going to Settings → Collaborators → Add people.

Download the starter code here: https://people.cs.umass.edu/~brun/class/2026Fall/CS520/hw1/hw1.zip

Clone your own repository locally, unzip the files from `hw1.zip` into that folder, add **all** the files to the repository, and make sure you can `push` them back to GitHub. Verify via GitHub's web interface that all the files are there.

You will need to figure out how to deploy the project locally, how to run the tests (using Maven's `mvn test` command), and so on. Use AI to help you perform these tasks. If you are using Windows, you will likely need to rely on WSL: Windows Subsystem for Linux (https://learn.microsoft.com/en-us/windows/wsl/install). Again, use AI to help you understand if it is needed and how to set it up. Class staff cannot provide technical support for setting up your environment, but AI is great at providing such help.

As you work on the assignment, make sure to commit all your changes to your git repo and push them to GitHub.

Context

Static website generators. Static website generators are tools that generate static web pages (https://en.wikipedia.org/wiki/Static_web_page) from various inputs. They allow users to create and customize web pages without writing HTML code directly.

Static web sites have become very popular with GitHub Pages (https://pages.github.com/). By default, GitHub uses Jekyll (https://jekyllrb.com/) static web site generator, but many other exist and it can be easy to write a custom one.

The project. In this homework, you take over a custom static website generator written to support a webpage for student clubs, that contain articles and events. The code is kind of functional, but not quite finished, and it is poorly designed in many ways. The original authors wanted to introduce events and a calendar and have written much of the code for it, but the complexity got out of hand so it was never finished. Additionally, no one has worked on making the produced website look nice yet. Documentation is fairly sparse and there are no automated tests, just an example project.

The project roughly works like this: Everything is organized as a project (top-level directory). A project has a title and is owned by an organization, configured in a YAML file. Each directory represents an event or an article. Events and articles can have other events or articles within them. Events or articles can have pictures and videos. They can be organized by topics (tags). Events have a name and a start and end date. The front page lists recent article/events and a calendar of upcoming events.

The project has three stages: a parser loads data (Markdown (https://daringfireball.net/projects/markdown/) files, YAML (https://en.wikipedia.org/wiki/YAML) files, etc.) from the disk (`parser` directory), the loaded data is represented as internal data structures (`project` directory), and finally, the content is written as HTML files (`rendering` directory) using a template engine. A command-line interface (CLI) coordinates all of this and can print summaries and statistics.

Refactoring. Refactoring is the improvement of code without changing its functionality. For example, to address the anti-pattern of a god class (https://wiki.c2.com/?GodClass) and improve cohesion of the implementation, a developer may decide to move methods from one class to another, and update all calls to the method. This change improves design but does not modify functionality; the same method will still

be called, just in a different place. It is usually a good idea to commit regularly as you refactor and make changes. If something goes wrong, you can always go back to the last commit.

Relationship to real-world projects. When joining a software engineering team in industry you almost always will join an existing project rather than starting from scratch. Most projects will be much much bigger than this one and documentation is rarely, if ever, great. The amount of testing done also differs wildly between organizations. Reading existing code and understanding a subsystem without needing to know everything about every detail of the code is an important and very common skill. Research shows that software engineers in practice often spend much more than 50 percent of their time on understanding existing code rather than writing new code. Taking notes when reading the code or sketching diagrams of the structures in a program (usually informally and incompletely) is quite common to get an overview and understanding how pieces of a system fit together.

Real-world code often has design problems, where prior code was poorly written or the problem changed and the software has only been patched as necessary. Teams rarely have the luxury of rewriting an entire system from scratch. Additionally, no one wants to do massive code changes out of fear of breaking existing functionality. Teams often even have a hard time getting permission to do minor improvements when management thinks they could work on new code instead.

In this assignment, we intentionally expect you to be very explicit about arguments about why certain designs are bad and how they can be improved. In addition, we require you to create a partial object model for one of your fixes. While you may not write this all down in practice, sometimes you may need to make similar arguments to colleagues to argue why a specific change is beneficial (when it is always easier to leave the code as is). We provide guidance for identifying certain kinds of design problems. But AI is your friend here and can help you reason about code and identify problems.

You will do multiple small design improvements and one slightly larger redesign to solve a specific problem (SubSubSubSubSubArticles). These design improvements will only refactor code, without adding new functionality.

Design Problems

In this assignment, you will address one instance of each of the following design problems:

1. Code Duplication — duplicate code that could be abstracted and reused using methods, inheritance, delegation, or design patterns.
2. Coupling — unnecessary coupling.
3. Cohesion — objects, classes, or methods that severely violate cohesion (e.g., god class).
4. SubSubSubSubSubArticles — the way nested articles are represented in the implementation is problematic.

Testing Problems

This code comes with tests that you can run with `mvn test` using Apache Maven (https://maven.apache.org/). However, the testing infrastructure is tiny, and each of the tests, or the code it tests, violates one or more of the SOLID principles (https://simple.wikipedia.org/wiki/SOLID_(object-oriented_design)). We have listed the principles for you below.

1. Single responsibility — Each test should only have a single reason to fail.
2. Open-Closed Principle — Designs should be open for extension, but closed for modification. You should be able to extend what a class does without changing its source code.

3. Liskov Substitution Principle — Each superclass should be replaceable with an object of its subclasses without breaking the application.

4. Interface Segregation Principle — Clients should not be forced to depend on interfaces that they do not use.

5. Dependency Inversion Principle — High-level classes should not depend on low-level classes.

Your Task

We ask you to perform the following tasks:

Improve design.

Incrementally and in small steps improve the design and implementation of the project to make it easier to maintain and extend in the future. You do not need to implement new functionality or add new tests for this part of the assignment.

Step 1: Look for a design problem in the implementation that fits one of the design problems listed above. Spend time getting an overview of the code and understanding parts of it. Use AI to help you, but you need to be the one who understands the code, not the AI.

Step 2: Create a GitHub issue using our provided template with the design problem in the title (e.g., "Fix coupling problem"). The issue text should point out a specific problem at a specific location in the code and explains why the specific code is problematic (e.g., why the coupling is bad in this specific case).

Step 3: Fix the design problem and commit the fix with a meaningful commit message.

Step 4: Close the issue with a message that explains how you fixed the issue and link the issue to the commit that fixed it. A fix may address multiple problems, and the same commit can be linked in multiple issues.

Step 5: Repeat the steps above until you have addressed all 4 design problems.

We expect 4 closed issues with links to one or more commits after the design improvement.

Improve testing.

Improve the tests included in the repository. This may require modifying the tests and/or modifying the code itself.

Step 1: For each test provided, identify which SOLID principle(s) is violated by the test itself or the code that it tests. A test may violate multiple principles, and all principles are not necessarily violated.

Step 2: Create a GitHub issue with the name of the test in the title (e.g., "Fix ArticleTest.java"). The issue text should identify which SOLID principle or principles is violated by the test, and explain why it violates it and why this is undesirable. Use the same GitHub issue template as above.

Step 3: Fix the test and/or the code problem with a meaningful commit message, ensuring the same functionality is tested and nothing more. You may need to add additional tests or files.

Step 4: Close the issue with a message that explains how you fixed the issue and link in the issue to the commit that fixed it. A fix may address both a design problem and a testing problem, and can be linked to multiple issues.

Step 5: Repeat the steps for all 4 of the included tests.

We expect 4 closed issues, one for each of the tests, with links to one or more commits for test improvement.

Document structure change For the "SubSubSubSubSubArticles" problem only, create a partial object model of the system after your fix. It should only involve the classes related to articles as needed to explain your fix. Make sure the diagram is consistent with your final implemented solution.

Commit the diagram as `sub-articles.pdf` in the root directory of your repository.

Hints for design problems

The code has many problems, and several methods are in the wrong place. Some methods clearly do a lot more work than they should and depend on knowing details of other objects. There is a lot of encapsulation violation throughout the code. The way topics are stored is weird. The way sorting is managed seems rather inflexible and repetitive. There is a lot of copy pasted code that could be abstracted, including several methods in several files and entire files. There are lots of `instanceof` checks that are a sign of possibly bad design. Some of these problems may benefit from using a design pattern, but for most problems a design pattern is not needed. Usually, reasoning with design heuristics and design principles is sufficient.

One part of the design seems particularly problematic: The way articles are nested is very hard to extend (design problem "SubSubSubSubSubArticles" above). There is one design pattern we discussed in lecture that would make this design much better, where the same operations can be applied to articles, subarticles, and content of those articles uniformly in a hierarchy. Making this change is nontrivial and will require changes to multiple code fragments where articles and their content is used. You can probably delete the `SubArticle` and `SubSubArticle` classes afterward. You may want to start with a few smaller problems to get familiar with the code base before getting to this one.

Reusing design changes. Note that some design fixes can address to multiple of the design problems above. For example, the fix to "SubSubSubSubSubArticles" will likely address multiple other design problems, too. Hence it may be possible to close multiple issues with the same fix. But your explanations for why the design change you made fixed the issue will most likely need to be different.

Excluded code. Focus primarily on the data structures to represent the project (`project` directory) and how it is used by the `Renderer` and the `CLI`. You can ignore the parser (file `ProjectParser`). We also do not expect you to make any changes to the template engine (in file `TemplateEngine` and the `data` directory and `.hbs` and `.css` files) though you may need to understand what is generated here to understand other parts of the program.

Submitting your work

Always commit and push all your changes to GitHub.

Create a `README` text file and add in the root directory of your repository. Make sure your `README` file contains your name, whom your worked with, and what AI tools you used.

Once you have pushed your final code, `README`, and `sub-articles.pdf`, submit a link to your final commit on Canvas. A link will look like `https://github.com/<username>/<reponame>/commit/<commitid>`. You can get to this link easily when you click on the last commit (above the list of files) in the GitHub web interface. We expect to find (closed) issues in the repository you link to. **Make sure your repository is private!**

Grading

The assignment is worth 110 points. **If you fail to share your GitHub repo with `UMassCS520`, if your repo is not private, or if your repo is missing important files from the assignment, you will get a grade of 0. If your repo becomes public in the future, you will retroactively receive a grade of 0, and your final course grade will be adjusted accordingly.**

We expect to grade the assignment approximately with this rubric:

20pt: The implementation was improved in some form. The improved implementation provides the same functionality as the original implementation. Improvements like supporting deeper nesting of articles or supporting events are okay, but breaking the existing functionality is not.

24pt: For each of the first 3 design problems (code duplication, coupling, cohesion; not "SubSubSubSubSubArticles"):

- 4pt: There is an issue on GitHub that (a) names the design problem in the title, (b) identifies an instance of the problem in the code, and (c) explains why this is a problem. The issue's text demonstrates an understanding of the design problem.
- 4pt: The issue is (a) closed with (b) a description of the fix and (c) a link to the commit(s) that contain the fix so that we can find the fix in the implementation. The change indeed improves the design as described without introducing new obvious design problems.

32pt: For each of the 4 tests in the code:

- 4pt: There is an issue on GitHub that (a) names the test in the title, (b) identifies which of the SOLID principles is violated, and (c) explains why this is a problem. the issue's text demonstrates an understanding of each of the design problems.
- 4pt: The issue is (a) closed with (b) a description of the fix and (c) a link to the commit(s) that contain the fix so that we can find the fix in the implementation. The change indeed improves the testability of the code without introducing new problems.

14pt: For "SubSubSubSubSubArticles": An issue (named "SubSubSubSubSubArticles" or similar) describes why the old implementation was problematic. The issue is closed with a link to the commit(s) that contain the fix and a description of how it was improved in a way that demonstrates an understanding of design reasoning and the used design pattern (if any). The fixed implementation better handles articles and subarticles with less code duplication and better implementations for common functionality such as `getTopics`, `printSize`, and parent/child relationships.

10pt: The repository contains the `README` with your name, partners (if any), and a list of which AI tools you used, and a partial object model in `sub-articles.pdf` in the root directory of your repository that describes the design fix for the "SubSubSubSubSubArticles" problem. It needs only include the classes involved with your fix. The diagram should be consistent with the actual implementation, use suitable notation, and be at the right level of abstraction.

10pt: The git commits are cohesive and have meaningful descriptions that indicate what fix they are part of. The changes did not introduce severe style or readability issues.

Appendix: Technical Hints

Viewing the webpage:

The code can be executed with the provided sample project, which executes most of the functionality of the project. To make this easier, we set up `mvn exec:exec` to execute the program with arguments `-d testProject --list-articles --all --list-events --topics --clean --size`, which executes most code of the project. You can also run the program from within your IDE using a configured `launch.json` with these arguments. These commands generate a directory called `_static` by default that contains all the HTML and CSS files to render the webpage.

When you initialize the starter repository, we recommend that you generate the webpage with the initial code and rename that directory (to e.g. `_static_original`), so you can have a reference to the original webpage. Refer to the Testing Section below for the commands to re-generate the webpage directory as you refactor and ensure the new output is the same as the original. If you begin refactoring without doing so and want a reference, you can check out the initial commit and generate the webpage with the starter code.

To view the webpage, go to the directory and open `index.html`. From VS Code, you can either click on 'Open in Default Browser' or 'Copy Path' and paste that into the address bar of your browser.

Testing:

The project includes automated tests that may be run with `mvn test`. Unfortunately, the tests included are not comprehensive, so shouldn't be relied on to verify that your fixes do not break the functionality of the project. To test this, you can compare the output produced by the program after your changes to the directory generated by the original code. While comparing the output on one example is not guaranteeing that the refactoring is behavior-preserving for all inputs, it provides assurance that nothing obvious was broken. To do this comparison, use a `diff` tool to compare the old generated output with the new one. Similarly you can save the command-line output into a file and compare the output before and after a fix. We recommend to perform the diff with

```
diff -r \
    -I 'class="date">' \
    -I 'class="created-date">' \
    -I "last updated" \
    _static _static_original
```

where `_static_original` is the renamed original directory. This will produce a difference because of the timestamps, but they can be safely ignored. If no functionality was changed, there will be no output from this `diff` command. In addition, opening the `index.html` file for each the of the generated directories in your browser and visually comparing them is a quick way to help confirm the website generates properly.