Joe Grassl

Jacob Morris

Joshua Lansing

**Dungeon Refactoring Documentation**

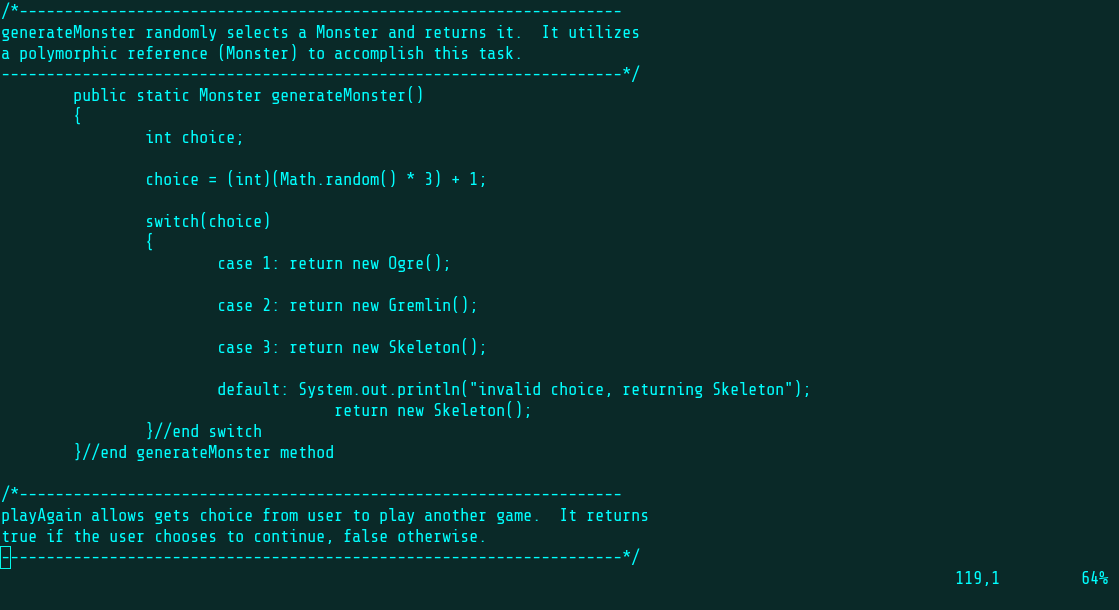
Git repo: https://github.com/delta-plus/dungeon

Refactors are listed here in no particular order.

1. Removed unnecessary comments

The code is riddled with fairly useless comments that describe what the code is doing even when it is quite obvious. Because this is a code smell, these were deleted.

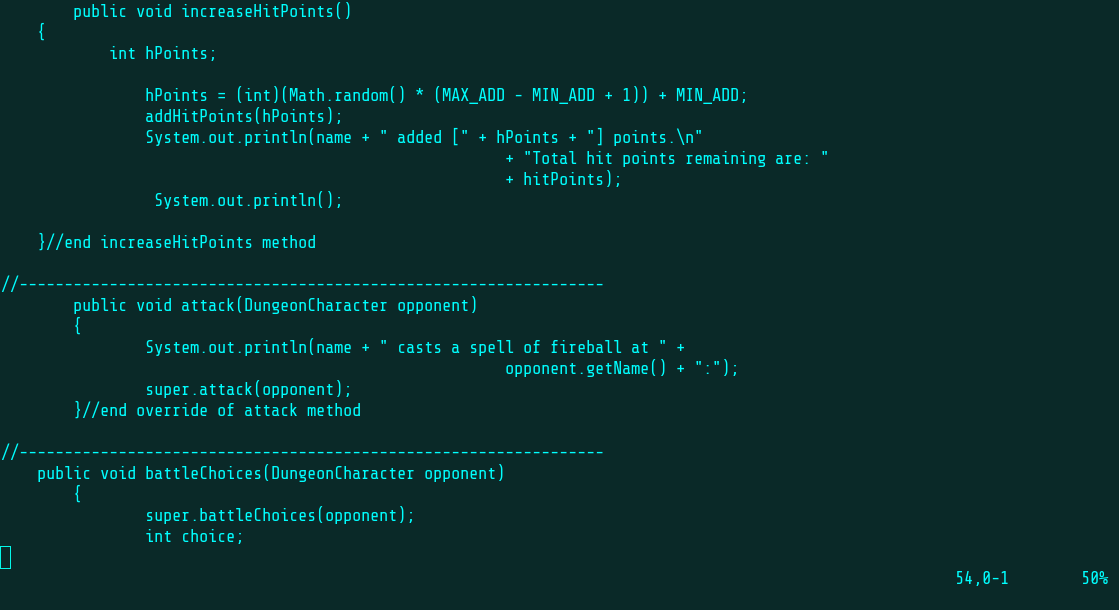
Example of offending code:



Note the equally useless “end generateMonster method” comment. The closing bracket shows this clearly enough by itself.

1. Improved Method and Variable Names

Many of the method and variable names were weird and unintuitive, so they were changed to be more consistent and easier to understand.

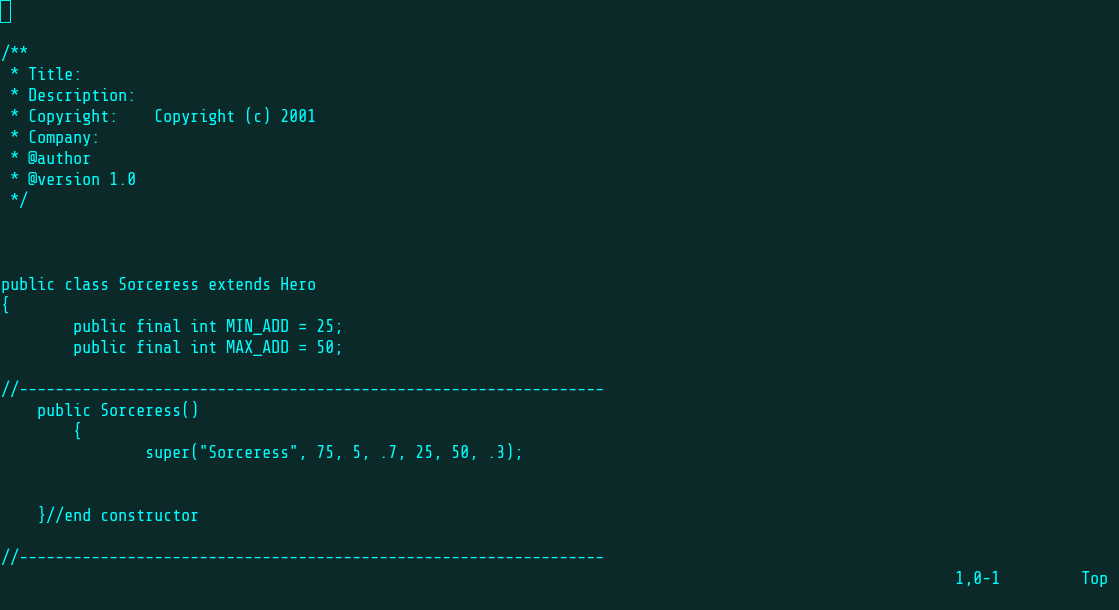
 Example of offending code:

In the image above, the Sorceress’s special ability is called “increaseHitPoints”. This is strange because the Monster class has an ability that is essentially the same but called “heal”. Renaming the Sorceress’s ability to “heal” is therefore more consistent and also just sounds better than the oddly formal “increaseHitPoints”. Also, “hPoints” is never used in any other function to refer to hitpoints, so this was changed to our standard “hitPointChange”. Monster.java’s “healPoints” was similarly modified.

1. Added Files to a Package

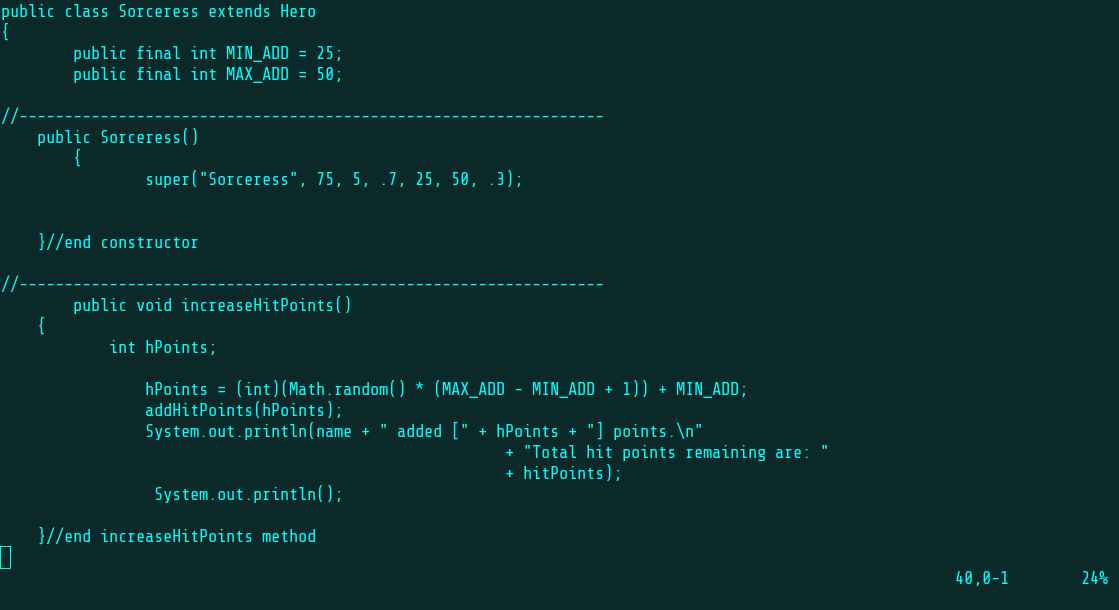
Adding files to a package improves maintainability and general organization of the project.

Originally the files had no package, as seen here:

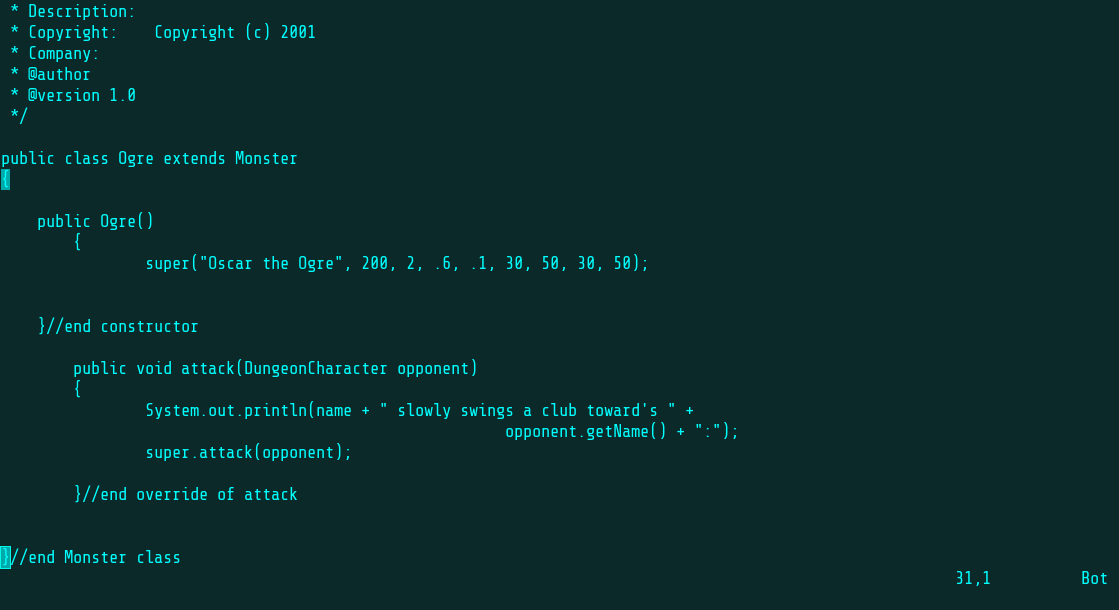


1. Improved General Readability

The indentation and spacing of the project was really inconsistent, so everything was refactored to conform to this style: use tabs, not spaces; curly braces go on the line after the declaration. Typos were also fixed.

 Bad spacing example:

Typo example:

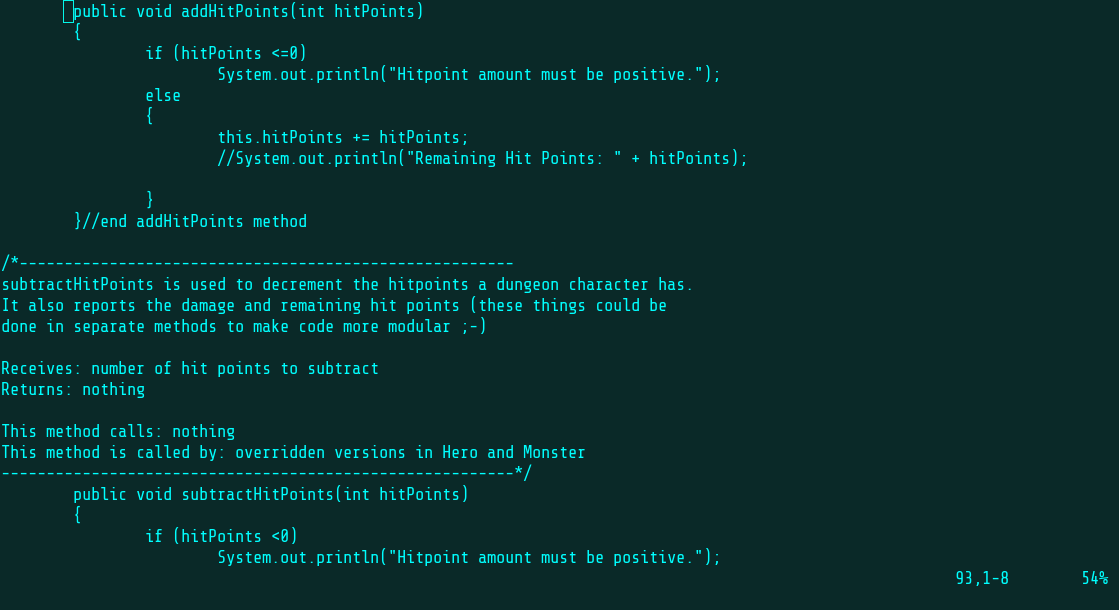


“toward’s” should be “towards”.

1. Combine addHitPoints and subtractHitPoints

Because these methods are so similar, they were combined into a single method called modifyHitPoints that simply does the appropriate thing based on whether the passed integer is positive or negative. This simplifies the implementation of other functions that need to change hitpoints in some way.

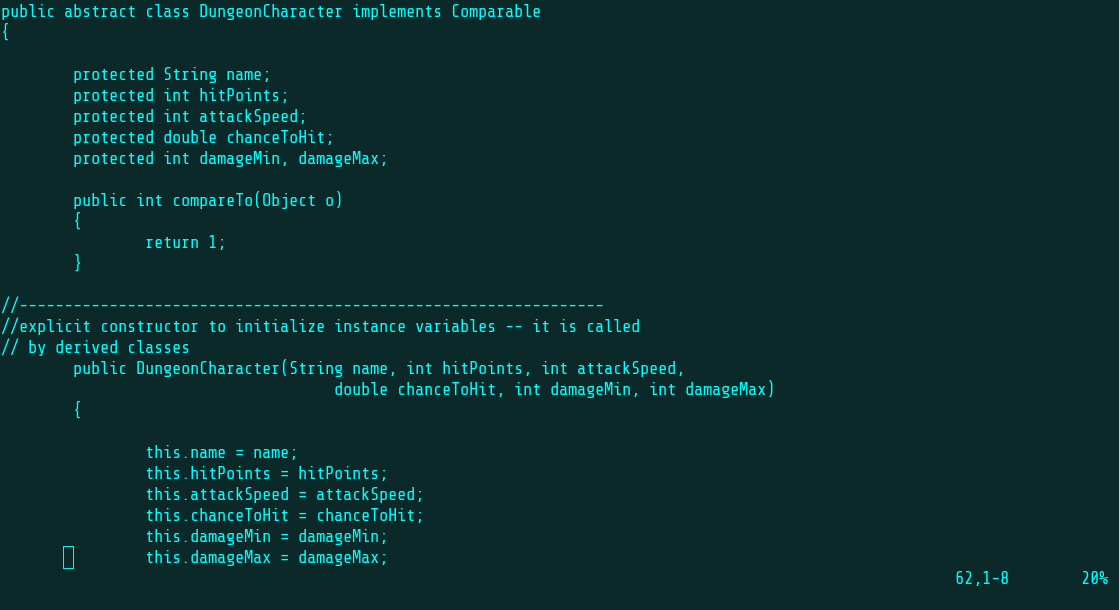
Example of offending code:



1. Got Rid of Comparable Implementation in DungeonCharacter

This was removed because first of all, the characters are never compared anywhere in the game code, and secondly, all it does is return 1 every time.

Example of offending code:

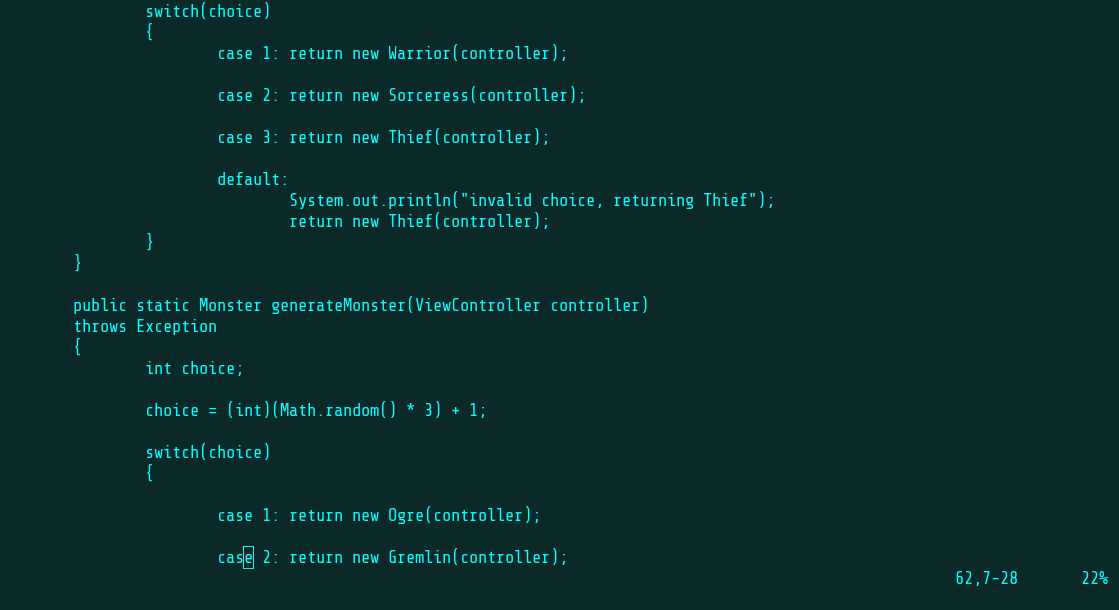


Seriously, what was that supposed to be?

1. Added Hero Factory

Direct calls to Hero subclass constructors were replaced with a Simple Factory pattern. This future-proofs the Hero creation in case more complex constructors are needed later. It also increases organization by decoupling this function from the chooseHero() method.

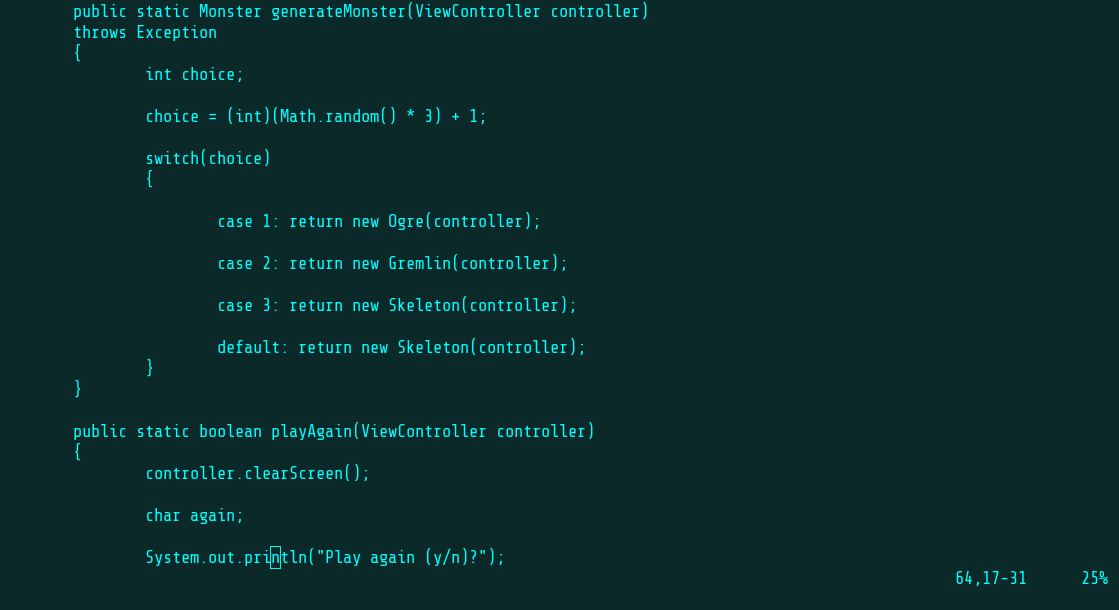
Example of offending code:



1. Added Monster Factory

Direct calls to Monster subclass constructors were replaced with a Simple Factory pattern. This future-proofs the Monster creation in case more complex constructors are needed later. This functionality is further decoupled from the main driver class by placing a createRandomMonster() method in the factory class.

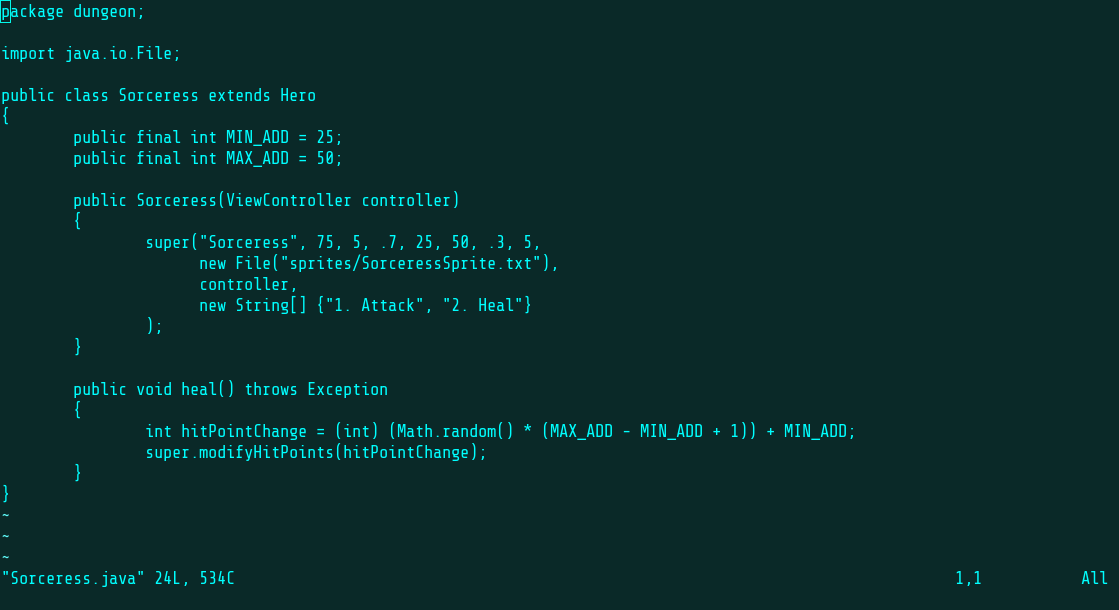
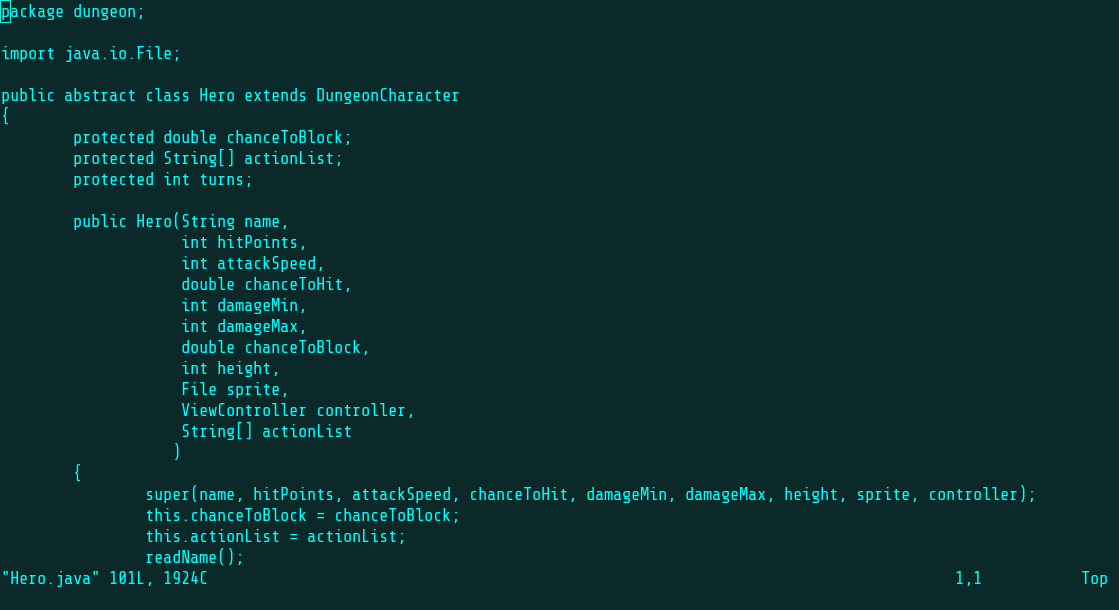
Example of offending code:



1. Replaced Protected and Public Fields with Private Fields

Following the principle of least privilege, all scopes are made as small as possible. Therefore, we changed the protected and public fields to private and implemented getter and setter methods as necessary.

Examples of offending code:



1. Replaced Keyboard Class with Scanner

The Keyboard class was outdated and contained non-standard and deprecated code. We removed this class entirely and just used the common Scanner(System.in) to replace its functionality wherever Keyboard was used.

Example of offending code:

