Coding Standards

GBST requires developers to follow a set of conventions when writing code. This is done to:

- Make code readable. The more readable your code is, the simpler it is for another developer to understand it. This makes it easier to maintain.
- Reduce the size of diffs resulting from code being reformatted without being functionally changed. This is a nightmare for code reviews and merge conflicts.

GBST follows the standard Sun/Oracle Coding Conventions with the addition of a small set of variations and/or clarifications.

General

- Name stuff sensibly. *i* is ok for a loop counter - because it's an accepted convention - but elsewhere your names should make sense without requiring comments.
- Keep your methods - and your classes - short. Methods should only do one thing, and classes should follow the single responsibility principle.
- Don't check in commented-out code. Delete it. We can always get it back from the VCS history. Similarly, try to avoid leaving TODOs and FIXMEs in the code. Delete main methods used for testing before you check in.
- Don't duplicate code. Being in a hurry isn't a good enough excuse.
- Don't be lazy. If you see something broken or poorly implemented, either fix it or create a ticket to get it fixed.
- Don't break the build. Checking in code that is only partially complete is fine (it's encouraged), but at a minimum it must compile and all the tests must pass.

Formatting

- Indentation should use 4 spaces. There should be *no tabs anywhere*.
- Delete all trailing whitespace. Most IDEs can do this via a shortcut.
- We don't enforce a specific line length, but be reasonable about it. You don't need to stick to 80 chars any more, but similarly 200 chars is unwieldy. Use 100 chars if that's your thing, but don't reformat someone else's code just because they chose 120 chars.
- Try to avoid reformatting other people's code. Don't blindly apply your custom code formatter to an entire class when you have modified a single line. It's painful to code review and you make no friends.

Documentation

- In general, try to make your code clear enough that comments are not necessary. Document where required, but keep it clear and succinct. Comments have to be maintained just the same as code does, so limit how much you need to maintain.
- Classes should have at a minimum documentation describing the responsibility of the class.

Logging

- Logging statements should be sentences. Write them in such a way that they actually help people figure out what is going on.
- Assume that anything logged at INFO level will always be logged.
- We use slf4j. Please learn to use parameterised log messages: http://www.slf4j.org/faq.html#logging_performance
- Use DEBUG logging for information that you don't typically need to see but which would not noticeably impact performance if it were enabled (use TRACE for that).
- Use WARN when something goes wrong but you don't expect it to affect the user or which you are able to recover from.
- User ERROR when something bad happens and you can't do anything about it other than record it.

Unit Tests

- All changes to the code should have unit tests that verify the new or modified functionality.
- A unit test should only test one thing.
- Unit tests should be fast (as in milliseconds). Do not put threads to sleep and do not perform timings in unit tests. If you really want to write a performance test, put it in an integration test that can be run separately.
- If you need to start up a Spring context to unit test a class, you're probably doing it wrong.
- Write your classes such that all external dependencies can be replaced with mock versions.