EECS 293
Software Craftsmanship
2020 Spring Semester

Programming Assignment 5

Due at the beginning of your discussion session on

February 18-21, 2019

Reading

In addition to the following topics, the quiz syllabus includes any material covered in the lectures:

- Read the quick reference on routine names (canvas module).
- Read Sections 7 (preamble), 7.1 (except "Hide pointers"), 7.2, 7.5, and 23.4 in Code Complete.

Grading Guidelines

An automatic C (or less) is triggered by:

- Any routine with complexity greater than 4, or by
- Any piece of code that is essentially repeated.

However, a submission that avoids these problems does not necessarily qualify for high quality craftsmanship.

Starting with Programming Assignment 7, an automatic C (or less) will be triggered by improperly named routines.

Programming

In this assignment, you will wrap up the project by completing or revising your previous submissions.

As usual, make all the changes discussed in your discussion section. Additionally, you should refactor your code to make sure that it adopts the principles covered in the reading assignments.

Optimization

The unification algorithm requires to "append the type group of t' to the type group of s'". However, the algorithm would work equally well in the opposite order if the type group of s' were appended to the type group of t'. The main difference between the two ordering is speed: since the second group's type entries must be updated, it is faster to append the smaller group to the first one. Implement an optimization to check for the size of the two type groups before deciding which one is appended to the other one.

String Representation

To display the result of unification, it is convenient to compute a string representation of the type entry's representatives.

First, define an

```
abstract protected String basicRepresentativeString(
                TypeSystem typeSystem)
```

in `AbstractTypeEntry`, which will return a string representation of a `TypeEntry` assuming that the `TypeEntry` happens to be the representative of its group. For `SimpleTypeEntry`, the `basicRepresentativeString` is the `toString`. For `CompoundTypeEntry`, the `basicRepresentativeString` is similar to the `toString`, but the sub-types are represented by their `representativeString` (and not by their `toString`). A challenge in this assignment is to implement `basicRepresentativeString` in `CompoundTypeEntry` while avoiding any code repetition with `toString`.

Define

```
String representativeString(TypeSystem typeSystem)
```

in `TypeEntry` and implement it in `AbstractTypeEntry` so as to return the representative's `basicRepresentativeString`. As usual, `representativeString` should throw a `NullPointerException` if the `typeSystem` is null.

General Considerations

This assignment concludes the type inference project. In the next assignment, we will turn to a different project.

These classes may contain as many auxiliary private and package-private methods as you see fit, and additional package-private helper classes may be defined. However, any modification or addition to public classes and methods must be approved by the instructors at the discussion board.

You should write JUnit tests to make sure that your primary methods work as intended. However, we will revisit testing later on in the course, so extensive testing is not yet recommended. Similarly, your code should have a reasonable number of comments, but documentation is going to be the topic of a future assignment. As a general guideline at this stage of the course, comments and tests should be similar to those accepted in EECS 132.

Discussion Guidelines

The discussion will focus on routines, including but not limited to appropriate routine names.

Submission

Bring a copy to discussion to display on a projector. Additionally, submit an electronic copy of your program to Canvas. In addition to your code, include a README file explaining how to compile and run the code. The code should be handed in a zip, tar.bz2, or tar.gz archive. Archives in 7z cannot be accepted.