EECS 293
Software Craftsmanship
2020 Spring Semester

Programming Assignment 4

Due at the beginning of your discussion session on

February 11-14, 2020

Reading

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

- In chapter 5 of Code Complete:
  - Section 5.1
  - Figure 5-2
  - "Form Consistent Abstractions"
  - "Encapsulate Implementation Details"
  - "Hide Secrets (Information Hiding)"
  - "Keep Coupling Loose"
- Section 19.6 in Code Complete
- Item 64 in Effective Java

Special Grading Guidelines

Starting with this assignment, 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.

Programming

First, 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.

The main topic of this assignment is to implement the unification algorithm. First, implement a package-private method `final void setTypeGroup(TypeEntry typeEntry, TypeGroup typeGroup)` in `TypeSystem` to update the type group associated with the given entry. Then, implement a package-private method `final void append(TypeGroup other)` in `TypeGroup` that appends all the types in the `other` type group to this one and correspondingly updates the group assignment in the type system. Additionally, `append` can change the group representative as follows: if either representative is a non-variable, it becomes the representative. In other words, the representative can be a variable only if both representatives are, at which point there is no choice. The asymmetry is important for the correct functioning of the unification algorithm.

Add to `TypeEntry` a new method `boolean hasEqualUnderlyingType(TypeEntry other)` to ascertain whether the sub-types of a compound types can be unified recursively, at least in principle. In the case of a `SimpleTypeEntry`, it returns false. In the case of a `CompoundTypeEntry`, it returns whether the two types are the same.

The core of the unification algorithm is `public final boolean unify(TypeEntry s, TypeEntry t)` in `TypeSystem`. Its pseudo-code is:

Algorithm unify(s, t):
s′, t′ ← representatives of s and t
if s′ and t′ are the same **then return** true
if s′ and t′ are compound types with the same underlying type **then**
    // The following is optimistic:
    append the type group of t′ to the type group of s′
    for each sub-type of s′ and t′
        return false **unless** unify(the two sub-types)
    return true

if either s′ or t′ are variables **then**
    append the type group of t′ to the type group of s′
    return true
return false

As usual, unify should throw a `NullPointerException` if either argument is null. You should write various tests for these methods, including one where the two compound types are:

- BiFunction<Function<S,T>,List<Integer>,List<T>>
- BiFunction<Function<Integer,U>,List<Integer>,Z>

General Considerations

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.

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.