EECS 293
Software Craftsmanship
2020 Spring Semester

Programming Assignment 11

Due at the beginning of your discussion session on

April 7-10, 2020

Reading

The quiz will cover the contents of the lectures and the following additional material:

- Section 19.6 in Code Complete and the Quick Reference Guide on Routine Names on canvas

The following list is optional helpful reading:

- Sections 6.1, 6.2, 6.3, and 6.4 in Code Complete
- Items 15 and 17 in Effective Java

Programming

The final project builds upon assignments 2 through 5 on type inference. Upon the conclusion of the project, it was realized that type inference could be extended to incorporate variable conversion and inheritance. In the final series of assignments, your objective is to design and implement ways to support type inference extensions.

Type Inference Extensions

The two main extensions involve automatic type conversion and inheritance. The main ideas are described in the rest of the assignment, whereas the specific requirements are part of your submission.

Type Conversion

In many cases, expressions of one type are automatically converted into another type. For example,

```
int x = 3;
double y = x;
```

automatically ensures the conversion of `x` into a double and its assignment to `y`. However, an explicit cast such as `x = (int) y;` cannot be considered an automatic conversion. Automatic type conversion can be incorporated in unification. For example,

- `Integer` and `Double` could unify as `Double` and, similarly,
- `List<Integer>` and `List<Double>` could unify as `List<Double>`
- `Collection` can be converted to a `List`, and so `Collection<S>` and `List<T>` should unify as `List<T>` (this unification assumes also that `S` can be unified with `T`).

Inheritance

If type `Derived` extends or implements type `Base`, then inheritance can be interpreted as the fact that `Derived` *is a* `Base`. Consequently, `Derived` and `Base` should be unified as a `Base`. Here are a few examples.

1. Since `Integer` is a `Number`, then `Integer` and `Number` can be unified as `Number`.
2. Since `Integer` and `Double` are both `Number`, then `Integer` and `Double` can be unified as a `Number`. Note that this case contradicts type conversion.
3. Since `ArrayList` and `LinkedList` are both `List`, then `ArrayList<Integer>` and `LinkedList<Double>` can be unified as `List<Number>` (or, assuming type conversion, `List<Double>`).
4. Since `List` is a `Collection`, then `Collection<S>` and `List<T>` should unify as `Collection<S>` (and `S` equivalent to `T`). Note that this case contradicts type conversion.
5. Since `TreeSet` implements `AbstractSet`, which in turn implements `AbstractCollection`, and since `LinkedList` implements `AbstractSequentialList`, which implements `AbstractList`, which in turn implements `AbstractCollection`, the expressions `<? super TreeSet>` and `<? super LinkedList>` would unify as `<? super AbstractCollection>`.

Design

You are expected to design classes and their methods to support additional features revolving around type conversion and inheritance. In other words, your program should maintain all previous functionality with the addition of these new features. In general, it is impossible to support unambiguously all cases of type conversion and inheritance. A major objective of your submission should be to define the conditions and scenarios under which types can and cannot be unified. Your design cannot address all of the examples above because of their conflicts. However, your submission should state which cases are and are not supported, both with examples and in general.

As you have learned in Section 5.1, you fully expect the design process to be sloppy, non-deterministic, based on heuristics, and iterative (and possibly very frustrating). However, you also shoot for a product that is tidy, clearly addresses the relevant trade-offs, priorities, and restrictions: in other words, it will be a beautiful architecture.

Create a design document that describes your architectural decisions. Your document should contain sketches and diagrams of your architecture. Your objective is that your design document should be used as a blueprint for the implementation. You can define classes or interfaces as you see fit. Commonalities among classes or interfaces should be expressed through inheritance or containment. For each class or interface, you should describe at least the abstraction it captures, its position in the inheritance hierarchy, the signature of its constructors and public methods, its private data structures (if any), and the pseudo-code of any complicated method. An integral part of software architecture is an approach for error handling. You should also outline your approach to testing.

You are not required to submit an implementation, which is the main topic of the next assignments.

Discussion Guidelines

The discussion will focus on class design.

Submission

Submit design documents that describes your architectural decisions. No implementation is required: you will implement your design in the next programming assignments. This assignment can be submitted on Canvas.