EECS 293
Software Craftsmanship
2020 Spring Semester

Programming Assignment 2

Due at the beginning of your discussion session on

January 28-31, 2019

Reading

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

- Chapter 14 in Code Complete
- Items 28, 50, 60, and 62 in Effective Java
- Java's BigInteger documentation (introduction only)
- Section 19.1 in Code Complete (excluding "Forming Boolean Expression Positively", "Guidelines for Comparison to 0", "In C-derived languages …", "In C++, consider ...")
- Section 15.1 ("Plain if-then Statements" only) in Code Complete
- Sections 17.1, 18.1 (excluding "Two Issues …"), and 19.4 in Code Complete ("Convert a nested if …" and "Factor deeply nested code …" only)

Programming

In this and the next few programming assignments, you will develop a *type inference* system similar to Java's and, in some cases, better than Java's.

In Java 10, you can sometimes let the compiler find the type of variables and avoid the overhead of finding and declaring types. For example, you can write a statement such as:

```
var n = new SomeClass();
```

and Java's type inference will infer that n is a variable of type SomeClass. Type inference can also address much more complicated scenarios. Suppose for example that you have just declared a function:

```
<S,T> List<T> foo(Function<S,T> f, List<Integer> l) {
          return new ArrayList<>();
}
```

Then, type inference should determine that the return value of

```
<U> bar(Function<Integer, U> f, List<Integer> l) {
          return foo(f, l);
}
```

is List<U>. Indeed, IDEs normally suggest adding List<U> as bar's return type. In this case, the core of type inference is to match T and U (and S with Integer).

In Programming Assignments 2 through 5, you will develop the core of a type inference system. Programming Assignment 2 gets you started with the representation of the type inference's underlying data structures.

Package

You should organize your project in a package. The package name is up to you: it could range from the simple ('typeinference') to the detailed ('edu.cwru.<cwruid>.airtravel).

Types

Create a public interface Type, which abstracts the concept of a type such as Integer, List, or T (in an expression such as List<T>). The interface has only one boolean method isVariable(), which is meant to return whether the type is a variable, such as T. Interfaces, such as Type, will omit any default implementation of declared methods.

Define a public final class TypeName that will be used to represent the names of a type, such as the strings "Integer" or "List" in such a way that two different types cannot share the same name. The TypeName has a private final String identifier along with a public getter. TypeName overrides toString to return the identifier. The TypeName has a private constructor that sets the identifier and a

```
public static final TypeName of(String identifier)
```

that returns a new `TypeName` with the given identifier. The `of` method throws a `NullPointerException` if the identifier is null and an `IllegalStateException` with an appropriate message if another `TypeName` with the same identifier has already been defined.

Create a `public interface TypeEntry`, which abstract the concept of a type within an expression. A difference between a `Type` and `TypeEntry` is for example that `List` is a `Type` but `List<T>` is a `TypeEntry`. The `TypeEntry` has two methods (additional methods will be defined in future assignments):

- `Type getType()`, which is meant to return the underlying `Type` of the `TypeEntry` (for example, the `getType` of `List<T>` returns `List`)
- `boolean isVariable()`, which delegates the method to the underlying `Type`.

Null Arguments

The programming assignments will ask you to implement various methods. These methods take arguments that are potentially null even though null parameters are not expected or meaningful. Whenever you implement such methods, you should make sure to throw a `NullPointerException` if the argument(s) are null. An example is `TypeName::of`. There are three exemptions to this rule:

- The assignments will specify build methods, such as `TypeName::of`, to construct objects or throw the exception. Therefore, your constructors can avoid checking whether their arguments are null.
- More generally, private methods do not need to check if arguments are null if they can only be invoked from methods that have already checked.
- If a method is automatically generated by the IDE, you do not need to add manually a validation for null parameters.

Skeletal Implementations

The `abstract class` `AbstractTypeEntry` implements `TypeEntry` to provide a default implementation of some of its methods. For the time being, it only defines `isVariable` to return false.

The `abstract class` `SimpleTypeEntry` extends `AbstractTypeEntry` and implements Type. Its purpose is to provide a default implementation of some of the `TypeEntry` methods that are adequate for simple type entries, such as `Integer` or `T`, but not for compound types, such as `List<T>`. Its `getType` method returns this simple type entry.

Basic Types

Basic types are simple types, such as `Integer`, `Boolean`, or `Double`. The `public final class` `BasicType` extends `SimpleTypeEntry` (thereby also indirectly implementing both `Type` and `TypeEntry`). The `BasicType` has a `private final TypeName typeName` along with a public getter. `BasicType` has a private constructor that sets the `typeName` and cannot throw any exception. It has a

`public static final BasicType of(String identifier)`

which returns a new `BasicType`. It delegates the `getIdentifier` and `toString` to the `typeName`.

Variables

Type variables are expressions such as `T` in `List<T>` and are essential to define generics. The `public final class` `VariableType` implements `Type` and simultaneously extends `SimpleTypeEntry`. It overrides `isVariable` to return true.

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. Additionally, comments should only be applied to the code sections that you feel are not self-documenting.

Canvas Resources

The module on Java Language Features contains a folder on useful Java features, such as enumerated types, regular expressions, and the optional class. A discussion board can be used to ask questions and post answers on this programming assignment.

Discussion Guidelines

The class discussion will focus on:

- High-level code organization
- The design and documentation of the implementation
- Straight-line code, conditional code

Your grade will be affected by the topics covered in this and in previous weeks. Your discussion leader may introduce material that has not been covered yet in the lecture. Although future topics will not contribute to your current grade, you are expected to follow your leader's advice in revising the assignment.

Submission

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. You can either bring a laptop to discussion to display your project on a projector, or present your project from the canvas submission.

Note on Academic Integrity

It is a violation of academic integrity not only to copy another group's work but also to provide your code for other groups to copy including but not limited to posts on public github projects or social media.