Refactoring List.
Departamento de Computação – UFSCar.
José de Oliveira Guimarães.

A refactoring tool takes as input a program in, say, Simples, and outputs the same program with some improvements. For example, the output may not have some unused variables, some classes may have been split in two or more, some classes may have been merged, some names may have been changed, etc. Of course, these changes are directed by the programmer, they are not made at will by the tool. The programmer usually chooses which kind of refactoring to apply by a Graphical User Interface, which asks for the information necessary to do the job. For example, the programmer may wish to change the name of class `Teacher` to `Professor`. The tool asks the name of the class to be changed and then the new name. The tool then changes all occurrences of class `Teacher` to `Professor`. This is not just a "Search and replace" operation. The tool must know the syntax and semantics of the language. For example, the first "`Teacher`" below should be replaced for it represents a class name. But the second should not be changed.

```
class Student
   private:
      var advisor : Teacher;
          Teacher : Person;
   public:
      ...
end
```

In the following items we show a small list of refactorings that a tool can do. We assume there will be a class `Refactoring` with one method for each refactoring. Each method of class `Refactoring` changes the AST. Then after each call to a `Refactoring` method, the compiler should call method `genSimples` (to be done) of class `Program` to print the new program created by the refactoring. Then the refactorings will not be called through a GUI. To use a refactoring one should call a method of class `Refactoring`. For example, suppose the AST for the program has been built by the compiler and variable `program` points to the root of the AST. To apply refactoring **extract class** (see below), one should do:

```
program  = compiler.compile(input, new PrintWriter(System.out) );
if ( program != null ) {
   PW pw = new PW();
   pw.set(printWriter);
   Refactoring refactoring = new Refactoring(program);
   refactoring.extractClass("Person", "Student",
        new String[] { "getCourse", "setCourse", "getNumber", "setNumber" },
        new String[] { "course", "number" } );
   program.genSimples(pw);
   if ( printWriter.checkError() ) {
      System.out.println("There was an error in the output");
   }
}
```

1. Extract class. Create a new class from a class. Some methods and instance variables are moved

from the original class to the new class. It is not necessary to check if the methods that are to be moved use only the instance variables that are to be moved.

This refactoring should be done by method `extractClass` that takes as parameters the name of the class, the name of the new class and the methods and instance variables that should be moved to the new class. Example:

```
refactoring.extractClass("Person", "Student",
   new Object[] { "getCourse", "setCourse" },
   new Object[] { "course"  } );
```

The above command will create a new class `Student` from a class `Person`. `Student` will have methods `getCourse` and `setCourse` and instance variable `course`. Of course, `Person` will no longer have these methods and this instance variable.

2. Collapse hierarchy. Merge superclass and subclass. Algorithm:

- add all subclass methods and instance variables into the superclass. Errors may occur;
- change all references to the subclass to references to the superclass;
- remove the subclass.

This refactoring should be done by method `collapseHierarchy` that takes as parameters the names of the class and subclass. It should be called like this:

```
refactoring.collapseHierarchy("A", "B");
```

3. Create a new subclass from a class. The same as previous item except that the new class inherits from the old one.

4. Move instance variables and methods to the superclass.

This refactoring should be done by methods `moveIVToSuperclass` and `moveMethodsToSuperclass`. Method `moveIVToSuperclass` takes as parameters the class name and an array with the instance variables to be moved. Example:

```
moveIVToSuperclass("UndergraduateStudent", new Object[] {
    "course", "universityName" } );
```

Method `moveMethodsToSuperclass` is similar.

5. Change the name of a method. Change the method name in its class and in all superclasses and subclasses. All the calls should be changed too. This refactoring should be done by method `changeMethodName` that takes as parameters the class and method names (new and old). Example:

```
changeMethodName("Student", "getCourse", "getStudentCourse");
```

6. Replace accesses to public or private instance variables by calls to `get` and `set` methods. This refactoring should be done by method `replaceAccessByCalls` that takes as parameters the class and instance variable names. Example:

```
replaceAccessByCalls("Student", "course");
```

7. Change a class name. Example: `changeClassName("Teacher", "Professor")`.