Practical on Polymorphism and Refactoring

Ian Bayley

In this practical, you will write a class `Drink` with a method `prepareRecipe` that prints one of two recipes according to whether the drink is coffee or tea. The method will use a conditional to inspect a field `DRINK_TYPE` to decide which recipe to print. As discussed in the lecture, this is the procedural approach of using a conditional for variant behaviour, as opposed to the object-oriented approach of using polymorphism. You will refactor the code to the latter form and call the resulting polymorphic method. Along the way, you will use some small-step refactorings that are built into Netbeans.

1. start by creating a new Netbeans project, ticking the box for Create Main Class.
2. create a class called `Drink` with a field `drinkType`. This will be an integer, with `1` representing coffee and any other value representing tea.
3. use the refactoring Encapsulate Field (obtained like all other refactorings by right-clicking on the code window) to produce getters and setters for the field `drinkType`
4. add a void method `prepareRecipe` that prints the recipe according to the following rules:
   (a) the first step is always to boil some water (so print ``Boiling water'' for example)
   (b) the second step is either to drip coffee through a filter, if coffee is required, or steep tea in boiling water, if tea is required
   (c) the third step is always to pour the liquid into the cup
   (d) the fourth step is either to add sugar and milk, if coffee is required, or add lemon, if tea is required

5. the resulting method is not that long but we shall pretend it has the Long Method code smell so use the Introduce Method refactoring to make method `prepareRecipe` call methods `step1`, `step2`, `step3` and `step4`. Make sure these methods are public.

6. those were really bad method names so use Rename to change each of the names to something more sensible; this is of course the Rename Method refactoring from the lecture

7. give class `Drink` two subclasses called `Coffee` and `Tea`

8. from class `Drink`, push down[1] the methods (you had previously called) `step2` and `step4`. This will create copies of `step2` and `step4` in both the `Coffee` class and the `Tea` class. Be sure to click the make method abstract option for both of these to make sure that methods `step2` and `step4` are still declared in the Drink class.

9. edit the definitions of `step2` and `step4` created in the `Coffee` class to print the right message without the conditional, which is superfluous because you know it is coffee

10. do the same with the `Tea` class

11. remove the field `drinkType` and its getter and setter; we didn't use these methods but introducing them was an opportunity to try out the Encapulate Field refactoring

12. in the main method, declare `d1` and `d2` to be of class Drink. Create a `Coffee` object and put it in `d1`. Create a `Tea` object and put it in `d2`. Call the method `prepareRecipe` on both `d1` and `d2`, and then run the program.

13. change the program so that `d1` holds the `Tea` object and `d2` holds the `Coffee` object. Note that the call to `prepareRecipe` remains unchanged. The call can be made in the same way whether the drink is really a coffee or a tea. This is one of the great advantages of polymorphism. You will see that with almost all design patterns.

14. Draw the UML diagram for the program you have created. This is an instance of the Template Method pattern, your first design pattern. Google and read about it if you still have time in the practical.

[1]Note there is a refactoring called Push Down Method but it is used when a method is relevant only for some of the subclasses; so this is not example of it.