Lab 4 — Java Reflection

CS 205

Lab objectives:

- Apply the reflection features of Java to examine the properties of objects.
- Override methods of the Object class.
- Apply inheritance and abstract classes to achieve polymorphism.

You will use Java's reflection feature to produce "object trees" for Java objects. Below is the output generated from the main program on the next page.

```
name=a, type=class java.util.ArrayList,id=0
   name=elementData, type=class [Ljava.lang.Object;,id=1
      name=[0], type=class java.lang.Integer,id=2
         name=value, type=int, value=123,id=3
      name=[1], type=class java.lang.String,id=4
         name=value, type=class [C,id=5
            name=[0], type=class java.lang.Character,id=6
               name=value, type=char, value=H,id=7
            name=[1], type=class java.lang.Character,id=8
               name=value, type=char, value=e,id=9
            name=[2], type=class java.lang.Character,id=10
               name=value, type=char, value=l,id=11
            name=[3], type=class java.lang.Character,id=12
               name=value, type=char, value=l,id=13
            name=[4], type=class java.lang.Character,id=14
               name=value, type=char, value=o,id=15
         name=hash, type=int, value=0,id=16
         name=hash32, type=int, value=0,id=17
      idref=0
      name=[3], type=class java.lang.Object, value=null,id=18
      name=[4], type=class java.lang.Object, value=null,id=19
      name=[5], type=class java.lang.Object, value=null,id=20
      name=[6], type=class java.lang.Object, value=null,id=21
      name=[7], type=class java.lang.Object, value=null,id=22
      name=[8], type=class java.lang.Object, value=null,id=23
      name=[9], type=class java.lang.Object, value=null,id=24
   name=size, type=int, value=3,id=25
```

```
public static void main(String[] args) {
    ArrayList a = new ArrayList();
    a.add(new Integer(123));
    a.add("Hello");
    a.add(a);
    Variable var = new Variable(a.getClass(), "a", a);
    VariableTreeModel model = new VariableTreeModel(var);
    TreePrinter printer = new TreePrinter(model);
    printer.print();
}
```

The contents of the ArrayList object `a` are displayed in a tree form by indenting the lines for the children. If the parent node is an object of a class, the children will be the instance variables of that object (including inherited instance variables, but excluding any static instance variables). If the parent node is an array, the children are the elements of the array. Primitive values and `null` values are leaves.

The tree printer gives an id to each node in the tree. If a node is repeated (as is the reference to the object `a`) the reference id is printed rather than showing the subtree further.

Implementation

1. Define a `Variable` class that has a declared `type` (`Class`), `name` (`String`), and `value` (`Object`) as instance variables. Define the `toString()` method to return a description containing the name, actual type, and the value. (The value should only be included if the type is a primitive type or the value is `null`). Define the `equals()` method carefully — it will need to act one way if the variable is an object, and another way if the variable is a primitive variable or if the value is `null`. You will also find it useful to implement `get` methods for each of the instance variables.

2. Define an `AbstractTreeModel` abstract class that implements the `TreeModel` interface. Make the `getRoot()`, `getChildCount()`, and `getChild()` methods abstract by declaring them abstract, and with no body. (They will have to be implemented in the concrete subclasses.) Implement the remaining methods of the interface. The `isLeaf()` method should simply return `true` if a node has no children. The `getIndexOfChild()` method can be implemented in terms of the `getChild method()`. (`isLeaf()` and `getIndexOfChild()` are examples of the Template Method pattern.) The remaining methods (`addTreeModelListener()`, `removeTreeModelListener()`, and `valueForPathChanged()` don't need to do anything — they are used for change notifications and we can simply assume that trees don't change.

3. Define a `VariableTreeModel` class that implements the tree consisting of a variable and its children. Most of the work here will be in implementing the `getChild()` method which will return a `Variable` representing the requested child or `null` if no such child exists. Both the `getChild()` and `getChildCount()` must handle these four cases for the `Parent` parameter: `null`, primitive variable, array, and object.

4. Define a `TreePrinter` class that can print the nodes in any tree model, not just the tree that shows variables. I will test your `TreePrinter` on another `TreeModel`, maybe like the one on the following page (*Hint, hint!*).

```
public class TestTreeModel extends AbstractTreeModel {

    public Object getChild(Object parent, int index) {
        if (parent.equals("Root")){
            if (index==0)
                return "Child #1";
            else if (index==1)
                return "Child #2";
            else
                return null;
        }
        else
            return null;
    }

    public int getChildCount(Object parent) {
        if (parent.equals("Root"))
            return 2;
        else
            return 0;
    }

    public Object getRoot() {
        return "Root";
    }
}
```

Add Javadoc comments to record the members of your team in the Class file containing your `main()` method and also to document the code that you write for the lab. Export your lab into a ZIP archive and submit it in GoucherLearn.