Exercise 4

Inheritance

October 21st, 2011

Unless otherwise stated, assume we are working with a language (such as Java) in which method dispatch is dynamic for the type of the receiver and static for the type of the arguments.

1. Consider two classes `Stack` and `Queue`, implementing the obvious data structures, both of which have methods with the following signatures:

```
void push(Object o);
Object pop();
bool isEmpty();
int size();
void reverse();
```

   - Despite having identical signatures, these two classes cannot be behavioral subtypes of one another. Why not?
   - When implementing these two classes, is there any possibility of code re-use? If so, give details.
   - What programming languages/features could support such code re-use without subtyping? Which of these do you think would be most suitable here?

2. Consider a class `Matrix` to implement matrices with integer values. A simple implementation would be to store a (private) 2-dimensional array of integers, and provide methods such as:

```
void set(int i, int j, int value);
int get(int i, int j);
Matrix add(Matrix m);
Matrix multiply(Matrix m);
```

   A *sparse matrix* is a matrix which contains mainly zeros. When such matrices are large it can be that an alternative representation of the matrix, which only stores the locations and values of non-zero entries, can provide much more efficient implementations for common expensive operations such as addition and multiplication with other sparse matrices. If a sparse matrix is to be added or multiplied with a standard matrix, it also is possible to define an implementation which is more efficient that the standard one (but not as good as for two sparse matrices).

   Consider writing a new class `SparseMatrix` to implement sparse matrices, with the similar methods available to those for `Matrix`.

   - Is it likely that there will be scope for reusing code from the class `Matrix`?
   - Does it seem that `SparseMatrix` can (and should?) be a behavioural subtype of `Matrix`?
   - What would be the implications of making `SparseMatrix` a subclass of `Matrix`?
   - What alternative ways are there of expressing the relationship between the classes?

3. Suppose from now on that `SparseMatrix` is to be implemented as a *subclass* of `Matrix`. Assume (reasonably!) that the two classes will use different internal representations (fields). If you sketch a possible implementation, it might help.
   - What would happen if client code could access the fields? e.g., suppose `entries` is the 2-d array field of `Matrix`, and `m` is a local `Matrix` variable, and consider:

     ```
     m.entries[i][j] = 4;
     if(m.get(i,j)!= 4) { // crash }
     ```

     What can go wrong here? To what extent are these problems avoided by making the fields private?
   - What might go wrong (or at least give unexpected behavior) if we do not override all of the methods of `Matrix` when writing `SparseMatrix`?
   - What difficulties might occur if we wanted to add extra methods to `Matrix` later?

4. (from a previous exam)

Consider the following Java classes:

```
public class B {
     public void foo(B obj) {
          System.out.print("B1 ");
     }
     public void foo(C obj) {
          System.out.print("B2 ");
     }

}

class C extends B {
     public void foo(B obj) {
          System.out.print("C1 ");
     }
     public void foo(C obj) {
          System.out.print("C2 ");
     }
     public static void main(String[] args) {
          B c = new C();
          B b = new B();
          b.foo(c);
          c.foo(b);
          c.foo(c);
     }
}
```

What is the output of the execution of method `main` in class C?