ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Exercise 5
Inheritance
October 23, 2015

Task 1

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? Explain your answer.

Task 2 Inheritance

From the midterm 2014.

Consider the following class in Java, which represents a fixed-size sequence of integers:

```
public class Seq {
   public Seq(int size) { a = new int[size]; } // all initialized to 0
   public int getSize(){ return a.length; }
   public int getAt(int i) { return a[i]; }
   public void setAt(int i, int x) { a[i]=x; }
   public void addTo(int i, int x) { a[i]+=x; }
   public void addToAll(int x){
      for (int i=0;i<a.length;i++)
         a[i]+=x;
   }
```

```
    private int[] a;
}
```

Consider also the following subclass of `Seq`, which adds a `getSum` method to `Seq` that is implemented efficiently:

```
public class SeqSum extends Seq {
    public SeqSum(int size) { super(size); }
    public int getSum() { return sum; }
    public void setAt(int i, int x) {
        int newSum=sum+x-getAt(i);
        super.setAt(i,x);
        sum = newSum;
    }
    public void addTo(int i, int x) {
        int newSum=sum+x;
        super.addTo(i,x);
        sum = newSum;
    }
    public void addToAll(int x) {
        super.addToAll(x);
        sum += getSize()*x;
    }

    private int sum=0;
}
```

In this question do not use downcasting or reflection. A "client" refers only to clients instantiating the class, not to subclasses.

A) Change the implementation of `Seq.addToAll` so that class `Seq` behaves exactly the same but `SeqSum.addToAll` calculates the wrong sum. Show a client that produces a different output with the original and modified implementations.

B) Assume the original implementation of both classes. Give an alternative implementation for `Seq.setAt` and separately for `SeqSum.addTo` so that each change alone leaves both classes behaving exactly the same, but putting both changes together would break the behavior of at least one method in class `SeqSum`. Show a client that observes the change in behavior.

Task 3

A) Compare dynamic type checking with the `dynamic` keyword to static type inference with `var` in C#:

- Give a correct program which can be realized with `dynamic` but not with `var`.
- Give an incorrect program which will be accepted by the compiler with `dynamic` but not with `var`.

B) C#'s most general type is `object`. Similar to `var` and `dynamic`, you can write `object x = ...` with an expression of any type on the right-hand side.

- Given a compiling program using `var`. Can we replace all `var` keywords by `object` and add explicit casts in the right places so that the program compiles and runs as before?
- Given a compiling program using `dynamic`. Can we replace all `dynamic` keywords by `object` and add explicit casts in the right places so that the program compiles and runs as before?

For both questions, either informally describe how to do the replacement, or give a counter-example where the transformation will always produce a program that does not compile or behaves differently. Note that explicit casts to `dynamic` are not allowed in the transformation.

C) Assume now a language like C#, but with covariant return types and contravariant parameter types. Given four classes `A`, `B`, `C` and `D`:

```
class A { int m (int x); }
class B { void m (dynamic x); }
class C { dynamic m (int x); }
class D { dynamic m (dynamic x); }
```

Develop a subtyping rule for the `dynamic` type annotation and informally explain the reasoning behind it. What are the potential subtypes among the four classes above?

Task 4

Assume we are working with a Java-like language in which method dispatch is dynamic for the type of the receiver and static for the type of the arguments. 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?

Task 5

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?

Task 6

Some research languages have symmetric multiple dispatch - methods are defined outside classes, and dispatch dynamically on all arguments regardless of order (no overloading at all). There is no designated receiver for a method but rather all arguments are of the same priority - this is intended to handle binary methods better which are often naturally symmetric. The runtime selects the most specific method to dispatch according to all arguments, and so there must be a single best implementation for each possible invocation of a method. The return type is not considered in the implementation selection. When compiling a package the compiler analyzes all types used in the package and all methods and makes sure that for each method and argument types combination there is a single best method to be called - or issues an error if that is not the case. Assume the following three classes in such a language:

```
package integer
class Integer
{
   ...
}
Integer add(Integer x,Integer y){...}

package natural
import integer.Integer
class Natural extends Integer
{
   ...
}
Integer add(Natural x,Integer y){...}
Integer add(Integer x,Natural y){...}
Natural add(Natural x,Natural y){...}

package even
import integer.Integer
class Even extends Integer
{
   ...
}

Integer add(Even    x,Integer y){...}
Integer add(Integer x,Even    y){...}
Even    add(Even    x,Even    y){...}
```

The elipsis in each class body represents (possibly) private data but no other methods.

Each package compiles successfully on its own.

A user has now written the following client:

```
package client
import even.*
import natural.*

void f(Integer x,Integer y)
{
   Integer z = add(x,y);
}
```

- What would be the problem in allowing this client to compile in a type safe multiple dispatch language? Show code that would expose the problem.
- Which requirement could we relax so that this call is valid? Dispatch must remain completely symmetric.
- What could we do in the client package, in order to resolve the problem, without modifying other packages and without relaxing the requirement mentioned above? What is the conceptual problem with this resolution?