Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Exercise 2

Types and Subtyping

October 3, 2014

Task 1

Suppose that we have a language with structural subtyping, contravariant parameter types and covariant return types. Consider the following types:

```
class A { int m(int x){...}; }
class B { int m(int x){...};  int n(int x) {...}; }
class C { int n(int y){...};  int m(int x) {...}; }
class D { C m(A a) {...}; }
class E { C m(B b) {...}; }
class F { A m(B e) {...}; }
class G { B m(C e) {...}; }
class H { G m(D d, E e) {...}; }
class I { F m(E e, D d) {...}; }
class J { A a; }
class K { B b; }
```

Find all the subtyping relations among them. Assume that `int` has no subtype other than itself.

Task 2

Consider the following Java program:

```
class B {
   protected int get() {...}
}

class A extends B {
   private int get() {...}
}

class C extends B {
   public int get() {...}
}
```

When we compile it, we obtain the following error:

```
get() in A cannot override get() in B; attempting to
assign weaker access privileges; was protected
        private int get() {...}
                    ^
```

Explain why this is the behavior of the Java compiler.

Task 3

Show:

- A program that is rejected by a statically typed language but is executed without typing errors in a dynamically typed language.
- A program that is rejected by a statically typed language and runs into a type error when executed in a dynamically typed language.

Task 4

Suppose that we have a C#-style programming language supporting the following kinds of parameters:

- "`in`" parameters. The caller provides an expression that is passed by value to the formal parameter.
- "`out`" parameters. The caller provides a variable as the actual parameter. An output from the method is returned and written to the actual parameter when the method terminates. The method does not read the initial value of the actual parameter and the actual parameter has no connection to the formal parameter during the execution of the method.
- "`in out`". The same as "out" but the method may also read the initial value of the actual parameter.
- "`ref`". The parameter is passed by reference. The caller provides a variable that is aliased by the formal parameter during the execution of the method.

What should the variance rules for these kinds of parameters be? Why? Refer to the contravariance rule for method parameters and the covariance rule for return values to explain your answer. Give examples that would not work, if your rules are not followed.

Could "`in out`" and "`ref`" be subtypes of each other? Explain your answer.

Task 5

In C++ object aliasing is achieved using pointers and it is possible to have a pointer to a pointer. Here is an example

```
class X {};

class Initializer {
   public:
      void init(X** x) {
         *x = new X();
      }
};

class Value {
   private:
      X* x = nullptr;
   public:
      Value(Initializer* i) {
         i->init(&x); // The initializer object will set the value of x
      }
};
```

How does the substitution principle apply to values of type pointer to pointer? Is it safe to call methods that have the signature of `init` with a value of type pointer to pointer to a subtype/supertype of X? Why?

Task 6 Union Types

Assume a language with nominal subtyping, covariant return types and contravariant arguments that allows types to be defined as a disjunction of other types, as in the following declarations:

```
String || Number get();
void set(String || Number newValue);
```

Such a type is called a *union type* and the different types that form the disjunction are its *components*. Classes can be thought of as union types with just one component.

A type `Sub` is a subtype of another type `Super`, i.e. `Sub <: Super`, if for each component C_{sub} of `Sub` there exists a component C_{sup} of `Super` such that $C_{sub} <: C_{sup}$. The usual nominal subtyping rules apply for classes.

A) Consider the signatures of the four methods below, assuming that `C <: B <: A` (`A`, `B`, and `C` are regular class types)

```
m1: B            foo (B b)
m2: A            foo (A || B ab)
m3: B || C       foo (A a)
m4: A || B || C  foo (C c)
```

Your task is to complete the table below. For each row and column, write 'yes', if the method at the left of the row could override the method at the top of the column. Otherwise write 'no'.

	m1	m2	m3	m4
m1	yes			
m2		yes		
m3			yes	
m4				yes

B) Assume that `A`, `B`, and `Q` are classes such that `B <: A` and `Q` is unrelated to `A` and `B`. Consider this code fragment:

```
void foo(A || Q arg) { arg.bar(42); }
```

(i) Assume that the type checker admits method `foo` if all components of `arg`'s static type have a method `bar(int)`. Do we need any run-time checks in order to avoid run-time errors? If so, what are they? Under what conditions could they be omitted?

(ii) Assume that the type checker admits method `foo` if at least one component of `arg`'s static type has a method `bar(int)`. Do we need any run-time checks in order to avoid run-time errors? If so, what are they? Under what conditions could they be omitted?

Answer the questions from (i) and (ii) for the code fragment below.

```
void foo(A || B arg) { arg.bar(42); }
```