Concepts of Object-Oriented Programming
AS 2022

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Exercise 7

Solution for Task 13

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`.

solution

```
static void Main() {
    dynamic x;
    if(condition()) {
        x = 5;
    } else {
        x = "hello";
    }

    Print(x);
}

static void Print(string str) {
    Console.WriteLine(str);
}

static void Print(int value) {
    Console.WriteLine(value);
}
```

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

solution

```
var x = 3;
x.substring(...);
```

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?

solution

This will be possible in all cases where we know what the type of the variable declared with `var` is. In those cases we can just cast the declared variable in all places where it is used to the most general type fulfilling all static type constraints on the corresponding variable. Since the original program compiled, such a type must exist.

In the case of anonymous types however, we do not know the name of the type to cast to. Consider:

```
var x = new { a = 108, b = "Hello" };
Console.WriteLine(x.b);
```

Here, we could change `var` to `object`, but we will not be able to cast `x` in the second line, because we do not know the type name which the compiler generates for this anonymous type.

- 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?

solution

Generally we cannot do this, as shown in the following example:

```
static void Main() {
    dynamic x;
    if(condition()) {
        x = 5;
    } else {
        x = "hello";
    }

    Print(x);
}

static void Print(string str) {
    Console.WriteLine(str);
}

static void Print(int value) {
    Console.WriteLine(value);
}
```

To make this code work with `object`, we would need to add explicit type checks and cast the argument to the proper static type.

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?

solution

Following the Substitution principle, `dynamic` is equivalent to `object`, in that it accepts any type. Therefore, the usual subtyping rules apply, treating `dynamic` as the most general

supertype of all other types. The potential subtyping relations are `A <: C` and `D <: C`.

There are two different ways of looking at class `B`. On the one hand, we could just say that `void` is a special keyword that indicates the absence of a return value, and thus the method `B.m` is unrelated to the other methods. Alternatively, we can allow methods with `void` return type to be overwritten by methods with any return type (assuming the parameter variance rules are satisfied): if a client code is written to expect `void` (no return value), then we could instead use a method which returns an arbitrary value and just discard it. In this second interpretation we will additionally have `D <: B`.