CSCU9Y4 Spring 2018

Programming Language Paradigms

Tutorial 18/3

Reading for this tutorial: Chapter 2 (2.4.2-2.4.6) and chapter 5 (5.3-5.4) of Introduction to Programming Languages (Bansal), Chapter 3 of Comparative Programming Languages. See also Y4_supplementary_notes_3.

Alternatively, Chapters 4 and 8 of Watt, Programming Language Design Concepts, or Chapter 5 and 6.10-6.11 of Sebesta, Concepts of Programming Languages. Or google "scope in programming languages", "lifetime or extent in programming languages", "type checking", "strong typing", "static typing" etc.

1. What is meant by the term **name-declaration binding** (also called **scope**)? How do scope rules relate to program blocks? What are the scope rules for Java and why do static scope rules ease the task of reading and understanding a program?

2. Distinguish between the *lifetime* and the *scope* of a variable.

3. In the course of running a program you may call the same procedure or method several times (perhaps on different objects, perhaps with the same object, perhaps recursively, but mainly not recursively). When you enter a procedure or method at run time, the *local* variables no longer have the values they had when the procedure or method was last executed. Why is this? How does this relate to scope?

continued over …

4. The different parts of this question refer to the following outline Java classes which you may assume are part of a larger program. Bear in mind that classes and objects are related but are not the same thing, also that class is a static entity (compile time) while objects are dynamic (run time).

```
public class A {
    private int s;
    ...
    public A(int x) {
        s = x;
    } // constructor
    ...
} // A

public class B {
    private int s;
    private A obj1;
    ...
    public void createTwo() {
        int s = 1;
        A obj2 = new A(s);
        obj1 = new A(23);
        ...
    } // createTwo
    ...
} // B
```

a) There are three places where a variable `s` is declared in the above. Why does this not cause any problems when `s` is used? What is the scope of `s` in each of these occasions?

b) Given a declaration and method call (elsewhere in the code):

```
B obj0 = new B(); obj0.createTwo();
```

describe in detail what happens when the method `createTwo` is called. In particular describe the scope and lifetime of:

i. the local variable `s`,

ii. the object reference `obj2`,

iii. the `A` object referred to by `obj2` (lifetime only),

iv. the object reference `obj1`,

v. the `A` object referred to by `obj1` (lifetime only),

vi. the formal parameter `x` in the constructor for A,

vii. the `A` attribute `s`.

Your answer should include what happens in each case to the space allocated when we exit from the method `createTwo`.