

Clarifications from April 26th Lecture

The example (upcasting/downcasting in an inheritance hierarchy) at the end of class today was rather tricky. Let's start at the beginning. We started with these assignments:

```
Person bob = new Person( );
Student ted = new Student( );
Student carol = new Student( );
GradStudent alice = new GradStudent( );
```

Your task, then, is to figure out whether the following four assignments are legal:

```
bob = ted;
carol = bob;
carol = (Student) bob;
alice = (GradStudent) ted;
```

To figure this out, keep in mind that there are two dimensions to casting:

1. What happens at compile time—when we care about types of reference variables in the stack.
2. What happens at runtime—when we care about the class of the actual objects on the heap.

So when you are asked to figure out which assignments are legal, start by examining whether the assignment **compiles**. The compiler knows the type of a reference variable, but it doesn't actually know the class of the object it will be pointing to. In order to compile, the type of the reference variable on the righthand side (which may be explicitly downcasted) must match the type of the variable on the lefthand side—either directly or through the inheritance hierarchy.

After we check this, we can ask whether the assignment **runs**. If there is no explicit cast, then it will run. If there is an explicit cast, then it will run as long as the class of the object on the heap matches the type it is supposed to be cast into.

Looking at the four assignments above, we observe right off the bat that the first statement compiles because “ted,” which is a reference variable of type Student, is being assigned to “bob,” which is a reference variable of type Person. Since a Student is a Person, the compiler is happy. At runtime, we are also fine: We have upcasted the Student object on the heap (pointed to by “ted”) to Person and so now the Person reference variable “bob” points to an object on the heap whose class is Student.

Next, we try to assign “bob” (a Person reference variable) to “carol” (a Student reference variable). What happens at compile time? We can't even get past the compiler. Since a Person is not a Student, we get a compile-time error: Eclipse tells us “Cannot convert

from Person to Student.” Note that we cannot do a downcast unless we do so explicitly, as in the third assignment statement above.

The third assignment statement makes it past the compiler because of the explicit downcasting of “bob” to a Student class. This downcasting creates an assignment statement where the type of the reference variable on the righthand side matches the type of the variable on the lefthand side. On the surface, everything looks fine. At runtime, we are also fine, but only because of the first assignment statement above! The point is that the variable bob is pointing to a Student object—and it is okay to explicitly downcast a Student object to a Student type. The result is that the Student reference variable “carol” points to a student object “bob.”

The fourth assignment statement makes it past the compiler. Why? Because—just like the previous case—the downcasting creates an assignment statement where the type of the reference variable on the righthand side matches the type of the variable on the lefthand side. On the surface, everything looks fine. But we run into trouble at runtime. Since “ted” is not pointing to a GradStudent object—it is pointing to a Student object—we cannot downcast the object pointed to by “ted” to be a GradStudent! The point is that “ted” doesn’t have the specific GradStudent components necessary to be viewed as a GradStudent. Note that this is different from the previous case where the downcast mapped a Student object into a Student type. In the current case, the attempt at casting the Student object to a GradStudent type brings about a ClassCastException:

```
Exception in thread "main" java.lang.ClassCastException: Student
    at Driver.main(Driver.java:18)
```