CSE 3302 Assignment 4
Due: 8 Apr 2010, 11am

Print out a hard copy of your answers, including code, and submit in class on the due date, or email them to me before the deadline. This is a hard deadline—extensions will not be granted. Type or write neatly.

1 Types

[10 pts] Recall that a record (struct) is a collection of named fields. Records can support two different notions of subtyping. Since a subtype is a type which allows all operations allowed on the original type, a record subtype should support the same operations on the fields as the original type supported.

In \textit{width subtyping}, a subtype record can adds more fields to the record. More formally, every field appearing in the width supertype will appear in the width subtype. We can write this as:

\[ \{x_1 : T_1, \ldots, x_n : T_n, x_{n+1} : T_{n+1}, \ldots, x_m : T_m\} \prec \prec \{x_1 : T_1, \ldots, x_n : T_n\} \]

Thus, any operation feasible on the supertype will be supported by the subtype. Thus, for example, the record type \{a: int, b: float, c: boolean\} is a subtype of \{a: int, b: float\}. The larger record type supports all the operations (a and b) as the smaller supertype plus the additional c operation.

The second notion, called \textit{depth subtyping}, replaces the various fields with their subtypes. That is, the fields of the subtype are subtypes of the fields of the supertype. Since any operation supported for a field in the supertype is supported for its subtype, any operation feasible on the record supertype is supported by the record subtype. Depth subtyping can be defined using the following inference rule:

\[
\frac{T_1 \ll T_1' \quad \cdots \quad T_n \ll T_n'}{\{x_1 : T_1, \ldots, x_n : T_n\} \ll \{x_1 : T_1', \ldots, x_n : T_n'\}}
\]

(a) Give a concrete example of two record types in which one record is a proper depth subtype of the other. (A proper subtype is a subtype that is not equal to the supertype.)

(b) Could a language support both kinds of subtyping?

(c) Assuming local variables of record type is allocated on the stack, what should happen when a subtype record is assigned into a variable of a \textit{width} supertype?

(d) What should happen when a subtype record is assigned into a variable of a \textit{depth} supertype?

(e) Many languages support \textit{mutable} records—that is, writes to record fields are permitted. Does this pose any problems if the language also supports width subtyping of records? If so, give an example that exhibits the problem.

(f) Do mutable records this pose any problems if the language also supports depth subtyping of records? If so, give an example that exhibits the problem.
2 Objects

[5 pts] In Java, a method can override a method with the same signature defined in a superclass. The overriding method in the subclass can call the superclass method through the special super receiver.

class Window {
    void paint() {
        ... /* paint the background */
    }
}
class Button extends Window {
    void paint() {
        super.paint(); /* paint the background */
        ... /* paint the button label */
    }
}

The language Beta, inverts this design. Rather than having subclasses invoke superclass methods via the super keyword, instead, superclasses can invoke subclass methods via via inner keyword. Rewriting the example in Beta (with Java-like syntax):

class Window {
    void paint() {
        ... /* paint the background */
        inner.paint(); /* paint the rest of the window */
    }
}
class Button extends Window {
    void paint() {
        ... /* paint the button label */
    }
}

When Window.paint invokes the paint method on inner, Button.inner is invoked.

(a) What are some advantages of the two approaches? Disadvantages?

(b) How could inner be simulated in Java? Rewrite the code above in legal Java so that Window.paint calls Button.paint.

(c) Note that the paint method is void. Can a non-void methods be invoked on inner? What happens if Window has no subclasses?