CMSC 245: Principles of Programming Languages  
Lab #8: Exam Review

Note: there will be no solution set posted for these problems. Instead, work out solutions on a computer or post on Piazza with a proposed solution.

Throughout this exam, assume we have classes Pet, Cat, and Dog, each with 0-parameter constructors, and such that Dog <: Pet and Cat <: Pet. Furthermore, assume Pet <: Comparable<Pet>.

1. Each of the following Java snippets contains one error relating to type variables and parameters. Find and correct the error.

(a) public static <A,B> A identity(B b)
    {
        return b;
    }

(b) public static A getFirst(ArrayList<A> list)
    {
        return list.get(0);
    }

c) public static void frob(ArrayList<Pet> pets) { ... }

d) public static void frob(ArrayList<? extends Dog> pets) { ... }

e) public static <A extends Comparable<A>> A max(A a1, A a2) { ... }

2. Write a Java program that, when run, will throw an ArrayStoreException.
3. Draw the subtyping relation between the following Java types by laying out the type names and drawing arrows from subtypes to supertypes:

Dog
Cat
Pet
Comparable<Pet>
Comparable<Dog>
ArrayList<Dog>
ArrayList<Pet>
List<Dog>
List<Pet>
Dog[]
Pet[]
ArrayList<? extends Pet>
ArrayList<? super Pet>
List<? extends Pet>
List<? super Pet>

4. Here is a way of writing a typing rule for Java’s subtraction operator:

\[
\Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2 \\
\text{ctnt}(\tau_1) \quad \text{ctnt}(\tau_2) \\
\Gamma \vdash e_1 - e_2 : \text{bnp}(\tau_1, \tau_2) \quad \text{MINUS}
\]

where ctnt(\tau_1) checks whether \( \tau_1 \) is convertible to a primitive numeric type and bnp(\tau_1, \tau_2) performs binary numeric promotion on \( \tau_1 \) and \( \tau_2 \).

This rule is derived from the following sentences in the Java Language Specification:

[T]he type of each of the operands of the binary - operator must be a type that is convertible to a primitive numeric type, or a compile-time error occurs.

The type of an additive expression on numeric operands is the promoted type of its operands.

Give typing rules for Java’s addition operator, described in these sentences:

If the type of either operand of a + operator is String, then the operation is string concatenation.

Otherwise, the type of each of the operands of the + operator must be a type that is convertible to a primitive numeric type, or a compile-time error occurs.

The result of string concatenation is a reference to a String object that is the concatenation of the two operand strings.

The type of an additive expression on numeric operands is the promoted type of its operands.

Write your typing rules here; you will need multiple.
5. The simply-typed λ-calculus is defined as follows:

\[
\begin{align*}
e & ::= x \mid e_1 e_2 \mid \lambda x: \tau. e \\
\tau & ::= \text{Int} \mid \tau_1 \rightarrow \tau_2 \\
x & ::= (\text{named variables}) \\
n & ::= (\text{integers}) \\
\Gamma & ::= \emptyset \mid \Gamma, x : \tau
\end{align*}
\]

\[
\begin{align*}
x : \tau & \in \Gamma & & \text{VAR} & & \Gamma \vdash e_1 : \tau_1 \rightarrow \tau_2 & & \Gamma \vdash e_2 : \tau_1 & & \text{APP} & & \Gamma, x : \tau_1 \vdash e : \tau_2 & & \text{ABS} \\
\Gamma \vdash n : \text{Int} & & \text{INT} & & \Gamma \vdash e_1 : \text{Int} & & \Gamma \vdash e_2 : \text{Int} & & \text{PLUS} & & \Gamma \vdash e : \tau & & \text{PAR}
\end{align*}
\]

Write proof trees for the following typing relationships, filling in the type to the right of the :.

(a) \(\emptyset \vdash (\lambda x : \text{Int}. x + x) 8 : \text{Int} \)
(b) \(x : \text{Int}, y : \text{Int} \rightarrow \text{Int} \rightarrow \text{Int} \vdash y x x : \text{Int} \rightarrow \text{Int} \rightarrow \text{Int} \)
(c) \(x : \text{Int} \vdash 1 + 2 + 3 : \text{Int} + \text{Int} + \text{Int} \)

6. Suppose we have a Java variable \(x\), of type \text{int}, with the value 5. For each expression below: (i) What will be the value of the expression? (ii) What will be \(x\)'s value after the expression is evaluated? (Each expression is to be considered independently, with \(x\) starting out as 5 each time.)

Recall that the result of evaluating an assignment expression (like \(x = 7\)) is the new value of the assigned variable (7, in the example).

(a) \(x + 5\)
(b) \((x = 7) + x\)
(c) \(x + (x = 7)\)
(d) \((x = 8) > 6 \ || (x = 3) < 2\)