CS211 Spring 2005
Prelim 1
March 10, 2005

Solutions

Instructions

Write your name and Cornell netid above. There are 6 questions on 9 numbered pages. Check now that you have all the pages. Write your answers in the boxes provided. Use the back of the pages for workspace. Ambiguous answers will be considered incorrect. The exam is closed book and closed notes. Do not begin until instructed. You have 90 minutes. **Good luck!**

	1	2	3	4	5	6	Σ
Score	/15	/24	/10	/15	/21	/15	/100
Grader							

1. (15 points) Prove by induction that for all $n \geq 0$,

$$\sum_{i=0}^{n-1} i(i+1) \;=\; \frac{n^3 - n}{3}.$$

Label clearly the parts of your proof. In the induction step, state the induction hypothesis and indicate exactly where in your argument it is used. Make sure your argument is perfectly clear—any ambiguity whatsoever will result in loss of points.

Basis: For $n = 0$, the left-hand side is the empty sum, which by convention is 0, and the right-hand side evaluates to 0.

Induction step: Let $n \geq 0$. Assume that

$$\sum_{i=0}^{n-1} i(i+1) \;=\; \frac{n^3 - n}{3}.$$

This is the induction hypothesis. We wish to prove under this assumption that

$$\sum_{i=0}^{n} i(i+1) \;=\; \frac{(n+1)^3 - (n+1)}{3}.$$

Starting from the left-hand side,

$$\begin{aligned}
\sum_{i=0}^{n} i(i+1) &= \sum_{i=0}^{n-1} i(i+1) + n(n+1) && \text{by taking out the last term} \\
&= \frac{n^3 - n}{3} + n(n+1) && \text{by the induction hypothesis} \\
&= \frac{(n+1)^3 - (n+1)}{3} && \text{by elementary algebra.}
\end{aligned}$$

2. (24 points) Say what is printed by the following Java programs.

(a)
```
class Xp {
   public static void main(String[] args) {
      System.out.println(new Xp().ff());
   }
   String ff() {
      char[][] a = {{'a','b','c'},{'d','e'}};
      StringBuilder s = new StringBuilder();
      for (char[] t : a) {
         for (char u : t) {
            s.append(u);
         }
      }
      return s.toString();
   }
}
```

```
abcde
```

(b)
```
class Enum {
   enum Season { WINTER, SPRING, SUMMER, FALL }
   public static void main(String[] args) {
      System.out.println(new Enum().pf());
   }
   String pf() {
      if (Season.WINTER.equals("WINTER")) return "equal";
      else return "not equal";
   }
}
```

```
not equal
```

(c)
```
class Fs {
   static String s = "hello";
   public static void main(String[] args) {
      new Fs().foo();
      System.out.println(s);
   }
   void foo() {
      String s = "world";
   }
}
```

hello

(d)
```
class Wg {
   static int x = 0;
   int y;
   Wg() { y = x++; }
   public String toString() {
      return String.valueOf(y);
   }
   public static void main(String[] args) {
      String s = new Wg().toString();
      s += new Wg().toString();
      s += new Wg().toString();
      System.out.println(s);
   }
}
```

012

(e)
```
class Rts {
   public static void main(String[] args) {
      System.out.println(zorg(new RtsC()));
   }
   static int zorg(RtsA x) {
      return x.f()*10 + x.a;
   }
}
abstract class RtsA {
   int a = 2;
   int f() { return a; }
}
class RtsB extends RtsA {
   int a = 3;
   int f() { return a; }
}
class RtsC extends RtsB {
   int a = 4;
}
```

32

(f)
```
class Yfk {
   public static void main(String[] args) {
      System.out.println(new YfkC().x);
   }
}
abstract class YfkA {
   int x = 3;
   YfkA() { x++; }
}
class YfkB extends YfkA {}
class YfkC extends YfkB {}
```

4

(g)
```
class Vbn {
   public static void main(String[] args) {
      System.out.println(new VbnC().zorg());
   }
}
class VbnB {
   String zorg(VbnB x) { return x.f(); }
   String f() { return "VbnB"; }
}
class VbnC extends VbnB {
   String zorg() { return super.zorg(this); }
   String f() { return "VbnC"; }
}
```

```
VbnC
```

(h)
```
class MyInteger {
   int value;
   MyInteger(int v) { value = v; }
}
class Tvm {
   public static void main(String[] args) {
      MyInteger s = new MyInteger(13);
      int t = 13;
      foo(s,t);
      System.out.println(s.value + " " + t);
   }
   static void foo(MyInteger u, int v) {
      u.value = 12;
      v = 12;
   }
}
```

```
12 13
```

3. (10 points)

(a) Explain, in at most 50 words, the difference between *reference types* and *primitive types* in Java.

Reference types are class names, interface names, and array types. A variable of reference type does not contain the data itself, but rather a pointer to a heap-allocated object. Primitive types are types such as `int`, `boolean`, and `char`. Variables of primitive type contain the data itself.

(b) Explain, in at most 50 words, the difference between *static types* and *dynamic types* in Java.

Static types are types of expressions in the program. They are determined at compile time from declarations. Dynamic types are the type of objects created at runtime, usually by a `new` instruction.

4. (15 points) Binary trees and lists were defined in class in terms of `TreeCell` and `ListCell`. Lists are terminated by `null`, and one or both children of a `TreeCell` may be `null`.

```
class TreeCell {
   Object datum;
   TreeCell left, right;
   ...
}

class ListCell {
   Object datum;
   ListCell next;
   ListCell(Object obj, ListCell c) {
      datum = obj;
      next = c;
   }
   ListCell(Object obj) {
      this(obj, null);
   }
}
```

Write a recursive method `tree2List` that, for a given binary tree, creates a list of the elements in preorder. For each `TreeCell` in the input tree, there should be exactly one `ListCell` with the same datum in the output list, and the order of elements in the list from front to back should be the same as preorder in the original tree. (*Hint.* Create the list from back to front. Use the second argument of the helper function to pass the accumulated list so far.)

```
static ListCell tree2List(TreeCell t) {return tree2List(t,null);}
static ListCell tree2List(TreeCell t, ListCell c) {
   if (t == null) return c;
   c = tree2List(t.right, c);
   c = tree2List(t.left, c);
   return new ListCell(t.datum, c);
}
```

5. (21 points) True or false?

(a) T [F] A downcast can change the dynamic type of an object.

(b) T [F] An upcast can change the dynamic type of an object.

(c) [T] F A `private` field can only be accessed by methods in the same class.

(d) T [F] Java supports multiple inheritance.

(e) [T] F A class that implements an interface `I` must also implement all superinterfaces of `I`.

(f) [T] F A call to `super` or `this` in a constructor must occur first.

(g) T [F] `"211".equals(211)`

(h) T [F] `"211" instanceof Integer`

(i) T [F] The static type of an expression is determined by whether the variables in it have been declared `static`.

(j) [T] F The static type of the expression `(Integer)x` is `Integer`.

(k) [T] F A `static` field is shared by all objects of the class.

(l) [T] F A non-abstract subclass of an abstract class `A` must implement all methods declared abstract in `A`.

(m) T [F] Subclasses and subtypes are the same thing.

(n) [T] F Abstract classes and interfaces cannot be instantiated.

(o) [T] F `int[]` is a reference type.

(p) T [F] Local variables of a method can be declared `static` provided the method is declared `static`.

Questions (q)–(u) refer to the following grammar for well-parenthesized Boolean expressions with operators `&` (and), `|` (or), and `!` (not), constants `true` and `false`, and variables V (not defined in the grammar). Spaces are not significant.

$$E \rightarrow V \qquad E \rightarrow \texttt{(}\ E\ \texttt{\&}\ E\ \texttt{)}$$
$$E \rightarrow \texttt{true} \qquad E \rightarrow \texttt{(}\ E\ \texttt{|}\ E\ \texttt{)}$$
$$E \rightarrow \texttt{false} \qquad E \rightarrow \texttt{!}E$$

(q) [T] F The following is a sentence: `!!!( true | true )`

(r) T [F] The following is a sentence: `((true))`

(s) [T] F The following is a sentence: `( !true & ( false | true ))`

(t) [T] F All sentences are of finite length.

(u) [T] F The number of possible sentences is infinite.

6. (15 points) Write an `equals` method for `String`. You may use the following methods from the `String` class:

`char charAt(int index)`	returns the `char` value at the specified index
`int length()`	returns the length of this string

Indexing starts at 0. Your method should return a value even if the argument is not a `String`.

```
public boolean equals(Object obj) {
   if (!(obj instanceof String)) return false;
   String s = (String)obj;
   if (length() != s.length()) return false;
   for (int i = 0; i < length(); i++) {
      if (charAt(i) != s.charAt(i)) return false;
   }
   return true;
}
```

End of Exam