CIS 120 Final Exam May 3, 2013

Name (printed): _____
Pennkey (login id): _____

My signature below certifies that I have complied with the University of Pennsylvania's Code of Academic Integrity in completing this examination.

Signature: _____ Date: _____

1	/10
2	/8
3	/10
4	/18
5	/10
6	/14
7	/12
8	/18
Total	/100

- Do not begin the exam until you are told to do so.
- You have 120 minutes to complete the exam.
- There are **100** total points.
- There are 15 pages in this exam, plus a 2 page appendix.
- Make sure your name and Pennkey (a.k.a. username) is on the top of this page.
- Be sure to allow enough time for all the problems—skim the entire exam first to get a sense of what there is to do.

1. True or False (10 points)

a. T F To call the constructor of the `ArrayList` class, you must specify a size. An instance of this class cannot store more than that number of elements.

b. T F The `LinkedList` class in the Java Collections framework implements immutable lists, just like OCaml's `list` type.

c. T F It is possible to write your own instance of the `Set` interface in Java.

d. T F The binary search tree invariant means that the process of determining whether a particular value appears in a tree does not require searching the entire tree.

e. T F Every mutable reference in OCaml could refer to `None`.

f. T F In a Java method, assignment to the `this` reference can be used to efficiently change the entire state of an object.

g. T F A method with the following declaration definitely will not throw an `IOException` to a calling context.

```
public void m() {...}
```

h. T F The declaration `public void m() throws IOException` indicates that the method `m` **must** throw an `IOException` every time that it is invoked.

i. T F In both the Java Swing libraries and our OCaml GUI library, JComponents/widgets should form a tree structure.

j. T F In Swing, the `paintComponent` method only executes the very first time the Component is displayed, when the application initially starts.

2. **Design Recipe** (8 points)

   List the four steps of the design recipe.

   a.

   b.

   c.

   d.

3. Array processing (10 points)

Consider a static method called `trim` that returns a copy of its string argument with leading and trailing whitespace omitted. Once implemented, `trim` should pass the following unit test.

```
@Test public void testTrim() {
   assertEquals("abcd", Main.trim("   abcd    "));
   assertEquals("", Main.trim("        "));
   assertEquals("a d", Main.trim("   a d    "));
}
```

Fill in the blanks below to complete the implementation of `trim`. In your answer, the only methods that you may use are `length` and `charAt` from the `String` class, and `isWhitespace` from the `Character` class. (Documentation for these methods appears in the Reference Appendix.)

```
class Main {
 public static String trim(String s) {

   int from = 0;

   while (_____________________________________ &&
               Character.isWhitespace(s.charAt(from))) {
     from = from + 1;
   }


   int to = _____________________________________________;


   while (_____________________________________________________________________) {
     to = to - 1;
   }



   char[] chars = new char[_____________________________];


   for (int i=0; i< chars.length; i++) {


     ____________________________________________________;
   }

   return new String(chars);
  }
}
```

4. Objects (18 points total)

Consider the following OCaml definitions:

```
type pet = { speak : unit -> string;
             get_name : unit -> string }

let cat (name : string) : pet = {
  speak  = (fun () -> "meow");
  get_name = (fun () -> name)
}

let dog (name : string) : pet = {
  speak = (fun () -> "woof");
  get_name = (fun () -> name)
}
```

a. (6 points) Here is some code that uses these definitions.

```
let main () =
  let bella = cat "Bella" in
  let owen = dog "Owen" in
  let esme = cat "Esme" in

  let pets = [bella; owen; esme] in

  let rec f (ps : pets) : unit =
    begin match ps with
     | [] -> []
     | p :: t ->
       print_endline (p.get_name () ^ " says " ^ p.speak());
       f t
    end in

  f pets
```

What is printed on the console when the `main` function is called?

b. (12 points) Suppose we were to translate the code on the previous page to Java. For example, one way to translate the `main` function is as follows:

```
public static void main(String[] args) {
  Cat bella = new Cat ("Bella");
  Dog owen = new Dog ("Owen");
  Cat esme = new Cat ("Esme");

  Pet[] pets = { bella, owen, esme };

  for (Pet p : pets) {
    System.out.println(p.getName () + " says " + p.speak() );
  }
}
```

Using the space below, define the `Pet` interface and `Cat` class, the translations of the OCaml definitions `pet` and `cat` on the previous page. (The `Dog` class is similar to `Cat`, so you don't need to do both for this problem.) Your definitions must be compatible with the `main` method above.

5. Java Types (10 points)

Consider the following excerpt from the class definitions used in Homework 9. (Additional documentation for the classes and interfaces in the Java standard library appears in the appendix):

```
class Token {
  public Token(boolean isWord, String tok) { ... }
  public boolean isWord() { ... }
}
class TokenScanner implements Iterator<Token> {
  public TokenScanner(Reader in) throws IOException { ... }
  public boolean isWordCharacter(int c) { ... }
  public boolean hasNext() { .. }
  public Token next();
}
```

Write down a type for each of the following Java variable definitions. Due to subtyping, there may be more than one correct answer. Any correct answer will be accepted. Write **ill-typed** if the compiler would flag an error anywhere in the code snippet (i.e. Eclipse would underline something in red). Assume that all of these definitions occur in the context of an exception handler (so that failing to catch an exception is *not* a error).

The first one has been done for you as a sample and is used in the remaining definitions.

a. `___FileReader_____` `reader = new FileReader("theFox.txt");`

b. `_____` `x = (char)38;`

c. `_____` `s = new TokenScanner(reader);`

d. `_____` `b = new TokenScanner(new BufferedReader(reader));`

e. `_____` `f = new BufferedReader("theFox.txt");`

f. `_____` `i = new Iterator<Token>();`

g. `_____` `u = new TokenScanner(reader).next();`

h. `_____` `t = reader.next();`

i. `_____` `m = reader.read();`

j. `_____` `w = TokenScanner.hasNext();`

k. `_____` `k = new Token(true, "120");`

6. Java ASM and Exceptions (14 points total)

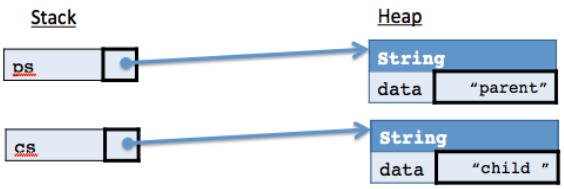
Consider the following class definitions (inspired by XKCD).

```
class Ball extends Throwable {}
class P {
  P target;
  String name;
  P (String name, P target) { this.name = name; this.target = target; }
  void aim(Ball ball) {
    try {
      System.out.println(name + " is throwing the ball.");
      throw ball;
    } catch (Ball b) {
      System.out.println(name + " caught the ball.");
      target.aim (b);
    }
  }
}
```

Suppose the following code is placed on the workspace of the Java ASM.

```
String ps = "parent";
String cs = "child ";
// ---------------- START ----------------
P parent = new P(ps,null);
P child = new P(cs,parent);
parent.target = child;
Ball ball = new Ball();
```

(10 points) The stack and heap diagram that corresponds to the point in the execution marked `START` is shown on the next page. Extend this diagram so that it displays the stack and heap at the point when execution terminates.


Stack
Heap
ps
String
data
"parent"
cs
String
data
"child "

(4 points) Now suppose the following code is placed on the workspace.

```
parent.aim(ball);
```

What happens next? Circle the correct behavior from the choices below.

a. The console prints

```
parent is throwing the ball.
child caught the ball.
```

and execution terminates.

b. The console prints

```
parent is throwing the ball.
parent caught the ball.
```

and execution terminates.

c. Nothing is printed to the console and the program immediately terminates.

d. The console prints

```
child is throwing the ball.
parent caught the ball.
parent is throwing the ball.
child caught the ball.
...
```

repeatedly and then eventually produces a `StackOverflowError`.

e. The console prints

```
parent is throwing the ball.
parent caught the ball.
child is throwing the ball.
child caught the ball.
...
```

repeatedly and then eventually produces a `StackOverflowError`.

f. The console prints

```
child is throwing the ball.
child caught the ball.
parent is throwing the ball.
parent caught the ball.
...
```

repeatedly and then eventually produces a `StackOverflowError`.

g. Nothing is printed to the console and the program produces a `StackOverflowError`.

7. Collections and Equality (12 points)

The following comment is taken from the Java library implementation of the `equals` method for both the `LinkedList` and `ArrayList` classes.

```
public boolean equals(Object o)
```

> *Compares the specified object with this list for equality. Returns true if and only if the specified object is also a list, both lists have the same size, and all corresponding pairs of elements in the two lists are equal. (Two elements* `e1` *and* `e2` *are equal if* `(e1==null ? e2==null : e1.equals(e2))`*.) In other words, two lists are defined to be equal if they contain the same elements in the same order.*
>
> *This implementation first checks if the specified object is this list. If so, it returns true; if not, it checks if the specified object is a list. If not, it returns false; if so, it iterates over both lists, comparing corresponding pairs of elements. If any comparison returns false, this method returns false. If either iterator runs out of elements before the other it returns false (as the lists are of unequal length); otherwise it returns true when the iterations complete.*

Consider the following creation of a linked list that contains a single element:

```
List<String> l1 = new LinkedList<String>();
String str = "CIS 120";
l1.add(str);
```

Each of the code snippets below first constructs a new list and then presents one or more comparisons. For each of the comparisons below, circle whether it returns **`true`** or returns **`false`**.

a.
```
List<String> l2 = new LinkedList<String>();
l2.add(str);
```

- `l1.equals(l2)`
  `true` **`false`**

- `l1 == l2`
  `true` **`false`**

b.
```
List<String> l3 = new ArrayList<String>();
l3.add(str);
```

- `l1.equals(l3)`
  `true` **`false`**

c.
```
List<Object> l4 = new LinkedList<Object>();
l4.add(str);
```

- `l1.equals(l4)`

  true **false**

d.
```
List<Object> l5 = new LinkedList<Object>();
l5.add(l1);
```

- `l1.equals(l5)`

  true **false**

e.
```
List<String> l6 = new LinkedList<String>();
l6.add("CIS 120");
```

- `l1.equals(l6)`

  true **false**

8. Binary Trees (18 points total)

Consider a Java implementation of the helices and evolutionary trees from Homework 2. We represent a sequence of nucleotides using the `Helix` class below. (Only the parts of this class that are relevant for the problem are shown.)

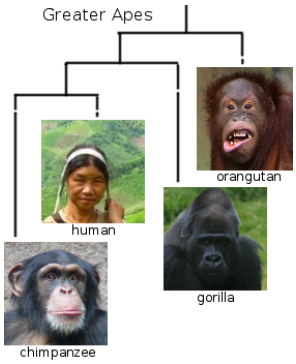
```
class Helix {
  // constants for the DNA data for particular species
   public static final Helix GORILLA = ... ;
   public static final Helix CHIMPANZEE = ... ;
   public static final Helix HUMAN = ... ;
   public static final Helix ORANGUTAN = ... ;

   // compute a guess for the parent DNA given the DNA of two descendants
   // if either argument is null, this method returns null
   public static Helix guessParent(Helix l, Helix r) { ... };
}
```

To represent evolutionary trees, we use the following class. *Leaf nodes* are those where both of the `left` and `right` fields are `null`. *Internal nodes* are those that have at least one non-null child.

```
class Node {
      public Node left;
      public Node right;
      public Helix helix;
      public Node (Node l, Helix h, Node r) {
            left = l; helix = h; right = r;
      }
}
```

For example, we can represent the following tree for the greater apes



using the following definitions.

```
Node c = new Node(null, Helix.CHIMPANZEE, null);
Node h = new Node(null, Helix.HUMAN, null);
Node g = new Node(null, Helix.GORILLA, null);
Node o = new Node(null, Helix.ORANGUTAN, null);

Node apes = new Node(new Node(new Node(c, null, h), null, g), null, o);
```

a. (6 points) Write a static method that counts the number of leaves of an evolutionary tree. For example, the method call `numberOfLeaves(apes)` should return 4.

```
public static int numberOfLeaves(Node n) {




}
```

b. (12 points) The sample tree `apes` is called *unlabeled* because we don't know the helices of the ancestors of the species at the leaves of the tree—the helix for each of the internal nodes is `null`. Recall that homework 2 took trees of this form and *labeled* them by repeatedly guessing the DNA sequences for each ancestor.

Write a static method that takes an unlabeled tree and produces a labeled tree, using the `guessParent` method of the `Helix` class. For example, the method call `labelTree(apes)` should return a new tree like `result` below. (The `apes` tree should not be modified by the method.)

```
Helix ch = Helix.guessParent(Helix.CHIMPANZEE, Helix.HUMAN);
Helix chg = Helix.guessParent(ch, Helix.GORILLA);
Helix chgo = Helix.getParent(chg, Helix.ORANGUTAN);
Node result = new Node(new Node(new Node(c, ch, h), chg, g), chgo, o);
```

```
public static Node labeledTree(Node n) {









}
```

Reference Appendix

Make sure all of your answers are written in your exam booklet. These pages are provided for your reference—we will *not* grade any answers written in this section.

java.lang

```
public class String
  public String(char[] value)
    // Allocates a new String so that it represents the sequence of
    // characters currently contained in the array argument
  public char charAt(int index)
    // Returns the char value at the specified index
  public int length()
    // Returns the length of this string
  public boolean equals(Object anObject)
    // Compares this string to the specified object. The result is true if and
    // only if the argument is not null and is a String object that represents
    // the same sequence of characters as this object.

public class Character
  public static boolean isWhiteSpace(char ch)
     // Determines if the specified character is whitespace
```

java.util (Collections Framework)

```
public interface Iterator<E>
   public boolean hasNext()
      // Returns true if the iteration has more elements. (In other words,
      // returns true if next would return an element rather than throwing an exception.)

   public E next()
      // Returns the next element in the iteration.
      // Throws: NoSuchElementException – iteration has no more elements.
```

java.io

```
public abstract class Reader
  public int read() throws IOException
    // Reads a single character. This method will block until a character
    //   is available, an I/O error occurs, or the end of the stream is reached.
    // Returns: The character read, as an integer in the range 0 to
    //   65535 (0x00-0xffff), or -1 if the end of the stream has been reached
    // Throws: IOException - If an I/O error occurs

public class BufferedReader extends Reader
  public BufferedReader(Reader in)
    // Creates a buffering character-input stream that uses a default-sized input buffer.
    // Parameters: in - A Reader

public class InputStreamReader extends Reader
  public InputStreamReader(InputStream in)
    // Creates an InputStreamReader that uses the default charset.
    // Parameters: in - An InputStream

public class FileReader extends InputStreamReader
  public FileReader(String fileName) throws FileNotFoundException
    // Creates a new FileReader, given the name of the file to read from.
    // Parameters: fileName - the name of the file to read from
    // Throws: FileNotFoundException - if the named file does not exist,
    //         is a directory rather than a regular file, or for some other
    //         reason cannot be opened for reading.
```