

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: _____

- Do not begin the exam until you are asked to do so.
- Make sure your name and Pennkey (a.k.a. username) is on the top of this page, and that your PennKey is on the front side of every page.
- There are 120 total points. You have 120 minutes to complete the exam. There are 14 pages in the exam.
- There is a separate Appendix for your reference. Answers written in the Appendix will not be graded.

1. OCaml: Higher-order Functions and Binary Search Trees (12 points)

Recall the definitions of generic `transform` and `fold` functions for lists and the type of binary trees and the binary search tree `insert` function, which are all given in Appendix A.

a. (2 points) What value for `v1` does the following program compute? (Choose one.)

```
let v1 : int list = transform (fun x -> x * x) [1; 2; 3]
```

[1; 2; 3] [3; 4; 5] [1; 4; 9] [0; 2; 4] it is ill-typed

b. (2 points) What value for `v2` does the following program compute? (Choose one.)

```
let v2 : int list =  
  transform (fun f -> f 2) [(fun x -> 0); (fun x -> x); (fun x -> x + x)]
```

[0; 1; 2] [0; 2; 4] [1; 2; 3] [2; 4; 6] it is ill-typed

c. (3 points) There are exactly five different binary search trees whose nodes are labeled by the integers 1, 2, and 3. We have drawn two of them below, using our usual convention of omitting the `Empty` constructors. Draw the missing three trees:

tree1	tree2	tree3	tree4	tree5
<pre> 3 / 2 / 1</pre>	<pre> 2 / \ 1 3</pre>			

d. (5 points) We can create a binary tree by repeatedly inserting nodes into the `Empty` tree. Conveniently, the `fold` function (given in Appendix A) allows us to do exactly that. For example, the program below produces the tree drawn as `tree1` in the picture above.

```
let list1 : int list = [1;2;3]  
let tree1 : int tree = fold insert Empty list1
```

Which of the following lists could we pass to `fold` as `list2` to construct `tree2`?

```
let list2 : int list = ???  
let tree2 : int tree = fold insert Empty list2
```

Mark all that are correct:

[1;3;2] [2;3;1] [2;1;3] [3;2;1] [3;1;2]

2. Abstraction and Encapsulation (16 points)

Recall the simple example of a stateful “counter” abstraction that supports two operations: `inc`, which changes the state, and `get` which returns an integer that is equal to the number of times that `inc` has been previously called. We have seen several ways to achieve this (both in OCaml and Java).

Consider the following OCaml module signature and implementation:

```
module type COUNTER = sig
  type t
  val create : unit -> t (* Creates a counter that has been incremented 0 times *)
  val inc : t -> unit (* Increment the counter *)
  val get : t -> int (* Return the number of times the ctr has been inc'd *)
end

module C : COUNTER = struct
  type t = {mutable ctr : int} (* a mutable record containing the field ctr *)
  let create () = {ctr = 0}
  let inc (c:t) : unit = c.ctr <- c.ctr + 1
  let get (c:t) : int = c.ctr
end
```

- a. (6 points) The OCaml type system enforces the abstraction by rejecting some programs *even though* they might yield a well-typed answer when executed by the Abstract Stack Machine. For each of the following code snippets, classify the program as “well-typed” (in which case it can’t crash), “ill-typed but safe” (meaning that the ASM would compute a good answer but the program is not type-correct), or “ill-typed but unsafe” (meaning that the program is ill-typed and the ASM would have to crash because it tries to perform an operation that is nonsensical, like projecting a field from a value that is not dynamically a record value.).

- i. well-typed ill-typed but safe ill-typed but unsafe

```
let x : int =
  let c = C.create () in
  C.inc c;
  C.get c
```

- ii. well-typed ill-typed but safe ill-typed but unsafe

```
let x : int =
  let c = C.create () in
  (C.inc c).ctr
```

- iii. well-typed ill-typed but safe ill-typed but unsafe

```
let x : int =
  let c = C.create () in
  c.ctr
```

- b. (5 points) Now consider this Java interface and class definition for a similar Counter type:

```
interface Counter {
    void inc(); // increment the counter
    int get(); // return the number of times the ctr has been inc'd
}

class C implements Counter {
    public int cnt;
    public void inc() { cnt = cnt + 1; }
    public int get() { return cnt; }
}
```

Does this implementation properly enforce the same abstraction that the OCaml version does? If not, say why and briefly explain how to fix the problem.

- c. (5 points) Now consider this variant of the counter that uses a much less efficient implementation:

```
class C2 implements Counter {
    private List<Boolean> cnt = new LinkedList<Boolean>();
    public void inc() { cnt.add(true); }
    public int get() { return cnt.size(); }
    public List<Boolean> toList() { return cnt; }
}
```

Does this implementation properly enforce the same abstraction that the OCaml version does? If not, say why and briefly explain how to fix the problem.

3. Java concepts (16 points)

a. True False

In Java, if `s` and `t` are values of type `String` such that `s == t` evaluates to **true**, then `s.equals(t)` always evaluates to **true**.

b. True False

In Java, if a method includes the code `throw new IOException()` (where `IOException` is a checked/declared exception) the method signature *must* have a “throws” clause. (e.g., `public void m1() throws IOException`)

c. True False

In Java inheritance, every subclass constructor must call a superclass constructor either implicitly or explicitly.

d. True False

In Java, in a given class, it's possible to have two methods with the same name.

e. True False

In Java, consider a class `C` that implements interfaces `A` and `B`. If interfaces `A` and `B` have four and five methods respectively, it's possible that `C` has exactly eight methods.

f. True False

In Java, if a class overrides `equals`, it must also override `hashCode` in a consistent manner.

g. True False

In Java, if casting of an object succeeds at compile time (i.e., no compile time errors), it's guaranteed to succeed at runtime (i.e., no runtime exceptions).

h. True False

In Java, the type `List<String>` is a subtype of the type `List<Object>`.

4. Java Typing and Dynamic Dispatch (24 points)

This problem uses the Java code shown in Appendix B, which contains an interface and several classes that might be part of a role playing game.

The following main method has some type annotations omitted—as indicated by the blanks. It can be made to successfully typecheck (i.e. compile without errors) in more than one way by filling the blanks with appropriate types.

```
1 public static void main(String[] args) {
2     _____ sarah = new Person("Sarah");
3     sarah.printGreeting();
4
5     _____ bob = new ShopKeeper("Bob", "potatoes", 10);
6     bob.sell();
7     bob.printGreeting();
8
9     _____ mac = new Mimic(sarah);
10    mac.changePersona(bob);
11    mac.printGreeting();
12
13    _____ eve = new EvilTwin(mac.getName());
14    eve.printGreeting();
15 }
```

Indicate which types (there may be one or more) can be correctly used for the declarations...

a. (3 points) of the variable `sarah` on line 2?

Character Person EvilTwin ShopKeeper Mimic

b. (3 points) of the variable `bob` on line 5?

Character Person EvilTwin ShopKeeper Mimic

c. (3 points) of the variable `mac` on line 9?

Character Person EvilTwin ShopKeeper Mimic

d. (3 points) of the variable `eve` on line 13?

Character Person EvilTwin ShopKeeper Mimic

e. (8 points) There are four calls to the method `printGreeting` in the code above. For each call, match it with the (unique) output that will be printed to the terminal when this program runs. Your choices are listed below.

line 3: _____ line 7: _____ line 11: _____ line 14: _____

(1) Hello! I am Sarah.

(5) Hello! I am Evil Sarah.

(2) Hello! I am Bob.

(6) Hello! I am Evil Bob.

(3) Hello! I am Bob.
Would you like to buy some potatoes?
I have 10 in stock.

(7) Hello! I am Evil Mimic Sarah.

(8) Hello! I am Evil Mimic Bob.

(4) Hello! I am Bob.
Would you like to buy some potatoes?
I have 9 in stock.

(9) Hello! I am Evil Mimic Bob.
Would you like to buy some potatoes?
I have 9 in stock.

f. (4 points) Suppose that we add the following two lines of code to the end of the `main` method given above (so that they run *after* all of the previous code).

```
mac.changePersona(mac);  
mac.printGreeting();
```

What behavior would you expect to see when this call to `printGreeting()` executes? (Choose one.)

The program will print:

```
Hello! I am Bob.  
Would you like to buy some potatoes?  
I have 9 in stock.
```

The program will print:

```
Hello! I am Mimic Bob.
```

The program will print:

```
Hello! I am Mimic Bob.  
Would you like to buy some potatoes?  
I have 9 in stock.
```

The program will loop, eventually crashing with a `StackOverflow` exception.

The program will crash with a `NullPointerException`.

5. Java: Swing and Inner Classes (12 points)

This question refers to the Java code for a variant of the Swing `LightBulb` program we saw in class. You can find the code in Appendix C.

a. (3 points) Which line of code defines an anonymous inner class? (Choose one.)

- line 2 line 7 line 24 line 35 line 52

b. (3 points) Suppose we add the following code at line 45.

```
    JButton button2 = new JButton("On/Off2");  
    panel.add(button2);  
    button2.addActionListener(al);
```

How would this change affect the program behavior? (Choose one.)

- The program would fail to compile (there is a type error).
 The program would compile, but fail with an exception when run.
 The program would compile, and when run would have two buttons. The new button does nothing when clicked.
 The program would compile, and when run would have two buttons. The new button flips the bulb state of the *same* lightbulb as the original button.
 The program would compile, and when run would have two buttons. The new button flips the bulb state of a *new* lightbulb, which is not displayed.

c. (6 points) Suppose instead that we add the following method to the GUI class at line 50:

```
    ActionListener makeAL(LightBulb b) {  
        return e -> { b.flip(); b.repaint(); };  
    }
```

Note that this code uses the new Java 8 “lambda” syntax, and the method above will compile. We could now replace lines 35–41 with the following code that has the same functionality:

```
    ActionListener al = makeAL(bulb);
```

True False

The *static* type of the object referred to by the variable `al` is `ActionListener`.

True False

The *dynamic* type of the object referred to by the variable `al` when execution reaches the end of the line above is `ActionListener`.

True False

The object instance referred by the variable `al` when execution reaches the end of the line above must contain a field for `b`.

6. Java: Design Problem – Iterators (40 points)

In this problem, you will use the design process to implement a (small) program that can find quotations in a text file or string. Imagine you had a file with the following contents (excerpt from Charles Dickens' A Christmas Carol):

```
"A merry Christmas, uncle! God save you!" cried
a cheerful voice. It was the voice of Scrooge's
nephew, who came upon him so quickly that this was
the first intimation he had of his approach.
```

```
"Bah!" said Scrooge, "Humbug!"
```

Our program would find the three quotations: A merry Christmas, uncle! God save you!, Bah!, and Humbug!. Once we have a list of all the quotes, we can do additional things like finding the longest quote in the text.

Step 1: Understand the problem. There is nothing to do for this step; your answers below will indicate your understanding.

Step 2: Design the interface. We decide to use Iterators for part of the problem. Similar to the `WordScanner` or `TokenScanner` examples from class and homework, we will create a `QuoteScanner` that implements the `Iterator` interface. The documentation for the `Iterator` interface can be found in Appendix D.

We will create a class `QuoteFinder` that uses this `QuoteScanner` to perform two tasks—to find all quotes in a book and to find the longest quote given a list of quotes. Overall, we will make the following assumptions and design decisions:

- Quotes begin and end with a `"`.
- Quotes can span multiple lines and can be arbitrarily long (i.e., there is no maximum length).
- The file is well-formed and there's an even number of `"` in the text.
- For `QuoteScanner`, the `Iterator` interface methods will have the following behavior:
 - `hasNext()` - Returns true if there is another quote available; false otherwise.
 - `next()` - Returns the next quote without the beginning and ending `"`, or throws a `NoSuchElementException` if none remain.
- For `QuoteFinder`, the methods will have the following behavior:
 - `findAllQuotes()` - Returns the list of all quotes in the text passed to the constructor.
 - `findLongestQuote(List<String> quotes)` - Returns the longest quote from the list of quotes, or throws an `IllegalArgumentException` if `quotes` is null or empty.

Step 3: Write test cases

Below are several example test cases for the various methods above. (No need to do anything but understand them.) A few things to note:

- If you use a `StringReader`, you will need to escape quotes. E.g., if the string you want to test is `They went out, and I howled for Jeeves. "Jeeves! What about it?"`, you need to escape the quotes as shown in the first test case.
- The file `christmas-carol-small.txt` has only the text excerpt shown on the previous page.

```
@Test
public void testGetNextQuote() throws IOException {
    Reader in = new StringReader("They went out, and I howled for Jeeves."
        + "\"Jeeves! What about it?\"");
    QuoteScanner d = new QuoteScanner(in);
    try {
        assertTrue("has next", d.hasNext());
        assertEquals("Jeeves! What about it?", d.next());

        assertFalse("reached end of stream", d.hasNext());
    } finally {
        in.close();
    }
}

/**
 * Finds quotes in a file
 * @param file The filename to read
 */
public static List<String> findQuotesInFile(String file) throws IOException {
    Reader in = new BufferedReader(new FileReader(file));
    QuoteFinder qf = new QuoteFinder(in);

    List<String> quotes = qf.findAllQuotes();
    in.close();

    return quotes;
}

@Test
public void testSimpleQuotes() throws IOException {
    List<String> quotes = findQuotesInFile("files/christmas-carol-small.txt");
    assertEquals("Excerpt has 3 quotes", 3, quotes.size());
}
```

(16 points) You will now write several test cases for the methods described above.

Write four JUnit tests for the program described above. Two of these tests should be for the `QuoteScanner` class and two of the tests should be for the `QuoteFinder` class. The first one you provide should test exceptional circumstances. The four tests should test *distinct* parts of the functionality and should have descriptive test names.

```
@Test (expected=_____ .class)
public void _____ () {
```

```
}
```

```
@Test
public void _____ () {
```

```
}
```

```
@Test
public void _____ () {
```

```
}
```

```
@Test
public void _____ () {
```

```
}
```

Step 4: Implement the behavior

- a. (18 points) Finally, you will now implement the behavior of these classes. First up, `QuoteScanner`. We have provided an incomplete implementation below. Fill out the missing pieces so that all the relevant tests from the previous section pass. The relevant parts of the `Iterator` and `List` interface and `Reader` class are shown in Appendix D.

```
/** Provides a Quote Iterator for a given Reader. */
public class QuoteScanner implements Iterator<String> {
    private Reader in;
    private String quote;

    /**
     * Creates a QuoteScanner for the argued Reader.
     *
     * As an Iterator, the QuoteScanner should only read from the Reader as much as is necessary to
     * determine hasNext() and next(). The QuoteScanner should NOT read the entire stream and compute
     * all of the tokens in advance.
     *
     * @param in The source Reader for character data
     * @throws IllegalArgumentException If the argued Reader is null
     */
    public QuoteScanner(java.io.Reader in) {
        if (in == null) {
            throw new IllegalArgumentException();
        }
        this.in = new BufferedReader(in);
        skipNonQuotes();
    }

    /**
     * Invariant: If there are no more quotes, quote is null, else it contains the next quote
     */
    private void skipNonQuotes() {
        try {

        } catch (IOException e) {

        }
    }
}
```

```

/**
 * Determines whether there is another quote available .
 * We use the invariant described above here.
 * @return true if there is another quote available
 */
public boolean hasNext() {
    return (quote != null);
}

/**
 * Returns the next quote without the beginning and ending ",
 * or throws a NoSuchElementException if none remain.
 * @return The next quote if one exists
 * @throws NoSuchElementException When the end of stream is reached
 */
public String next() {

}

}

/**
 * We don't support this functionality with QuoteScanner, but since the method is required when
 * implementing Iterator , we just <code>throw new UnsupportedOperationException();</code>
 *
 * @throws UnsupportedOperationException Since we do not support this functionality
 */
public void remove() {
    throw new UnsupportedOperationException();
}
}

```

- b. (6 points) Lastly, you will implement `QuoteFinder`. We have provided an incomplete implementation below. Fill out the missing pieces so that all the relevant tests from the previous section pass. (You can ignore the `import` statements.)

```
/**
 * Find quotes from a given Reader
 */
public class QuoteFinder {
    private QuoteScanner qs;

    /**
     * Creates a QuoteScanner from the argued Reader
     * @param in the piece of text to look for quotes
     */
    public QuoteFinder(Reader in) {
        qs = new QuoteScanner(in);
    }

    /**
     * Finds all quotes in a piece of text
     * It will use the Reader passed to the constructor as the source of text
     * @return a list of quotes
     */
    public List<String> findAllQuotes() {

    }

    /**
     * This method will find the longest quote in the list of quotes provided.
     * If there are multiple quotes tied for longest, it returns the first one.
     * It should not read contents from the text again.
     * @param quotes the list of quotes
     * @return the longest quote
     * @throws IllegalArgumentException if the argument is null or empty
     */
    public String findLongestQuote(List<String> quotes) {

    }
}
```