Name:_____

# CSE331 Winter 2014, Final Examination
# March 17, 2014
## Please do not turn the page until 8:30.

Rules:

- The exam is closed-book, closed-note, etc.
- **Please stop promptly at 10:20.**
- There are **116 points** total, distributed **unevenly** among **11** questions (many with multiple parts):

| Question | Max | Earned |
|---|---|---|
| 1 | 10 | |
| 2 | 10 | |
| 3 | 10 | |
| 4 | 13 | |
| 5 | 9 | |
| 6 | 14 | |
| 7 | 6 | |
| 8 | 12 | |
| 9 | 15 | |
| 10 | 9 | |
| 11 | 8 | |

Advice:

- Read questions carefully. Understand a question before you start writing.
- **Write down thoughts and intermediate steps so you can get partial credit. But clearly indicate what is your final answer.**
- The questions are not necessarily in order of difficulty. **Skip around.** Make sure you get to all the problems.
- If you have questions, ask.
- Relax. You are here to learn.

Name:_____

1. (**10** points)

   The following method specification was written correctly by a proud CSE331 graduate. Unfortunately, it is written in an obscure foreign language.

   ```
   @requires snurf flarg
   @modifies blech murph
   @effects roar
   @throws woiefio fi gonzo
   ```

   For parts (a)–(c), give exactly one of these answers, *no explanation required*:

   - *stronger* if the specification given below is definitely stronger than (or equivalent to) the specification above
   - *weaker* if the specification below is definitely weaker than (or equivalent to) the specification above
   - *neither* if the specification given below is definitely incomparable in strength to the specification above
   - *unknown* if answering the question would require knowing the right foreign language

   **Don't miss part (d) below.**

   (a)
   ```
   @modifies blech murph
   @effects roar
   @throws woiefio fi gonzo
   ```

   (b)
   ```
   @requires snurf flarg
   @modifies blech murph
   @throws woiefio fi gonzo
   ```

   (c)
   ```
   @requires snurf flarg
   @modifies blech murph
   @effects roar
   ```

   (d) Suppose this is the method being specified:

   ```
   void foo(Bar x) {
      assert(x != null);
      x.baz(this);
   }
   ```

   Does this additional information change any of your answers above? If so, how? *Explain your answer in 1–2 English sentences.*

   **Solution:**

   (a) stronger (requires less)

   (b) weaker (promises nothing about what effects occur (on the modified objects))

   (c) stronger (promises not to throw exceptions)

   (d) No, the implementation does not change whether specifications are stronger or weaker. In particular, defensive-programming assertions are nice and all but have no effect on specifications.

Name:_____

2. (**10** points) Suppose you try to compile a Java file containing this method definition:

```
class A extends B {
    ...
    public void foo(boolean x, int y) {
        if(x) {
            if(y > 7) {
                bar(); // this is line 8
            } else {
                baz();
            }
        }
    }
}
```

Suppose the type-checker reports only this one error message:

```
final.java:8: error: unreported exception ClosingTimeException; must
be caught or declared to be thrown
                bar();
                   ^
```

(a) Rewrite the *body* of `foo` such that the type-checking error goes away and `foo` behaves the same way if an exception is not thrown. (If an exception is thrown, your code can do anything and we will not grade on style even though in practice what you do would matter.)

(b) Ignoring your answer to part (a) (i.e., assuming you do not do it), instead rewrite the *signature* of `foo` such that the type-checking error goes away and `foo` behaves the same way.

(c) Describe in English how your answer to part (b) might lead to a different type-checking error for this definition of method `foo`, depending on other code in the program.

**Solution:**

(a) A wider scope for the catch block is also okay. We also did not specify what to do in the catch block or that a more general exception couldn't be caught.

```
public void foo(boolean x, int y) {
    if(x) {
        if(y > 7) {
          try {
            bar(); // this is line 8
          } catch (ClosingTimeException e) {
              throw new RuntimeException();
          }
        } else {
            baz();
        }
    }
}
```

(b) `public void foo(boolean x, int y) throws ClosingTimeException`

(c) If `foo` in `A` is overriding a method `foo` defined in `B`, then we cannot add this throws clause unless the method being overridden already has a signature indicating this exception (or a supertype) might be thrown. Note: Some partial credit for talking about `foo`'s callers, but the question asked about a type-checking error for the *definition* of `foo`.

Name:_____

3. (**10** points) Suppose a programmer writes all the following code:

```
// helper method used below: is supposed to sort its argument
private void specialSortingRoutine(int[] arr) { ... }

// helper method used below; requires argument is sorted
private int someOtherRoutine(int[] arr) { ... }

// @requires arr is not null
// @returns <something not relevant to the question>
public int provideValuableService(int[] arr) {
   assert(arr != null);
   specialSortingRoutine(arr);
   assert(arr[0] < arr[1]); // quick sanity check (faster than checking whole array)
   return someOtherRoutine(arr);
}
```

(a) In 1–3 English sentences, describe the *purpose* of the first assertion (`assert(arr != null)`). That is, *why* is it *useful* to have this assertion in this program?

(b) In 1–3 English sentences, describe the *purpose* of the second assertion (`assert(arr[0] < arr[1])`). That is, *why* is it *useful* to have this assertion in this program?

(c) The second assertion is *buggy*.

i. Describe one situation where it can fail even though the code has no bug.

ii. Describe one situation where it can cause an exception to be thrown that would not otherwise be thrown.

iii. Rewrite the second assertion to fix the two problems you described above while still being a very fast (constant-time) check as intended.

**Solution:**

(a) It is defensive programming to check that the caller satisifes the precondition before performing other computations. Because the precondition is cheap to check here, this is good style.

(b) Here the purpose is to check for an error in the code in the sorting routine. This has the potential to catch an error before causing more errors in `someOtherRoutine` and/or from returning a wrong result to the client.

(c) i. If the two (or more) smallest elements in the array are equal

ii. If the array passed by the client has fewer than two elements

iii. `assert(arr.length < 2 || arr[0] <= arr[1]);`

Name:_____

4. (**13** points) Consider these classes (where for simplicity constructors are not shown — assume they take reasonable parameters and are correct):

```
// Represents immutable drink orders.
// Each drink has a name, a cost, and a size that is either small or large.
class BeverageOrder {
  protected String name;
  protected int numOunces; // (12 for small, 30 for large)
  protected int cost;

  // @returns the name of the beverage (e.g., "root beer")
  public String drinkName() { return name; }

  // @returns true if the order size is small
  public boolean isSmall() { return numOunces==12; }

  // @returns true if the order size is large
  public boolean isLarge() { return !isSmall(); }

  // @returns the cost in cents of this drink
  public int cost() { return cost; }
}

// Represents immutable drink orders.
// Each drink has a name, a cost, and a size that is one of small, medium, large.
class BeverageOrderWithMedium extends BeverageOrder {
  public boolean isMedium() { return numOunces==20; }
}
```

(a) Is `BeverageOrderWithMedium` a Java subtype of `BeverageOrder`? Explain your answer in 1–2 sentences.

(b) Is `BeverageOrderWithMedium` a true subtype of `BeverageOrder`? Explain your answer in 1–2 sentences.

(c) The implementation of `BeverageOrderWithMedium` has a bug. Explain in 1–2 sentences what the bug is.

(d) Propose a fix for the bug you identified in part (c). Indicate the exact change you would make to the code.

**Solution:**

(a) Yes, every subclass is always a subtype and this particular subclass type-checks because it only adds new methods (that type-check).

(b) No, it has an incomparable specification. Clients of the supertype can assume that either `isSmall` or `isLarge` returns true, but instances of the subtype do not necessarily meet this specification.

(c) The `isLarge` method returns `true` when `numOunces==20`, i.e., the object has size medium.

(d) Full credit for either changing `isLarge` in the supertype or overriding it in the subtype, though often only the latter is an option. Either way, the body should be `return numOunces==30;`.

Name:_____

*More room for your answer to question 4 in case you need it. Don't panic if you don't need it.*

Name:_____

5. (**9** points) In this problem, assume we already have Java types `HighHeeledShoe`, `Shoe`, and `FootWear` defined where `FootWear` is a supertype of `Shoe` and `Shoe` is a supertype of `HighHeeledShoe`. We also have this Java class:

```
class Foo extends Object {
   Shoe m(Shoe x, Shoe y) { ... }
}
```

and a Java subclass:

```
class Bar extends Foo {
  ...
}
```

For each method below, if the method were part of class `Bar`, indicate which one of the following would be true in Java by writing the italicized word (no explanation required):

- The result is method *overriding*
- The result is method *overloading*
- The result is a *type-error*
- *None* of the above

(a) `FootWear m(Shoe x, Shoe y) { ... }`

(b) `Shoe m(Shoe q, Shoe z) { ... }`

(c) `HighHeeledShoe m(Shoe x, Shoe y) { ... }`

(d) `Shoe m(FootWear x, HighHeeledShoe y) { ... }`

(e) `Shoe m(FootWear x, FootWear y) { ... }`

(f) `Shoe m(Shoe x, Shoe y) { ... }`

(g) `Shoe m(HighHeeledShoe x, HighHeeledShoe y) { ... }`

(h) `Shoe m(Shoe y) { ... }`

(i) `Shoe z(Shoe x, Shoe y) { ... }`

**Solution:**

(a) type-error

(b) overriding

(c) overriding

(d) overloading

(e) overloading

(f) overriding

(g) overloading

(h) overloading

(i) none

Name:_____

6. (**14** points) **Don't miss that this problem has a part (b) on the next page.**

In this problem, assume these class definitions:

```
class Animal {
   void eat(Food f) {...}
}
class Dog extends Animal {
   void bark() {...}
}
class PuppyDog extends Dog {
   int cutenessLevel(int x) { return x * 1000; }
}
```

Now consider this method:

```
void addAfterEverybodyBarks(List<Dog> dogs, Dog newDog) {
   for(Dog d : dogs)
      d.bark();
   newDog.bark();
   dogs.add(newDog);
}
```

(a) For *each* call to `addAfterEverybodyBarks` below, indicate whether or not the code type-checks — just write "yes" for type-checks or "no" for does not type-check.

```
List<Animal>   la = new ArrayList<Animal>();
List<Dog>      ld = new ArrayList<Dog>();
List<PuppyDog> lp = new ArrayList<PuppyDog>();
Animal   a = new Animal();
Dog      d = new Dog();
PuppyDog p = new PuppyDog();

addAfterEverybodyBarks(la, a);

addAfterEverybodyBarks(ld, a);

addAfterEverybodyBarks(lp, a);

addAfterEverybodyBarks(la, d);

addAfterEverybodyBarks(ld, d);

addAfterEverybodyBarks(lp, d);

addAfterEverybodyBarks(la, p);

addAfterEverybodyBarks(ld, p);

addAfterEverybodyBarks(lp, p);
```

(b) Show how to change the `addAfterEverybodyBarks` method by making it more generic so that (1) all the calls above that type-check still do, (2) at least one more call type-checks and (3) the method body has the same behavior. Indicate which method call or calls now type-check that did not before.

**Solution:**

i. The only two that type-check are:

```
addAfterEverybodyBarks(ld, d);
addAfterEverybodyBarks(ld, p);
```

ii. Note you cannot use wildcards here.

```
<T extends Dog> void addAfterEverybodyBarks(List<T> dogs, T newDog) {
   for(Dog d : dogs) // Also okay for d to have type T
      d.bark();
   newDog.bark();
   dogs.add(newDog);
}
```

Now this (and only) this call also type-checks:

```
addAfterEverybodyBarks(lp, p);
```

Name:_____

7. (**6** points) For each of the Java method signatures below, give an equivalent method signature that does not use wildcards.

   (a) `String m1(List<? extends Foo> x);`

   (b) `void m2(List<? extends Foo> x, List<? extends Foo> y);`

   (c) `<T> void m3(List<? extends T> x, List<Object> y, List<T> z);`

   **Solution:**
   Choice of type-variable name does not matter.

   (a) `<T extends Foo> String m1(List<T> x);`

   (b) `<T1 extends Foo, T2 extends Foo> void m2(List<T1> x, List<T2> y);`

   (c) `<T,T2 extends T> void m3(List<T2> x, List<Object> y, List<T> z);`

Name:_____

8. (**12** points)

   (a) Describe *two different* steps you should take after receiving a bug report *before* looking at the program code to try to identify the cause. (One sentence each should be plenty.)

   (b) Give *two different* reasons a failure might occur only when assertions are disabled. (One sentence each should be plenty.)

   (c) Describe how binary search can be used effectively in debugging. (A few sentences should be plenty.)

   **Solution:**

   (a) Here are three: Make sure the bug is reproducible. Find a minimal input that reproduces the bug. Add a failing test case to the project's test suite. We accepted a number of other answers as well.

   (b) A bad assertion may have a side effect that causes the failure not to occur. Timing changes may cause the failure not to occur.

   (c) Find a point in the execution where no error has occurred and a later point where an error has occurred. Now check the program state halfway inbetween to see if an error has occurred there. Repeat with either the first half or the second half of the interval until the defect is found.

Name:_____

9. (**15** points) Here is an alphabetical list of some design patterns we studied. Note some patterns are more specific instances of other patterns.

   Adapter, Builder, Composite, Decorator, Factory, Flyweight, Intern, Interpreter, Observer, Procedural, Prototype, Proxy, Singleton, Visitor, Wrapper

   For each statement below, list *all* design patterns from the list above that meet the description. Your answer may include one or more than one design pattern. (Zero is not the right answer.)

   (a) The pattern involves creating a class where most of the functionality is provided by one other class.

   (b) The pattern should be used only for immutable classes.

   (c) The pattern is fundamental to how Java's GUI libraries are organized.

   (d) To *require* clients to use the pattern, it is necessary to make constructors private.

   (e) The pattern is a way to implement the Procedural pattern without using `instanceof` tests.

   (f) One of the purposes of the pattern can be to delay creating an object that may be expensive to create.

   **Solution:**

   (a) Adapter, Decorator, Proxy, Wrapper

   (b) Intern (and full credit either way for Flyweight since it's ambiguous: the intrinsic state should be immutable but the extrinsic state is not)

   (c) Observer

   (d) Builder, Factory, Prototype, Singleton, Intern

   (e) Visitor (and we allowed also listing Procedural)

   (f) Singleton, Proxy (and full credit either way for Wrapper since Proxy is a kind of Wrapper)

Name:_____

10. (**9** points)

    (a) The purpose of a Model-View-Controller design is to keep code for the model, the view, and the controller separate. The Java GUI library's use of listeners and callbacks helps keep two of these three things separate. Which two? (No explanation required.)

    (b) Why is it bad for a GUI callback method to take a long time to return? (1–2 sentences should be plenty.)

    (c) What happens if a client of `JButton` calls `addActionListener` on the same button multiple times passing in different objects? (1–2 sentences should be plenty.)

    (d) What happens if a client of `JButton` calls `addActionListener` on multiple buttons passing in the same object? (1–2 sentences should be plenty.)

    (e) True or false (no explanation required): To register an event listener in Java's GUI library, you need to create an anonymous inner class.

    **Solution:**

    (a) The view and the controller

    (b) It executes on the UI thread, so the application cannot respond to any GUI interactions while the callback executes.

    (c) They are all registered and all get notified when the button gets pressed.

    (d) The object's `actionPerformed` object will be called when any of the buttons is pressed. It can use its argument to determine which button was pressed.

    (e) False

Name:_____

11. (**8** points)

    (a) For each of the following, indicate whether you would prefer a top-down or a bottom-up implementation strategy. No explanations required.

        i. You are concerned that the design includes too few user-visible features.

        ii. You are concerned the system relies fundamentally on new image-processing algorithms that might be too slow.

        iii. Writing test drivers will be unusually difficult.

        iv. Writing stubs will be unusually difficult.

    (b) Give two different reasons to write code that is never intended to be part of the code delivered in a final product.

    (c) Why is "80% of the code is written" a poor project milestone?

    **Solution:**

    (a) i. top-down

        ii. bottom-up

        iii. top-down

        iv. bottom-up

    (b) (1) You need automated tests and test harnesses. (2) You often need to write stubs for modules that are not yet implemented. (3) There are other things like automatically building, logging results, etc., that can also be considered coding.

    (c) It is not verifiable: because we don't know how much code is needed, we can't know when we have reached 80%.