**University of Washington**
**CSE 331 Software Design & Implementation**
**Spring 2013**

# Final exam

**Monday, June 10, 2013**

**Name:** ______________________________

**Section:** ______________________________

**CSE Net ID (username):** ______________________________

**UW Net ID (username):** ______________________________

---

This exam is closed book, closed notes. You have **110 minutes** to complete it. It contains 38 questions and 13 pages (including this one), totaling 220 points. Before you start, please check your copy to make sure it is complete. Turn in all pages, together, when you are finished. **Write your initials on the top of *ALL* pages** (in case a page gets separated during test-taking or grading).

**Please write neatly**; we cannot give credit for what we cannot read.

Good luck!

| Page | Max | Score |
|---|---|---|
| 2 | 26 | |
| 3 | 8 | |
| 4 | 24 | |
| 5 | 18 | |
| 6 | 18 | |
| 7 | 24 | |
| 8 | 24 | |
| 9 | 24 | |
| 10 | 16 | |
| 11 | 16 | |
| 12 | 22 | |
| Total | 220 | |

## 1 True/False

**(2 points each) Circle the correct answer. T is true, F is false.**

1. **T / F** The `List` method `get` returns a reference to an element in the list, and therefore commits representation exposure.
2. **T / F** A specification for method `m` that says "`m` may modify variable `x`" is not useful to clients, because it does not give the client knowledge of the post-value in the same way as "`m` increments `x` by 1" (or, equivalently, "$x_{post¿} = x_{pre} + 1$").
3. **T / F** Every getter method is an observer method, but not all observer methods are getters.
4. **T / F** It is possible (and useful) for a specification to mention a specification field that does not correspond to any concrete field or concrete method result.
5. **T / F** It is permitted to specify a precondition (@requires clause), *and* for the implementation to check that condition and perform a specific action (such as throwing a specific exception).
6. **T / F** If every method in a Java class returns only immutable objects, then the class is immutable.
7. **T / F** In an ADT, more than one concrete state can represent the same abstract value.
8. **T / F** A test suite that detects every defect in an implementation has 100% statement coverage.
9. **T / F** Specification tests and black box tests are different names for the same concept.
10. **T / F** Implementation tests and white box tests are different names for the same concept.

For the remaining questions, suppose you have an implementation and a test suite that detects every defect in that implementation (for example, the test inputs were generated by taking a set of revealing subdomains and choosing one element from each).

11. **T / F** If you change the implementation without changing the specification or the tests, the test suite still detects every flaw in the implementation.
12. **T / F** If you change the specification without changing the implementation or the tests, the test suite still detects every flaw in the implementation.
13. **T / F** If you change the implementation in a correct way, without changing the specification or the tests, then the implementation tests may fail.

## 2 Multiple choice — single answer

**For each part of each question, circle the *single* best choice.**

14. (4 points) Given Spec A which asserts

```
    @requires value occurs in a
    @returns i such that a[i] = value
    int find(int value, int[] a)
```

and Spec B which asserts

```
    @returns i such that a[i] = value
       or -1 if value is not in a
    int find(int value, int[] a)
```

Mark one of the following:

(a) Spec A is stronger than spec B
(b) Spec B is stronger than spec A
(c) The specifications are equally strong
(d) The specification are incomparable (they differ, and neither is stronger than the other)

15. (4 points) Choose the best specification for the `add` method for a collection class:

```
public void add(E element) {
   if (element == null) {
      throw new IllegalArgumentException();
   }
   ... // do things that add element to the collection
}
```

(a)
```
@requires nothing
@throws   IllegalArgumentException when element is null
@modifies this
@effects  adds element to this collection
```

(b)
```
@requires nothing
@modifies this
@effects  adds element to this collection
```

(c)
```
@requires element is not null
@throws   IllegalArgumentException if element is null
@modifies this
@effects  adds element to this collection
```

(d)
```
@requires element is not null
@modifies this
@effects  adds element to this collection
```

16. (8 points) For each part of each question, circle the single best choice from among those in curly braces.

    - True subtyping for generics is the same as **{ contravariant / covariant / invariant }** subtyping.
    - Java generics use **{ contravariant / covariant / invariant }** subtyping.
    - True subtyping for arrays is the same as **{ contravariant / covariant / invariant }** subtyping.
    - Java arrays use **{ contravariant / covariant / invariant }** subtyping.

17. (16 points) Suppose you have defined multiple objects (such as representations of code expressions) and multiple operations on each object (such as type-checking and printing). There is an implementation of each operation for each each object (the blank cells of this table):

| | | Objects | |
|---|---|---|---|
| | | CondExpr | EqualOp |
| Operations | type-check | | |
| | pretty-print | | |

Selecting an implementation to execute is equivalent to selecting a row and selecting a column. When is are the selections made, and based on what information?

The possible answers for when the selection is made are:

- run time
- compile time

The possible answers for how the selection is made are:

- by dynamic dispatch on the type of the receiver object
- by overloading resolution on the types of the arguments
- by `instanceof` tests
- by the name of the method being invoked

Circle one choice for each place you have an option below.

(a) For the interpreter pattern, the column is determined at **{ compile / run }** time by **{ dispatch / overloading / instanceof / name }**.

(b) For the interpreter pattern, the row is determined at **{ compile / run }** time by **{ dispatch / overloading / instanceof / name }**.

(c) For the visitor pattern, the column is determined at **{ compile / run }** time by **{ dispatch / overloading / instanceof / name }**.

(d) For the visitor pattern, the row is determined at **{ compile / run }** time by **{ dispatch / overloading / instanceof / name }**.

## 3 Multiple choice — multiple answers

**Mark all of the following that must be true, by circling the appropriate letters.**

18. (9 points) Sometimes, attempting a formal proof of a particular software component is not a productive use of time. Mark each reason that justifies *not* attempting a formal proof.

    (a) The code is complex, making the formal proof difficult.
    (b) The code is simple and therefore is unlikely to contain unnoticed errors.
    (c) Representation exposure is not possible, eliminating a major source of errors.
    (d) Representation exposure is a possibility, complicating the proof.
    (e) The specification is vague, making the goal of the formal proof unknowable.
    (f) Visual appearance is the measure of quality, as in the look of user interface components.
    (g) It is early in the design process, when the specification is subject to change.
    (h) The code is returning results for an Internet search algorithm, where usefulness to people is the measure of quality.
    (i) The code is an isolated or relatively unimportant module, whose failure does not jeopardize the success of the overall system.

19. (5 points) What kinds of files should be committed to your source control repository?

    (a) code files
    (b) documentation files
    (c) output files
    (d) automatically-generated files that are required for your system to be used
    (e) your "scientific notebook" of experiments about debugging a problem

20. (4 points) Which of the following are benefits of good software architecture? Select all that apply.

    (a) a well architected system will uses parallel processing
    (b) a well architected system will help ensure components which are are constructed separately will interact correctly.
    (c) a well architected system allows clear monitoring of development progress
    (d) a well architected system will cluster code in as few classes as possible (often only one)

21. (6 points) Assume that the following code passes the Java typechecker:

```
A a = new A();
B b = new C(a);
D d = b;
```

    Mark all of the following that must be true:

    (a) `A` is a supertype of `B`
    (b) `A` is a subtype of `B`
    (c) `A` is a supertype of `C`
    (d) `A` is a subtype of `C`
    (e) `A` is a supertype of `D`
    (f) `A` is a subtype of `D`
    (g) `B` is a supertype of `C`
    (h) `B` is a subtype of `C`
    (i) `B` is a supertype of `D`
    (j) `B` is a subtype of `D`
    (k) `C` is a supertype of `D`
    (l) `C` is a subtype of `D`

22. (6 points) Mark every word that completes the following sentence to make a true statement.

    In Java, every ______________________ has a compile-time type.

    (a) declaration
    (b) expression
    (c) statement
    (d) value
    (e) variable

23. (6 points) Suppose that you have a private inner class that is only intended to be instantiated once. What are the benefits of writing the class as an anonymous inner class instead?

    (a) shorter code (less boilerplate code)
    (b) prevents the client from accessing the class
    (c) prevents the client from instantiating the class

## 4 Short answer

24. (8 points) Suppose that a specification field's value can be determined from the values of other specification fields. Such a relationship will be stated in the specification. You may declare the specification field as a regular specification field or as a derived specification field.

    In one sentence, state an advantage of declaring the specification field as a derived specification field rather than as a regular specification field.

25. (8 points) Given the following declarations, write additional code that contains no casts and satisfies the Java type system (that is, there is no compile-time error or warning), but does not satisfy true subtyping and therefore crashes at run time due to an `ArrayStoreException`.

```
Date[] da = new Date[] { new Date() };
// The above line is the same as:  Date[] da = new Date[1]; da[0] = new Date();
Object[] oa = new Object[] { "hello" };
```

26. (8 points) Write a `min` method that returns the smaller of its two arguments and passes the following two tests. You do not have to write documentation — just the code. For ease of grading, please name the two formal parameters `a` and `b`.

```
assert min(3, 4) == 3;
assert min(2.718, 3.14) == 2.718;
assert min("hello", "goodbye") == "goodbye";
```

27. (8 points) Unlike the CSE 331 Campus Maps application, the UW map application at `http://www.washington.edu/maps/` permits users to select a building using either free text search *or* a dropdown menu.

    (a) Explain a circumstance and/or class of users for whom the free text search is likely to be more useful, and explain why. (1–2 sentences)

    (b) State a circumstance and/or class of users for whom the dropdown menu is likely to be more useful, and explain why. (1–2 sentences)

28. (4 points) Name the key design pattern used in event-driven programming, in one word or short phrase: The ________________ Pattern.

29. (12 points) State three benefits of using a version control system. Use one sentence or phrase each. State benefits that are as different from one another as possible.

    (a)

    (b)

    (c)

30. (4 points) In no more than one sentence, state an advantage of bottom-up implementation.

31. (4 points) In no more than one sentence, state an advantage of top-down implementation.

32. (12 points) Suppose that you have written a class that contains a private method. Your test class cannot call that method. In one sentence each, give three distinct approaches for writing tests that detect errors in the private method's implementation. Give reasons that are as different from one another as possible.

    (a)

    (b)

    (c)

    (4 points) State which of the three is best, and justify your choice with a single sentence.

## 5 Free Willy (from ignorance)

33. (8 points) Recall Willy Wazoo's implementation of the absolute value function:

```
int abs(int x) {
  if (x < -2) return -x;
  else return x;
}
```

Suppose that you partition a procedure's domain so that every subdomain is revealing. (Recall that in a partition, each input is in exactly one subdomain.) Then creating a set of test inputs by choosing one element from each subdomain will guarantee the discovery of the error.

Give a different partition of the domain of `abs` — not into revealing subdomains — such that creating a set of test inputs by choosing one element from each subdomain still guarantees the discovery of the error.

34. (8 points) Willy Wazoo specified the following method, but he forgot to write down the generic types.

```
  /**
   * Make "pruned" a copy of "orig" without duplicates.  Does not modify "orig".
   * @modifies pruned
   */
  static void removeDuplicates(Collection orig, Collection pruned);
```

Please rewrite the entire method signature (you don't have to rewrite the Javadoc) with the most appropriate signature.

35. (16 points) Willy Wazoo has written the following code. He is sure that the final line, `count.toString()`, never throws a `NullPointerException`, but he isn't sure *why* he's sure.

    Help Willy by adding enough Nullness Checker annotations to the class `Keiko` to guarantee that `count.toString()` does not throw a `NullPointerException`, (under the assumption that every method body satisfies its specification as expressed by the annotations). Write all necessary properties; do not rely on default annotations. Do not write any extra, unnecessary annotations. Do not use postcondition annotations such as `@AssertNonNullAfter`.

    Write the annotations in the space on the right-hand-side of the page, and draw an arrow from each to its code location on the left-hand side.

```
class Keiko {

  Map<String, Integer> counts;

  String name;

  Keiko() {
    ...
  }

  String getAString() {
    ...
  }

  void doSomeThings() {
    ...
  }

  Object doOtherThings() {
    ...
  }

  void doMoreThings(Object arg) {
    ...
  }

  void myMethod() {
    doSomeThings();
    name = getAString();
    Object o = doOtherThings();
    Integer count = counts.get(name);
    doMoreThings(o);
    count.toString();
  }

}
```

Willy Wazoo has written the `Point2D`, `Point3D`, and `Point4D` classes that appear on page 13.

Answer the following questions without defining new classes or modifying the existing code.

36. (6 points) Give an example of how the **symmetry** of equals could be broken.
The end of your answer should show a series of calls to `equals` and should state the values to which they evaluate.

37. (6 points) Give an example of how the **transitivity** of equals could be broken.
The end of your answer should show a series of calls to `equals` and should state the values to which they evaluate.

38. (10 points) Mark all of the following `hashCode` implementations that would be valid (they satisfy the spec of `hashCode`) in the `Point3D` class.

(a)
```
public int hashCode() {
  return 42;
}
```

(b)
```
public int hashCode() {
  return (new Random()).nextInt();
}
```

(c)
```
public int hashCode() {
  return x + y;
}
```

(d)
```
public int hashCode() {
  return x * y * z;
}
```

(e)
```
public int hashCode() {
  if (this instanceof Point4D)
    return new Random().nextInt() * ((Point4D) this).w;
  else
    return z;
}
```

You may tear off this page if you wish. You do **not** need to turn it in.

```
public class Point2D {
  protected int x, y;

  public Point2D(int x, int y) {
    this.x = x;
    this.y = y;
  }

  public boolean equals(Object o) {
    if (! (o instanceof Point2D))
      return false;
    return ((Point2D) o).x == x &&
           ((Point2D) o).y == y;
  }
}

public class Point3D extends Point2D {
  protected int z;

  public Point3D(int x, int y, int z) {
    super(x, y);
    this.z = z;
  }

  public boolean equals(Object o) {
    if (!(o instanceof Point3D))
      return false;
    return ((Point3D) o).x == x &&
           ((Point3D) o).y == y &&
           ((Point3D) o).z == z;
  }
}

public class Point4D extends Point3D {
  protected int w;

  public Point4D(int x, int y, int z, int w) {
    super(x, y, z);
    this.w = w;
  }

  public boolean equals(Object o) {
    if (o instanceof Point4D) {
      return ((Point4D) o).x == x &&
             ((Point4D) o).y == y &&
             ((Point4D) o).z == z &&
             ((Point4D) o).w == w;
    } else if (o instanceof Point3D) {
      return ((Point3D) o).equals(this);
    } else {
      // not a Point3D
      return false;
    }
  }
}
```