Quiz 2 (March 12, 2014)

Your name:______________________________________________________

Your Athena username:__________________________________________

Circle your recitation time: 10am 11am 12pm 1pm 2pm

You have 50 minutes to complete this quiz. It contains 8 pages (including this page) for a total
of 100 points.

The quiz is closed-book and closed-notes, but you are allowed one two-sided page of notes.

Please check your copy to make sure that it is complete before you start. Turn in all pages,
together, when you finish. Before you begin, write your name on the top of every page.

Please write neatly. **No credit will be given if we cannot read what you write.**

For questions which require you to choose your answer(s) from a list, do so clearly and unam-
biguously by circling the number(s) or entire answer(s). Do not use check marks, underlines,
or other annotations – they will not be graded.

Good luck!

<table>
<thead>
<tr>
<th>Problem</th>
<th>Points</th>
<th>Grade</th>
<th>Grader</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: Short Answer</td>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2: Abstract Data Types</td>
<td>30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3: Rep Invariants</td>
<td>25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4: Inheritance</td>
<td>30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>100</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Problem 1 (Short Answer) (15 points).
Circle all correct answers for the following questions.

(a) Consider the datatype definition:
Geology = Core(a:int, b:int, c:int, d:int) + Planet(core:Core, a:int, b:int, c:int, d:int) + System(geology:Geology, a:int, b:int)
Suppose you have a reference to a Geology object. How many integers could it have in its representation?
A. 1
B. 2
C. 4
D. 8
E. 10
F. 12

(b) Suppose you hear the following statements in a discussion. Which statements are sensible?
A. “The constructor’s precondition is violated, so it’s legal for it to create an object that violates the rep invariant.”
B. “This Java interface needs to have a rep invariant written in a comment.”
C. “The way to tell that a class is immutable is to look for the final keyword in its instance variables.”
D. “An immutable class cannot mutate any of its instance variables once the constructor returns.”
E. “hashCode() is a good shortcut for testing equality – if two things have the same hashCode(), you know they’re equal.”

(c) Consider the following code, which is supposed to be an implementation of Java’s String class.

```java
public class String {
    public final List<Character> chars = new ArrayList<Character>();
}
```
Which of the following statements are true?

A. The class’s immutability is threatened because the rep is public.
B. The class’s immutability is threatened because it uses Character.
C. The class’s immutability is threatened because it uses final.
D. This rep is immutable because it uses final.
Problem 2 (Abstract Data Types) (30 points).
Suppose you have an abstract datatype Vegetable with the following operations:

leafy: Vegetable -> boolean
    requires: non-null Vegetable
    returns: true if and only if the Vegetable is leafy

crunch: Vegetable -> int
    requires: non-null Vegetable
    returns: crunchiness as an integer from 1 (mushy) to 10 (teeth shattering)

crunchiest: List<Vegetable> -> Vegetable
    requires: ... see part (a) below ...
    returns: ... see part (a) below ...

We'll implement this ADT with a Java class:

class Vegetable {

    public Vegetable(...) {
        ...
    }

    public boolean leafy() {
        ...
    }

    public int crunch() {
        ...
    }

    public static Vegetable crunchiest(List<Vegetable> veggies) {
        ...
    }
}
(a) Below, the letters A, B, and C are different specifications for the *crunchiest* operation, and the diagrams at the bottom show different possible relationships among these specifications, when the ovals are interpreted as sets of legal implementations for each specification.

A) requires: list of Vegetables  
effects: returns a vegetable with greatest crunchiness

B) requires: list of Vegetables  
effects: returns the vegetable with greatest crunchiness, or throws ChompingException in the case of a tie

C) requires: non-empty list of Vegetables  
effects: returns the vegetable with greatest crunchiness, or throws ChompingException in the case of a tie

Circle the number of the diagram that correctly describes the relationship among A, B, and C.  
On the diagram you chose, label each oval with A, B, and C, using the blanks that stick out from each oval.
(b) For each of the following implementations of a Java `equals` method for `Vegetable`, select the **single best answer**:

```java
@Override
public boolean equals(Object obj) {
    return this == obj;
}
```

(circle only one)

A. not reflexive  
B. not transitive  
C. not symmetric  
D. causes a compile-time error due to types  
E. causes a runtime error, even for valid input  
F. correctly implements observational equality  
G. none of the above

```java
@Override
public boolean equals(Object obj) {
    if (! (obj instanceof Vegetable)) { return false; }
    return this.leafy() == obj.leafy() && this.crunch() == obj.crunch();
}
```

(circle only one)

A. not reflexive  
B. not transitive  
C. not symmetric  
D. causes a compile-time error due to types  
E. causes a runtime error, even for valid input  
F. correctly implements observational equality  
G. none of the above

```java
@Override
public boolean equals(Object obj) {
    if (! (obj instanceof Vegetable)) { return false; }
    Vegetable v = (Vegetable)obj;
    return this.leafy() == v.leafy() && Math.abs(this.crunch() - v.crunch()) < 3;
}
```

(circle only one)

A. not reflexive  
B. not transitive  
C. not symmetric  
D. causes a compile-time error due to types  
E. causes a runtime error, even for valid input  
F. correctly implements observational equality  
G. none of the above
(c) Below are three possible representations for Vegetable. For each representation, the left column shows the rep and rep invariant, the middle column shows how two key methods are implemented with that rep, and the right column shows part of the abstraction function.

Each column has some blanks for you to fill in. **Fill in the blanks** so that all three representations are correct and consistent.

<table>
<thead>
<tr>
<th>Rep Operations</th>
<th>Operation implementations</th>
<th>Abstraction function</th>
</tr>
</thead>
<tbody>
<tr>
<td>private boolean leafy;</td>
<td>public boolean leafy() { return leafy; }</td>
<td>(true, 5) -&gt; (true, 5)</td>
</tr>
<tr>
<td>private int crunch;</td>
<td></td>
<td>(false, 15) -&gt; (false, 10)</td>
</tr>
<tr>
<td>// rep invariant:</td>
<td>}</td>
<td>(true, -5) -&gt; not mapped</td>
</tr>
<tr>
<td>//</td>
<td>public int crunch() {</td>
<td></td>
</tr>
<tr>
<td>// _______________</td>
<td>return Math.min(crunch,10);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>}</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rep Operations</th>
<th>Operation implementations</th>
<th>Abstraction function</th>
</tr>
</thead>
<tbody>
<tr>
<td>private boolean leafy;</td>
<td>public boolean leafy() { return leafy; }</td>
<td>(true, 5) -&gt; (true, 5)</td>
</tr>
<tr>
<td>private int crunch;</td>
<td></td>
<td>(false, 15) -&gt; not mapped</td>
</tr>
<tr>
<td>// rep invariant:</td>
<td></td>
<td>(true, -5) -&gt; not mapped</td>
</tr>
<tr>
<td>// 1 &lt;= crunch &lt;= 10</td>
<td>}</td>
<td></td>
</tr>
<tr>
<td></td>
<td>public int crunch() {</td>
<td></td>
</tr>
<tr>
<td></td>
<td>}</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rep Operations</th>
<th>Operation implementations</th>
<th>Abstraction function</th>
</tr>
</thead>
<tbody>
<tr>
<td>private int score;</td>
<td>public boolean leafy() { return score &lt; 0; }</td>
<td>(5) -&gt; ________</td>
</tr>
<tr>
<td>// rep invariant:</td>
<td>}</td>
<td>(15) -&gt; ________</td>
</tr>
<tr>
<td>// true</td>
<td>public int crunch() {</td>
<td></td>
</tr>
<tr>
<td></td>
<td>return (Math.abs(score) % 10) + 1;</td>
<td>(-5) -&gt; ________</td>
</tr>
<tr>
<td></td>
<td>}</td>
<td></td>
</tr>
</tbody>
</table>
Problem 3 (Rep Invariants) (25 points).
Consider the following abstract data type, which has some placeholders shown in all caps.

```java
/**
 * Room is an immutable datatype representing a room with a width and a height,
 * and a set of obstacles that a robot might have to maneuver around.
 * It makes use of Point, an immutable datatype representing 2D positions (x,y).
 */
public class Room {

    private final double width, height;
    private final String roomNumber;
    private Set<Point> obstacles;
    /* REP INVARIANT */
    ... 

    /**
    * requires: PRECONDITION
    * effects: POSTCONDITION
    */
    public List<Point> findPath(Point initial, Point goal) throws PathNotFoundException {
        ... 
        List<Point> path = new ArrayList<Point>();
        ...
        return path;
    }
    ...
}
```

From the following list, choose the necessary parts for REP INARIANT, PRECONDITION, and POST-
CONDITION. Write RI, PRE, or POST next to each statement according to whether it belongs in the
rep invariant, precondition, or postcondition comment. Leave a statement blank if it is redundant with or
contradicts information that’s already present in the code above.

______ Point objects in obstacles satisfy 0<=x<=width and 0<=y<=height
______ initial and goal both satisfy 0<=x<=width, 0<=y<=height
______ width > 0 and height > 0
______ throws PathNotFoundException if no obstacle-free path exists from initial to goal
______ returns a list of points that form a path from initial to goal
    without colliding into any obstacles.
______ roomNumber consists only of letters, digits, dashes, or spaces
______ roomNumber is a mutable string of characters
Problem 4 (Inheritance) (30 points).
Consider these two classes:

```java
public class Shape {
    public void match(Shape s) {
        System.out.println(name() + " has a " + s.name());
    }
    public String name() { return "Shape"; }
}
public class Triangle extends Shape {
    public void match(Triangle t) {
        System.out.println(name() + " got a " + t.name());
    }
    public String name() { return "Triangle"; }
}
```

For each code fragment below, determine what happens when it executes. Either show what the code prints by circling the right words or phrases from each column, or circle Error if the code produces a compile-time error or runtime exception. **Assume the code fragments appear in sequence in a Java file, so that a fragment can use the variables declared in the earlier fragments.**

(a) Shape s = new Triangle();
    s.match(s);

```
Shape has a Shape
Triangle got a Triangle
```

Error

(b) Triangle t = new Triangle();
    t.match(t);

```
Shape has a Shape
Triangle got a Triangle
```

Error

(c) t.match(s);

```
Shape has a Shape
Triangle got a Triangle
```

Error

(d) Triangle u = new Shape();
    u.match(u);

```
Shape has a Shape
Triangle got a Triangle
```

Error