Quiz 1 (March 22, 2017)

Your name:_________________________________________________________

Your Athena username:______________________________________________

You have 50 minutes to complete this quiz. It contains 10 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 unambiguously by circling the letter(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>
</tr>
</thead>
<tbody>
<tr>
<td>1: Specifications</td>
<td>21</td>
</tr>
<tr>
<td>2: Testing</td>
<td>16</td>
</tr>
<tr>
<td>3: ADTs</td>
<td>25</td>
</tr>
<tr>
<td>4: Code Review</td>
<td>20</td>
</tr>
<tr>
<td>5: Multiple Choice</td>
<td>18</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>100</strong></td>
</tr>
</tbody>
</table>
Problem 1 (Specifications) (21 points).
Given the code, answer the questions below.

```java
/** Schedule represents a schedule of non-overlapping events */
public class Schedule {
    private final List<Event> events;
    // events is sorted in increasing order of start time
    // AND for all i<j, events[i].end <= events[j].start

    /**
     * Make a new Schedule
     * @param events a list of non-overlapping events sorted by increasing start time
     */
    public Schedule(List<Event> events) {
        this.events = events;
        checkRep();
    }

    private void checkRep() {
        ...
    }

    /** Event represents an immutable event with a name, a start time, and an end time. */
    interface Event {
        public String name();
        public Date start();
        public Date end();
    }
```

(a) Write the line number(s) that state the preconditions of a creator operation for the Schedule ADT.

(b) Write the line number(s) that state the rep invariant of the Schedule ADT.

(c) Write the line number(s) that state the rep of the Schedule ADT.
Consider the following independent changes to this code. What effect do they have on the spec of the Schedule ADT?

(d) If “sorted by increasing start time” is removed from line 9, the spec for Schedule is:

A. stronger
B. weaker
C. same
D. incomparable

(e) If the call to checkRep() is removed from line 13, the spec for Schedule is:

A. stronger
B. weaker
C. same
D. incomparable

(f) If we add code to the Schedule constructor that satisfies the rep invariant regardless of whether the client satisfied the precondition, the spec for Schedule is:

A. stronger
B. weaker
C. same
D. incomparable
Problem 2 (Testing) (16 points).
Given this specification:

```java
/**
 * Split a string on a delimiting character.
 *
 * @param text a string
 * @param delim a delimiter by which to split the string
 * @param limit an upper bound on the number of elements to return;
 *   if limit < 0, there is no upper bound; limit != 0
 * @return a list of strings [s1, s2, ..., sN] such that:
 *   - text = s1 + delim + s2 + delim + ... + delim + sN
 *   - N <= limit if limit > 0
 *   - none of s1, s2, ..., sN contain delim
 * @throws IllegalArgumentException if limit > 0 and
 *   there are more than limit-1 occurrences of delim in text.
 */
public static List<String> split(String text, char delim, int limit);
```

(a) Start implementing a systematic testing strategy for this function by writing one good partitioning of the input space on input limit alone, i.e., the partition should not mention either text or delim.

(b) Now, write one good partitioning of the input space on the relationship between limit and the occurrences of delim in text. Your partition should mention all three inputs.
Problem 3 (ADTs) (25 points).
Given this code:

```java
/** An immutable class representing a dog show. */
public class DogShow {
    private final Map<String,Integer> dogs;
    public DogShow(List<String> dogsInShow) { ... }
    public DogShow copy() { ... }
    public List<String> getDogs() { ... }
    ...
}
```

(a) Classify each operation according to its type, using the letters C, M, O, P.

DogShow() is a _______

copy() is a _______

getDogs() is a _______

(b) Which of the following are possible abstraction functions for this ADT? (circle all that apply)

A. AF: dogs in a dog show are stored in dogs, with their weights as values
B. AF(dogs) = a dog show where dogs.get(breed) is the number of dogs in the show with the given breed
C. AF(dogs) = a map from a dog’s name to its weight in grams
D. AF(dogs) = a dog show where the nth dog to appear onstage is the dog whose name maps to n in dogs

(c) Which of the following are possible rep invariants for this ADT? (circle all that apply)

A. dogs contains no negative integers as values
B. dogs.size() is 0, 1, >1
C. dogsInShow has no repeats
D. each dog in dogs represents a dog
E. dogs.size() is <= 50

(d) If this ADT had good rep independence, which of the following would be true? (circle all that apply)

A. the implementer could change the precondition of the constructor without telling the client
B. the implementer could change the rep invariant without telling the client
C. the implementer could change the return type List<String> in the signature of getDogs() without telling the client
D. the implementer could make the abstraction function one-to-one without telling the client
E. the implementer could change dogs to a Map<String, String> without telling the client
(e) For each implementation below, is the rep safe or exposed? **Briefly explain.**

```java
/** An immutable class representing a dog show. */
public class DogShow {
    private final List<String> dogs;

    public DogShow(List<String> dogsInShow) {
        dogs = Collections.unmodifiableList(dogsInShow);
    }

    public DogShow copy() {
        return new DogShow(dogs);
    }

    public List<String> getDogs() {
        return dogs;
    }
}
```

Safe? (Y/N)

Reason:

(f) /* An immutable class representing a dog show. */

```java
/** An immutable class representing a dog show. */
public class DogShow {
    private final List<String> dogs;

    public DogShow(List<String> dogsInShow) {
        this.dogs = new ArrayList<>();
        dogs.addAll(dogsInShow);
    }

    public DogShow copy() {
        return new DogShow(dogs);
    }

    public List<String> getDogs() {
        return Collections.unmodifiableList(dogs);
    }
}
```

Safe? (Y/N)

Reason:
Problem 4 (Code Review) (20 points).
Alyssa P. Hacker wrote the following method and asked Ben Bitdiddle to review her code:

```java
/**
 * @param start the start node, must be a key in edges
 * @param target the target node, must be a value in edges
 * @param edges a map where a key-value pair represents a directed edge from
 * the key to the value
 * @return the number of edges that must be traversed to get from start to
 * target, or -1 if no path exists
 */
public static int pathLength(final String start, final String target,
    final Map<String, String> edges) {
    String current = start;
    String next;
    int edgesTraversed = 0;
    while (edges.containsKey(current)) {  
        next = edges.get(current); // traverse edge
        edgesTraversed++;
        if (next.equals(target)) return edgesTraversed;
        edges.remove(current); // avoid entering an infinite cycle
        current = next;
    }
    return -1;
}
```

Ben wrote a series of code review comments. For each comment below,  
1. indicate whether you AGREE or DISAGREE with the comment;  
2. provide one sentence explaining why.

(a) Ben says: “on line 18, removing entries from `edges` will be disallowed by Java at runtime because `edges` is declared as `final`.”

AGREE or DISAGREE? [ ]

Explanation:
(b) Ben says: “on line 18, the implementation doesn’t satisfy the spec.”

AGREE or DISAGREE?

Explanation:

(c) Ben says: “on line 17, next and target are Strings, so they should be compared using == instead of .equals().”

AGREE or DISAGREE?

Explanation:

(d) Ben says: “On line 12, the scope of the variable next should be minimized.”

AGREE or DISAGREE?

Explanation:

(e) Ben says: “The implementation doesn’t fail fast on inputs that violate the precondition.”

AGREE or DISAGREE?

Explanation:
Problem 5 (Multiple Choice) (18 points). (a) Suppose you have the following class:

```java
public class Square {
    private int sideLength; // 1
    // Rep invariant: ... // 2
    // Abstraction function: ... // 3

    /**
     * @return area of this shape // 4
     */
    public int getArea() {
        ...
    }
}
```

Now suppose that you’d like to separate your class out and have it implement an interface, like so:

```java
public interface Shape {
    ...
}
```

```java
public class Square implements Shape {
    ...
}
```

For each of the following commented lines (1, 2, 3, 4) in the old class, denote whether you think it belongs in the new interface or the new implementation class:

(circle one best answer)

A. 1, 2, and 4 belong in the implementation; 3 belongs in the interface.
B. 1, 2, and 3 belong in the implementation; 4 belongs in the interface.
C. 1 and 2 belong in the implementation; 3 and 4 belong in both.
D. 1, 2, 3, and 4 belong in the implementation.
E. 1, 2, 3, and 4 belong in both.

(b) Given the following snippet of code:

```java
int[] numbers = new int[] { 1, 2, 3, 4, 5 };
for (int i = 0; i < numbers.length; i++) {
    numbers[i] /= 2;
}
System.out.println(Arrays.toString(numbers));
```

Which of the following happens when you try to run it?

(circle one best answer)

A. The code has a static type error because it tries to assign double values to int variables.
B. The code will have a dynamic ArrayIndexOutOfBoundsException.
C. The code runs successfully and prints out [0, 1, 1, 2, 2].
D. The code runs successfully and prints out [0.5, 1, 1.5, 2, 2.5].
(e) In a version control system, what would a cycle in the commit history graph indicate? (circle one best answer)
A. One commit undid the changes of a previous commit.
B. Commits were made in multiple clones of the same repository but not yet merged.
C. One commit is the result of merging multiple diverging changes.
D. This occurs when you cannot automatically merge two changes.
E. This is impossible.

(d) Consider the following function:

```java
public static double SquarePyramidVolume(int h, int s) {
    double pyramidDenominator = 3.0;
    return s*s*h/pyramidDenominator;
}
```

What are some constructive code review comments you could make? (circle all that apply)
A. A better method name would be a verb phrase and written in camelCase.
B. The extra variable `pyramidDenominator` is unnecessary, and you could replace the `return` statement in the function with `s*s*h/3`.
C. The variable `pyramidDenominator` should be a `final` constant.
D. The parameters to the function should be renamed to something more descriptive like `height` and `baseWidth`.
E. The variable `pyramidDenominator` should be named `p` to match the style of the method parameters.

(e) Suppose you have the following class:

```java
public class Name {
    private final String name;
    ...
    public boolean equals(Object obj) {
        if (!(obj instanceof Name)) return false;
        Name that = (Name)obj;
        return this.name.toLowerCase().equals(that.name);
    }
}
```

Which of the following expressions return true? (circle all that apply)
A. `new Name("Mary").equals(new Name(""))`
B. `new Name("Mary").equals(new Name("Mary"))`
C. `new Name("Mary").equals(new Name("MARY"))`
D. `new Name("Mary").equals(new Name("mary"))`
E. `new Name("mary").equals("mary")`

(f) Which of the following are properties of an equivalence relation that this `Name.equals()` method violates? Ignore null references. (circle all that apply)
A. antisymmetry
B. invariance
C. reflexivity
D. rep independence
E. symmetry