Solutions to Quiz 1 (March 11, 2015)

Problem 1 (Multiple choice I) (20 points).
Circle all correct answers for the following questions.
(a) Given:

```java
static int[] sort(int[] arr)
    requires: no duplicate elements in arr
    effects: returns a copy of arr with elements sorted in increasing order
```

With the usual 6.005 standards, this specification is (choose all correct answers):

A. deterministic
B. non-deterministic
C. under-determined
D. declarative
E. operational

Solution. A, D.

(b) Given:

```java
static void magic()
    requires: user is in Cambridge, MA
    effects: causes zero or more cannons to mysteriously vanish

define magic()
    requires: user is in MIT Lobby 7
    effects: causes all cannons to suddenly and mysteriously vanish
```

Which of the following are true? (choose all correct answers)

A. The `magic` precondition is weaker than `magic` precondition
B. The `magic` precondition is stronger than `magic` precondition
C. The `magic` postcondition is weaker than `magic` postcondition
D. The `magic` postcondition is stronger than `magic` postcondition
E. The `magic` spec is stronger than `magic` spec

Solution. B, D.

(c) The Java code `private final Set<Person> committee;` demonstrates the concepts of (choose all correct answers):

A. information hiding
B. static typing
C. mutability/immutability  
D. abstract data type  
E. none of these

Solution. A, B, C, D.

(d) The Java code private static Date today = new Date(); demonstrates the concepts of (choose all correct answers):

A. global constant  
B. producer operation  
C. observer operation  
D. representation invariant  
E. none of these

Solution. E.

Problem 2 (Multiple choice II) (9 points).
Circle one correct answer for the following questions.

(a) Which of the following might be a precondition of an operation of mutable StringBuilder but not of immutable String? (choose one best answer)

A. returns the number of characters in this String/StringBuilder  
B. this String/StringBuilder has not been modified since its creation  
C. truncates this String/StringBuilder to at most 16 characters  
D. returns the result of concatenating this String/StringBuilder with itself

Solution. B.

(b) This code... (choose one best answer)
List<Integer> x = new ArrayList<>();
List<Integer> y = x;
x = null;
y.add(3);

A. fails with a compile-time error  
B. fails with a runtime error  
C. results in no variables that point to the ArrayList  
D. results in one variable that points to the ArrayList  
E. results in two variables that point to the ArrayList

Solution. D.

(e) This code... (choose one best answer)
Set<Integer> set1 = new HashSet<>();
set1.add(1);
Set<Integer> set2 = Collections.unmodifiableSet(set1);
set1.add(3);
set2 = set1;

A. fails with a compile-time error
B. fails with a runtime error
C. results in no variables that point to the unmodifiable Set
D. results in one variable that points to the unmodifiable Set
E. results in two variables that point to the unmodifiable Set

Solution. C.

(d) Given the function:

```java
public static void setToUnity(int x) { x = 1; }
```

This code in the same class... (choose one best answer)

```java
final int y = 3;
setToUnity(y);
```

A. fails with a compile-time error
B. fails with a runtime error
C. results in \( y \) equal to 1
D. results in \( y \) equal to 3

Solution. D.

Problem 3 (AF and RI) (25 points).
Louis Reasoner is pretty happy with the Sudoku data type he’s written for 6.005: it represents blank spaces efficiently by using arrays shorter than the size of the puzzle.

For example, for this puzzle:

```
1 2 3 4
- - - -
2 3 _ _
4 1 _ _
```

Louis’ rep is:

```java
cells = { { 1, 2, 3, 4 },
{ },
{ 2, 3 },
{ 4, 1 } }
```

Unfortunately, Alyssa and Ben are not impressed.
Here is the first part of the Sudoku class:
/*
 * Represents a valid, non-zero-size Sudoku puzzle.
 */

public class Sudoku {

  1) public final int[][] cells;

  // Abstraction function:
  2) // the puzzle is of size (cells.length by cells.length)
  3) // cells[i][j] is the number at row i and column j,
  4) // or the cell is blank if j is not a valid index in cells[i]
  // Rep invariant:
  5) // cells.length > 0
  6) // for all sub-arrays row in cells, row.length <= cells.length
  7) // for all values val in all sub-arrays in cells, 1 <= val <= cells.length

  /**
   * Create a new blank puzzle.
   */
  public Sudoku(int size) {
    this.cells = new int[size][size];
    // ... initialize every element of cells with an empty int[] ...
  }

In the spaces below, pinpoint each of the following interrelated problems in the code above.
Write the line number (1-9) of the statement directly responsible for the issue.
Write an explanation of no more than fifteen words that identifies the bug.
You may not use the same line more than once, but you may use the same or similar explanations.

(a) Representation exposure
Line no:  1 Explanation: (1) makes the cells rep field publicly-accessible.

(b) Abstraction function maps legal reps to outside abstract space
Line no:  2 Explanation: (2) cells.length might not be a perfect square - or - (3) row/column/block values may not be unique

(c) Representation invariant allows reps that map to outside abstract space
Line no:  5 Explanation: (5) cells.length might not be a perfect square - or - (7) row/column/block values may not be unique

(d) Representation invariant allows no rep for some abstract values
Line no:  7 Explanation: (7) cannot represent blank cells in middle of non-blank row
(e) Representation invariant not preserved

Solution. (9) does not check that size is nonzero - or - (8) does not impose a precondition on size

Problem 4 (ADTs) (24 points).
Consider the following abstract data type, which has some placeholders shown in all caps.

```java
/**
 * BoxOfBalls is a mutable ADT representing a set of solid spheres
 * (balls) of various sizes and at various locations inside a bounded
 * three-dimensional space (a cube-shaped box with hard walls).
 */
class BoxOfBalls {
    // Abstraction function:
    // AF
    // Rep invariant:
    // RI
    private double w;
    private double[] x;
    private double[] y;
    private double[] z;
    private double[] r;
    /**
     * requires: PRE
     * effects: POST
     */
    public void move(int i, double dx, double dy, double dz) { ... }
}
```

For the following list of statements, write AF, RI, PRE, or POST next to each statement below, according to whether it belongs in the abstraction function, rep invariant, precondition, or postcondition comment, respectively. Leave a statement blank if it cannot belong in any of them, or is redundant with or contradicts information in other statements or in the code above.

- ______ the width, height, and depth of the box is w
- ______ the position of ball i is (x[i], y[i], z[i]) relative to the origin corner of the box
- ______ the radius of ball i is r[i]
- ______ w > 0
- ______ r[i] <= x[i], y[i], z[i] <= w-r[i] for all 0 <= i < r.length
- ______ |dx|, |dy|, |dz| <= w
- ______ x.length == y.length == z.length == r.length
- ______ x.length == w
- ______ the box contains N balls where N = x.length
for every pair of balls, the distance between their centers is at least
the sum of their radii

\[ \theta \leq i < \text{number of balls in the box} \]

changes ball \( i \)'s position by moving it along a vector \((dx, dy, dz)\),
ending it at the first collision with either another ball or a wall of the box

Solution. AF, AF, AF, RI, RI, \(\_\_\_\_\_\_\_\), RI, \(\_\_\_\_\_\_\_\), AF, RI, PRE, POST

Problem 5 (Recursion) (22 points).

A palindrome is a string of characters that spells out the same regardless of whether it is read forwards or backwards. Some examples of palindrome words are “noon”, “stats”, or “rotator”. Tired of rubbing his eyes and checking words manually, Ben Bitdiddle decides to write a PalindromeChecker to check for palindromes.

Ben’s strategy for the problem of determining whether a given word is a palindrome is to use recursion: check if the first and last characters of word are the same, then check if the sub-string for word between them is a palindrome.

Ben writes the first version of his program, shown below.

```java
public class PalindromeChecker {

    /**
     * Check if a given word is a palindrome. Palindromes are case-insensitive.
     * @param word the word containing only alphanumerics characters and no spaces
     * @return true if word is a palindrome, false otherwise
     */
    public static boolean isPalindrome(String word) {
        char firstChar = word.charAt(0);
        char lastChar = word.charAt(word.length()-1);
        if (firstChar == lastChar) {
            return isPalindrome(word.substring(1, word.length()-1));
        }
        return false;
    }

    public static void main(String[] args) {
        System.out.println(PalindromeChecker.isPalindrome("noon")); // true
        System.out.println(PalindromeChecker.isPalindrome("toon")); // false
    }
}
```

(a) Which of the following is true if Ben runs the program? (circle one)

A. The program will not compile because of a static error
B. The program will throw a StackOverflowError
C. The program will throw a StringIndexOutOfBoundsException
D. The program will never terminate
Solution. C.

Note: this question is based on the example at http://beginnersbook.com/2014/01/java-program-to-check-palindrome-string-using-recursion/

Alyssa takes a look at Ben’s code and notices that his recursive function is missing a base case! She suggests checking whether word has length zero.

Ben takes her advice and adds the check to isPalindrome:

```java
public static boolean isPalindrome(String word) {
    if (word.length() == 0) {
        return true;
    }
    char firstChar = word.charAt(0);
    char lastChar = word.charAt(word.length()-1);
    if (firstChar == lastChar) {
        return isPalindrome(word.substring(1, word.length()-1));
    }
    return false;
}
```

Now the program runs to completion and prints the correct answers. Hooray! Ben decides to celebrate by converting the tests in his main method into proper unit tests. His test suite is show below:

```java
public class PalindromeCheckerTest {
    @Test public void testPalindrome() {
        assertTrue(PalindromeChecker.isPalindrome("noon"));
    }
    @Test public void testNotPalindrome() {
        assertFalse(PalindromeChecker.isPalindrome("toon"));
    }
}
```

(b) Unfortunately, Ben’s code still has bugs. In the spaces below, write two tests that will reveal all the remaining bugs. You should keep the standards of 6.005 in mind when naming and writing your tests.

```java
@Test public void testBug1() {
    // Test code
}
```

```java
@Test public void testBug2() {
    // Test code
}
```
Solutions to Quiz 1 (March 11, 2015)

Solution. The implementation does not handle words of odd length, and does not satisfy the case-insensitivity requirement.

```java
@Test public void testOddLength() {
    assertTrue(PalindromeChecker.isPalindrome("non"));
}
@Test public void testMixedCase() {
    assertTrue(PalindromeChecker.isPalindrome("ToOt"));
}
```

Below is Ben’s current code, with Alyssa’s base case:

```java
public class PalindromeChecker {

/**
 * Check if a given word is a palindrome. Palindromes are case-insensitive.
 * @param word the word containing only alphanumerical characters and no spaces
 * @return true if word is a palindrome, false otherwise
 */

public static boolean isPalindrome(String word) {
    if (word.length() == 0) {
        return true;
    }
    char firstChar = word.charAt(0);
    char lastChar = word.charAt(word.length()-1);
    if (firstChar == lastChar) {
        return isPalindrome(word.substring(1, word.length()-1));
    }
    return false;
}
}
```

(c) Above, circle the line numbers of statements responsible for the bugs.

Solution. 2, 5, and 6.

(d) For each buggy statement identified in the previous question, propose a one-line fix. Give only one fix for each buggy statement. You do not have to use all the boxes.

<table>
<thead>
<tr>
<th>Line no:</th>
<th>Fix:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Solution. Handle odd-length words:

2) \[
\text{if (word.length()} \leq 1) \{
\]

Handle mixed case:

5) \[
\text{char firstChar} = \text{word.toLowerCase().charAt(0)};
\]

6) \[
\text{char lastChar} = \text{word.toLowerCase().charAt(word.length()-1)};
\]