Quiz 1 (March 14, 2016)

Your name:_____

Your Athena username:_____

You have 50 minutes to complete this quiz. It contains 12 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!

DO NOT WRITE BELOW THIS LINE

Problem	Points	Grade	Grader
1: Multiple Choice	20		
2: Specifications	20		
3: AFs & RIs	20		
4: Testing	20		
5: Rep Exposure	20		
Total	100		

Problem 1 (Multiple Choice) (**20 points**).

(**a**) Which of the following must be true of an underdetermined function specification? (choose all that apply)

A. An underdetermined spec means the implementation is unwritten.

B. An underdetermined spec means the implementation is nondeterministic.

C. An underdetermined spec allows multiple valid outputs for some input.

D. An underdetermined spec allows multiple valid inputs that give some output.

(**b**) After the following code is executed, what is the value of array `arr`? (choose one answer)

```
final String[] arr = new String[2];
String s = "6.005";
arr[0] = s;
s = "is";
arr[1] = s;
String t = arr[0];
t = "fun";
```

A. ["is", "is"]

B. ["6.005", "is"]

C. ["fun", "is"]

D. *none of the above*

(**c**) The line of Java code `String t = arr[0];` involves... (choose all that apply)

A. assignment

B. equivalence relation

C. mutation

D. static typing

(d) Alyssa P. Hacker is designing an immutable type to represent users in her computer system. The `User`'s login name is stored as:

```
private final String kerberos;
```

She defines two `User` objects as equal if their login names are the same, **ignoring case**:

```
@Override public boolean equals(Object other) {
  if (!(other instanceof User)) { return false; }
  User that = (User)other;
  return this.kerberos.equalsIgnoreCase(that.kerberos);
}

@Override public int hashCode() { /* TODO */ }
```

Which of the following implementations of `hashCode()` would be valid, satisfying the `Object` contract? (choose all that apply)

  A. `return 31;`

  B. `return this.kerberos.hashCode();`

  C. `return this.kerberos.toLowerCase().hashCode();`

  D. `return this.kerberos.toUpperCase().hashCode();`

(e) If the code in answer choice (A) above appeared in your 6.005 code review assignment, which of the following comments would be appropriate criticisms? (choose all that apply)

  A. The code isn't DRY.

  B. The code uses magic numbers.

  C. The code exposes `User`'s representation.

  D. The code is unnecessary, we don't need to override `hashCode` if we only return a constant.

Problem 2 (Specifications) (**20 points**).

A **tetromino** is a shape made out of four adjacent squares. These shapes are most famous from the game *Tetris* where the player must rotate and translate falling tetrominoes in order to fit them together. There are seven possible tetrominoes that lie on the 2D plane, and each is identified by a letter it looks like:
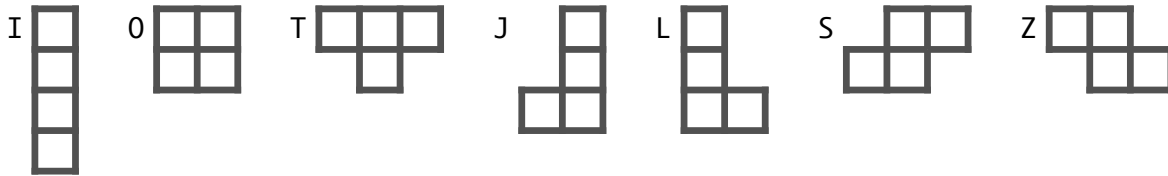
Define a **tetromino shape letter** as one of the seven letters I O T J L S Z, either upper- or lowercase. Tetrominoes may be rotated by 0, 90, 180, or 270 degrees.

A diagram of all seven tetrominoes in all four orientations is on the last page of this quiz.

Let's define an abstract data type to represent tetrominoes:

```java
public class Tetromino {

    // determine if a character is a valid tetromino shape letter
    public static boolean isValidShape(char shape) { ... }

    // make a new tetromino
    public Tetromino(char shapeLetter, int rotationDegrees) { ... }

    // rotate this tetromino
    public void rotateClockwise() { ... }

    // get the shape of this tetromino
    public char shape() { ... }
}
```

(a) Fill in this table with information about the operations of `Tetromino`.

The answers for `isValidShape` are already given.

operation	type signature	classify the type of ADT operation	Java implementation strategy
isValidShape	char → boolean	not applicable	static method
Tetromino			
rotateClockwise			
shape			

(b) Consider these different specifications for the `isValidShape` function:

    public static boolean isValidShape(**char** shape)

Spec A
```
/**
 * @param shape any character
 * @return true iff shape is a lowercase tetromino shape letter
 */
```

Spec B
```
/**
 * @param shape an English alphabetic character
 * @return true iff shape is a lowercase tetromino shape letter
 */
```

Spec C
```
/**
 * @param shape a lowercase tetromino shape letter
 * @return true
 */
```

Compare these specifications. For each pair below, **circle one correct option** and write **write a brief explanation** to complete the sentence.

A vs. B:

Spec A is | equivalent to | spec B because
| weaker than |
| stronger than |
| incomparable to |

B vs. C:

Spec B is | equivalent to | spec C because
| weaker than |
| stronger than |
| incomparable to |

Problem 3 (AFs & RIs) (**20 points**).
Here is an implementation of the `Tetromino` ADT.

```java
/** A rotatable tetromino. */
public class Tetromino {

    // ... static method isValidShape ...

    private final char shape;
    private int rotation;

    // Abstraction function
    //    TODO
    // Representation invariant
    //    TODO
    // Safety from rep exposure
    //    TODO

    /**
     * Make a new Tetromino with the given shape and rotation.
     * @param shapeLetter uppercase tetromino shape letter
     * @param rotationDegrees clockwise rotation in degrees,
     *                        must be 0, 90, 180, or 270
     */
    public Tetromino(char shapeLetter, int rotationDegrees) {
        this.shape = shapeLetter;
        this.rotation = rotationDegrees / 90;
    }

    /**
     * TODO
     */
    public void rotateClockwise() {
        rotation = (rotation + 1) % 4;
    }

    /**
     * @return shape of this tetromino: 'I' 'O' 'T' 'J' 'L' 'S' or 'Z'
     */
    public char shape() {
        return shape;
    }

    @Override
    public String toString() {
        return shape + "-shape@" + rotation * 90 + "deg";
    }
}
```

 (a) For each of the statements below, say whether it should be included in the internal documentation of `Tetromino` by writing:

AF if the statement belongs in the abstraction function

RI   ...   the rep invariant

EXP   ...   the argument that type has no rep exposure

NONE if it should not be included in any of those

You should include in the AF, RI, or EXP **all good statements** that are compatible with the code and specs on the previous page.

Do not include statements that are not compatible with the code and specs.

	`shape is private and an immutable value`
	`rotation is private and an immutable value`
	`this Tetromino is never returned to clients`
	`shape is an uppercase tetromino shape letter`
	`shapeLetter is an uppercase tetromino shape letter`
	`the tetromino has the shape given by shape`
	`rotation = 0`
	`the tetromino is rotated clockwise by rotation`
	`the tetromino is rotated clockwise rotation times 90 degrees`
	`0 <= rotation < 4`
	`0 <= rotation < 360`
	`rotation is one of { 0, 90, 180, 270 }`

Consider these different specifications for the `rotateClockwise` method:

```
public void rotateClockwise()
```

For each possible specification below, **write a one-sentence code review comment that identifies the most serious problem** with the spec.

(b) `/**`
     `* Update this tetromino's rotation number to add 1, mod 4.`
     `*/`

Comment:

(c) `/**`
     `* @return this tetromino rotated by 90 degrees clockwise`
     `*/`

Comment:

(d) `/**`
     `* Rotate this tetromino (does not otherwise modify the tetromino).`
     `*/`

Comment:

Problem 4 (Testing) (**20 points**).

Let's consider the problem of packing tetrominoes into a rectangle by rotating and translating them in the 2D plane. For example, here are tetrominoes I, L, & Z packed into a 5×4 rectangle, and a 7×2 rectangle:
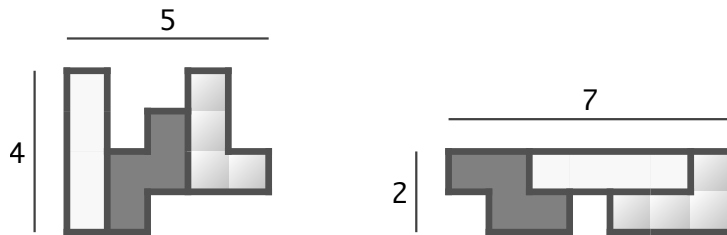
For those three tetrominoes, the minimum area they can be packed into is $7 \times 2 = 14$ squares.

(a)
```
/**
 * @param shapes string of tetromino shape letters
 * @return a 2-element list representing a minimum-area rectangle into which
 *         the tetrominoes given by shapes can be packed
 *         (for example, pack("ILZ") might return the list [ 2, 7 ])
 */
public static List<Integer> pack(String shapes) { ... }
```

Start writing a black box testing strategy for `pack(..)` by giving **one good partitioning** for input `shapes`:

For each of the test cases below, in the first box **write YES or NO** in the first box to say whether the test valid or not. If the test is not valid, **write a one-sentence reason why not**. We'll use Python's syntax to represent lists for brevity.

(b) `shapes = "X"`
    `rectangle = [ 0, 0 ]`

Valid? ☐   Reason if invalid: ☐

(c) `shapes = "I"`
    `rectangle = [ 1, 4 ]`

Valid? ☐   Reason if invalid: ☐

(d) `shapes = "LO"`
    `rectangle = [ 3, 3 ]`

Valid? ☐   Reason if invalid: ☐

Problem 5 (Rep Exposure) (**20 points**).

Let's define a mutable abstract data type `TetrominoGrid` to represent tetrominoes arranged on a fixed-size grid, where every tetromino fits on the grid and none of the tetrominoes overlap. For example:
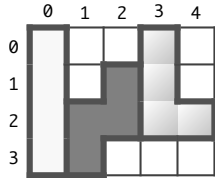
On the next page is an implementation of `TetrominoGrid`. In its rep, `TetrominoGrid` stores the location of each tetromino using a map from integers to tetrominoes. The integer keys are square numbers:

Each square in the grid is numbered starting from 0 in the upper-left corner.

On a grid of width *width*, square at row *row* and column *col* is numbered *row* × *width* + *col*.

For example, here's how our type would represent the example above:
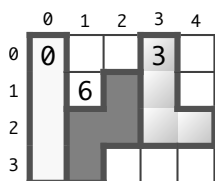

```
width = 5, height = 4
tetrominoLocations = {
    0: (Tetromino: I-shape rotated 0 degrees),
    3: (Tetromino: L-shape rotated 0 degrees),
    6: (Tetromino: Z-shape rotated 90 degrees)
}
```

The `TetrominoGrid` ADT is implemented using the same **mutable** `Tetromino` ADT from previous questions.

(a) Identify all instances of rep exposure in `TetrominoGrid` on the next page. For each one, write:

1. the line number most directly responsible for the problem,
2. at most one-sentence description of the rep exposure, and
3. at most one-sentence description of how to fix the problem, or a single corrected line of code.

There may be more boxes than you need.

Line #: [ ]   Explanation:

Line #: [ ]   Explanation:

Line #: [ ]   Explanation:

Line #: [ ]   Explanation:

Line #: [ ]   Explanation:

You may detach this page from your quiz, but you must write your name above and turn in all pages.

```
  /**
   * Mutable type representing a fixed-size grid with a valid arrangement of
   * tetrominoes: every tetromino fits on the grid without overlapping.
   */
  public class TetrominoGrid {

1)    private final int width;
2)    private final int height;
3)    public final Map<Integer, Tetromino> tetrominoLocations;

      // create a new grid and try to add some tetrominoes
      public TetrominoGrid(int width, int height, List<Tetromino> initial) {
4)        this.width = width;
5)        this.height = height;
6)        this.tetrominoLocations = new HashMap<>();
          for (Tetromino tetromino : initial) {
7)            this.add(tetromino.shape());
          }
      }

      // get the tetromino whose upper-left corner is the given square
      public Tetromino getTetrimino(int row, int col) {
8)        return tetrominoLocations.get(row * width + col);
      }

      // return a list of all tetriminos currenty on the grid
      public List<Tetromino> getTetriminosOnBoard() {
9)        List<Tetromino> tetrominoes = new ArrayList<>();
          for (Tetromino tetromino : tetrominoLocations.values()) {
10)           tetrominoes.add(tetromino);
          }
11)       return Collections.unmodifiableList(tetrominoes);
      }

      // try to add a tetromino to the grid, if it can fit without overlap
      public boolean add(char shape) {
12)       boolean canBePlaced = false; // can we fit the tetromino anywhere?
13)       int topLeft = -1;
14)       int rotation = 0;
          // ... code to check whether the new tetromino fits and doesn't overlap ...
          // ... updates the values of the local variables accordingly ...
          if (canBePlaced) {
15)           tetrominoLocations.put(topLeft, new Tetromino(shape, rotation));
          }
16)       return canBePlaced;
      }

      // ... other operations ...
  }
```
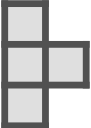
You may detach this page from your quiz, but you must write your name above and turn in all pages.

6.005 Software Construction

Spring 2016