Quiz 1 (October 24, 2018)

Your name:__________________________________________

Your Kerberos 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 Kerberos username 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: AFs &amp; RIs</td>
<td>18</td>
</tr>
<tr>
<td>2: ADTs</td>
<td>32</td>
</tr>
<tr>
<td>3: Specs I</td>
<td>20</td>
</tr>
<tr>
<td>4: Specs II</td>
<td>15</td>
</tr>
<tr>
<td>5: Testing</td>
<td>15</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>100</strong></td>
</tr>
</tbody>
</table>
For this quiz, *Mini Rush Hour* is a sliding block puzzle played on a 5×5 grid of cells numbered 1 to 25:

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
<tr>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
</tr>
<tr>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
</tr>
</tbody>
</table>

← The board has an *exit* to the right of cell 15.

*Cars* are 1-cell × 2-cells pieces placed horizontally or vertically on the grid. Every Mini Rush Hour puzzle starts with the *VIP car* on cells 11 and 12. The goal is to move the VIP car to the right so it reaches cells 14 and 15, at which point it can exit the board:

![Diagrams of Mini Rush Hour puzzle](image)

Blocking the VIP car’s way are other cars, for example:

![Diagrams of Mini Rush Hour puzzle](image)

While they are labeled in this diagram, cars other than the VIP car are not differentiated except by their placement on the board, they have no distinguishing characteristics. All cars in the game move only along their axis of orientation: horizontal cars move left and right, vertical cars move up and down. Cars never overlap. They can only move through and stop on empty cells.

The player wins by making a series of moves that allow the VIP car to reach the exit. The following page shows a solution to this example puzzle, plus another example.

The problems in this quiz refer to the code for two different Mini Rush Hour ADTs, *RushHourPuzzle* on page 9 and *RushHourGame* on page 10, which you may detach.

Notes which are not relevant to this quiz:

- *Rush Hour*® is a trademark of ThinkFun, Inc. The full game uses varying-size cars on a larger board.
- On arbitrarily large grids, *Rush Hour* and *Size-2 Rush Hour* are PSPACE-complete (Flake & Baum, 2002; Tromp & Cilibrasi, 2005).
Solving the example puzzle in 3 moves:

1. 

2. 

3. 

Solving another example in 6 moves:

1. 

2. 

3. 

4. 

5. 

6.
Problem 1 (AFs & RI) (18 points).
Immutable `RushHourPuzzle` represents the starting layout of a Mini Rush Hour puzzle.

Ben proposes the following rep for this type:
```java
private final List<Integer> horizCars;
private final List<Integer> vertCars;
```

He will use empty lists to represent the minimal starting puzzle, with just the VIP car in cells 11 and 12:

```
List
horizCars
List
vertCars
```

And here is how he intends to represent the example puzzle from page 2:

```
List
horizCars
List
vertCars
```

Help construct the abstraction function and rep invariant for this implementation.

(a) Write a concise but complete abstraction function for this rep, consistent with Ben’s examples and with your (partial) rep invariant below.

(b) Assumed in 6.031:

(c) One statement required for your AF that involves only `horizCars`:

(d) One statement required for your AF that relates `horizCars` and `vertCars`:
Problem 2 (ADTs) (32 points).
Ben is trying to implement equality for immutable RushHourPuzzle using a sameValue(..) helper:

```java
private boolean sameValue(RushHourPuzzle that) {
    return horizCars.equals(that.horizCars) && vertCars.equals(that.vertCars);
}
```

(a) Unfortunately, this implementation requires a stronger rep invariant in order to work. Use Python list notation to give a plausible example of two different reps, containing as few cars as possible, where Ben’s implementation would return the wrong result:

(b) Ben does not want to strengthen the RI, and plans to fix sameValue instead. While he does that, implement a valid hashCode that will also work without strengthening the RI. Do not return a constant.

```java
@Override public int hashCode() {
    // Implement hashCode logic here
}
```

(c) For each of these operations of RushHourPuzzle, complete its type signature, and write what kind of operation it is in our taxonomy of ADT operations.

<table>
<thead>
<tr>
<th>inputs</th>
<th>outputs</th>
<th>kind</th>
</tr>
</thead>
<tbody>
<tr>
<td>withHorizontal :</td>
<td>→</td>
<td>is a</td>
</tr>
<tr>
<td>difficulty :</td>
<td>→</td>
<td>is a</td>
</tr>
</tbody>
</table>

Alyssa proposes a different rep for RushHourPuzzle, using the Direction type at the top of page 10:

```java
private final Map<Integer, Direction> cars;
```

(d) What do you know about this rep that provides safety from rep exposure?

(e) What one additional assumption about this rep completes the SRE argument, without any reference to methods or their signatures? (Partial credit: complete the argument with an assumption about the methods.)
Problem 3 (Specs I) (20 points).
Alyssa realizes that the spec for `withHorizontal(..)` does not account for attempts to add an overlapping car (`withVertical` has a similar problem, and both may have other problems).

Given the following changes, where the rest of the spec is unchanged in each:

- Pick one best solution to the specific problem of overlapping cars in `withHorizontal`. Circle “YES” and explain in one sentence what properties make it the best solution.
- For every other option, circle “NO” and explain in one sentence why it is not a good solution.

(a) * ...
* @param left requires 1 <= left <= 25, and adding left to horizCars does not
*    violate the RI above
* ...

Best fix? YES NO

(b) * Make a Mini Rush Hour puzzle identical to this but with a horizontal car as
*    close as possible to cell left.
* ...
* @return this puzzle with an additional horizontal car in left and left+1 if
*    possible, or in the nearest empty 2-cell-wide space otherwise

Best fix? YES NO

(c) * ...
* @return this puzzle with an additional horizontal car in left and left+1
* @throws OverlapException if another car occupies cells left or left+1
*/

Best fix? YES NO

(d) * ...
* @return true if and only if left and left+1 were empty and the car was added
*/
public boolean withHorizontal(int left)
Problem 4 (Specs II) (15 points).
Mutable RushHourGame (on page 10) allows the client to play a Mini Rush Hour game, starting from an initial RushHourPuzzle and making a series of moves with the move( ) method.

For each of the following changes to the spec of move, where the rest of the spec is unchanged in each:

- Circle “STRONGER,” “WEAKER,” or “INCOMPARABLE” to indicate whether the new spec is stronger than, weaker than, or not comparable to the original spec of move.
- Explain why in one sentence that mentions the pre- and postconditions and uses them to draw a conclusion.

(a) /**
 * Move the car that occupies cellNum one cell in the given direction, if possible.
 * ...

(b) * ...
 * @param cellNum indicates car to move, requires 1 <= cellNum <= 25
 * ...
 * @throws IllegalArgumentException if no car occupies cellNum
 */

(c) * ...
 * @param direction direction to move, must be LEFT/RIGHT or UP/DOWN when the car
 * is horizontal or vertical, respectively
 * ...

Problem 5 (Testing) (15 points).
Using the specs for mutable RushHourGame on page 10, start devising a testing strategy for move().
Each of your partitions below should divide the space into 2 parts.

(a) Write one correct and useful 2-part partitioning of the input space on input this alone:

(b) Write one correct and useful 2-part partitioning on only this and cellNum:

(c) Write one correct and useful 2-part partitioning on all of the inputs together:
Quiz 1 (October 24, 2018)

You may detach this page. Write your username at the top, and hand in all pages when you leave.

```java
/** Immutable starting layout of a Mini Rush Hour puzzle. */
public class RushHourPuzzle {

    // ... rep ...

    /**
     * Make a Mini Rush Hour puzzle with the VIP car in the starting location
     * and no other cars on the board.
     */
    public RushHourPuzzle() { ... }

    /**
     * Make a Mini Rush Hour puzzle identical to this but with a horizontal car
     * whose left half is in cell left.
     * @param left requires 1 <= left <= 25
     * @return this puzzle with an additional horizontal car in left and left+1
     */
    public RushHourPuzzle withHorizontal(int left) { ... }

    /**
     * Make a Mini Rush Hour puzzle identical to this but with a vertical car
     * whose top half is in cell top.
     * @param top requires 1 <= top <= 25
     * @return this puzzle with an additional vertical car in top and top+5
     */
    public RushHourPuzzle withVertical(int top) { ... }

    /**
     * @return the difficulty of this puzzle, measured as the minimum number of
     *        moves required to solve it; or -1 if this puzzle is unsolvable
     */
    public int difficulty() { ... }

    /**
     * @return a string representation of this puzzle that describes all the
     *        cars on the board
     */
    @Override public String toString() { ... }

    // ... other operations ...
}
```
public enum Direction { LEFT, RIGHT, UP, DOWN }

/**
* Make a new Mini Rush Hour game starting from the given puzzle layout.
* @param starting puzzle to play
*/
public RushHourGame(RushHourPuzzle starting) {... }

/**
* Move the car that occupies cellNum as many cells as it can move in the
given direction.
* @param cellNum indicates car to move, requires 1 <= cellNum <= 25, and a
car must occupy that cell
* @param direction direction to move
* @return true if-and-only-if the VIP car is in the game-winning position
*/
public boolean move(int cellNum, Direction direction) {... }

For reference, the grid and the example puzzle: