Question 3: JumpingCritter (GridWorld)

Part (a)  getEmptyLocations  5 points

Intent: Create and return ArrayList<Location> of all empty locations in grid

+½ Declares and constructs empty ArrayList<Location>
+½ Accesses all locations in grid (no bounds errors)
+2 Identifies empty location in grid in context of loop
   +1 Creates new location in grid
   +1 Determines if created location is empty
+1 Includes all and only identified empty locations in constructed arraylist exactly once
+1 Returns the constructed arraylist (code must have examined grid)

Part (b)  Class: JumpingCritter  4 points

Intent: Define extension to Critter class that jumps to randomly selected empty location in its grid

+½ class JumpingCritter extends Critter

+1½ Override getMoveLocations
   +½ public ArrayList<Location> getMoveLocations()
   +½ GridWorldUtilities.getEmptyLocations(getGrid())
   +½ Returns arraylist containing empty locations

+1 Handles null location case correctly in selectMoveLocation

+1 Handles random location case correctly (must override getMoveLocations)

Question-Specific Penalties

-1 (s) Causes inappropriate state change in world (Grid, Actor, ...)

-1 (t) Overrides act
Complete method `getEmptyLocations` below.

```java
/**
 * Gets all the locations in grid that do not contain objects.
 * @param grid a reference to a BoundedGrid object
 * @return an array list (possibly empty) of empty locations in grid.
 * The size of the returned list is 0 if there are no empty locations in grid.
 * Each empty location in grid should appear exactly once in the returned list.
 */
public static ArrayList<Location> getEmptyLocations(Grid<Actor> grid)
{
    ArrayList<Location> emptyLocs = new ArrayList<Location>();
    grid = getGrid();
    for (int a = 0; a < grid.getNumRows(); a++)
    {
        for (int b = 0; b < grid.getNumCols(); b++)
        {
            Location c = new Location(a, b);
            if (grid.get(c) == null)
            {
                emptyLocs.add(c);
            }
        }
    }
    return emptyLocs;
}
```

Part (b) begins on page 18.
Assume that the GridWorldUtilities getEmptyLocations method works as specified, regardless of what you wrote in part (a). Solutions that reimplement the functionality of this method will not receive full credit.

Write the complete JumpingCritter class. Do NOT override the act method. Remember that your design must not violate the postconditions of the methods of the Critter class.

```java
public class JumpingCritter extends Critter {

    public ArrayList<Location> getMoveLocations() {
        Grid<Actor> grid = getGrid();
        ArrayList<Location> locs = GridWorldUtilities.getEmptyLocations(grid);
        return locs;
    }

    public void makeMove(Location loc) {
        if (loc.equals(getLocation()) || loc == null) {
            removeSelfFromGrid();
        } else {
            moveTo(loc);
        }
    }
}
```
Complete method `getEmptyLocations` below.

```java
/**
 * Gets all the locations in `grid` that do not contain objects.
 * @param grid a reference to a `BoundedGrid` object
 * @return an array list (possibly empty) of empty locations in `grid`.
 * The size of the returned list is 0 if there are no empty locations in `grid`.
 * Each empty location in `grid` should appear exactly once in the returned list.
 */

public static ArrayList<Location> getEmptyLocations(Grid<Actor> grid) {
    ArrayList<Location> hold = new ArrayList<Location>();
    for (int r = 0; r < grid.getNumRows(); r++) {
        for (int c = 0; c < grid.getNumCols(); c++) {
            if (grid.get(new Location(r, c)) instanceof Wall) {
                hold.add(new Location(r, c));
            }
        }
    }
    return hold;
}
```

Part (b) begins on page 18.
Assume that the GridWorldUtilities getEmptyLocations method works as specified, regardless of what you wrote in part (a). Solutions that reimplement the functionality of this method will not receive full credit.

Write the complete JumpingCritter class. Do NOT override the act method. Remember that your design must not violate the postconditions of the methods of the Critter class.

class JumpingCritter extends Critter

public JumpingCritter()
    super();

public ArrayList<Location> getMoveLocations()
{
    return GridWorldUtilities.getEmptyLocations(getGrid());
}

public selectMoveLocation(ArrayList<Location> loc)
{
    if (loc.size() == 0 || loc.size() == null)
    remove(self() from loc);;
else
    super.selectMoveLocation(loc);
}
Complete method `getEmptyLocations` below.

```java
/**
 * Gets all the locations in grid that do not contain objects.
 * @param grid a reference to a BoundedGrid object
 * @return an array list (possibly empty) of empty locations in grid.
 * The size of the returned list is 0 if there are no empty locations in grid.
 * Each empty location in grid should appear exactly once in the returned list.
 */
public static ArrayList<Location> getEmptyLocations(Grid<Actor> grid) {
    for (int row = 0; row < grid.length; row++) {
        for (int col = 0; col < grid.length; col++) {
            ArrayList<Location> emptyLocations = new ArrayList<>();
            if (grid[row][col].getEmptyAdjacentLocations().isEmpty()) {
                emptyLocations.addAll(grid[row][col].getEmptyAdjacentLocations());
                return emptyLocations;
            }
        }
    }
    return new ArrayList<>();
}
```

Part (b) begins on page 18.
Assume that the `GridWorldUtilities` `getEmptyLocations` method works as specified, regardless of what you wrote in part (a). Solutions that reimplement the functionality of this method will not receive full credit.

Write the complete `JumpingCritter` class. Do NOT override the `act` method. Remember that your design must not violate the postconditions of the methods of the `Critter` class.

```java
public class JumpingCritter extends Critter {

    public ArrayList<Location> getMoveLocations() {
        Location loc = getGrid().getEmptyLocation(getLocation());
        ...
    }

    public void makeMove(Location loc) {
        if (loc == null) {
            removeIfFromGrid();
        } else {
            moveTo(loc);
        }
    }

    ...
}
```
Question 3

Overview

This question involved reasoning in the context of the GridWorld case study. Part (a) required writing a static method in a utilities class, traversing a two-dimensional data structure included in a Grid, working with a list (instantiating an ArrayList of Location objects, adding elements and testing for empty), and returning values from a method. Part (b) required the writing of a Critter subclass, understanding inheritance and polymorphism, overriding selected methods of the Critter class, and paying attention to specific post-conditions.

Students commonly approached part (a) in either of two ways.

1. Start with an empty ArrayList and add empty locations.
2. Start with an ArrayList of all locations and remove occupied locations.

In part (b), students needed a good understanding of GridWorld to determine which two methods (getMoveLocations and selectMoveLocation) to override. Overriding makeMove instead of selectMoveLocation violates makeMove's post-condition that getLocation() == loc in the case loc is null.

Sample: 3A
Score: 8

In part (a), the ArrayList is successfully declared and constructed as an ArrayList of Location objects. All the locations in grid are accessed using two nested loops, correctly using grid.getNumRows() and grid.getNumCols() as the loop bounds. A Location object within the grid is correctly created using the new operator. The empty location test is done correctly by accessing the object at that location and comparing the object (using ==) to null. If the test succeeds, the location is then correctly added to the ArrayList. The constructed ArrayList is returned correctly after the two loops have examined the entire grid. Part (a) earned 5 points.

In part (b), the class header class JumpingCritter extends Critter is correct. The method header for getMoveLocations is correct. In getMoveLocations, the call to method getEmptyLocations is correctly qualified with the class name GridWorldUtilities and uses the correct grid object as the argument. The resulting ArrayList is returned. The makeMove method is overridden. In the case where loc is null, the result of calling removeSelfFromGrid() violates makeMove's post-condition that getLocation() == loc, so the point for the null case is lost. The random case is correctly handled through the inherited selectMoveLocation method to identify a location and the call to moveTo in the overridden makeMove. Part (b) earned 3 points.

Sample: 3B
Score: 6

In part (a), the ArrayList is successfully declared and constructed as an ArrayList of Location objects. All the locations in grid are accessed using two nested loops, correctly using grid.getNumRows() and grid.getNumCols() as the loop bounds. A Location object within the grid is correctly created using the new operator. The object at that location is correctly accessed [grid.get(location)]; however, the empty location test is incorrect because the comparison test "instance of null" is incorrect, so the point for the empty test is lost.
The location is correctly added to the ArrayList. The constructed ArrayList is returned correctly after the two loops have examined the entire grid. Part (a) earned 4 points.

In part (b), the class header class JumpingCritter extends Critter is correct. The method header for getMoveLocations is correct. In getMoveLocations, the call to method getEmptyLocations is correctly qualified with the class name GridWorldUtilities and uses the correct grid object as the argument. The resulting ArrayList is returned. The selectMoveLocation method is overridden. In the case where loc is null, the result of calling removeSelfFromGrid() violates selectMoveLocation's post-condition that "the state of all actors is unchanged." The post-condition that "The returned location is an element of locs, the critter's current location, or null" is also violated, so the point for the null case is lost. Although there is a correct call to super.selectMoveLocation, the resulting location is not returned, so the random case is not handled correctly, thereby losing 1 point for the random case. Part (b) earned 2 points.

Sample: 3C
Score: 2

In part (a), an ArrayList of Location objects is not declared and constructed (-½ point). The loop boundaries are incorrect and the increments (row+2 and col+2) are also incorrect so the solution loses the access point (-½ point). A new Location is not created (-1 point). The test for an empty location is incorrect (-1 point). Empty locations are not accumulated in the ArrayList (the attempt to declare the ArrayList is done inside the loop) so the solution does not receive credit for including identified empty locations (-1 point). The premature return from inside the loop loses the return point (-1 point). Part (a) earned 0 points.

In part (b), the class header class JumpingCritter extends Critter is correct. The method header for getMoveLocations is also correct. However, in getMoveLocations, the method getEmptyLocations is called incorrectly, because it is not qualified with the class name GridWorldUtilities and the argument in the method call is not the current grid (-½ point). The resulting ArrayList is not returned (-½ point). The makeMove method is overridden. In the case where loc is null, removeSelfFromGrid() violates makeMove's post-condition that getLocation() == loc, so the point for the null case is lost. The random case is correctly handled in the inherited selectMoveLocation method and the call to moveTo in makeMove. Part (b) earned 2 points.