4. This question involves reasoning about the GridWorld case study. Reference materials are provided in the Appendix.

   In this question, you will write two unrelated methods of the `GridChecker` class that will process a `BoundedGrid<Actor>` object. Recall that the `BoundedGrid` class implements the `Grid` interface. Also note that the methods in the `Grid` interface that return an array list will return an empty array list when no objects meet the return criteria.

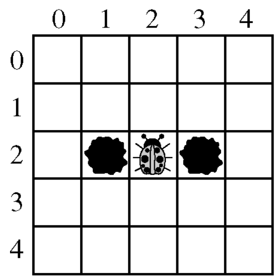
   The declaration of the `GridChecker` class is shown below.

```
public class GridChecker
{
  /**  The grid to check; guaranteed never to be null  */
  private BoundedGrid<Actor> gr;

  /** @return an Actor in the grid gr with the most neighbors; null if no actors in the grid.
   */
  public Actor actorWithMostNeighbors()
  {  /* to be implemented in part (a) */  }

  /** Returns a list of all occupied locations in the grid gr that are within 2 rows
   *  and 2 columns of loc. The object references in the returned list may appear in any order.
   *  @param loc a valid location in the grid gr
   *  @return a list of all occupied locations in the grid gr that are within 2 rows
   *          and 2 columns of loc.
   */
  public List<Location> getOccupiedWithinTwo(Location loc)
  {  /* to be implemented in part (b) */  }

  // There may be instance variables, constructors, and methods that are not shown.
}
```

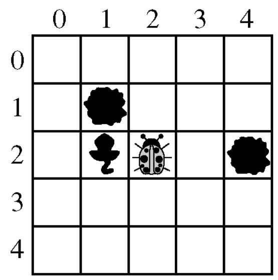
GO ON TO THE NEXT PAGE.

(a) The method `actorWithMostNeighbors` returns an `Actor` in the grid `gr` that has the most neighbors. A neighbor of a given actor is another actor that occupies any of the given actor's 8 adjacent locations. Consider the following examples.

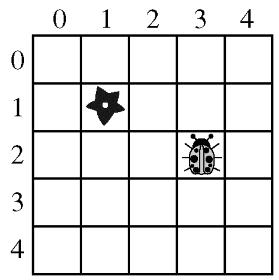
Example 1



Example 2



Example 3



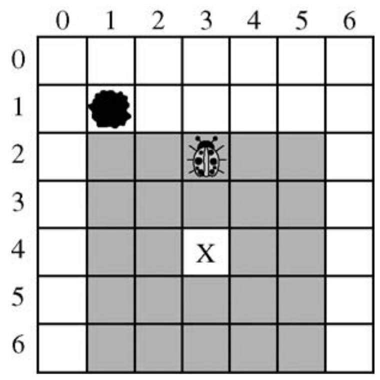
In Example 1, the method `actorWithMostNeighbors` will return the `Actor` (in this case a bug) in location (2, 2). In Example 2, there are three `Actor` objects that have the same largest number of neighbors—the rock in location (1, 1), the critter in location (2, 1), and the bug in location (2, 2). In this case, any one of those three `Actor` objects may be returned. Similarly, either of the `Actor` objects shown in Example 3 could be returned. If there are no `Actor` objects in the grid, the method returns `null`.

Complete method `actorWithMostNeighbors` below.

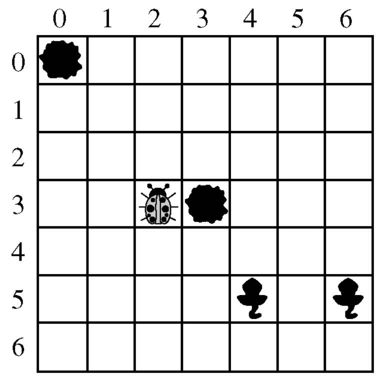
```
/** @return an Actor in the grid gr with the most neighbors; null if no actors in the grid.
 */
public Actor actorWithMostNeighbors()
```

GO ON TO THE NEXT PAGE.

(b) The method `getOccupiedWithinTwo` returns a list containing all occupied locations in the grid `gr` that are within 2 rows and 2 columns of the parameter `loc`. The location `loc` is not included in the returned list, even if that location is occupied. The object references in the returned list may appear in any order. The shaded area in the following diagram shows the group of locations that are within 2 rows and 2 columns of the location labeled X.



For example, consider the following grid.



The table below shows the results of several calls to the method `getOccupiedWithinTwo`.

`loc`	`getOccupiedWithinTwo(loc)`
`(1, 1)`	`[(0, 0), (3, 2), (3, 3)]`
`(0, 0)`	An empty list is returned.
`(3, 3)`	`[(3, 2), (5, 4)]`
`(5, 4)`	`[(3, 2), (3, 3), (5, 6)]`
`(5, 6)`	`[(5, 4)]`

Complete method `getOccupiedWithinTwo` below.

```
/** Returns a list of all occupied locations in the grid gr that are within 2 rows
 *  and 2 columns of loc. The object references in the returned list may appear in any order.
 *  @param loc a valid location in the grid gr
 *  @return a list of all occupied locations in the grid gr that are within 2 rows
 *          and 2 columns of loc.
 */
public List<Location> getOccupiedWithinTwo(Location loc)
```

STOP

END OF EXAM