|  |  |
| --- | --- |
| **AP Computer Science GridWorld Case Study (GWCS)** | |
| **GridWorldLab09** | **Revisit the Grid Interface**  **Introduction of the Critter Class** |

|  |
| --- |
| **Lab Objectives** |
| Revisit the Grid Interface and study the "ArrayList" methods.  Observe the behavior of the Critter class.  Study the methods of the Critter class. | |

|  |
| --- |
| **Lab Prerequisites** |
| **Completed ExpoJava, Chapter 15, and completed GridWorldLab8**  Have an understanding of the ArrayList class and its methods. | |

|  |  |
| --- | --- |
| **Lab Sequence of Steps** | |
| **#** | **Actions** | | **Comments** |
| **01a** | **Create, Compile and Execute Project GridWorldLab09**  Create Project **GridWorldLab09**  Compile and execute the project.  In **Figure 01** you will see the start of the execution.  **Figure 01** | | You have seen the blue **Critter** objects in a previous lab exercise. At that time there was little time devoted to observe the precise behavior. You also did not see any of the methods used by the **Critter** object. |

|  |  |  |
| --- | --- | --- |
| **01b** | **Create, Compile and Execute Project GridWorldLab09**  Observe the execution of project **GridWorldLab09**.  After a short run period, you will observe something like **Figure 02**.  **Figure 02** | The output in **Figure 02** was produced after a short run. In that short period of time two of the bugs have disappeared. The rocks are fine and have not been bothered. |
| **01c** | **Create, Compile and Execute Project GridWorldLab09**  Continue to run the execution. Speed the runtime to fast and observe.  After a while you will get a display like **Figure 03**.  **Figure 03** | **Figure 03** shows what will eventually happen to any execution.  **Critter** objects eat some other objects. In our demonstration proximity to a critter is not good for a flower or a bug.  You should notice that critters do not eat rocks or other critters. |
| **02** | **Examine CritterRunner.java**  Load **CritterRunner.java** in the edit window, like **Figure 04**.  **Figure 04** | There is nothing unusual with the **CritterRunner.java** file. It is similar to many other files and shows the construction of three **Bug** objects, three **Rock** objects and the **Critter** objects.  The manner in which the objects are added to the **world** indicates that each object will be placed at a random location. This means that the output display will not be identical to the ones shown in this lab exercise manual. |
| **03** | **Revisit the Grid Interface**  Load **Grid.java** in the edit window, like **Figure 05**.  **Figure 05** | The **Grid** interface was investigated in an earlier lab exercise. At that time there were five methods at the end that were not explained.  Each of the methods involved something with the **ArrayList** class. Back then the **ArrayList** was completely foreign. Now you have had an introduction to this dynamic array and there is an opportunity check the remainder of the method of the **Grid** interface. |

|  |  |  |
| --- | --- | --- |
| **04a** | **Revisit the Grid Interface Methods**  Examine the **getOccupiedLocations** method in **Figure 06**.  **Figure 06** | Method **getOccupiedLocations** return an **ArrayList** with all the locations in the current grid that are occupied.  The declaration **ArrayList<Location>** indicates that every element of the **ArrayList** object is a **Location** object. |
| **04b** | **Revisit the Grid Interface Methods**  Examine the **getValidAdjacentLocations** method in **Figure 07**.  **Figure 07** | Every coordinate or (row,col) location in the grid has eight potential adjacent locations. Method **getValidAdjacentLocations** returns an array of locations that are valid from a provided location. |
| **04c** | **Revisit the Grid Interface Methods**  Examine the **getEmptyAdjacentLocations** method in **Figure 08**.  **Figure 08** | Moving an object requires knowledge of available cells. Method **getEmptyAdjacentLocations** returns an array of locations that are adjacent to the provided location and empty. |
| **04d** | **Revisit the Grid Interface Methods**  Examine the **getOccupiedAdjacentLocations** method in **Figure 09**.  **Figure 09** | Method **getOccupiedAdjacentLocations** returns an array of locations that are adjacent to the provided location and occupied. |

|  |  |  |
| --- | --- | --- |
| **04e** | **Revisit the Grid Interface Methods**  Examine the **getNeighbors** method in **Figure 10**.  **Figure 10** | Method **getNeighbors** returns an array of objects that are adjacent to the provided location.  Be careful with this method. At first it may seem that method **getOccupiedAdjacentLocations** and method **getNeighbors** are identical. They are not.  Method **getOccupiedAdjacentLocations** returns an array of **location** objects.  Method **getNeighbors** returns an array of objects that occupy the adjacent locations. |
| **05a** | **Examine the Critter Methods**  Examine the **getActors** method in **Figure 11**.  **Figure 11** | Method **getActors** returns an array of actorsthat occupy neighboring grid locations. |
| **05b** | **Examine the Critter Methods**  Examine the **processActors** method in **Figure 12**.  **Figure 12** | Method **processActors** removes all the actors that are passed in an array. The exception is that no rock or critter objects are removed.  Do not get confused. This is not a case where adjacent actors are removed. It appears that way during execution. This method removes objects that are in an array. How these objects arrived in the doomed actor's bag is another story. |
| **05c** | **Examine the Critter Methods**  Examine the **getMoveLocations** method in **Figure 13**.  **Figure 13** | Method **getMoveLoactions** returns an array of locations that are empty. |
| **05d** | **Examine the Critter Methods**  Examine the **selectMoveLocation** method in **Figure 14**.  **Figure 14** | Method **selectMoveLocation** returns the location for the next move. This move is randomly selected from an array of possible locations. |
| **05e** | **Examine the Critter Methods**  Examine the **makeMove** method in **Figure 15**.  **Figure 15** | Method **makeMove** moves a critter to a new location if the location is not **null**, otherwise the critter removes itself from the grid. |
| **05f** | **Examine the Critter Methods**  Examine the **act** method in **Figure 16**.  **Figure 16** | Method **act** goes through the following steps:  1. Get an array of actors.  2. Process the actors, which means removal  for all but critters and rocks.  3. Get an array of new move locations.  4. Move to a random new location. |
| **06** | **Complete the GWExercises09 Work Sheet.** |  |