|  |  |
| --- | --- |
| **AP Computer Science GridWorld Case Study (GWCS)** | |
| **GridWorldLab10** | **Introduction of the Critter Classes**  **ChameleonCritter and CrabCritter** |

|  |
| --- |
| **Lab Objectives** |
| Observe the behavior of the ChameleonCritter class.  Observe the behavior of the CrabCritter class.  Learn the ChameleonCritter class.  Learn the CrabCritter class. | |

|  |
| --- |
| **Lab Prerequisites** |
| **Completed ExpoJava, Chapter 15, and completed GridWorldLab09**  Have an understanding of the ArrayList class and its methods. | |

|  |  |
| --- | --- |
| **Lab Sequence of Steps** | |
| **#** | **Actions** | | **Comments** |
| **01a** | **Create, Compile and Execute Project GridWorldLab10**  Create Project **GridWorldLab10**.  Compile and execute the project.  In **Figure 01** you will see the start of the execution.  **Figure 01** | | This lab exercise introduces two new classes, which are the **ChameleonCritter** class and the **CrabCritter** class.  The **ChameleonCritter** object is blue and is located in the top-left cell.  The **CrabCritter** object is red and looks like a small crab. |

|  |  |  |
| --- | --- | --- |
| **01b** | **Create, Compile and Execute Project GridWorldLab10**  Observe the running of project **GridWorldLab10**.  After a short run period, you will observe something like **Figure 02**.  **Figure 02** | The running period was quite short. Can you determine the behavior of the two new critter objects?  Remember that the original critter seems to remove any object besides rocks and other critters. Is this true for the new critters? |
| **01c** | **Create, Compile and Execute Project GridWorldLab10**  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** | The outcome of a longer run is not healthy if you are an object of any class besides a rock or a critter.  Can you now tell the behavior of the new critter objects? Probably not. The problem is that this environment is not very good for testing purpose. There are too many critters and other objects moving at the same time. |
| **02a** | **Examine the ChameleonCritter Behavior**  Load **Lab10Tester.java** in the edit window, like **Figure 04**.  Comment out line 40 and line 42.  **Figure 04**    If for some reason, the line numbers are different in your file or the line numbers do not appear, do not be concerned. The object here is not to  place two specific objects on the grid yet.  Specifically, comment out the addition of the **Critter** object and also the addition of the **CrabCritter** object. | You already know the behavior of rocks, bugs, flowers and critters.  You have observed that critters are quite tough because they do not disappear. Right now the **Critter** object and the **CrabCritter** object are commented out.  This allows focus on the **ChameleonCritter** object. |

|  |  |  |
| --- | --- | --- |
| **02b** | **Examine the ChameleonCritter Behavior**  Re-compile and re-execute the project.  The display looks like **Figure 05**.  **Figure 05** | The only critter left is the **ChameleonCritter** object. The testing needs to proceed slowly so you can see what happens when the lonely critter encounters other objects. |
| **02c** | **Examine the ChameleonCritter Behavior**  Do not run the project. Click step and watch one change at a time.  The critter will suddenly turn yellow in **Figure 06**.  **Figure 06** | If you watched carefully you saw the blue **ChameleonCritter** object move close to the yellow **Bug** and its trail of yellow **Flowers**.  The close encounter did not remove any objects, but it did change the critter color from blue to yellow.  *NOTE: It is possible (though rare) for the critter to completely miss the yellow bug and its trail of yellow flowers and instead have a close encounter with the black rock or some other object. If that happens, the color will be different.* |
| **02d** | **Examine the ChameleonCritter Behavior**  Now run the project and continue to watch the **ChameleonCritter** behavior.  After a while the critter may become hopelessly trapped like **Figure 07**.  **Figure 07** | The **ChameleonCritter** object seems quite a bit gentler than its super class **Critter** relative.  There was never any object removed. This occurred even at the expense of the critter trapping itself with flowers and other objects.  What behavior did you observe that happened with our gentle critter? |
| **03a** | **Examine the CrabCritter Behavior**  Load **Lab10Tester.java** in the edit window, like **Figure 08**.  Comment out line 40 and line 41.  **Figure 08** | Now comes the time to test the **CrabCritter** object by itself. Change the commenting like it appears in **Figure 08**. |
| **03b** | **Examine the CrabCritter Behavior**  Re-compile and re-execute the project.  The display looks like **Figure 09**.  **Figure 09** | So far everything is quiet. Keep an eye on the conspicuous red crab object. What will it do? |
| **03c** | **Examine the CrabCritter Behavior**  Do not run the project. Click step and watch one change at a time.  Step around twenty times to get **Figure 10**.  **Figure 10** | You are convinced that the **CrabCritter** moves sideways, but does it disturb its neighbors at all? |
| **03d** | **Examine the CrabCritter Behavior**  Now run the project and continue to watch the critter behavior.  Speed up the execution and watch what happens in **Figure 11**.  **Figure 11** | There appears no additional information by extending the runtime of the execution.  It may be that the starting environment does not allow a good test of the **CrabCritter** object. Sometimes it is easy to draw conclusions from a test, but the test data may be faulty. We need to try some other situations. |
| **03e** | **Examine the CrabCritter Behavior**  Load the **Lab10Tester.java** file.  Change the statements to match the appearance of **Figure 12**.  **Figure 12** | The location of the **CrabCritter** object is intentionally in such a position that all the **Bug** objects will march in front of the critter.  This may change the testing environment and allow us to see the true colors of the **CrabCritter** object. |

|  |  |  |
| --- | --- | --- |
| **03f** | **Examine the CrabCritter Behavior**  Re-compile and re-execute.  There are now four **Bug** objects lined up like **Figure 13**.  **Figure 13** | You know that the **CrabCritter** object does not move up or down. It only moves sideways.  You also know how the **Bug** object behaves. All four of the bugs will move North and then make a right turn. Each one will move to the right on the top row right on front of the critter. |
| **03g** | **Examine the CrabCritter Behavior**  Step through the execution around fourteen times and watch.  The critter seems not so gentle after all in **Figure 14**.  **Figure 14** | It appears that the crab will snatch objects that are in front of its face. Each object is removed the moment it tries to move in front of the crab. |

|  |  |  |
| --- | --- | --- |
| **04** | **Examine the ChameleonCritter Methods**  Examine the methods in **Figure 15**.  **Figure 15** | The **ChameleonCritter** class may seem very short. Do remember that this class extends the **Critter** class, which has many methods.  Observe the **processActors** method, which is responsible for the object changing colors.  Notice that the **Color** of the *other* object is retrieved, and then used to become the new **Color** of the **ChameleonCritter**. |
| **05a** | **Examine the CrabCritter Methods**  Examine the constructor and method **getActors** in **Figure 16**.  **Figure 16**    To understand what is happening, compare the method above to the same method from the **Critter** class. This method has only one statement.  **return getGrid().getNeighbors(getLocation());**  **Critter**’s **getActors** method will return an **ArrayList** of all of the neighbors, meaning every object that is adjacent to the **Critter**.  **Critter**’s **processActors** method will then remove all of the objects in this  **ArrayList** from the **Grid**.  **CrabCritter** redefines the **getActors** method will return an **ArrayList** of only the neighbors in front of the **CrabCritter**. These would be the objects the **CrabCritter** can reach and grab with its pincers. | The **CrabCritter** constructor is pretty tame. It sets the color to red and is done.  Method **getActors** is redefined in the **CrabCritter** class. If you check the method closely you will observe that it only gets actors from three possible locations in front of the crab.    There is no **processActors** method in the **CrabCritter** class because it is simply *inherited*from the **Critter** superclass. |
| **05b** | **Examine the CrabCritter Methods**  Examine methods **getMoveLocations** and **makeMove** in **Figure 17**.  **Figure 17** | Method **getMoveLocations** is necessary, because the crab only moves sideways if possible.  Method **makeMove** is redefined. The crab has a way out if it cannot move. Only in such a situation does randomly turn left or right. |
| **05c** | **Examine the CrabCritter Methods**  Examine methods **getLocationsInDirections** in **Figure 18**.  **Figure 18** | This method looks complex. The goal of this method is to consider possible valid locations from different directions. |
| **06** | **Complete the GWExercises10 Work Sheet.** |  |