|  |  |
| --- | --- |
| **AP Computer Science GridWorld Case Study (GWCS)** | |
| **GridWorldLab05** | **Create New Objects During Program Execution** |

|  |
| --- |
| **Lab Objectives** |
| Add and remove objects from the GridWorld display.  Learn the difference between a class and an object.  Understand the special class method, called the constructor.  Use constructors with parameter information.  Learn about overloaded constructors. | |

|  |
| --- |
| **Lab Prerequisites** |
| **Completed ExpoJava, Chapter 08 and completed GridWorldLab04**  Understand how to construct Class objects with the **new** operator. | |

|  |  |
| --- | --- |
| **Lab Sequence of Steps** | |
| **#** | **Actions** | | **Comments** |
| **01** | **Check the Files for GridWorldLab05**  Open **My Documents** and navigate to the **GridWorldLab05** folder.  You will get the display with 13 files, like **Figure 01**.  If you have these exact same files, you are ready to proceed.  **Figure 01** | | This lab assignment is designed to help understand the concept of creating objects with known classes. This should clarify the difference between a class and an object.  You will not be altering any files this time or create any new bugs for the assignment. Everything is strictly an investigation and  try-it-out type of exercise. |

|  |  |  |
| --- | --- | --- |
| **02** | **Create a Project for GridWorldLab05**  Close the **GridWorldLab05** folder.  Create a project for **GridWorldLab05**.  That will get you to the display in **Figure 02**.  **Figure 02** | This lab assignment has many classes. There is one file for each class. Every file that stores a class ends in **.java**. It is possible for multiple classes to exist in one file, but proper program design calls for *one class, one file*.  In many cases the complex interaction of multiple objects in a complex program, like GridWorld, is only possible when each class is located in its own file.  Java requires that the file identifier - minus the java suffix - is the same as the class identifier. |
| **03** | **Compile and Execute the GridWorldLab05 Project**  Compile and execute the project.  You will get the display in **Figure 03**, with eight objects.  **Figure 03** | The execution displays many objects, which you have not seen previously. The details of the behavior of any new object are not the goal of this lab assignment.  Our primary concern is to learn how to create objects for each one of the available classes. Furthermore, you will learn how you can determine what classes are available for any particular GridWorld execution. |

|  |  |  |
| --- | --- | --- |
| **04** | **Examine the Driver Class**  In the **File View** window, Double click **Lab05Tester.java** to load the file in the edit window.  You will get the display in **Figure 04**.  Note the commands that create eight different objects.  Also note that each newly constructed object must be added to **world**.  **Figure 04** | There is very significant information shown in the **Lab05Tester.java** file. You notice one object is constructed for each one of the available classes.  Two statements are necessary for an object to be included in the GridWorld.  First, an object must be constructed with a statement that uses the **new** operator, like:  **Actor obj1 = new Actor();**  In that statement **Actor** is the class.  **obj1** is object of the **Actor** class.  **obj1** is also called one *instance* of **Actor**.  The second **Actor()** is the constructor call.  Constructing an object is not sufficient.  The newly constructed object must be added to the GridWorld.  Most of the previous lab assignments have randomly added objects to the GridWorld. In this example you will notice that each object is inserted in the GridWorld **world** object at a specific starting location.  Observe that a special class, called **Location** is used with each object. The parameter information for each **new Location(row,col)** statement indicates where the new object will be located in the GridWorld. |

|  |  |  |
| --- | --- | --- |
| **05** | **Observe the Execution**  Click the **Run** button and watch the execution.  At some point you will get a display, similar to **Figure 05**.  Stop the execution when there are only three objects left.  **Figure 05** | The movement of most objects is quite random; however, the behavior of the different objects is such that after a reasonable amount of time only three objects are left.  If it seems to be taking a while for all of the **Flower** objects to get eaten by the **Critter**s, crank the speed all the way up to **Fast**. Eventually, there will be just 3 objects left.  These are objects of the **Critter**, **CrabCritter** and **ChameleonCritter** classes.  The exact nature of the behavior and how this all works will be investigated in future lab assignments.  NOTE: In rare instances a **Critter** object gets eaten and you may have only 2 objects left. You may also have several **Flower** object left because there is not a **Critter** object to eat them. **Critter** objects are not suppose to be able to eat other **Critter** objects; however, it can does happen on rare occasions. This could be due to a glitch in the **Critter** class which is in the **.jar** file. |
| **06** | **Clear All Objects From the Grid**  Remove all three remaining objects from the grid.  To do this, click on the object and then select **removeSelfFromGrid**  or use the **Delete** key.  At the conclusion you should have an empty grid like **Figure 06**.  **Figure 06** | The simplest way to delete an object from the GridWorld is to use the **Delete** key.  This option is available before the objects start moving as a result of clicking **Run**. The **Delete** key can also be used after using **Step**.  Each class has a **removeSelfFromGrid** method, which can be used to manually remove objects with mouse clicking. That method is used by the program anytime an objects needs to be removed from the GridWorld.  Click on any object and the window of available methods appears. Click on **removeSelefFromGrid** and the object is no longer in the GridWorld.  You can also press the **Delete** key to remove an object from the grid. |
| **07** | **Display Available Classes**  The **Lab05Tester.java** class added eight classes to the GridWorld**.**  It is now possible to add an object of any one of the eight classes to any  available cell location on the GridWorld.  **Figure 07** shows the display after clicking on the **(0,0)** cell in the grid.  **Figure 07** | There are eight classes available. Each class identifier appears in blue. If you are looking at a black-and-white display, look for identifiers with upper-case letters. Classes always start with an upper-case letter.  You will note that the first five classes appear without the **info.gridworld.actor** names preceding the class name. This is very logical.  The **BoxBug**, **ChameleonCritter**, **CrabCritter**, **OctagonBug** and the **SpiralBall** classes all have something in common. The files, which declare these classes are all located in the same folder as the **GridWorldLab05-Ch07** project.  The **Actor**, **Bug** and **Critter** classes are available to this program execution because a statement like  **import info.gridworld.actor.Actor;**  is used for each one of the classes.  When you follow the path indicated by the **import** statement you will find each one of the mentioned classes.  You will note some other features. Some of the class names are followed by an empty set of parentheses. These classes construct a **new** object without any information for the constructor. Some other classes display data types between the parentheses, which indicate the parameter data type that needs to be used.  There are two classes, **Bug** and **Flower**, which show up twice. The first time there are empty parentheses following the **Bug**/**Flower** identifiers. Using this constructor will create an object with a random color. You can also use the *overloaded* constructor and provided the specific color for the new **Bug**/**Flower** object. |

|  |  |  |
| --- | --- | --- |
| **08a** | **Add Objects to the GridWorld Interactively**  Add an **Actor** object by clicking on the **Actor** bar at location **(0,0)**.  Move to the other three corners and add **Actor** objects to each corner.  Your display must now appear like **Figure 08**.  **Figure 08** | You created four **Actor** objects intentionally. Right now you are looking at four objects. Each object is identical in appearance, but there are four different objects nevertheless, in four different locations.  This may help in understanding the difference between an *object* and a *class*. There is only one class. The class is the template; it is the cookie cutter; it is the general set of instructions to create something.  One object is one instance, one example of a class. In this case there are four objects or four instances of the **Actor** class. There is not a limit of four objects. It is possible to fill the entire grid with **Actor** objects. |
| **08b** | **Add Objects to the GridWorld Interactively**    Move to grid location **(6,2)** and click. Click on the **BoxBug** class.  A small input window, like **Figure 09** appears.  Enter **5** and click **OK** to see **Figure 10**.  **Figure 09**    **Figure 10** | In an earlier lab assignment you observed that **BoxBug** objects move in a square pattern. This means that **BoxBug** objects require information. Each new object needs to know how large the box or square will become.  This means you need to provide an integer, which indicates the number of cells that the **BoxBug** object travels before it turns. |

|  |  |  |
| --- | --- | --- |
| **08c** | **Add Objects to the GridWorld Interactively**  Run the GridWorld long enough to see the **BoxBug** object form a square.  You now have the **Figure 11** display.  **Figure 11** | Observant students may notice that each side of the square pattern, created by the **BoxBug** path is six cells. This may seem odd, since the input box received number **5** for the parameter information.  Number **5** indicates the number of moves that the object makes before it turns.  The **BoxBug** object already occupies one cell after it is constructed. Once you click run, **5** cells are visited before the bug starts to turn. This results in a total of six cells for each side of the square. |
| **08d** | **Add Objects to the GridWorld Interactively**  Continue now and add objects to create the **Figure 12** display.  **Figure 12** | If the display is in black-and-white, you may have difficulty identifying the colors of the objects.  The higher bug, in the right column, is a default **red**.  The lower bug in the right column, is **green**.  The higher flower is **black**.  The lower flower is **yellow**.  The **OctagonBug** object has input **3**.  The **SpiralBug** object has input **3**. |
| **08e** | **Add Objects to the GridWorld Interactively**  Continue on your own experimenting with different objects and executions to observe how each class constructs objects.  Do this for about five minutes. | It is good to observe class behavior. However, the key concern for this lab assignment is to understand how to construct class objects. Additionally, you need to understand that some constructors require parameters information. Finally, there are classes with multiple constructors. It is possible that the method (constructor in this case) has the same identifier, but there must always be a different method signature. |
| **09a** | **Add the ClassView Window**  Click on **View**, which opens the window in **Figure 13**.  Click on **Class View**.  **Figure 13** | All the lab assignments, up to this point, displayed the **File View** window only.  The **File View** window is vital when you work with projects. You can instantly see all the files that are included with the project.  This also allows convenient editing of any file by double clicking the file name.  You will see that there is another window with useful information for program development.  It is the **Class View** window, which provides information about a selected class identifier. |
| **09b** | **Add the ClassView Window**  Click on **BoxBug.java** in the **File View** window.  Observe the result in the **Class View** window, like **Figure 13**.  **Figure 13** | The **Class View** window displays the members of the class highlighted in the **File View** window.  The blue rectangles indicate *attributes*, also called *instance variables*.  The pink rectangles indicate the *methods* in the class along with any required parameter information.  A small lock in front of a class member means that the member is not accessible from outside the class. This is called a **private** member.  Normally, methods are **public** and can be accessed from anywhere, and attributes are **private**. Methods of the same class have access to **private** attributes. |

|  |  |  |
| --- | --- | --- |
| **10** | **Complete the GWExercises05 Worksheet**  Some answers come from observing the execution.  Other answers come from observing the figures in this lab assignment. |  |