|  |  |
| --- | --- |
| **AP Computer Science GridWorld Case Study (GWCS)** | |
| **GridWorld Lab 02**  **1-Day Minor Assignment** | **Observing Methods of the GridWorld Classes** |

|  |
| --- |
| **Lab Objectives** |
| Investigate objects of four GridWorld classes.  Identify the methods for each object.  Observe the result of activating methods with different parameter values.  Gain better understanding of the difference between classes and objects. | |

|  |
| --- |
| **Lab Prerequisites** |
| **Completed ExpoJava Chapter 03 and completed GridWorld Lab 01.**  Understand creating projects with JCreator, including using JAR files. | |

|  |  |
| --- | --- |
| **Lab Sequence of Steps** | |
| **#** | **Actions** | | **Comments** |
| **01** | **Create a Project Name and Location for GridWorldLab02**  You will now repeat steps performed in **GridWorld Lab 01**.  When you are done, your screen should resemble **Figure 01**.  **Figure 01** | | The only difference is that now you will use the **GridWorldLab02** folder.  This assignment has 2 parts, just like GridWorld Lab 01.  The first part requires that you follow the steps on the next several pages.  The second part will be to answer more questions based on what you observe and learn during the first part. |

|  |  |  |
| --- | --- | --- |
| **02** | **Saving Your Workspace**  Click **File** and select **Save Workspace** as shown in **Figure 02**.  **Figure 02** | A *project* can contain several *files*.  A *workspace* can contain several *projects*.  For most GridWorld Labs, our *workspace* will only have one project.  There are times that you might not finish a GridWorld lab by the end of class and may need to complete it later. By saving the *workspace* you save all of the project information.  **NOTE: Saving the workspace does NOT save the individual files in that workspace.** |
| **03** | **Closing Your Workspace**  Click **File** and select **Close Workspace** as shown in **Figure 03**.  **Figure 03** | **NOTE: It is very important to *close your workspace* when you are finished with a GridWorld lab.**  If you do not close the workspace a peculiar side effect can happen. You might be working on a new completely different lab, one that has nothing to do with GridWorld. When you compile and execute that lab, your GridWorld project will execute instead because the workspace is still open.  For now, we are closing the workspace simply to demonstrate reopening it. |

|  |  |  |
| --- | --- | --- |
| **04a** | **Loading Your Workspace**  Click **File** and select **Open Workspace** as shown in **Figure 04**.  **Figure 04** |  |
| **04b** | **Loading Your Workspace**  Browse to the folder:  **\LearnAPCS\APCS-Learning Units\APCS-04-Program Organization \Labs04\GridWorldLab02**  as shown in **Figure 05**.  **Figure 05**    Select the **.jcw** (JCreator Workspace) file, and click **Open**. | If your “Files of Type” is set to  “All Files” you will see several  additional files. |
| **04c** | **Loading Your Workspace**  Your screen should now look like **Figure 06**.  In the **File View** window, double click the **BugRunner.java** file to make that file show up in the edit window.  **Figure 06** | Your workspace is loaded, and the **GridWorldLab02** project is loaded  along with it.  Loading a workspace does NOT automatically load any file in the edit window. You must select which file  you want to work with manually by double-clicking it. |
| **04d** | **Loading Your Workspace**  Your screen should now look like **Figure 07**.  **Figure 07**    Now that everything is loaded, we can compile and execute this project. |  |

|  |  |  |
| --- | --- | --- |
| **05** | **Compile the Project**  Click the **Compile Project** (or **Build Project**) icon.  You will see the display of **Figure 08**.  **Figure 08** | After each project is created, it must be compiled once. From that point you can execute multiple times without re-compiling.  However, any time you make a change in any file, it is necessary to re-compile program.  You also need to be careful that you do not unintentionally execute an older program. Imagine the following scenario. You start with a small program, which compiles and executes. Now you make some additions. You pay little attention to the output window and you do not realize the compile errors. You do observe the **Process Completed** message. This message is fine without errors, but when there are errors the completion means that the compiler is finished.  Now comes the mistake. You continue and execute the program. Remember that you previously compiled and executed the program. This means that a fully functional bytecode or **\*.class** file exists and is ready for execution. This means you get the previous execution. You may be very surprised that none of your new additions are apparent. |
| **06** | **Execute the Project**  Click on the Execute icon to get a display similar to **Figure 09.**  **Figure 09** | This GridWorld shows four objects. Each object is one instance of a class. You see an object of the **Bug** class, **Rock** class, **Flower** class and the **Actor** class.  Right now it may seem difficult to appreciate the difference between a *class* **and an *object***. Think of **Cat**. **Cat** is a class and **SnowFlake** is one example, or as computer scientists say, one instance of **Cat**.  A class is a toolkit that stores tools. If a program is well designed then you will find that tools with a common purpose are placed in one toolkit.  The primary goal of this lab is to investigate three different classes of the **GridWorld** program and observe the tools available for each class.  In Java the tools or actions of a class are called **methods**. All methods perform some type of action. there are methods, which can process without information and other methods require information before the process can start.  The information required by a method is called a parameter. The number of parameters, also called arguments, can fluctuate. A good example is the computation of the *square root*. It is not possible to start computing a square root without an initial number. The square root of 100 is 10.  The square root of 144 is 12 and so on. |
| **07a** | **Check the Classes Available in GridWorld**  Click on any open cell to get a display similar to **Figure 10.**  **Figure 10** |  |
| **07b** | **Creating GridWorld Objects**  Go ahead and select the first choice: **info.gridworld.actor.**Actor**()**  Your display should be similar to **Figure 11.**  **Figure 11** |  |
| **07c** | **Creating GridWorld Objects**  Click on 6 more empty cells and in each create another kind of GridWorld object. Some of these objects will be created with a default color. Some will require you to specify a color.  **Figure 12** | You already created an **Actor**.  You need to create the following:  2 kinds of **Bugs**  2 kinds of **Flowers**  2 kinds of **Rocks** |
| **08** | **Check the Methods of an Actor Object**  Click on either **Actor** object.  A window of **Actor** methods is shown **Figure 13**.  **Figure 13** | When you click on an object a window appears with all the available methods.  In **Figure 13** you observe that the **Actor** class has many methods. Anytime a *blue* method name is followed by an empty set of parentheses, it means that the method does not require information. When there is information between the parentheses, it is an indication that one or more pieces of information are required.  You will see plenty of information in the window that looks plenty confusing. This case study is meant to be understood over time. You cannot instantly comprehend everything all at once.  Right now the main purpose is to realize that each of the objects you see is one instance of a class. We see objects of four different classes, which are **Actor**, **Bug**, **Flower** and **Rock**.  Each class has multiple methods. Methods perform desired operations for the objects.  When you click on any object a list of methods appears. We will now examine various methods of each of the other classes. |
| **09a** | **Check the Methods of a Bug Object**  Click on any **Bug** object and your display should be similar to **Figure 14**.  **Figure 14** |  |
| **09b** | **Check the Methods of a Bug Object**  If you left-click on the **moveTo** method an input window appears, shown  in **Figure 15**, which needs a **row** and **column** location.  Initially you will see the coordinate value of the current bug location.  Enter **0** and **0** for the new location.  Click **OK** and watch the bug switch to the top-left corner.  **Figure 15** | When you click on the **moveTo** method of the **Bug** class a small window appears. This window requires information. The **moveTo** method needs two parameters, two pieces of information to handle a process correctly.  In the case of the **moveTo** method the action is to move to a new location. Every location on the grid is identified by a *row* and ***column*** value. The top-left cell in the grid is location (0,0), which is the location the bug will move to after the **OK** button is clicked. |

|  |  |  |
| --- | --- | --- |
| **09c** | **Check the Methods of a Bug Object**  Click on the **Bug** object in its new location (Row 0, Column 0)  This time select the **turn** method.  The bug will make a clock-wise, 45 degree turn.  Click the bug and select turn two more times. (For a total of 3 turns)  The bug will appear as you see in **Figure 16**.  **Figure 16** | The next method to examine is the **turn** method of the **Bug** class. The parentheses are empty. This method does not require any information. Every time the method is activated, the bug turns 45 degrees in a clock-wise direction.  You need to understand something. With program code a method is activated by using the name of a method in a program statement together with any required parameter information.  In this Gridworld case study the creators of the program - a Collegeboard committee - designed the program in such a way that the behavior of class methods can be observed in a graphics display.  In this manner you not only learn about the capabilities of the GridWorld case study and its classes, but also about the manner in which classes and methods behave in general. |
| **10** | **Check the Methods of a Rock Object**  Later in the exercises, you will do many more method investigations of each the four classes. Right now let us look at the **Rock** object.  Right-click on any **Rock** object to open the method window.  Left-click on the **setColor** method and open the pull down window.  You will see a variety of available colors, as shown in **Figure 17**.  Click on **red** and then **OK**. The rock will change color.  **Figure 17** | The **Bug** class is an interesting class. The **Rock** is less exciting, but it does have various capabilities.  One of the methods of the **Rock** class is the ability to change color. In this program, you handle the parameter information with a *pulldown* menu. Normally, the parameter information would be a group of color values. |
| **11** | **Check the Methods of a Flower Object**  Now click on the **Flower** object. There many similar methods.  Click on the **removeSelfFromGrid** method and watch what happens.  The **Flower** object is gone.  It literally did remove itself from the grid.   |  |  | | --- | --- | | **Before** | **After** | |  |  | | Every object has the ability to remove itself from the **GridWorld** display. This happens for various reasons that will be investigated in future labs.  When you activate the **removeSelfFromGrid** method of the **Rock** object or any other object, that object will disappear. |
| **12** | **Complete the GWExercises02 Worksheet**  You will earn a grade at the conclusion of each GridWorld lab.  Sometimes the lab asks you to follow steps and observe behavior.  In such a case there will be an exercise sheet with short answer questions about the observation you made during the lab.  This is the case for the first 2 GridWorld Labs.  At other times you are expected to use the existing methods to create a certain output, alter the behavior of some objects or create brand-new objects. This will happen in future labs. |  |