CS 371 Laboratory 7: UML

In this lab, you will primarily be learning to use a UML (Unified Modeling Language) tool called Dia. While Dia is one of many different diagramming tools, UML is the standard for the vast majority of such tools.

UML appears frequently in software design. The primary goal in this lab is to practice creating use-case diagrams and class diagrams.

Part 0: Setup

Log in to the engineering kiosk via desktop.up.edu.

Locate and run Dia via the Start menu on the lab PC. You should see an interface very similar to a paint program. Dia is used to create diagrams of all sorts, including UML.

In the middle of the left-hand pane (it could be a separate undocked window), find the UML tool pane. Examine these and try a few out. Not all will be familiar, but many should be. Mouse over the shapes in your toolbox if you’re unsure what they are. You can either drag shapes onto the canvas or click once on the component then again on the canvas. Use the handles on the corners to resize.

Options for a shape (such as its name, instance variables and methods for a class, etc.) are generally set by double clicking on it. In many cases, you can also type a name immediately after inserting a shape. If you need to edit text that you typed in this way, click the “select text” icon at the top of the tool pane (to the right of the pointer icon), then select the text that you wish to edit. Make sure to change back to the select (pointer) tool before trying to place or reposition additional shapes.

This illustrates an important point about the way Dia works – unlike some tools there is not a specialized “Class Diagram” or “Use Case Diagram” mode. You are always presented with the same “canvas”, but you choose the sort of shapes which you will place there. This does mean you can mix and match shapes from different types of diagrams, but with great power comes great responsibility – you will probably only confuse others and yourself if you try to incorporate circuit schematic symbols into a UML class diagram.

The representations in Dia may not look exactly identical to what you’ve seen in class (it draws some arrows slightly differently, etc.), but that’s okay – the purpose of this lab is to gain experience using a UML diagraming tool and see how it can streamline the design process, not to make pixel-perfect diagrams.

Part 1: Create a New Eclipse Project

Download the BallAnimation.zip file from the course web page (learning.up.edu) and unzip it somewhere on your P: drive. It will not run in Android Studio because it is not an Android app. Instead, run the Eclipse IDE and create a new project. Then add the .java files. Even if you have never used Eclipse before, you should find the paradigm familiar and quite similar to Android Studio. The steps generally are as follows, but see hints below before you execute these steps:

1. Create a new Java project. See Hints below.
2. Unzip the BallAnimation.zip file.
3. Add the files from the BallAnimation folder to the project. You can drag and drop them into the src folder in Eclipse.
4. Run the program.

Hints:

* Uncheck the box to create a module-info.java file when creating a new project.
* If you get asked to open the Java Perspective, click Yes.
* Close the tab for the welcome window to see your new project.
* You may need to set up the Run Configuration before you can run the program. Set it as a Java Application and set the Main class to the class that has the public static void main method in your project.

When you run the program, you should see a dialog followed by an animated graphics window. Click on the animation window to see what happens when you do this. Close the program when you are finished. You may want to run the program again to see the different object types.

**Checkpoint 1 (15 points):** Show your instructor or lab assistant that you have created an Eclipse project and that the Ball Animation program runs correctly.

**Part 2: Use Case Diagram**

To complete this checkpoint, create a simple use-case diagram for the BallAnimation program.

First, go to File -> Save and choose a location (I recommend the lab7 folder on your P: drive) to save the .dia file which you will be creating.   
  
A note for the future: You should notice that Dia has an export functionality, so you can generate a jpg or a png for use in documents and presentations. Or send one home for Mom to put on the fridge.

Create a diagram with the following elements. You can see the name of an element by hovering over it.

* A (large) package representing the program
* An actor named “User” who is interacting with the program
* A use case for each of the following major interactions:
* Select Object Type
* Re-center the Animation
* Close the Program

Add appropriate associations between the above components by selecting Association from the toolbox (it should be in the second row):

* Add an Association by clicking on the first object and dragging to the second.
* The line adjusts if you move objects around.
* To detach a connection, click on the line, then click and drag the red square next to the object from which you wish to detach it.
* Most shapes have multiple “attachment points” – they are indicated by the x’s surrounding the shape. Bring a line near one of the attachment points to attach to that shape.
* Double-click (or right-click) to bring up a Properties window.
* By default, Associations have small arrows near the middle of the line; these are for
* use when naming the relationship. To add large arrows at the ends of a line, which you need for your Use Case diagram, use Show Arrow on the Properties page.

**Checkpoint 2 (15 points):** Have your instructor or lab assistant verify that you have created an appropriate use case diagram.

Part 3: Create a class diagram

Now you will create a class diagram that contains a UML object for the Ball class. Examine the Ball.java file.

Select File -> New to get a blank canvas. Be sure to give it a file name.

Create a new class by selecting the Class object from the Objects list and clicking in the diagram. Name this new class Ball.

Now add two properties (instance variables) to the class:

* Double-click on the class object and go to the Attributes pane. Click “New” to add a new property. Name this new property radius and set its type to int. Give it a visibility level of protected.
* Add a second property named speed with type int and visibility private.

Click “Ok” and notice that Dia automatically adjusts the way the properties are displayed to match their visibility.

Using the source code as a guideline, repeat the above process to add the rest of the instance variables from the ball class. Refer to the source code for the details of each property.

Methods are added by a process similar to that for attributes. Follow these steps:

* Select the Operations tab in the class properties window.
* Select “New” and add an operation for the constructor. Dia does not treat constructors specially; just create a public method with no type and name Ball.
* Make sure “Inheritance Type” is set to “Polymorphic (virtual)”.

Fill in the fields for the first parameter:

* Click “New” below Parameters. Note that the list in the lower left will populate as you add parameters.
* Set Name to rad
* Set Type to int
* You can leave the rest of the fields as-is

Add each of the remaining four parameters. You should refer to the source code for their types.

Consult the source code for Ball.java and you will see that in addition to the constructor there are five other methods. Add all of these methods in your UML diagram. Be sure that each one has the correct parameters, return type (including void – you need to specify this to differentiate from the constructor, which has no return type, which is not the same as a void return type) and visibility. Again, you can leave the rest of the fields as they are.

Now add the Balloon class to the diagram as defined in Balloon.java.

Balloon extends (inherits from) the Ball class so you will need to use the Generalization connection to indicate this. Be sure to drag from the parent to the child class so that the arrow ends up pointing the right direction.

Balloon has no additional properties. Its constructor has the same arguments as Ball. It also overrides the tick method. The rest of Balloon’s methods are inherited from Ball, so we do not duplicate them here.

(Hint: If you’re feeling lazy, copy-paste does work on shapes in Dia. Balloon even starts with the string “Ball”...)

**Checkpoint 3 (30 points):** Have your instructor or lab assistant verify that you have created an appropriate class diagram.

Part 4: Comments and Code Generation

Add descriptive comments to your classes, including a brief description of the class as a whole and a brief description of each method. Comments can be added in the properties dialog that appears when you double click a class. If you want to see the comments to make sure they got inserted correctly, check the Comments visible box. But uncheck this for your checkpoint because it creates too much clutter.

For the purposes of this lab, you can cut-and-paste text from the code. Be careful **not** to include the Java comment symbols (**//**, **/\***, **\*/**).

Your comments should make use of Javadoc tags (they start with @). Each tag should be on a separate line. In each class comment, include:

* @author followed by the author’s name (you can use your name as the author)
* @version followed by the version number (you can use any version number you want)

For method comments, use the method comment entry box (a ridiculously small box on the right side of the properties page, near the middle). Make the properties page as large as possible; this will enlarge that box so that you can see what you are doing. (Alternatively, compose your comments in Eclipse or Notepad and paste them into the teeny tiny box).

Each method comment should consist of:

* A brief description of the method’s function
* @param followed by the parameter’s name (not the type) and a brief description of its purpose/meaning. Repeat this for each parameter, one per line.
* @return followed by a brief description of what the method returns. This is omitted for constructors and void methods.
* (You don’t need it for this lab, but if your method throws exceptions then you should also include a line @throws followed by the exception type and the conditions for which it is thrown.)

*Note:* While Dia does have an entry box for method parameter comments, these unfortunately do not yield appropriate @param entries in the resulting method header. So, do not use them. Instead, you will have to add @param entries to the method comment as described above.

Each instance variable also has a comment entry box. Each parameter comment should consist of a brief description of the parameter’s purpose (no need to preface with “@param”).

Dia can be used to generate Java code from a class diagram. Generate code by the following process:

1. Make sure your current diagram file is saved. Since Dia has problems generating code from files saved on the P drive, save the file somewhere on the C drive temporarily (the Desktop, for instance, will work fine).

2. Select “Export” from the File menu. An Export file selection dialog appears.

3. Select the folder where you want the generated source code files to be placed (the Desktop is good for now).

4. Choose “XSL Transformation filter (\*.code)” under “Export Options” and click Save.

5. Specify that you are exporting from UML to Java in the next dialog box.

6. That should have generated a .code file and two .java files (one for each class in the diagram).

1. Open Eclipse in a new workspace.

2. Create a new Java project.

3. Expand the project and locate the src folder in the project tree.

4. Right click on the src folder and select Import…

5. Expand the General folder and select File System.

6. Locate the files Dia generated for you and import them into the project. You won’t be able to see them inside the folder selection window if they aren’t in a folder, but you’ll be able to select them after you choose the destination (i.e. Desktop).

7. Review the source code that Dia generated from your class diagram. Dia’s generation is not perfect. Perform the following steps:

1. Clean up poor formatting. (ctrl + shift + F to auto format)
2. Make sure that all method header comments begin with /\*\* instead of /\* as this is required for JavaDoc comments.
3. Add import statements as needed. (Eclipse will offer these fixes for you.)
4. Add dummy return statements, e.g., return 0, to any method that returns an int
5. Dia creates an @ for every method parameter, but these are not recognized by JavaDoc, which is why you added @param for those parameters. So remove the extra lines with @.
6. If you fix everything and Eclipse still shows errors in the file, trying saving the file, which forces a recompile of that class.

**Checkpoint 4 (30 points***):* Have your instructor or lab assistant verify that you have successfully imported your generated code into Eclipse with no syntax errors and appropriate formatting.

Part 5: Generate JavaDoc Documentation

JavaDoc is a tool that will examine the comments in source code and automatically generate HTML pages that act as a reference guide for the source code.

* 1. In Eclipse, you do this by selecting Generate Javadoc from the Project menu. A wizard appears. The path to the javadoc.exe should already be correct.
  2. Be sure that the checkbox next to your project’s name is checked. If it’s filled with a black square, that means it is only partially selected. Change the selection to a checkmark.
  3. Select “Private” in the radio button so that all classes will be documented.
  4. Select a destination folder for your Javadoc files (P:\CS371\Lab7\Javadoc would be good – a lot of files will be generated).
  5. Press “Next” until you reach the final page. The default “JRE Source Compatibility” of 16 is fine.
  6. Click Finish.

Open the newly generated Ball.html and Balloon.html files and check the results.

**Checkpoint 5 (10 points):** Have your instructor or lab assistant review your HTML pages and verify that they are complete and correct. Demonstrate that you can hover your cursor over a method name in your code and see your documentation display.

Finish up

Make sure that you have **saved all files on your P: drive (including the temporary files on the Desktop)** and before you log off.