COMPSCI 230 S1 C - Assignment 1

Due Date: Friday 25th March, 2011 (9:00pm)

Assessment

- Due: Friday 25th March, 2011 (9:00 pm)
- Worth: 7% of your final mark

Files

- A copy of this document, as well as relevant files for this assignment can be obtained from the CompSci 230 assignments page on the web:
  http://www.cs.auckland.ac.nz/courses/compsci230s1c/assignments/
- Submit your files using the web drop box (https://adb.ec.auckland.ac.nz/adb/).
- See at the end of this document for what files to submit.

Aims of the assignment

- Solving problems using Java, using an object-oriented approach, using classes, interfaces and abstract classes to familiarise with inheritance and polymorphism.

Please note

- The work done on this assignment must be your own work. Think carefully about any problems you come across, and try to solve them yourself before you ask anyone for help. Under no circumstances should you take or pay for an electronic copy of someone else's work.
- Penalties for copying will be severe – to avoid being caught copying, don't do it.
- To ensure you are not identified as cheating you should follow these points:
  - Always do individual assignments by yourself.
  - Never give another person your code.
  - Never put your code in a public place (e.g., forum, your web site).
  - Never leave your computer unattended. You are responsible for the security of your account.
  - Ensure you always remove your USB flash drive from the computer before you log off.

Your name, UPI and other notes

- In this assignment your UPI must appear in the title of the frame (i.e. replace "upi123" with your actual UPI).
- All your files must include your name, UPI and a comment at the beginning of each file.
- All your program files must compile without requiring any editing.
- All your program files must include good layout structure.

1 Bounce

The application given is a simple bouncing program. Different shapes move around in various paths.

Shape creation

The user can create a new shape by clicking anywhere within the panel area of the program. The properties of the newly created shape are based on the default values saved in the program.

Selecting/deselecting shapes

The user can select a shape by clicking anywhere on the shape. If a shape is selected, all its handles are shown. The user can change the path type/width/height for all selected shapes by changing the default values with the help of the tools provided at the top of the application interface (but the shape type can't be modified once a shape has been created). Clicking on a selected shape will deselect it.

What you are to do

First, become familiar with the program supplied. Download all source files from the assignment course page. It is strongly recommended to start as early as you can and implement the parts you know as soon as they are taught in lectures. Your assignment is divided into several stages for ease of completion. Please complete the assignment in order of the stages.

2 Tasks

2.1 *[Task 1]* UML of original system (5 marks)

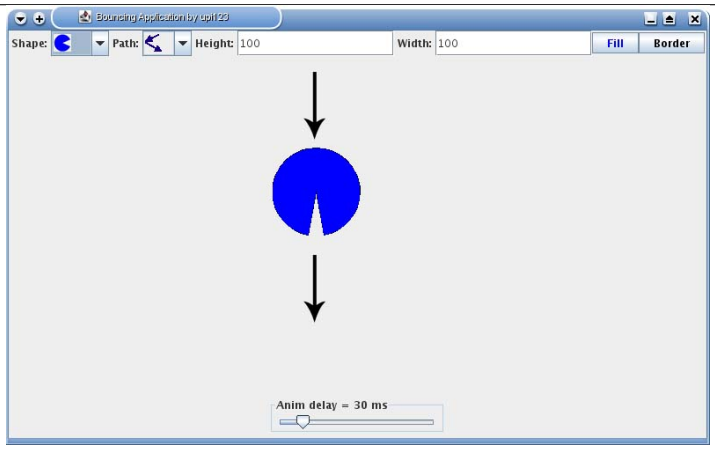
To help you get familiar with the structure of the program, draw a UML class diagram of the existing system. As well as helping you understand the structure, this step will also give you practice drawing UML class diagrams. It will also help you in extending the assignment in the following tasks. Use any tool you wish, even sketching and scanning – as long as it is neat and legible. Name this file "originalUML.jpg" (or use any file type that is common and easy to open for the markers – but if in doubt, at least convert to PDF). You should show the most important methods and attributes also, but do not list every single attribute or method. You should judge which are the most important ones to include. You want to only convey the main concepts of the system, not show a detailed implementation view.

2.2 *[Task 2]* The final UML (5 marks)

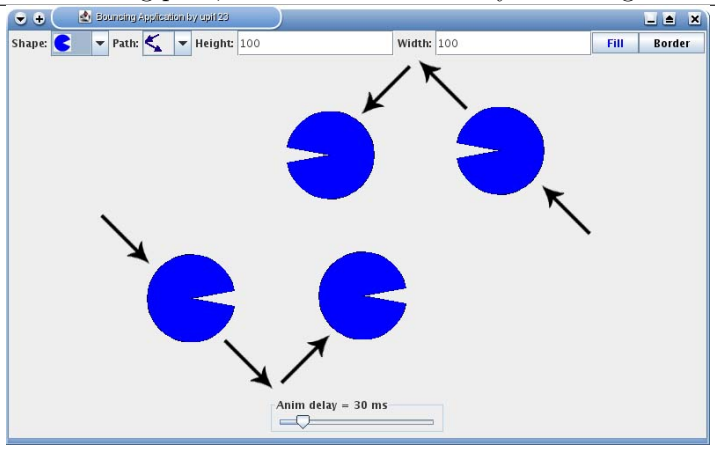
You will now practice more with UML class diagrams – and the purpose now is to plan out your changes by modifying the class diagram *before* you write the respective code. Although you may be tempted to code first (and then draw the UML at the end), you should see this exercise as in a real job that requires you to plan the architecture of your program before you code anything. From such a UML diagram, you will easily be able to explain your system to others and implement your chosen design since you will have a plan to follow. Name this diagram "finalUML.jpg". You may refine this UML as you progress through the assignment, since in reality you might refine the design incrementally. This diagram will also help the markers understand your software system. Again, you do not need to show too much detail – just enough to convey the concepts of how the program is constructed with a few of the main attributes and methods as appropriate.

2.3 *[Task 3]* Pacman's mouth (10 marks)

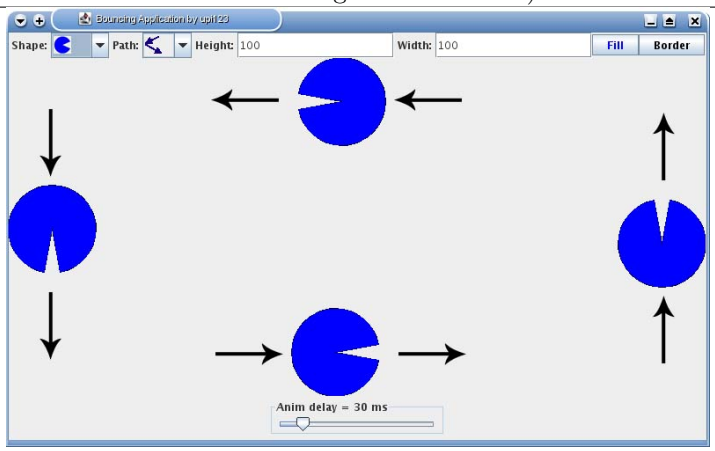
Run the program and create a new Pacman shape. As the Pacman shape moves around, try different paths. You will see that no matter what direction Pacman is moving, its mouth is always facing to the right and not in the direction of its movement. You are now to modify the drawing of the MovingPacman shape so that the mouth is correctly facing the direction of movement, as in the images:



In the falling path, the mouth should always be facing down.



In the bouncing path, the mouth should either be facing to the left (in the case Pacman is moving towards the left on the screen), or facing to the right (in the case Pacman is moving towards the right of the screen).

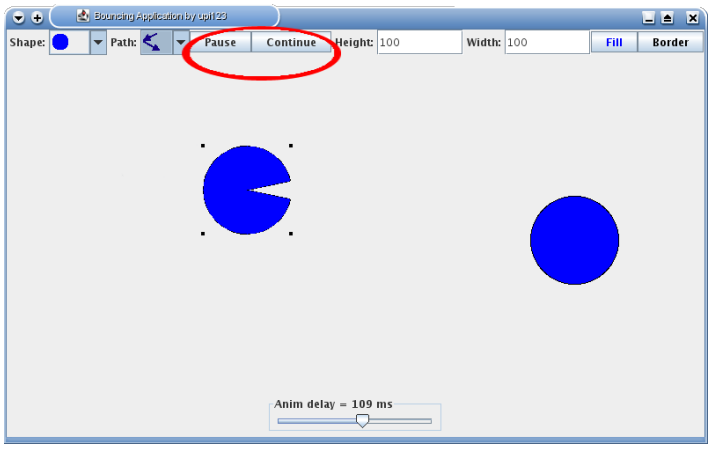


In the border path, the mouth should be facing either left, right, up or down depending on which side of the border Pacman is on.

Hint: make sure you understand how the angles work in drawing the arc of the mouth (i.e. refer to the API for the Graphics2D class). Where does angle 0 refer to? Where does angle 90 refer to? What does angle extent mean? Once you understand this, correcting the draw() method inside MovingPacman will be easy. You will also need to update each of the moving paths (i.e. BouncingPath, FallingPath and BoundaryPath) so that the direction is updated every time the path moves. You will need to add in the appropriate accessor method inside MovingPath to get the direction. You should be making use of the constants inside MovingPath (LEFT, RIGHT, UP, DOWN) throughout where appropriate. In fact, have a look at BoundaryPath to get an understanding how these constants are used.

2.4 *[Task 4]* Pausing and starting the selected shapes (15 marks)

We now want to be able to pause (and then continue) shapes by selecting them. Add two buttons after the path combo box, as shown in the figure below.



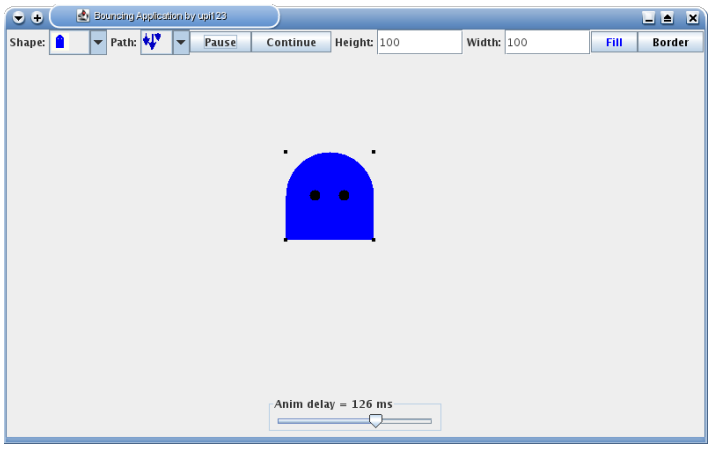
In this example, you will see that Pacman is selected. The user has then paused Pacman and it is therefore not moving. You must make sure that even though Pacman is paused, it should still be opening and closing its mouth – it just doesn't move across the screen. Also in the example above, the circle on the right was not selected, therefore it should continue moving around the panel (while Pacman remains paused). To continue a shape moving again, it must be selected and then the continue button pressed. If a shape is paused, the user should be able to still change the colours, path, width, and height of the selected shape. Therefore, make sure that the user will still have this ability and that the paused shape still remains visible.

Also, when a shape continues moving again, it will continue in the same path and at the same speed as before it was paused (unless of course the user selected a new path from the combo box while the shape was selected and paused). Have a look at how the fillColour and borderColour buttons work, because you should use a similar approach for the new buttons.

When you implement this, you should be thinking in terms of maximising code reuse so that all shapes reuse the same logic. Hint, you will only need to work in MovingShape and of course AnimationPanel and A1. You should not need to write any code in the concrete shape classes (i.e. MovingPacman, MovingRectangle, etc). This is the beauty of inheritance. You will see in a later task that any new shape added (i.e. MovingGhost) will automatically inherit this behaviour.

2.5 *[Task 5]* Add a ghost (20 marks)

You are now required to add a new shape, called MovingGhost, that extends MovingRectangle. First, treat the shape as another shape and focus on making it look like a ghost. Drawing a ghost is simple: all you need is a semi-circle for the head, a rectangle for the body and a couple of eyes. You have to make sure that the ghost is drawn within the boundary of the shape (this is evident when the ghost is selected, as shown by the markers). The example below shows a selected ghost:



Get as creative as you like. You should use the fill colour for the body of the ghost, while the border colour will refer to the ghost eyes. Therefore, if the user selects a ghost, the colours can be changed.

The next thing you want to do is to make the ghost blink on and off. Rather than blinking on and off every single time it moves (the blinking will be too rapid), we want to make the blinking slower. You can implement this by having a visibility counter that decrements. This starts at VISIBILITY_THRESHOLD, and when it reaches zero the ghost becomes invisible. The counter then keeps decreasing (and the ghost remains invisible) until it reaches -VISIBILITY_THRESHOLD, at which case it resets to VISIBILITY_THRESHOLD and becomes visible again.

Think carefully where (and when) the visibility counter decreases. You want it in such a way that if the ghost is selected and paused, it should still keep blinking. Hint, you should be thinking in terms of method refining for the move() method (as opposed to method overloading). Remember, a paused ghost should still blink – just like a paused Pacman will still open and close its mouth. You are given a "ghost.gif" file, use this as the icon for the combo box.

2.6 *[Task 6]* Victims and monsters (25 marks)

We now want the shapes to interact with each other when they collide. The idea is that some shapes are monsters. If 2 monsters collide, nothing should happen. When 2 victims (non-monsters) collide, nothing should happen. But when a victim collides with a monster, something happens depending on the monster type:

- Monsters:
  - MovingPacman: the victim MovingShape should disappear and never be re-drawn again.
  - MovingGhost: the victim MovingShape should get "scared" and pause. To move it again, the user needs to select the victim and press the start button you implemented above to get it moving again. Remember, you have already implemented the logic to pause and continue a shape – think code reuse! ;-)

To denote a shape as a monster, it should implement the Monster interface. This interface has been given to you. Remember, only MovingPacman and MovingGhost should implement this interface. The method in the Monster interface, capture(MovingShape victim), defines what to do to the victim shape (outlined above for the respective monster). In the case of pausing a shape, re-use your earlier code to pause shapes. In the case of making shapes disappear, you should just make them invisible (i.e. nothing gets painted inside the draw() method of the respective shape) rather than try and remove the shape completely (e.g. by making it null or removing it from the shapes vector).

Complete the method checkForCollisions() inside AnimationPanel to loop over all the shapes and check it there were any collisions. To ease the implementation, just use the imaginary rectangular border to determine when shapes collide (i.e. complete and then use intersects() and getBorder() inside MovingShape). Therefore it is fine when Pacman and a circle collide with each other even though the visually drawn shapes do not appear to collide (but the imaginary corners of the rectangles around the shapes happen to intersect). Hint: inside checkForCollisions(), you will be making use of instanceof and casting objects.

2.7 *[Task 7]* Add your own feature (15 marks)

You are now required to get creative and add your own special feature that will make the Bounce program more interesting! You can do anything you like. Maybe it is to add an introduction screen, or keep a count of eaten shapes, or display the number of eaten shapes for Pacmans, or the number of frightened shapes for each ghost, or shows off the use of polymorphism, inheritance, etc (you don't have to, but its just another way to earn marks), etc. To get the marks for this part, the feature should be non-trivial to implement and reasonably interesting. You should be sure to explain what your extra feature is in the A1.txt file (see below), including a very short overview of how you implemented it (what methods you added, in which classes, etc). You shouldn't need to write more than a few sentences.

Files to submit

- A1.txt
- originalUML.jpg (or any common file type)
- finalUML.jpg (or any common file type)
- all your *.java files (even if some files you didn't need to modify)
- all the *.gif files (used as icons in the program)

Please double check that you have included all the files required to run your program (*.java and *.gif) and the A1.txt in the zip file before you submit it. No marks will be awarded if your program does not compile and run.

What to include inside the A1.txt file

You must include a text file named A1.txt in your submission. There will be a 5 mark penalty for not doing so. This text file must contain the following information:

- Your name
- Your login name and ID number
- What is the extra feature that you have implemented? If you don't specify it in this file, the markers might not notice it and you won't get the marks!
- A very brief description of how you implemented this feature – what methods/classes/interfaces you added or modified.
- How much time did the assignment take overall?
- What areas of the assignment did you find easy?
- What areas of the assignment did you find difficult?
- Which topics of the course did the assignment most help you understand?
- Any other comments you would like to make.

Marking

The marks are outlined above for the respective tasks (total of 95 marks). In addition to this, there are 5 marks given for good programming style (i.e. indentation and meaningful variable names). There will be a 5 mark penalty for not including a completed A1.txt file.

Competition!!

There will be a competition, and entering it is voluntary. Should you wish to enter the competition, please send your submission to ngia003@aucklanduni.ac.nz by the due date (25th March, 9pm). The top programs will be demonstrated during one of the lectures, and the class will vote for the best program! When you submit your program to the competition, please also include in the email what features you would like to show-off to the class should it be short-listed. The lecture in which we will see the programs will be announced closer to the due date.