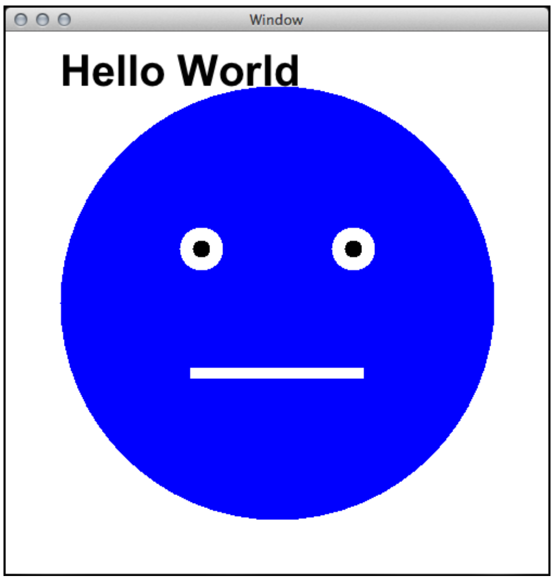
159.261 Lab 3
Java Graphics

1) Smiley Face

This exercise will get you started with writing simple graphical programs in Java. You are given a starting program that creates a window and draws a string on the screen. You must write your own code to draw the smiley face.

1. Open the file `SmileyDrawing.java` in the Lab folder.
2. Compile and run the program, a window should appear with Hello World drawn on the screen.
3. Read through the functions that have already been written in the program. In future labs (and assignments) these will be hidden away inside our Java game engine but you will still need to understand how to use them.
4. Using the functions `drawSolidCircle`, `drawSolidRectangle` and `changeColor` recreate the image of a smiley face shown below (exact dimensions are not too important).
5. Put this code inside a function called `drawFace` that draws a smiley face at a given position. This function should take three parameters – the graphics object `Graphics g` and a position on the screen `int x`, `int y`. Try calling this function from the `paintComponent` with different parameters to make sure the face is displayed in the locations you expect.



2) Simple Animation

In this exercise we will start to look at a graphical program that has a simple game loop which will continuously refresh the display. We will also see how we can listen to input from the user and have our program respond to it.

1. Open file `BouncingBall.java` and compile and run the program to make sure it works (it should just display a blank window at this point).

In particular, pay attention to the following code, this is our main game loop that repeatedly updates the game and draws it to the screen.

```
while(true) {
    // Control the framerate (30 fps)
    double dt = simpleFramerate(30);

    // Update the Game
    update(dt);

    // Tell the window to paint itself
    repaint();
}
```

2. Add code to the `paintComponent` function to draw a circle in the centre of the screen.
   You should use the function:

   ```
   drawSolidCircle(Graphics g, double x, double y, double radius).
   ```

3. Add some code to the `keyPressed` function to change the background colour of the screen whenever the user presses the `'b'` key. You will need to add some way to store the current background colour and use this in the `paintComponent` function.

   note: The `keyPressed` function is called by the operating system whenever it detects that a key has been pressed.

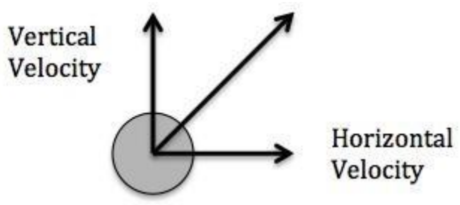
4. Create two variables `x` and `y` that will be used to store the position of the circle. Add some code to the `update` function to make the circle move to the right (change the value of `x`) each time the update function is called. Make sure you modify the `paintComponent` function to draw the circle at the point represented by `(x,y)`.

5. Modify the `update` function to make the circle bounce around inside the window. Use a `boolean` to record whether the ball is moving left or right. Whenever the ball reaches the left or right side of the window (0 and 500), flip the value of the `boolean` to make the ball go back the other way.

3) Bouncing Ball

Representing the motion of a ball using a `boolean` is rather crude. A better method is to store the velocity of the ball. Starting with just the horizontal component, create a variable to store the velocity of the ball `vx`.

Left and right movement of the ball can be implemented by setting `vx` to either a positive or negative value.



1. Modify your code to use `vx` to implement the ball bouncing left and right (reversing the direction of the ball can now be implemented by multiplying `vx` by -1.

2. Extend the motion of the ball to two-dimensions by adding a vertical velocity vy. The horizontal and vertical velocities can be implemented independently of each other.

3. Try changing the framerate by modifying the line:

   ```
   double dt = simpleFramerate(30);
   ```

   Does this change how fast the ball moves? The speed of the objects in our game shouldn't depend on how fast (or slow) the framerate is.

   Modify your `update` function to make use of the parameter `dt` that tells you how much time has passed since the last update (as a fraction of a second). Make sure your ball moves at the same speed when you change the framerate.

4. The framerate of the game is currently implemented using the (not very good) function `simpleFramerate`. Replace this function with the more advanced framerate controlling method described the Lectures. Test the game to make sure it works correctly when changing the framerate.

4) Adding Gravity

Try adding gravity to your game to accelerate the ball towards the bottom of the screen. You can achieve this by applying a constant acceleration to the vertical velocity each time you update the game.

1. Modify your `update` function to add a constant value (multiplied by `dt`) to the vertical velocity `vy` every time `update` is called.

Acceleration due to gravity is usually $9.8ms^{-2}$. However, distance in our game isn't measured in metres, it's measured in pixels. You may have to experiment to find a value of g that gives you the behaviour you want.