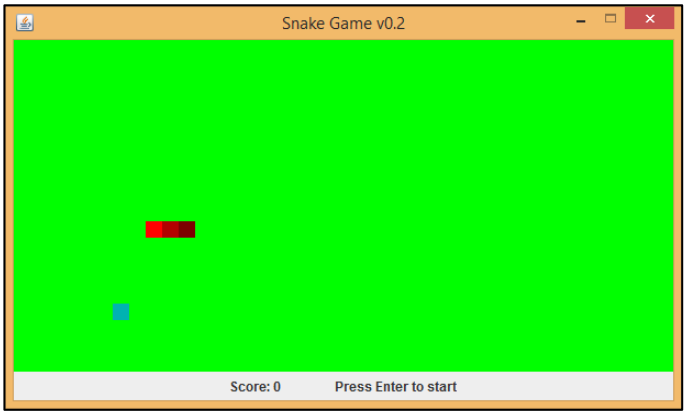


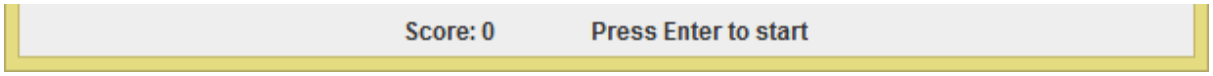
Introduction

You've made some great progress already, now it's time to make the Snake move and add some keyboard controls. Some of this is beyond the scope of just learning Java so I will provide you with some of the files so you can get straight to the interesting stuff!

Tasks

2.1 Set the Score Bar text

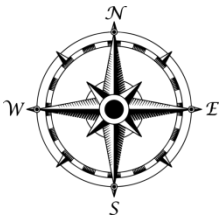
The Score Bar is the part of the window shown below, which shows the score and a status. In this case the status is "Press Enter to start", but we want to be able to change that, for example when we start the game. This is represented by the class ScoreBar.java. If you look in that class you can see there is already a field called statusLabel, which we want to set. The statusLabel field is of type JLabel, which is a way of showing text on the screen.



In the ScoreBar class (in package com.snake.ui):

- Create a new **public** method called setStatus
  - Public void setStatus(String status) {

    }
- Inside the setStatus method:
  - JLabel objects have a 'setText()' method which you can use:
  - statusLabel.setText(status);



2.2 Create a Direction enum

This will be used to keep track of which direction each Cell is facing, and we will use the compass directions to represent this: NORTH, EAST, SOUTH and WEST. We also want to include NONE, for example for empty Cells.

Create a new enum by doing to File -> New File -> Java Enum, and call it Direction. Make sure it is in package com.**snake.ui**
In the Direction enum (Direction.java):

- Add the following values to the enum file:
  - NORTH, EAST, SOUTH, WEST, NONE

You can now close the Direction.java file, as you won't need to edit it again.

2.3 Set the direction of the Snake

This next method will be used by the keyboard adapter which you will copy into your project shortly, to update the direction which the head of the Snake is facing, telling java which way to make the Snake move!

In the Snake class:

- Create a new field of type Direction called dir
  - Remember fields are private variables defined inside the class but outside any methods!
  - Set the value of this field to whichever Direction value makes sense, depending on where you have placed your snake on the board.
    - E.g private Direction dir = Direction.EAST;
- Add a new method called setDirection
  - It will be void and take a Direction parameter:
    - `public void setDirection(Direction dir) {`
  - Inside the method you should set the value of the field this.dir to the parameter dir:
    - this.dir = dir;

2.4 Handle keyboard input

Now it's time to start handling the keyboard input, so when you press keys on the keyboard Java can do something! To do this we use what's known as a KeyListener, the details are beyond the scope of this course, but basically the KeyListener "listens out" for any key presses on the keyboard, and let's you act on the key presses. To save time, and let you get straight into the interesting bits I've included the class SnakeKeyAdapter.java, which you will need to copy and paste into the 'com.snake' package in Netbeans.

However just copying the class into the project isn't where it stops – Java still needs to know to use the listener. The code for adding the listener is shown in the instruction box below.

First, copy the file SnakeKeyAdapter.java into the 'com.snake' package

In the Snake**UI** class (in the com.snake.ui package):

- At the bottom of the SnakeUI constructor method (called SnakeUI):
  - Create a new SnakeKeyAdapter variable, and put a new SnakeKeyAdapter object in it.
  - Call 'addKeyListener(<your variable here>);'
    - Replace '<your variable here>' with the name of your SnakeKeyAdapter variable

2.5 Start the game!

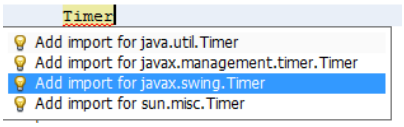
Great stuff! Now we have a class that can manage the keyboard, we just have to start doing things when keys are pressed. If you look in the SnakeKeyAdapter class you'll see that when the enter key is pressed, a method called start() in the Board class is called… This is what we're going to implement next!

In this method we want to do 2 main things:

- Start the game timer – this will allow the Snake to move later on.
- Set the status using the method we made earlier to tell the user that the game has started.

For this to work we need to use a Timer object – this is basically just an object that waits a set amount of time, then 'ticks'. We also need to use another listener (similar to the SnakeKeyAdapter we saw earlier), to 'listen out' for these ticks. I've included the TimerListener.java file for just this purpose.

Important: You will need to import the Timer object. To do this, you can type 'Timer' in Netbeans, and when the red underline appears, press 'ALT+ENTER', and import **'javax.swing.Timer'**



First, copy the file TimerListener.java and GameOverListener.java into the 'com.snake' package

In the **Board** class:

- Create 2 fields
  - A field of type **Timer** called **timer**
  - A field of type **TimerListener** called **timerListener**
- In the start() method:
  - Assign the **timerListener** field to a new TimerListener object
    - The TimerListener constructor takes a Snake object, so you can pass in the **snake** field.
  - Assign the **timer** field to a new **Timer** object
    - The constructor takes 2 parameters, an int (the delay in ms) and a TimerListener object
    - You can use any sensible value for the int (I used 300) – the smaller the number the faster the Snake will move
    - You can use the **timerListener** field for the second parameter
  - Start the timer by calling timer.start();
  - Call the setStatus() method you created earlier, and pass in a String telling the user that the game has started.

2.6 Allow other classes to start/stop the timer

If you remember from previous exercises, we declare fields as private variables defined **inside the class but outside any other methods**. Usually, we would create a public getTimer() method to access the timer, but since there are only 2 things we will want to do with the timer (start it or stop it), we can just create a slightly different method.

Instead, we will just create a setTimer() method that takes a boolean parameter. Inside this method you will create an if/else statement.

In the **Board** class:

- Add a new method called **setTimer**
  - The return type should be **void**
  - It should take a boolean parameter
- Inside the setTimer method:
  - Use an if/else statement to evaluate the Boolean parameter
  - If the parameter is true, call timer.start();
  - Else call timer.stop();

You now have 2 choices, you can either have the game start paused, or have the game start automatically. To have the game started automatically, you don't have to do anything, because in SnakeUI the start() method is called from the main method. If you want the game to start paused, then you will need to remove the call to 'board.start()' in the main method of SnakeUI.

2.7 Add a direction to the Cell class

We're almost ready to start moving the Snake! First though, we want to make each Cell object keep track of its own direction. We will do this by adding a Direction field to the Cell, as well as a get and set method for the Direction field. While we're editing the Cell class, we should add methods to get the row and column from each Cell.

In the **Cell** class:

- Add a new field of type Direction, called dir. Initially, set this to Direction.NONE
- Add a new method called getDirection()
  - It should be public, return type should be a Direction object and take no parameters
  - It should return the dir field
- Add a new method called setDirection()
  - It should be public, return type should be void, and it should take a Direction parameter
  - In the method, set the dir field to the value of the parameter.
    - E.g: this.dir = dir; (if you called the parameter dir as well)
- Create a new method called getRow()
  - It should return an int, and take no parameters
  - Return the value of the row field.
- Now do the same for getCol()

2.8 Set up the Snake class to track its own head, body and tail

When we start moving the Snake, we'll need to know where the head, body and tail are. We will do this by adding a new field for each. However, since the Snake will grow, we need to keep track of multiple body Cells. We've already covered arrays as a way to do this, but a more flexible way is to use what's called a List. A List lets us add Cells to the list, and remove Cells from the list, without having to worry about what index it is in.

In the **Snake** class:

- Add 3 new fields:
  - A Cell field called 'head', and one called 'tail'
  - A List<Cell> field called 'body'
    - You will need to import 'java.util. list', by pressing ALT+ENTER with the cursor on List
    - We also want to set up the array now, so you can use the code below:
    - private List<Cell> body = new ArrayList<Cell>();
- In the place() method, after you have set each of the head, tail and body Cells, also add the Cell object to the relevant field. The example below shows how I did this, by first assigning each Cell to the relevant field (i.e one to head, one to body and one to tail), then using setType on the field, followed by setDirection(dir). An example is shown below

Before:

```
/**
 * Place the Snake on the board
 */
public void place() {
   board.getCell(0, 2).setType(CellType.HEAD);
   board.getCell(0, 1).setType(CellType.BODY);
   board.getCell(0, 0).setType(CellType.TAIL);
}
```

After:

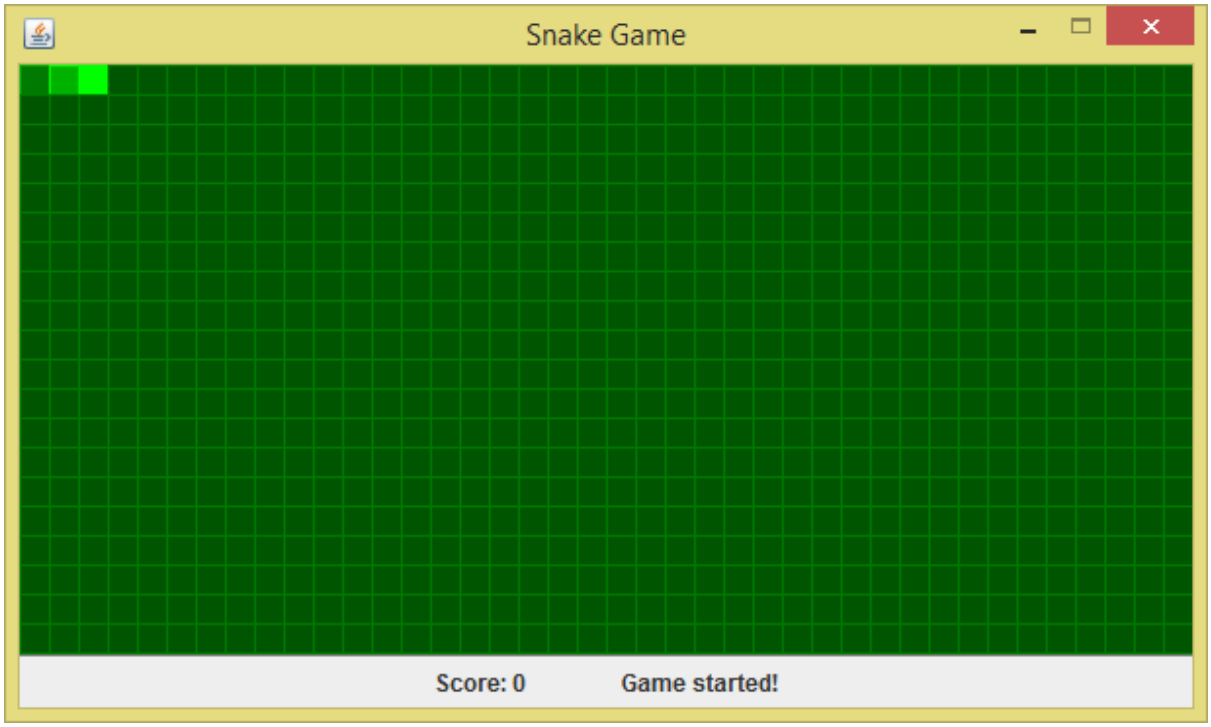
```
/**
 * Place the Snake on the board
 */
public void place() {
   head = board.getCell(0, 2);
   head.setType(CellType.HEAD);
   head.setDirection(dir);

   Cell body = board.getCell(0, 1);
   body.setType(CellType.BODY);
   // Notice we use .add() method to add to the list
   this.body.add(body);
   head.setDirection(dir);

   tail = board.getCell(0, 0);
   tail.setType(CellType.TAIL);
   tail.setDirection(dir);
}
```

2.9 Move the Snake

For now, we will just set the Snake's head to move, and we will cover how to make the whole Snake move in the next worksheet. This is because the full solution will require some more extensive changes. For now, figure out which direction your Snake will be facing, based on where you placed it. For example, the Snake head below (the lightest green square) is currently facing EAST, and make sure the 'dir' field is set to this initially.



To make the Snake move, we will use the value in the 'dir' field. Every time you press one of the arrow keys, the code in the SnakeKeyAdapter class uses the code you wrote earlier (the setDirection() method in the Snake class) to update the value of this 'dir' field. In this task we will create a move() method in the Snake class, and call it from the TimerListener class, so that whenever the game ticks, the Snake moves in the specified direction.

This part is a bit more complex, so here is some pseudocode to help you understand it (pseudocode is basically half way between English and a programming language.

```
Move() method:
        // Store the current row and col values of the head Cell into variables
        // Set the type of the current head cell to EMPTY, and the direction to NONE –
            this resets the current head cell so we don't end up drawing 2 of them!

        // Switch on the direction variable
                // if it's NORTH, decrease row by 1 then break
                // if it's EAST, increase col by 1 then break
                // if it's WEST, decrease col by 1 then break
                // if it's SOUTH, increase row by 1 then break
        // Set the head field to the Cell at the new row and column values
        // Set the type of the Cell to HEAD, and the direction to dir
        // Refresh the display
```

Make sure you understand what's happening in the switching section, particularly why we increase or decrease row or column values at that point. If you don't understand then feel free to ask a friend or teacher.

In the **Snake** class:

- In the move() method:
  - Create a new int variable called row, and use head.getRow() to get the row of the current head cell.
  - Do the same for a col variable, using head.getCol()
  - Call the head.setType() method, passing in CellType.EMPTY
  - Call the head.setDirection() method , passing in Direction.NONE
  - Write a switch statement on the dir field
    - Include cases for NORTH, EAST, SOUTH and WEST
    - Follow the instructions in the pseudocode above to change either the row or column variable in each case
    - Remember to include 'break;' at the end of each case block
  - Now we need to update the head field
    - Set the head field to the result of board.getCell(row, col)
    - Now you can use head.setType() to set the type to CellType.HEAD
    - Do the same with setDirection(), setting it to the value of the dir field
  - Finally, use the following line of code to redraw / refresh the display so you can see the Snake's head move!
    - board.getBoardUI().repaint();

In the **TimerListener** class:

- In the moveSnake() method:
  - Call the move() method on the Snake field

Now it's time to run your program to make sure the Snake head moves around the board, and responds to your key presses. Remember, depending on what you chose to do earlier, you might have to press Enter to start the game.