EECS 2030 Project:

Snake World

Professor: Slawomir Kmiec

Students:

Jason Skinner -- ID: 215115678, YorkU ID: skinner1, Email: jasonskinner65@hotmail.com

Alessa Ivascu -- ID: 214733422, YorkU ID: alexis01, Email: alexis01@my.yorku.ca

Dawood Choksi -- ID: 209092032, YorkU ID: dawoodc, Email: dawoodchoksi@gmail.com

Leroy Nguyen -- ID: 210314870, YorkU ID: leroy25, Email: leroy25@my.yorku.ca

Kristiana Papajani -- ID: 214466387 , YorkU ID: kristi12, Email: papajani.kristiana@gmail.com

Game Instructions

The player controls the snake through the use of the Arrow Keys. The snake is always moving.

The player must eat pizza scattered about the map to increase their score. After eating a slice of pizza, the snake grows in size randomly, adding 1-3 body parts.

Upon level completion, the snake moves faster and the level gets harder.

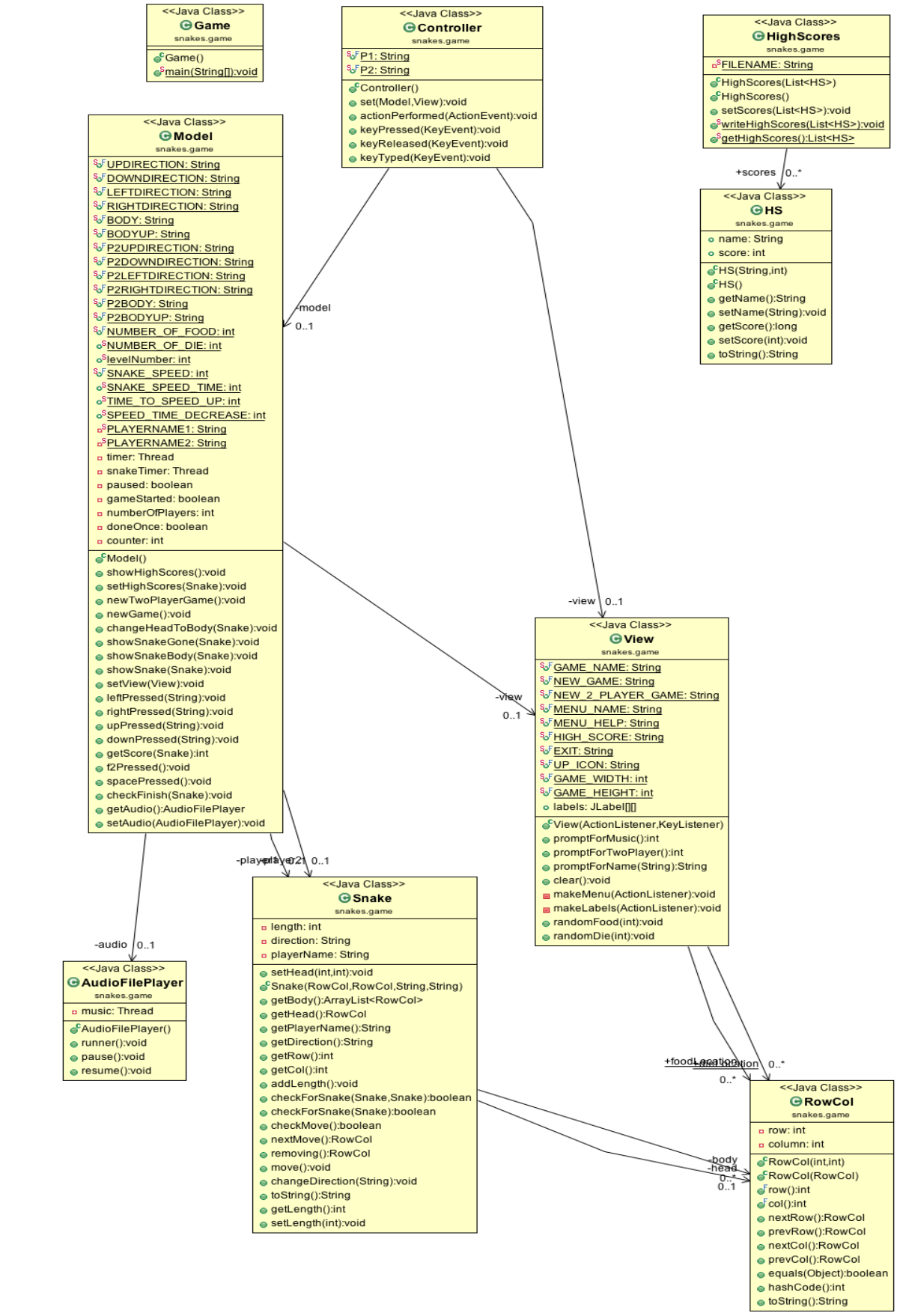
Touching dynamite will end the game. Touching the snake's body will end the game. Touching the boundary will end the game.

A 2nd player can be added through the Multiplayer Game option. This player uses WASD keys to move their snake.

The same rules apply to the 2nd player, but if the 2nd player touches the first player, the game ends.

Press F2 to start a new game. Press Spacebar to pause the game. Finally, enjoy the music by James Brown!

The UML Diagram



Description of Code

- The snake game project consists of many classes that are blueprints for its objects.
- The project consist of the main classes are MVC (**Model**.java, **Controller.**java**,** and **View**.java), which is a software architectural pattern for implementing user interfaces on computers.
- There are other classes that take care of the objects of the game which are: **AudioFilePlayer.java**, **AudioFileSkeleton.java, HS.java**, **RowCol.java**, **getScoreTest.java NetWorking.java, NoNegativesAllowedException.java** and **Snake.java**. Lastly, in order to run and execute the snake game, it requires a PSVM which is found in Game.java. These classes all adhere to the MVC which embodies the objects of the Snake Game.
- The **Model** separates the logic of a program from the rest of the user interface. It manages the data and fundamental behaviours of a program. It is the data and a data-management portion of the program.
    - In our Model class, we have *constant* fields of numbers and strings which are public, private, static, and final.
    - We have an empty Model class constructor which adheres to the MVC pattern.
    - A showHighScores() is constructed, where the Model retrieves the high scores from the HS class and then displays it them into in a pop-up message dialog window.
        - In order to implement the method, it requires a for loop technique.
    - The game includes a multiplayer mode, where it introduces a second player on the left-hand side of the Graphic User Interface (GUI), whereas the first player is on the right-side. Thus the new game is set up for both players.
        - The newTwoPlayerGame() method is responsible for this.
    - The model takes care of the image of the snake as it moves across the game board for both single and multiplayer, as it moves in the game board or background of the GUI. For example, as the snake head moves in a direction forward, the body of the snake follows the previous position of what the snake head was just in. There are two types of images for the snake's body and four types for the snakes head that must be considered depending on the DIRECTION (up, down, right, left) of the snake's movement.
        - Thus, changeHeadToBody(Snake snake) is implemented for this behaviour.
        - The method showSnakeBody(Snake snake) takes care of the movement of the snake's body as it moves around at any direction.

    - At the very start of the game after running Game.java, the player will be immediately prompted and be asked to optionally if he/she wants to play the background music, and as well as choosing their player's usernames. Then after, the game refreshes and sets up for the player(s).
        - This is evident in setView(View view) method.
    - When the snake dies in the game, the game immediately stops and resets when the user wants to try and play again.
        - So the Model calls the showSnakeGone(Snake snake) and erases the snake in the game.
    - So depending on which direction of the snake is moving, it will change it to either left, right, upward or downward direction.
        - The methods: leftPressed(String snake), rightPressed(String snake), upPressed(String snake), and downPressed(String snake) is manages the movements of the player(s) snakes.
- The **Controller** class is an aggregration of a model and a view. It handles logic from the Model and handles the view for the user accordingly. The class essentially is the interaction of the user and is responsible for managing the behavior of the **Model** class
    - In our constructor Controller() inititializes the controller where it has no model and no view – meaning they're set as *null*. However, we set the model and view for the controller in our set(Model model, View view) method.
    - The actionPerformed(ActionEvent e) method is used in order to respond to the user clicking a button or menu item in the view.
    - During the gameplay, we need to use keyboard buttons in order for the user to play and interact with the snake.
        - Hence, the keyPressed(KeyEvent e) allows user to choose which key buttons that will be utilized to control the snake in the game. The method handles the button pressing on the keyboard.
        - Depending which buttons on the keyboard is being pressed, this will only depends on the behavior of the **Model** method.
    - Both keyReleased(KeyEvent e) and keyTyped(KeyEvent arg0) are empty methods that is used for keyRelease and keyTyped events.
- The **View** class handles the GUI; how are you displaying and visually presenting it to the user in the snake game.
    - The View is the child class of the JFrame
    - We have *constant* fields of numbers and strings which are public, private, static, and final.
    - In our constructor View(ActionListener listener, KeyListener keyListener), this constructor creates display of the background game board of the GUI.

- Furthermore, we have window first being displayed, then the labels, listeners and layouts are added and packed so all its contents are set at preferred, optimal size.
- We have other methods prompts the user with dialog boxes with music options, entering second player's name for multiplayer mode, and a dialog when the game is being launched allowing user to enter first player's name.
    - promptforMusic()
    - promptForTwoPlayer()
    - promptForName()
- In our clear() method in the View class, everything in the GUI is being cleared and the game is resets; the snake returns back to its initial position with food/death items are cleared, as well as the usernames which are reset.
    - In the View, we needed to create a menu bar with two options labeled as GAME and HELP (see more in the javadoc comment)
    - makeMenu(ActionListener listener) is responsible for this
- In addition, the game also include buttons for the View, where the action command should be unison to the text of the button label, so the user(s) can visually understand and comprehend the text of the label before pressing it.
    - makeLabels(ActionListener listener) takes care of this.

  Lastly we need to create random food across the background of our GUI when the snake eats it (when the snake traverses it). When the food item is eaten, the snake's body grows per every image of the body. Food units are added random places around the map grid, and only 1 is shown to the user, and a new one is generated at random spots after the previous is eaten. This goes the same for death items which represents as TNT candle bomb.
    - randomFood(int max) takes care of food items.
    - randomDie(int max) takes care of death items

- The **RowCol** class deals with the creation of row and column which simulate coordinates of an object representing a 2D Cartesian plane.
    - We have them in private attributes
    - The parameterized constructor takes and sets attributes of an instance of the RowCol class
        - RowCol(int row, int col)
    - We also have a copy constructor which takes another RowCol object's attributes and copies
- The **AudioFilePlayer** class is what we need to input the music of the background when user optionally chooses it.

    - The **AudioFileSkeleton** class, is an abstract class which provides a skeleton for the AudioFilePlayer class. This contains only an abstract method resume, that will be implemented later in the AudioFilePlayer class.
- The **getScoreTest** class handles all the JUnit cases for all possibilities so that methods are working appropriately for the snake game.
    - We have used regular standard testing with positive integers
    - We have used negative boundary methods with negative values and exceptions for various possibilities.
    - We also have used upper boundary method to test this.
- The **NetWorking** class is used to implement networking functionality to the Snake game.
- The **NoNegativesAllowedException** class handles unchecked exception when handling unusual cases of negative values.
- In the **Snake** class takes care of everything related to the snake itself. Its body, movement, addition to its body after it eats, as well as its direction and its location on the board. Everything about a snake object is written here.
    - we have all our fields/attributes **private** (i.e; private int length, private RowCol head, private ArrayList<RowCol>…etc.) and getters which are being accessed by the Model class.
    - The methods that make up the snake, is all accessing from its child class **RowCol**.
- The **HS** class is a generic class which stores a name and a number representing the player and the score the player has respectively.

Application of Techniques Learned

The following techniques are used in multiple locations throughout the project's code, however, only some specific instances are highlighted for the purposes of the project's requirements.

All 12 **Basic Techniques** are used in the project, as well as all the **Advanced Techniques**.

1) Encapsulation / Information Hiding:

- Used throughout all classes in the project.
- For example, the class Public Static HighScores class inside the Model class; the fields are marked as private, as well as every constructor and non-getter method.
- The code in this class is therefore hidden and cannot be accessed randomly.

2) Overloading / Constructors:

- Overloading occurs in many places in the code throughout this project.
- For example, in the RowCol class, the constructors are overloaded.
- One takes in an integer value for the number of rows and columns and creates an object, setting the objects number of rows and columns according to the numbers inputted.
- The second constructor for the RowCol class takes in another RowCol object as a parameter and copies its number of rows and columns and creates a new object using those numbers to set the number of rows and columns.
- An example of overloaded methods is in the Snake class, and specifically the checkForSnake methods.
- One method accepts a single snake object as a parameter, representing a player, and checks to see if the snake collides with itself.
- The other method of checkForSnake accepts 2 snake objects for parameters, each representing different players, and checks to see if the snakes collide with each other.

3) Static Methods and Static Variables:

- Static Methods and Static Variables are used throughout the project, namely to keep a counter or instance of a single object, or for allowing certain methods to not require an object to perform their function.
- The class Public Static HighScores inside the Model Class is an example of a class using static methods and static variables.

- All the fields are static, such as the field maintaining the highscores.txt file, as only 1 instance is needed for all objects of HighScores.
- The methods are static and belong to the HighScores class, and do not act on objects/instances of HighScore.

4) Mutable and Immutable Classes:

- Most classes in this project are mutable for the purposes of functionality.
- An example of a mutable class is the Snake class. The methods and constructors are all Public, and specifically the settors are public as well, thus Snake objects can have its values changed. The Snake's length and initial head position settors are Public, and as such, can be changed to different lengths or initial positions.
- An example of an immutable class is the class Public Static HighScores inside the Model class; all fields are marked private and most methods are private, including the settors. Only the getter is public, and since it is coded properly, it will not have access to HighScores objects. The class itself is also static and thusly independent of the Model class.
- A common example of the use of immutable classes is the popular String class, which is repeatedly used within the project code.

5) Inner Classes:

- Inner classes are used throughout the project, though they may not be widely noticeable as they're usually anonymous classes inside of other classes.
- For example, in the Model class, inside the newGame method, there is a runnable stream that creates an anonymous inner class for the Runnable object to create the timer stream.
- A more apparent and obvious inner class is the public Static HighScores class inside the Model class.

6) Interfaces/Abstract Classes:

- Interfaces are used in a few locations in this project.
- One of the key uses of interfaces is in the Controller class.
- The Controller class implements the ActionListener and KeyListener interfaces and their associated methods.
- The ActionListener interface allows for methods that "listen" for user inputs such as mouse clicks, and thus those clicks can then be used to perform something, such as clicking on a menu option would open a dialog box.
- The KeyListener interface allows for methods to "listen" for user inputs from the keyboard, and thus those key presses can be used as parameters for methods to perform a function when a key is pressed, such as moving the snake for a single player or 2 players depending on the key pressed.

- An example of an abstract class is the AudioFileSkeleton class. It contains only 1 abstract method called 'resume' and this class is inherited by the AudioFilePlayer class.

7) Inheritance:

- A crucial example of inheritance in this project is in the View.
- The public class View inherits from the JFrame class; all of JFrame's methods and attributes are available to the View and thusly to the project.
- Many of the things in JFrame are needed for Swing and to construct valuable features of the Snake game.
- For example, the entire Snake game window is constructed using various JFrame methods like setText and setting dimensions of the window itself. Other things like setting the Menu bar using JMenuBar and JMenuItems lets us implement menus for the Snake game. Even simple things like JFrame.EXIT_ON_CLOSE lets the application terminate when the application is closed, like when the "X" in the top right of the game's window is clicked.
- Another example of inheritance is the use of the NoNegativesAllowedException. It is a child class of Runtime exception, so it is also an unchecked exception. It is used within the getScore() method, because it is an unusual situation if the score, the snake's length, or the counter is in the negatives.

8) Polymorphism:

- Polymorphism is used in many places throughout this project.
- A simple example of the use of polymorphism is in the RowCol class.
- At the very end of the class, there is a method named public String toString(). Since a toString() method is defined, the objects of RowCol use this toString() method, and not the toString() method from the Object class. This occurs due to Late Binding.
- Another example of polymorphism is within the HighScores class. One if its fields, *scores*, has its declared class type List, however its actual type is ArrayList. This gives it access to List's methods, however it actually uses ArrayList's overridden methods for its purposes due to dynamic binding.

9) Generics:

- Generics are used throughout the project in terms of ArrayLists. However, a specific example of a Generic class is the HS class.
- This is a generic class with 2 type parameters. All the constructors and methods are coded in a way such that the 2 types, S and N, can be any type.
- In the Model, the types for this Generic class is then defined as needed. In our project, we require the high scores to be of the format where the first type is a String and the second type is an Integer.

- S and N are used as the names of the types to provide some meaning to the variable names; the S represents a shortened form of String, while N represents a shortened form of Number. It is important to note these are not always String and numbers/Integers, as the class is a generic, but these variable names help remember the order that a high score should appear in (String, Number/Integer).

10) Swing/GUI/Event-Driven Programming:

- Swing/GUI is used to make the actual window and display the game.
- All of the "drawing" of the game is handled in the class called View.
- For example, the constructor of the view uses many Swing/GUI coding commands like making the window itself, making the window resizable, setting the initial background (which is later changed), setting the Snake image, making menus, packing it all together, etc.
- The key part of the View is the Event-Driven programming which occurs in the constructor which accepts parameters of type ActionListener and KeyListener.
- The ActionListener allows for methods that "listen" for user inputs such as mouse clicks, and thus those clicks can then be used to perform something, such as clicking on a menu option would open a dialog box.
- The KeyListener allows for methods to "listen" for user inputs from the keyboard, and thus those key presses can be used as parameters for methods to perform a function when a key is pressed, such as moving the snake for a single player or 2 players depending on the key pressed.
- There are methods that also accept these Listeners are parameters, such as the makeMenu method which "listens" for an action such as a mouse click and then generates the appropriate menu depending on which button is clicked.
- The advantage of using ActionListener and KeyListener is that it helps with Event-Driven Programming; the methods that listen for these actions and key presses only execute when an action or key press is detected, and thus are not performing all the time.

11) Array and ArrayList Collections:

- ArrayList collections are used throughout the project.
- An example of an ArrayList is in the Snake class, where the private attribute 'body' is an ArrayList Collection.
- This is useful as the 'body' ArrayList keeps track of the number of body elements of the snake; as the snake eats food units, the 'body' ArrayList increases in length between 1-3 units randomly. Therefore, an ArrayList, and not a simple array with a defined length, is useful here.

- Another example of an ArrayList is in the HighScores inner class of the Model class. The HighScores class has an ArrayList called ‘scores’ that is used to contain all the high scores.
- Arrays are seldom used in our project. A simple example is in the View, in the randomFood method. Here, a simple Array containing the ‘x, y’ position is used for the food unit’s placement. The array is a simple 2D array and its elements are calculated randomly using the game’s width and height.

12) Exceptions and File I/O:

- Exceptions are used throughout our project.
- An example of an exception used multiple times is the InterruptedException, which is used when a thread is interrupted rather put to sleep.
- This occurs in our Model many times, but a key place is in the newGame method, where the Timer stream is coded. A Try/Catch block is used around the code of the Timer, and if the timer thread is interrupted instead of sleeping, the Catch block will catch the InterruptedException and alert the user.
- A Throwable exception is used in the last method, getLines, of the inner class HighScores inside the Model class. The method will throw an IOException to the user if there is a problem with the input/output in the reading/returning of highscores.
- In the inner class HighScores inside the Model Class, the high scores are stored in an ArrayList and written to a text file. The high scores are also read off the text file and shown to the user. This all occurs in the writeHighScores method, getHighScores method, and getLines method.

Application of ADVANCED Techniques Learned

In this section, most of the advanced techniques and how they're used in the code of this project is discussed.

1) Recursion:

- An example of Recursion is in the public static HighScores class inside the Model class. The very last method, getLines(), functions recursively in its execution.
- The getLines() method reads the lines of a file one by one, and adds them to an ArrayList. When there are no lines left to read, the method returns the ArrayList.
- This follows the guidelines for proper use of Recursion:
    - The method getLines calls itself.
    - The method getLines has a base case; the base case is 0 lines left to read.
    - The method getLines is always getting closer to the base/shrinking; the number of lines to read starts at the max and gets to 0 eventually.

2) Model/View/Controller:

- The Model View Controller classes are used in this project.
- The Model handles the majority of "data" or "functionality" code, such as making a new game and its timer, handling high scores, handling multiplayer, etc.
- The View handles the "drawing" of the Snake game and the Swing/GUI elements.
- The Controller connects the Model and View together. Things like key presses are handled in here, where the key press is detected by the View and "sent" to the model to update the Snake on what should happen after the key press, and then "sent" back to the view to update the Snake/game to show what happens after the key is pressed.

3) Functional Programming:

- A lambda function is used to demonstrate Functional Programming.
- The lambda function is used in the Model class of the project, in the setHighScores method, around line 150. The Object 'scores' has it's high scores (integer values) compared and sorted.

4) Multithreading:

- Multithreading is used in a few places in the code for the project.
- An example of multithreading is used in the coding of the Timer. This happens in the Model class, in the newGame() method.

- A new thread is created after the new Runnable is created, which is called 'time'. Thus, a 2nd thread is created here to handle the Timer shown in the Snake game window.

5) Networking and Sockets:

- The Networking for Multiplayer is handled in the Networking class.
- First a stream for the Networking portion is created.
- Then a network listener is created by making a new Runnable object. In here, a new serverSocket object is created, which listens to an incoming client/other player. A clientSocket is also created, which "talks"/sends information from the client.
- All the received information from the sockets is converted into a string.
- This string output is then sent to the browser/displayed to the user.
- Afterward, all the open sockets are closed, ex. Web streams, client and server sockets.
- All of the networking information, aka web input/output, is logged to a text file.
- The networking is all done in a separate multithread.

ADVANCED Features

We added 2 advanced features to the game:

1) Upon a win/level completion, the user is presented with a new and more challenging board and the game continues after a nice and short message. The moment the user clicks OK, the game resumes on the new level, with a faster snake and more dangerous board. Be careful!
2) You can connect to another player over the network and then you can play against each other.

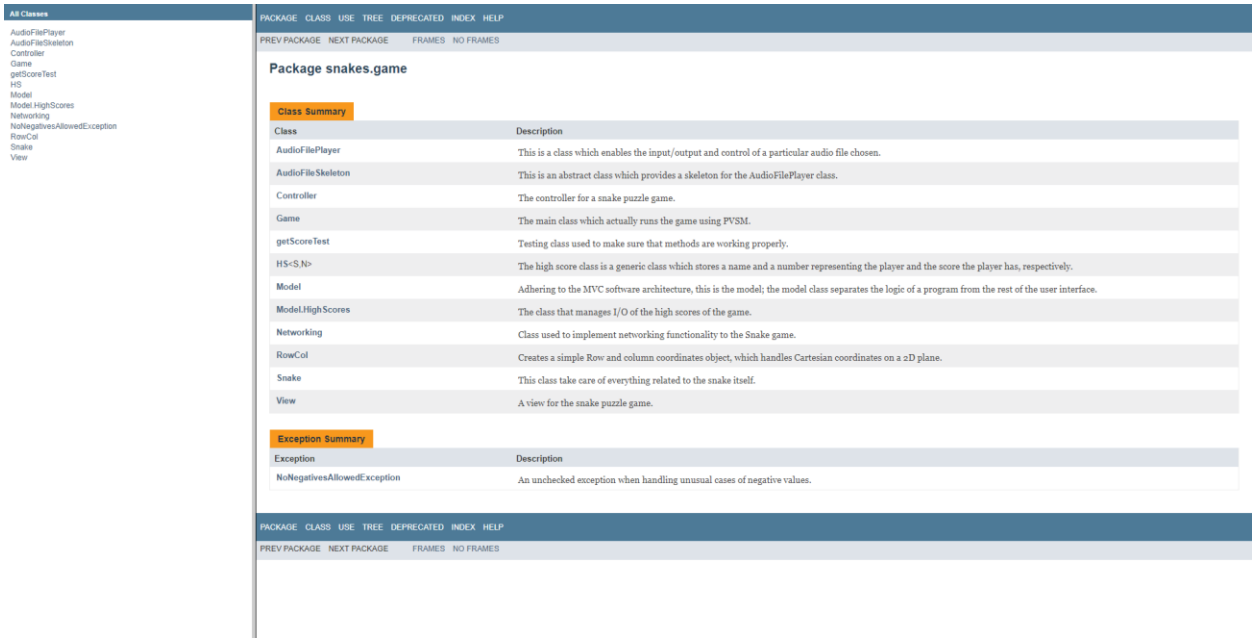
JUnit Testing Strategy

- There was not much pure calculations in code, so little testing was required. There were few "units" of work on code since everything was closely linked together and needed the instantiation of the model, the view, or the controller, as well as potentially other classes which needed an instance to call any calculating methods.

- However, testing was still done, and an example of this is found in the getScoreTest class. In this class, we test whether the final score of the game was calculated correctly. As per appropriate testing protocol, tests were done on standard cases, cases near the boundary, and cases past both the upper and lower boundaries.

Javadoc
(also attached inside the project folder)

Snippets of the JavaDoc:



All Classes

AudioFilePlayer
AudioFileSkeleton
Controller
Game
getScoreTest
HS
Model
Model.HighScores
Networking
NoNegativesAllowedException
RowCol
Snake
View

PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV PACKAGE NEXT PACKAGE FRAMES NO FRAMES

Package snakes.game

Class Summary

Class	Description
AudioFilePlayer	This is a class which enables the input/output and control of a particular audio file chosen.
AudioFileSkeleton	This is an abstract class which provides a skeleton for the AudioFilePlayer class.
Controller	The controller for a snake puzzle game.
Game	The main class which actually runs the game using PVSM.
getScoreTest	Testing class used to make sure that methods are working properly.
HS<S,N>	The high score class is a generic class which stores a name and a number representing the player and the score the player has, respectively.
Model	Adhering to the MVC software architecture, this is the model; the model class separates the logic of a program from the rest of the user interface.
Model.HighScores	The class that manages I/O of the high scores of the game.
Networking	Class used to implement networking functionality to the Snake game.
RowCol	Creates a simple Row and column coordinates object, which handles Cartesian coordinates on a 2D plane.
Snake	This class take care of everything related to the snake itself.
View	A view for the snake puzzle game.

Exception Summary

Exception	Description
NoNegativesAllowedException	An unchecked exception when handling unusual cases of negative values.

PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV PACKAGE NEXT PACKAGE FRAMES NO FRAMES

snakes.game

Class AudioFilePlayer

java.lang.Object
  snakes.game.AudioFileSkeleton
    snakes.game.AudioFilePlayer

```
public class AudioFilePlayer
extends AudioFileSkeleton
```

This is a class which enables the input/output and control of a particular audio file chosen.

Author:
Jason Skinner, Leroy Nguyen, Dawood Choksi, Alessa Ivascu, Kristiana Papajani

Constructor Summary

Constructors

Constructor and Description
`AudioFilePlayer()` The AudioFilePlayer constructor always calls upon the runner method, to handle the running of the audio file.

Method Summary

All Methods | **Static Methods** | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
`void`	`pause()` This is a void method which makes it possible to pause the music.
`void`	`resume()` This method resumes the music after it has been paused The notify method is called whenever the music has been resumed
`static void`	`runner()` This is a static method which has an anonymous inner class inside of it.

Methods inherited from class java.lang.Object

`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Methods inherited from class java.lang.Object

`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

AudioFilePlayer

```
public AudioFilePlayer()
```

The AudioFilePlayer constructor always calls upon the runner method, to handle the running of the audio file.

Method Detail

runner

```
public static void runner()
```

This is a static method which has an anonymous inner class inside of it. Inside of it is all the code which gets the clip/music and it plays it continuously

pause

```
public void pause()
          throws java.lang.InterruptedException
```

This is a void method which makes it possible to pause the music.

Throws:
`java.lang.InterruptedException - when a thread is waiting or sleeping and another thread interrupts it using the interrupt method in class Thread. The method wait is called whenever the music is interrupted`

resume

```
public void resume()
```

This method resumes the music after it has been paused The notify method is called whenever the music has been resumed

Specified by:
`resume in class AudioFileSkeleton`

snakes.game

Class AudioFileSkeleton

java.lang.Object
  snakes.game.AudioFileSkeleton

Direct Known Subclasses:
AudioFilePlayer

```
public abstract class AudioFileSkeleton
extends java.lang.Object
```

This is an abstract class which provides a skeleton for the AudioFilePlayer class. This contains only an abstract method resume, that will be implemented later in the AudioFilePlayer class.

Author:
Jason Skinner, Leroy Nguyen, Dawood Choksi, Alessa Ivascu, Kristiana Papajani

Constructor Summary

Constructors

Constructor and Description
AudioFileSkeleton()

Method Summary

All Methods | **Instance Methods** | **Abstract Methods**

Modifier and Type	Method and Description
abstract void	resume() An abstract method resume, used by the AudioFilePlayer class, and filled in with the implementation in that class.

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

AudioFileSkeleton

```
public AudioFileSkeleton()
```

All Classes

AudioFilePlayer
AudioFileSkeleton
Controller
Game
getScoreTest
HS
Model
Model.HighScores
Networking
NoNegativesAllowedException
RowCol
Snake
View

Jason Skinner, Leroy Nguyen, Dawood Choksi, Alessa Ivascu, Kristiana Papajani

Constructor Summary

Constructors

Constructor and Description
AudioFileSkeleton()

Method Summary

All Methods | **Instance Methods** | **Abstract Methods**

Modifier and Type	Method and Description
abstract void	resume() An abstract method resume, used by the AudioFilePlayer class, and filled in with the implementation in that class.

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

AudioFileSkeleton

```
public AudioFileSkeleton()
```

Method Detail

resume

```
public abstract void resume()
```

An abstract method resume, used by the AudioFilePlayer class, and filled in with the implementation in that class.

snakes.game

Class Controller

java.lang.Object
  snakes.game.Controller

All Implemented Interfaces:

`java.awt.event.ActionListener, java.awt.event.KeyListener, java.util.EventListener`

```
public class Controller
extends java.lang.Object
implements java.awt.event.ActionListener, java.awt.event.KeyListener
```

The controller for a snake puzzle game. The controller is an aggregation of a model and a view. It handles logic figured out from the Model and changes the view for the user accordingly.

Author:

Jason Skinner, Leroy Nguyen, Dawood Choksi, Alessa Ivascu, Kristiana Papajani

Field Summary

Fields

Modifier and Type	Field and Description
`static java.lang.String`	`P1`
`static java.lang.String`	`P2`

Constructor Summary

Constructors

Constructor and Description
`Controller()` Initializes the controller so that it has no model and no view.

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description

All Classes

AudioFilePlayer
AudioFileSkeleton
Controller
Game
getScoreTest
HS
Model
Model.HighScores
Networking
NoNegativesAllowedException
RowCol
Snake
View

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
`void`	`actionPerformed(java.awt.event.ActionEvent e)` This is an implementation of the method from the ActionListener interface.
`void`	`keyPressed(java.awt.event.KeyEvent e)`
`void`	`keyReleased(java.awt.event.KeyEvent e)`
`void`	`keyTyped(java.awt.event.KeyEvent arg0)`
`void`	`set(Model model, View view)` Set the model and view for this controller.

Methods inherited from class java.lang.Object

`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Field Detail

P1

```
public static final java.lang.String P1
```

See Also:

`Constant Field Values`

P2

```
public static final java.lang.String P2
```

See Also:

`Constant Field Values`

Constructor Detail

Controller

```
public Controller()
```

Initializes the controller so that it has no model and no view.

Constructor Detail

Controller

```
public Controller()
```

Initializes the controller so that it has no model and no view.

Method Detail

set

```
public void set(Model model,
                View view)
```

Set the model and view for this controller.

Parameters:

`model` - the model

`view` - the view

actionPerformed

```
public void actionPerformed(java.awt.event.ActionEvent e)
```

This is an implementation of the method from the ActionListener interface. It is used in order to respond to the user clicking a button or menu item in the view.

Specified by:

`actionPerformed` in interface `java.awt.event.ActionListener`

Parameters:

`e` - the action event to respond to.

keyPressed

```
public void keyPressed(java.awt.event.KeyEvent e)
```

Specified by:

`keyPressed` in interface `java.awt.event.KeyListener`

keyReleased

```
public void keyReleased(java.awt.event.KeyEvent e)
```

Specified by:
Set the model and view for this controller.

Parameters:

`model` - the model

`view` - the view

actionPerformed

```
public void actionPerformed(java.awt.event.ActionEvent e)
```

This is an implementation of the method from the ActionListener interface. It is used in order to respond to the user clicking a button or menu item in the view.

Specified by:

`actionPerformed` in interface `java.awt.event.ActionListener`

Parameters:

`e` - the action event to respond to.

keyPressed

```
public void keyPressed(java.awt.event.KeyEvent e)
```

Specified by:

`keyPressed` in interface `java.awt.event.KeyListener`

keyReleased

```
public void keyReleased(java.awt.event.KeyEvent e)
```

Specified by:

`keyReleased` in interface `java.awt.event.KeyListener`

keyTyped

```
public void keyTyped(java.awt.event.KeyEvent arg0)
```

Specified by:

`keyTyped` in interface `java.awt.event.KeyListener`

snakes.game

Class Game

java.lang.Object
  snakes.game.Game

```
public class Game
extends java.lang.Object
```

The main class which actually runs the game using PVSM.

Author:
Jason Skinner, Leroy Nguyen, Dawood Choksi, Alessa Ivascu, Kristiana Papajani

Constructor Summary

Constructors

Constructor and Description
Game()

Method Summary

All Methods | Static Methods | Concrete Methods

Modifier and Type	Method and Description
static void	main(java.lang.String[] args) The main method which handles the execution of the snake game.

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Game

```
public Game()
```

Method Detail

All Classes

AudioFilePlayer
AudioFileSkeleton
Controller
Game
getScoreTest
HS
Model
Model.HighScores
Networking
NoNegativesAllowedException
RowCol
Snake
View

Constructors

Constructor and Description
Game()

Method Summary

All Methods | Static Methods | Concrete Methods

Modifier and Type	Method and Description
static void	main(java.lang.String[] args) The main method which handles the execution of the snake game.

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Game

```
public Game()
```

Method Detail

main

```
public static void main(java.lang.String[] args)
```

The main method which handles the execution of the snake game. Here, the model, view, and controller are all put together to handle the running of the game itself.

Parameters:
args - default expression

All Classes

AudioFilePlayer
AudioFileSkeleton
Controller
Game
getScoreTest
HS
Model
Model.HighScores
Networking
NoNegativesAllowedException
RowCol
Snake
View

snakes.game

Class getScoreTest

java.lang.Object
  snakes.game.getScoreTest

public class **getScoreTest**
extends java.lang.Object

Testing class used to make sure that methods are working properly.

Author:
Jason Skinner, Leroy Nguyen, Dawood Choksi, Alessa Ivascu, Kristiana Papajani

Field Summary

Fields

Modifier and Type	Field and Description
org.junit.rules.ExpectedException	expectedEx

Constructor Summary

Constructors

Constructor and Description
getScoreTest()

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
void	test_Big_Numbers()
void	test_Boundary_Lower_No_Negatives()
void	test_Close_To_Boundary_Lower()
void	test_Negative_Exception_Both_Values()
void	test_Negative_Exception_Counter()
void	test_Negative_Exception_Snake_Length()
void	test_Negative_Exception()
void	test_Standard_2()

All Classes

AudioFilePlayer
AudioFileSkeleton
Controller
Game
getScoreTest
HS
Model
Model.HighScores
Networking
NoNegativesAllowedException
RowCol
Snake
View

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
void	test_Big_Numbers()
void	test_Boundary_Lower_No_Negatives()
void	test_Close_To_Boundary_Lower()
void	test_Negative_Exception_Both_Values()
void	test_Negative_Exception_Counter()
void	test_Negative_Exception_Snake_Length()
void	test_Negative_Exception()
void	test_Standard_2()
void	test_Standard()
void	test_Upper_Boundary()

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

expectedEx

```
public org.junit.rules.ExpectedException expectedEx
```

Constructor Detail

getScoreTest

```
public getScoreTest()
```

Method Detail

test_Standard

```
public void test_Standard()
```

Method Detail

test_Standard

```
public void test_Standard()
```

test_Big_Numbers

```
public void test_Big_Numbers()
```

test_Boundary_Lower_No_Negatives

```
public void test_Boundary_Lower_No_Negatives()
```

test_Close_To_Boundary_Lower

```
public void test_Close_To_Boundary_Lower()
```

test_Standard_2

```
public void test_Standard_2()
```

test_Negative_Exception

```
public void test_Negative_Exception()
```

test_Negative_Exception_Snake_Length

```
public void test_Negative_Exception_Snake_Length()
```

test_Negative_Exception_Counter

```
public void test_Negative_Exception_Counter()
```

test_Negative_Exception_Both_Values

```
public void test_Negative_Exception_Both_Values()
```

All Classes

AudioFilePlayer
AudioFileSkeleton
Controller
Game
getScoreTest
HS
Model
Model.HighScores
Networking
NoNegativesAllowedException
RowCol
Snake
View

test_Boundary_Lower_No_Negatives

```
public void test_Boundary_Lower_No_Negatives()
```

test_Close_To_Boundary_Lower

```
public void test_Close_To_Boundary_Lower()
```

test_Standard_2

```
public void test_Standard_2()
```

test_Negative_Exception

```
public void test_Negative_Exception()
```

test_Negative_Exception_Snake_Length

```
public void test_Negative_Exception_Snake_Length()
```

test_Negative_Exception_Counter

```
public void test_Negative_Exception_Counter()
```

test_Negative_Exception_Both_Values

```
public void test_Negative_Exception_Both_Values()
```

test_Upper_Boundary

```
public void test_Upper_Boundary()
```

All Classes

AudioFilePlayer
AudioFileSkeleton
Controller
Game
getScoreTest
HS
Model
Model.HighScores
Networking
NoNegativesAllowedException
RowCol
Snake
View

Class HS<S,N>

java.lang.Object
  snakes.game.HS<S,N>

Type Parameters:

`S` - the name of the player

`N` - the score of the player

```
public class HS<S,N>
extends java.lang.Object
```

The high score class is a generic class which stores a name and a number representing the player and the score the player has, respectively.

Author:

Jason Skinner, Leroy Nguyen, Dawood Choksi, Alessa Ivascu, Kristiana Papajani

Field Summary

Fields

Modifier and Type	Field and Description
`S`	`name`
`N`	`score`

Constructor Summary

Constructors

Constructor and Description
`HS()` EXAMPLE OF OVERLOADED CONSTRUCTOR This is a default constructor of the HS class.
`HS(S name, N score)` EXAMPLE OF OVERLOADED CONSTRUCTOR This constructor takes in a name and a score which are held as attributes within the instance of the class.

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
`S`	`getName()` This getter method returns the name of the player.

All Classes

AudioFilePlayer
AudioFileSkeleton
Controller
Game
getScoreTest
HS
Model
Model.HighScores
Networking
NoNegativesAllowedException
RowCol
Snake
View

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
`S`	`getName()` This getter method returns the name of the player.
`N`	`getScore()` The getter method which returns the score of a player.
`void`	`setName(S name)` This setter method sets the name of the player.
`void`	`setScore(N score)` A setter method which sets the score of a player
`java.lang.String`	`toString()`

Methods inherited from class java.lang.Object

`equals, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Field Detail

name

```
public S name
```

score

```
public N score
```

Constructor Detail

HS

```
public HS(S name,
          N score)
```

EXAMPLE OF OVERLOADED CONSTRUCTOR This constructor takes in a name and a score which are held as attributes within the instance of the class.

Parameters:

`name` - is the players name

Constructor Detail

HS

```
public HS(S name,
          N score)
```

EXAMPLE OF OVERLOADED CONSTRUCTOR This constructor takes in a name and a score which are held as attributes within the instance of the class.

Parameters:

`name` - is the players name

`score` - is the score that will be tied to the specific player

HS

```
public HS()
```

EXAMPLE OF OVERLOADED CONSTRUCTOR This is a default constructor of the HS class. Fields name and score are set to their default values.

Method Detail

getName

```
public S getName()
```

This getter method returns the name of the player.

Returns:

`name` the name of the player

setName

```
public void setName(S name)
```

This setter method sets the name of the player.

Parameters:

`name` - the new name of the player.

getScore

```
public N getScore()
```

The getter method which returns the score of a player.

This getter method returns the name of the player.

Returns:

`name` the name of the player

All Classes

AudioFilePlayer
AudioFileSkeleton
Controller
Game
getScoreTest
HS
Model
Model.HighScores
Networking
NoNegativesAllowedException
RowCol
Snake
View

setName

```
public void setName(S name)
```

This setter method sets the name of the player.

Parameters:

`name` - the new name of the player.

getScore

```
public N getScore()
```

The getter method which returns the score of a player.

Returns:

`score` the score of a player

setScore

```
public void setScore(N score)
```

A setter method which sets the score of a player

Parameters:

`score` - the new score of the player

toString

```
public java.lang.String toString()
```

Overrides:

`toString` in class `java.lang.Object`

All Classes

AudioFilePlayer
AudioFileSkeleton
Controller
Game
getScoreTest
HS
Model
Model.HighScores
Networking
NoNegativesAllowedException
RowCol
Snake
View

Class Model

java.lang.Object
  snakes.game.Model

```
public class Model
extends java.lang.Object
```

Adhering to the MVC software architecture, this is the model; the model class separates the logic of a program from the rest of the user interface. It manages the data and fundamental behaviors of a program. it is the data and data-management portion of the program. Specifically applying to this instance, the model takes care of the backbone of the rest of the game. It puts together all of the logic which handles the highscores, movement of the snake game, as well as resetting and setting up the game board, to name a few.

Author:

Jason Skinner, Leroy Nguyen, Dawood Choksi, Alessa Ivascu, Kristiana Papajani

Nested Class Summary

Nested Classes

Modifier and Type	Class and Description
static class	Model.HighScores The class that manages I/O of the high scores of the game.

Field Summary

Fields

Modifier and Type	Field and Description
static java.lang.String	BODY
static java.lang.String	BODYUP
static java.lang.String	DOWNDIRECTION
static java.lang.String	LEFTDIRECTION
static int	levelNumber
static int	NUMBER_OF_DIE
static int	NUMBER_OF_FOOD
static java.lang.String	P2BODY
static java.lang.String	P2BODYUP
static java.lang.String	P2DOWNDIRECTION
static java.lang.String	P2LEFTDIRECTION
static java.lang.String	P2RIGHTDIRECTION

All Classes

AudioFilePlayer
AudioFileSkeleton
Controller
Game
getScoreTest
HS
Model
Model.HighScores
Networking
NoNegativesAllowedException
RowCol
Snake
View

Field Summary

Fields

Modifier and Type	Field and Description
static java.lang.String	BODY
static java.lang.String	BODYUP
static java.lang.String	DOWNDIRECTION
static java.lang.String	LEFTDIRECTION
static int	levelNumber
static int	NUMBER_OF_DIE
static int	NUMBER_OF_FOOD
static java.lang.String	P2BODY
static java.lang.String	P2BODYUP
static java.lang.String	P2DOWNDIRECTION
static java.lang.String	P2LEFTDIRECTION
static java.lang.String	P2RIGHTDIRECTION
static java.lang.String	P2UPDIRECTION
static java.lang.String	RIGHTDIRECTION
static int	SNAKE_SPEED
static int	SNAKE_SPEED_TIME
static int	SPEED_TIME_DECREASE
static int	TIME_TO_SPEED_UP
static java.lang.String	UPDIRECTION

Constructor Summary

Constructors

Constructor and Description
Model() This is an empty Model class constructor.

Method Summary

All Methods | Instance Methods | Concrete Methods

All Classes

AudioFilePlayer
AudioFileSkeleton
Controller
Game
getScoreTest
HS
Model
Model.HighScores
Networking
NoNegativesAllowedException
RowCol
Snake
View

Method Summary

All Methods | Instance Methods | Concrete Methods

Modifier and Type	Method and Description
void	changeHeadToBody(Snake snake) This method takes care of the image of the snake as it moves across the game board.
void	checkFinish(Snake snake) This method checks where the snake is on the game board and acts according to its status.
void	downPressed(java.lang.String snake) Manages the movement of the players' snakes.
void	f2Pressed() This is the button to start a new game.
AudioFilePlayer	getAudio() This is a getter method for the audio file for the game.
int	getCounter() A getter method for the counter field.
int	getScore(Snake snake) Gets the score for the snake object.
void	leftPressed(java.lang.String snake) Manages the movement of the players' snakes.
void	newGame() This method sets up the new game, first erasing everything in the game board, then going back to level one, setting all of the settings back to their default original, and setting the players back to their original positions.
void	newTwoPlayerGame() This starts and sets up the game for two players.
void	resetNumberOfPlayers()
void	rightPressed(java.lang.String snake) Manages the movement of the players' snakes.
void	setAudio(AudioFilePlayer audio) This is a setter method for the audio file for the game.
void	setCounter(int newCount) A setter method for the game's time, counter.
void	setHighScores(Snake snake) This sets the high scores which are found in a file, and sorts them from highest score to lowest score.
void	setView(View view) This sets up the game board for the player.
void	showSnakeBody(Snake snake) This method takes care of the movement of the snake's body as it moves across the game board.
void	showSnakeGone(Snake snake) If the snake is dead or for some reason the snake needs to be taken off the game board, this method erases it.
void	spacePressed() This is the pause button for the game.
void	upPressed(java.lang.String snake) Manages the movement of the players' snakes.

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

UPDIRECTION

```
public static final java.lang.String UPDIRECTION
```

See Also:
Constant Field Values

DOWNDIRECTION

```
public static final java.lang.String DOWNDIRECTION
```

See Also:
Constant Field Values

LEFTDIRECTION

```
public static final java.lang.String LEFTDIRECTION
```

See Also:
Constant Field Values

RIGHTDIRECTION

```
public static final java.lang.String RIGHTDIRECTION
```

See Also:
Constant Field Values

All Classes

AudioFilePlayer
AudioFileSkeleton
Controller
Game
getScoreTest
HS
Model
Model.HighScores
Networking
NoNegativesAllowedException
RowCol
Snake
View

RIGHTDIRECTION

public static final java.lang.String RIGHTDIRECTION

See Also:
Constant Field Values

BODY

public static final java.lang.String BODY

See Also:
Constant Field Values

BODYUP

public static final java.lang.String BODYUP

See Also:
Constant Field Values

P2UPDIRECTION

public static final java.lang.String P2UPDIRECTION

See Also:
Constant Field Values

P2DOWNDIRECTION

public static final java.lang.String P2DOWNDIRECTION

See Also:
Constant Field Values

P2LEFTDIRECTION

public static final java.lang.String P2LEFTDIRECTION

See Also:
Constant Field Values

All Classes

AudioFilePlayer
AudioFileSkeleton
Controller
Game
getScoreTest
HS
Model
Model.HighScores
Networking
NoNegativesAllowedException
RowCol
Snake
View

P2LEFTDIRECTION

public static final java.lang.String P2LEFTDIRECTION

See Also:
Constant Field Values

P2RIGHTDIRECTION

public static final java.lang.String P2RIGHTDIRECTION

See Also:
Constant Field Values

P2BODY

public static final java.lang.String P2BODY

See Also:
Constant Field Values

P2BODYUP

public static final java.lang.String P2BODYUP

See Also:
Constant Field Values

NUMBER_OF_FOOD

public static final int NUMBER_OF_FOOD

See Also:
Constant Field Values

NUMBER_OF_DIE

public static int NUMBER_OF_DIE

levelNumber

public static int levelNumber

```
public static int NUMBER_OF_DIE
```

levelNumber

```
public static int levelNumber
```

SNAKE_SPEED

```
public static final int SNAKE_SPEED
```

See Also:
Constant Field Values

SNAKE_SPEED_TIME

```
public static int SNAKE_SPEED_TIME
```

TIME_TO_SPEED_UP

```
public static int TIME_TO_SPEED_UP
```

SPEED_TIME_DECREASE

```
public static int SPEED_TIME_DECREASE
```

Constructor Detail

Model

```
public Model()
```

This is an empty Model class constructor. It is here to properly instantiate any objects of the Model class.

Method Detail

showHighScores

```
public void showHighScores()
```

All Classes

AudioFilePlayer
AudioFileSkeleton
Controller
Game
getScoreTest
HS
Model
Model.HighScores
Networking
NoNegativesAllowedException
RowCol
Snake
View

Method Detail

showHighScores

```
public void showHighScores()
```

This class gets the High Scores from the HighScores class and displays them in a pop-up message dialog.

setHighScores

```
public void setHighScores(Snake snake)
```

This sets the high scores which are found in a file, and sorts them from highest score to lowest score. If there are more than 10 scores, the list takes the lowest score and erases it, making space available, for the newest high score.

Parameters:
`snake - a snake object`

newTwoPlayerGame

```
public void newTwoPlayerGame()
```

This starts and sets up the game for two players. The second player is placed across from the first player, and a new game is set up for both players.

resetNumberOfPlayers

```
public void resetNumberOfPlayers()
```

newGame

```
public void newGame()
```

This method sets up the new game, first erasing everything in the game board, then going back to level one, setting all of the settings back to their default original, and setting the players back to their original positions.

changeHeadToBody

```
public void changeHeadToBody(Snake snake)
```

This method takes care of the image of the snake as it moves across the game board. As the snake moves, the head moves forward, and this takes care of moving the body in the position that the head just was in.

Parameters:
`snake - takes in a snake object`

showSnakeGone

```
public void showSnakeGone(Snake snake)
```

showSnakeGone

`public void showSnakeGone(Snake snake)`

If the snake is dead or for some reason the snake needs to be taken off the game board, this method erases it.

Parameters:
`snake` - a snake object

showSnakeBody

`public void showSnakeBody(Snake snake)`

This method takes care of the movement of the snake's body as it moves across the game board.

Parameters:
`snake` - a snake object

showSnake

`public void showSnake(Snake snake)`

This method shows the snake's head in a specific direction it is heading.

Parameters:
`snake` - a snake object

setView

`public void setView(View view)`

This sets up the game board for the player. It take's the view, sets it, prompts the player for music as well as the player's name, then resets the game board and places the fresh snake into it.

Parameters:
`view` - a view object

leftPressed

`public void leftPressed(java.lang.String snake)`

Manages the movement of the players' snakes. Depending on which direction the snake is heading it will change it to the left direction.

Parameters:
`snake` - an instance of the snake class

rightPressed

All Classes

AudioFilePlayer
AudioFileSkeleton
Controller
Game
getScoreTest
HS
Model
Model.HighScores
Networking
NoNegativesAllowedException
RowCol
Snake
View

leftPressed

public void leftPressed(java.lang.String snake)

Manages the movement of the players' snakes. Depending on which direction the snake is heading it will change it to the left direction.

Parameters:
snake - an instance of the snake class

rightPressed

`public void rightPressed(java.lang.String snake)`

Manages the movement of the players' snakes. Depending on which direction the snake is heading it will change it to the right direction.

Parameters:
`snake` - a snake object

upPressed

`public void upPressed(java.lang.String snake)`

Manages the movement of the players' snakes. Depending on which direction the snake is heading it will change it to the up direction.

Parameters:
`snake` - a snake object

downPressed

`public void downPressed(java.lang.String snake)`

Manages the movement of the players' snakes. Depending on which direction the snake is heading it will change it to the down direction.

Parameters:
`snake` - a snake object

getScore

`public int getScore(Snake snake)`

Gets the score for the snake object.

Parameters:
`snake` - a snake object

Returns:
`int` an `int` representing the length of the snake + the counter of the game time.

getScore

```
public int getScore(Snake snake)
```

Gets the score for the snake object.

Parameters:
`snake` - a snake object

Returns:
`int` an int representing the length of the snake + the counter of the game time.

Precondition
that the snake's length - 2 multiplied by the counter doesn't reach the max value for integers., that there are no negative values being used

Postcondition
returns an int value representing the score.

getCounter

```
public int getCounter()
```

A getter method for the counter field.

Returns:
`int` a counter for the game's time.

setCounter

```
public void setCounter(int newCount)
```

A setter method for the game's time, counter.

Parameters:
`newCount` - a new value to set the count.

f2Pressed

```
public void f2Pressed()
```

This is the button to start a new game. If a new game is agreed to start, then the old game is stopped, the timers are reset, and then a new game is started.

spacePressed

```
public void spacePressed()
```

This is the pause button for the game. If f2 is pressed, it pauses the game. It works by pausing the timer that moves the snake as well as the timer in the game itself.

All Classes

AudioFilePlayer
AudioFileSkeleton
Controller
Game
getScoreTest
HS
Model
Model.HighScores
Networking
NoNegativesAllowedException
RowCol
Snake
View

f2Pressed

```
public void f2Pressed()
```

This is the button to start a new game. If a new game is agreed to start, then the old game is stopped, the timers are reset, and then a new game is started.

spacePressed

```
public void spacePressed()
```

This is the pause button for the game. If f2 is pressed, it pauses the game. It works by pausing the timer that moves the snake as well as the timer in the game itself.

checkFinish

```
public void checkFinish(Snake snake)
```

This method checks where the snake is on the game board and acts according to its status. If the snake is over the bounds of the game, it's game over. If the snake is over a food object, it grows larger. If the snake hits a certain length, then the next level is loaded. If the snake hits poison, it's game over as well.

Parameters:
`snake` - a snake object

getAudio

```
public AudioFilePlayer getAudio()
```

This is a getter method for the audio file for the game.

Returns:
`AudioFilePlayer` a file that can play music!

setAudio

```
public void setAudio(AudioFilePlayer audio)
```

This is a setter method for the audio file for the game.

Parameters:
`audio` - an audio file that has music!

snakes.game

Class Model.HighScores

java.lang.Object
  snakes.game.Model.HighScores

Enclosing class:
Model

```
public static final class Model.HighScores
extends java.lang.Object
```

The class that manages I/O of the high scores of the game.

Author:
Jason Skinner, Leroy Nguyen, Dawood Choksi, Alessa Ivascu, Christiana Papajani

Method Summary

All Methods | Static Methods | Concrete Methods

Modifier and Type	Method and Description
static java.util.List<HS>	getHighScores() Looks at the file which contains the high scores and reads it, then stores it in an arrayList.

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Method Detail

getHighScores

```
public static java.util.List<HS> getHighScores()
```

Looks at the file which contains the high scores and reads it, then stores it in an arrayList.

Returns:
List a list which contains HS, or also high score objects.

snakes.game

Class Networking

java.lang.Object
  snakes.game.Networking

```
public class Networking
extends java.lang.Object
```

Class used to implement networking functionality to the Snake game.

Author:
Jason Skinner, Leroy Nguyen, Dawood Choksi, Alessa Ivascu, Kristiana Papajani

Method Summary

All Methods | Static Methods | Concrete Methods

Modifier and Type	Method and Description
static void	runServer() This method handles establishing a connection between a client and a server through a network socket, then reading and writing to that server through an input and output stream.

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Method Detail

runServer

```
public static void runServer()
```

This method handles establishing a connection between a client and a server through a network socket, then reading and writing to that server through an input and output stream. After it is finished, the streams are closed, and so are the sockets.

snakes.game

Class NoNegativesAllowedException

java.lang.Object
  java.lang.Throwable
    java.lang.Exception
      java.lang.RuntimeException
        snakes.game.NoNegativesAllowedException

All Implemented Interfaces:

`java.io.Serializable`

```
public class NoNegativesAllowedException
extends java.lang.RuntimeException
```

An unchecked exception when handling unusual cases of negative values.

Author:

Jason Skinner, Leroy Nguyen, Dawood Choksi, Alessa Ivascu, Kristiana Papajani

See Also:

Serialized Form

Constructor Summary

Constructors

Constructor and Description
`NoNegativesAllowedException()` A default constructor which calls upon the parent's class constructor.
`NoNegativesAllowedException(java.lang.String message)` A parameterized constructor which calls upon the parent's class constructor.

Method Summary

Methods inherited from class java.lang.Throwable

`addSuppressed, fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, getSuppressed, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace, toString`

Methods inherited from class java.lang.Object

`equals, getClass, hashCode, notify, notifyAll, wait, wait, wait`

All Classes

AudioFilePlayer
AudioFileSkeleton
Controller
Game
getScoreTest
HS
Model
Model.HighScores
Networking
NoNegativesAllowedException
RowCol
Snake
View

Constructors

Constructor and Description
`NoNegativesAllowedException()` A default constructor which calls upon the parent's class constructor.
`NoNegativesAllowedException(java.lang.String message)` A parameterized constructor which calls upon the parent's class constructor.

Method Summary

Methods inherited from class java.lang.Throwable

`addSuppressed, fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, getSuppressed, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace, toString`

Methods inherited from class java.lang.Object

`equals, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Constructor Detail

NoNegativesAllowedException

```
public NoNegativesAllowedException()
```

A default constructor which calls upon the parent's class constructor.

NoNegativesAllowedException

```
public NoNegativesAllowedException(java.lang.String message)
```

A parameterized constructor which calls upon the parent's class constructor.

Parameters:

`message - a string to alert the user`

snakes.game

Class RowCol

java.lang.Object
  snakes.game.RowCol

```
public class RowCol
extends java.lang.Object
```

Creates a simple Row and column coordinates object, which handles Cartesian coordinates on a 2D plane.

Author:
Jason Skinner, Leroy Nguyen, Dawood Choksi, Alessa Ivascu, Kristiana Papajani

Constructor Summary

Constructors

Constructor and Description
`RowCol(int row, int col)` A parameterized constructor which takes and sets the attributes of an instance of the RowCol class.
`RowCol(RowCol other)` A copy constructor which takes another RowCol object's attributes and copys it into a new RowCol object.

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
int	`col()` This final method returns the col's coordinate.
boolean	`equals(java.lang.Object obj)`
int	`hashCode()`
RowCol	`nextCol()` This is a method which will return the next column after we have already gotten a column
RowCol	`nextRow()` This is a method which will return the next row after we have already gotten the previous row
RowCol	`prevCol()` This is a method which will return the previous column after we have already gotten a column
RowCol	`prevCol()` This is a method which will return the previous column after we have already gotten a column
RowCol	`prevRow()` This is a method which will return the previous row after we have already gotten a row
int	`row()` This final method returns the row's coordinate.
java.lang.String	`toString()`

Methods inherited from class java.lang.Object

`getClass, notify, notifyAll, wait, wait, wait`

Constructor Detail

RowCol

```
public RowCol(int row,
              int col)
```

A parameterized constructor which takes and sets the attributes of an instance of the RowCol class.

Parameters:
`row` - an int representing the number of rows
`col` - an int representing the number of columns

RowCol

```
public RowCol(RowCol other)
```

A copy constructor which takes another RowCol object's attributes and copys it into a new RowCol object.

Parameters:
`other` - another RowCol object.

Method Detail

row

All Classes

AudioFilePlayer
AudioFileSkeleton
Controller
Game
getScoreTest
HS
Model
Model.HighScores
Networking
NoNegativesAllowedException
RowCol
Snake
View

Method Detail

row

```
public final int row()
```

This final method returns the row's coordinate.

Returns:
`row the number of rows from a RowCol object.`

col

```
public final int col()
```

This final method returns the col's coordinate.

Returns:
`col the number of cols from a RowCol object.`

nextRow

```
public RowCol nextRow()
```

This is a method which will return the next row after we have already gotten the previous row

Returns:
`RowCol a RowCol object with an extra row from the previous object.`

prevRow

```
public RowCol prevRow()
```

This is a method which will return the previous row after we have already gotten a row

Returns:
`RowCol a RowCol object with a previous row from the other object.`

nextCol

```
public RowCol nextCol()
```

This is a method which will return the next column after we have already gotten a column

All Classes

AudioFilePlayer
AudioFileSkeleton
Controller
Game
getScoreTest
HS
Model
Model.HighScores
Networking
NoNegativesAllowedException
RowCol
Snake
View

nextCol

```
public RowCol nextCol()
```

This is a method which will return the next column after we have already gotten a column

Returns:
`RowCol a RowCol object with a next column from the other object.`

prevCol

```
public RowCol prevCol()
```

This is a method which will return the previous column after we have already gotten a column

Returns:
`RowCol a RowCol object with a previous column from the other object.`

equals

```
public boolean equals(java.lang.Object obj)
```

Overrides:
`equals in class java.lang.Object`

hashCode

```
public int hashCode()
```

Overrides:
`hashCode in class java.lang.Object`

toString

```
public java.lang.String toString()
```

Overrides:
`toString in class java.lang.Object`

All Classes

AudioFilePlayer
AudioFileSkeleton
Controller
Game
getScoreTest
HS
Model
Model.HighScores
Networking
NoNegativesAllowedException
RowCol
Snake
View

snakes.game

Class Snake

java.lang.Object
  snakes.game.Snake

```
public class Snake
extends java.lang.Object
```

This class take care of everything related to the snake itself. Its body, movement, addition to its body after it eats, as well as its direction and its location on the board. Everything about the a snake object is written here.

Author:
Jason Skinner, Leroy Nguyen, Dawood Choksi, Alessa Ivascu, Kristiana Papajani

Constructor Summary

Constructors

Constructor and Description
`Snake(RowCol head, RowCol body, java.lang.String direction, java.lang.String player)` This parameterized constructor sets everything related to a new snake object.

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
void	`addLength()` Creates a random number, and uses it in order to randomly add 1-3 segments to the snake after it has eaten a food item.
void	`changeDirection(java.lang.String direction)` Changes the direction of a snake into a new direction set by this method.
boolean	`checkForSnake(Snake p1)` Checks to see if there is a collision of the snake's head with its own body.
boolean	`checkForSnake(Snake p1, Snake p2)` Checks to see if there is a collision with each of the player's heads in the other player's bodies.
boolean	`checkMove()` Checks to see if the snake is out of bounds of the game's borders.
java.util.ArrayList<RowCol>	`getBody()` This getter method returns a cloned ArrayList of a body of a snake.
	Checks to see if the snake is out of bounds of the game's borders.
java.util.ArrayList<RowCol>	`getBody()` This getter method returns a cloned ArrayList of a body of a snake.
int	`getCol()` A getter method which returns a column integer of a snake's head.
java.lang.String	`getDirection()` A getter method which returns the direction that a snake is heading in.
RowCol	`getHead()` A getter method which returns the position of the head of a snake.
int	`getLength()` A getter method to get the length of the snake.
java.lang.String	`getPlayerName()` A getter method which returns the player's name of a snake.
int	`getRow()` A getter method which returns the row of the head of a snake.
void	`move()` Moves the visual representation of the snake in a certain direction, using the 2D coordinates laid down by the RowCol class.
RowCol	`nextMove()` Prepares the visual representation of the snake to move in a certain direction.
RowCol	`removing()` Removes the snake's body from the game.
void	`setHead(int row, int col)` This setter method sets the position of the head on the game board.
void	`setLength(int length)` A setter method to set the length of the snake.
java.lang.String	`toString()`

Methods inherited from class java.lang.Object

`equals, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Constructor Detail

Snake

All Classes

AudioFilePlayer
AudioFileSkeleton
Controller
Game
getScoreTest
HS
Model
Model.HighScores
Networking
NoNegativesAllowedException
RowCol
Snake
View

Constructor Detail

Snake

```
public Snake(RowCol head,
             RowCol body,
             java.lang.String direction,
             java.lang.String player)
```

This parameterized constructor sets everything related to a new snake object.

Parameters:

`head` - the head and its position on the game board.

`body` - the body and how many segments are added to the snake itself.

`direction` - the direction that the snake is heading in.

`player` - the name of the player which controls a certain snake.

Method Detail

setHead

```
public void setHead(int row,
                    int col)
```

This setter method sets the position of the head on the game board.

Parameters:

`row` - a row value

`col` - a column value

getBody

```
public java.util.ArrayList<RowCol> getBody()
```

This getter method returns a cloned ArrayList of a body of a snake.

Returns:

`ArrayList` a body of a snake.

All Classes

AudioFilePlayer
AudioFileSkeleton
Controller
Game
getScoreTest
HS
Model
Model.HighScores
Networking
NoNegativesAllowedException
RowCol
Snake
View

getHead

getHead

```
public RowCol getHead()
```

A getter method which returns the position of the head of a snake.

Returns:

`RowCol` a RowCol object representing the position of the head of a snake.

getPlayerName

```
public java.lang.String getPlayerName()
```

A getter method which returns the player's name of a snake.

Returns:

`String` an appropriate player name.

getDirection

```
public java.lang.String getDirection()
```

A getter method which returns the direction that a snake is heading in.

Returns:

`String` the direction a snake is heading in, in a string interpretation.

getRow

```
public int getRow()
```

A getter method which returns the row of the head of a snake.

Returns:

`int` a row of the snake's head.

getCol

```
public int getCol()
```

A getter method which returns a column integer of a snake's head.

Returns:

`int` an int representing the column of the snake's head.

getCol

```
public int getCol()
```

A getter method which returns a column integer of a snake's head.

Returns:
`int an int representing the column of the snake's head.`

addLength

```
public void addLength()
```

Creates a random number, and uses it in order to randomly add 1-3 segments to the snake after it has eaten a food item.

checkForSnake

```
public boolean checkForSnake(Snake p1,
                             Snake p2)
```

Checks to see if there is a collision with each of the player's heads in the other player's bodies.

Parameters:
`p1 - the snake for Player 1`
`p2 - the snake for Player 2`

Returns:
`boolean a true or false value depending on if there is a collision.`

checkForSnake

```
public boolean checkForSnake(Snake p1)
```

Checks to see if there is a collision of the snake's head with its own body.

Parameters:
`p1 - a snake player`

Returns:
`boolean a true or false value depending on whether there is a collision or not.`

checkMove

```
public boolean checkMove()
```

Checks to see if the snake is out of bounds of the game's borders.

All Classes

AudioFilePlayer
AudioFileSkeleton
Controller
Game
getScoreTest
HS
Model
Model.HighScores
Networking
NoNegativesAllowedException
RowCol
Snake
View

checkMove

```
public boolean checkMove()
```

Checks to see if the snake is out of bounds of the game's borders.

Returns:
`boolean a true or false value depending on whether the snake is still in the game's borders, or out of bounds, accordingly.`

nextMove

```
public RowCol nextMove()
```

Prepares the visual representation of the snake to move in a certain direction.

Returns:
`RowCol a new RowCol object representing the next step in the snake's direction.`

removing

```
public RowCol removing()
```

Removes the snake's body from the game.

Returns:
`RowCol a RowCol object representing the point of origin in the game.`

move

```
public void move()
```

Moves the visual representation of the snake in a certain direction, using the 2D coordinates laid down by the RowCol class.

changeDirection

```
public void changeDirection(java.lang.String direction)
```

Changes the direction of a snake into a new direction set by this method.

Parameters:
`direction - the new direction to set the snake into.`

toString

```
public void move()
```

Moves the visual representation of the snake in a certain direction, using the 2D coordinates laid down by the RowCol class.

changeDirection

```
public void changeDirection(java.lang.String direction)
```

Changes the direction of a snake into a new direction set by this method.

Parameters:

`direction` - the new direction to set the snake into.

toString

```
public java.lang.String toString()
```

Overrides:

`toString` in class `java.lang.Object`

getLength

```
public int getLength()
```

A getter method to get the length of the snake.

Returns:

an int representing the length of the snake.

setLength

```
public void setLength(int length)
```

A setter method to set the length of the snake.

Parameters:

`length` - an int to set the length of the snake to.

All Classes

AudioFilePlayer
AudioFileSkeleton
Controller
Game
getScoreTest
HS
Model
Model.HighScores
Networking
NoNegativesAllowedException
RowCol
Snake
View

snakes.game

Class View

java.lang.Object
  java.awt.Component
    java.awt.Container
      java.awt.Window
        java.awt.Frame
          javax.swing.JFrame
            snakes.game.View

All Implemented Interfaces:

`java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable, javax.accessibility.Accessible, javax.swing.RootPaneContainer, javax.swing.WindowConstants`

```
public class View
extends javax.swing.JFrame
```

A view for the snake puzzle game. This class handles showing the snake game to the end user.

Author:

Jason Skinner, Leroy Nguyen, Dawood Choksi, Alessa Ivascu, Kristiana Papajani

See Also:

`Serialized Form`

Nested Class Summary

Nested classes/interfaces inherited from class java.awt.Window

`java.awt.Window.Type`

Nested classes/interfaces inherited from class java.awt.Component

`java.awt.Component.BaselineResizeBehavior`

Field Summary

Fields

Modifier and Type	Field and Description
`static java.util.ArrayList<RowCol>`	`dieLocation`
`static java.lang.String`	`EXIT`
`static java.util.ArrayList<RowCol>`	`foodLocation`

Field Summary

Fields

Modifier and Type	Field and Description
static java.util.ArrayList<RowCol>	dieLocation
static java.lang.String	EXIT
static java.util.ArrayList<RowCol>	foodLocation
static int	GAME_HEIGHT
static java.lang.String	GAME_NAME
static int	GAME_WIDTH
static java.lang.String	HIGH_SCORE
javax.swing.JLabel[][]	labels
static java.lang.String	MENU_HELP
static java.lang.String	MENU_NAME
static java.lang.String	NEW_2_PLAYER_GAME
static java.lang.String	NEW_GAME
static java.lang.String	UP_ICON

Fields inherited from class javax.swing.JFrame

EXIT_ON_CLOSE

Fields inherited from class java.awt.Frame

CROSSHAIR_CURSOR, DEFAULT_CURSOR, E_RESIZE_CURSOR, HAND_CURSOR, ICONIFIED, MAXIMIZED_BOTH, MAXIMIZED_HORIZ, MAXIMIZED_VERT, MOVE_CURSOR, N_RESIZE_CURSOR, NE_RESIZE_CURSOR, NORMAL, NW_RESIZE_CURSOR, S_RESIZE_CURSOR, SE_RESIZE_CURSOR, SW_RESIZE_CURSOR, TEXT_CURSOR, W_RESIZE_CURSOR, WAIT_CURSOR

Fields inherited from class java.awt.Component

BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT, TOP_ALIGNMENT

Fields inherited from interface javax.swing.WindowConstants

DISPOSE_ON_CLOSE, DO_NOTHING_ON_CLOSE, HIDE_ON_CLOSE

Fields inherited from interface java.awt.image.ImageObserver

All Classes

AudioFilePlayer
AudioFileSkeleton
Controller
Game
getScoreTest
HS
Model
Model.HighScores
Networking
NoNegativesAllowedException
RowCol
Snake
View

Fields inherited from interface javax.swing.WindowConstants

DISPOSE_ON_CLOSE, DO_NOTHING_ON_CLOSE, HIDE_ON_CLOSE

Fields inherited from interface java.awt.image.ImageObserver

ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH

Constructor Summary

Constructors

Constructor and Description
View(java.awt.event.ActionListener listener, java.awt.event.KeyListener keyListener) This constructor handles the creation and display of the game board to the end user.

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
void	clear() This method clears the View and resets the game.
int	promptForMusic() The method shows a dialog box with the option to choose if you want music running while playing the game.
java.lang.String	promptForName(java.lang.String name) The method shows a dialog box when the Game is launched.
int	promptForTwoPlayer() The method shows a dialog box when the Multiplayer Game option is clicked in the Game menu.
void	randomDie(int max) This method creates random poison/toxic food units across the map that the snake traverses.
void	randomFood(int max) This method creates a random food across the map that the snake traverses.

Methods inherited from class javax.swing.JFrame

getAccessibleContext, getContentPane, getDefaultCloseOperation, getGlassPane, getGraphics, getJMenuBar, getLayeredPane, getRootPane, getTransferHandler, isDefaultLookAndFeelDecorated, remove, repaint, setContentPane, setDefaultCloseOperation, setDefaultLookAndFeelDecorated, setGlassPane, setIconImage, setJMenuBar, setLayeredPane, setLayout, setTransferHandler, update

Methods inherited from class javax.swing.JFrame

getAccessibleContext, getContentPane, getDefaultCloseOperation, getGlassPane, getGraphics, getJMenuBar, getLayeredPane, getRootPane, getTransferHandler, isDefaultLookAndFeelDecorated, remove, repaint, setContentPane, setDefaultCloseOperation, setDefaultLookAndFeelDecorated, setGlassPane, setIconImage, setJMenuBar, setLayeredPane, setLayout, setTransferHandler, update

Methods inherited from class java.awt.Frame

addNotify, getCursorType, getExtendedState, getFrames, getIconImage, getMaximizedBounds, getMenuBar, getState, getTitle, isResizable, isUndecorated, remove, removeNotify, setBackground, setCursor, setExtendedState, setMaximizedBounds, setMenuBar, setOpacity, setResizable, setShape, setState, setTitle, setUndecorated

Methods inherited from class java.awt.Window

addPropertyChangeListener, addPropertyChangeListener, addWindowFocusListener, addWindowListener, addWindowStateListener, applyResourceBundle, applyResourceBundle, createBufferStrategy, createBufferStrategy, dispose, getBackground, getBufferStrategy, getFocusableWindowState, getFocusCycleRootAncestor, getFocusOwner, getFocusTraversalKeys, getIconImages, getInputContext, getListeners, getLocale, getModalExclusionType, getMostRecentFocusOwner, getOpacity, getOwnedWindows, getOwner, getOwnerlessWindows, getShape, getToolkit, getType, getWarningString, getWindowFocusListeners, getWindowListeners, getWindows, getWindowStateListeners, hide, isActive, isAlwaysOnTop, isAlwaysOnTopSupported, isAutoRequestFocus, isFocusableWindow, isFocusCycleRoot, isFocused, isLocationByPlatform, isOpaque, isShowing, isValidateRoot, pack, paint, postEvent, removeWindowFocusListener, removeWindowListener, removeWindowStateListener, reshape, setAlwaysOnTop, setAutoRequestFocus, setBounds, setBounds, setCursor, setFocusableWindowState, setFocusCycleRoot, setIconImages, setLocation, setLocation, setLocationByPlatform, setLocationRelativeTo, setMinimumSize, setModalExclusionType, setSize, setSize, setType, setVisible, show, toBack, toFront

Methods inherited from class java.awt.Container

add, add, add, add, add, addContainerListener, applyComponentOrientation, areFocusTraversalKeysSet, countComponents, deliverEvent, doLayout, findComponentAt, findComponentAt, getAlignmentX, getAlignmentY, getComponent, getComponentAt, getComponentAt, getComponentCount, getComponents, getComponentZOrder, getContainerListeners, getFocusTraversalPolicy, getInsets, getLayout, getMaximumSize, getMinimumSize, getMousePosition, getPreferredSize, insets, invalidate, isAncestorOf, isFocusCycleRoot, isFocusTraversalPolicyProvider, isFocusTraversalPolicySet, layout, list, list, locate, minimumSize, paintComponents, preferredSize, print, printComponents, remove, removeAll, removeContainerListener, setComponentZOrder, setFocusTraversalKeys, setFocusTraversalPolicy, setFocusTraversalPolicyProvider, setFont, transferFocusDownCycle, validate

Methods inherited from class java.awt.Component

action, add, addComponentListener, addFocusListener, addHierarchyBoundsListener, addHierarchyListener, addInputMethodListener, addKeyListener, addMouseListener, addMouseMotionListener, addMouseWheelListener, bounds, checkImage, checkImage, contains, contains, createImage, createImage, createVolatileImage, createVolatileImage, disable, dispatchEvent, enable, enable, enableInputMethods, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, getBaseline, getBaselineResizeBehavior, getBounds, getBounds, getColorModel, getComponentListeners, getComponentOrientation, getCursor, getDropTarget, getFocusListeners, getFocusTraversalKeysEnabled, getFont, getFontMetrics, getForeground, getGraphicsConfiguration, getHeight, getHierarchyBoundsListeners, getHierarchyListeners, getIgnoreRepaint, getInputMethodListeners, getInputMethodRequests, getKeyListeners, getLocation, getLocation, getLocationOnScreen, getMouseListeners, getMouseMotionListeners, getMousePosition, getMouseWheelListeners, getName, getParent, getPeer, getPropertyChangeListeners, getPropertyChangeListeners, getSize, getSize, getTreeLock, getWidth, getX, getY, gotFocus, handleEvent, hasFocus, imageUpdate, inside, isBackgroundSet, isCursorSet, isDisplayable, isDoubleBuffered, isEnabled, isFocusable, isFocusOwner, isFocusTraversable, isFontSet, isForegroundSet, isLightweight, isMaximumSizeSet, isMinimumSizeSet, isPreferredSizeSet, isValid, isVisible, keyDown, keyUp, list, list, list, location, lostFocus, mouseDown, mouseDrag, mouseEnter, mouseExit, mouseMove, mouseUp, move, nextFocus, paintAll, prepareImage, prepareImage, printAll, removeComponentListener, removeFocusListener, removeHierarchyBoundsListener, removeHierarchyListener, removeInputMethodListener, removeKeyListener, removeMouseListener, removeMouseMotionListener, removeMouseWheelListener, removePropertyChangeListener, removePropertyChangeListener, repaint, repaint, repaint, requestFocus, requestFocusInWindow, resize, resize, revalidate, setComponentOrientation, setDropTarget, setEnabled, setFocusable, setFocusTraversalKeysEnabled, setForeground, setIgnoreRepaint, setLocale, setMaximumSize, setName, setPreferredSize, show, size, toString, transferFocus, transferFocusBackward, transferFocusUpCycle

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

All Classes

AudioFilePlayer
AudioFileSkeleton
Controller
Game
getScoreTest
HS
Model
Model.HighScores
Networking
NoNegativesAllowedException
RowCol
Snake
View

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Methods inherited from interface java.awt.MenuContainer

getFont, postEvent

Field Detail

GAME_NAME

public static final java.lang.String GAME_NAME

See Also:
Constant Field Values

NEW_GAME

public static final java.lang.String NEW_GAME

See Also:
Constant Field Values

NEW_2_PLAYER_GAME

public static final java.lang.String NEW_2_PLAYER_GAME

See Also:
Constant Field Values

MENU_NAME

public static final java.lang.String MENU_NAME

See Also:
Constant Field Values

MENU_HELP

MENU_HELP

```
public static final java.lang.String MENU_HELP
```

See Also:
Constant Field Values

HIGH_SCORE

```
public static final java.lang.String HIGH_SCORE
```

See Also:
Constant Field Values

EXIT

```
public static final java.lang.String EXIT
```

See Also:
Constant Field Values

UP_ICON

```
public static final java.lang.String UP_ICON
```

See Also:
Constant Field Values

GAME_WIDTH

```
public static final int GAME_WIDTH
```

See Also:
Constant Field Values

GAME_HEIGHT

```
public static final int GAME_HEIGHT
```

See Also:
Constant Field Values

All Classes

AudioFilePlayer
AudioFileSkeleton
Controller
Game
getScoreTest
HS
Model
Model.HighScores
Networking
NoNegativesAllowedException
RowCol
Snake
View

GAME_HEIGHT

```
public static final int GAME_HEIGHT
```

See Also:
Constant Field Values

labels

```
public javax.swing.JLabel[][] labels
```

foodLocation

```
public static final java.util.ArrayList<RowCol> foodLocation
```

dieLocation

```
public static final java.util.ArrayList<RowCol> dieLocation
```

Constructor Detail

View

```
public View(java.awt.event.ActionListener listener,
            java.awt.event.KeyListener keyListener)
```

This constructor handles the creation and display of the game board to the end user. The window is first displayed, and then the labels, listeners, and layouts are added and packed so that all of its contents are set at a preferred, optimal size.

Parameters:
`listener` - an ActionListener designed to listen for the user's button clicks.

`keyListener` - a KeyListener designed to listen for the user's keyboard inputs.

Method Detail

promptForMusic

```
public int promptForMusic()
```

Method Detail

promptForMusic

```
public int promptForMusic()
```

The method shows a dialog box with the option to choose if you want music running while playing the game. The dialog box has the options "Yes", "No", and an "X" symbol in the corner.

Returns:
an int value corresponding to the option selected or CLOSED_OPTION if the "X" is clicked.

Postcondition
will either play the music, or will stay silent.

promptForTwoPlayer

```
public int promptForTwoPlayer()
```

The method shows a dialog box when the Multiplayer Game option is clicked in the Game menu. The dialog box allows the user to enter a 2nd player's name. The dialog box has the options "OK", "Cancel", and an "X" symbol in the corner.

Returns:
an int value corresponding to the option selected or CLOSED_OPTION if the "X" is clicked.

Postcondition
will display the player name entered, or null if nothing is entered.

promptForName

```
public java.lang.String promptForName(java.lang.String name)
```

The method shows a dialog box when the Game is launched. The dialog box allows the user to enter the 1st/main player's name. The dialog box has the options "OK", "Cancel", and an "X" symbol in the corner. If an invalid string name is entered, the user is alerted and then taken back to the original dialog box to enter a name.

Parameters:
name - - a string inputed by the user that cannot contain a "," character.

Returns:
the inputed string from the user.

Precondition
The inputed String name from the user must not contain a "," as that character is used in the High Score display format.

Postcondition
will display the player name entered, or null if nothing is entered.

All Classes

AudioFilePlayer
AudioFileSkeleton
Controller
Game
getScoreTest
HS
Model
Model.HighScores
Networking
NoNegativesAllowedException
RowCol
Snake
View

the inputed string from the user.

Precondition
The inputed String name from the user must not contain a "," as that character is used in the High Score display format.

Postcondition
will display the player name entered, or null if nothing is entered.

clear

```
public void clear()
```

This method clears the View and resets the game. The snake position is reset and all shown food are cleared. The various art, background, and user name are all reset.

randomFood

```
public void randomFood(int max)
```

This method creates a random food across the map that the snake traverses. If a food unit is eaten, it will cause the snake's body to grow. The food units have their own custom icon. The food unit are added randomly across the map grid, and only 1 is shown to the user, and a new one is generated at a random location when the previous one is eaten.

Parameters:
max - - a value determined by from the Model, and starts at 1. After 6 food units are eaten, a poison unit is created. After 10 food units are eaten, the level is advanced.

Precondition
the max int value is never negative.

randomDie

```
public void randomDie(int max)
```

This method creates random poison/toxic food units across the map that the snake traverses. If a poisoned/toxic food unit is eaten, it will cause the game to end, and your score is shown. The poisoned/toxic food units have their own custom icon. The poisoned units are added randomly across the map grid, and gradually increase in number as more food units are eaten by the snake.

Parameters:
max - - a value determined by from the Model, and starts at 5. After 6 food units are eaten, a poison unit is created. Once a poison unit is eaten, the game ends.

Precondition
the max int value is never negative.