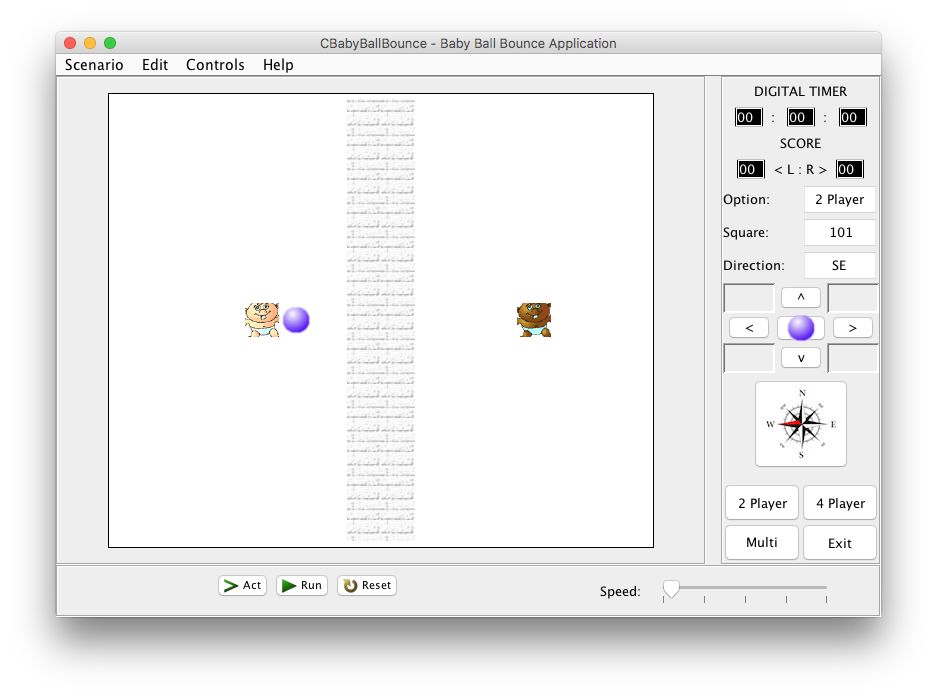
**CSY1020: Problem Solving & Programming Assignment 2: Programming (Java) (50%)**

Ashleigh Claire Foster

16427265

**BSc Business Computing (Systems)

University of Northampton

Submission: **Sunday, 30th April 2017, by 23h59.**

**Table of Contents**

[University of Northampton 1](#_Toc481344648)

[1. Introduction & Problem Specification 3](#_Toc481344649)

[1.1 Problem Statement 3](#_Toc481344650)

[1.2 Aims 4](#_Toc481344651)

[Figure 1.1: Emulation - CBabyBallBounce.java Application – Opening State. 4](#_Toc481344652)

[1.3 Objectives [Take from the assignment brief] 5](#_Toc481344653)

[2 Analysis 6](#_Toc481344654)

[2.1 Inputs 6](#_Toc481344655)

[2.2 Outputs 6](#_Toc481344656)

[2.3 Calculations 6](#_Toc481344657)

[2.4 Procedures 7](#_Toc481344658)

[2.5 Assumptions 7](#_Toc481344659)

[2.5. System Requirements: Essential (Graphical User Interface) [From brief] 8](#_Toc481344660)

[2.6. System Requirements: Additional (Functionality & Complexity) [Brief] 8](#_Toc481344661)

[3 Design 9](#_Toc481344662)

[3.1 GUI (Graphical user interface) Designs 9](#_Toc481344663)

[Figure 1.2: Emulation - CBabyBallBounce.java Application – Opening State. 9](#_Toc481344664)

[3.2 Flowcharts 10](#_Toc481344665)

[4. Implementation 14](#_Toc481344666)

[Figure 1.3s: Emulation - CBabyBallBounce.java Application – Opening State. 18](#_Toc481344667)

[5. Testing 44](#_Toc481344668)

[5.1 Test Plan 44](#_Toc481344669)

[System Requirements: Essential (Graphical User Interface): 49](#_Toc481344670)

[System Requirements: Additional (Functionality & Complexity): 49](#_Toc481344671)

[6. Conclusion & Recommendations 51](#_Toc481344672)

[References 55](#_Toc481344673)

[7. Screenshots – Application Build up 57](#_Toc481344674)

**TABLE OF FIGURE**

Figure 1.1: Emulation - CBabyBallBounce.java Application – Opening State. 4

Figure 1.2: Emulation - CBabyBallBounce.java Application – Opening State. 9

Figure 1.3: Emulation - CBabyBallBounce.java Application – Opening State 18

**LIST OF TABLES**

Figure 1.1: Test Table - CBabyBallBounce.java Test Application 44

# **Introduction & Problem Specification**

## **1.1 Problem Statement**

The main purpose of the problem statement section of the report is to produce a Java GUI (graphical user interface) called CBabyBallBounce which has a similar layout to the Greenfoot game that the author produced in the first term of the 1st assignment. However, in this scenario the author is now going to change the layout slightly so that it doesn’t show the class names on the right-hand side of the GUI but rather a navigation panel which does/shows the following:

* Simulates the control of the ball so that the ball can move across the world in whichever direction possible using the X and Y axis.
* The type of keys and buttons which are going to be used are the left, right, down, and up keys and buttons.
* A digital timer will be added to the top right-hand corner of the right panel so that there is a set time allocated on the game.
* A score counter will be placed on the right-hand panel rather than on the centre panel itself i.e. L standing for left and R standing for right.
* The game will include several features including a direction option and it will display how many players there will be in the game i.e. 2 players or 4 players using the compass, the option part and the buttons which will allow you to select whether the end-user wants to play in a 2-player game or a 4-player game.

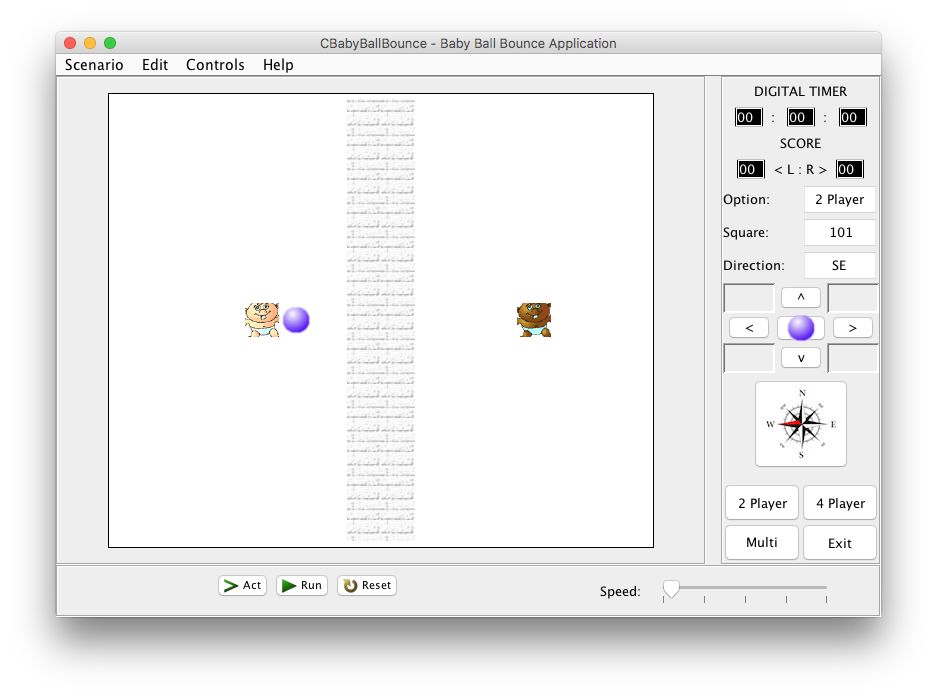
The author is also going to add a bottom panel to the CBabyBallBounce game which will display a speed slider i.e. the end-user adjusting the speed to however fast or slow the end-user wants to play the game and there will be 3 different buttons showing at the bottom which allows certain parts of the game to function which are called “Act, Run and Reset”.

The game will include various images such as the 2 – 4 different baby images (depending on which option the end-user selects) as well as a centre panel with a dimension of 825x585 in which the game will function and operate.

Finally, there will be a menu bar at the top of the game which will have a “Scenario, Edit, Control and Help” label which will allow the end-user to open different options and functions within the game such as a debugger, opening files, sound recorder and various tutorials and websites directing the end-user on how to play the game.

Based on the assignment brief there are several aims and objectives which need to be covered (again these have been directly copied from the assignment brief).

## **1.2 Aims**

**The main aims which the client has specified is that the author needs to produce both a technical report and a graphical user interface (GUI) application with the main purpose of the application being a simulated game (like Greenfoot) i.e. simulating the control of the ball being kicked back and forth around the pitch using 4 different arrow keys (left, right, up and down keys). The client has also specified that he would like us to take screenshots of the application during the testing and implementation stages of the application as well as an appendix of all the code used once the application has been completed. At week 30 he has asked us if he can see a demonstration of the application product working in action as well as checking to see whether everything works as he expects it too. The client has provided the author with a designed version of how he wants the application to look like as shown below:

### Figure 1.1: Emulation - CBabyBallBounce.java Application – Opening State.

## **Objectives [Take from the assignment brief]**

**Rules (Basic)** Create a simulation of the ball moving around the pitch, where:

* The ball can move anywhere within the pitch and across the wall in the middle of the screen.
* If the ball touches a baby it is deflected/rebounds back in the opposite direction.
* The ball must move one whole ‘white/wall’ block at a time every time a movement button (via a direction button (<, > v ^)) /key is pressed (when movement is possible).
* The solution must use the scenario provided. i.e. if the babies are left in the unmodified codes starting position the ball would move between them continuously.
* The ball must stop when the perimeter of the pitch is reached.
* The basic solution must be completed using the ‘act’ button (accessing the **kickBall()** method within the **CBabyBallBounce.class**).

**Rules (Intermediate and advanced)** Create a simulation of a ball moving around the pitch, where:

* Your solution must still use the scenario provided (all the basic features above).
* Add appropriate extra features to the solutions, e.g. a) The ball can bounce off a baby or babies in random direction, b) Two new ‘player’ babies are added to each side of the line, that move vertically towards the ball ready to potentially bounce the ball back, c) Add a scoring system for each side (a, b or c for **Intermediate,** a, b & c for **Advanced**).
* For higher grades on the solution part of the assignment see the marking scheme at the front of the brief.
* You must NOT change the layout and all changes should still meet the criteria of **Rules (Basic**).

# **Analysis**

In this section of the report the author needs to decide what the following inputs, outputs, calculations, procedures, and assumptions i.e. deciding how the program should work and operate as well as analysing and resolving each of these procedures so that it is easier to manage when it comes to designing and implementing the GUI for the CBabyBallBounce game. The author is now going to discuss these procedures in more detail:

### **2.1 Inputs**

* The speed slider on the bottom right-hand side to determine how fast or how slow the game needs to run.
* The buttons need to be pressed in order to carry out an operation i.e. when “2 players” is pressed the following output in the options will be “2 players”. When the arrow keys are pressed, the following output produced would be the babies moving in different directions depending on which arrow was pressed. The same applies to the middle ball button as when the user presses the middle ball button it automatically starts the game off rather than having to press the space bar. The act, run and reset buttons also perform operations i.e. the run button will run the game, the reset button will reset the game and the act button will start from the beginning (a bit like the run button).
* Every time the end-user clicks on one of the JLabels at the top i.e. scenario, edit, controls or help then it will list the number of items depending on the label i.e. scenario will have “exit” and the help label will have “help topic” and “about”.

### **2.2 Outputs**

* The game performs at either a slow, medium or fast speed depending on how the user has inputted that action in the first place and is displayed on the screen.
* The buttons need to be pressed in order to carry out an operation i.e. when “2 players” is pressed the following output in the options will be “2 players”. When the arrow keys are pressed, the following output produced would be the babies moving in different directions depending on which arrow was pressed. The same applies to the middle ball button as when the user presses the middle ball button it automatically starts the game off rather than having to press the space bar. The act, run and reset buttons also perform operations i.e. the run button will run the game, the reset button will reset the game and the act button will start from the beginning (a bit like the run button). It will then display this information.
* Every time the end-user clicks on one of the JLabels at the top i.e. scenario, edit, controls or help then it will list the number of items depending on the label i.e. scenario will have “exit” and the help label will have “help topic” and “about”. It will then display this information.

### **2.3 Calculations**

* The game needs to calculate how fast or how slow the game needs to perform by calculating the exact amount the user has inputted originally so that the inputted number can be determined.
* The digital timer and the score counter will need to perform a series of calculations. For the digital timer, it is basically like a countdown until the game finishes or ends which means that the timer will need to perform calculations which keep going down in value or by each integer. For the score counter, the counter will need to perform calculations every time the ball bounces off the baby and depending on the baby will depend on which score counter gets added i.e. if the left baby bounced the ball it would add up by 1 point for the left baby whereas if the ball bounced off the right baby the score will add up by 1 point but for the right counter instead of the left counter.

### **2.4 Procedures**

* The computer will produce the following CBabyBallBounce java game by following a series of procedures that were programmed into the Eclipse Neon program and then producing the final output in a java application. This will include a working game that will allow the end-user to select several different options as well as allowing the end-user to play a simple java game. The code will be compiled first as the computer will not be able to read a high-level programming language as it only reads in 0’s and 1’s which is a low-level language.
* A series of numbers will need to be converted in both the digital timer itself and the scoring counter.

### **2.5 Assumptions**

Based on what was read in the assignment brief and the authors own assumptions the game that the author needs to design and create is a replicate of the Greenfoot simulated game which was produced in assignment 1 but with additional features added i.e. a different right-hand side panel with different features so that the game is functioning more like a game and not a programming application. There are a few rules in which the author needs to abide by including:

* The ball can move anywhere in the game using the X and Y axis/coordinates as well as being able to cross the wall in the middle of the screen.
* The ball will need to bounce off the babies (whether it is 2 players or 4 players) and pass between each of the players without any problems.
* The ball can be moved using the up, down, left and right keys but can only be moved one white square at a time when a key is pressed.
* The author needs to abide by the following scenario “If the babies are left in the unmodified codes starting position the ball would move between them continuously”.
* When the perimeter pitch has reached the final level on the score counter then the ball must stop.
* The act method must be used so that it can access the “kickball()” method that is contained inside in the CBabyBallBounce class.
* Appropriate features must be added to game so that the ball can bounce off of the babies in a random direction, added 2 new player babies to each side of the line so that it displays 6 babies rather than 4 as well as making sure those babies move vertically upwards so that in can bounce the ball back and a scoring system needs to be added to the right-hand side in the “digital timer” section so that it counts the score depending on whether the baby is on the left side or the right side of the pitch.

**The content below is taken from the assignment brief for further analysis:**

## **2.5. System Requirements: Essential (Graphical User Interface) [From brief]**

* 13 x 16 grid of **JButton**’s or Icon’s.
* 4 **JButton**’s for the game options ‘2 Player, 4 Player, Multi’ and ‘*Exit*’.
* 3 **JButton**’s for ‘Act’, ‘Run’ and ‘*Reset*’.
* 9 **JButton**’s for ‘*Forward >*’, ‘Backwards <’, ‘Up ^’, ‘Down v’ should move the ball in the appropriate direction by one square for each press (plus 5 blank).
* The compass icon (**JButton)** should illustrate the current direction for the ball.
* **3 JLabel**’s for ’Option’, ‘*Square*’ and ‘*Direction*’.
* **3 JTextField**’s for the current ‘Option’, Location/*’Square’* and *‘Direction’* of the ball. Use the square identification method e.g. 0 to 207 and N, E etc.
* 3 **JLabel’s** for the ‘DIGITAL TIMER and the two :’, with 3 **JTextField’**s for the hours, minutes and seconds.
* 2 **JLabel’s** for the ‘SCORE and ‘<L:R>’, with 2 **JTextField’**s for the scores (L & R).
* Create a **JFrame** application, which opens to the set size (825 \* 585).
* **JFrame** title set as "*CBabyBallBounce – Baby Ball Bounce Application*".

## **2.6. System Requirements: Additional (Functionality & Complexity) [Brief]**

* Application icon for the **JFrame** used.
* The ‘Run’ **JButton** should show the ball moving between the babies continuously from the initial position (2 Player – default opening state).
* The ‘Reset’ **JButton** should clear/reset the application to its starting/default opening state.
* The ‘Act’ **JButton** should step through the above ‘Run’ sequence one move at a time.
* Discuss and implement the different options for the 3 configurations.
* The ‘2 Player, 4 Player, Multi’ **JButton**’s should display different obstacle/car configurations/locations.
* A **JMenuBar** could be included with **JMenu**’s for the *Scenario, Edit, Controls* and *Help*, which include **JMenuItem**’s of *Exit (Scenario)*, *Help Topic* and *About (Help)*.
* Additional **JButton**’s may be used to improve the applications usability e.g. ball bounce – in random direction, deflection angle etc.
* Create a **JFrame** application, which is not resizable.
* Create a **JFrame** application, which centres itself on the monitor.
* Use of additional baby images indicating the current position and direction of the baby.
* Discuss the possibilities for incorporating intelligence/checks for whether moves are valid.
* Digital Timer should start and stop when run is pressed and stopped when a baby misses the ball (with the ball continuing to the left or right boundary and stopping itself and the timer).
* Implement intelligence/checks for whether moves are valid.
* A **kickBall()** method should be used to solve the problem. The **kickBall()** method should include **move(left), move(right), move(up), move(down)** methods (see below).

public void kickBall()

{

move(left);

……………….

move(right);

}

The applications **must** be demonstrated (see below). The source code file containing the **main()** method and the compiled byte code **class** files should be named as follows:

**CBabyBallBounce.java** & **CBabyBallBounce.class**

# **3 Design**

## ***F:\Northampton Uni\Bcs Business Computing - Year 1\Problem Solving & Programming\Assignment2\csy1020Ass2\bin\csy1020Ass2\baby1.pngF:\Northampton Uni\Bcs Business Computing - Year 1\Problem Solving & Programming\Assignment2\csy1020Ass2\bin\csy1020Ass2\ball.pngF:\Northampton Uni\Bcs Business Computing - Year 1\Problem Solving & Programming\Assignment2\csy1020Ass2\bin\csy1020Ass2\bricks2.jpgF:\Northampton Uni\Bcs Business Computing - Year 1\Problem Solving & Programming\Assignment2\csy1020Ass2\bin\csy1020Ass2\bricks2.jpgF:\Northampton Uni\Bcs Business Computing - Year 1\Problem Solving & Programming\Assignment2\csy1020Ass2\bin\csy1020Ass2\bricks2.jpgF:\Northampton Uni\Bcs Business Computing - Year 1\Problem Solving & Programming\Assignment2\csy1020Ass2\bin\csy1020Ass2\bricks2.jpgF:\Northampton Uni\Bcs Business Computing - Year 1\Problem Solving & Programming\Assignment2\csy1020Ass2\bin\csy1020Ass2\bricks2.jpgF:\Northampton Uni\Bcs Business Computing - Year 1\Problem Solving & Programming\Assignment2\csy1020Ass2\bin\csy1020Ass2\bricks2.jpgF:\Northampton Uni\Bcs Business Computing - Year 1\Problem Solving & Programming\Assignment2\csy1020Ass2\bin\csy1020Ass2\bricks2.jpgF:\Northampton Uni\Bcs Business Computing - Year 1\Problem Solving & Programming\Assignment2\csy1020Ass2\bin\csy1020Ass2\bricks2.jpgF:\Northampton Uni\Bcs Business Computing - Year 1\Problem Solving & Programming\Assignment2\csy1020Ass2\bin\csy1020Ass2\bricks2.jpgF:\Northampton Uni\Bcs Business Computing - Year 1\Problem Solving & Programming\Assignment2\csy1020Ass2\bin\csy1020Ass2\bricks2.jpgF:\Northampton Uni\Bcs Business Computing - Year 1\Problem Solving & Programming\Assignment2\csy1020Ass2\bin\csy1020Ass2\bricks2.jpgF:\Northampton Uni\Bcs Business Computing - Year 1\Problem Solving & Programming\Assignment2\csy1020Ass2\bin\csy1020Ass2\bricks2.jpgF:\Northampton Uni\Bcs Business Computing - Year 1\Problem Solving & Programming\Assignment2\csy1020Ass2\bin\csy1020Ass2\bricks2.jpgF:\Northampton Uni\Bcs Business Computing - Year 1\Problem Solving & Programming\Assignment2\csy1020Ass2\bin\csy1020Ass2\bricks2.jpgF:\Northampton Uni\Bcs Business Computing - Year 1\Problem Solving & Programming\Assignment2\csy1020Ass2\bin\csy1020Ass2\baby2.png3.1 GUI (Graphical user interface) Designs***

JIcon and JButton

JLabels

JTextField

JPanels

JSlider

JMenu and JMenuItem

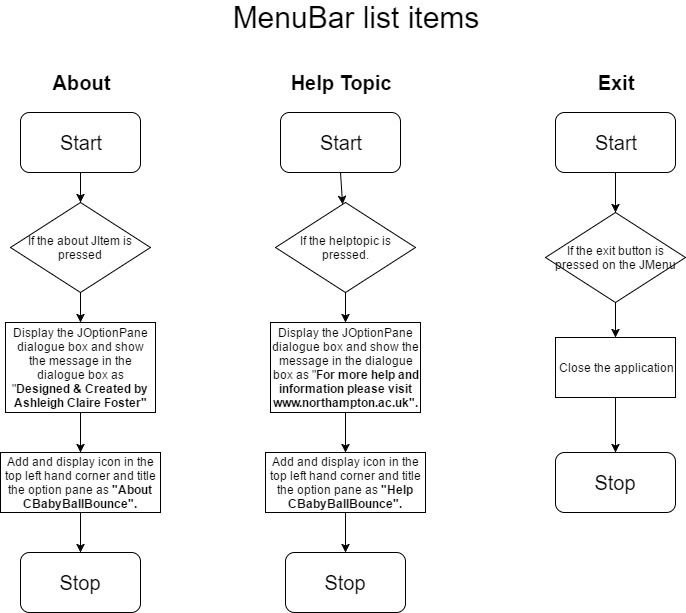
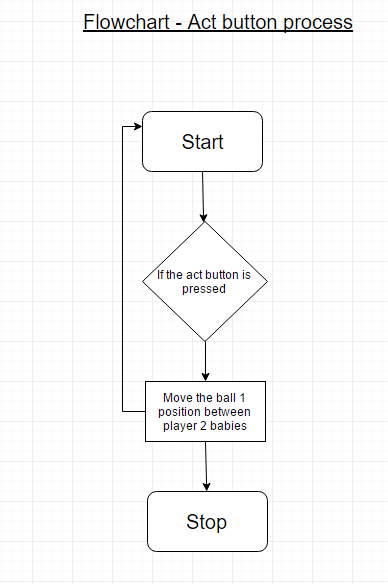
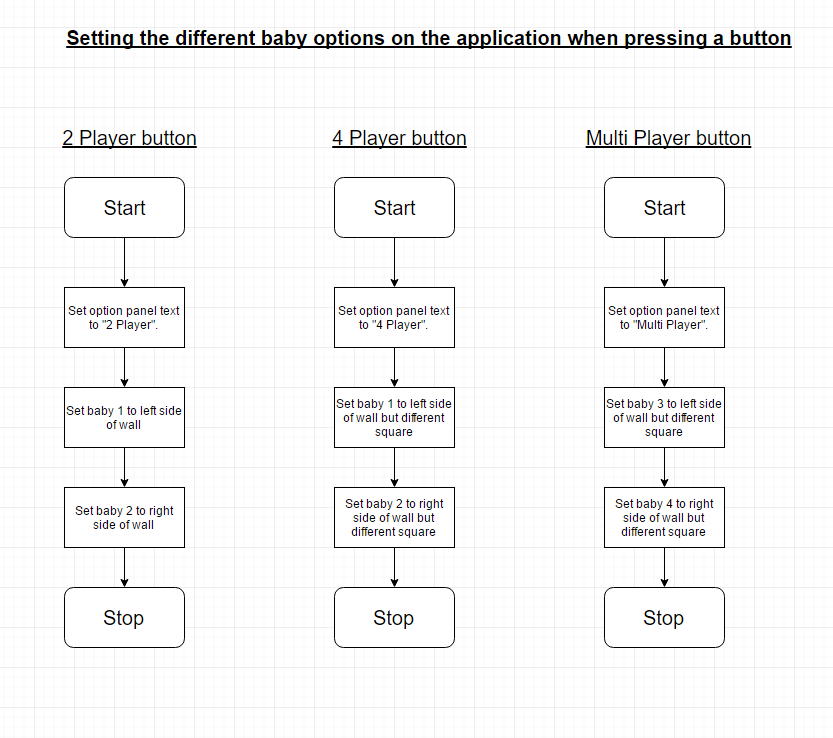
JButtons

### Figure 1.2: Emulation - CBabyBallBounce.java Application – Opening State.

1. Grey panels.
2. White main inner background.
3. The JButtons around the arrow keys will be set to a “non-editable” mode meaning the end-user can’t press on those buttons and let it interfere with the timer.
4. The application mustn’t be resizable.

## C:\Users\acf96\AppData\Local\Microsoft\Windows\INetCache\Content.Word\rest_button.png***3.2 Flowcharts***

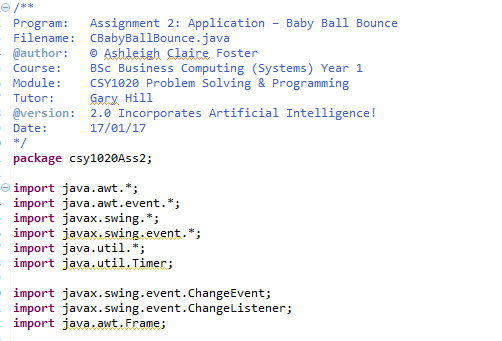
Ignore this bit!

C:\Users\acf96\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Untitled Diagram.png

C:\Users\acf96\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Untitled Diagram.png

# **4. Implementation**

In this section of the report, the author is now going to explain how they have gone about implementing the relevant code into the CBabyBallBounce application as well as how they have approached and accomplished the overall design of the application. The author has made sure that all of the necessary requirements have been met as stated in the clients brief.

**CBabyBallBounce – Imports**

Before even getting started with the overall design and construction of the CBabyBallBounce application the author needs to make sure that they have imported the relevant java libraries which correspond to the java code that they will be using to implement various parts of the design to the application. As shown in the imports screenshot above, the author has accidently added more java libraries than they needed too. The author is now going to explain what each of the imported libraries is used for in the java application:

**Import java.awt.\*;** - This java library basically provides the foundations for the overall design of the game as these forms each of the components of the GUI (graphical user interface). This could include components such as the buttons, drawing images and various other graphical components. The \* means all classes within that library.

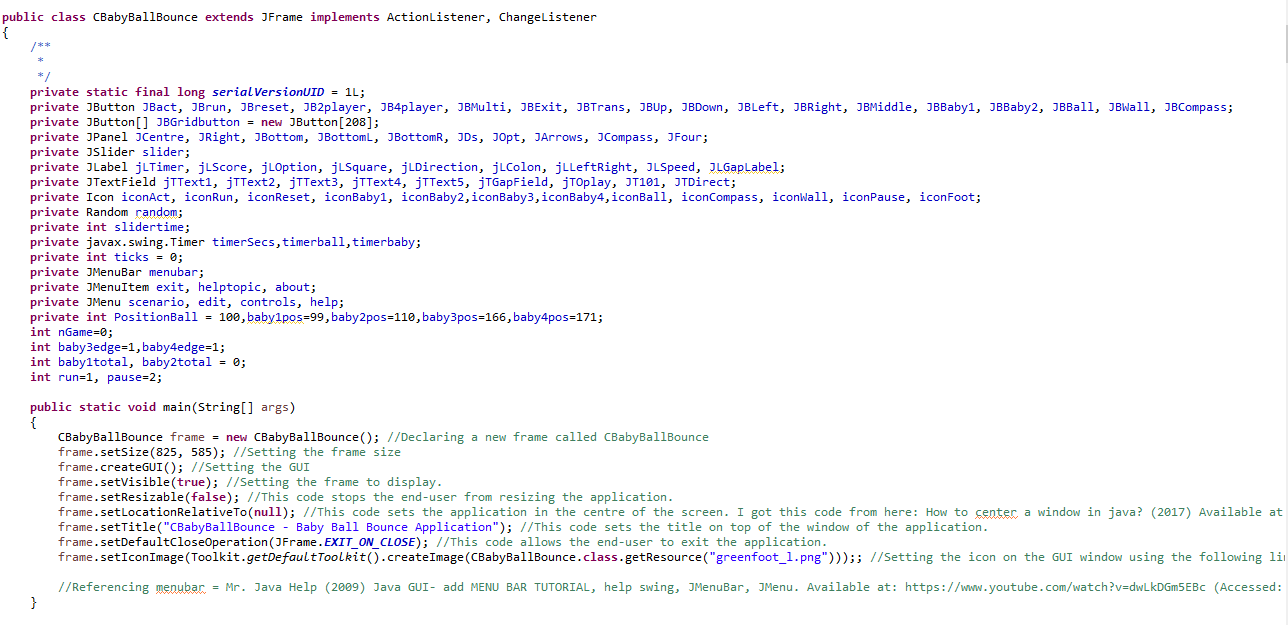
**Import java.awt.event.\*;** - This java library is similar to the java.awt java library, however, the only difference is that it deals with certain events that occur as a result of the implementation of a java.awt component. A good example of this would be the action performed method in the CBabyBallBounce application. The \* means all classes within that library.

**Import javax.swing\*;** - This java library is similar or like the java.awt library in that it deals with the graphical components of the application, however, the only difference is that it deals with a lot more graphical components such as the JFrame, icons, layout managers (i.e. grid layout), buttons, the menu bar with its menu items, the popup panels, the slider and the text fields. This shows that the javax.swing library is at the centre of most the design work which helped to build the overall GUI for the CBabyBallBounce application.

**Import java.util\*;** - This java library is mostly concentrated on the utilities used throughout the java application. The author hasn’t really included anything for this java library except for a random feature which isn’t being used but has been declared.

**Import javax.swing.event.ChangeEvent;** - According to Oracle docs, the change event is where it notifies certain objects in the java application that one of the events is changing in the “event source” part of the code.

**Import javax.swing.event.ChangeListener;** - The main purpose of a change listener is that it changes the value of an objects and or component within the java application.

**CBabyBallBounce – Declaring and the frame**

The next part of the overall design and construction of the java developed game is to declare all the variables as “private” which means that the only time those private methods can be seen is through one of the declared classes either in the GUI (graphical user interface) itself and or the other methods declared in the java developed game. The “int” part on some of the variables is basically stating that it must declare an integer value i.e. a number value only so that the game can operate correctly. There are a number of elements and objects which have been declared at the top of the java code including the different types of buttons, the array of buttons (which are in the central panel), each of the individual panels, the slider, each of the individual labels (or plain text on the application), the text fields (which are used to input data into), the various different icons (which are basically the images for the game), the 3 timers (timerSecs, timerball & timerbaby) for each of the events in the program, the ticks for the digital timer, the menu bar with its menu items, the exact position of each of the babies, whether the babies have detected the edge, declaring the run/pause integers so that the buttons are detected depending on the time you pressed the button and declaring nGame which basically solves the problem of the different options within the game such as the 2 player, 4 player and multiplayer buttons/babies and depending on the button pressed depends on how nGame is set i.e. for 4 player nGame is set to the integer of 2 so that when the mouse’s clicks on the 4 player button it automatically brings up those options/settings. The whole purpose of declaring the variables is so that they are set in stone and are the building blocks for the GUI (graphical user interface).

For the main method part of the java application, the author had to set out the relevant framework for the CBabyBallBounce application itself. There were a number of objects or building blocks which needed to be set so that the application met the required specification which the client produced. The client has specifically asked the author to set the frame size of the application by 825x585 as stated in this quote “Create a JFrame application, which opens to the set size (825\*585). Therefore, based on the clients feedback the author went about and designed the whole application to comply with that set size as well as making sure that all of the panels and other GUI elements fitted within the size of the application itself. The author is now going to point out what each of the code elements is used for in the main method:

**CBabyBallBounce frame = new CBabyBallBounce** – This code calls and declares the application by its name (CBabyBallBounce).

**Frame.setSize(825, 585)** – This code is basically setting the overall size of the java application by 825\*585 as mentioned above.

**Frame.createGUI();** - This code is for setting the actual graphical user interface of the application.

**Frame.setVisible(true)** – This code makes sure that the end-user can see the frame being displayed on the screen.

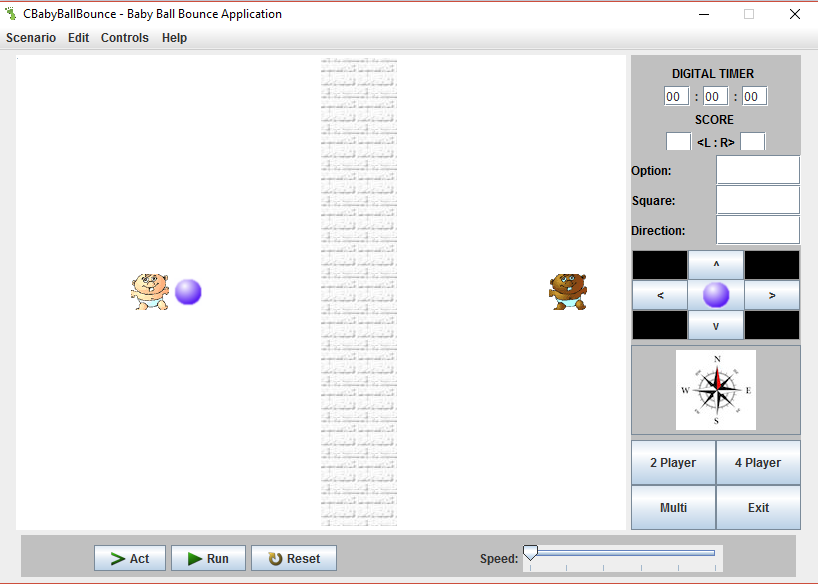
**Frame.setResizable(false)** – This code basically stops the end-user from resizing the application as the client specifically stated that they didn’t want the end-user to make the application bigger or smaller “Create a JFrame application, which is not resizable”.

**Frame.setLocationRelativeto(null)** – This code sets the application to the centre of the computer screen no matter how big or how small the screen size is. The client specifically stated that they wanted the application to be in the middle of the screen “Create a JFrame application, which centres itself on the monitor”.

**Frame.setTitle(“CBabyBallBounce – Baby Ball Bounce Application”)** – This code basically sets the title of the frame to CBabyBallBounce – Baby Ball Bounce Application. This is to make sure that the frame's title matches up with the mock version of the application.

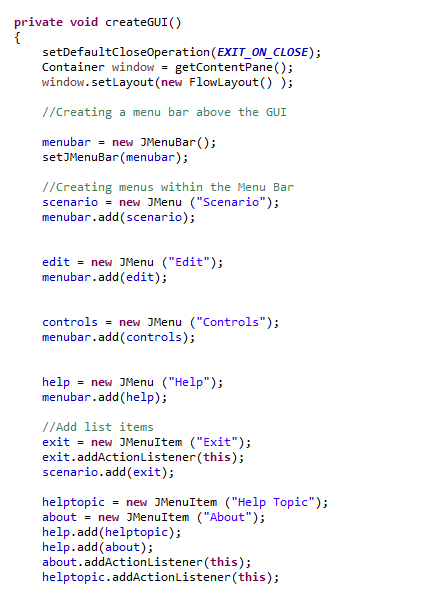
**Frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE)** – This code allows the user to exit the application when they close the application down.

**Frame.setIconImage(Toolkit.getDefaultToolkit().createImage(CBabyBallBounce.class.getResource(“greenfoot\_1.png”)))** – This code basically creates an image icon onto the frame application itself from the image file source and then sets that icon on the top left-hand corner of the java application.



1. Naming the frame CBabyBallBounce.
2. Setting the size of the frame.
3. Setting the GUI (graphical user interface).
4. Shows the application on the screen.
5. The java application isn’t resizable.
6. Setting the frame in the centre of the screen.
7. Setting the frame's title to say “CBabyBallBounce – Baby Ball Bounce Application”.
8. Making sure the application closes when the end-user presses the exit key.
9. Setting the logo of the game to promote the game and its brand.

### Figure 1.3s: Emulation - CBabyBallBounce.java Application – Opening State.



This code basically creates the menubar at the top of the JFrame panel with “Scenario”, “Edit”, “Controls” and “Help”.

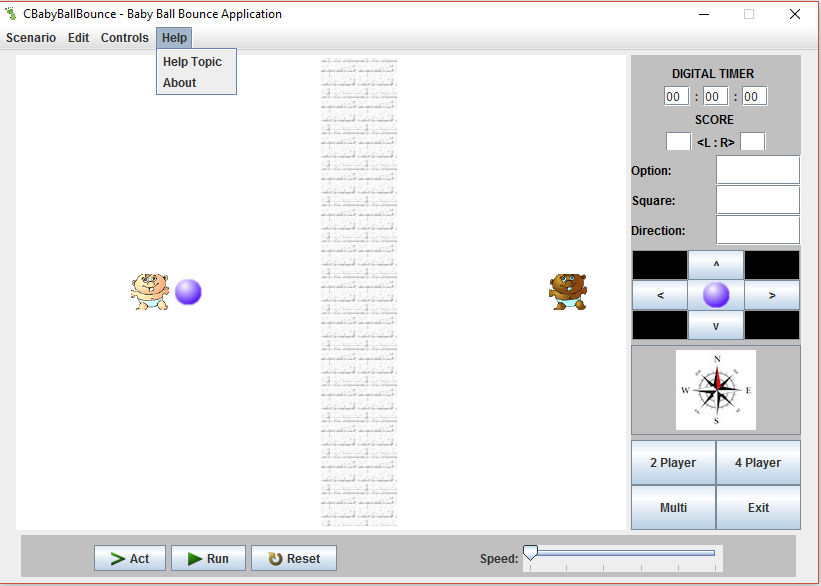
This code basically creates the individual items for the menu bar which is at the top of the JFrame. These items include “Exit”, “Help topic” and “About”.

The next stage of the application is the design stages of the application. To start off with the author must declare the createGUI() method in order to put all the graphical components onto the application frame that was initially created at the beginning. The author then decided to create a menubar with menu items to represent the planned design that the client provided. The client specifically said to create “A JMenuBar could be included with JMenu’s for the Scenario, Edit, Controls and Help, which include JMenuItem’s of Exit(Scenario), Help Topic and About (Help)”. Based on this assumption the author has created the following code:

**setDefaultCloseOperation(EXIT\_ON\_CLOSE)** – This code basically means that the JFrame application will close fully rather than partially meaning that the program doesn’t run in the background even after closing the JFrame application.

**Container window = getContentPane();** - This code basically gets a hold of the JFrame’s centre part of the frame and then allows the programmer to add various different graphical objects to the panel part of the JFrame. This allows graphical components to be added onto the JFrame application layer by layer.

**Window.setLayout(new FlowLayout();** - This code is essentially defining the actual layout of the content. So, for the “flow layout” part of the code, it is basically defining how the elements are structured in a flow rather than the elements being squished together.

**Resulting GUI for the JMenu bar and the JMenuItem’s**

Here is the JMenu bar with one of the JMenuItem’s listed.

**Adding the images to the GUI application**

This code deals with adding the babies image icons to the CBabyBallBounce java application. When one of the icons doesn’t work, or doesn’t appear on the application then it will produce an error message saying “Baby1 icon imageicon” etc.

Adding the green foot icon to the JFrame application itself and then producing an error message “Green foot icon imageicon” if the image icon doesn’t show.

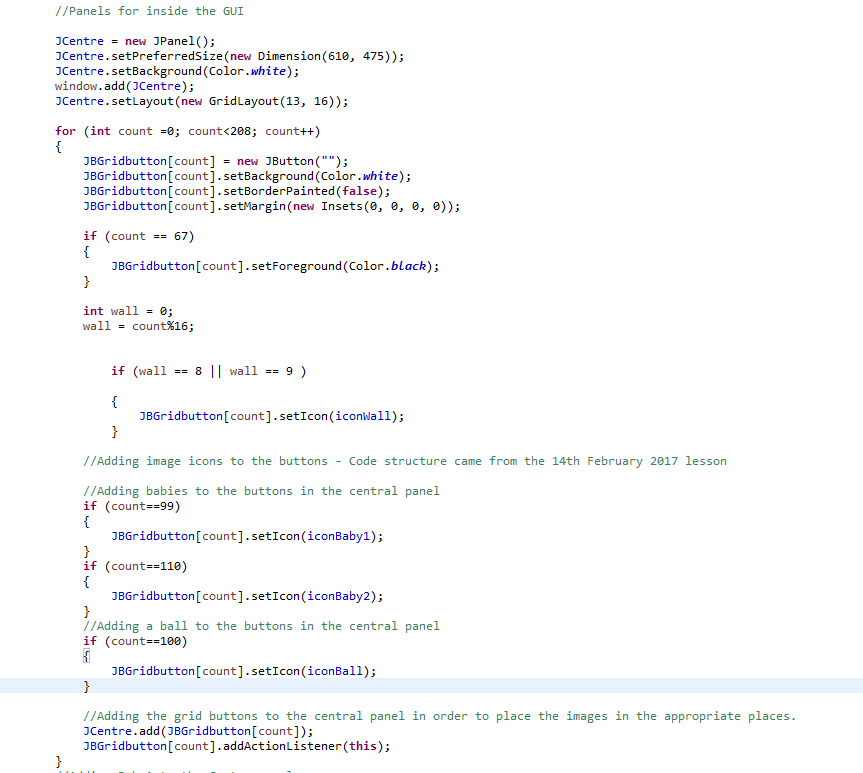
Adding the pause icon to the java application and then producing an error message “Pause Icon imageicon” when the image icon doesn’t show.

Adding the wall icon to the java application and then producing an error message “Wall icon imageicon” when the image icon doesn’t show.

Adding the balls image icon to the java application and then producing an error message “Ball icon imageicon” when the image icon doesn’t show.

**Resulting GUI for the image icons**

Each of the circled components are the images which have been added by the following code above. There are other images which the author will be discussing in the next sections of the code.

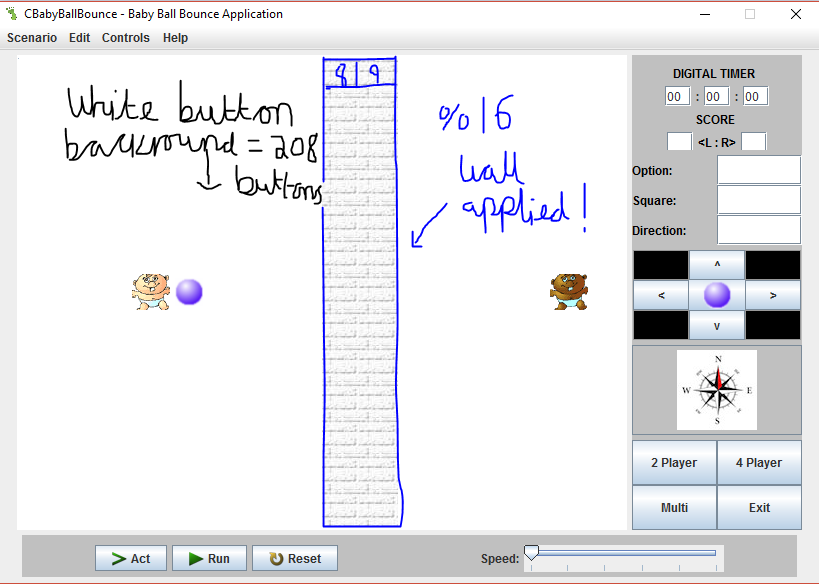
**Adding to the central panel**

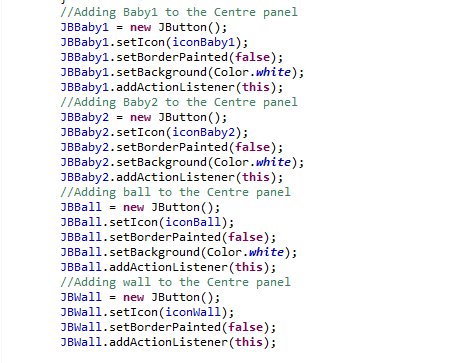
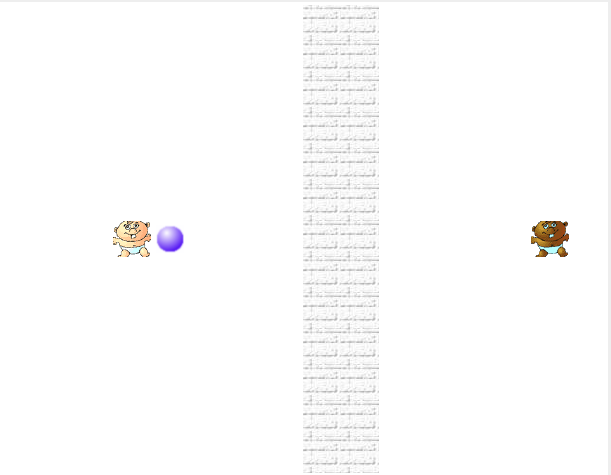
This code is basically creating a new central panel, setting the size to 610x475, setting the background colour to white, adding the panel to the frame and then setting the layout to grid layout so that the elements can flow.

This code is adding both baby icons and the ball icon to the central panel so when the application first fires up the icons will automatically appear on the central panel on the set buttons which are set in the “if” statements. The code below is adding those buttons to the central panel as well as attaching them to an action listener where the buttons allow actions to be performed on them.

This code is basically getting a hold of the edge of the left edge “int wall = 0” and then using modules “wall = count%16” to calculate the total length of the central panels grid buttons from the right edge. Then in-between 0 and 16 there should be buttons 8 and 9 which when set will automatically set the whole of the wall icons down those specific lines starting at 8 and 9 and down towards buttons 200 and 201.

This code is basically getting the whole of the central panel and adding around about 209 buttons (0-208) in total whilst setting the buttons on the central panel to white and setting the margins of each of the buttons to 0. The button on 67 was a test button to see if the ball worked.

**Resulting GUI for adding grid buttons to the central panel**

**How the icons are applied to the java application and the resulting GUI application**

This code basically sets both babies to the central panel. The background is set to match the central panels background (white), an action listener has been applied so the baby can perform actions and the border painted has been set to false which is basically undoing the border around the button of the baby.

This code basically sets both the ball and the wall to the central panel. The background is set to match the central panels background (white), an action listener has been applied so the ball and wall can perform actions and the border painted has been set to false which is basically undoing the border around the button of both the ball and the wall.

**Adding additional panels to the GUI application**

This part of the code is creating a new panel within the long right panel right underneath all the other panels. The set size of the JFour panel is 170x90 with the current background set as black (which will be changed to light grey). The grid layout is set out as 2x2 to represent the 2 different buttons side by side to each other i.e. 2x2 = 4.

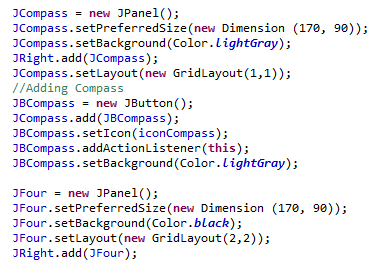
This section of the code is basically creating the compass panel by setting the dimensions to 170x90, setting the background colour to light grey, making sure that button is set to grid layout so that the compass gets aligned in the centre of the button. The compass icon is then set to it with an action listener to change the compasses direction when the ball moves.

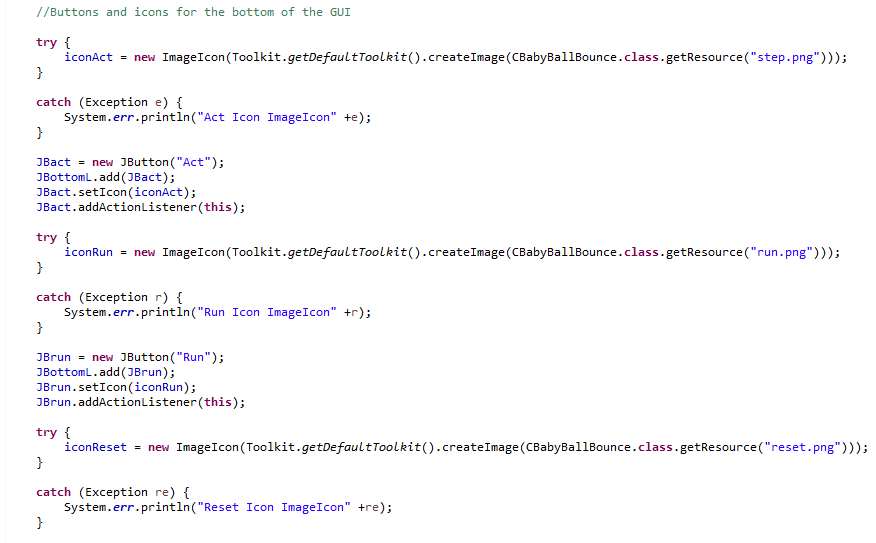
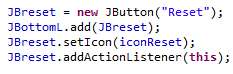
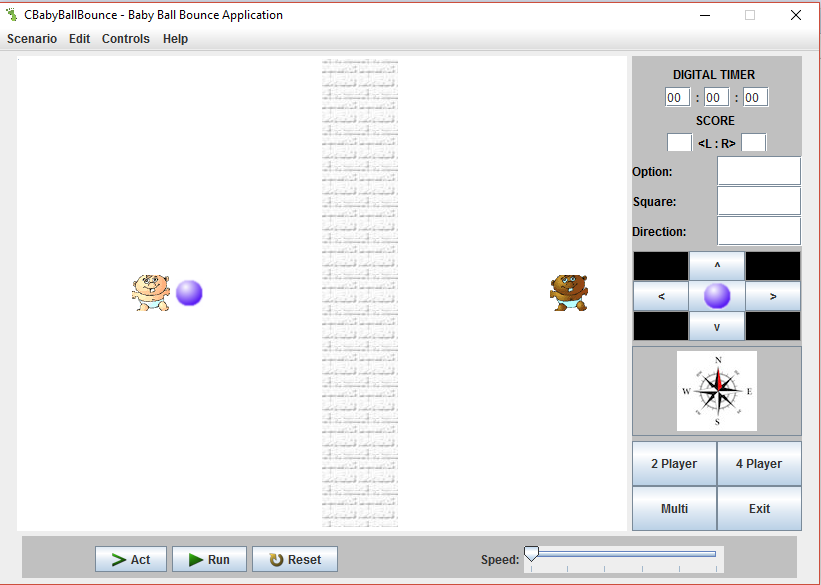
This code is used to add the compass image to the JCompass panel with an error message displaying “Compass Icon imageicon” in case the compass image doesn’t show up.

The JDs panel that was created is the panel for setting the timer with the dimensions of 170x90 and the colour set as light grey. The JOpt panel that was created is the panel for setting the different options and directions of the ball with the dimensions of 170x90 and the colour set as light grey. The JArrows panel that was created is the panel for setting the button arrows too with the dimensions of 170x90 and the colour set as light grey.

The JBottom, the JBottomL and the JBottomR panels are added to the bottom of the java application. The left and right bottom panels are added in the main bottom panel so that the components such as the slider and the buttons are added as they should be rather than all over the place.

The JRight panel is basically setting the long panel which contains the other panels on the right-hand side of the java application at a set size of 170x475 with a light grey background.



**Resulting GUI from adding the additional panels**

This next section of the code is adding the buttons to the bottom left panel as well as renaming the buttons “Act”, “Run” and “Reset”. The buttons are also attached to an action listener to perform actions.

This code is then displaying the error message if the image doesn’t show on the application.

The first bit of the code is getting a hold of the “step.png”, “run.png” and “reset.png” image from the images folder from the assignment 2 folder.

**Adding image icons and buttons to the individual buttons at the bottom left panel**

**JBottom** panel.

**JBottomR** – R for right panel.

**JBottomL** – L for left panel.

**JDs** – D for digital and s for score.

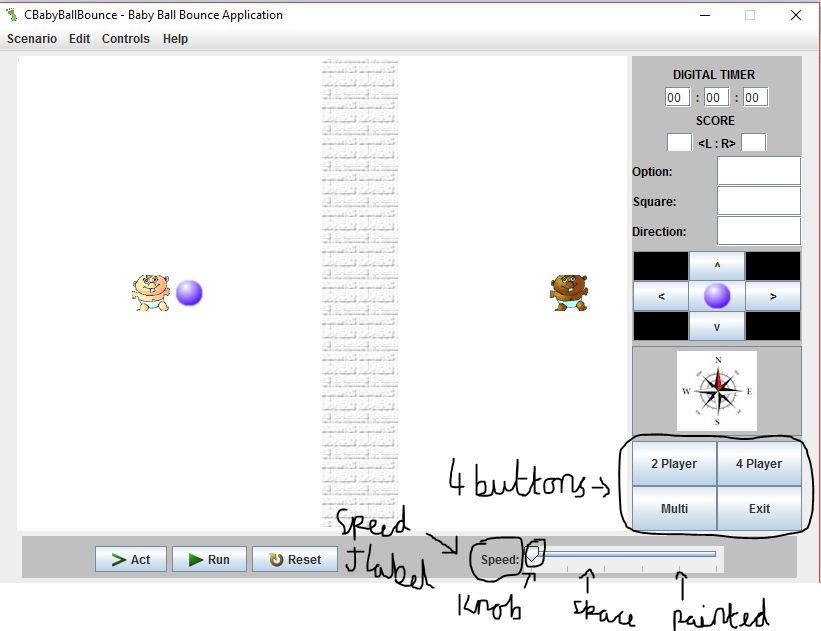
**JFour** – Four for the 4 buttons.

**JOpt** – Opt for option.

**JCompass** – Compass for the compass icon.

**JArrows** – Arrows for arrow buttons.

JRight panel

**Setting the buttons to the JFour panel and adding the speed label and slider to the bottom panel**

This section of the code is basically adding a new label on the left-hand side of the slider but in the same JPanel as the slider which is in the JBottomR. The reason the label is first is because the label needs to be on the left of the slider rather than on the right-hand side of the slider as that was set out specifically in the client’s brief.

This section of the code is essentially declaring and creating the new slider onto the application. The first part is making sure that the slider is set horizontally by 1000x1000 on the JBottomR panel. The setInterverted part basically means to set the knob on the slider to correspond to the ticks/lines on the slider. The setMajorTickSpacing is the spacing inbetween the ticks/lines and the setPaintTicks code is to set the ticks/lines on the slider.

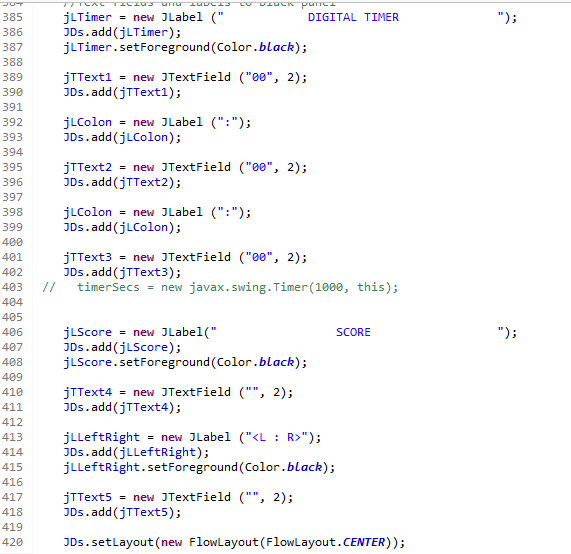
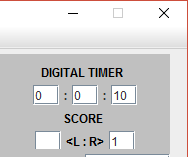
For this next section of the code the 4 buttons had to be placed on the JFour panel. To do this the author had to create 4 new buttons labelled “2 player”, “4 player”, “Multi” and “Exit”. The next step was to add each of those 4 buttons onto the JFour panel and then after that place an action listener onto the buttons so that when a certain button is pressed it performs a different action.

**Adding content to the JDs panel**

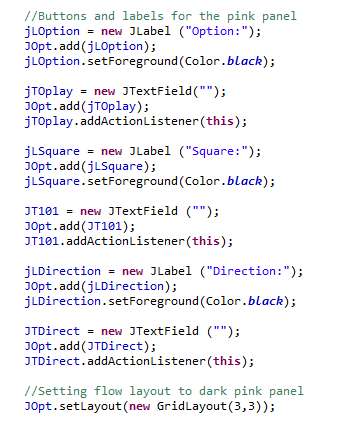
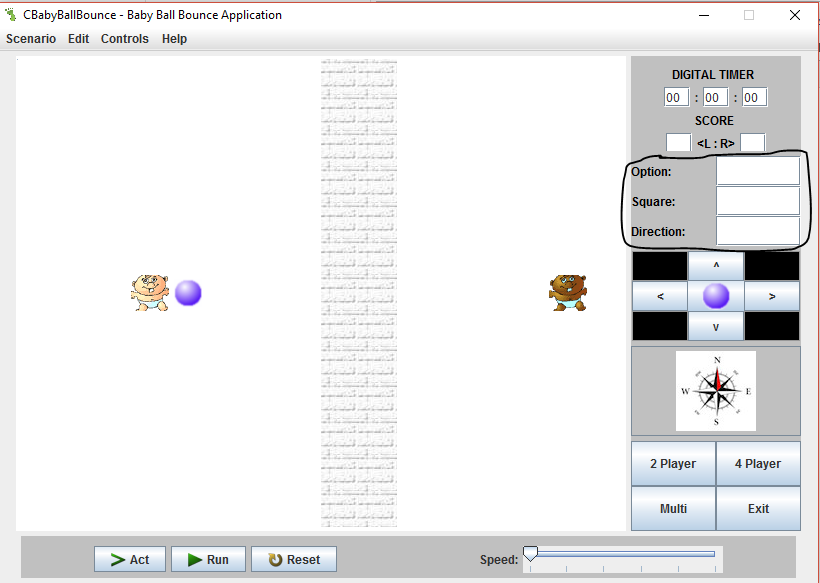
In this section of the code the author is using the same panel as the code above, however, this code is setting up the scoring system rather than the timer. The first part to initiate the scoring system is to label the scoring system “SCORE” as a label. The next parts of the code need to set out the text fields that will hold the score for each of the player’s sides and this is marked as a blank string. The layout will be set to flow layout but in the centre. This means that the objects within the JDs panel will structure as they should do within the panel.

For the first part of the code the author needs to name the jLTimer label “DIGITAL TIMER” and add it to the JDs panel. Then set the colour of the text to “black”.

For the next part of the code the author needs to structure out the text fields and labels so that they are in the correct order when being displayed on the application. They should be in the exact format as the one specified in the client’s brief. The timer should perform in these text fields.



Here is an example of a section of the GUI (graphical user interface) of the digital timer, score, colons and <L:R> labels as well as how the numbers work inside the text fields which are done through the action performed method. All of this is inside the JDs panel.

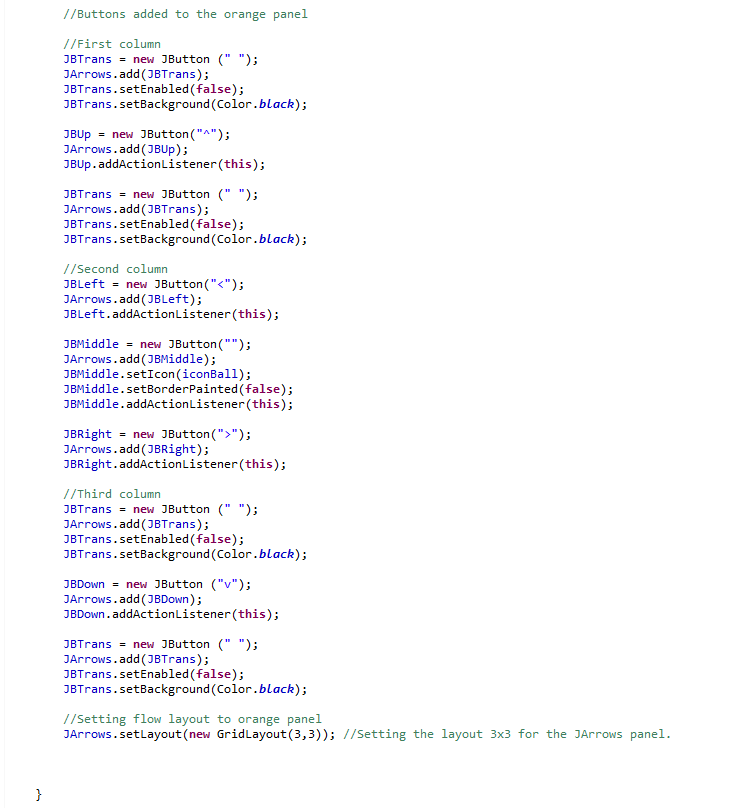
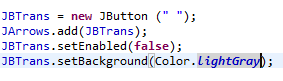
**Adding content to the JOpt panel**

This is the JOpt panel. These are the 3 different labels and 3 different text fields which have been added to the panel using the grid layout 3x3.

This part of the JOpt panel code is basically adding the relevant objects to that panel. The first part is adding the “Direction” label to the panel and changing that labels font colour to “black”. Then adding a text field next to that label with an implemented action listener to perform the relevant actions. The last bit is setting the layout of the panel as a grid layout by 3x3.

This part of the JOpt panel code is basically adding the relevant objects to that panel. The first part is adding the “Option” label to the panel and changing that labels font colour to “black”. Then adding a text field next to that label with an implemented action listener to perform the relevant actions.

This part of the JOpt panel code is basically adding the relevant objects to that panel. The first part is adding the “Square” label to the panel and changing that labels font colour to “black”. Then adding a text field next to that label with an implemented action listener to perform the relevant actions.

**Adding to the actual buttons within the JArrows panel**

Resulting GUI

The JBTrans button is basically the outer button of the arrow buttons. The setEnabled(false) code is stating that the end-user can’t press the JBTrans button for that area of the panel. The background of the button is black although it can change to light grey if need be.

The JArrows panel has been set so that the buttons are in the format of a grid layout. This means that there is a total of 3 buttons on one row, 3 buttons on another row and another 3 buttons on the final row.

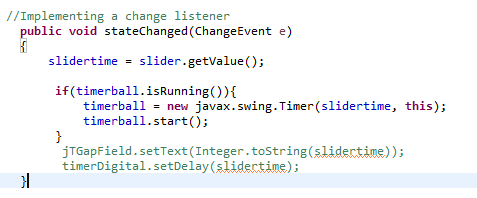
The JBLeft button has been created to produce the left-arrow key in the JArrows panel. To do this the author needs to declare what string is going to be used on that button, in this case it is the “<” arrow. The left arrow is then placed on the JArrows panel with an implemented action listener so the button can perform the left-movement actions.

The JBMiddle button doesn’t have a set string value to it (left blank) and is added to the JArrows panel just like the other arrow keys. However, the only difference is that it has an icon added to the middle of it known as the iconBall. There isn’t a border around the button as it has been set to false but it does have an implemented action listener to perform actions.

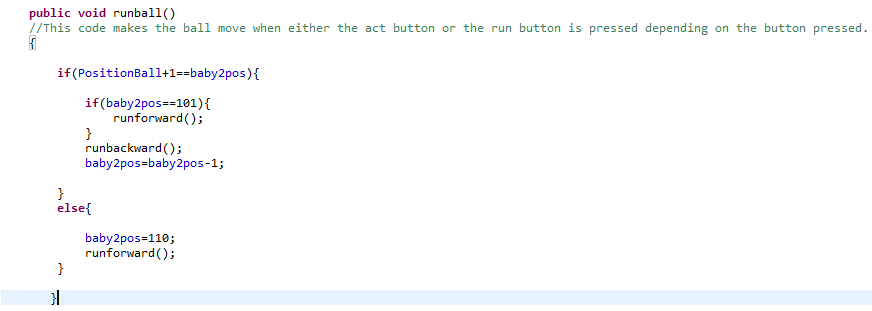
The JBRight button has been created to produce the right-arrow key in the JArrows panel. To do this the author needs to declare what string is going to be used on that button, in this case it is the “>” arrow. The right arrow is then placed on the JArrows panel with an implemented action listener so the button can perform the right-movement actions.

The JBDown button has been created to produce the down-arrow key in the JArrows panel. To do this the author needs to declare what string is going to be used on that button, in this case it is the “V” arrow. The down arrow is then placed on the JArrows panel with an implemented action listener so the button can perform the down-movement actions.

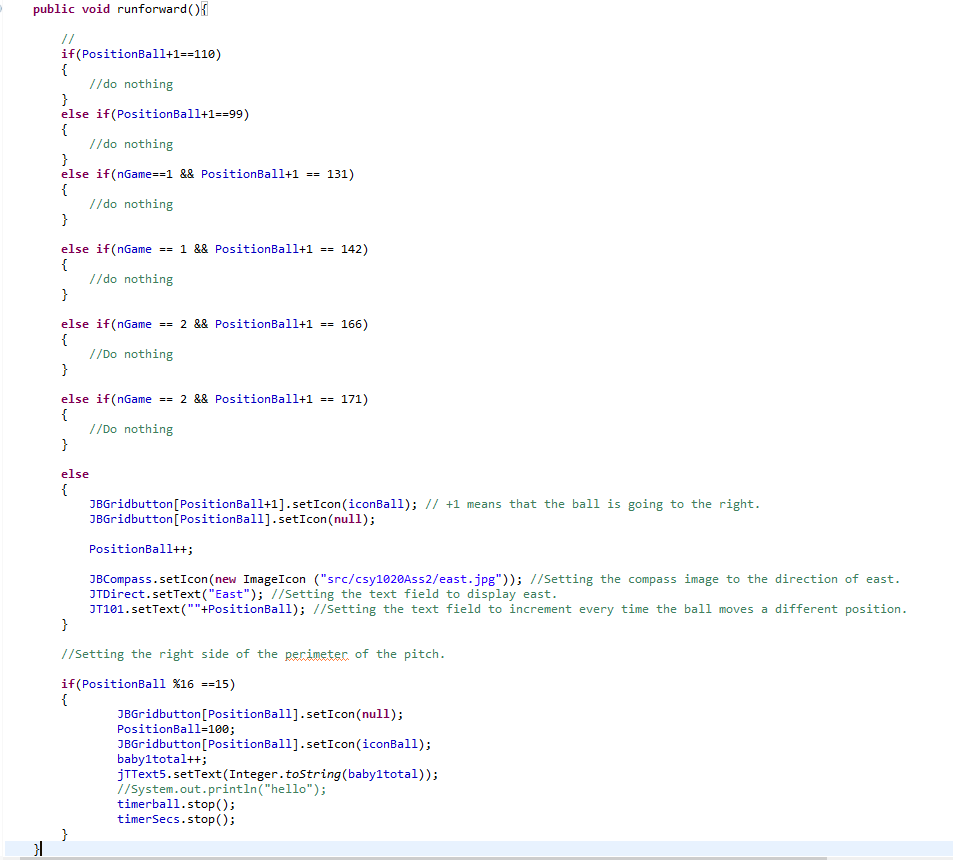
The JBUp button has been created to produce the up-arrow key in the JArrows panel. To do this the author needs to declare what string is going to be used on that button, in this case it is the “^” arrow. The up arrow is then placed on the JArrows panel with an implemented action listener so the button can perform the up-movement actions.

**Implementing a change listener and starting all of the timers**

The main purpose of a change listener is that it changes the value of an objects and or component within the java application. This section of the code is basically changing the value of the timer which is currently set at 0 for all the timers to whatever value the end-user sets it to depending on the action performed. The slidertimer is for the slider and the timerball is for moving the ball back and forth between the babies. The slider will use the ticks as the change event and the ball will use the main timer as the change event.

 **Moving the ball between baby 1 and baby 2**

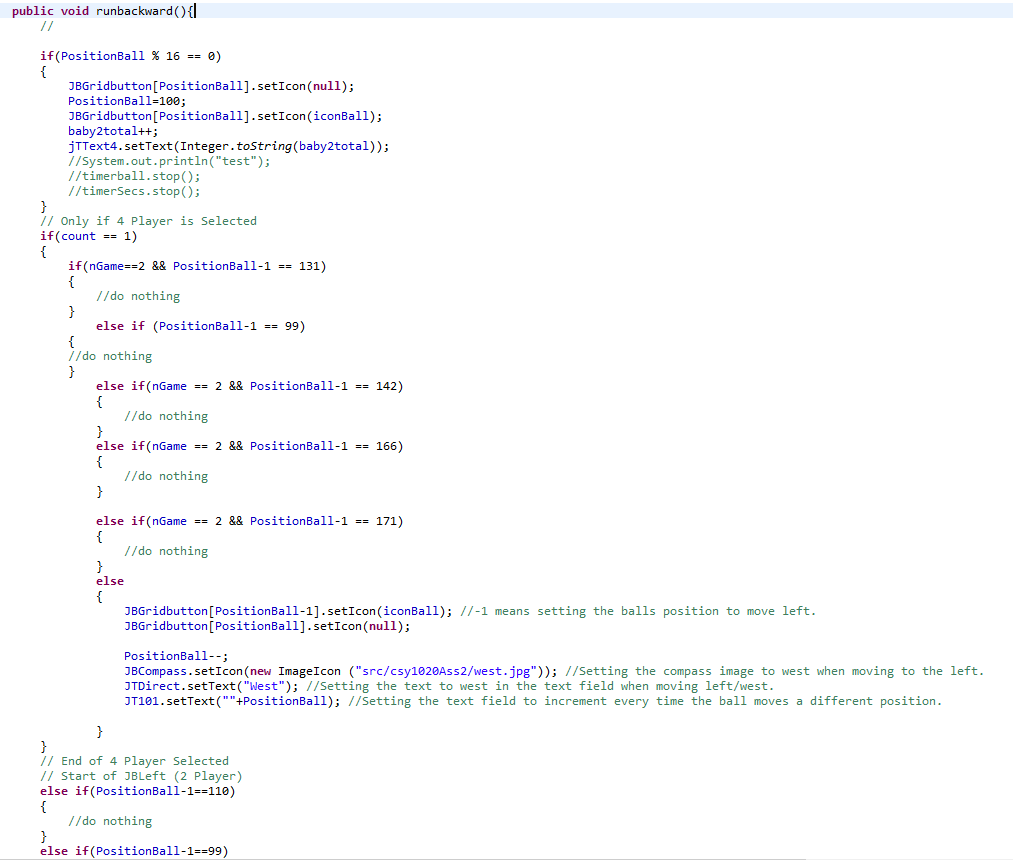
The first part in making sure that the ball moves between both babies (baby 1 and baby 2), the author firstly needs to declare a method which in this case is called “runball”. The author then needs to calculate the current balls position against baby 2’s position and make sure that the balls current position is incremented by 1 each time the act button is pressed or when the run button is pressed as well as being equal to the baby 2’s position. The next part of the code is stating that if the ball reaches baby 2’s position at 101 then it must move forward using the runforward method. If the ball has reached baby 2’s current position then it must get a hold of the runbackward method which basically makes the ball move backwards from the current baby’s position (in this case baby 2’s position) and minus it by 1 each time rather than adding it by 1. If the balls position isn’t at the baby 2’s current position and is closer to baby 1 then the runforward method is called again to make the ball move forwards back to baby 2 again.



For this part of the code the author has tried setting the right side of the perimeter of the pitch so that when the position of the ball touches the edge of the screen (%16) the position of the ball rebounds back to the original balls starting position. The timers are then stopped so that the timer on the ball and on the main timer are synced together in time so that the timers can then restart back to the beginning from 0.

In this part of the code there are 2 icons being set to both the grid buttons in the central panel which is the ball icon and on the compass button which is setting the compass image to the “east.jpg” image. The text is then being set to “East” in the direction text field and the position of the ball is being calculated and then displayed in the square text field.

In this section of the code the ball is moving to the right-hand side (or moving forward). A lot of the code in the runforward method was originally in the right arrow button. But then it made logical sense when building the application that the code should be put in the runforward method and then get called in the JBRight button just to shorten the code down a bit rather than repeating the same code twice. As shown in the runforward method whenever the position of the ball reaches one of the icons it must rebound or stop at that position. For example, baby 1 is at 99 whilst baby 2 is at 110. When the ball reaches the square before it must stop on that square. The same applies to all the other baby icons i.e. 131, 142, 166 and 171.



In this part of the code there are 2 icons being set to both the grid buttons in the central panel which is the ball icon and on the compass button which is setting the compass image to the “west.jpg” image. The text is then being set to “West” in the direction text field and the position of the ball is being calculated and then displayed in the square text field.

In this section of the code the ball is moving to the left-hand side (or moving backwards). A lot of the code in the runbackward method was originally in the left arrow button. But then it made logical sense when building the application that the code should be put in the runbackward method and then get called in the JBLeft button just to shorten the code down a bit rather than repeating the same code twice. As shown in the runbackward method whenever the position of the ball reaches one of the icons it must rebound or stop at that position. For example, baby 1 is at 99 whilst baby 2 is at 110. When the ball reaches the square before it must stop on that square. The same applies to all the other baby icons i.e. 131, 142, 166 and 171.

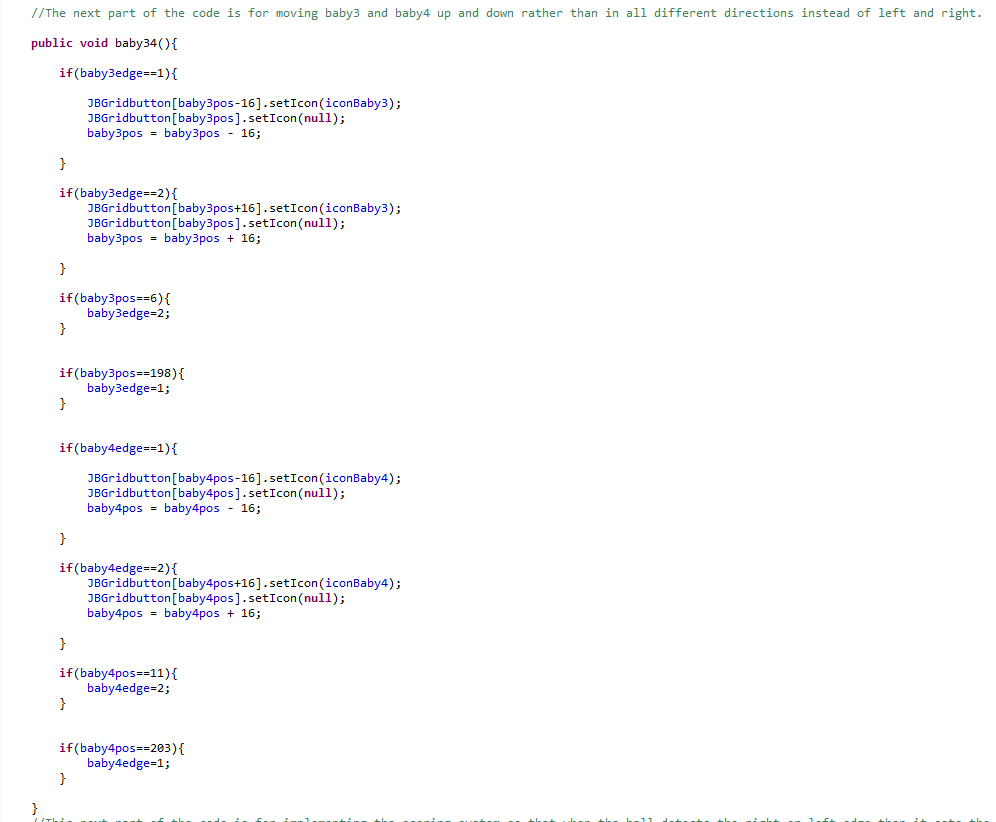
For this part of the code the author has tried setting the left side of the perimeter of the pitch so that when the position of the ball touches the edge of the screen (%16) the position of the ball rebounds back to the original balls starting position. The timers aren’t being stopped until the very end of the next code where the code is repeated but with the added ball and main timer being stopped.



In this part of the code there are 2 icons being set to both the grid buttons in the central panel which is the ball icon and on the compass button which is setting the compass image to the “west.jpg” image. The text is then being set to “West” in the direction text field and the position of the ball is being calculated and then displayed in the square text field.

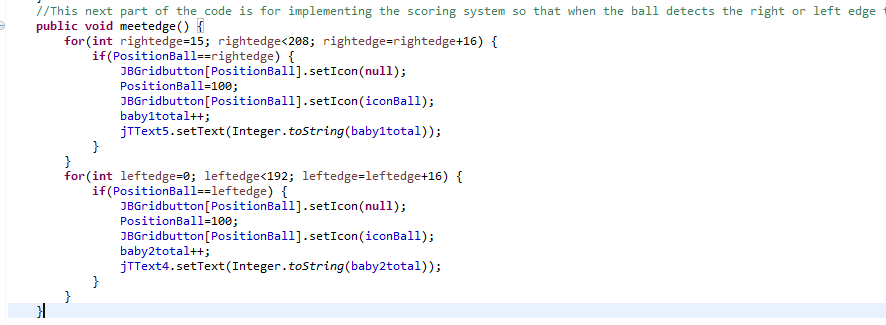
For this part of the code the author has tried setting the left side of the perimeter of the pitch so that when the position of the ball touches the edge of the screen (%16) the position of the ball rebounds back to the original balls starting position. The timers are then stopped so that the timer on the ball and on the main timer are synced together in time so that the timers can then restart back to the beginning from 0.

In this section of the code the ball is moving to the left-hand side (or moving backwards). A lot of the code in the runbackward method was originally in the left arrow button. But then it made logical sense when building the application that the code should be put in the runbackward method and then get called in the JBLeft button just to shorten the code down a bit rather than repeating the same code twice. As shown in the runbackward method whenever the position of the ball reaches one of the icons it must rebound or stop at that position. For example, baby 1 is at 99 whilst baby 2 is at 110. When the ball reaches the square before it must stop on that square. The same applies to all the other baby icons i.e. 131, 142, 166 and 171.

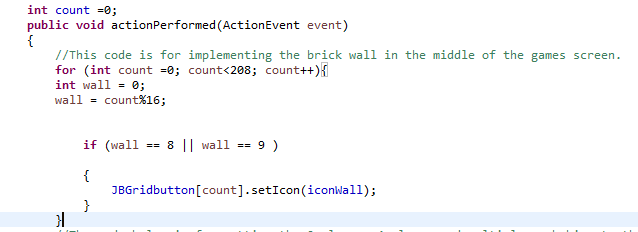


In this section of the code the multi babies that have been set to the application are known as baby 3 and baby 4. This part of the code deals with the multi babies (baby 3 and baby 4) going up and down in front of the other babies. Originally the plan was to have the ball bounce off those babies but due to time constraints the author decided it was best to leave it be and include that as an improvement to the application. The first part of the code is basically stating that when baby 3 reaches the edge of the application that the icon must still be set to the baby 3 icon but is then set to null when the movement is taking place as there is no fixed position for that baby at the time. The -16 part of the code is stating that baby 3 needs to be moved down until it reaches 198 then it must repeat the baby3edge = 1 code again to move baby 3 back up. The +16 part of the code is stating that baby 3 needs to be moved upwards until it reaches position 6 and then repeats the code for moving the baby downwards.

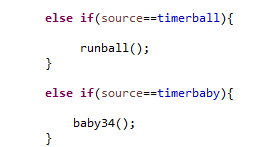
Regarding baby 4’s edge and position it is applying the same method as described above with baby 3 but with baby 4 moving up and down. However, the only difference is that baby 4’s position when going up needs to reach to position 11 rather than 6. When baby 4 is going down it needs to reach the 203 position rather than 198 before calling the baby to repeat itself i.e. go up again. It is basically using a continuous loop of the babies going up and down.

**Scoring System for when the ball touches the edge of the application**

For this part of the code the author would like to implement a scoring system so that when the ball detects either the right edge of the screen or the left edge of the screen then the ball must bounce back to its initial starting position. The first thing that must be done is to declare a method to implement the scoring system into the application. In this case, the author has decided to call the method “meetedge” as when the ball meets the edge it counts the score. The next part is the use of a “for loop” to determine which edges will be used to determine the scoring. For the right edge, the top right hand corner is at 15 and the bottom right hand corner is at 208 which means that these numbers are used in the for loop to detect the right edges. For the for loop in the left edge side the 0 part means the top left position and the 192 part means the bottom left position of the application. The statements after the for loops need to include an if statement i.e. if(PositionBall==rightedge) or if(PositionBall==leftedge) is saying that if the balls position is at that edge of the screen then the balls position must be reset back to 100 (PositionBall = 100). When the balls position is set back to 100 (the original position of the ball when starting up the application) then the balls icon must be set back to how it was. The baby1total++ and baby2total++ are variables which are used to calculate the overall score and then JTText4.setText etc. basically sets that calculated score total onto one of the score text fields.

**Setting the wall & buttons in the CBabyBallBounce java application game**

The action performed method basically means that any movements or actions that you want to happen in the CBabyBallBounce game need to all be based in the action performed method so that the objects in the java application can move. The int count = 0; code means that the current grid buttons are being set from 0 to 207 with 0 being the starting integer point. For the “for loop” the count grid buttons are being set from 0 to 208 in total on the central panel. The bricks themselves need to be set to 0 as well “int wall = 0;”. Then the bricks and the count are calculated together along with the if statement so that the wall (or bricks) go all the way down starting from 8 and 9 down to the bottom of the application with the icon being set to “iconWall”. This allows the wall to go straight down rather than implementing a ton of code for each brick.

**Setting the 2 players, 4 player and multiplayer babies to the java application**

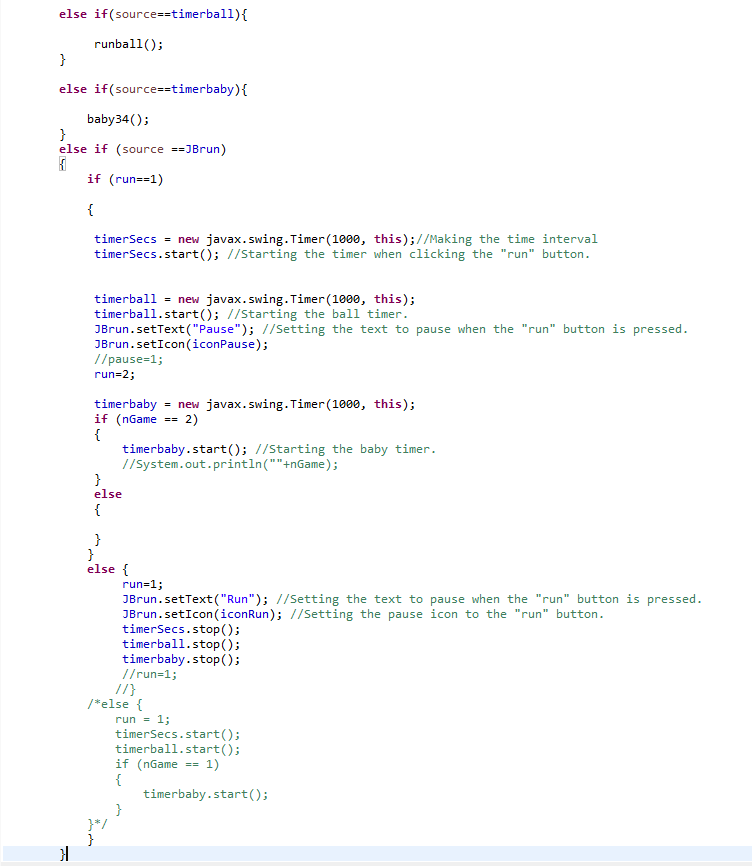
The timerball method is basically the method used to time the ball against the specific ball timer which has been set and declared at the top of the application. In the open brackets the “runball();” method has been declared so that it runs through the relevant steps in the runball method which the author has described about previously. The timerbaby method is basically the method used to run a timer on the babies. The babies which need the timer are the multi babies which move up and down on the application which in this case is “baby34();”.

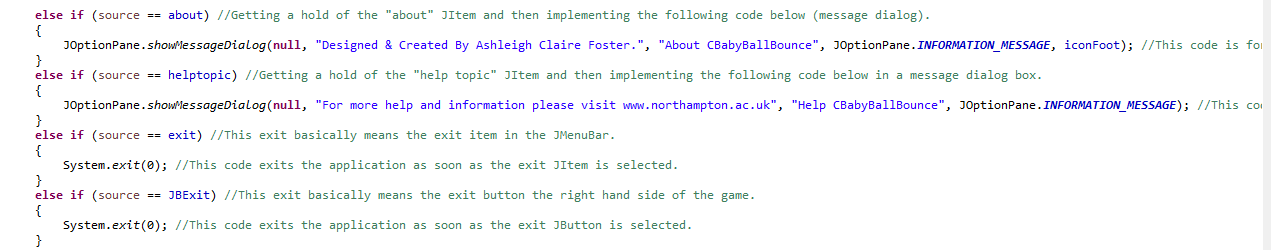
This part of the code is for setting the 2 player, 4 player and multiplayer babies to the CBabyBallBounce java application. The object source = event.getSource(); code basically means that it is getting a hold of the object i.e. the baby and then an “event” is taking place i.e. the baby being set to the java application and then finally getting a hold of the source with the “getSource” element. The code basically activates when you press a button. The “nGame” part in each of the sections of the code is basically getting a hold of the button and then setting a specific integer that depending on the integer will set the appropriate babies. For example, nGame = 0 will set the 2 player babies to the 2-player button. The jTOplay text field is then setting the text in that text field depending on the button which has been pressed in the java application. The JBGridbutton code then sets the babies to a specific part of the java application depending on what grid number has been set. For example, players 3 and 4 have been set to 142 and 131. The source == is basically saying that when that button is pressed perform the relevant actions in the brackets.

**Setting the run button and its relevant actions to the java application**

This code is basically setting the run button to perform the relevant actions which are contained within the brackets. The “if (run==1)” code is stating that if the run button is pressed then start the whole of the action performed method i.e. in the ball moving between the babies as well as starting the timer “timerSecs.start()”. The “run=2” method is stating that when the integer has been set to 2 that it must pause the game and detect that value and then set the text in the button to “Pause” and the icon in the button to the pause icon “iconPause”. When nGame is set to 2 rather than 1 i.e. nGame = 2 is for the multi button & babies then start the baby timer “timerbaby.start();” so that it corresponds to with the multi babies and the multi button.

To set the run button back to running the ball and the babies the “run =1” needs to be declared, then the run button needs to have the text set as “Run” with the run icon set to it rather than the pause icon. Once this has been reset then the timers need to be stopped first i.e. timerSecs.stop(), timerball.stop() and timerbaby.stop() so that they can be reset back to the beginning i.e. so that the application can run again. This means that the program won’t run again until all the timers have been started i.e. timerSecs.start(), timerball.start() and timerbaby.start().



**Setting the pop-up dialogue boxes and the exit buttons**

This section of the code is basically setting the pop-up dialogue boxes and the exit buttons in the CBabyBallBounce java game application. For the first part of the code it is saying:

*else if (source == about)*

*{*

*JOptionPane.showMessageDialog(null, “Designed & Created by Ashleigh Claire Foster.”, “About CBabyBallBounce”, JOptionPane.INFORMATION\_MESSAGE, iconFoot);*

*}*

This code is stating that if the user clicks the “about” item under the “Help” section on the menu bar then show a dialog box which says, “Designed & Created by Ashleigh Claire Foster” with the “About CBabyBallBounce” at the top of the JFrame window. The iconFoot is the icon used as the application icon at the top left-hand side of the screen.

***else******if*** *(source == helptopic)*

*{*

*JOptionPane.showMessageDialog(****null****, "For more help and information please visit www.northampton.ac.uk", "Help CBabyBallBounce", JOptionPane.****INFORMATION\_MESSAGE****);*

*}*

This section of the code is stating that if the user clicks the “helptopic” menu item under the “Help” section on the menu bar then show a dialog box saying, “For more help and information please visit [www.northampton.ac.uk](http://www.northampton.ac.uk)” with the “Help CBabyBallBounce” at the top of the JFrame window. The INFORMATION\_MESSAGE is the icon with an “I” icon.

***else******if*** *(source == exit)*

*{*

*System.exit(0);*

*}*

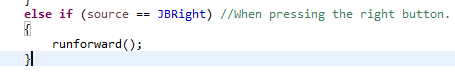
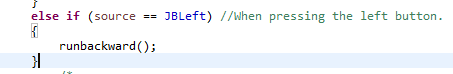
This section of the code is stating that if the “exit” item is pressed under the “Scenario” part of the menu bar then make the application close and or exit i.e. “System.exit(0);”. The same applies for this code but as a button rather than a menu item:

***else******if*** *(source == JBExit)*

*{*

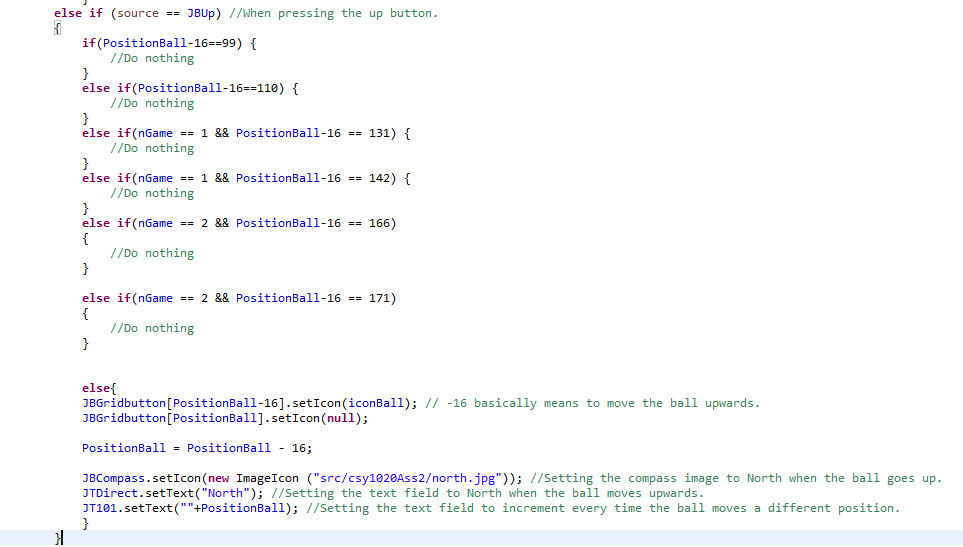
*System.exit(0);*

*}*

**When the right button and the left button are pressed**

For this section of the code the application is getting a hold of the left button “JBLeft” and then calling the “runbackwards()” method which has all the relevant moving left actions as described above in the authors report under the “moving the ball between the babies” section of the report. This method was implemented as a language tool i.e. to prevent repetitive code.

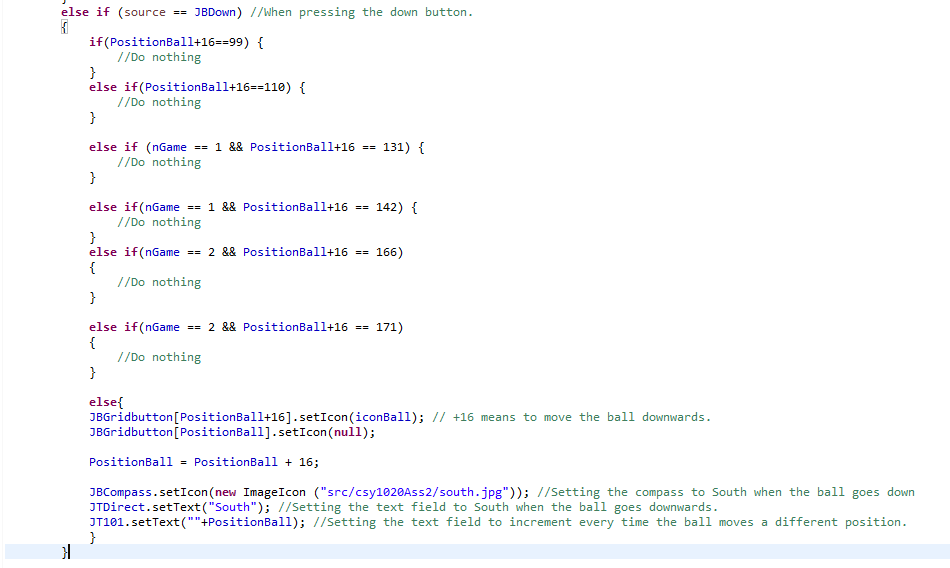
For this section of the code the application is getting a hold of the right button “JBRight” and then calling the “runforward()” method which has all the relevant moving right actions as described above in the authors report under the “moving the ball between the babies” section of the report. This method was implemented as a language tool i.e. to prevent repetitive code.



For this section of the code the -16 part of the code represents the ball going upwards. For the “JBGridbutton[positionball-16].setIcon(iconBall)” part of the code it is setting the ball icon to the grid buttons whilst making sure that the ball goes upwards.

The next part of the code is basically setting the compass up to go to the North. The “JBCompass.setIcon” is basically setting the north compass icon “north.jpg”. The direction text is then set to North as stated in “JTDirect.setText(“North”)”. The position of the ball is then calculated by where exactly the ball is placed on the central panel and then the ball position is set to the “square” text field.

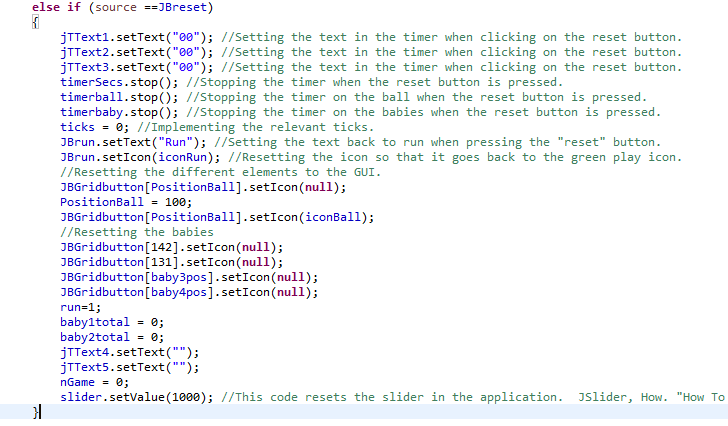
In this section of the code the ball is moving upwards when the up-button arrow key is pressed “JBUp”. In the “position ball” code this part is basically saying that depending on the grid button square in the central panel the position of the ball will stop at that specific square. For example, the “if(PositionBall-16==99” is basically saying that if the position of the ball reaches the grid button 99 when going upwards then it must stop. The same applies for the rest of the grid buttons i.e. if the ball reaches grid button 110, 131, 142 or the 166 when going upwards then it must stop at that grid button square. The reason why nGame == 1 is because these buttons are set as the 4 player babies. The reason why nGame == 2 is because these buttons are set to the multi babies when the multi button is pressed.



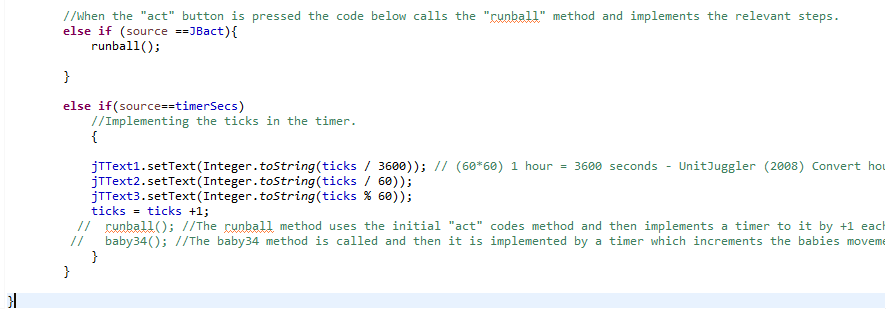
For this section of the code the +16 part of the code represents the ball going downwards. For the “JBGridbutton[positionball+16].setIcon(iconBall)” part of the code it is setting the ball icon to the grid buttons whilst making sure that the ball goes downwards.

The next part of the code is basically setting the compass to go to the South. The “JBCompass.setIcon” is basically setting the south compass icon “south.jpg”. The direction text is then set to “South” as stated in “JTDirect.setText(“South”)”. The position of the ball is then calculated by where exactly the ball is placed on the central panel and then the ball position is set to the “square” text field.

In this section of the code the ball is moving downwards when the down-button arrow key is pressed “JBDown”. In the “position ball” code this part is basically saying that depending on the grid button square in the central panel the position of the ball will stop at that specific square. For example, the “if(PositionBall+16==99” is basically saying that if the position of the ball reaches the grid button 99 when going downwards then it must stop. The same applies for the rest of the grid buttons i.e. if the ball reaches grid button 110, 131, 142 or the 166 when going downwards then it must stop at that grid button square. The reason why nGame == 1 is because these buttons are set as the 4 player babies. The reason why nGame == 2 is because these buttons are set to the multi babies when the multi button is pressed.



In this section of the code it is basically resetting the whole of the application back to its original state i.e. back to how it is when the CBabyBallBounce application first loads up. The first part of the reset is setting the text in the text fields to 00 for each of the 3 text fields so that when the timer is reset it displays the 00’s in the text field. When the reset is button all the timers in the application must stop completely i.e. none of the babies must run or the ball or the main timer. To stop the timers the author needed to set all the timers to .stop i.e. “timerSecs.stop()”, “timerball.stop()” and “timerbaby.stop()”. The ticks = 0 is stating that when the timer is reset the ticks in the slider and the timer must also reset and setting it as 0 basically means setting the ticks back to the beginning. To reset the ball back to its original, position the author firstly needs to reset the icon by setting it to null and then re-add the icon back to iconBall when the position of the ball has been set back to the original square “PositionBall = 100”. To reset the babies back to their original applications state when pressing the reset button the author needs to get a hold of the code from the 2 player, 4 player and multiplayer sections of the code and paste them into the reset button section so that it resets those babies back to when the application first loads i.e. none babies showing up. To do this the author must set each of the babies to “null” so that they don’t show up on the application. The “run = 1” part of the reset button section is basically resetting the run button back to its original state when pressing the pause button on the run button. The baby1total and baby2total parts of the code is basically resetting both the perimeter of the pitch back to the beginning as well as the scoring system back to 0. Once that has done the text in the scoring fields are then set back to null i.e. no value in the string. The nGame part of the reset button section is basically the button presses of the application and this means that when nGame = 0 then the 0 part is resetting those button presses back to the beginning. For the slider.setValue(1000) part of the code it is basically resetting the slider back to 1000 i.e. back to how it was at the beginning when loading up the application. Originally the author set it to 0 but the timer seems to be mixed up and the dial on the slider reset to the left side of the slider rather than the right side of the slider.

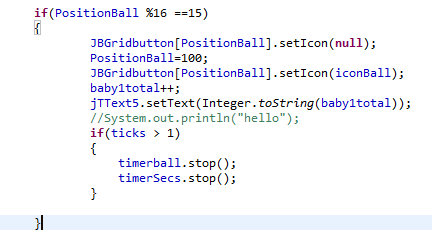
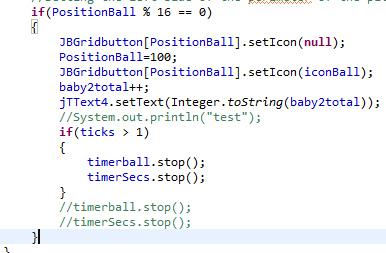
**The code for the act button and the main timer (timerSecs)**

For the timerSecs i.e. the main timer of the CBabyBallBounce java application the first thing the author needs to do is declare the “else if(source == timerSecs)”. This part of the code is basically stating that if the object in the java application is the main timer i.e. timerSecs then produce the following code in the brackets. In the brackets code the 3 text fields are set individually depending on where they are in the application and whether they are for the hours, the minutes or the seconds. For the first text field on the left-hand side of the panel it is setting the hours which means that 3600 seconds are in an hour so jText1.setText(Integer.toString(ticks/3600)) would set the hour’s text field. For the minute’s field (which is in the middle of the hours and seconds field) there are a total of 60 seconds in a minute which means that for the minute’s text field it needs to be setup like “JTText2.setText(Integer.toString(ticks/60))”. For the last text field (which is the second’s text field) there is a total of 1 second i.e. 60/1 = 1. So, for the seconds field the author needs to set it to “JTText3.setText(Integer.toString(ticks % 60)) as the modulus part of the code is basically the remainder of 60 which is 1. The final part of the timerSecs is the “ticks = ticks+1” part of the code. This means that each time the timer changes it is changing by +1 i.e. adding 1 each time the timer changes.

For the act button of the code the “else if (source ==JBact)” is basically saying that if the object in the java application is the act button “JBact” then run the “runball()” method. To see the run ball method, refer to page 29 of the report.

**21/04/17 – Just added additional code to prevent a known error that was occurring in the runforward and runbackward methods**

**Runforward** – The additional part which the author has added to the runforward method is putting the timers in an if statement rather than putting both timers in the if position ball method again. It was throwing up an error on the timerball.stop and timerSecs lines. The error has been solved by using an if statement and making sure that ticks is more than 1 as ticks at the top of the application has been set and declared as 0.



**Runbackwards** - The additional part which the author has added to the runbackwards method is putting the timers in an if statement rather than putting both timers in the if position ball method again. It was throwing up an error on the timerball.stop and timerSecs lines. The error has been solved by using an if statement and making sure that ticks is more than 1 as ticks at the top of the application has been set and declared as 0.

# **5. Testing**

## **5.1 Test Plan**

|  |  |  |  |
| --- | --- | --- | --- |
| **What are you testing?** | **Expected outcome** | **Actual result** | **Any improvements?** |
| Are the 13x16 JButtons placed on the centre panel properly? | The 13x16 JButtons should be placed on the centre panel and there should be a total of 208 buttons. | There is a total of 208 buttons in the centre panel and they are all placed on the centre panel by 13x16. The 13x16 buttons don’t interfere with anything i.e. they don’t increment the timer by 1 each time. | None. |
| Have you added the 4 game option buttons “2 players”, “4 player”, “multi” and “exit”? | There should be a total of 4 game option buttons “2 players”, “4 player”, “multi” and “exit” on the right-hand side panel. | Yes, there is a total of 4 game options buttons on the right-hand side of the GUI labeled “2 players”, “4 player”, “multi” and “exit”. | None. |
| Have you added the 3 buttons for the bottom panel labeled “Act”, “Run” and “Reset”? | There should be a total of 3 buttons on the bottom panel called “Act”, “Run” and “Reset”. | Yes, there is a total of 3 buttons on the bottom panel labeled “Act”, “Run” and “Reset” plus they also have their relevant corresponding icon. | None. |
| Have you added the relevant arrow button keys as well as the blank spaces around it? | There should be a total of 4 buttons with the up, down, left and right arrow buttons as well as 5 blank spaces. | Yes, there is a total of 4 buttons each with the up, down, left and right arrow buttons. There are only 4 blank spaces though. | Could improve it by implementing another blank space which the author is guessing it could be in the ball part of the buttons as the missing blank space. |
| Does the compass icon direction correspond to the position of the current ball? | The compass icon should change depending on the current direction the ball is going. | Yes, the compass icon does change in correlation to the current balls position i.e. if the ball is going right the compass will display the “East” direction. | None. |
| Have you added the 3 JLabels “Option”, “Square” and “Direction” and JTextFields for these options? | There should be 3 JLabels and 3 JTextFields added to one of the panels on the right-hand side of the application. | Yes, there are 3 JLabels and 3 JTextFields added to the application. The “Option” text field can detect which current mode is set in the game, the “Square” text field is able to detect the position of the ball in the centre panel and the “Direction” text field is able to detect which direction i.e. North, East, South or West the ball is going. | None. |
| Have you added the 3 JLabels for “Digital Timer” and the two colons “:”? | There should be a total of 3 JLabels with “Digital Timer” being at the top of the panel and the 2 colons being just below that to demonstrate the time. | Yes, there are a total of 3 JLabels on the top panel with the “Digital Timer” at the top and the two colons which demonstrates the time. | None. |
| Have you added the two JLabels “Score” and “L:R”? | There should be a total of two JLabels on the top right-hand side panel named “Score” and “L:R”. | Yes, there is a total of two JLabels on the top right-hand side panel. | None. |
| Have you made sure that the total size of the JFrame application is 825x585? | The frame size should be exactly 825x585 for the CBabyBallBounce game. | Yes, the JFrame application size is set exactly to 825x585 “frame.setSize(825, 585);” | None. |
| Have you set the JFrames title to “CBabyBallBounce – Ball Bounce Application”. | The application's title should be set to “CBabyBallBounce – Ball Bounce Application”. | Yes, the JFrames title is set to “CBabyBallBounce – Ball Bounce Application” frame.setTitle("CBabyBallBounce - Baby Ball Bounce Application");. | None. |
| Have you set an application icon for the CBabyBallBounce application? | There should be a Green foot icon set to the JFrame application at the top left-hand corner. | Yes, the author has set a Green foot icon at the top-left hand corner of the JFrame application. | None. |
| When you press the run button, does it show the ball moving between the 2 babies? | The button should move between the 2 babies when the run button is pressed. | Yes, the ball does move between the 2 babies when pressing the run button. I did this by declare separate methods and calling methods within the code. | None. |
| When you press the reset button, does it reset the whole application? | The reset button should reset all the required elements back to the beginning. | 3/04/2017 - Yes, the reset button resets the relevant elements. However, there are still problems trying to reset the 4 player and multi-player babies back to their original position. The ball resets without any problems and the 2 player babies are in a fixed position.  10/04/2017 – Yes everything resets back to how it should be. Just need to reset the score and the slider.  11/04/2017 – Yes everything works now. Both the slider and the score counter have been reset and work without any problems. | 3/04/2017 - The author could try and find a solution to reset the 4 player and multi-player babies back to their starting position.  10/04/2017 - The author has managed to reset the both the 4 player and multiplayer babies back into position. The author still needs to reset the score and slider.  11/04/2017 – None. |
| When the act button is pressed does it run through the run sequence one step at a time? | The act button should move the position of the ball by 1 each time between the 2 player babies. | Yes, the act button moves between both 2 player babies without any problems. The ball is incremented by 1 when the “act” button is pressed rather than the ball moving automatically like it does on run. | None. |
| Do the 2 players, 4 players, and multiplayer JButtons display different options when you click on them? | The 2 player, 4 player, and multiplayer buttons should display the relevant corresponding options. | Yes, the 2 players, 4 player and multiplayer JButtons display different obstacles. For example, the 2-player option displays 2 babies, the 4-player option displays 4 babies and the multiplayer option displays another 2 babies but with the babies moving up and down. | None. |
| Is there a JMenubar at the top of the application with JMenus labeled “Scenario”, “Edit”, “Controls” and “Help” as well as the JItems? | There should be a JMenubar at the top of the application with the “Scenario”, “Edit”, “Controls and “Help” JMenus as well as the relevant JItems “Exit”, “Help Topic” and “About”. | Yes, the author has added a JMenubar at the top of the application with the “Scenario”, “Edit”, “Controls” and “Help” JMenus as well as the relevant JItems with “Exit” being under the “Scenario” menu and “Help Topic” & “About” being under the “Help” JMenu. | None. |
| Have you included and or added any additional buttons to the application? | There should possibly be a random direction button on the application to help deter the ball in a different direction. | No there are no additional buttons which have been added to the application such as a random direction button. | The author should have added a random direction button to the application. |
| Have you created a JFrame application which isn’t resizable? | The JFrame application shouldn’t be resizable i.e. when you click on the maximize/minimize square on the application then it shouldn’t make the application bigger or smaller. | Yes, the JFrame application doesn’t resize at all i.e. you can’t make the application bigger or smaller in size. The application is set to a fixed size which is 825x585 which means that it can’t go anywhere bigger or small than that size. The code which I have implemented to centre the application:  frame.setResizable(**false**); | None. |
| Does the JFrame application centre itself on the monitor? | The JFrame application should centre itself on the monitor without any problems. | Yes, the JFrame application does centre itself on the monitor. To do this I simply implemented the following code:  frame.setLocationRelativeTo(**null**); | None. |
| Have you added any additional babies to the application? i.e. more than 2 babies? | The author should have included more than 2 babies on the CBabyBallBounce application. | Yes, the author has included more than 2 babies in the application. The author has done this by added 2 additional babies for the 4-player option and another 2 babies for the multi-option making the total number of babies 6 (3 on each side). | None. |
| Have you incorporated intelligence checks for whether the moves are valid? | The author should incorporate intelligence for whether the moves are valid by allowing the ball to move between the babies and detect each of the babies before turning back to the other baby. The perimeter of the application also needs to be set so that when the ball reaches the edge of the application the ball bounces back to its starting position (grid button 100). | Yes, the author has incorporated intelligence checks for whether the moves are valid. To do this the application has detected that there are at least 2 babies in the 2-player mode. When the ball detects one of the babies it must bounce back in the opposite direction. This is known as collision detection. Yes, the perimeter of application works without any problems. As soon as the ball touches the edge of the application on both the left-hand side and the right-hand side it automatically bounces back to the balls starting position when the application was first fired up (on grid button 100). | None.  30/04/2017 – The perimeter of the pitch works for both the left-hand side and the right-hand side but unfortunately the author didn’t realise that the perimeter also had to be set for the up and down. Unfortunately, due to time constraints the author was unable to implement it but thinks that this could be improved by using the same code as the left and right perimeter but replacing it with %13 rather than %16. |
| Does the digital timer stop and start when the run button is pressed and stopped when the baby misses the ball? | The author should make sure that when the run button is pressed that it starts the digital timer. When the baby or babies miss the ball, it should stop the timer until the run button has been pressed again. | Yes, the digital timer does stop and start without any problems as well as being able to reset the necessary elements. However, the digital timer doesn’t perform how it is supposed to or required too. For example, the digital timer does start when the run button is pressed. However, the timer doesn’t stop when the ball misses the baby as it hasn’t been set up to perform that way. | The author could improve on this section by trying to implement or find a way to make the timer stop and function as required having given more time. |

Figure 1.1: Test Table - CBabyBallBounce.java Test Application

**Additional Testing**

|  |  |
| --- | --- |
| System Requirements: Essential (Graphical User Interface): |  |
| 13 x 16 grid of **JButton**’s or Icon’s. | 🗸 |
| 4 **JButton**’s for the game options ‘2 Player, 4 Player, Multi’ and ‘*Exit*’. | 🗸 |
| 3 **JButton**’s for ‘Act’, ‘Run’ and ‘*Reset*’. | 🗸 |
| 9 **JButton**’s for ‘*Forward >*’, ‘Backwards <’, ‘Up ^’, ‘Down v’ should move the ball in the appropriate direction by one square for each press (plus 5 blank). | 🗸 |
| The compass icon (**JButton)** should illustrate the current direction for the ball. | 🗸 |
| **3 JLabel**’s for ’Option’, ‘*Square*’ and ‘*Direction*’. | 🗸 |
| **3 JTextField**’s for the current ‘Option’, Location/*’Square’* and *‘Direction’* of the ball. Use the square identification method e.g. 0 to 207 and N, E etc. | 🗸 |
| 3 **JLabel’s** for the ‘DIGITAL TIMER and the two :’, with 3 **JTextField’**s for the hours, minutes and seconds. | 🗸 |
| 2 **JLabel’s** for the ‘SCORE and ‘<L:R>’, with 2 **JTextField’**s for the scores (L & R). | 🗸 |
| Create a **JFrame** application, which opens to the set size (825 \* 585). | 🗸 |
| **JFrame** title set as "*CBabyBallBounce – Baby Ball Bounce Application*". | 🗸 |
| System Requirements: Additional (Functionality & Complexity): |  |
| Application icon for the **JFrame** used. | 🗸 |
| The ‘Run’ **JButton** should show the ball moving between the babies continuously from the initial position (2 Player – default opening state). | 🗸 |
| The ‘Reset’ **JButton** should clear/reset the application to its starting/default opening state. | 🗸 |
| The ‘Act’ **JButton** should step through the above ‘Run’ sequence one move at a time. | 🗸 |
| Discuss and implement the different options for the 3 configurations. | 🗸 |
| The ‘2 Player, 4 Player, Multi’ **JButton**’s should display different obstacle/car configurations/locations. | 🗸 |
| A **JMenuBar** could be included with **JMenu**’s for the *Scenario, Edit, Controls* and *Help*, which include **JMenuItem**’s of *Exit (Scenario)*, *Help Topic* and *About (Help)*. | 🗸 |
| Additional **JButton**’s may be used to improve the applications usability e.g. ball bounce – in random direction, deflection angle etc. | 🗴 |
| Create a **JFrame** application, which is not resizable. | 🗸 |
| Create a **JFrame** application, which centres itself on the monitor. | 🗸 |
| Use of additional baby images indicating the current position and direction of the baby. | 🗸 |
| Discuss the possibilities for incorporating intelligence/checks for whether moves are valid. | 🗸 |
| Digital Timer should start and stop when run is pressed and stopped when a baby misses the ball (with the ball continuing to the left or right boundary and stopping itself and the timer). | 🗸/🗴 |
| Implement intelligence/checks for whether moves are valid. | 🗸 |
| A **kickBall()** method should be used to solve the problem. The **kickBall()** method should include **move(left), move(right), move(up), move(down)** methods | 🗸 |
| **CBabyBallBounce.java** & **CBabyBallBounce.class** | 🗸 |
| **Predicted Grade:** | A |

Key: Blue GUI; Yellow Testing Application; Red Code.

# **6. Conclusion & Recommendations**

The main purpose of the CBabyBallBounce java game was to create an application which looked alike and or identical to the Greenfoot application which the author had previously used in assignment 1 except for a few additional changes made to the application (mostly on the right-hand side of the application). The overall purpose of the application is slightly different in that one is used to help beginners design as well as program for the first time whereas the other is an already custom built application. For example, Greenfoot is being used to help individuals who are new to programming to develop a graphical simulation of the game rather than being asked to play a game which has already been created from scratch.

When the author was building the CBabyBallBounce application there were several objectives which the author had to follow including:

**Rules (Basic)** Create a simulation of the ball moving around the pitch, where:

* The ball can move anywhere within the pitch and across the wall in the middle of the screen.
* If the ball touches a baby it is deflected/rebounds back in the opposite direction.
* The ball must move one whole ‘white/wall’ block at a time every time a movement button (via a direction button (<, > v ^)) /key is pressed (when movement is possible).
* The solution must use the scenario provided. i.e. If the babies are left in the unmodified codes starting position the ball would move between them continuously.
* The ball must stop when the perimeter of the pitch is reached.
* The basic solution must be completed using the ‘act’ button (accessing the **kickBall()** method within the **CBabyBallBounce.class**).

**Rules (Intermediate and advanced)** Create a simulation of a ball moving around the pitch, where:

* Your solution must still use the scenario provided (all the basic features above).
* Add appropriate extra features to the solutions, e.g. a) The ball can bounce off a baby or babies in random direction, b) Two new ‘player’ babies are added to each side of the line, that move vertically towards the ball ready to potentially bounce the ball back, c) Add a scoring system for each side (a, b or c for **Intermediate,** a, b & c for **Advanced**).
* For higher grades on the solution part of the assignment see the marking scheme at the front of the brief.
* You must NOT change the layout and all changes should still meet the criteria of **Rules (Basic**).

Based on these objectives the author then had to implement the CBabyBallBounce java game application. The first requirement which the author was required to do was to make sure that “the ball can move anywhere within the pitch and across the wall in the middle of the screen”. To do this the author had to make sure that the arrow keys that were being pressed allowed the ball to be moved anywhere within the central panel as well as when running the java application with the ball moving between the 2 babies which then brings us to the authors next objective “If the ball touches a baby (or babies) it is deflected/rebounds back in the opposite direction”. To do this the author had to make sure that when the run button is pressed and or the act button is pressed that the ball bounces between the babies without any problems. This was done in the java code by creating a “run ball” method which calculated the position of the current ball against the baby 2’s position whilst incrementing the ball by +1 each time the act or run button is pressed. Two other methods have been used in this process such as the “runforward” method and the “runbackward” method. Both methods have been implemented so that the ball moves right when the runforward method is called and moves left when the runbackward method is called. These methods also allow the ball to move when either the left key or the right key are pressed on the arrow buttons and are called in the left and right arrow keys in the action performed method.

This then brings the author onto the next objective which was to make sure that “the ball must move one whole ‘white/wall’ block at a time every time a movement button (via a direction button (<, > v ^)) /key is pressed (when movement is possible)”. To do this the author had to initially create 4 button methods in the GUI and then get a hold the source of those buttons using “source==” to allow the ball to move when one of the arrow keys is pressed as well as using the declared “Position Ball” variable to position the ball by +1 each time. In each of the arrow buttons including the runforward and runbackward methods which are called on the left and right arrows are using the position of the ball and setting certain squares so that when the position of the ball reaches a specific square such as squares 110, 99, 131, 142, 166 & 171 which all contain babies then stop before eating up the baby in that square.

The next objective would be that “if the babies are left in the unmodified codes starting position the ball would move between them continuously”. For this part, the author had to make sure that both baby 1 and baby 2 were set to a “null” position i.e. set to that specific square only to make sure that the ball moved between the babies continuously without moving past them onto the next grid buttons line. This was done using the grid buttons which were set on the central panel.

Regarding “the ball must stop when the perimeter of the pitch is reached” the author had to make sure that the ball stopped moving when the ball had reached either side of the pitch. To do this the author had to create an if statement which stated that if the position of the ball was on the right-hand side “**if**(PositionBall %16 ==15)” (as %16 basically means on side of the application and 15 being the top right-hand corner of the application) then the ball needs to be set back to its original position on the central panel i.e. grid button 100 “PositionBall=100” as well as the icon. The same applied to the left-hand side of the pitch but instead of setting it as 15 it would be set as 0 as 0 is the grid button for the top left-hand corner of the application.

The final part of the basic solution objective is “The basic solution must be completed using the ‘act’ button (accessing the kickball method() within the CBabyBallBounce.class application)”. This means that the act button must work without any problems and that the position of the ball must be moved by +1 each time the act button is being pressed. The ball must also be able to move between both babies 1 and 2 in the java application. The author didn’t, however, call it the “kickball” method in the code.

For the advanced objectives, one of the requirements is to “Add appropriate extra features to the solutions, e.g. a) The ball can bounce off a baby or babies in random direction, b) Two new ‘player’ babies are added to each side of the line, that move vertically towards the ball ready to potentially bounce the ball back, c) Add a scoring system for each side (a, b or c for **Intermediate,** a, b & c for **Advanced**)”. For the first section of the list, it states that “The ball can bounce off a baby or babies in a random direction”. The author hasn’t implemented this section due to time constraints. This method could be implemented into my java application by using the X and Y axis of the wall and or making the method like the score counter method in my java code by detecting the right wall and the left wall using modulus 16 (%16), setting the balls position as well as the balls icon and then finding a method to rotate the ball in a different direction. For the second part of the list, it states that “Two new ‘player’ babies are added to each side of the line, that move vertically towards the ball ready to potentially bounce the ball back”. The author has created the extra 2 lots of babies making a total of 6 babies. Only the multi-babies are set to go up and down (or vertically) in the java application. The only aspect which hasn’t been implemented is the babies bouncing the ball back in a different direction. The author could solve this problem by making sure that when the ball is in contact with the baby that it bounces back in a different direction rather than just bouncing forward. The author could implement another method called “bounceball” which allows the ball to bounce in a different direction when one of the multi-babies touches the ball whilst the multi-babies are going up and down. For the last part of the list, it states that the author needs to “Add a scoring system for each side.”. To do this the author had to set up a “meetedge” method with a for loop of the different edges of the CBabyBallBounce application. For the right-hand side of the java application, the author had to set the grid button for the top right-hand corner which in this case is 15 and the grid button in the bottom right-hand corner which in this case is 208. The same applies for the left-hand side of the java application but with different grid buttons i.e. for the top left-hand side of the application it would be set at 0 and for the bottom left-hand side of the application, it would be set at 192. Once the right-hand side and the left-hand side have been set in a for loop the author then must specify where the ball is and set the ball icon back to the beginning. The score gets calculated as soon as the ball touches the last square on either the right-hand or left-hand side of the java application.  
There are a few limitations which both the author and the teaching assistant (Ryan) have noticed in the CBabyBallBounce application. The first limitation of the CBabyBallBounce java application is that the multi-player babies are still getting eaten up despite changing the nGame integer value and the grid button to stop the ball from moving further. The rest of the arrow keys are working without any problems and the ball isn’t eating up the babies from the up, down and right arrow button keys but are for the left arrow button keys. Another limitation within the app is that as soon as you fire up the java application it automatically pops up with an error and this is because the “run” button hasn’t yet been pressed. As soon as the timer on the run button has started after the end-user has pressed the run button then the error doesn’t pop up anymore. There wasn’t really a way of fixing this issue despite trying several ways to fix it. Another limitation in the CBabyBallBounce application is that when the run button is pressed the ball only goes to the right-hand side when you move the ball upwards away from the 2 babies. The author could have improved on this by finding a way to implement it so that the ball could also move to the left-hand side also.

The perimeter of the pitch works for both the left-hand side and the right-hand side but unfortunately the author didn’t realise that the perimeter also had to be set for the up and down. Unfortunately, due to time constraints the author was unable to implement it but thinks that this could be improved by using the same code as the left and right perimeter but replacing it with %13 rather than %16.

**How would your approach differ given the opportunity to do the assignment again?**

The next time the author is given the opportunity to do the assignment like this again the author would firstly plan what they are implementing or applying to the application first before just diving into the application and coding from scratch without a fully-fledged plan. The author could have started this off by using the designed flowcharts to plan the process as well as a checklist to see what has been implemented in the java application and what hasn’t been implemented in the java application. The author could have probably asked for more help with some of the application, especially regarding the part where the ball bounces off the walls in a random direction as the author was a little bit confused in how to approach this scenario in the given CBabyBallBounce java application.

# **References**

Eclipse Foundation (2014). Eclipse SDK 4.4.1 [online]. Available from: [http://www.eclipse.org](http://www.eclipse.org/) [Accessed 17 Jan 2017]

Assignment 2 - Problem Solving & Programming". csy1020Ass2. https://nile.northampton.ac.uk/bbcswebdav/pid-3067634-dt-content-rid-3255566\_1/courses/CSY1020-STD-1617/csy1020Ass2%281%29.pdf . Gary Hill., 2017. Web. 1 Jan.2017

Bell, D., & Parr, M. (2010). *Java for students*, 6th ed. Upper Saddle River, NJ: Prentice Hall, ISBN 027373122X.

Sun Microsystems (2014a) Lesson: Exceptions. Available from: <http://java.sun.com/docs/books/tutorial/essential/exceptions> [Accessed 20 may 2014].

Wikipedia (2014) Code Refactoring [online]. Available from: <http://en.wikipedia.org/wiki/Refactoring> [Accessed 12 May 2014].

Anon., 2017. *JLabel color change.* [Online]   
Available at: http://stackoverflow.com/questions/15882886/jlabel-color-change  
[Accessed 22 January 2017].

Anon., n.d. *How to center a window in java?.* [Online]   
Available at: http://stackoverflow.com/questions/144892/how-to-center-a-window-in-java  
[Accessed 10 February 2017].

Help, M. J., 2009. *Java GUI- add MENU BAR TUTORIAL, help swing, JMenuBar, JMenu.* [Online]   
Available at: https://www.youtube.com/watch?v=dwLkDGm5EBc  
[Accessed 22 January 2017].

MicroLearning, T., 2014. *How to use JLabel and JTextField in java?.* [Online]   
Available at: https://www.youtube.com/watch?v=\_Se\_q3L2JwQ  
[Accessed 22 January 2017].

UnitJuggler, 2008. *Convert hours to seconds - time converter.* [Online]   
Available at: https://www.unitjuggler.com/convert-time-from-h-to-s.html  
[Accessed 3 February 2017].

JSlider, How. "How To Change Default/Start Value Of Jslider". *Stackoverflow.com*. N.p., 2017. Web. 11 Apr. 2017.

What Is Abstract Window Toolkit (AWT)? - Definition From Whatis.Com". WhatIs.com. N.p., 2017. Web. 15 Apr. 2017.

Java.Awt.Event (Java Platform SE 7 )". Docs.oracle.com. N.p., 2017. Web. 15 Apr. 2017.

Javax.Swing (Java Platform SE 7 )". Docs.oracle.com. N.p., 2017. Web. 15 Apr. 2017.

Java, JFrame. "Jframe Exit On Close Java". *http://stackoverflow.com/questions/7799940/jframe-exit-on-close-java*. N.p., 2017. Web. 16 Apr. 2017.

content, Java. "Java GUI: About Getcontentpane( ) Method And Content". http://stackoverflow.com/questions/16744152/java-gui-about-getcontentpane-method-and-content. N.p., 2017. Web. 16 Apr. 2017.

"How To Write A Change Listener (The Java™ Tutorials > Creating A GUI With JFC/Swing > Writing Event Listeners)". https://docs.oracle.com/javase/tutorial/uiswing/events/changelistener.html N.p., 2017. Web. 18 Apr. 2017.

getActionCommand(), the. "The Getsource() And Getactioncommand()". http://stackoverflow.com/questions/8214958/the-getsource-and-getactioncommand. N.p., 2017. Web. 20 Apr. 2017.

Changeevent (Java Platform SE 7 )". https://docs.oracle.com/javase/7/docs/api/javax/swing/event/ChangeEvent.html. N.p., 2017. Web. 20 Apr. 2017.

# **7. Screenshots – Application Build up**

