|  |
| --- |
| CS225: Computer Science II, Section 02 |
| Final Project: Ballistic Computer |
| Francisco Javier Carrera Arias |
| **01/12/2014** |

|  |
| --- |
|  |

**Problem Overview**

For this assignment, we had to make a somewhat sophisticated software project that contained elements of almost everything we had covered in class, including GUI design, Object Oriented Design (OOP) and event handling. My specific project consists of a ballistic computer which is capable of performing calculations for 4 different types of ammunition and that is able to provide an animation of the trajectory of the projectile when given two points X and Y.

**Requirements**

The main goal of the program was to make an accurate ballistic computer that computed correctly a diverse kind of calculations that are needed in some manner to determine the trajectory of the projectile in question, while at the same time complying with the requirements of the project itself.

The requirements for the project itself were the following:

1. The program will utilize GUI for user input and to output the results
2. All user interactions must occur via GUI elements.
3. The project must make use of file I/O
4. The program must include at least one case of exception handling
5. The project must include multiple objects and at least one case of inheritance

The requirements specific to my ballistic computer were the following:

1. It must calculate parameters for different types of ammo, which must include Bullets and missiles
2. It must provide an interactive animation of the projectile trajectory, which simulates a graph.
3. The results of the calculations must be in SI units.
4. Every ammo type must have its own GUI to allow calculations of multiple types of ammo at the same time and to prevent errors.
5. The program must return errors every time the user inputs an incorrect angle or velocity

**Algorithm Discussion**

The algorithm that I used for this program can be found below:

1. The very first thing I did was design the main GUI layout. This GUI which contains the selection of the ammo types that are available, the readme and the music selection.
2. Then, I created the GUI classes for each type of ammunition and the GUI that contains the trajectory animation. The ammunition specific GUIs are in charge of taking the input from the user and to output the results of the calculations. They are also in charge of talking the input of the graph animation parameters and after that calling the graph.
3. Then, I made the classes that contained the physics equations. Each ammo type has its own physics class, since each ammo type has special characteristics unique to its type. An exception had to be made for arrows and artillery since both of them share very similar calculations.
4. After that, I embedded the physics into my ammo GUIs, (by creating objects that call the formulas in each class) allowing my program to make the input from the user and process it to make the calculations.
5. After that, I configured the graph animation to take inputs from the user, in order to be able to explore the path the hypothetical projectile will follow. To input a new path one has to reinitialize the ammunition specific GUI and input new parameters (The graph has been made independent of the calculation due to time constraints).
6. Then, I took care of the exception handling and the overall esthetics of the program. I made so by using try/catch blocks for the exception handling, and pictures, color and music for the overall esthetics.
7. And finally I took care of the file I/O by allowing the user to save the results of the calculations in text files for a future use that may require the information.

**Test Cases and Results**

I would like to start saying that this project has been probably the most complex I have done so far, even more than my breeding simulator that I did in MATLAB. My ballistic computer accomplishes all the requirements of the project both the ones that came as general directives for the project (requirements 1 to 5) and my own requirements (requirements 5-10). In the following paragraph I will provide a detailed analysis of how I achieved the requirements, what were my test cases and what are some flaws that I have observed in my program.

For the first and second requirements, which are that all user interactions including output and input must occur via GUI. I made my GUI in a way that you do not have to look elsewhere for anything (meaning with this the command line or the code itself).

In essence I made the main GUI have control over the ammunition specific GUI’s rather than making you go back or restarting the program to select different ammunition. When you click the button of any specific ammunition type it takes you to the ammunition specific stage where it performs all of the calculation relates to that type of ammunition. The user has to input the values in the text fields and then click the button “calculate” that allows my program to display the results of the calculations.

Other thing that each of my ammo specific panes has is the ballistic trajectory simulation. I have to admit that it is not as perfect as I would have initially intended but it does provide an accurate trajectory of the projectile if you input the correct data. This data refers to the initial velocity and the range of the projectile.

I did not experience much difficulty with my GUI design. I did have an inefficient algorithm at the beginning that made you input your first ammunition choice in the main GUI rather than creating its own but afterwards you had to restart the program to calculate the parameters of other types of ammo.

I have also been told through the main tests that I made to this program, which were 2 usability tests when I finished, it that I ask for data so technical that the user gets trapped if he/she is not knowledgeable about ballistics. To tackle this problem I made instructions in a bar menu in the main GUI with a big “READ ME FIRST!!! “ label, which will allow any user to use my program without needing hours of research.

Another thing that I found out through usability testing is that the users for some reason do not like my ammo specific GUI set up. I have designed this program be in a multiscreen system with at least two monitors, which if it is seen in that way makes perfect sense to have separate GUI’s for each ammo type.

The last feedback that I got from usability testing was that the screen resolution was way too big. I admit that I was biased by my own main screen which is a 32 inch smart TV and I made program as big as that in the beginning after seeing that my program was unable to fit in any normal computer I adjusted the stages to the minimum resolution that I was capable, which is 1500 x 1000 (Which is still big for many computers so I recommend running my program in a big screen).

Due to my code being too long I am not going to paste my GUI class here it will be placed in the important source code area.

For the third requirement, which is that the program must use file I/O. I recognize that I have had a lot of trouble and I might not have accomplished this requirement correctly. I have functional file reading, when I read the images and the music that I have foes esthetics. I also have file output, which saves to a text file some of the most important parameters of the projectiles. This text files are stored in the folder outside eclipse every time you click the save results button in the pane.

The code for reading some sample images and the music can be found below:

Image img = **new** Image("exacto-smart-bullet.jpg");

ImageView imgView = **new** ImageView(img);

Image img2 = **new** Image("ArtilleryTactics.jpg");

ImageView imgView2 = **new** ImageView(img2);

Media m = **new** Media(Paths.*get*("EpicMusic.mp3").toUri().toString());

MediaPlayer mediaPlayer = **new** MediaPlayer(m);

The fourth general requirement which was exception handling; is very close to my own fifth requirement so I will analyze them together. Implementing exception handling caused me absolutely no trouble. I have two ways of doing it. First I have try/catch blocks for input errors in every specific ammo class, this means that I do not allow putting strings in my text fields instead of doubles( this fulfills the fourth general requirement) and then I made my own exception class that pops up an error if you input wrong parameters for the velocity and the angle(which fulfills my fifth requirement).

The code for a sample try/catch block can be found below:

**try**{

String text = InitialVelocity.getText();

Double.*parseDouble*(text);

String text2 = WindSpeed.getText();

Double.*parseDouble*(text2);

String text3 = T.getText();

Double.*parseDouble*(text3);

String text4 = DragCoefficient.getText();

Double.*parseDouble*(text4);

String text5 = AirDensity.getText();

Double.*parseDouble*(text5);

String text6 = CrossSecArea.getText();

Double.*parseDouble*(text6);

String text7 = Mass.getText();

Double.*parseDouble*(text7);

String text8 = EmpiricalCorrection.getText();

Double.*parseDouble*(text8);

String text9 = DistancetoTarget.getText();

Double.*parseDouble*(text9);

String text10 = Range.getText();

Double.*parseDouble*(text10);

String text11 = FinalVelocity.getText();

Double.*parseDouble*(text11);

}**catch**(NumberFormatException e){

**final** JPanel panel = **new** JPanel();

JOptionPane.*showMessageDialog*(panel,"OK many things could have happened:\n"

+ "1) You are shooting a railgun\n"

+ "2) You simply did not know the input I was asking for (Check instructions for help) or\n"

+ "3) You decided that physics is going to operate with things other than numbers, PLEASE INPUT VALID DOUBLES ","Error", JOptionPane.***ERROR\_MESSAGE***);

InitialVelocity = **new** TextField();

WindSpeed = **new** TextField();

T = **new** TextField();

DragCoefficient = **new** TextField();

AirDensity = **new** TextField();

CrossSecArea = **new** TextField();

Mass = **new** TextField();

EmpiricalCorrection = **new** TextField();

DistancetoTarget = **new** TextField();

Range = **new** TextField();

FinalVelocity = **new** TextField();

}

}

The HIghVelocityandAngleException()class can be found below:

**public** **class** HighVelocityandAngleException **extends** Exception{

**public** **void** VelocityException(**double** Velocity){

**final** JPanel panel = **new** JPanel();

JOptionPane.*showMessageDialog*(panel,"OK Slow down the velocity you used is wrong for sure:\n"

+ "1) I you are using bullets with a speed over 1000 m/s you are delirious\n"

+ "2) If you are shooting arrows with a speed over 250 m/s either you are shooting the mythical Caladbolg or an alien bow\n"

+ "3) If you are shooting Artillery and your speed exceeds 2000 m/s I have to remind you that this program does not support railguns ","Error", JOptionPane.***ERROR\_MESSAGE***);

}

**public** **void** AngleException(**double** Velocity){

**final** JPanel panel = **new** JPanel();

JOptionPane.*showMessageDialog*(panel,"OK Slow down the velocity you used is wrong for sure:\n"

+ "1) If you are shooting arrows with an angle below 5 degrees......well yes its ok but why are you using my ballistic computer????\n"

+ "2} If you are shotting ICBM with an angle of 90 degrees or below 10 degrees say goodbye to your country\n"

+ "2) If you are shooting Artillery and your angle is below 5 degrees again why are you using my program","Error", JOptionPane.***ERROR\_MESSAGE***);

}

}

For the fifth general requirement which was that the program has two include multiple objects and at least one case of inheritance I made the program with this in mind. My program has indeed multiple objects that can be found across the entire assignment and two cases of inheritance. These cases of inheritance are the ammoPhysics () class, which is the parent of all the specific ammo physics classes and the GUI () class, which is the parent of all the ammo specific GUI’s. For more details on the design refer to the UML diagram.

While I was taking about the general requirements, I have covered most of my own requirements as well. However I would like to cover in detail the requirements that I have regarding SI units and the animation. My calculations are done using SI units (m/s, kg, degrees….) and they output SI units. Every test field has the SI unit required next to it to prevent confusion. However, I am aware that for example putting the cross sectional area of a bullet in meters square is quite tedious (0.00097 an example of a bullet) and I wanted to correct it so I was more user friendly but time constraints have not allowed me to do it. If you do not know what to put in the text field refer yourself to the instructions in the program I have values that will perform a correct computation.

The graphing animation is precise in terms of trajectory. However it is complex to operate. The very first value has to be the initial velocity always (in the case of the ICBM that they can reach velocities of 2000m/s and above thousands of m/s write the nearest hundred for example if you have 5000m/s type 500 in the first field). The second field is the tricky one you must write the range of your projectile in X (with the drag) again the rule of the thousands apply (if you have more than 2000 anything type the up to the hundredth unit of the number, for example if you have 32000 meters of max distance in X for artillery type 320 in the second field). I am aware of the difficulty of operating the graph animation and so I was told in my usability testing, but it was the only way of incorporating the vital visual element into my program and because of hardware restrictions I cannot make a stage that accommodates simulating a 32 km shot. I have provided good values in the program instructions to avoid confusion. However, if any problem rises and you cannot get a credible trajectory please call me I will explain in person and show the functioning personally.

Last but not least. I would like to comment about the empirical validation test of my physics calculations. I came across the idea of the validation test while I was using my program to calculate arrow trajectories in a videogame (Farcry 4). My program showed a nice capability for predicting the arrow trajectory. Of course Farcry 4 is just a videogame and I wanted to see if my calculations were really that precise so I decided to go to an archery club in Miami during Thanksgiving a conduct and empirical test.

The test results were much better than I predicted. Concretely, the time of flight of the arrow was extremely accurate. The arrow that I shot had the same specifications as the one that is typed as the suggested arrow in my program instructions (that one is guaranteed for sure to be precise).

The data that I got when I performed the calculation during the test are as it follows:

The maximum distance in X that the program gave me was 361.073 meters after providing an input of an initial velocity of 110m/s and an angle of 10 degrees. I shot with a 50 pound bow that propelled the arrow at 108.4m/s with an angle of approximately 10 degrees and the distance that I measured after the shot afterwards was 350.4 meters.

The time of flight that the program gave me with the same inputs was 3.89 seconds and the time that I measured in the range was of 3.78 seconds.

These results validate the physics of my program (at least the arrow and probably the artillery since they share a lot). I wish I could have conducted test with the other ammunition types but it was simply not possible for varied reasons (especially ICBM’s).

There are still a lot of edges that I can polish if I keep working in this program as I hope to do. I consider it as of now a very nice beta version of what a final version could look like. My only hope us that I can find the time to make a really nice program which may be even useful to other people sometime.

**Important Source Code**

In previous homework Jessica asked us to provide the source code in the word document as well (ignore this section if you did not want this I am doing out of habit since this was Jessica’s homework format). However, due to the size of my program I am going to paste only some of the most important code as an example. This includes the main GUI class a sample of a specific ammunition GUI class and a physics class:

**The main GUI class**

**public** **class** GUI **extends** Application {

// Create the panes that are going to form my GUI

BorderPane borderPane = **new** BorderPane();

GridPane valuesPane = **new** GridPane();

BorderPane vpane = **new** BorderPane();

BallisticAnimation Graph = **new** BallisticAnimation();

BorderPane firstammo = **new** BorderPane();

BorderPane secondammo = **new** BorderPane();

BorderPane thirdammo = **new** BorderPane();

BorderPane forthammo = **new** BorderPane();

Media m = **new** Media(Paths.*get*("EpicMusic.mp3").toUri().toString());

MediaPlayer mediaPlayer = **new** MediaPlayer(m);

MenuItem About = **new** MenuItem("Program Instructions");

**private** MenuBar Bar;

**private** Menu Help;

TextField Header1 = **new** TextField("What ammo type are you shooting (Bullet, Artillery, Arrow, ICBM): ");

Image img = **new** Image("exacto-smart-bullet.jpg");

ImageView imgView = **new** ImageView(img);

Image img2 = **new** Image("ArtilleryTactics.jpg");

ImageView imgView2 = **new** ImageView(img2);

// Create the Button Objects

**protected** Button

BulletBT = **new** Button("Bullets"),

ArtilleryBT = **new** Button("Artillery"),

ArrowsBT = **new** Button("Arrow"),

ICBMBT = **new** Button("ICBM"),

BallisticAnimationBT = **new** Button("Show Trajectory"),

SaveBT = **new** Button("Save Results"),

MusicBT1 = **new** Button("Music"),

CalculateBT = **new** Button("Calculate");

// Initialize the Constructor

**public** GUI(){

borderPane.setStyle("-fx-background-color: cyan;");

BulletBT.setPrefWidth(500);

ArtilleryBT.setPrefWidth(500);

ArrowsBT.setPrefWidth(500);

ICBMBT.setPrefWidth(500);

MusicBT1.setPrefWidth(500);

// Menu Creation

Help = **new** Menu("READ ME FIRST !!!!!!");

Bar = **new** MenuBar();

Help.getItems().add(About);

Bar.getMenus().addAll(Help);

// Add the text fields to the grid pane

Header1.setPrefWidth(375);

valuesPane.setVgap(55);

valuesPane.setHgap(2);

valuesPane.setStyle("-fx-background-color: cyan;");

valuesPane.add(Header1, 0, 0);

valuesPane.add(BulletBT, 0, 1);

valuesPane.add(ArtilleryBT, 0, 2);

valuesPane.add(ArrowsBT, 0, 3);

valuesPane.add(ICBMBT, 0, 4);

// vpane setup

TextField Title = **new** TextField("Music Toggle");

Title.setPrefWidth(200);

vpane.setTop(Title);

vpane.setCenter(MusicBT1);

vpane.setBottom(imgView2);

vpane.setStyle("-fx-background-color: cyan;");

// put all the shapes in the shape pane and set its color

}

@Override

**public** **void** start(Stage Stage) **throws** Exception { // Start method and stage initialization

// Event Handlers

BulletBT.setOnAction(**new** BulletChoicePane());

ArtilleryBT.setOnAction(**new** ArtilleryChoicePane());

ArrowsBT.setOnAction(**new** ArrowChoicePane());

ICBMBT.setOnAction(**new** ICBMChoicePane());

BallisticAnimationBT.setOnAction(**new** BallisticGraph());

MusicBT1.setOnAction(**new** Music());

About.setOnAction(e ->About());

// Put everything in the scene

Scene scene1 = **new** Scene(borderPane, 1500, 1000);

borderPane.setCenter(valuesPane);

borderPane.setBottom(imgView);

borderPane.setRight(vpane);

borderPane.setTop(Bar);

// Configure the stage

Stage.setScene(scene1);

Stage.setTitle("Ballistic Computer Sovereign Cerebrum");

Stage.show();

}

**public** **static** **void** main(String[] args) { // main method

GUI.*launch*(args); // launch the GUI

}

**public** **class** BallisticGraph **implements** EventHandler<ActionEvent>{

@Override

**public** **void** handle(ActionEvent e){

**double** ValueX = Double.*parseDouble*(JOptionPane.*showInputDialog*("Please provide a value X to graph (This can be the max distance in X: "));

**double** ValueY = Double.*parseDouble*(JOptionPane.*showInputDialog*("Please provide a value Y to graph (This can be the max distance in Y: "));

Group rootGroup = **new** Group();

Stage Stage2 = **new** Stage();

Scene scene2 = **new** Scene(rootGroup,700,600,Color.***GHOSTWHITE***);

Graph.applyAnimation(rootGroup,ValueX,ValueY);

Stage2.setScene(scene2);

Stage2.show();

}

}

**public** **class** Music **implements** EventHandler<ActionEvent>{

@Override

**public** **void** handle(ActionEvent e){

mediaPlayer.play();

}

}

**public** **class** ArtilleryChoicePane **implements** EventHandler<ActionEvent>{ // inner class that calculates the area of the triangle

@Override

**public** **void** handle(ActionEvent e){

ArtilleryChoice Shell1 = **new** ArtilleryChoice();

Shell1.AmmoPane();

}

}

**public** **class** ArrowChoicePane **implements** EventHandler<ActionEvent>{

@Override

**public** **void** handle(ActionEvent e){

ArrowChoice arrow = **new** ArrowChoice();

arrow.AmmoPane();

}

}

**public** **class** ICBMChoicePane **implements** EventHandler<ActionEvent>{

@Override

**public** **void** handle(ActionEvent e){

ICBMChoice missile = **new** ICBMChoice();

missile.AmmoPane();

}

}

**public** **class** BulletChoicePane **implements** EventHandler<ActionEvent>{

@Override

**public** **void** handle(ActionEvent e){

BulletChoice bullet = **new** BulletChoice();

bullet.AmmoPane();

}

}

**public** **void** About(){

**final** String aboutText = "This program was written entirely by Francisco Javier Carrera Arias with very appreciated contributions from Sean Holden and Dustin Marx."

+ " The program as it is right now is completely functional and provides accurate physics calculations for all the ammo options.\n"

+ "However, I consider this program a beta of what could be a much larger program with more detailed physics and a better graph. This does not mean that the program as it is right now does not give accurate calculations,"

+ "which it does it means that with more work this program could be very useful in real situations outside the classroom.\n"

+ "One flaw that I recognize is that many of my calculations ask for technical data that is not precisely easy to obtain, lucky for you user, I included sample runs just below:\n\n"

+ "For a nice result with a bullet type the following:\n"

+ "Speed: 400 m/s\n"

+ "Windspeed: 15 mph\n"

+ "Temperature: 70 F\n"

+ "Drag Coefficient: 0.295\n"

+ "Cross Sectional Area: 0.00013\n"

+ "Air Density: 1.225 kg/m^3\n"

+ "Distance of test shot: 15\n"

+ "Distance to target in hundreds of yards: 5\n"

+ "Range of rifle in hundreds of yards: 10\n"

+ "Bullet Mass: 0.0097 kg\n"

+ "Downrange Velocity: 295 m/s\n\n"

+ "For a nice result with artillery type the following:\n"

+ "Speed: 700 m/s\n"

+ "Angle: 25 degrees\n"

+ "Drag Coefficient: 0.41\n"

+ "Cross Sectional Area: 0.00045\n"

+ "Air Density: 1.225 kg/m^3\n"

+ "Bullet Mass: 5 kg\n\n"

+ "For a nice result with an arrow type the following:\n"

+ "Speed: 110 m/s\n"

+ "Angle: 25 degrees\n"

+ "Drag Coefficient: 0.35\n"

+ "Cross Sectional Area: 0.00035\n"

+ "Air Density: 1.225 kg/m^3\n"

+ "Bullet Mass: 0.08 kg\n\n"

+ "For a nice result with an ICBM type the following:\n"

+ "Speed: 5500 m/s\n"

+ "Angle: 25 degrees\n"

+ "Horizontal distance to target: 4800000 m\n"

+ "Name: The one you like";

// Create the text label

Label HelpLabel = **new** Label();

HelpLabel.setWrapText(**true**);

HelpLabel.setTextAlignment(TextAlignment.***LEFT***);

HelpLabel.setFont(Font.*font*("Arial", 12));

HelpLabel.setText(aboutText);

// Add the label to a StackPane

StackPane pane = **new** StackPane();

pane.getChildren().add(HelpLabel);

// Create and display said the pane in a new stage

Scene scene = **new** Scene(pane, 700, 700);

Stage stage = **new** Stage();

stage.setScene(scene);

stage.setTitle("Basic Intructions for operating the program");

stage.setResizable(**false**);

stage.show();

}

} // end of class mai

**The ammunition specific GUI**

**public** **class** BulletChoice **extends** GUI{

GridPane valuesPane = **new** GridPane();

GridPane Calculations = **new** GridPane();

BorderPane Estetics = **new** BorderPane();

Image img = **new** Image("speeding\_bullet.jpg");

ImageView imgView = **new** ImageView(img);

Image img2 = **new** Image("BULLET.jpg");

ImageView imgView2 = **new** ImageView(img2);

TextField Header2 = **new** TextField("Please provide the initial velocity: ");

TextField InitialVelocity = **new** TextField();

TextField Header3 = **new** TextField("Please provide the wind speed (in mph): ");

TextField WindSpeed = **new** TextField();

TextField Header4 = **new** TextField("Please provide the Temperature: ");

TextField T = **new** TextField();

TextField Header5 = **new** TextField("Please provide the drag coefficient: ");

TextField DragCoefficient = **new** TextField();

TextField Header6 = **new** TextField("Please provide the air density: ");

TextField AirDensity = **new** TextField();

TextField Header7 = **new** TextField("Please provide the cross sectional area of the projectile: ");

TextField CrossSecArea = **new** TextField();

TextField Header8 = **new** TextField("Please provide the projectile mass: ");

TextField Mass = **new** TextField();

TextField Header9 = **new** TextField("Please provide the observed distance of test shot (in inches): ");

TextField EmpiricalCorrection = **new** TextField();

TextField Header10 = **new** TextField("Please provide the distance to your target (in hundreds of yards): ");

TextField DistancetoTarget = **new** TextField();

TextField Header11 = **new** TextField("Please provide weapon range (in hundreds of yards): ");

TextField Range = **new** TextField();

TextField Header12 = **new** TextField("Please provide the downrange velocity: ");

TextField FinalVelocity = **new** TextField();

TextField Table1 = **new** TextField();

TextField Table2 = **new** TextField();

TextField Table3 = **new** TextField();

TextField Table4 = **new** TextField();

TextField Table5 = **new** TextField();

TextField Table6 = **new** TextField();

TextField Table7 = **new** TextField();

TextField Table8 = **new** TextField();

TextField Table9 = **new** TextField();

**public** **void** AmmoPane(){

Header2.setPrefWidth(425);

valuesPane.setVgap(10);

valuesPane.setHgap(2);

valuesPane.add(Header2, 0, 1);

valuesPane.add(InitialVelocity, 1, 1);

valuesPane.add(Header3, 0, 2);

valuesPane.add(WindSpeed, 1, 2);

valuesPane.add(Header4, 0, 3);

valuesPane.add(T, 1, 3);

valuesPane.add(Header5, 0, 4);

valuesPane.add(DragCoefficient, 1, 4);

valuesPane.add(Header6, 0, 5);

valuesPane.add(AirDensity, 1, 5);

valuesPane.add(Header7, 0, 6);

valuesPane.add(CrossSecArea, 1, 6);

valuesPane.add(Header8,0,7);

valuesPane.add(Mass,1,7);

valuesPane.add(Header9,0,8);

valuesPane.add(EmpiricalCorrection,1,8);

valuesPane.add(Header10,0,9);

valuesPane.add(DistancetoTarget,1,9);

valuesPane.add(Header11,0,10);

valuesPane.add(Range,1,10);

valuesPane.add(Header12,0,11);

valuesPane.add(FinalVelocity,1,11);

valuesPane.add(BallisticAnimationBT,2,12);

valuesPane.add(CalculateBT,0,12);

BallisticAnimationBT.setOnAction(**new** BallisticGraph());

CalculateBT.setOnAction(**new** BulletCalc());

forthammo.setStyle("-fx-background-color: cyan;");

forthammo.setCenter(valuesPane);

forthammo.setRight(imgView);

forthammo.setBottom(Estetics);

Estetics.setLeft(Calculations);

Estetics.setRight(imgView2);

Stage Stage5 = **new** Stage();

Scene scene5 = **new** Scene(forthammo,1500,1000,Color.***GHOSTWHITE***);

Stage5.setScene(scene5);

Stage5.setTitle("Bullet");

Stage5.show();

}

**public** **void** InputCheck() **throws** NumberFormatException{

**try**{

String text = InitialVelocity.getText();

Double.*parseDouble*(text);

String text2 = WindSpeed.getText();

Double.*parseDouble*(text2);

String text3 = T.getText();

Double.*parseDouble*(text3);

String text4 = DragCoefficient.getText();

Double.*parseDouble*(text4);

String text5 = AirDensity.getText();

Double.*parseDouble*(text5);

String text6 = CrossSecArea.getText();

Double.*parseDouble*(text6);

String text7 = Mass.getText();

Double.*parseDouble*(text7);

String text8 = EmpiricalCorrection.getText();

Double.*parseDouble*(text8);

String text9 = DistancetoTarget.getText();

Double.*parseDouble*(text9);

String text10 = Range.getText();

Double.*parseDouble*(text10);

String text11 = FinalVelocity.getText();

Double.*parseDouble*(text11);

}**catch**(NumberFormatException e){

**final** JPanel panel = **new** JPanel();

JOptionPane.*showMessageDialog*(panel,"OK many things could have happened:\n"

+ "1) You are shooting a railgun\n"

+ "2) You simply did not know the input I was asking for (Check instructions for help) or\n"

+ "3) You decided that physics is going to operate with things other than numbers, PLEASE INPUT VALID DOUBLES ","Error", JOptionPane.***ERROR\_MESSAGE***);

InitialVelocity = **new** TextField();

WindSpeed = **new** TextField();

T = **new** TextField();

DragCoefficient = **new** TextField();

AirDensity = **new** TextField();

CrossSecArea = **new** TextField();

Mass = **new** TextField();

EmpiricalCorrection = **new** TextField();

DistancetoTarget = **new** TextField();

Range = **new** TextField();

FinalVelocity = **new** TextField();

}

}

**public** **class** BulletCalc **implements** EventHandler<ActionEvent>{ // inner class that calculates the area of the triangle

@Override

**public** **void** handle(ActionEvent e){

Bullets Formulas = **new** Bullets();

HighVelocityandAngleException except = **new** HighVelocityandAngleException();

InputCheck();

String text = InitialVelocity.getText();

**double** Velocity = Double.*parseDouble*(text);

**if**(Velocity > 2000){

except.VelocityException(Velocity);

}

String text2 = WindSpeed.getText();

**double** Wind = Double.*parseDouble*(text2);

String text3 = T.getText();

**double** T = Double.*parseDouble*(text3);

String text4 = DragCoefficient.getText();

**double** Cd = Double.*parseDouble*(text4);

String text5 = AirDensity.getText();

**double** Density = Double.*parseDouble*(text5);

String text6 = CrossSecArea.getText();

**double** Area = Double.*parseDouble*(text6);

String text7 = Mass.getText();

**double** M = Double.*parseDouble*(text7);

String text8 = EmpiricalCorrection.getText();

**double** Correct = Double.*parseDouble*(text8);

String text9 = DistancetoTarget.getText();

**double** Distance = Double.*parseDouble*(text9);

String text10 = Range.getText();

**double** Range = Double.*parseDouble*(text10);

String text11 = FinalVelocity.getText();

**double** Vf = Double.*parseDouble*(text11);

**double** time = Formulas.FlightTime(Velocity,Distance,Vf);

**double** MOAChange = Formulas.MOAChange(Correct, Distance);

**double** MOAWind = Formulas.MOAWind(Velocity,Range,Wind);

**double** TailWind = Formulas.TailWindComp(Distance, Wind);

**double** TComp = Formulas.TComp(T, Distance);

**double** Drop = Formulas.BulletDrop(Velocity, Distance);

**double** Vt = Formulas.TerminalVelocity(M, Density, Area, Cd);

**double** BC = Formulas.BallisticCoeficient(Cd, M, Area);

**double** Energy = Formulas.ImpactEnergy(M, Velocity);

TextField Table1 = **new** TextField("The flightime to target is: " + time + " seconds");

TextField Table2 = **new** TextField("The minute of angle change to that target is: " + MOAChange + " MOA's");

TextField Table3 = **new** TextField ("The minute of angle change due to wind is: " + MOAWind + " MOA's");

TextField Table4 = **new** TextField("The tail and head wind compensation is: " + TailWind + " Yards in range");

TextField Table5 = **new** TextField("The temperature compensation is: " + TComp + " yards");

TextField Table6 = **new** TextField("The bullet drop is: " + Drop + " meters");

TextField Table7 = **new** TextField("The terminal velocity of this projectile is: "+ Vt + "m/s");

TextField Table8 = **new** TextField("The Ballistic Coefficient is: "+ BC);

TextField Table9 = **new** TextField("The Projectile Energy at impact is: "+ Energy + " J");

Table1.setPrefWidth(500);

Calculations.setHgap(2);

Calculations.setVgap(20);

Calculations.add(Table1,0,0);

Calculations.add(Table2,0,1);

Calculations.add(Table3,0,2);

Calculations.add(Table4,0,3);

Calculations.add(Table5,0,4);

Calculations.add(Table6,0,5);

Calculations.add(Table7,0,6);

Calculations.add(Table8,0,7);

Calculations.add(Table9,0,8);

}

}

}

**The Physics Class**

**public** **class** ICBM **extends** AmmoPhysics{

**double** TimeforImpact;

**final** **double** EarthRadius = 6371000;

**public** **double** TimeImpact(**double** InitVelocity, **double** angle, **double** DistanceToTarget){

angle = Math.*toRadians*(angle);

TimeforImpact = (DistanceToTarget)/(InitVelocity \* Math.*cos*(angle));

**return** TimeforImpact;

}

}