

Project 1

Compute the Projectile Motion

Distribute on September 2, 2025

Due before September 28, 2025 (Sunday) at 11:59 PM before 12:00 midnight

Learning Outcomes (CLO vs SO Mapping)

- Recognize the software and hardware components of a computer system (1)vs(6)
- Recognize and apply the software development phases (2)vs(6)
- Utilize Java syntax in fundamental programming algorithms (3)vs(1)
- Recognize and apply the various input and output devices in programming (4)vs(2)

Objectives

In completing this project, you will gain experience with the following Java features:

- definition of a Java class
- variable declaration and assignment
- JOptionPane dialog boxes for input and output
- the Math class
- the String class
- the String.format() method and the printf() method
- selection logic
- arithmetic expressions and data manipulations

Problem Statement

We encounter projectile motion whenever we attempt to catch a fly ball, hit a long drive, or shoot a free throw. Projectile motion is the motion of an object thrown or projected into the air, subject only to the acceleration of gravity. The object is called a projectile, and its path is called its trajectory, which is of the form $y = ax + bx^2$, in which a and b are constants. The laws of physics, which describe projectile motion, are well known. Napoleon's generals first studied them.

To simplify the problem, we will assume that we can

- consider the surface of the Earth to be a plane
- ignore the effect of the drag from air friction

Under the simplifying conditions above, the following formulas hold for a projectile with an initial velocity of v feet/sec and a launch angle of r . (Note that these expressions are not written according to the syntax of the Java language.)

1. Time span of the flight: $T = 2 v \sin r / g$
2. Maximum height: $y_{\max} = (v \sin r) t - \frac{1}{2} g t^2$, where $t = T/2$
3. Range (distance traveled by the projectile): $x_{\max} = (v \cos r) T$

Here, g is the gravitational acceleration $g = 32 \text{ ft/sec}^2$, and r is measured in radians.

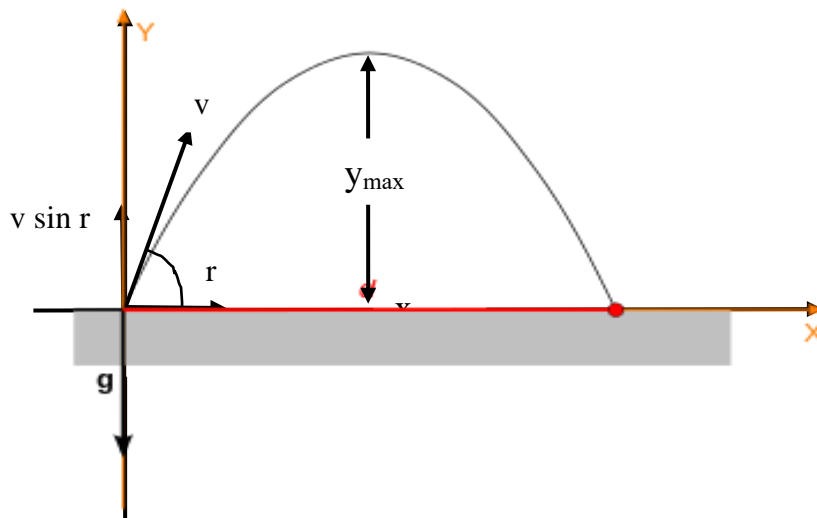


Figure: Projectile motion – a trajectory and distance of a projectile.

This project simulates the process of repeated attempts to hit a target with a projectile. The goal is to shoot the projectile within a 1-foot distance from the target, as such a short miss is considered a hit. You will construct a Java program that:

- can determine the trajectory data of a projectile for a given initial velocity and a launch angle, in particular for an initial choice of a 45-degree angle;
- can check whether the initial attempt has a launch angle of 45 degrees. If the first attempt is not at the launch angle of 45 degrees, the window displays the information, as shown in Figure 5a, and then the program exits. Then, it restarts with a launch angle of 45;
- can check if the shot of the initial attempt falls short of the target with less than -1 foot (such as target missed by -1.1, -25.0, -154.55, etc.); if so, the process is terminated and then restarted with an increased initial velocity. Note that for any initial velocity, the longest range is attained if the launch angle is 45 degrees;
- can determine the error of a shot (error = projectile range – distance to target);
- can check if the error is less than 1 foot in absolute value; if so, the user is notified about the result, and the process is terminated. That is, the error is between (-1, 1);
- can offer the user four (4) chances to modify the launch angle and try to hit the target; and, finally,
- can keep track of the smallest error produced by the subsequent attempts; the best result is reported to the user.

Analysis and requirements

The analysis describes the logical structure of the problem in a way that helps us to plan (design) a solution.

Input

Initial input values are

- initial velocity** (feet/sec),
- distance** to the desired target (feet), and

- (iii) **launch angle** (degree), and
- (iv) the gravitational acceleration (a defined constant).

If applicable, additional input values are the launch angles for the repeated attempts. The angle must always be in the range of 0.0 – 45.0 degrees.

Distance, velocity, and launch angle are solicited from the user on JOptionPane input windows. See Figures 1, 2, and 3.

To use the dialog boxes of the JOptionPane class, you need to have the following statement

```
import javax.swing.JOptionPane;
```

before the class ProjectileMotion and the main() method, and include a statement

```
System.exit(0);
```

before the end of the body of the main() method, shown as follows:

```
import javax.swing.JOptionPane; //Need for the JOptionPane class.
```

```
public class ProjectileMotion {  
  
    public static void main(String[] args) {  
        ...  
        System.exit(0);  
    } //end of main  
} //end of class ProjectileMotion
```

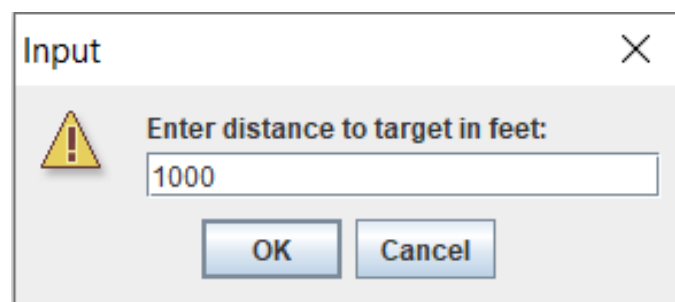
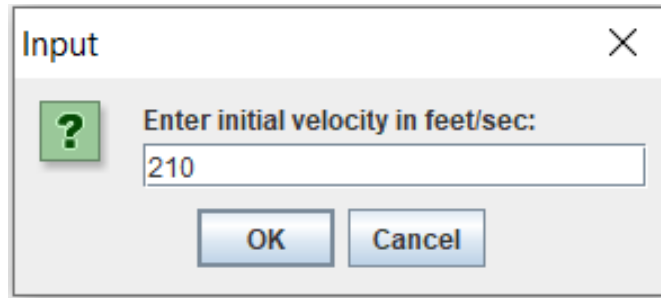


Figure 1



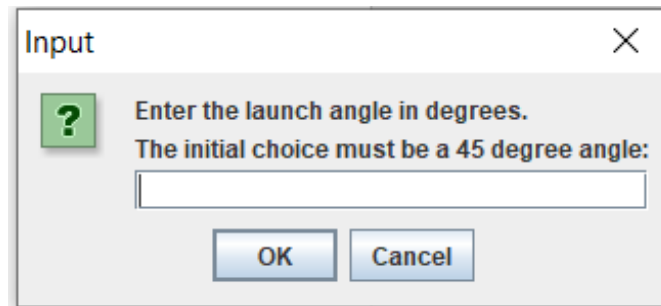
Input

Enter initial velocity in feet/sec:

210

OK Cancel

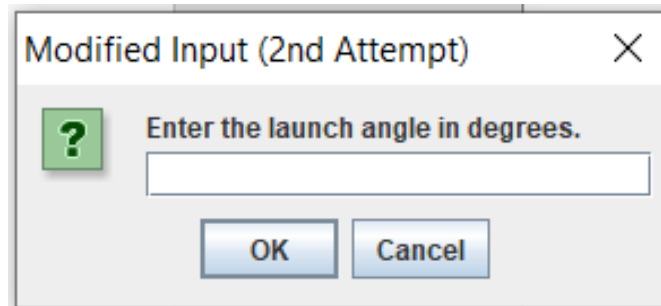
Figure 2



Input

Enter the launch angle in degrees.
The initial choice must be a 45 degree angle:

OK Cancel

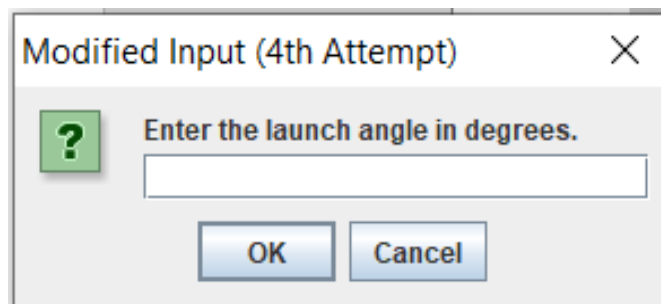


Modified Input (2nd Attempt)

Enter the launch angle in degrees.

OK Cancel

...



Modified Input (4th Attempt)

Enter the launch angle in degrees.

OK Cancel

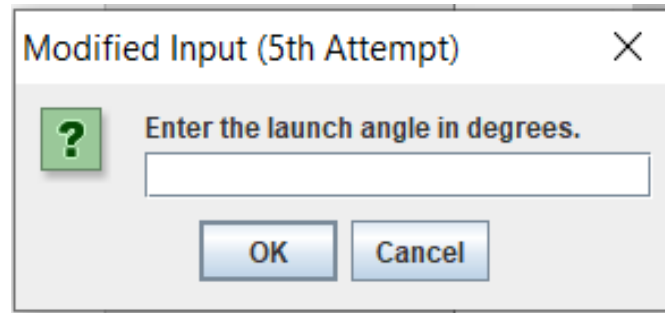


Figure 3

Output

Output messages are displayed both on JOptionPane windows and on the console. Every time the program executes a launch, a report describing the trajectory details must be displayed, as shown in Figure 4.

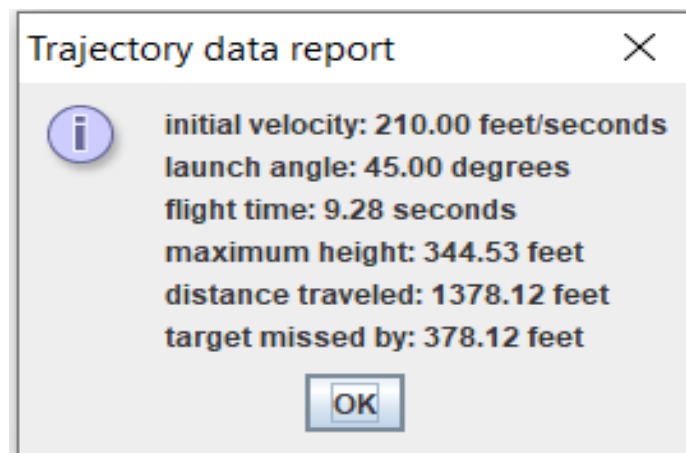


Figure 4

Each report must contain

- (1) the initial velocity in feet/sec,
- (2) the current launch angle in degrees,
- (3) the flight time in seconds,
- (4) the maximum height attained in feet,
- (5) the distance traveled by the projectile (range) in feet and
- (6) the error with which the projectile missed the target in feet.

Note that the error is a positive value when the projectile overshoots and a negative value for short shots. All real numbers displayed must be rounded to the nearest hundredth. Hints: an exemplary code is as follows:

```
String range = String.format("launch angle: %.2f degrees.\n" +  
                             "distance traveled: %.2f feet.\n", launchAngle, distanceTraveled);  
String title = "Trajectory data report";
```

```
JOptionPane.showMessageDialog(null, range, title,  
JOptionPane.INFORMATION_MESSAGE);
```

The first trajectory data report is based upon a launch angle of 45 degrees (which provides the longest possible range) to see if the target is within reach for the given velocity. The program enforces the initial (first) attempt with a launch angle of 45 degrees. If the initial (first) attempt *falls short* of the target, another window, as shown in Figure 5, displays the information, and then the program exits. Suppose the initial (first) attempt is not at the launch angle of 45 degrees after displaying the trajectory data report. In that case, the window displays the information, as shown in Figure 5a, and then the program exits.

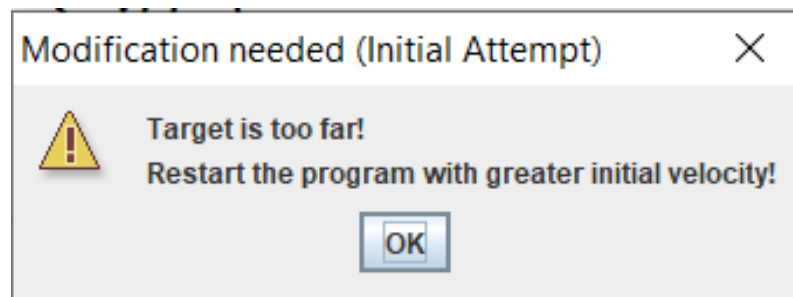


Figure 5

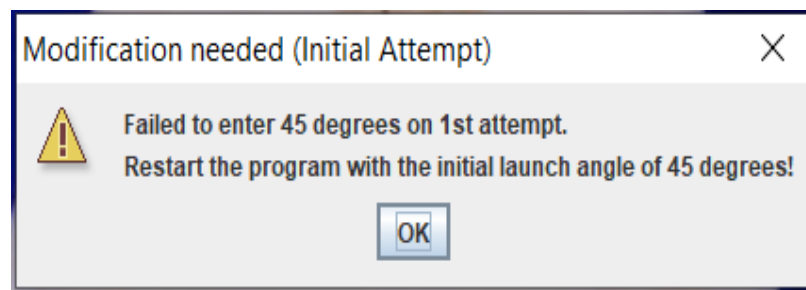


Figure 5a

At the launch angle of 45 degrees in the initial (first) attempt, if the first shot is long enough (i.e., an overshoot occurs), the window of Figure 3 shall be used to input all subsequent launch angle modifications as chosen by the user. The title of the window also displays the number of attempts, such as "Modified Input (2nd Attempt)". The corresponding reports in Figure 4 show the recalculated trajectories. Naturally, the user will try to adjust the angle to make the projectile land closer and closer to the target. After each unsuccessful attempt, a warning is printed on the console; see a sample in Figure 6.

Shot went beyond the target. Decrease the launch angle from 45.0!

Figure 6

Figure 7 shows a sample output on the console after all five angle modifications failed to hit the target. One of the following statements is displayed on the console.

Second Attempt: Shot went beyond the target. Decrease the launch angle from 40.0!
Second Attempt: Shot fell short of the target. Increase the launch angle from 20.0!
Third Attempt: Shot fell short of the target. Increase the launch angle from 10.0!
Third Attempt: Shot went beyond the target. Decrease the launch angle from 30.0!
Fourth Attempt: Shot went beyond the target. Decrease the launch angle from 24.0!
Fourth Attempt: Shot fell short of the target. Increase the launch angle from 23.0!
Fifth Attempt: Shot went beyond the target. Decrease the launch angle from 23.5!
Fifth Attempt: Shot fell short of the target. Increase the launch angle from 23.26!
Your best shot missed the target by 7.90 feet.
Your best shot missed the target by 0.01 feet.

Figure 7

It is necessary to keep track of the least absolute error in the series of attempts, as it must be reported as one of the last two lines in Figure 7.

Figure 8 shows a successful launch. Such a report is followed by the message in Figure 9.

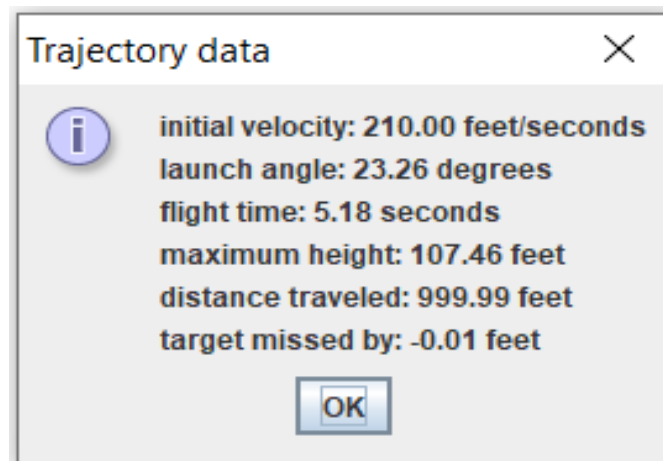
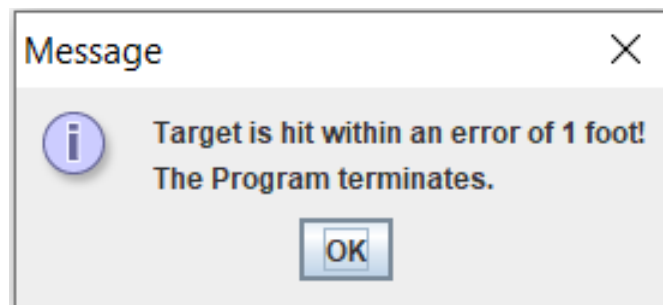


Figure 8



Your best shot missed the target by 0.01 feet.

or
 Your best shot missed the target by -0.91 feet.
 or
 Your best shot missed the target by 0.00 feet.

Figure 9

If the fifth attempt fails to hit the target, after displaying one of the messages in Figure 7 on the console, the program also displays the following message to the console.

End of the fifth attempt. Restart the program and try again.

Figure 10

Relevant Formulas

The basic formulas 1, 2, and 3 in the **Problem Statement** will be used for all trajectory computations. The input value for a launch angle shall be solicited and given in degrees. The previous formulas shall be implemented using the trigonometry methods `Math.sin(angle)` and `Math.cos(angle)` from the `Math` class. These methods require the parameter angle in radians. Therefore, an angle given in degrees must be converted to radians. The conversion formula runs as follows:

$$(\text{angle in radians}) = (\text{angle in degrees}) \pi / 180$$

Note that it can be used in either direction, from degrees to radians or vice versa.

The π (PI) value is available as a named constant **`Math.PI`** from the `Math` class of the Java library.

The Java API (Application Programmer Interface) provides a class named `Math`, which contains numerous methods, such as `Math.sqrt()`, `Math.cbrt()`, `Math.PI()`, `Math.sin()`, `Math.cos()`, `Math.abs()`, and `Math.pow()` methods.

Design

For this project, you will design a class that contains all the necessary data and operations. A suggested class name is **`Projectile`**. You must decide upon the required import(s).

The **`Projectile`** class contains the `main()` method, which has all the variable declarations.

The `main()` method should carry out the following tasks in the order given:

- declares and assigns a named constant for the gravitational acceleration; the value is 32.
- declares the following local variables (mentioned in the following table) in the `main()` method.
- solicits and stores the input values as explained in the Analysis and Requirements section; to accomplish this task, create a call statement and define a called method that uses the `JOptionPane` dialog box to input the value of a launch angle in degrees and returns the value of the launch angle in degrees to the call statement. An example is as follows:

```
public class Projectile {
    public static void main(String[] args) {
        ...
        //Write a call statement
        //      reenterLaunchAngle(nosOfAsttempts)
        //to call a method, reenterLaunchAngle(int nosOfAttempts){...}, and
```



```
//passes a nosOfAttempts value of integer data type to the called method.
//This call method accepts a returned value of the double type from the called
//method, and assigns the returned value to the variable launchAngle.
```

```
        launchAngle = reenterLaunchAngle(nosOfAttemptd);//a call statement
    ...
} //end of main()

//A called statement which prompts users to enter a value and returns an input double-typed value
public static double reenterLaunchAngle(int nosOfAttempts) {
    ...

    return (Double.parseDouble(JOptionPane.showInputDialog(null,
        task, title, JOptionPane.QUESTION_MESSAGE)));
} // end of reenterLaunchAngle(...)
...
} //end of class Projectile
```

- computes all the trajectory data and saves them in variables;
- builds up and stores the output message in a single string variable (the content of Figures 4 and 8), including the following two bullet tasks: create a call statement, and use a called method() to accomplish these tasks. An example is as follows:

```
//in the main() part, create a call statement
```

```
trajectoryDataReport(initialVelocity, launchAngle, flightTime,
    highestPoint, distanceTraveled, error);
```

```
//The called trajectoryDataReport(...) method is as follows:
```

```
public static void trajectoryDataReport(double initialVelocity,
    double launchAngle, double flightTime,
    double highestPoint, double distanceTraveled, double error) {

    String title;
    ...

    JOptionPane.showMessageDialog(null, trajectory, title,
        JOptionPane.INFORMATION_MESSAGE);

} // end of trajectoryDataReport
```

- displays the output windows;
- numbers in the output are formatted to the nearest hundredth; for this purpose, the String.format() and printf() methods are used;
- uses if and if-else logic to decide if additional launches are necessary and repeats the operations at most four times if needed; and
- terminates the program when it is due.

The above two tasks are related to Figures 6, 7, 9, and 10. This can be done by creating a call statement

```
errorAnalysis(attempt, error, minError, launchAngle);
```

which calls the following method:

```
public static void errorAnalysis(String attempt, double error,
                                double minError, double launchAngle) {

    String shotBeyond = "Shot went beyond the target. " +
                        "Decrease the launch angle from ";
    String fellShort = "Shot fell short of the target. " +
                        "Increase the launch angle from ";

    ...

}
```

See also the detailed class description below. You may declare any additional local variables in the main method.

Projectile	
METHOD	
main	The method opens input windows (see Figures 1, 2, and 3) to obtain the initial data. The input values are assigned to the corresponding data fields. The method uses a while loop to ensure there are five attempts to hit the target. The method calls a module for each attempt to compute the flight time, maximum height, and range. The method calls another module to display results in a message window (see Figure 4). Display Figures 5 – 9 as needed. The method also calls a module to compute the error of each hit of the target. After the last attempt, the program exits.
Local variables in main()	
GRAVITATION	Named constant with final , initialized with a value of 32; this value cannot change during the program's execution.
distanceToTarget	double; to store the distance in feet to the desired target (input)
initialVelocity	double; to store the projectile's initial velocity in feet/second (input)
launchAngle	double; to store the current input for launch angle in degrees (input)
radian	double; to store the value of the angle in radians (calculated)
flightTime	double; to store the flight time of the projectile as defined by formula 1 in the description (calculated)
highestPoint	double; to store the maximum height of the projectile as defined by formula 2 in the description (calculated)
distanceTraveled	double; to store the distance traveled by the projectile as defined by formula 3 in the description (calculated)

error	double; to store the difference between the distance traveled and the distance to the target (calculated)
minError	double; to store the least absolute error; update every time a new error value is generated (calculated)
trajectory	String; to store the current output message built upon the trajectory data
nosOfAttempts	int; count the number of attempts, including the initial attempt as the first attempt
MAX_NOS_OF_ATTEMPTS	Create a name constant with the final to specify five (5) as the value of the maximum number of attempts.
attempt	String; to record the number of attempts of input for launch angle; such as, for the initial attempt, assign attempt = "initial attempt"; for the second attempt, assign attempt = "2nd Attempt"; for the third attempt, assign attempt = "3rd Attempt"; for the fourth attempt, assign attempt = "4th Attempt"; for the fifth attempt, assign attempt = "5th Attempt";.

Testing

Test your program with input values shown in the description. See if your results match the output data.

Requirements and Hints

1. Having your program tested, run the program with a new set of input data of your own choice. **The distance to the target must be at least 800 feet.** Copy your trajectory report as a comment block after the class code.
2. You must have a comment block preceding the header of each of your classes that has the following content:

```

/*
 * <your name>
 * CS 16000-01 - 02/03, Fall Semester 2025
 * (specify your lab section only)
 * Project 1: Compute the Projectile Motion
 *
 * Description. <Summarize the purpose of the class here.>
 *
 */

```

3. You can use iteration structures (loops) on the attempt **or** without using iteration structures to code five (5) repeated attempts. For the latter approach, the relevant part of the code that calculates and displays the trajectory results must be copied four times into the program. However, for the iterative approach, you only need to write the code once it calculates and displays the trajectory results. Here is the iterative pseudo-code version of the code in your program:

- In the declaration block, define local variables.

- Use the JOptionPane input dialog box to enter the distance to the target and initial velocity.
- Use a while-loop to run its body of statements five (5) times,

```
while (nosOfAttempts <= MAX_NOS_OF_ATTEMPTS) {
```

```
//The body of statements for the while loop includes:
```

- Enter a launch Angle using a call statement, *reenterLaunchAngle*(nosOfAttempts).
- Calculate the launch angle in radians.
- Re-calculate all trajectory data.
- Re-calculate the error and update the minimum error.
- Write a call statement: *trajectoryDataReport*(initialVelocity, launchAngle, flightTime,
 - highestPoint, distanceTraveled, error);
- which displays a report of the trajectory data.
- Enforce the initial attempt with a launch angle of 45 degrees. See Figure 5a.
- When the initial attempt is with the launch angle of 45 degrees, if the first attempt falls short, display a warning message to the user, as in Figure 5, and end the program.
- Write a call statement: *errorAnalysis*(attempt, error, minError, launchAngle);
If the absolute error is less than 1, display a hit on the window, and terminate the program
- Update the number of attempts.

```
}//end of while-loop
```

```
// End of the program with System.exit(0), required statement for JOptionPane
```

```
System.exit(0);
```

```
}//end of main()
```

4. Use the *Math.abs()* method to calculate the absolute value of the error when you decide the shot's accuracy.
5. Set up a variable *minError* as suggested in the class description. Every time the program generates a new error, update the *minError* to store the least absolute error that occurred thus far. Use the *Math.abs()* and *Math.min()* methods.
6. Use the *Math.sin()* and *Math.cos()* methods for the sine and cosine functions.
7. The first input for the launch angle must be 45 degrees; you choose the four (4) additional modifications
8. Do not round the numeric values while needed in calculations. Apply formatting only when the output message is composed

Note that in 3, the relevant part of the code that calculates and displays the trajectory and error analysis results must be repeated four times in the program. A straightforward approach to writing code involves calling a method four times, such as `errorAnalysis(...)`, `trajectoryDataReport(...)`, and `errorAnalysis(...)`, for this purpose, rather than copying a repetitive code segment. (Hint!). This modularization approach can also be applied to the iterative version.

Evaluation

Documentation and style: 10 points.

Your program must conform to the Computer Science Department's Java Documentation and Style Requirements. Emphasis will be placed on having the required banner and internal comments, as well as indentation and a stylish overall appearance. Comments must be written with correct grammar and spelling. Do not neglect the documentation and style.

Correctness: 90 points.

These points will be allocated as follows:

1. Error-free compilation of the Java source file..... 10 points
2. Correct declaration of the variables.....20 points
3. Correct implementation of the projectile formulas.....15 points
4. Correct use of selection logic.....15 points
5. Correct application of dialog windows.....10 points
6. Correct display of correctly formatted output messages20 points

Total: 100 points