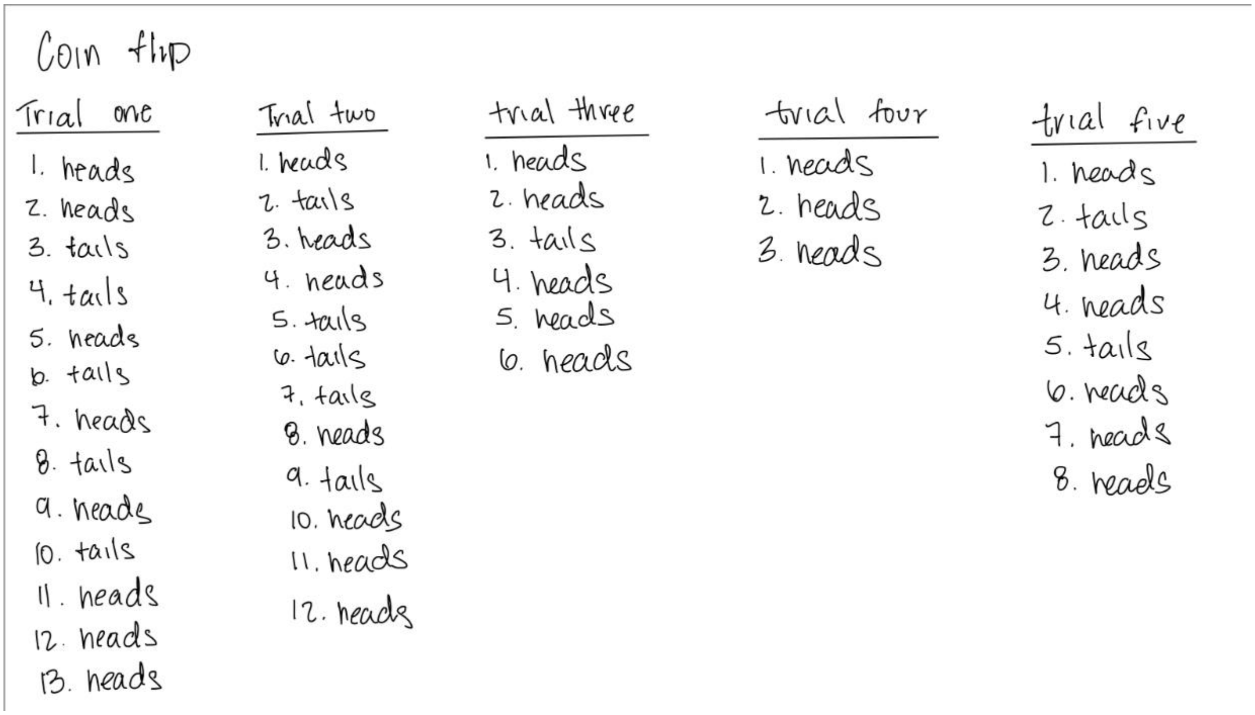
TEAM MEMBERS: MUHAMMAD AHMAD HASAN, ABDUL MUEED MOHAMMED, MARYAM BEILANI, DING QUANG.

THOUGHT PROCESS

Flip a coin until you get 3 heads in a row. How many flips did it take? 13

Repeat the above trial 4 or 5 times and record the following
The most flips it took to get 3 heads in a row: 13
The fewest flips it took to get 3 heads in a row: 3



Coin flip

Trial one
1. heads
2. heads
3. tails
4. tails
5. heads
6. tails
7. heads
8. tails
9. heads
10. tails
11. heads
12. heads
13. heads

Trial two
1. heads
2. tails
3. heads
4. heads
5. tails
6. tails
7. tails
8. heads
9. tails
10. heads
11. heads
12. heads

trial three
1. heads
2. heads
3. tails
4. heads
5. heads
6. heads

trial four
1. heads
2. heads
3. heads

trial five
1. heads
2. tails
3. heads
4. heads
5. tails
6. heads
7. heads
8. heads

The average number of flips we'd expect to need in order to get 12 heads in a row is 4096 flips. Our reasoning is that an unbiased coin would have a 50% chance of flipping heads and a 50% chance of flipping tails. The probability of flipping heads 12 times in a row is (0.5)^12. Using the expected value formula gives us (1/probability of success) = average number of coin flips needed which equals (1/(0.5)^12) = 4096.

SIMULATION CLASS

Variables:

sideOnTop: Integer variable that stores the current side on top (0 for heads, 1 for tails).

Constants:

DEFAULT_SIDE_ON_TOP: Static final integer constant with the default value for sideOnTop (0).

Libraries:

java.util: Used for the Random class to generate random numbers.

Constructors:

Default constructor:

1. Input arguments: None
2. Return type: None
3. Description: Initializes sideOnTop to the default value (0).

Non Default constructor:

1. Input arguments: (int newSideOnTop)
2. Return type: None
3. Description: Initializes sideOnTop to the provided value if it is valid (between 0 and 1). Otherwise, it remains at the default value (0).

Methods:

1. **setSideOnTop(int newSideOnTop):**
   - A. Input arguments: newSideOnTop (an integer representing the new side on top).
   - B. Return type: None
   - C. Description: Updates sideOnTop to the provided value if it is valid (between 0 and 1). Otherwise, it remains unchanged.

2. **getSideOnTop():**
   - A. Input arguments: None
   - B. Return type: int

C. Description: Returns the current value of sideOnTop (0 for heads, 1 for tails).

3. **toString():**
   A. Input arguments: None
   B. Return type: String
   C. Description: Currently just returns "flip = ".

4. **flip():**
   A. Input arguments: None
   B. Return type: int
   C. Description: Generates a random integer between 0 and 1, updates sideOnTop with the generated value, and returns the new value.

TEST CASES:

a. Default constructor should set the value of sideOnTop to zero.
b. Non-Default constructor can only set the value to zero or one in all other cases it becomes zero.
c. setSideOnTop can only set the value to zero or one in all other cases it becomes zero.
d. getSideOnTop should return only zero or one.
e. Flip should generate zero or one randomly and return the answer.

MAIN.JAVA

Constants:

A. **HEADS:** Integer constant representing heads (value: 0)
B. **TAILS:** Integer constant representing tails (value: 1)
C. **YES:** String constant representing "Y"
D. **NO:** String constant representing "N"
E. **percent:** DecimalFormat object used to format percentages ("##0%")

Libraries:

A. **Java.text.*:**
   DecimalFormat: Formats the percentage difference.
B. **Java.util.*:**

Random: Generates random numbers for the target head streak.
Scanner: Reads user input from the console.

1. Initialization:

A. **Simulation coin = new Simulation():** Creates a new Simulation object named coin.
B. **boolean playAgain=true:** Sets a boolean variable playAgain to true to initiate the main loop.

2. Main Loop:

A. **while(playAgain):** The main loop continues as long as playAgain is true.
B. **int flipCount=0:** Initializes a variable flipCount to keep track of the number of flips for each iteration.
C. **double sum=0:** Initializes a variable sum to accumulate the total number of flips across all iterations.

3. Target Streak:

A. **Random t = new Random():** Creates a new Random object to generate random numbers.
B. **int targetHeadStreak=(t.nextInt(7))+6:** Generates a random integer between 6 and 12 (inclusive) to represent the target head streak.
C. **System.out.println**("Your target run is "+ targetHeadStreak +" heads"): Prints the target head streak to the console.

4. User Guess:

A. **Scanner in = new Scanner(System.in):** Creates a new Scanner object to read user input.
B. **System.out.println**("What is your guess of how many coin tosses are needed on average to reach that exact target run?"): Prompts the user for their guess on the average number of flips.

5. Validating User Input:

A. **while(!in.hasNextInt()):** This loop keeps iterating until the user enters a valid integer.

B. **System.out.println("you did not type an int"):** Prints an error message if the user enters anything other than an integer.
C. **String garbage= in.next():** Reads and stores the invalid input to avoid affecting the next iteration.
D. **double yourGuess = in.nextInt():** Stores the user's valid integer guess in the yourGuess variable.

6. Simulation Loop:

A. **for(int i=1; i<=1000;i++):** This loop runs 1000 times to simulate the coin toss experiment.
B. **int headStreak=0:** Initializes a variable headStreak to keep track of the current head streak for each simulation.

7. Inner Loop:

A. **while(headStreak<targetHeadStreak):** This loop continues until the current head streak reaches the target streak.
B. **int randomValue = coin.flip():** Calls the flip() method of the coin object to simulate a coin toss.
C. **flipCount++:** Increments the flipCount for each toss.
D. **if(randomValue==HEADS):** If the toss is heads, increment the headStreak.
E. **else:** If the toss is tails, reset the headStreak to zero.

8. Calculating Average:

A. **sum+=flipCount:** Accumulates the total number of flips for each simulation in the sum variable.
B. **flipCount=0:** Resets the flipCount to zero for the next simulation.

9. Calculating and Reporting Accuracy:

A. **double average= sum/1000:** Calculates the average number of flips across all simulations.
B. **if(yourGuess<average):** If the user guessed less than the average, calculate the absolute difference and format it as a percentage using percent.format().
C. **else if(yourGuess>average):** If the user guessed more than the average, calculate the relative difference and format it as a percentage using percent.format().

D. **System.out.println(...):** Prints a message to the console informing the user of the average number of flips needed and their accuracy relative to the average.

10. Play Again:

A. **System.out.println**("Do you want to play again (Y=yes, N=no)?"): Prompts the user to play the game again.

11. Validating Play Again Input:

A. **while (!valid):** This loop ensures the user enters a valid input (Y or N) to continue or quit.
B. **System.out.println**("Please enter Y or N"): Prompts the user to enter a valid input.
C. **String input = in.next():** Reads the user's input.
if (!(input.equals(NO) || input.equals(YES))): Checks if the input is valid (Y or N). If invalid, the loop continues.

TEST CASES:

a. If a user enters a non-integer number, e.g String, generates a statement saying “you did not type an int” and throws an exception by asking you to go again and take another guess.
b. If the user enters any integer below the actual magnitude of the actual guess the answer will be off by less than hundred percent.
c. If the user enters any integer equal to the actual magnitude of the actual guess the answer will be off by zero percent.
d. If the user enters a guess double the magnitude of the actual guess the result may be two hundred percent and will change accordingly.
e. If the user enters zero the answer will be off by hundred percent.
f. If the user enters any integer which is negative it will consider only the magnitude and return the value accordingly.
g. On prompting to play again if the user enters a numeric value or a string other than “Y” or “N” it will generate an error message and throws an exception by asking you to go again and give an answer.
h. If the user Enters “Y” the boolean flag remains true and the loop runs again from the start.
i. If the user Enters “N” the boolean flag becomes false and the loop stops execution.