

Iteration FRQ 1

1. The method `longestStreak` is intended to determine the longest substring of consecutive identical characters in the parameter `str` and print the result.

For example, the call `longestStreak("CCAAAAATTT!")` should print the result "A 5" because the longest substring of consecutive identical characters is "AAAAA".

Write the code segment to be used in the body of the method below. The parameter `str` is a properly initialized `String` variable. Your implementation should conform to the example above.

```
public static void longestStreak(String str)
{
    String maxChar = "";
    int maxLen = 0;

    int cur = 0;
    String curChar = "";
    for(int i=0; i < str.length(); i++) {
        curChar = str.substring(i,i+1);
        cur = i+1;
        while(cur < str.length() &&
            str.substring(cur,cur+1).equals(curChar))
        {
            cur++;
        }
        if(cur-i > maxLen) {
            maxLen = cur-i;
            maxChar = curChar;
        }
    }
    System.out.println(maxChar + " " + maxLen);
}
```

// Total: 6

// +1 [Skill 3.C] Assess every character in `str` without bounds error

// +1 [Skill 3.C] Access all substrings of consecutive identical characters in `str` without bounds error

// +1 [Skill 3.C] Compares consecutive characters

// +1 [Skill 3.A] Gets individual characters from `str` using calls to `substring`

// +1 [Skill 3.C] Updates length and character of longest consecutive identical characters appropriately

// +1 [Skill 3.C] Prints the correct longest streak of characters

// Penalties

// -1 (w) Extraneous code that causes side-effect (e.g., printing to output, incorrect precondition check)

// -1 (x) Local variables used but none declared

// -1 (y) Destruction of persistent data (e.g. changing value referenced by parameter)

// -1 (z) Void method or constructor that returns a value

Iteration FRQ 1

2. This question involves a simulation of a two-player game. In the game, two simulated players each start out with an equal number of coins. In each round, each player chooses to spend either 1, 2, or 3 coins. Coins are then awarded to each player according to the following rules.

- *Same rule:* If both players spend the same number of coins, player 2 is awarded 1 coin.
- *Off-by-one rule:* If the players do not spend the same number of coins and the positive difference between the number of coins spent by the two players is 1, player 2 is awarded 1 coin.
- *Off-by-two rule:* If the players do not spend the same number of coins and the positive difference between the number of coins spent by the two players is 2, player 1 is awarded 2 coins.

The following is an example of a game played with a starting value of 10 coins and a game length of 5 rounds.

Round Number	Player 1 Coin Count At Beginning of Round	Player 2 Coin Count At Beginning of Round	Player 1 Spends	Player 2 Spends	Outcome	Player 1 Coin Count At End of Round	Player 2 Coin Count At End of Round
1	10	10	2	1	Off-by-one	$10 - 2 = 8$	$10 - 1 + 1 = 10$
2	8	10	2	2	Same	$8 - 2 = 6$	$10 - 2 + 1 = 9$
3	6	9	1	3	Off-by-two	$6 - 1 + 2 = 7$	$9 - 3 = 6$
4	7	6	2	2	Same	$7 - 2 = 5$	$6 - 2 + 1 = 5$
5	5	5	3	1	Off-by-two	$5 - 3 + 2 = 4$	$5 - 1 = 4$
End of Game	4	4			Tie game		

The game ends when the specified number of rounds have been played or when a player's coin count is less than 3 at the end of a round.

The `CoinGame` class contains the following instance variables and methods. You will write code segments to complete two methods in the `CoinGame` class.

- `int startingCoins;` - an instance variable that represents the starting number of coins for each player
- `int maxRounds;` - an instance variable that represents the maximum number of rounds played in the game
- `public int getPlayer1Move()` - Returns the number of coins (1, 2, or 3) that player 1 will spend.
- `public int getPlayer2Move(int round)` - Returns the number of coins (1, 2, or 3) that player 2 will spend. You will write a code segment for the body of this method, as described in part (a).
- `public void playGame()` - Plays a simulated game between two players. You will write a code segment for the body of this method, as described in part (b).

In the simulation, player 2 will always play according to the same strategy. The number of coins player 2 spends is based on what round it is, as described below.

Iteration FRQ 1

2.

- (a) You will write the code segment for the body of the method `getPlayer2Move`, which returns the number of coins that player 2 will spend in a given round of the game. In the first round of the game, the parameter `round` has the value 1, in the second round of the game, it has the value 2, and so on. The method returns 1, 2, or 3 based on the following rules.

- If `round` is divisible by 3, then return 3.
- If `round` is not divisible by 3 but is divisible by 2, then return 2.
- If `round` is not divisible by 3 and is not divisible by 2, then return 1.

Write the code segment to complete the method `getPlayer2Move` below by assigning the correct value to `result` to be returned. The parameter `round` is a properly initialized `int` variable.

```
public static int getPlayer2Move(int round)
{
    int result;
    //Your code segment will be inserted here.
    if(round%3 == 0) {
        result = 3;
    } else if(round%2 == 0) {
        result = 2;
    } else {
        result = 1;
    }

    return result;
}
```

// **Total: 2**
// +1 [Skill 3.C] Checks for divisibility by 2 or 3
// +1 [Skill 3.C] Correctly assigns 1, 2, or 3, to be returned
// **Penalties**
// -1 (w) Extraneous code that causes side-effect (e.g., printing to output, incorrect precondition check)

You will write the code segment for the body of the method `playGame`, which simulates a game between player 1 and player 2, based on the rules and example shown at the beginning of the question. Both player 1 and player 2 start the game with `startingCoins` coins. Computer player 1 spends 1, 2, or 3 coins based on the value returned by the method `getPlayer1Move()`. Computer player 2 spends 1, 2, or 3 coins based on the value returned by the method `getPlayer2Move(int round)`.

The game ends when `maxRounds` rounds have been played or when a player's coin count is less than 3 at the end of a round.

At the end of the game, the winner is determined according to the following rules.

If both players have the same number of coins at the end of the game, the method prints "tie game".

If player 1 has more coins than player 2, the method prints "player 1 wins".

If player 2 has more coins than player 1, the method prints "player 2 wins".

Iteration FRQ 1

- (b) Assume that `getPlayer2Move` works as specified, regardless of what you wrote in part (a). You must use `getPlayer1Move` and `getPlayer2Move` appropriately to receive full credit.

Write the code segment to complete the method `playGame()` below.

```
public static void playGame()
{
    int p1Coins, p2Coins;
    int p1Move, p2Move;
    p1Coins = p2Coins = startingCoins;
    int round = 1;

    while(round<=maxRounds && p1Coins > 3 && p2Coins > 3) {

        p1Coins -= p1Move = getPlayer1Move(round);
        p2Coins -= p2Move = getPlayer2Move(round);

        // Same rule:
        if(p1Move == p2Move) {
            p2Coins++;
        }

        // Off-by-one rule:
        if(Math.abs(p1Move-p2Move) == 1) {
            p2Coins++;
        }

        // Off-by-two rule:
        if(Math.abs(p1Move-p2Move) == 2) {
            p1Coins+=2;
        }

        round++;
    }
    printWinner(p1Coins,p2Coins);
}

public static void printWinner(int p1Coins, int p2Coins) {
    if(p1Coins > p2Coins) {
        System.out.println("player 1 wins");
    } else if (p2Coins > p1Coins) {
        System.out.println("player 2 wins");
    } else {
        System.out.println("tie game");
    }
}
```

Iteration FRQ 1**// Total: 7**

```
// +1 [Skill 3.C] Uses instance variables to set initial coins and number of rounds to play
// +1 [Skill 3.C] Loops appropriate number of times and ends when maxRounds is
    reached or when either player's coin count is less than 3 at the end of the round
// +1 [Skill 3.A] For each round of the game, calls getPlayer1Move and calls
    getPlayer2Move with parameter reflecting round number
// +1 [Skill 3.C] Compares coins spent to determine equality, off-by-one, or off-by-two
// +1 [Skill 3.C] Updates each player's coin total based on results and rules of the game
// +1 [Skill 3.C] Determines winner based on number of coins at end of game
// +1 [Skill 3.C] Prints winner or tie in the specified format
```

// Penalties

```
// -1 (w) Extraneous code that causes side-effect (e.g., printing to output, incorrect
    precondition check)
// -1 (x) Local variables used but none declared
```