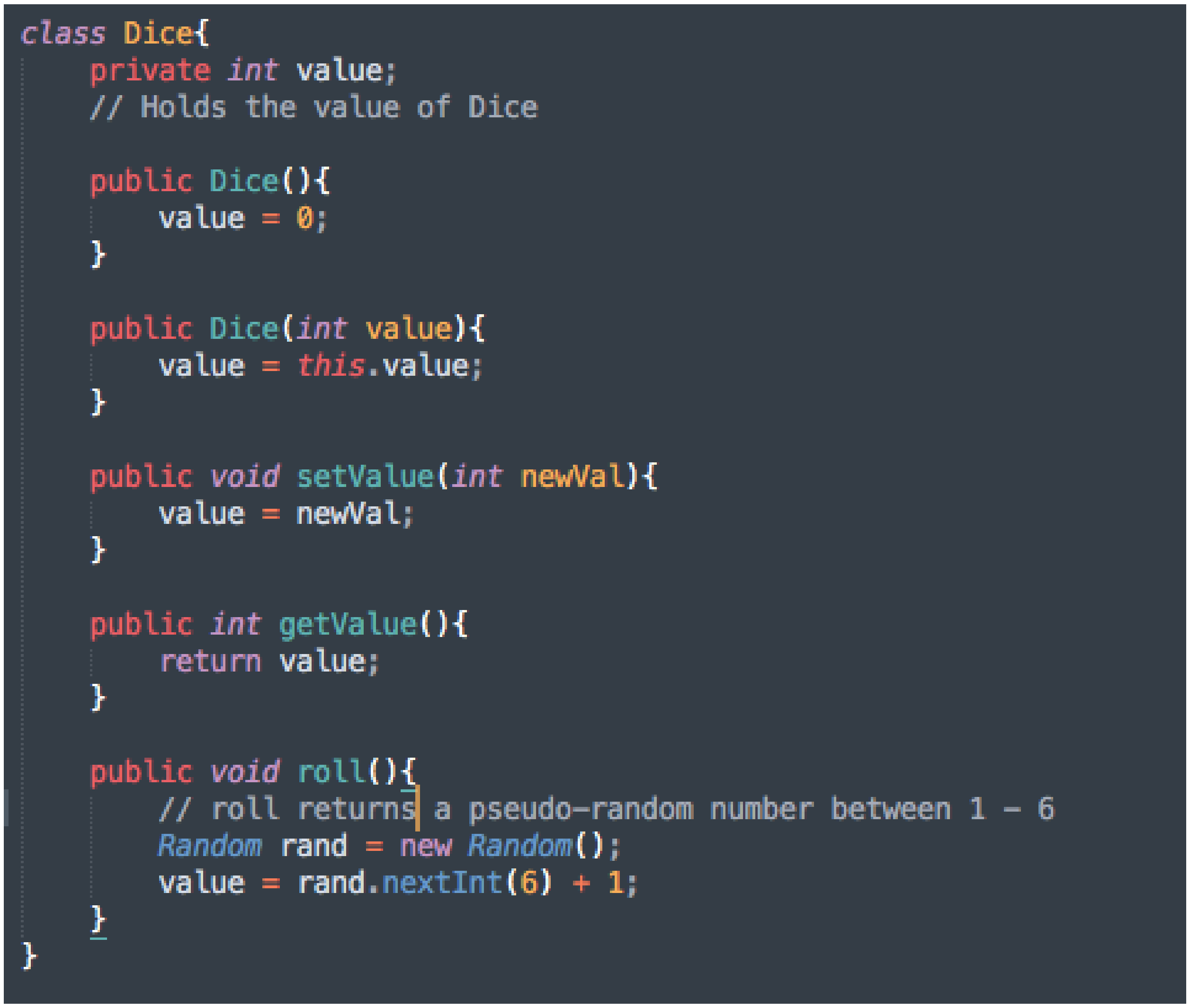
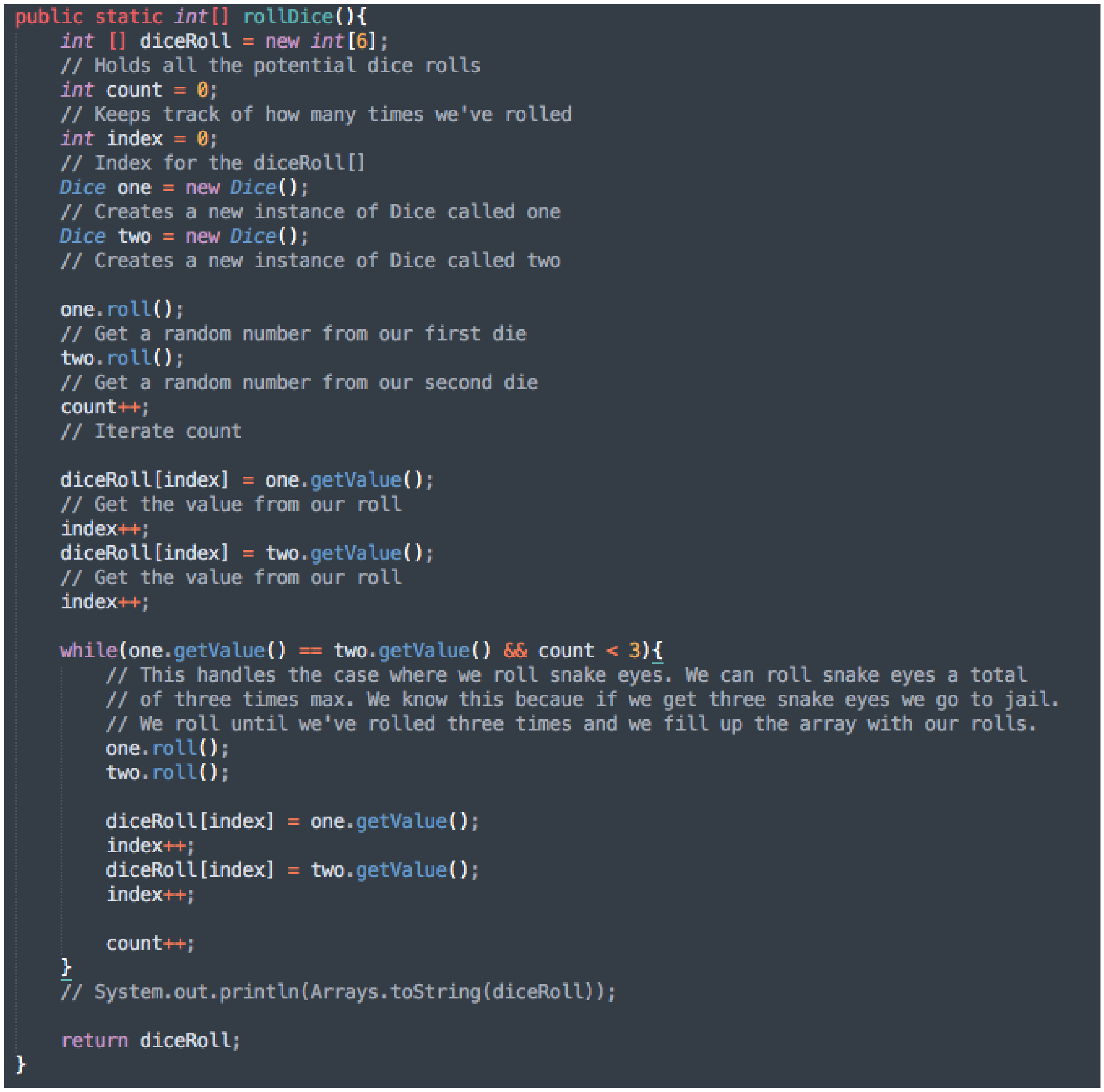
Section 1: Monopoly

1. **Write code to generate random numbers corresponding to rolling a pair of dice and summing the ouput taking into account the doubling rule (when you roll doubles you get to roll again). Include all the code with comments describing what you are doing (*) (1pt)**



```java
class Dice{
    private int value;
    // Holds the value of Dice

    public Dice(){
        value = 0;
    }

    public Dice(int value){
        value = this.value;
    }

    public void setValue(int newVal){
        value = newVal;
    }

    public int getValue(){
        return value;
    }

    public void roll(){
        // roll returns a pseudo-random number between 1 - 6
        Random rand = new Random();
        value = rand.nextInt(6) + 1;
    }
}
```

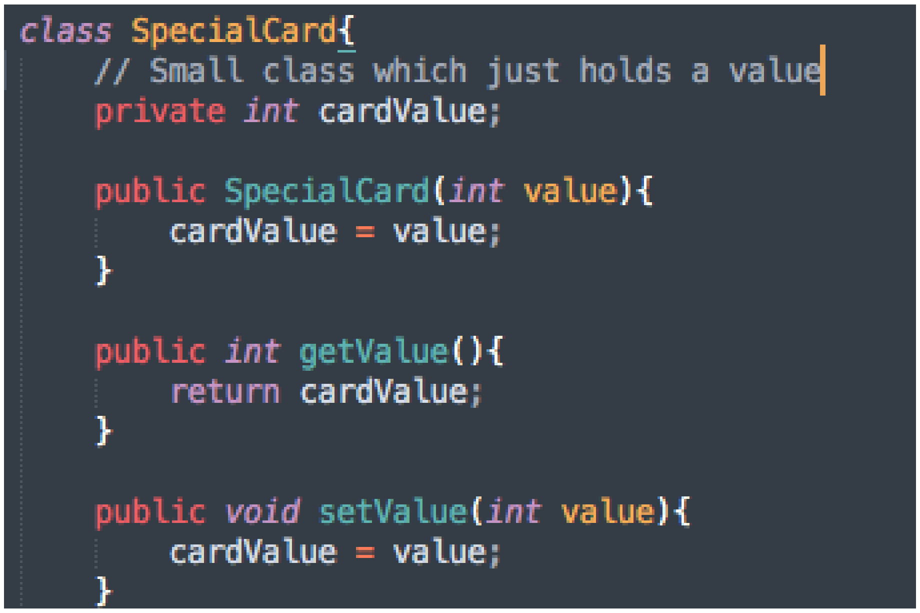


```java
public static int[] rollDice(){
    int [] diceRoll = new int[6];
    // Holds all the potential dice rolls
    int count = 0;
    // Keeps track of how many times we've rolled
    int index = 0;
    // Index for the diceRoll[]
    Dice one = new Dice();
    // Creates a new instance of Dice called one
    Dice two = new Dice();
    // Creates a new instance of Dice called two

    one.roll();
    // Get a random number from our first die
    two.roll();
    // Get a random number from our second die
    count++;
    // Iterate count

    diceRoll[index] = one.getValue();
    // Get the value from our roll
    index++;
    diceRoll[index] = two.getValue();
    // Get the value from our roll
    index++;

    while(one.getValue() == two.getValue() && count < 3){
        // This handles the case where we roll snake eyes. We can roll snake eyes a total
        // of three times max. We know this becaue if we get three snake eyes we go to jail.
        // We roll until we've rolled three times and we fill up the array with our rolls.
        one.roll();
        two.roll();

        diceRoll[index] = one.getValue();
        index++;
        diceRoll[index] = two.getValue();
        index++;

        count++;
    }
    // System.out.println(Arrays.toString(diceRoll));

    return diceRoll;
}
```

For the dice in our monopoly game a class called Dice was created. This class held one variable, value, and the main method we're concerned about is roll(). This method returns a random integer between 1 and 6. It then saves that number as our value and also returns that value.

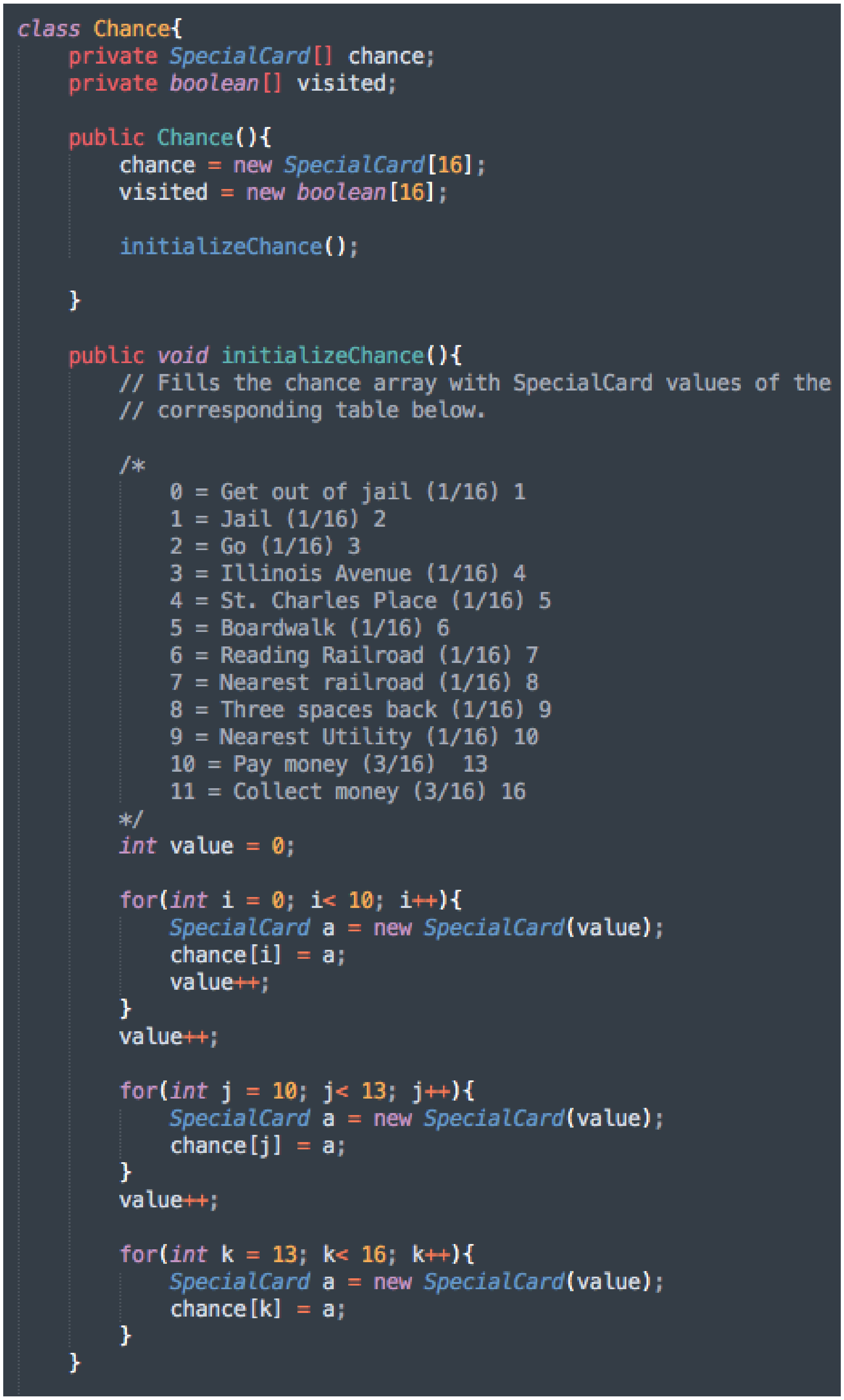
In the method rollDice() the cases with the actual rolling of the dice are handled. two instances of Dice are created and rolled. All of the rolls are stored in an integer array of size six. This means a maximum of three rolls are possible. If three snake eyes are rolled, the player would go straight to jail. This part is handled in another method. The integer array is then returned from this method.

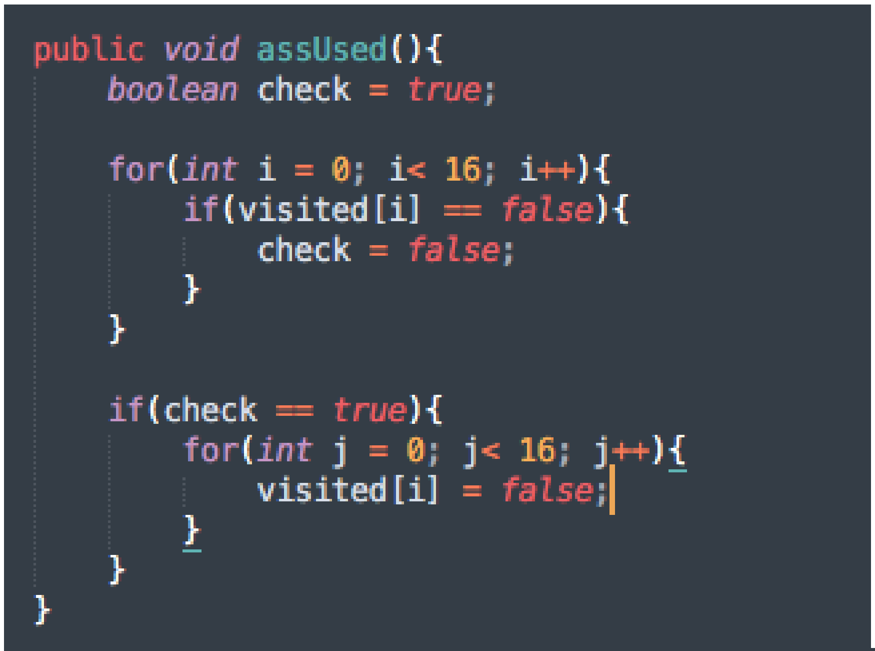
2. **Write code for selecting a chance card at random based on the information provided in the article about the math of Monopoly (*) (1pt)**



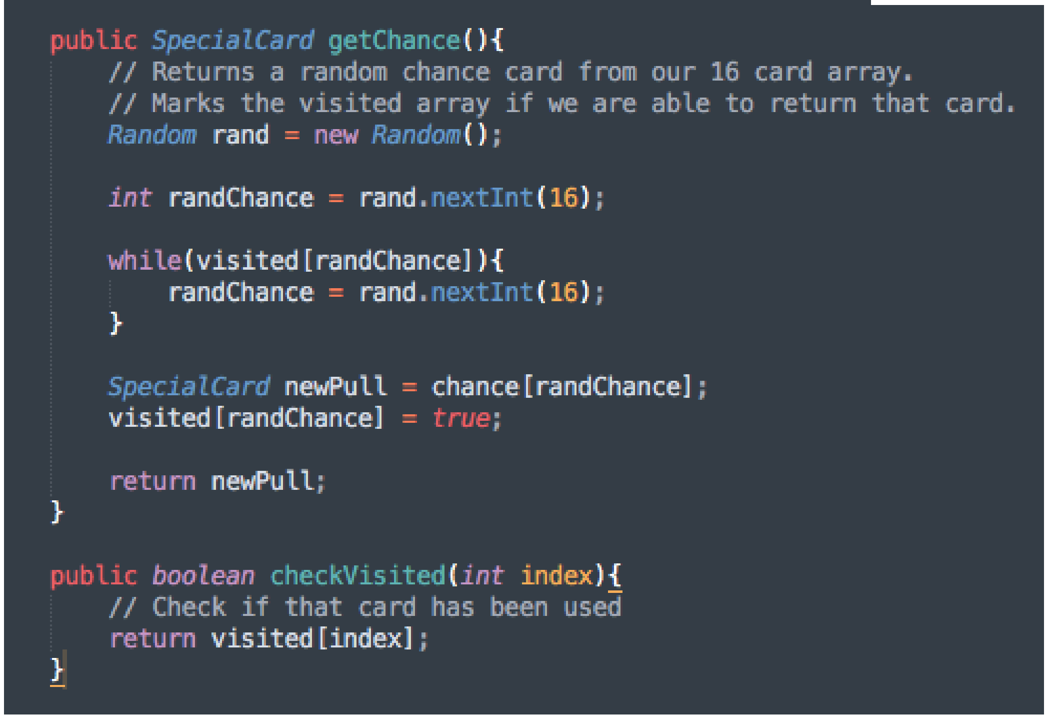
```java
class SpecialCard{
    // Small class which just holds a value
    private int cardValue;

    public SpecialCard(int value){
        cardValue = value;
    }

    public int getValue(){
        return cardValue;
    }

    public void setValue(int value){
        cardValue = value;
    }
```

Both chance and community chest are implemented with the class SpecialCard. SpecialCard is similar to Dice in that it just holds a value.

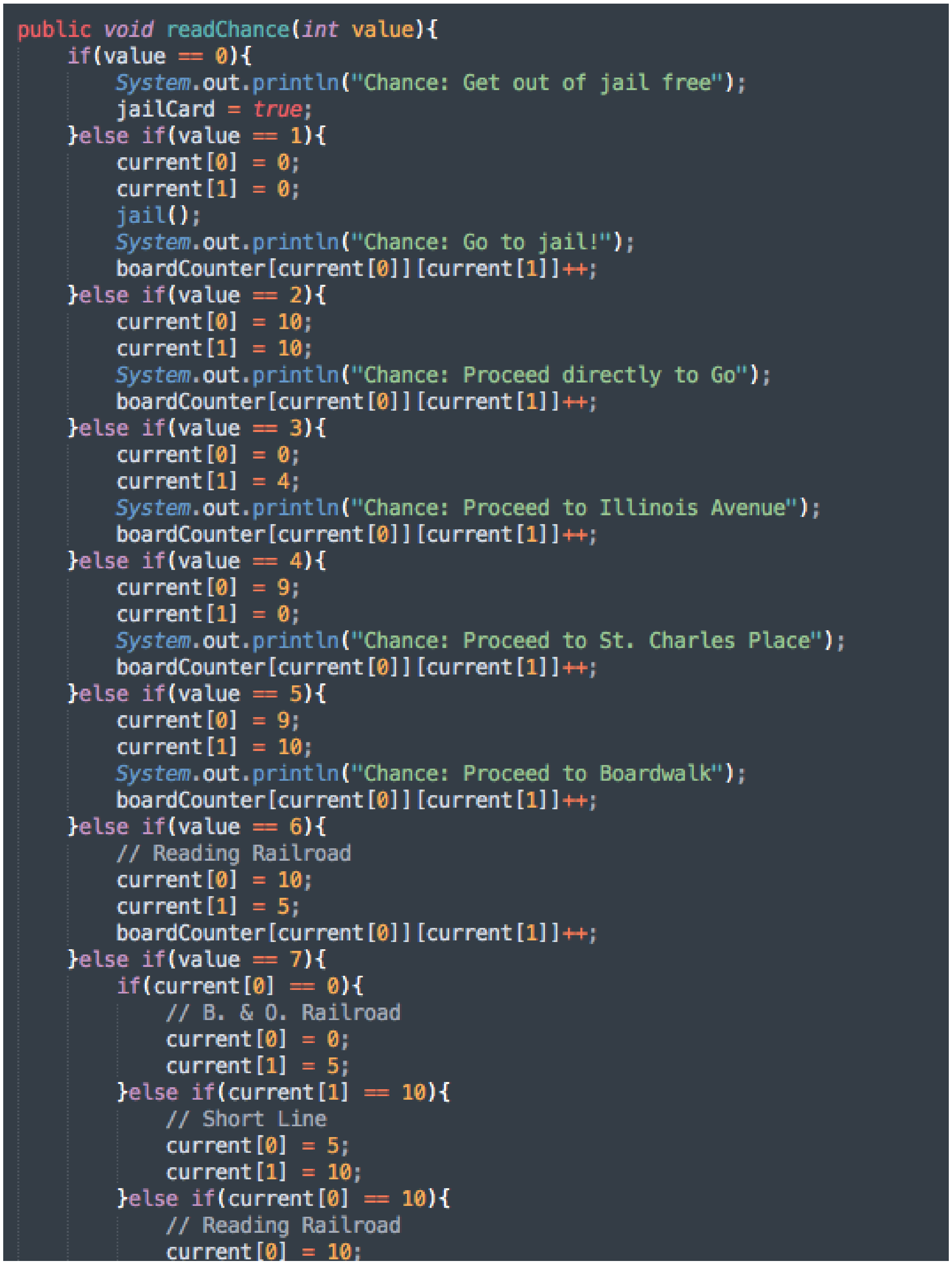

```java
class Chance{
    private SpecialCard[] chance;
    private boolean[] visited;

    public Chance(){
        chance = new SpecialCard[16];
        visited = new boolean[16];

        initializeChance();

    }

    public void initializeChance(){
        // Fills the chance array with SpecialCard values of the
        // corresponding table below.

        /*
            0 = Get out of jail (1/16) 1
            1 = Jail (1/16) 2
            2 = Go (1/16) 3
            3 = Illinois Avenue (1/16) 4
            4 = St. Charles Place (1/16) 5
            5 = Boardwalk (1/16) 6
            6 = Reading Railroad (1/16) 7
            7 = Nearest railroad (1/16) 8
            8 = Three spaces back (1/16) 9
            9 = Nearest Utility (1/16) 10
            10 = Pay money (3/16)  13
            11 = Collect money (3/16) 16
        */
        int value = 0;

        for(int i = 0; i< 10; i++){
            SpecialCard a = new SpecialCard(value);
            chance[i] = a;
            value++;
        }
        value++;

        for(int j = 10; j< 13; j++){
            SpecialCard a = new SpecialCard(value);
            chance[j] = a;
        }
        value++;

        for(int k = 13; k< 16; k++){
            SpecialCard a = new SpecialCard(value);
            chance[k] = a;
        }
    }
```

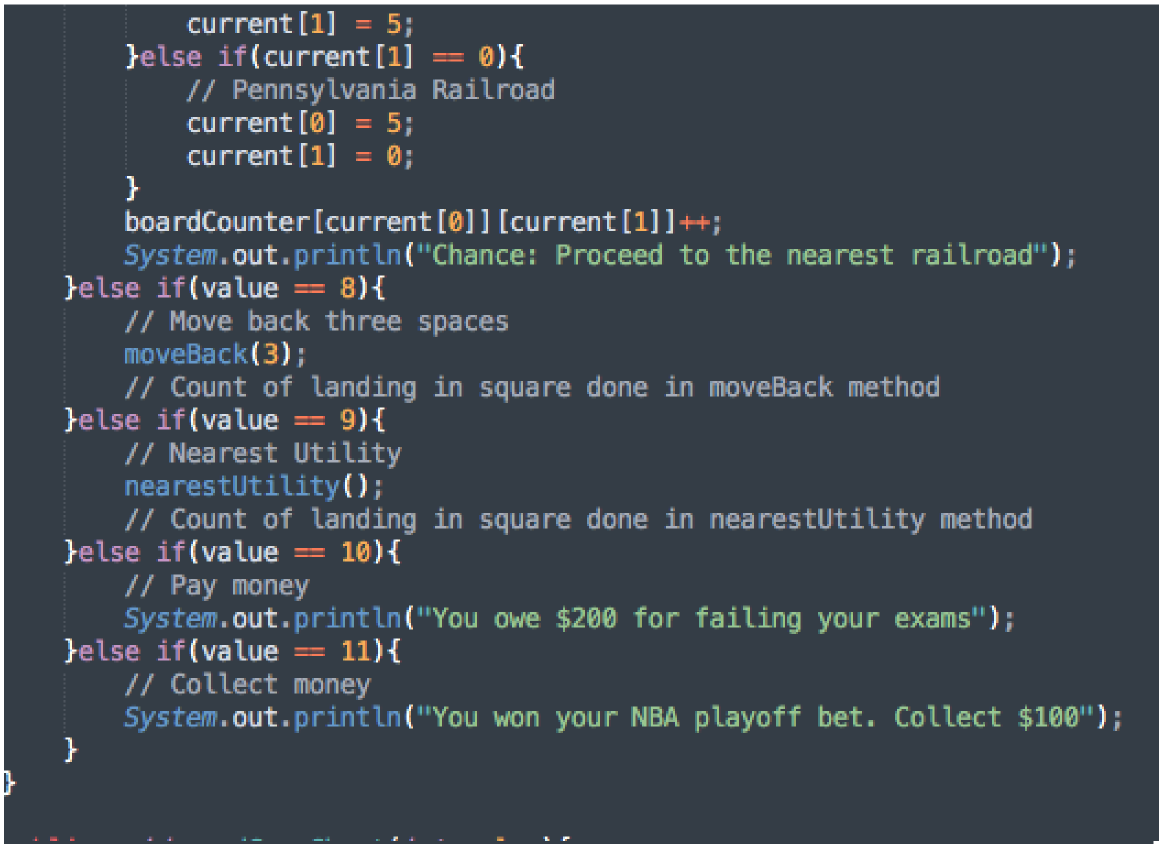


```
public void assUsed(){
    boolean check = true;

    for(int i = 0; i< 16; i++){
        if(visited[i] == false){
            check = false;
        }
    }

    if(check == true){
        for(int j = 0; j< 16; j++){
            visited[i] = false;
        }
    }
}
```

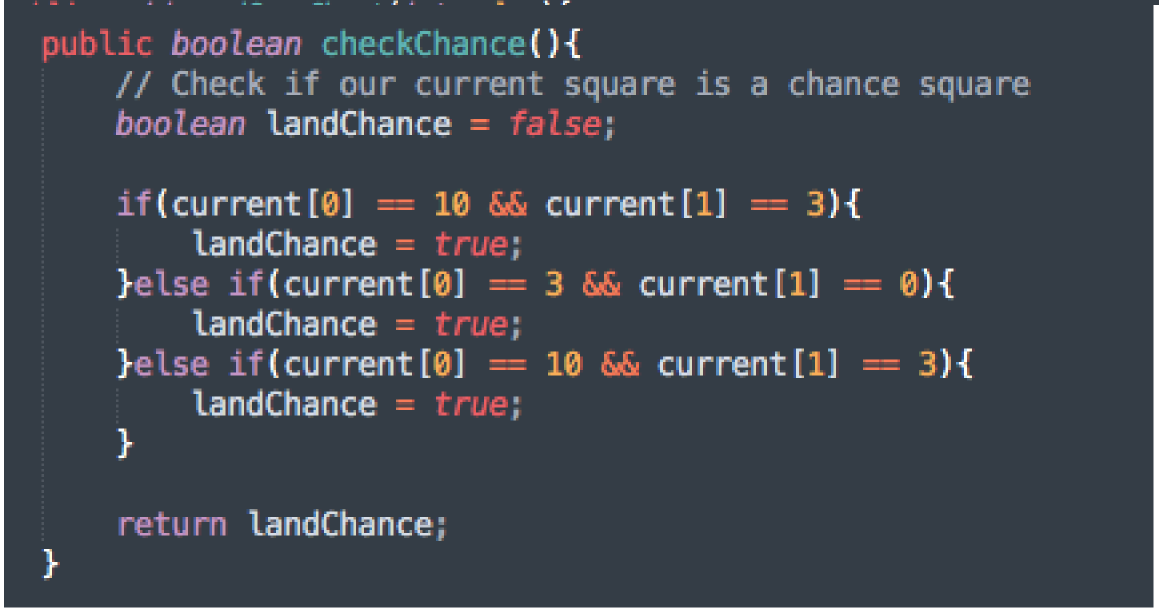


```
public SpecialCard getChance(){
    // Returns a random chance card from our 16 card array.
    // Marks the visited array if we are able to return that card.
    Random rand = new Random();

    int randChance = rand.nextInt(16);

    while(visited[randChance]){
        randChance = rand.nextInt(16);
    }

    SpecialCard newPull = chance[randChance];
    visited[randChance] = true;

    return newPull;
}

public boolean checkVisited(int index){
    // Check if that card has been used
    return visited[index];
}
```

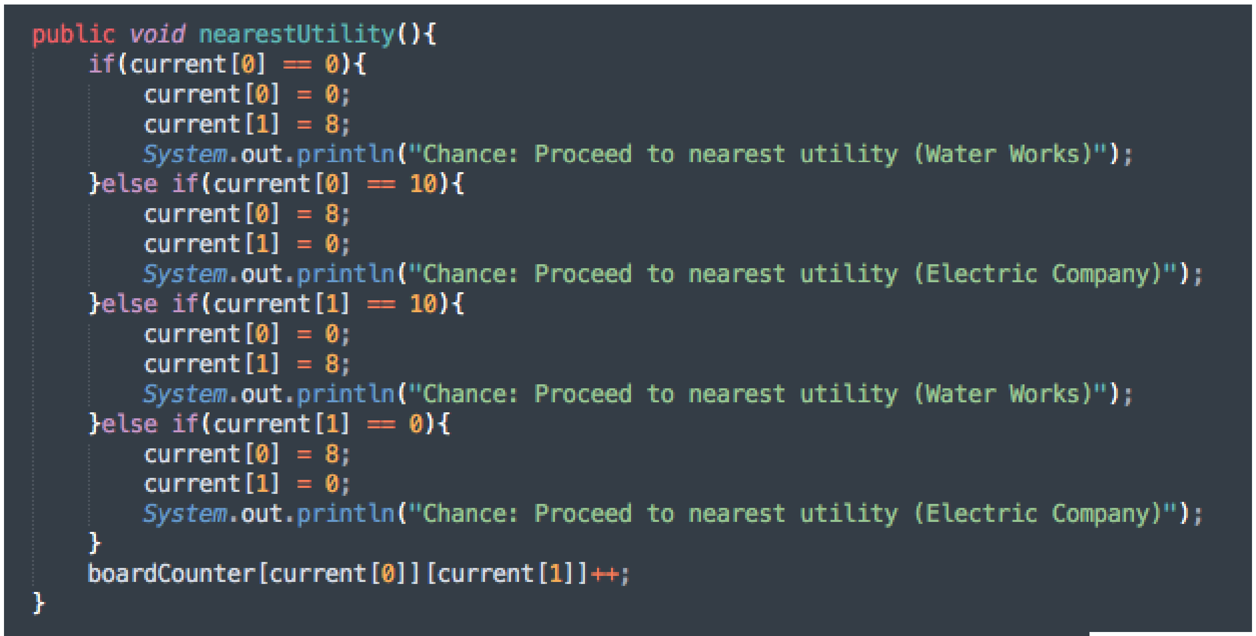

```java
public void readChance(int value){
    if(value == 0){
        System.out.println("Chance: Get out of jail free");
        jailCard = true;
    }else if(value == 1){
        current[0] = 0;
        current[1] = 0;
        jail();
        System.out.println("Chance: Go to jail!");
        boardCounter[current[0]][current[1]]++;
    }else if(value == 2){
        current[0] = 10;
        current[1] = 10;
        System.out.println("Chance: Proceed directly to Go");
        boardCounter[current[0]][current[1]]++;
    }else if(value == 3){
        current[0] = 0;
        current[1] = 4;
        System.out.println("Chance: Proceed to Illinois Avenue");
        boardCounter[current[0]][current[1]]++;
    }else if(value == 4){
        current[0] = 9;
        current[1] = 0;
        System.out.println("Chance: Proceed to St. Charles Place");
        boardCounter[current[0]][current[1]]++;
    }else if(value == 5){
        current[0] = 9;
        current[1] = 10;
        System.out.println("Chance: Proceed to Boardwalk");
        boardCounter[current[0]][current[1]]++;
    }else if(value == 6){
        // Reading Railroad
        current[0] = 10;
        current[1] = 5;
        boardCounter[current[0]][current[1]]++;
    }else if(value == 7){
        if(current[0] == 0){
            // B. & O. Railroad
            current[0] = 0;
            current[1] = 5;
        }else if(current[1] == 10){
            // Short Line
            current[0] = 5;
            current[1] = 10;
        }else if(current[0] == 10){
            // Reading Railroad
            current[0] = 10;
```



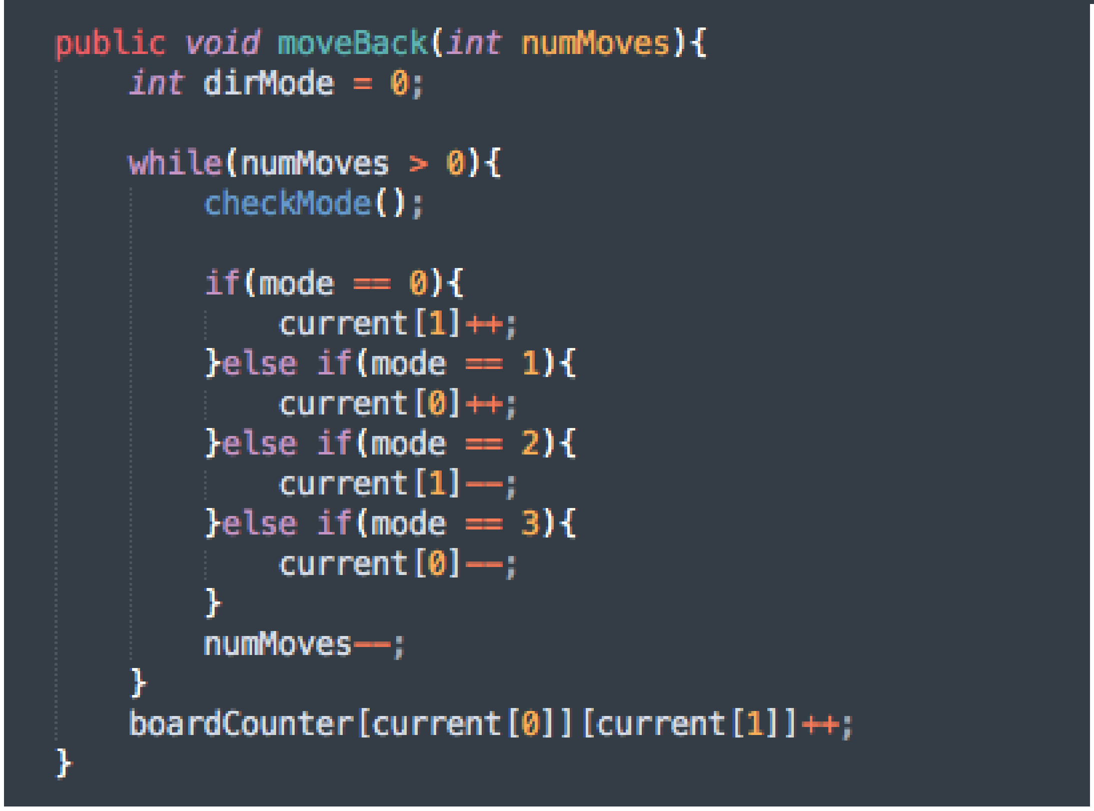
```java
            current[1] = 5;
        }else if(current[1] == 0){
            // Pennsylvania Railroad
            current[0] = 5;
            current[1] = 0;
        }
        boardCounter[current[0]][current[1]]++;
        System.out.println("Chance: Proceed to the nearest railroad");
    }else if(value == 8){
        // Move back three spaces
        moveBack(3);
        // Count of landing in square done in moveBack method
    }else if(value == 9){
        // Nearest Utility
        nearestUtility();
        // Count of landing in square done in nearestUtility method
    }else if(value == 10){
        // Pay money
        System.out.println("You owe $200 for failing your exams");
    }else if(value == 11){
        // Collect money
        System.out.println("You won your NBA playoff bet. Collect $100");
    }
}
```



```java
public boolean checkChance(){
    // Check if our current square is a chance square
    boolean landChance = false;

    if(current[0] == 10 && current[1] == 3){
        landChance = true;
    }else if(current[0] == 3 && current[1] == 0){
        landChance = true;
    }else if(current[0] == 10 && current[1] == 3){
        landChance = true;
    }

    return landChance;
}
```



```
public void nearestUtility(){
    if(current[0] == 0){
        current[0] = 0;
        current[1] = 8;
        System.out.println("Chance: Proceed to nearest utility (Water Works)");
    }else if(current[0] == 10){
        current[0] = 8;
        current[1] = 0;
        System.out.println("Chance: Proceed to nearest utility (Electric Company)");
    }else if(current[1] == 10){
        current[0] = 0;
        current[1] = 8;
        System.out.println("Chance: Proceed to nearest utility (Water Works)");
    }else if(current[1] == 0){
        current[0] = 8;
        current[1] = 0;
        System.out.println("Chance: Proceed to nearest utility (Electric Company)");
    }
    boardCounter[current[0]][current[1]]++;
}
```



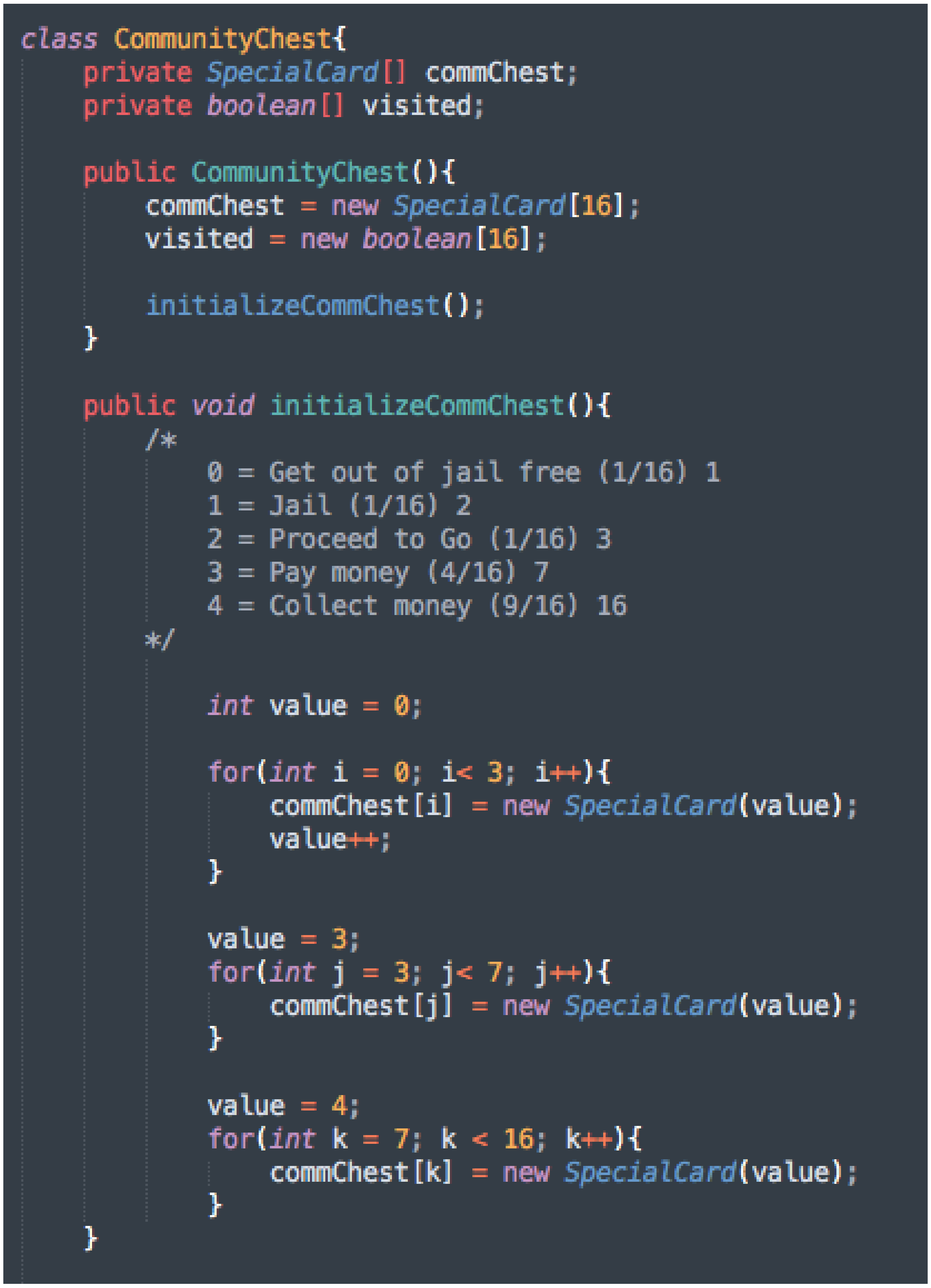
```
public void moveBack(int numMoves){
    int dirMode = 0;

    while(numMoves > 0){
        checkMode();

        if(mode == 0){
            current[1]++;
        }else if(mode == 1){
            current[0]++;
        }else if(mode == 2){
            current[1]--;
        }else if(mode == 3){
            current[0]--;
        }
        numMoves--;
    }
    boardCounter[current[0]][current[1]]++;
}
```

Chance is implemented using a class called Chance. This class holds an array of 16 SpecialCards. The initializeChance method fills the array with SpecialCards of the corresponding values. The most important method of this class is getChance. This method generates a random number between 1 and 15. If the card hasn't been used (visited[random] = false), then that card is returned and the visited array is set to true. If it has been used, we generate random numbers until we find a valid card. The method allUsed, checks if all of the cards have been used.

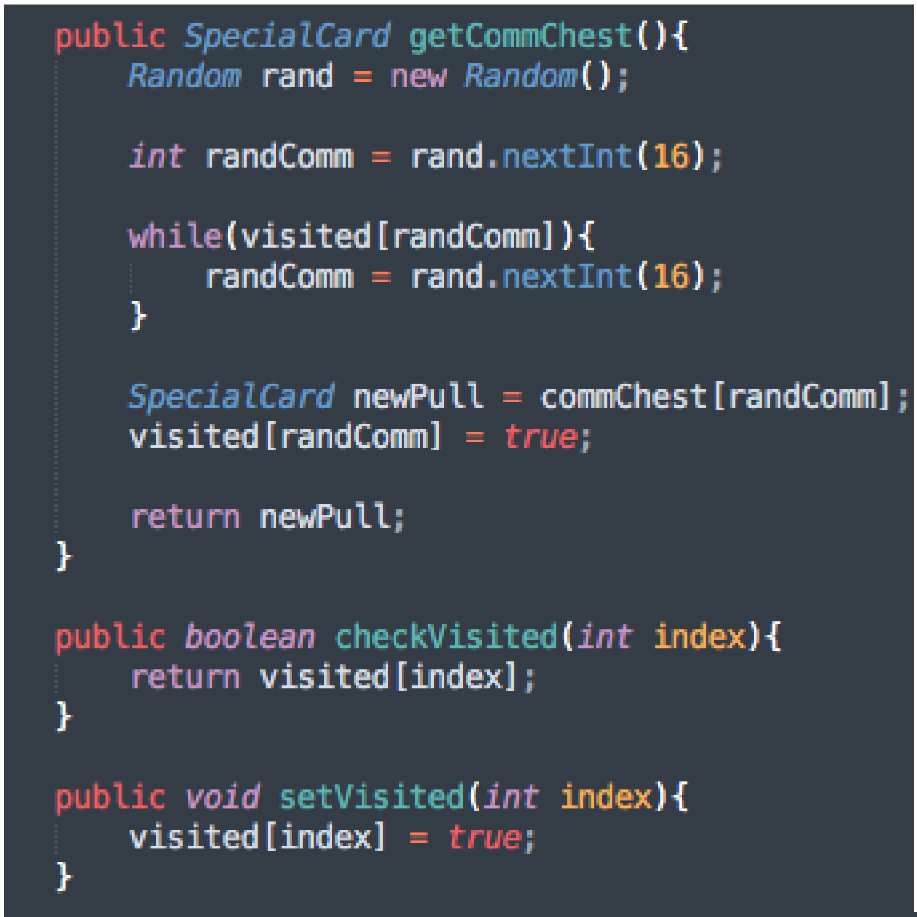
The actual reading of the card happens in the playGame method, which will be detailed later. The method readChance is displayed above. This method reads the value returned in the SpecialCard and performs a set of actions depending on what value is revealed. A get out of jail free card was implemented inside our GameBoard class. It's a boolean variable which we can set to be true or false. When we go to jail, it checks the GameBoard for this variable and lets the play go free if it is true.

The method checkChance checks if the space we landed on is a chance space. This is called every time we finish a turn. Furthermore, the method nearestUtility was written to handle that specific chance card. According to where our current spot on the board is, it sends us to the nearest utility square. Another card that had to be implemented separately is move back three spaces. The code for it is in the moveBack method. It checks for the mode to handle the cases where we switch directions on the corners of the board.

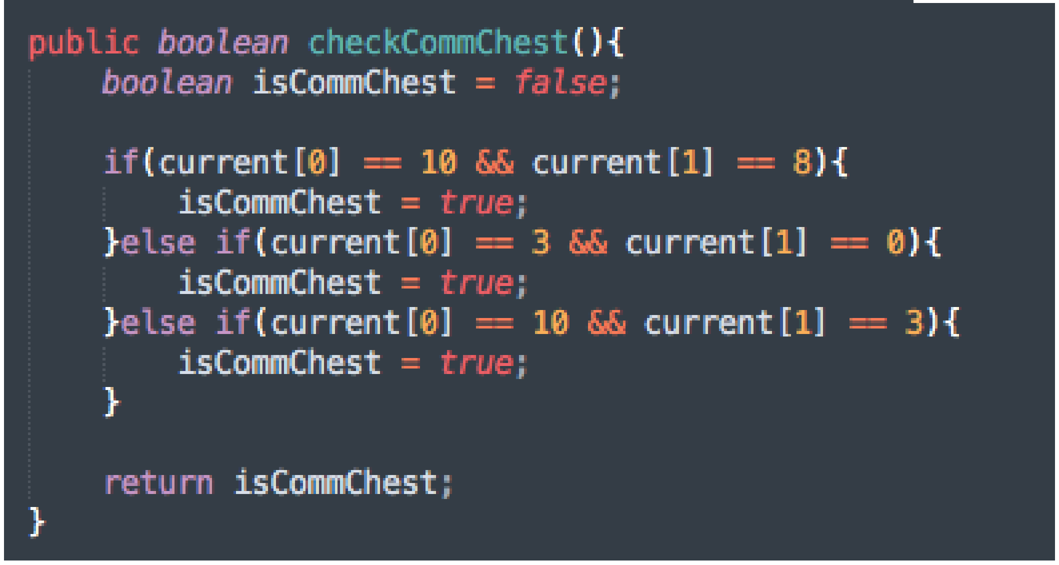
The code for community chest is included below, it runs very similar to chance so snippets will be shown.



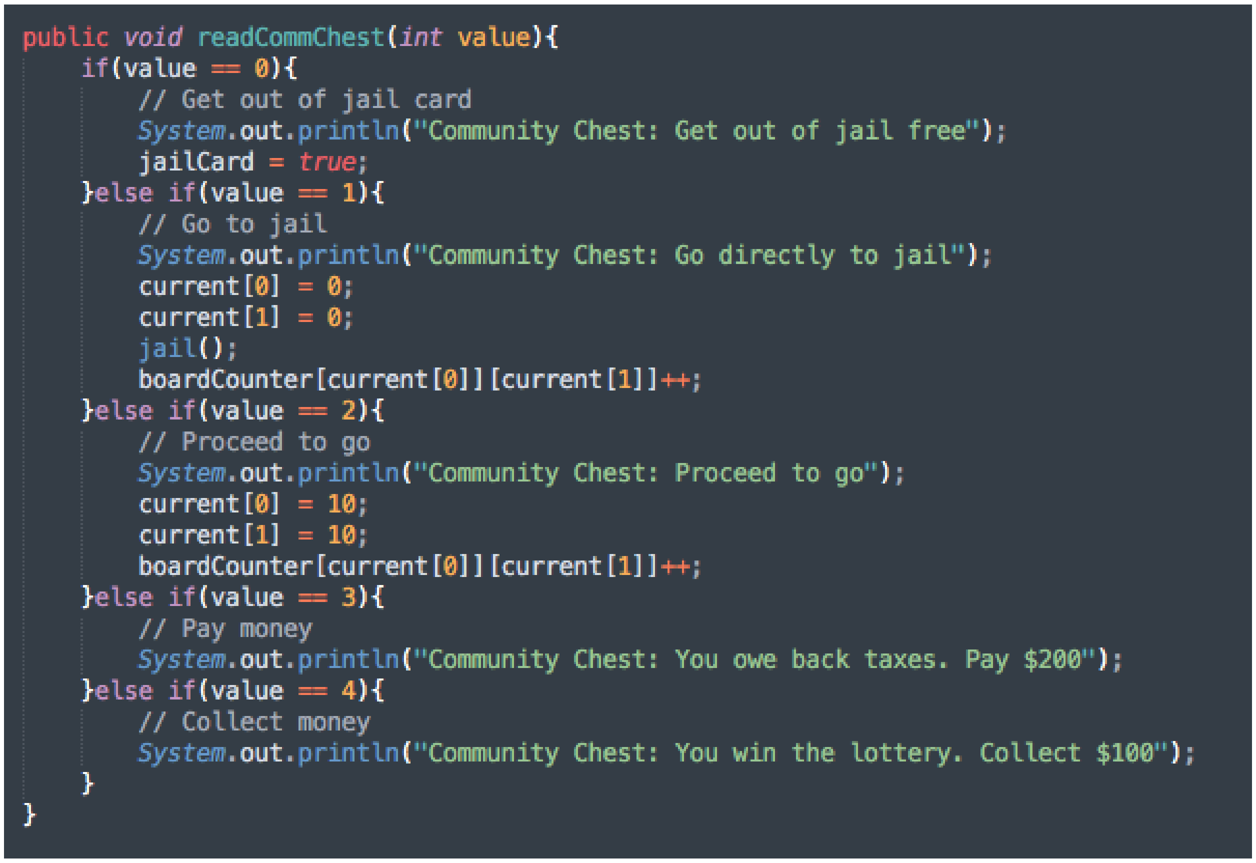
```java
class CommunityChest{
    private SpecialCard[] commChest;
    private boolean[] visited;

    public CommunityChest(){
        commChest = new SpecialCard[16];
        visited = new boolean[16];

        initializeCommChest();
    }

    public void initializeCommChest(){
        /*
            0 = Get out of jail free (1/16) 1
            1 = Jail (1/16) 2
            2 = Proceed to Go (1/16) 3
            3 = Pay money (4/16) 7
            4 = Collect money (9/16) 16
        */

            int value = 0;

            for(int i = 0; i< 3; i++){
                commChest[i] = new SpecialCard(value);
                value++;
            }

            value = 3;
            for(int j = 3; j< 7; j++){
                commChest[j] = new SpecialCard(value);
            }

            value = 4;
            for(int k = 7; k < 16; k++){
                commChest[k] = new SpecialCard(value);
            }
    }
```



```
public SpecialCard getCommChest(){
    Random rand = new Random();

    int randComm = rand.nextInt(16);

    while(visited[randComm]){
        randComm = rand.nextInt(16);
    }

    SpecialCard newPull = commChest[randComm];
    visited[randComm] = true;

    return newPull;
}

public boolean checkVisited(int index){
    return visited[index];
}

public void setVisited(int index){
    visited[index] = true;
}
```

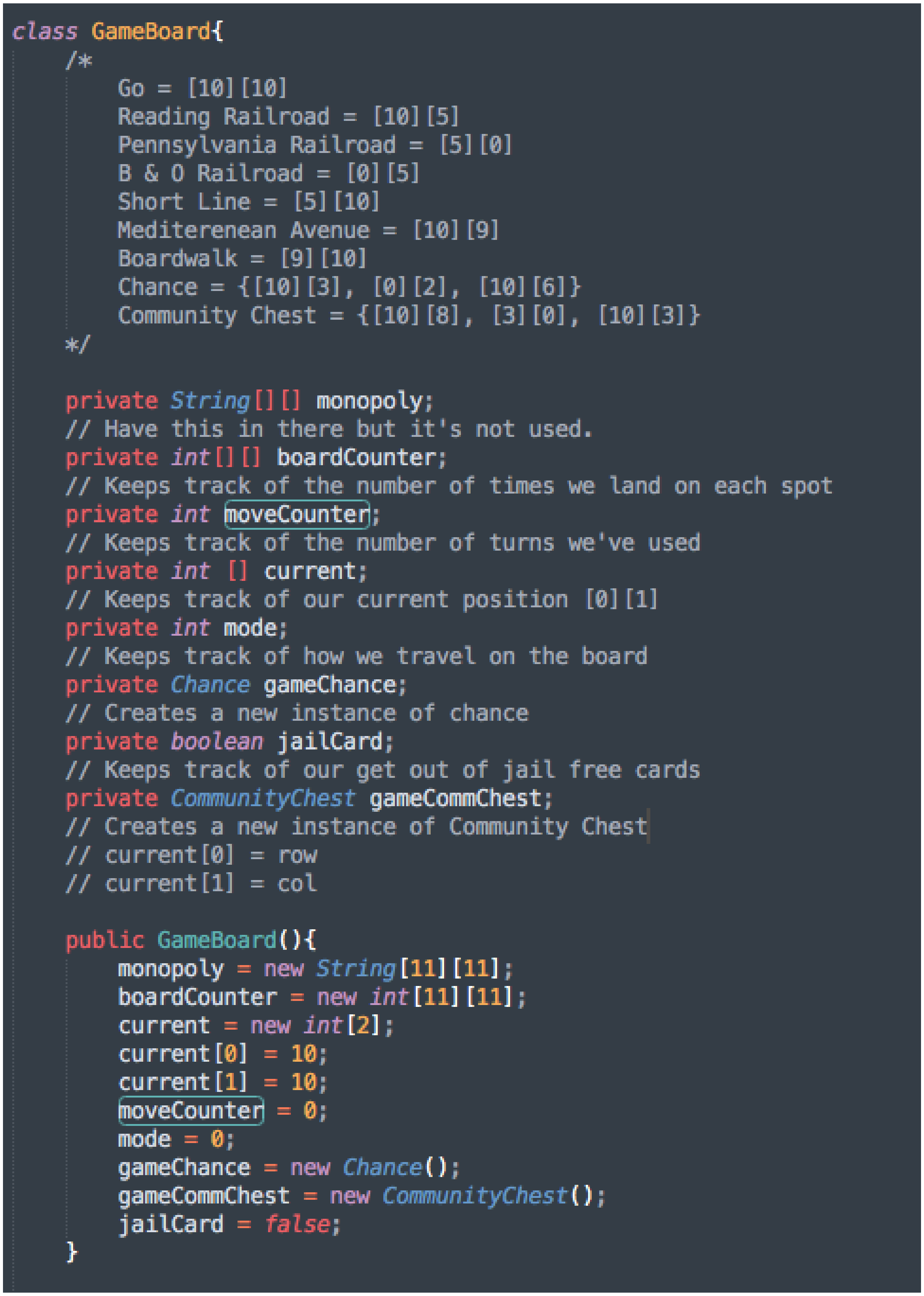


```
public boolean checkCommChest(){
    boolean isCommChest = false;

    if(current[0] == 10 && current[1] == 8){
        isCommChest = true;
    }else if(current[0] == 3 && current[1] == 0){
        isCommChest = true;
    }else if(current[0] == 10 && current[1] == 3){
        isCommChest = true;
    }

    return isCommChest;
}
```

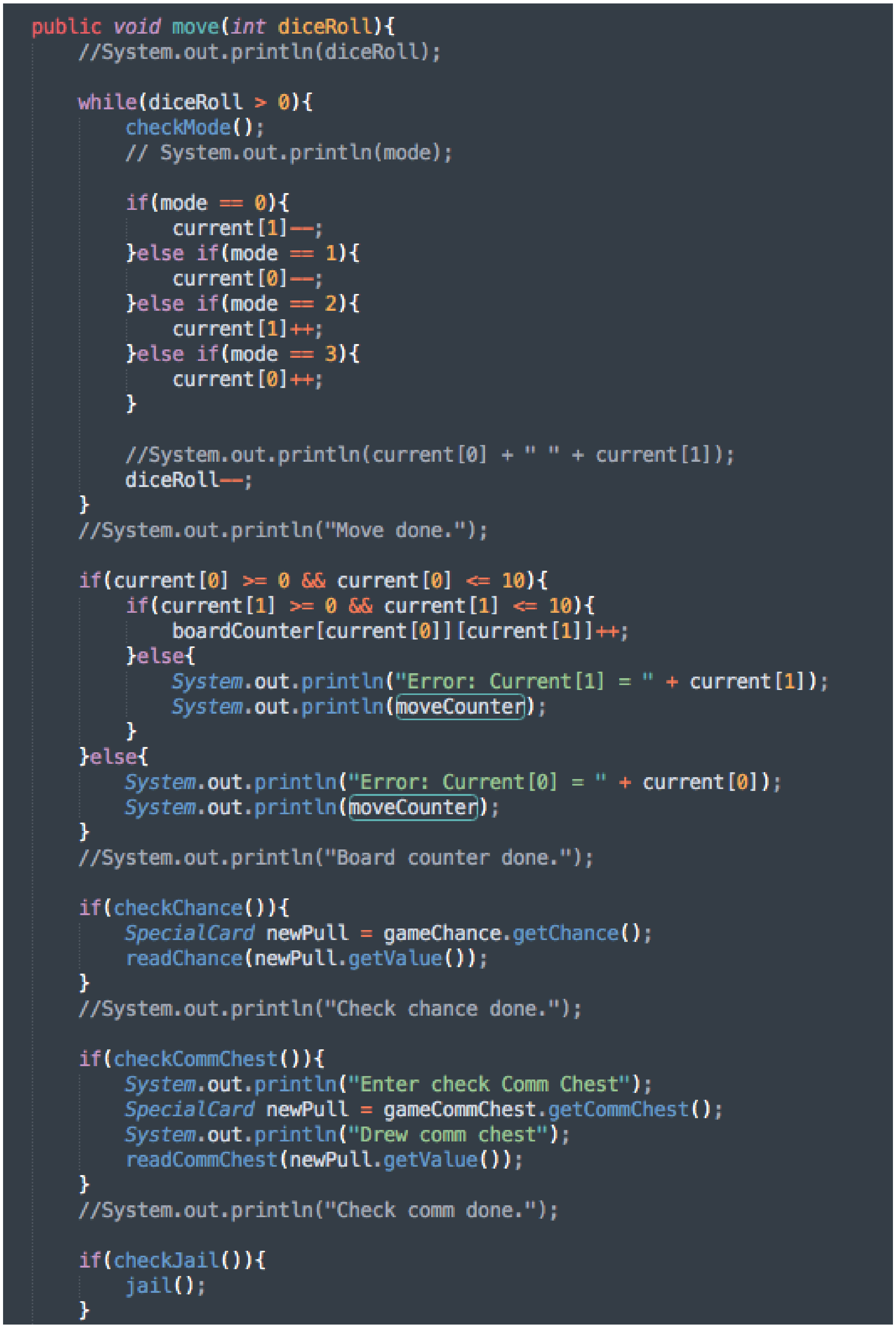


```java
public void readCommChest(int value){
    if(value == 0){
        // Get out of jail card
        System.out.println("Community Chest: Get out of jail free");
        jailCard = true;
    }else if(value == 1){
        // Go to jail
        System.out.println("Community Chest: Go directly to jail");
        current[0] = 0;
        current[1] = 0;
        jail();
        boardCounter[current[0]][current[1]]++;
    }else if(value == 2){
        // Proceed to go
        System.out.println("Community Chest: Proceed to go");
        current[0] = 10;
        current[1] = 10;
        boardCounter[current[0]][current[1]]++;
    }else if(value == 3){
        // Pay money
        System.out.println("Community Chest: You owe back taxes. Pay $200");
    }else if(value == 4){
        // Collect money
        System.out.println("Community Chest: You win the lottery. Collect $100");
    }
}
```

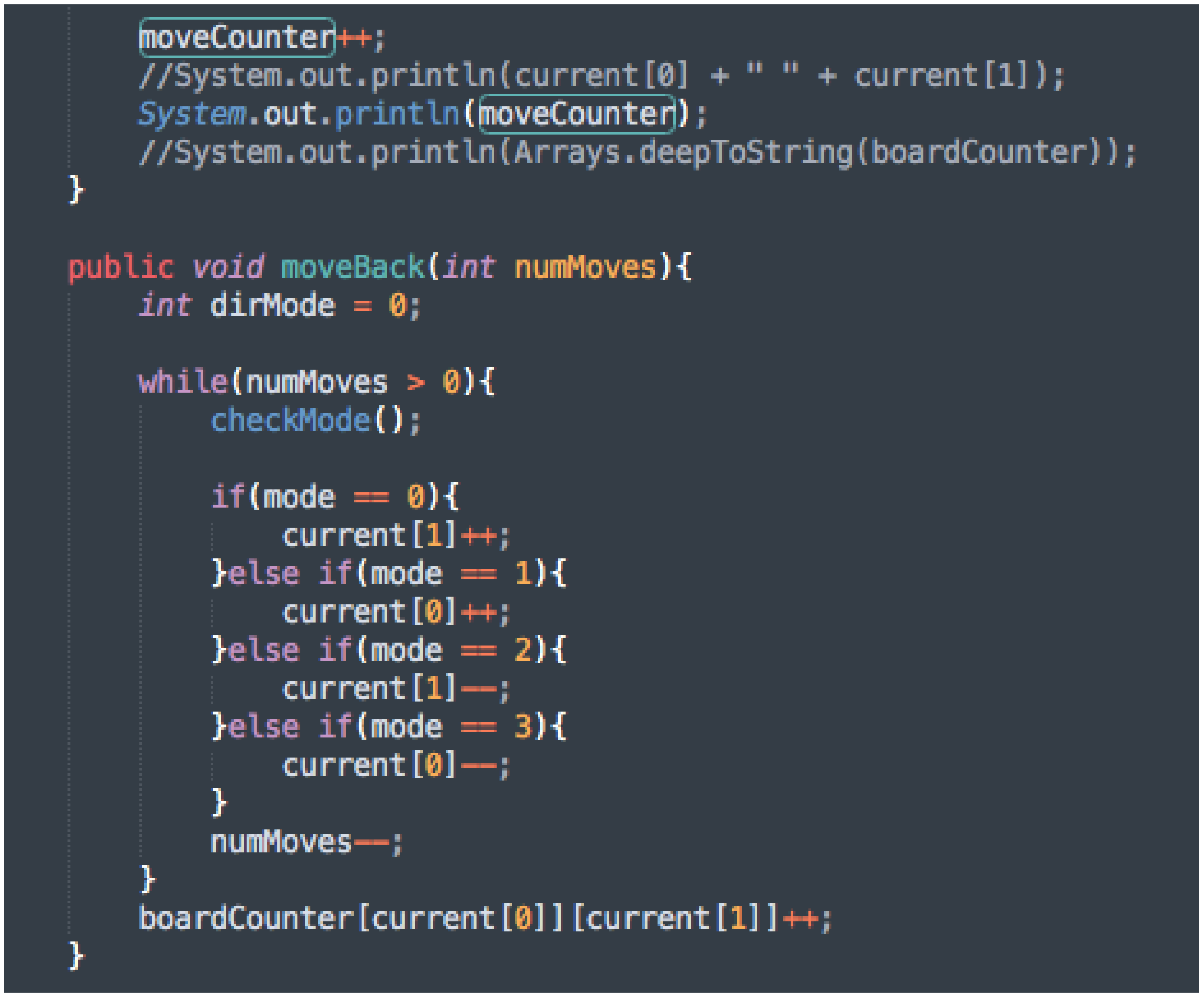
3. **Write code for simulating a game of Monopoly with a single player, go to jail, and chance cards. Record how many times you land on each square after playing a game consisting of 100 moves. Run the 2 simulations 1000 times and average the landing results. Show in a table the following probabilities: each railway station, the GO square, Mediterranean Avenue and Boardwalk. (**) (2pt)**



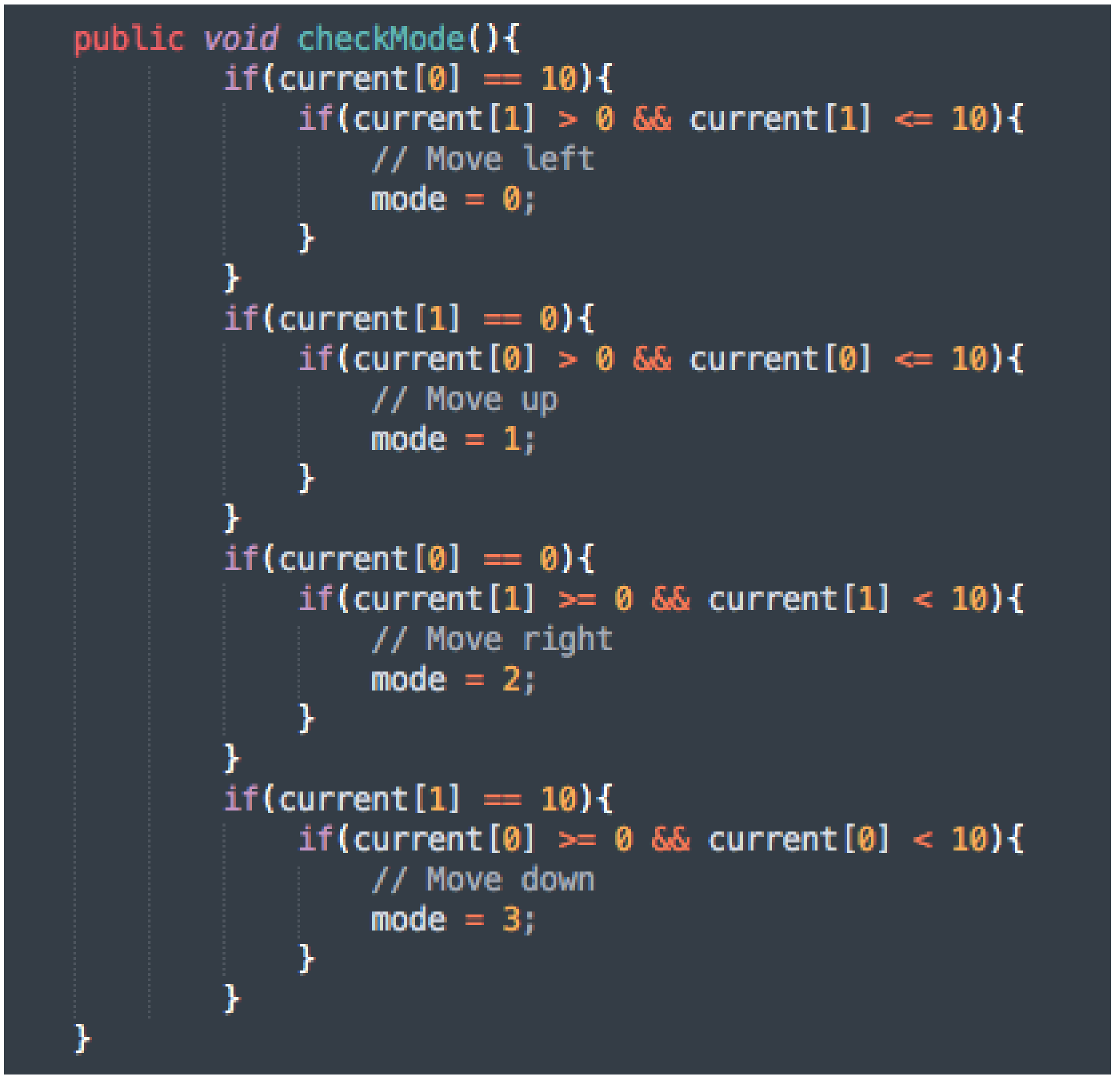
```
class GameBoard{
    /*
        Go = [10][10]
        Reading Railroad = [10][5]
        Pennsylvania Railroad = [5][0]
        B & O Railroad = [0][5]
        Short Line = [5][10]
        Mediterenean Avenue = [10][9]
        Boardwalk = [9][10]
        Chance = {[10][3], [0][2], [10][6]}
        Community Chest = {[10][8], [3][0], [10][3]}
    */

    private String[][] monopoly;
    // Have this in there but it's not used.
    private int[][] boardCounter;
    // Keeps track of the number of times we land on each spot
    private int moveCounter;
    // Keeps track of the number of turns we've used
    private int [] current;
    // Keeps track of our current position [0][1]
    private int mode;
    // Keeps track of how we travel on the board
    private Chance gameChance;
    // Creates a new instance of chance
    private boolean jailCard;
    // Keeps track of our get out of jail free cards
    private CommunityChest gameCommChest;
    // Creates a new instance of Community Chest
    // current[0] = row
    // current[1] = col

    public GameBoard(){
        monopoly = new String[11][11];
        boardCounter = new int[11][11];
        current = new int[2];
        current[0] = 10;
        current[1] = 10;
        moveCounter = 0;
        mode = 0;
        gameChance = new Chance();
        gameCommChest = new CommunityChest();
        jailCard = false;
    }
```



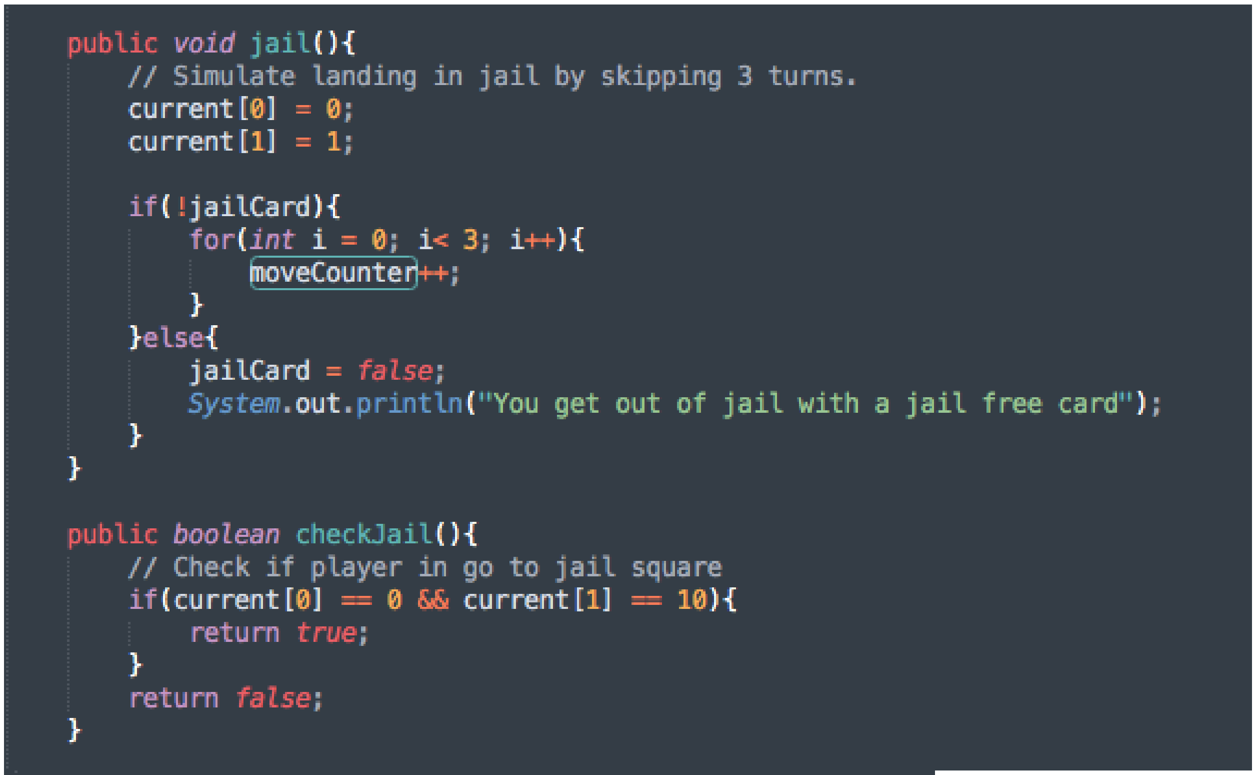
```java
public void move(int diceRoll){
    //System.out.println(diceRoll);

    while(diceRoll > 0){
        checkMode();
        // System.out.println(mode);

        if(mode == 0){
            current[1]--;
        }else if(mode == 1){
            current[0]--;
        }else if(mode == 2){
            current[1]++;
        }else if(mode == 3){
            current[0]++;
        }

        //System.out.println(current[0] + " " + current[1]);
        diceRoll--;
    }
    //System.out.println("Move done.");

    if(current[0] >= 0 && current[0] <= 10){
        if(current[1] >= 0 && current[1] <= 10){
            boardCounter[current[0]][current[1]]++;
        }else{
            System.out.println("Error: Current[1] = " + current[1]);
            System.out.println(moveCounter);
        }
    }else{
        System.out.println("Error: Current[0] = " + current[0]);
        System.out.println(moveCounter);
    }
    //System.out.println("Board counter done.");

    if(checkChance()){
        SpecialCard newPull = gameChance.getChance();
        readChance(newPull.getValue());
    }
    //System.out.println("Check chance done.");

    if(checkCommChest()){
        System.out.println("Enter check Comm Chest");
        SpecialCard newPull = gameCommChest.getCommChest();
        System.out.println("Drew comm chest");
        readCommChest(newPull.getValue());
    }
    //System.out.println("Check comm done.");

    if(checkJail()){
        jail();
    }
```



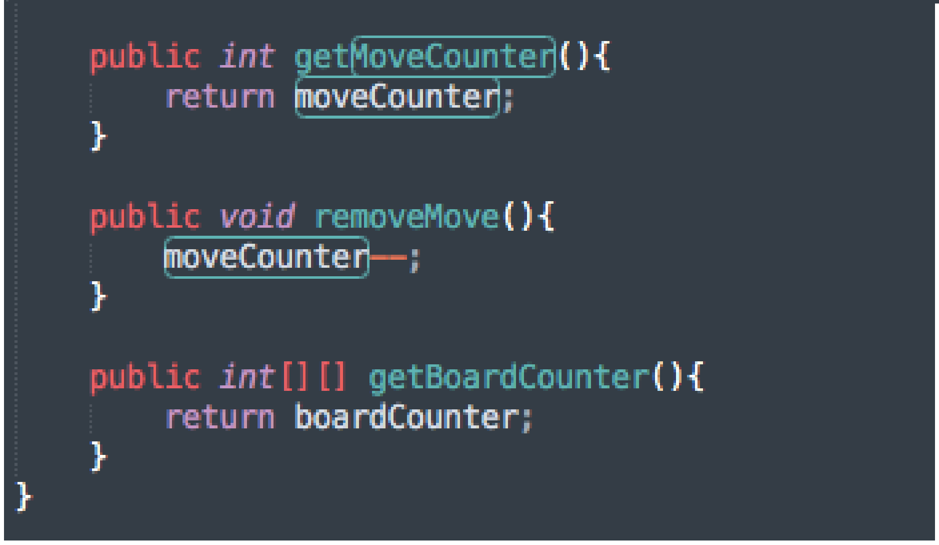
```
        moveCounter++;
        //System.out.println(current[0] + " " + current[1]);
        System.out.println(moveCounter);
        //System.out.println(Arrays.deepToString(boardCounter));
    }

    public void moveBack(int numMoves){
        int dirMode = 0;

        while(numMoves > 0){
            checkMode();

            if(mode == 0){
                current[1]++;
            }else if(mode == 1){
                current[0]++;
            }else if(mode == 2){
                current[1]--;
            }else if(mode == 3){
                current[0]--;
            }
            numMoves--;
        }
        boardCounter[current[0]][current[1]]++;
    }
```



```
public void checkMode(){
        if(current[0] == 10){
            if(current[1] > 0 && current[1] <= 10){
                // Move left
                mode = 0;
            }
        }
        if(current[1] == 0){
            if(current[0] > 0 && current[0] <= 10){
                // Move up
                mode = 1;
            }
        }
        if(current[0] == 0){
            if(current[1] >= 0 && current[1] < 10){
                // Move right
                mode = 2;
            }
        }
        if(current[1] == 10){
            if(current[0] >= 0 && current[0] < 10){
                // Move down
                mode = 3;
            }
        }
}
```



```
    public void jail(){
        // Simulate landing in jail by skipping 3 turns.
        current[0] = 0;
        current[1] = 1;

        if(!jailCard){
            for(int i = 0; i< 3; i++){
                moveCounter++;
            }
        }else{
            jailCard = false;
            System.out.println("You get out of jail with a jail free card");
        }
    }

    public boolean checkJail(){
        // Check if player in go to jail square
        if(current[0] == 0 && current[1] == 10){
            return true;
        }
        return false;
    }
```

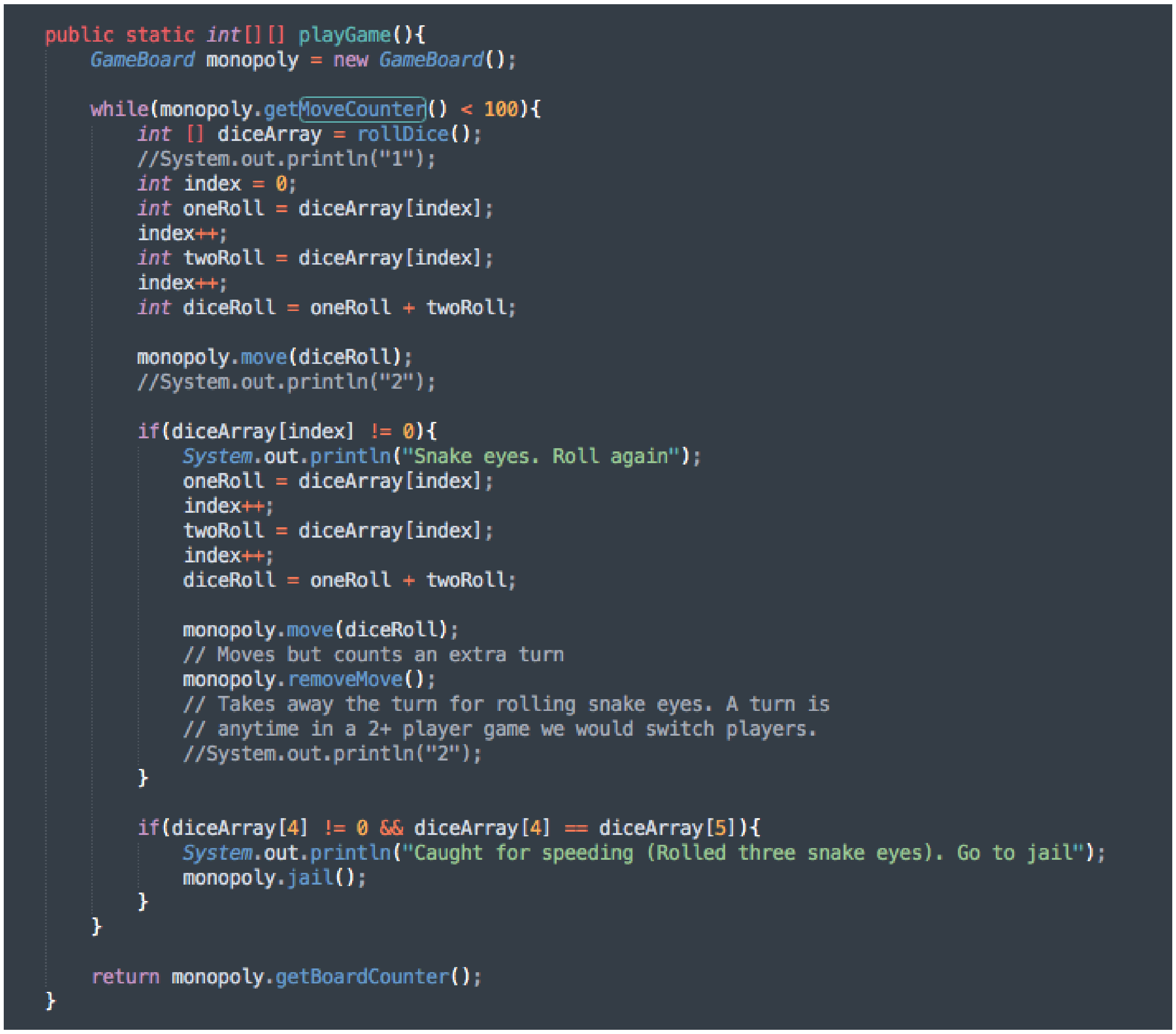


```
    public int getMoveCounter(){
        return moveCounter;
    }

    public void removeMove(){
        moveCounter--;
    }

    public int[][] getBoardCounter(){
        return boardCounter;
    }
}
```

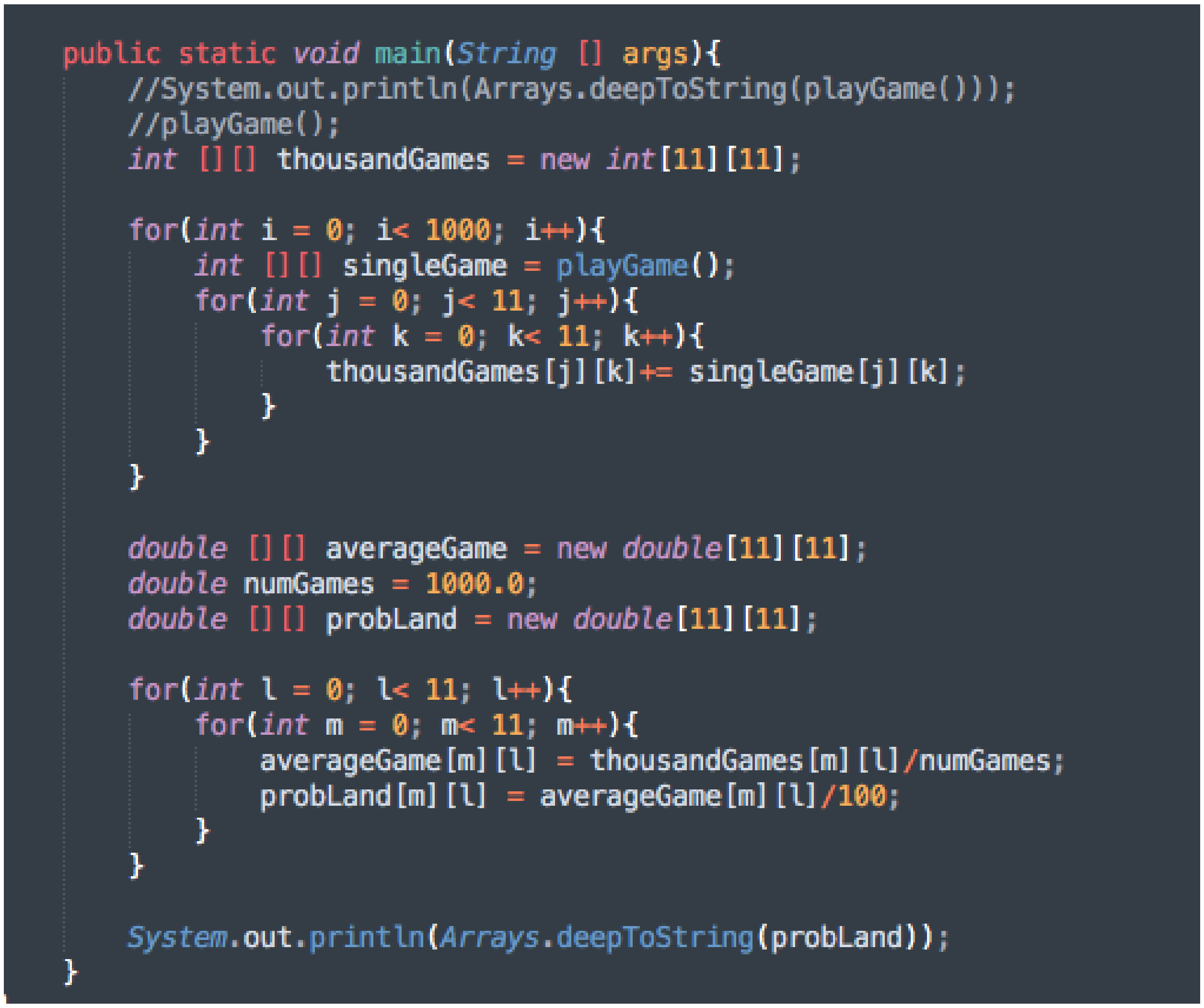
Above are the snippets of code for the last class of this project, GameBoard. The GameBoard class simulates an entire board by having an 11 x 11 integer array which keeps track of the number of times we land on the square. This class is also where Chance and Community Chest are called and evaluated.

This class also handles the moving along the game board. This is done by feeding into the class move, each roll of the dice from the previously described rollDice method. A while loop the same number of times as our dice roll. With every iteration of the while loop, we check for the mode. This mode tells our program which direction on the board we are traveling. The mode is set through the checkMode method.

Each time the while loop finishes, we have finished moving due to the dice roll. Now, the method checks if the spot we landed on is either, chance, community chest, or the go to jail square. If none of these are stepped on then we finish and are ready for the next turn. If they aren't then we handle each successive function. Chance and Community Chest have been described already. Jail is a new method. Jail simulates staying in jail by iterating the moveCounter three times. This simulates a play staying in jail for three moves.



```java
public static int[][] playGame(){
    GameBoard monopoly = new GameBoard();

    while(monopoly.getMoveCounter() < 100){
        int [] diceArray = rollDice();
        //System.out.println("1");
        int index = 0;
        int oneRoll = diceArray[index];
        index++;
        int twoRoll = diceArray[index];
        index++;
        int diceRoll = oneRoll + twoRoll;

        monopoly.move(diceRoll);
        //System.out.println("2");

        if(diceArray[index] != 0){
            System.out.println("Snake eyes. Roll again");
            oneRoll = diceArray[index];
            index++;
            twoRoll = diceArray[index];
            index++;
            diceRoll = oneRoll + twoRoll;

            monopoly.move(diceRoll);
            // Moves but counts an extra turn
            monopoly.removeMove();
            // Takes away the turn for rolling snake eyes. A turn is
            // anytime in a 2+ player game we would switch players.
            //System.out.println("2");
        }

        if(diceArray[4] != 0 && diceArray[4] == diceArray[5]){
            System.out.println("Caught for speeding (Rolled three snake eyes). Go to jail");
            monopoly.jail();
        }
    }

    return monopoly.getBoardCounter();
}
```

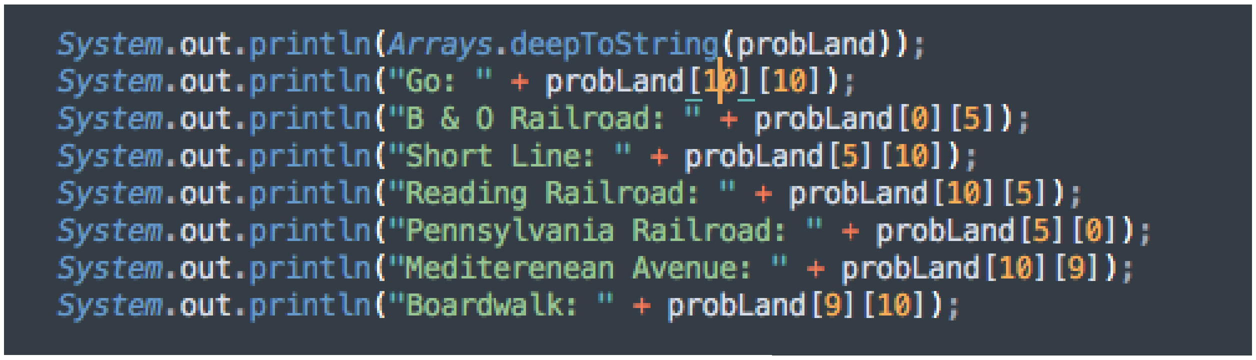


```java
public static void main(String [] args){
    //System.out.println(Arrays.deepToString(playGame()));
    //playGame();
    int [][] thousandGames = new int[11][11];

    for(int i = 0; i< 1000; i++){
        int [][] singleGame = playGame();
        for(int j = 0; j< 11; j++){
            for(int k = 0; k< 11; k++){
                thousandGames[j][k]+= singleGame[j][k];
            }
        }
    }

    double [][] averageGame = new double[11][11];
    double numGames = 1000.0;
    double [][] probLand = new double[11][11];

    for(int l = 0; l< 11; l++){
        for(int m = 0; m< 11; m++){
            averageGame[m][l] = thousandGames[m][l]/numGames;
            probLand[m][l] = averageGame[m][l]/100;
        }
    }

    System.out.println(Arrays.deepToString(probLand));
}
```

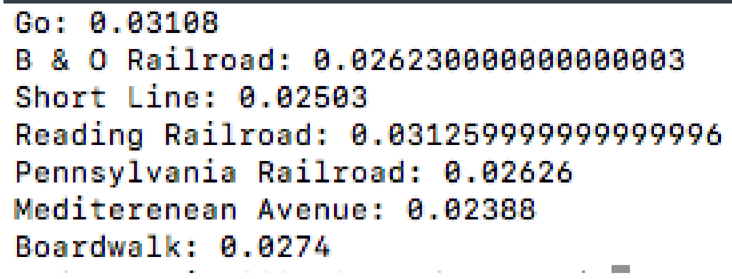
The playGame method handles the rolling of dice, creation of our GameBoard monopoly and the logic for speeding. This method simulates 100 turns of monopoly on our board. To check for speeding, the method takes the integer array returned from our rollDice method and checks if the third roll is also snake eyes. If it is, then we go to jail and lose three turns.

The final method is our main, which handles simulating 1000 games of 100 turns of monopoly. It keeps track of the number of times we land on each square using the thousandGames array. For each iteration of the monopoly game we run, we add to the thousand games array to get our total count. We can then take that thousandGame array and divide each square by 1000 to find the average times we land on each square. Taking that divided by 100 in our probLand array gives us the probability that the square will be landed on in a turn.

Go	0.031
B&O Railroad	0.026
Short Line	0.025
Reading Railroad	0.031
Pennsylvania Railroad	0.026
Mediterranean Avenue	0.024
Boardwalk	0.027



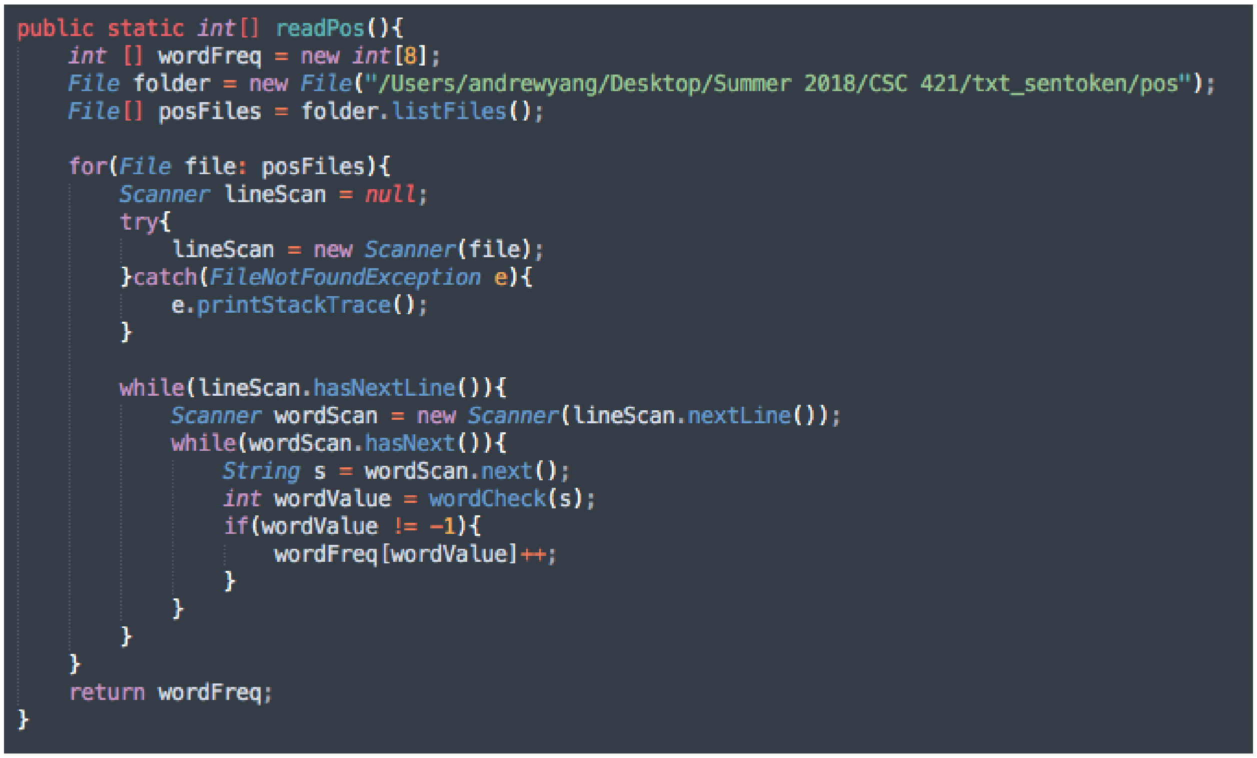
```
System.out.println(Arrays.deepToString(probLand));
System.out.println("Go: " + probLand[10][10]);
System.out.println("B & O Railroad: " + probLand[0][5]);
System.out.println("Short Line: " + probLand[5][10]);
System.out.println("Reading Railroad: " + probLand[10][5]);
System.out.println("Pennsylvania Railroad: " + probLand[5][0]);
System.out.println("Mediterenean Avenue: " + probLand[10][9]);
System.out.println("Boardwalk: " + probLand[9][10]);
```



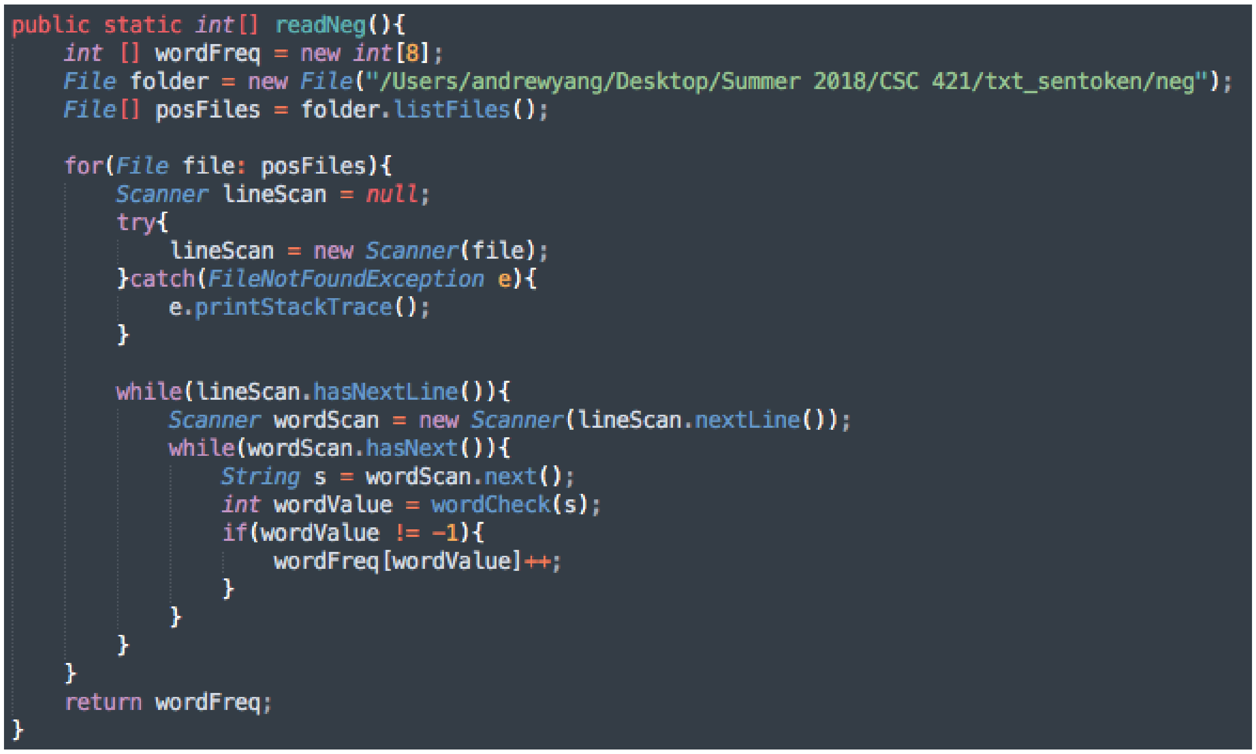
```
Go: 0.03108
B & O Railroad: 0.026230000000000003
Short Line: 0.02503
Reading Railroad: 0.031259999999999996
Pennsylvania Railroad: 0.02626
Mediterenean Avenue: 0.02388
Boardwalk: 0.0274
```

Section 2: Bayes Naïve Classification

1. **Write code that parses the text files and calculates the probabilities for each dictionary word given the review polarity (*) (1pt)**

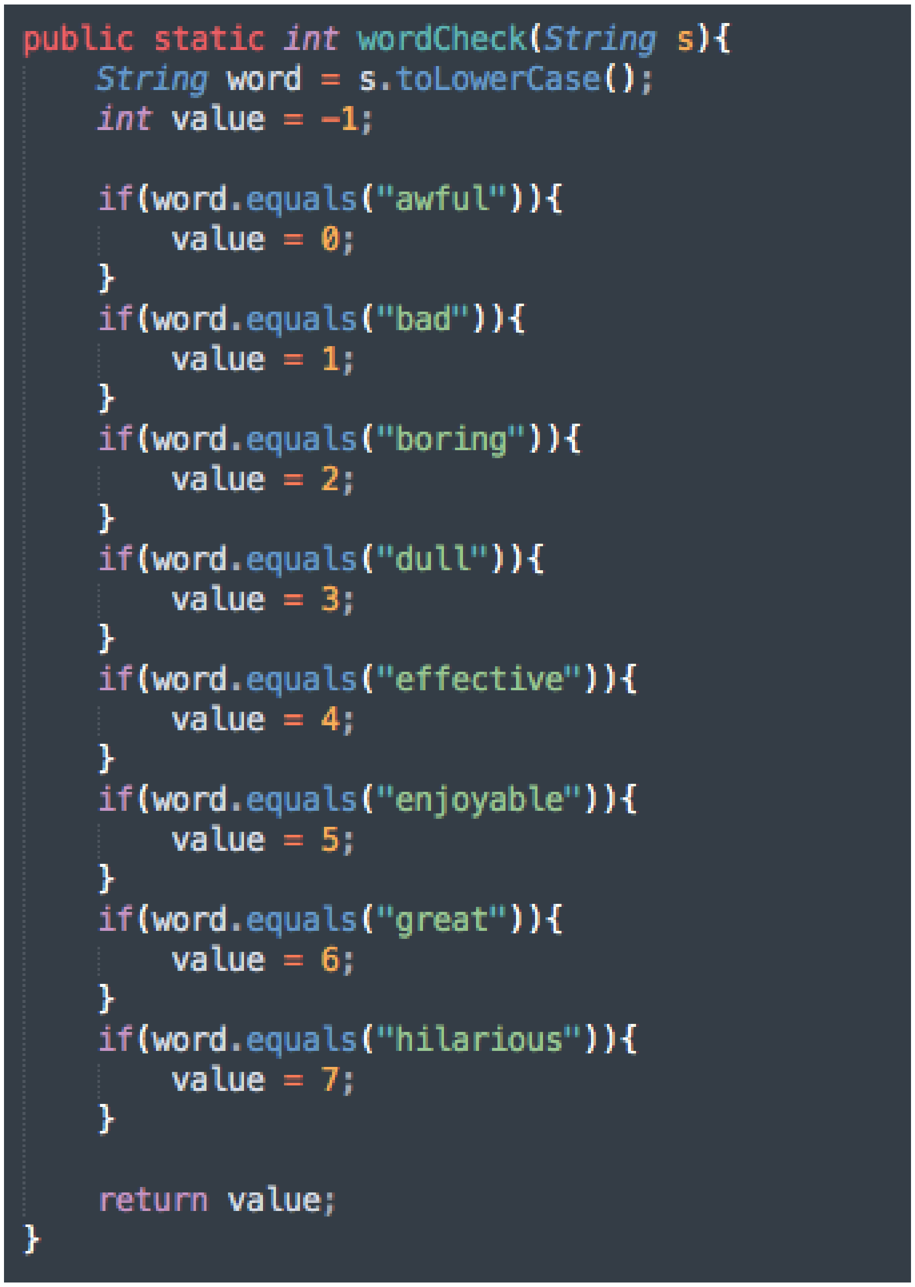


```java
public static int[] readPos(){
    int [] wordFreq = new int[8];
    File folder = new File("/Users/andrewyang/Desktop/Summer 2018/CSC 421/txt_sentoken/pos");
    File[] posFiles = folder.listFiles();

    for(File file: posFiles){
        Scanner lineScan = null;
        try{
            lineScan = new Scanner(file);
        }catch(FileNotFoundException e){
            e.printStackTrace();
        }

        while(lineScan.hasNextLine()){
            Scanner wordScan = new Scanner(lineScan.nextLine());
            while(wordScan.hasNext()){
                String s = wordScan.next();
                int wordValue = wordCheck(s);
                if(wordValue != -1){
                    wordFreq[wordValue]++;
                }
            }
        }
    }
    return wordFreq;
}
```



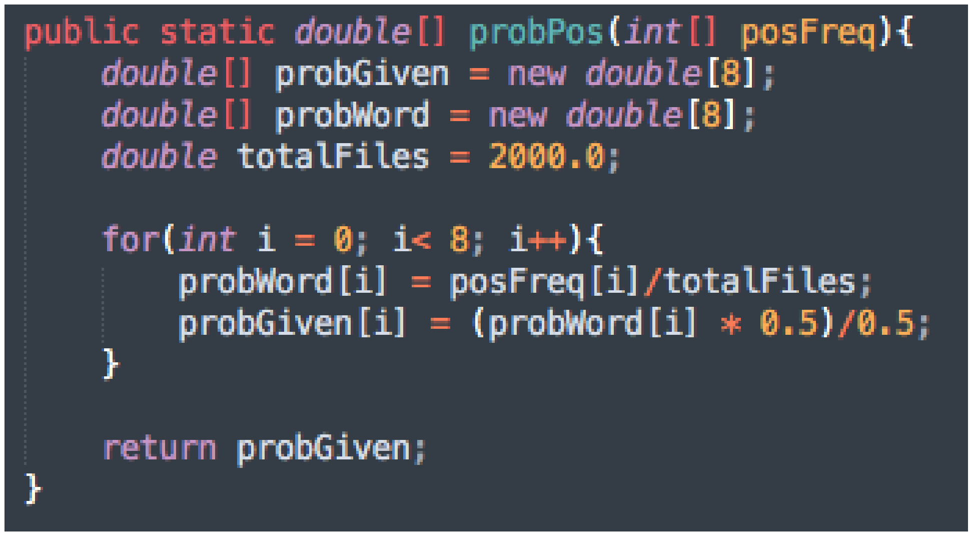
```java
public static int[] readNeg(){
    int [] wordFreq = new int[8];
    File folder = new File("/Users/andrewyang/Desktop/Summer 2018/CSC 421/txt_sentoken/neg");
    File[] posFiles = folder.listFiles();

    for(File file: posFiles){
        Scanner lineScan = null;
        try{
            lineScan = new Scanner(file);
        }catch(FileNotFoundException e){
            e.printStackTrace();
        }

        while(lineScan.hasNextLine()){
            Scanner wordScan = new Scanner(lineScan.nextLine());
            while(wordScan.hasNext()){
                String s = wordScan.next();
                int wordValue = wordCheck(s);
                if(wordValue != -1){
                    wordFreq[wordValue]++;
                }
            }
        }
    }
    return wordFreq;
}
```

The above two methods readPos() and readNeg() handles reading the files and outputs the frequency that we see the words we are concerned with. Running through the method readPos(), we open the folder and collect the list of contents. Then a for-loop is run for each file that reads each individual word with a scanner and adds a tick to each corresponding array index if we see any of the eight words that we are looking for.

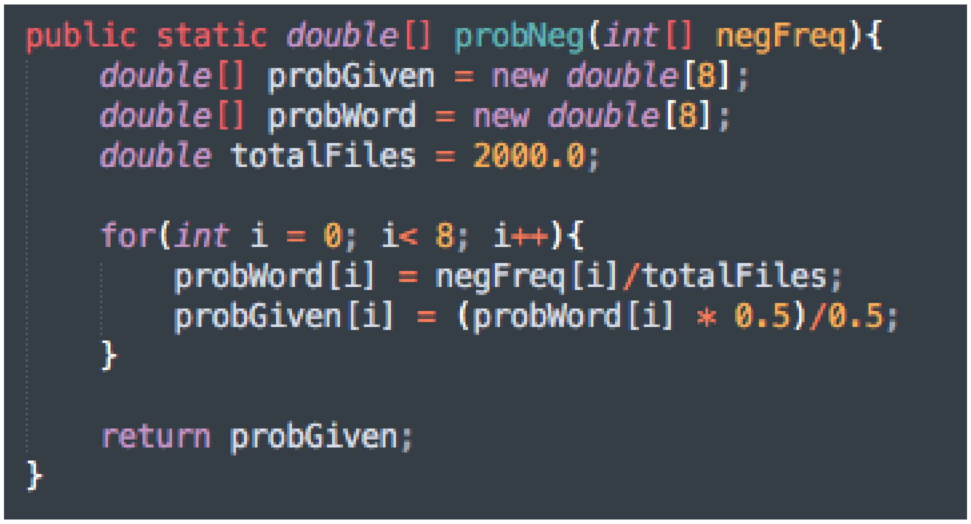


```java
public static int wordCheck(String s){
    String word = s.toLowerCase();
    int value = -1;

    if(word.equals("awful")){
        value = 0;
    }
    if(word.equals("bad")){
        value = 1;
    }
    if(word.equals("boring")){
        value = 2;
    }
    if(word.equals("dull")){
        value = 3;
    }
    if(word.equals("effective")){
        value = 4;
    }
    if(word.equals("enjoyable")){
        value = 5;
    }
    if(word.equals("great")){
        value = 6;
    }
    if(word.equals("hilarious")){
        value = 7;
    }

    return value;
}
```

To check for those words, every word read by the scanner is passed through the wordCheck() method which returns an integer value if the word is recognized. The integer values returned correspond to the index of the array position in readPos() and readNeg(). Using the integer value from this, the corresponding frequency arrays for each method can be tallied up.

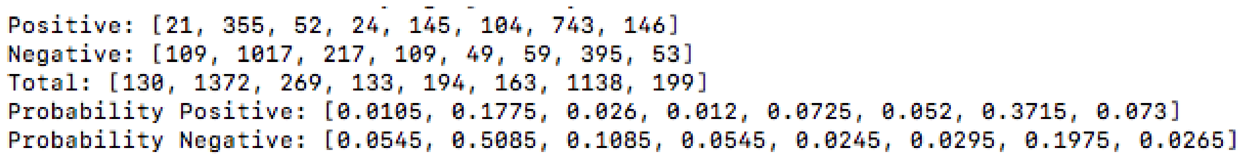


```java
public static double[] probPos(int[] posFreq){
    double[] probGiven = new double[8];
    double[] probWord = new double[8];
    double totalFiles = 2000.0;

    for(int i = 0; i< 8; i++){
        probWord[i] = posFreq[i]/totalFiles;
        probGiven[i] = (probWord[i] * 0.5)/0.5;
    }

    return probGiven;
}
```



```java
public static double[] probNeg(int[] negFreq){
    double[] probGiven = new double[8];
    double[] probWord = new double[8];
    double totalFiles = 2000.0;

    for(int i = 0; i< 8; i++){
        probWord[i] = negFreq[i]/totalFiles;
        probGiven[i] = (probWord[i] * 0.5)/0.5;
    }

    return probGiven;
}
```

The probability is calculated using the formula, P(word|Positive) = P(word ∧ Positive)/P(Positive).This is calculated for each word in the array and the resulting array is output.



```
Positive: [21, 355, 52, 24, 145, 104, 743, 146]
Negative: [109, 1017, 217, 109, 49, 59, 395, 53]
Total: [130, 1372, 269, 133, 194, 163, 1138, 199]
Probability Positive: [0.0105, 0.1775, 0.026, 0.012, 0.0725, 0.052, 0.3715, 0.073]
Probability Negative: [0.0545, 0.5085, 0.1085, 0.0545, 0.0245, 0.0295, 0.1975, 0.0265]
```

2. **Explain how these probability estimates can be combined to form a Naive Bayes classifier. You can look up Bernoulli Bayes model for this simple model where only presence/absence of a word is modeled. (*) (1pt)**

   To use these probability estimates to form a Naïve Bayes classifier we first determine which set of words would likely exist in each review polarity.

Our set of words are w = {Awful, Bad, Boring, Dull, Effective, Enjoyable, Great, Hilarious}. The prior probabilities from the training data are P(pos) = 0.5 and P(neg) = 0.5.

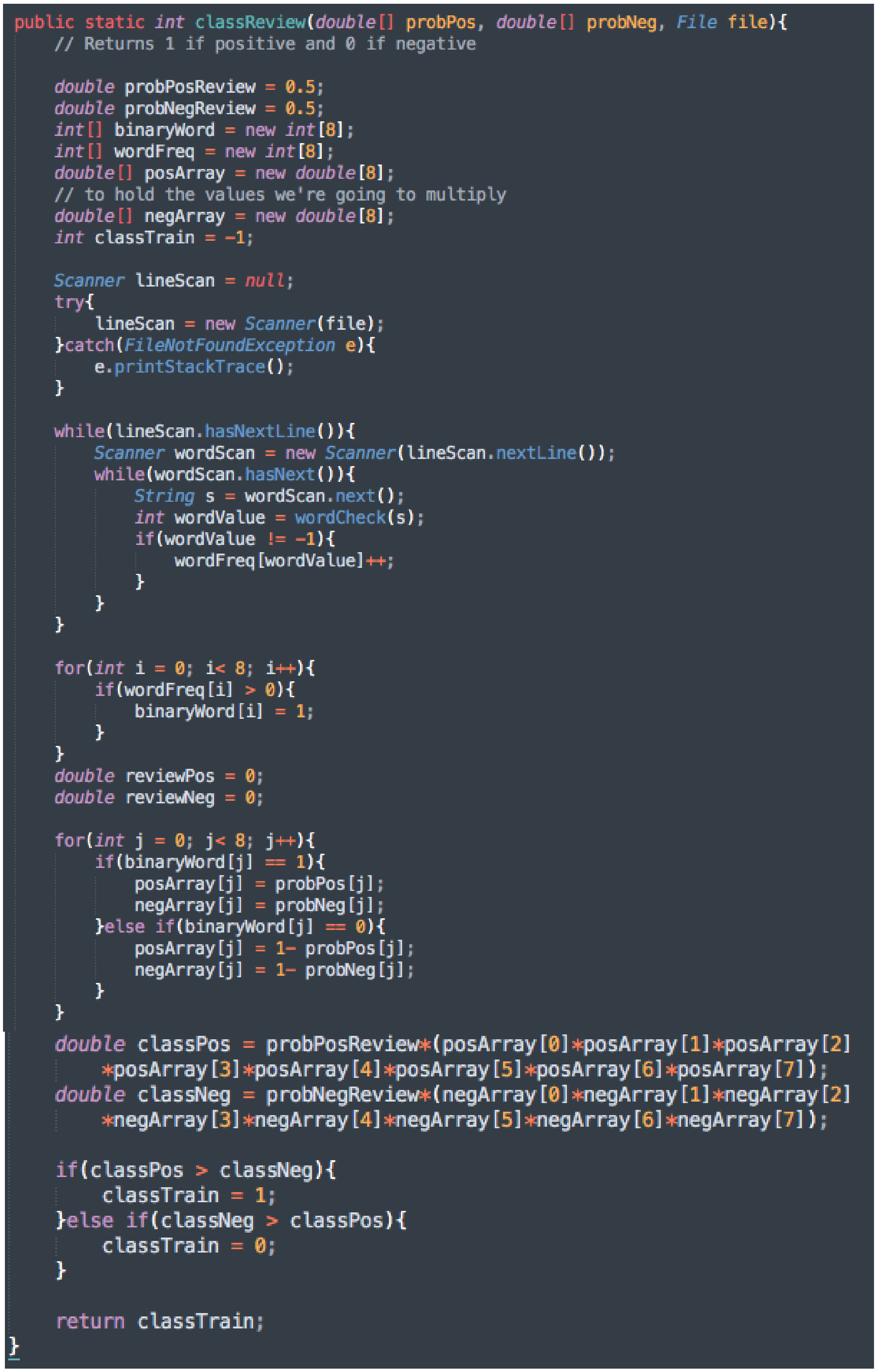
Using the probability estimates from question 1, we can calculate the probability of a review is either positive or negative. With a Bernoulli Bayes model, with a document that we're trying to test, we would want a vector with just binary values for each word. A 1 would mean that word exists in the file and a 0 means that it doesn't. For each polarity, a 1 means we use the probability of that word given polarity, and a 0 means we use the complement of the probability of that word given polarity. We would then sum them up and multiply by P(pos) or P(neg) respectively. The resulting value with the highest probability is what we would classify that review as.

3. **Calculate the classification accuracy and confusion matrix that you would obtain using the whole data set for both training and testing. (**) (1pt)**

	Trained: Positive	Trained: Negative
Actual: Positive	731	269
Actual: Negative	386	614

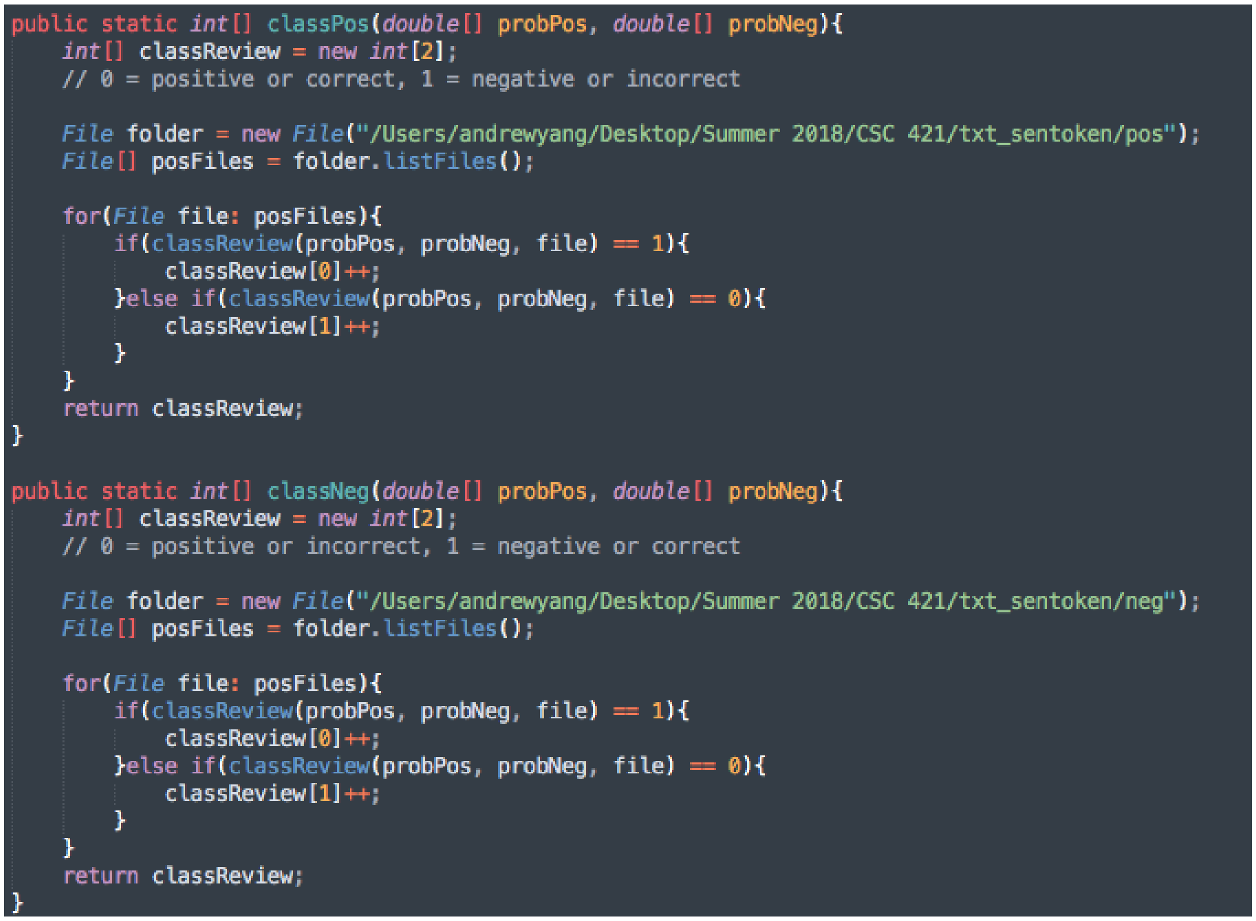
Training was done using a Bernoulli Bayes model, as laid out in question 2.

This results in a classification accuracy of: (731 + 614) / (2000) = 67.25%.



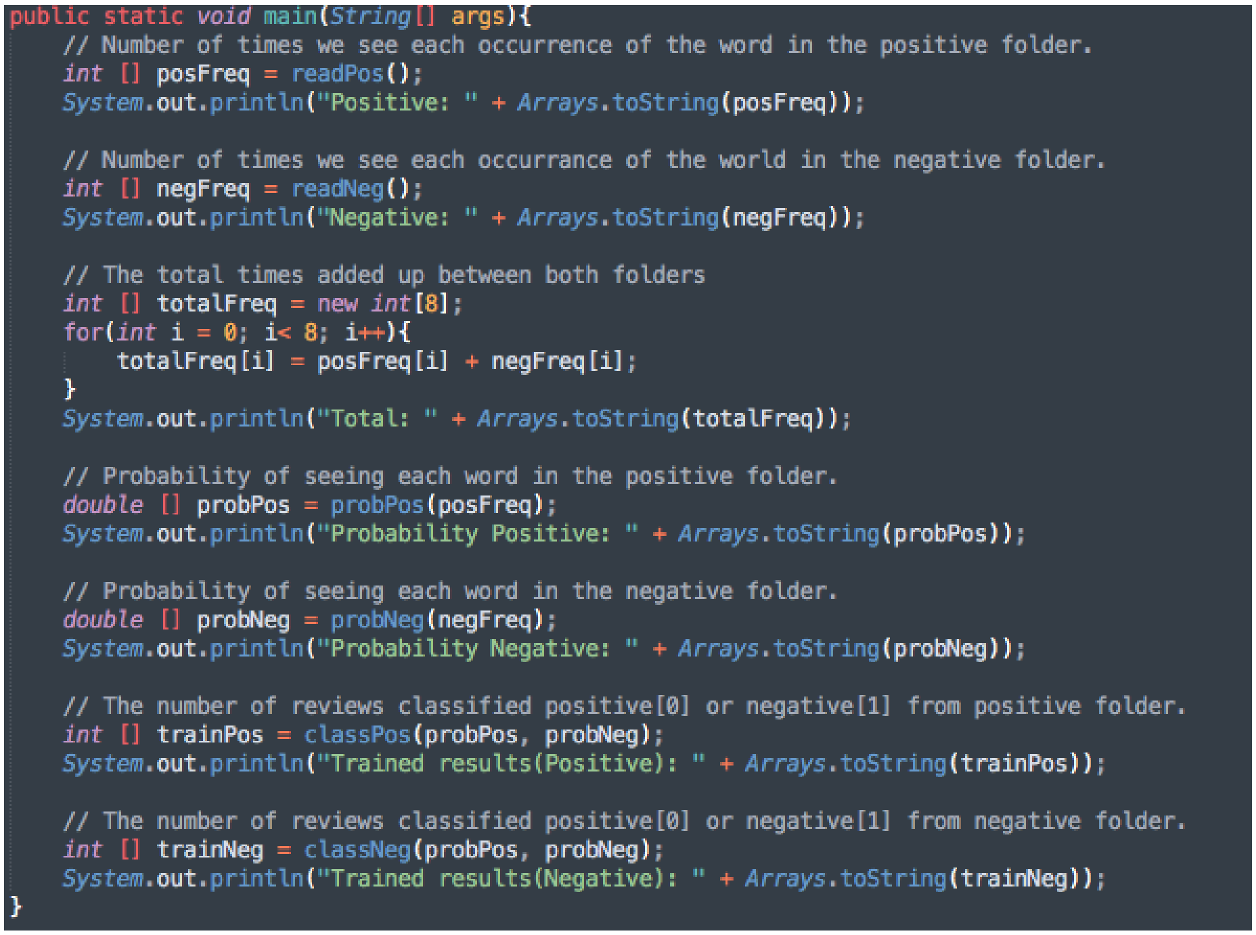
```java
public static int classReview(double[] probPos, double[] probNeg, File file){
    // Returns 1 if positive and 0 if negative

    double probPosReview = 0.5;
    double probNegReview = 0.5;
    int[] binaryWord = new int[8];
    int[] wordFreq = new int[8];
    double[] posArray = new double[8];
    // to hold the values we're going to multiply
    double[] negArray = new double[8];
    int classTrain = -1;

    Scanner lineScan = null;
    try{
        lineScan = new Scanner(file);
    }catch(FileNotFoundException e){
        e.printStackTrace();
    }

    while(lineScan.hasNextLine()){
        Scanner wordScan = new Scanner(lineScan.nextLine());
        while(wordScan.hasNext()){
            String s = wordScan.next();
            int wordValue = wordCheck(s);
            if(wordValue != -1){
                wordFreq[wordValue]++;
            }
        }
    }

    for(int i = 0; i< 8; i++){
        if(wordFreq[i] > 0){
            binaryWord[i] = 1;
        }
    }
    double reviewPos = 0;
    double reviewNeg = 0;

    for(int j = 0; j< 8; j++){
        if(binaryWord[j] == 1){
            posArray[j] = probPos[j];
            negArray[j] = probNeg[j];
        }else if(binaryWord[j] == 0){
            posArray[j] = 1- probPos[j];
            negArray[j] = 1- probNeg[j];
        }
    }
    double classPos = probPosReview*(posArray[0]*posArray[1]*posArray[2]
        *posArray[3]*posArray[4]*posArray[5]*posArray[6]*posArray[7]);
    double classNeg = probNegReview*(negArray[0]*negArray[1]*negArray[2]
        *negArray[3]*negArray[4]*negArray[5]*negArray[6]*negArray[7]);

    if(classPos > classNeg){
        classTrain = 1;
    }else if(classNeg > classPos){
        classTrain = 0;
    }

    return classTrain;
}
```

classReview() uses the Bernoulli Bayes method to classify each file as positive or negative. The probabilities calculated in question 1 were used to do the testing. The Bernoulli Bayes method is laid out in question 2. In this method, if the probability of positive was higher it returns 1, and 0 for negative. This method handles just one file at a time and returns a value for that file.



```java
public static int[] classPos(double[] probPos, double[] probNeg){
    int[] classReview = new int[2];
    // 0 = positive or correct, 1 = negative or incorrect

    File folder = new File("/Users/andrewyang/Desktop/Summer 2018/CSC 421/txt_sentoken/pos");
    File[] posFiles = folder.listFiles();

    for(File file: posFiles){
        if(classReview(probPos, probNeg, file) == 1){
            classReview[0]++;
        }else if(classReview(probPos, probNeg, file) == 0){
            classReview[1]++;
        }
    }
    return classReview;
}

public static int[] classNeg(double[] probPos, double[] probNeg){
    int[] classReview = new int[2];
    // 0 = positive or incorrect, 1 = negative or correct

    File folder = new File("/Users/andrewyang/Desktop/Summer 2018/CSC 421/txt_sentoken/neg");
    File[] posFiles = folder.listFiles();

    for(File file: posFiles){
        if(classReview(probPos, probNeg, file) == 1){
            classReview[0]++;
        }else if(classReview(probPos, probNeg, file) == 0){
            classReview[1]++;
        }
    }
    return classReview;
}
```

classPos() and classNeg() were used to iterate through all the files in their corresponding folders and fed each file into classReview(). The values returned from classReview() were then tallied up into integer arrays and returned.



```java
public static void main(String[] args){
    // Number of times we see each occurrence of the word in the positive folder.
    int [] posFreq = readPos();
    System.out.println("Positive: " + Arrays.toString(posFreq));

    // Number of times we see each occurrance of the world in the negative folder.
    int [] negFreq = readNeg();
    System.out.println("Negative: " + Arrays.toString(negFreq));

    // The total times added up between both folders
    int [] totalFreq = new int[8];
    for(int i = 0; i< 8; i++){
        totalFreq[i] = posFreq[i] + negFreq[i];
    }
    System.out.println("Total: " + Arrays.toString(totalFreq));

    // Probability of seeing each word in the positive folder.
    double [] probPos = probPos(posFreq);
    System.out.println("Probability Positive: " + Arrays.toString(probPos));

    // Probability of seeing each word in the negative folder.
    double [] probNeg = probNeg(negFreq);
    System.out.println("Probability Negative: " + Arrays.toString(probNeg));

    // The number of reviews classified positive[0] or negative[1] from positive folder.
    int [] trainPos = classPos(probPos, probNeg);
    System.out.println("Trained results(Positive): " + Arrays.toString(trainPos));

    // The number of reviews classified positive[0] or negative[1] from negative folder.
    int [] trainNeg = classNeg(probPos, probNeg);
    System.out.println("Trained results(Negative): " + Arrays.toString(trainNeg));
}
```