



Search or jump to...

Pull requests Issues Marketplace Explore



pratiksh665 / tp\_2\_skunk

[Watch](#) 0 [Star](#) 0 [Fork](#) 0[Code](#)[Issues 1](#)[Pull requests 0](#)[Projects 0](#)[Wiki](#) [Security](#)[Insights](#)[Settings](#)**CODE SMELL: Got rid of Turn.java, unnecessary class, better split those**

methods and variable into Dice.java and Player.java

[Browse files](#)

by anna\_junit (#2) + anna\_refactoring (#2)

Macks2514 committed 14 days ago

1 parent a08f3e4 commit b6a97f535ebb2e6e614fc2d57db788016c720db3

Showing 13 changed files with 624 additions and 770 deletions.

[Unified](#) [Split](#)

5 SkunkProject/bin/.gitignore

...

```
... ... @@ -1,5 +1,6 @@
1   1   /AppRunner.class
2   2   /AppRunnerTest.class
3 + 3   + /Controller.class
4   4   /Dice.class
5   5   /DiceTest.class
6   6   /Die.class
7 @@ -9,7 +10,3 @@
9  10  /Skunkapp.class
10 11  /Turn.class
11 12  /TurnTest.class
12 - 13  - /TestSuite.class
13 - 14  - /TestSuiteRunner.class
14 - 15  - /TestsuiteTestSuiteExample.class
15 -  - /Controller.class
```

119 SkunkProject/src/AppRunner.java

...

```
... ... @@ -1,71 +1,48 @@
1   1   - import java.io.BufferedReader;
2   2   - import java.io.FileNotFoundException;
3   3   - import java.io.FileReader;
4   4   - import java.io.IOException;
5   5   - import java.util.ArrayList;
6   6   - import java.util.Scanner;
7   -
8   - import edu.princeton.cs.introcs.*;
9   -
10  - //-----TP 1.2 changes-----
11  - public class AppRunner {
12  -
13  -     public int numPlayers;
14  -
15  -
16  -     public void displayGame() {
17  -         Controller controller = new Controller();
18  -         StdOut.println("Welcome to 635 Skunk project");
19  -         StdOut.println("Creating the dice object");
20  -         Scanner scan = new Scanner(System.in);
21  -         StdOut.println("Would you like to see the rules? (Yes/No)");
22  -         if (scan.next().equalsIgnoreCase("Yes")) {
23  -             controller.showRules();
24  -         }
25  -         StdOut.println("Number of players: ");
26  -         numPlayers = scan.nextInt();
27  -
28  -         for (int i = 1; i <= numPlayers; i++) {
29  -             StdOut.println("Player " + i + " Name: ");
30  -             controller.createPlayer(scan.next());
31  -         }
32  -
33  -         // Player Turn (For loop was used because we know that a skunk game has 5 turns
34  -         // )
35  -
36  -         // while (!controller.isHundred) {
37  -
38  -             for (Player player : controller.playerList) {
39  -                 StdOut.println(player.name + " start your turn? (Yes/No)"); // user prompt
40  -                 String decision = scan.next(); // storing user input
41  -                 if (decision.equalsIgnoreCase("No")) { // execute if user input = no
42  -                     StdOut.println(player.name + " skipped turn");
43  -                 }
44  -
45  -                 while (decision.equalsIgnoreCase("Yes")) { // execute if user input = yes
46  -                     controller.playerTurn();
47  -                     StdOut.println(player.name + " would you like to roll? (Yes/No)");
48  -                     String rollTurn = scan.next();
49  -                     if (rollTurn.equalsIgnoreCase("Yes")) {
50  -                         StdOut.println(controller.playerTurnContinue(player));
51  -
52  -                     }
53  -
54  -                     else if (rollTurn.equalsIgnoreCase("No")) {
55  -                         controller.playerTurnEnd(player);
56  -                         break;
57  -                     }
58  -                 } // end of while loop
59  -
60  -             } // end of for loop
61  -
62  -             controller.showRules();
```

```

63      -
64      -
65      -         } // end of while loop
66      -
67      -     } // end of playgame method
68      -
69      -     // Updated showRules() method to display info from a text file called SkunkRules for code reusability
70      -
71  } // end of class
1 + import java.io.BufferedReader;
2 + import java.io.FileNotFoundException;
3 + import java.io.FileReader;
4 + import java.io.IOException;
5 + import java.util.ArrayList;
6 + import java.util.Scanner;
7 +
8 + import edu.princeton.cs.introcs.*;
9 +
10 + //-----TP 1.2 changes-----
11 + public class AppRunner {
12 +     boolean turnInProg;
13 +     boolean roundInProg;
14 +     Player currentPlayer;
15 +
16 +     public void displayGame() {
17 +         Controller c = new Controller();
18 +         StdOut.println("Welcome to 635 skunk project");
19 +         StdOut.println("Creating the dice object");
20 +         Scanner scan = new Scanner(System.in);
21 +         StdOut.println("Would you like to see the rules? (Yes/No)");
22 +         if (scan.next().equalsIgnoreCase("Yes")) {
23 +             c.showRules();
24 +         }
25 +         StdOut.println("Number of players: ");
26 +         int numPlayers = scan.nextInt();
27 +         c.createPlayer(numPlayers);
28 +         roundInProg = true;
29 +
30 +         while(roundInProg) {
31 +             currentPlayer = c.currentPlayer;
32 +             StdOut.println(currentPlayer.name + " start your turn? (Yes/No)");
33 +             String decision = scan.next();
34 +             c.playerTurn(decision);
35 +
36 +             while (c.turnInProg) {
37 +                 StdOut.println(c.currentPlayer.name + " would you like to roll? (Yes/No)");
38 +                 String rollTurn = scan.next();
39 +                 if (rollTurn.equalsIgnoreCase("yes")) {
40 +                     StdOut.println(c.playerTurnContinue(currentPlayer));
41 +                 }
42 +                 else if (rollTurn.equalsIgnoreCase("no")) {
43 +                     StdOut.println(c.playerTurnEnd(currentPlayer));
44 +                 }
45 +             }
46 +         }
47 +     }
48 + } // end of class

```

```

@@ -1,128 +1,167 @@
...
...
...
-
1 - import java.io.BufferedReader;
2 - import java.io.FileNotFoundException;
3 - import java.io.FileReader;
4 - import java.io.IOException;
5 - import java.util.ArrayList;
6 -
7 - import edu.princeton.cs.introcs.StdOut;
8 -
9 - public class Controller {
10 -
11 -     public ArrayList<Player> playerList = new ArrayList();
12 -     public Turn turn;
13 -
14 -
16 -     public Player createPlayer(String name) {
17 -         Player player = new Player(name);
18 -         playerList.add(player);
19 -         return player;
20 -     }
21 -
22 -     public void playerTurn() {
23 -         turn = new Turn();
24 -     }
25 -
26 -     public String playerTurnContinue(Player player) {
27 -         turn.turnARoll();
28 -         return rollInfo(player);
29 -     }
30 -
31 -     public String rollInfo(Player player) {
32 -         String message = "Roll Info: "
33 -             + "\nDie 1: " + getDie1Value() + ";" + " Die 2: " + getDie2Value()
34 -             + "\nRoll total: " + getDiceValue()
35 -             + "\nTurn total: " + getTurnScore()
36 -             + "\nGame total: " + player.getScore()
37 -             + "\n" + checkSkunk(player)
38 -             + "\n" + hundredCheck(player);
39 -
40 -         return message;
41 -     }

```

```

42     -         public String playerTurnEnd(Player player) {
43     -             player.setScore(getTurnScore());
44     -             String message = player.name + "is ending turn \nTurn score was " + getTurnScore() + "\nTotal score is " + player
45     -             return message;
46     -
47     -     }
48     -
49     -     public String checkSkunk(Player player) {
50     -         String message = "";
51     -         if (turn.isSkunk() || turn.isDoubleSkunk()) {
52     -             if (turn.isSkunk()) {
53     -                 message = "You rolled a Skunk; end of turn \nTotal score is " + player.getScore();
54     -                 player.chipPenalty(1);
55     -             }
56     -             else if (turn.isDoubleSkunk()) {
57     -                 player.setScore(0);
58     -                 message = "You rolled a Double Skunk; end of turn \nTotal score is " + player.getScore();
59     -                 player.chipPenalty(2);
60     -             }
61     -         }
62     -         return message;
63     -     }
64     -
65     -     public String hundredCheck(Player player) {
66     -         String message = "";
67     -         if (player.getScore() >= 100) {
68     -             message = roundEnd(player);
69     -         }
70     -         return message;
71     -
72     -     }
73     -
74     -     public String roundEnd(Player player) {
75     -         String message = player.name + " won the game with " + player.playerScore + " points!";
76     -         return message;
77     -     }
78     -
79     -     public void showRules() {
80     -
81     -         // The name of the file to open.
82     -         String fileName = "SkunkRules";
83     -
84     -         // This will reference one line at a time
85     -         String line = null;
86     -
87     -         try {
88     -             // FileReader reads text files in the default encoding.
89     -             FileReader fileReader = new FileReader(fileName);
90     -
91     -             // Always wrap FileReader in BufferedReader.
92     -             BufferedReader bufferedReader = new BufferedReader(fileReader);
93     -
94     -             //read line by line in the text file until the line is not null
95     -             while((line = bufferedReader.readLine()) != null) {
96     -                 System.out.println(line);
97     -             }
98     -
99     -             // Always close files.
100            bufferedReader.close();
101        }
102        catch(FileNotFoundException ex) {
103            System.out.println("Unable to open file '" + fileName + "'");
104        }
105        catch(IOException ex) {
106            System.out.println("Error reading file '" + fileName + "'");
107        }
108    }
109    -
110    public int getDie1Value() {
111        return turn.getDie1Value();
112    }
113    -
114    public int getDie2Value() {
115        return turn.getDie2Value();
116    }
117    -
118    public int getDiceValue() {
119        return turn.getDiceValue();
120    }
121    -
122    public int getTurnScore() {
123        return turn.getTurnScore();
124    }
125    -
126    -
127    -
128    }
1+ import java.io.BufferedReader;
2+ import java.io.FileNotFoundException;
3+ import java.io.FileReader;
4+ import java.io.IOException;
5+ import java.util.ArrayList;
6+ import java.util.Scanner;
7+
8+ import edu.princeton.cs.introcs.StdOut;
9+
10+ public class Controller {
11+
12+     public ArrayList<Player> playerList = new ArrayList();
13+     public Dice dice;
14+     public int kitty;
15+     public int roundGoal;

```

```

16 +     public void addPlayer(Player player) {
17 +         playerList.add(player);
18 +     }
19 +
20 +     public void createPlayer(int numPlayers) {
21 +         for (int j = 1; j <= numPlayers; j++) {
22 +             StdOut.println("Player " + j + " Name: ");
23 +             Scanner scan = new Scanner(System.in);
24 +             Player player = new Player(scan.next());
25 +             playerList.add(player);
26 +         }
27 +         currentPlayerIndex = 0;
28 +         currentPlayer = playerList.get(currentPlayerIndex);
29 +
30 +         StdOut.println("First player: " + currentPlayer.name);
31 +     }
32 +
33 +     public void nextPlayer() {
34 +         if (currentPlayerIndex == (playerList.size() - 1)) {
35 +             currentPlayerIndex = 0;
36 +             currentPlayer = playerList.get(currentPlayerIndex);
37 +         } else {
38 +             currentPlayerIndex++;
39 +             currentPlayer = playerList.get(currentPlayerIndex);
40 +         }
41 +     }
42 +
43 +     //Questioning if player wants to begin turn or skip
44 +     public String playerTurn(String decision) {
45 +         String message = null;
46 +         if (decision.equalsIgnoreCase("No")) { // execute if user input = no
47 +             turnInProg = false;
48 +             message = currentPlayer.name + " skipped turn ";
49 +             nextPlayer();
50 +         }
51 +         else if (decision.equalsIgnoreCase("Yes")) {
52 +             turnInProg = true;
53 +             dice = new Dice();
54 +             message = currentPlayer.name + "'s turn";
55 +         }
56 +
57 +         return message;
58 +     }
59 +
60 +     //Method for player continuing their turn
61 +     public String playerTurnContinue(Player player) {
62 +         dice.roll(player);
63 +         return rollInfo(player);
64 +     }
65 +
66 +     public String rollInfo(Player player) {
67 +         String message = "Roll Info: "
68 +             + "\nDie 1: " + getDie1Value() + ";" + " Die 2: " + getDie2Value()
69 +             + "\nRoll total: " + getDiceValue()
70 +             + "\nTurn total: " + player.getTurnScore()
71 +             + "\nGame total: " + player.getScore()
72 +             + "\nKitty total: " + player.getChip()
73 +             + "\n" + checkSkunk(player)
74 +             + "\n" + checkHundred(player);
75 +
76 +         return message;
77 +     }
78 +
79 +     public String playerTurnEnd(Player player) {
80 +         int turnScore = player.getTurnScore();
81 +         player.addScore(turnScore);
82 +         String message = player.name + " is ending turn \nTurn score was " + player.getTurnScore() + "\nRound score is "
83 +         turnInProg = false;
84 +         nextPlayer();
85 +
86 +     }
87 +
88 +     public String checkSkunk(Player player) {
89 +         String message = "";
90 +         if (dice.isSkunk() || dice.isDoubleSkunk()) {
91 +             if (dice.isSkunk()) {
92 +                 message = "You rolled a Skunk; end of turn \nTotal score is " + player.getScore();
93 +                 player.addSubChip(-1);
94 +                 kitty += 1;
95 +                 playerTurnEnd(currentPlayer);
96 +             }
97 +             else if (dice.isDoubleSkunk()) {
98 +                 player.addScore(0);
99 +                 message = "You rolled a Double Skunk; \nend of turn \nTotal score is " + player.getScore();
100 +                player.addSubChip(-2);
101 +                kitty += 2;
102 +                playerTurnEnd(currentPlayer);
103 +            }
104 +        }
105 +        return message;
106 +    }
107 +
108 +    public String checkHundred(Player player) {
109 +        String message = "";
110 +        if (player.getTurnScore() >= 100) {
111 +            message = player.name + " has reached 100 points. Continue rolling to raise the round goal, or stop here";
112 +            roundGoal = player.getScore();
113 +        }
114 +        return message;
115 +    }
116 +
117 +    public String roundEnd(Player player) {

```

```

118 +         String message = player.name + " won the round with " + player.playerScore + " points!";
119 +     return message;
120 + }
121 +
122 +
123 + public void showRules() {
124 +
125 +     // The name of the file to open.
126 +     String fileName = "SkunkRules";
127 +
128 +     // This will reference one line at a time
129 +     String line = null;
130 +
131 +     try {
132 +         // FileReader reads text files in the default encoding.
133 +         FileReader fileReader = new FileReader(fileName);
134 +
135 +         // Always wrap FileReader in BufferedReader.
136 +         BufferedReader bufferedReader = new BufferedReader(fileReader);
137 +
138 +         //read line by line in the text file until the line is not null
139 +         while((line = bufferedReader.readLine()) != null) {
140 +             System.out.println(line);
141 +         }
142 +
143 +         // Always close files.
144 +         bufferedReader.close();
145 +     }
146 +     catch(FileNotFoundException ex) {
147 +         System.out.println("Unable to open file '" + fileName + "'");
148 +     }
149 +     catch(EOFException ex) {
150 +         System.out.println("Error reading file '" + fileName + "'");
151 +     }
152 + }
153 +
154 + public int getDie1Value() {
155 +     return dice.getDie1();
156 + }
157 +
158 + public int getDie2Value() {
159 +     return dice.getDie2();
160 + }
161 +
162 + public int getDiceValue() {
163 +     return dice.getDiceValue();
164 + }
165 +
166 +
167 + }

```

```

▼ 192 SkunkProject/src/Dice.java ...
... @@ -1,96 +1,96 @@
1 /**
2 * Dice represents a single pair of rollable Die objects, randomly generating
3 * sums of their two values
4 *
5 * This is a Javadoc comment: add more to your finished class below
6 *
7 * @author eric
8 *
9 */
10
11 public class Dice {
12 {
13     // Instance fields (variables) may be declared anywhere in class body
14     // Convention: put at top
15
16     private int lastRoll;
17     private Die die1; //Dice talks to Die
18     private Die die2; //Dice talks to Die
19
20     // Constructors (object initializers) also can be declared anywhere
21     // Convention: after instance fields/variables
22
23     public Dice()
24     {
25         // initialize instance variables die1 and die2 by
26         // creating a new instance of each
27
28         this.die1 = new Die();
29         this.die2 = new Die();
30     }
31
32     // public Dice(int[] programmableroll)
33     //{
34     //    //int[] programmableroll= programmableroll;
35     //    this.die1 = new Die(programmableroll);
36     //    this.die1 = new Die(programmableroll);
37     //}
38     //}
39
40     public Dice(Die die1, Die die2) // overloaded constructor
41     {
42         this.die1 = die1;
43         this.die2 = die2;
44     }
45
46     // Instance methods can also be declared anywhere
47     // Convention: after constructors

```

```

48
49     -     public int getLastRoll()
50     -     {
51     -         return this.lastRoll;
52     -     }
53
54     -     public int getDie1()
55     -     {
56     -         return die1.getLastRoll();
57     -     }
58
59     -     public int getDie2()
60     -     {
61     -         return die2.getLastRoll();
62     -     }
63
64     -     public void roll()
65     -     {
66     -         // roll each of die1, die2, sum their last rolls,
67     -         // then set Dice.lastRoll to this value
68     -         die1.roll();
69     -         die2.roll();
70     -         this.lastRoll = die1.getLastRoll() + die2.getLastRoll();
71     -     }
72
73     -     // public void roll(int[] programmableroll)
74     -     {
75     -         // roll each of die1, die2, sum their last rolls,
76     -         // then set Dice.lastRoll to this value
77     -         die1.roll(programmableroll);
78     -         die2.roll(programmableroll);
79     -         this.lastRoll = die1.getLastRoll() + die2.getLastRoll();
80     -     }
81
82
83
84     -     // the following method converts the internals of
85     -     // this Dice object, and returns a descriptive String:
86     -     //
87     -     // Roll of 7 => 4 + 3
88     -     //
89
90     -     public String toString()
91     -     {
92     -         return "Dice with last roll: " + getLastRoll() + " => " + die1.getLastRoll() + " + " + die2.getLastRoll();
93     -     }
94
95     -     // static methods can go anywhere - but at end is standard
96     - }
+
1  + public class Dice
2
3
4     +
5     +     private boolean isSkunk;
6     +     private boolean isDoubleSkunk;
7     +     private Die die1; //Dice talks to Die
8     +     private Die die2; //Dice talks to Die
9
10    +     public Dice()
11    +     {
12    +         // initialize instance variables die1 and die2 by
13    +         // creating a new instance of each
14    +
15    +         this.die1 = new Die();
16    +         this.die2 = new Die();
17    +     }
18
19    +     // public Dice(int[] programmableroll)
20    +     {
21    +         //int[] programmableroll= programmableroll;
22    +         this.die1 = new Die(programmableroll);
23    +         this.die2 = new Die(programmableroll);
24    +
25    +     }
26
27    +     public Dice(Die die1, Die die2) // overloaded constructor
28    +     {
29    +         this.die1 = die1;
30    +         this.die2 = die2;
31    +     }
32
33    +     public int getDie1(){
34    +         return die1.getLastRoll();
35    +     }
36
37
38    +     public int getDie2() {
39    +         return die2.getLastRoll();
40    +     }
41
42    +     // Instance methods can also be declared anywhere
43    +     // Convention: after constructors
44
45    +     public void roll(Player player)
46    +     {
47    +         // roll each of die1, die2, sum their last rolls,
48    +         // then set Dice.lastRoll to this value
49    +         die1.roll();
50    +         die2.roll();
51    +
52    +         if (die1.getLastRoll() == 1 || die2.getLastRoll() == 1 && getDiceValue() != 2)
53    +             {

```

```

54 +     player.turnScore = 0;
55 +     isSkunk = true;
56 +
57 +
58 +     else if (getDiceValue() == 2)
59 +     {
60 +         player.turnScore = 0;
61 +         isDoubleSkunk = true;
62 +     }
63 +
64 +     else
65 +     {
66 +         player.turnScore += getDiceValue();
67 +         isSkunk = false;
68 +     }
69 +
70 +
71 +
72 +     public boolean isSkunk()
73 +     {
74 +         return isSkunk;
75 +     }
76 +
77 +     public boolean isDoubleSkunk()
78 +     {
79 +         return isDoubleSkunk;
80 +     }
81 +
82 +     public int getDiceValue()
83 +     {
84 +         return die1.getLastRoll() + die2.getLastRoll();
85 +     }
86 +
87 +     // public void roll(int[] programmableroll)
88 +     //
89 +     // roll each of die1, die2, sum their last rolls,
90 +     // then set Dice.lastRoll to this value
91 +     // die1.roll(programmableroll);
92 +     // die2.roll(programmableroll);
93 +     // this.lastRoll = die1.getLastRoll() + die2.getLastRoll();
94 +     //
95 +
96 + }

```

```

▼ 168 SkunkProject/src/Die.java ...
...
1 ... @@ -1,89 +1,79 @@
2 - import edu.princeton.cs.introcs.*;
3 -
4 - public class Die
5 - {
6 -     private int lastRoll;
7 -
8 -     // -----TP 1.1 changes-----//
9 -     private int[] programmedRoll; // stores user input of preProgrammedDieRolls
10 -    private boolean isItARandomRoll; // controls alternative execution of the roll method
11 -    private int arrayIndex;
12 -
13 -    // -----TP 1.1 changes-----//
14 -    public Die()
15 -    {
16 -        this.isItARandomRoll = true;
17 -        // this.roll(); // executes the roll method
18 -    }
19 -
20 -    // -----TP 1.1 changes-----//
21 -    // Step8: modifying Die to allow it to be initialized with a sequence of
22 -    // "pre-programmed"
23 -    // die values returned by repeatedly rolling such a "loaded" die.
24 -
25 -    public Die(int[] runTimeArgForProgrammedRollArray)
26 -    {
27 -
28 -        if (runTimeArgForProgrammedRollArray == null)
29 -        {
30 -            // creating an instance and throwing it instead calling a method.
31 -            throw new NullPointerException("Empty Array Initialized");
32 -        } else // if runTimeArgForProgrammedRollArray == null
33 -        {
34 -            this.isItARandomRoll = false;
35 -            this.programmedRoll = runTimeArgForProgrammedRollArray;
36 -            this.arrayIndex = 0;
37 -            // this.roll();
38 -        }
39 -    }
40 -    // -----TP 1.1 changes-----//
41 -
42 -    public int getLastRoll() // getter or accessor method
43 -    {
44 -        return this.lastRoll;
45 -    }
46 -
47 -    public int getDie1()
48 -    {
49 -        return this.getLastRoll();
50 -    }
51 -
52 -    public int getDie2()
53 -    {
54 -        return this.getLastRoll();
55 -    }

```

```

56
57     public void roll() // note how this changes Die's state, but doesn't return anything
58     {
59         if (isItARandomRoll == true)
60         {
61             this.lastRoll = (int) (Math.random() * 40 + 1);
62         }
63         // -----
64         // changes-----
65         else // if isItARandomRoll == false
66         {
67
68             this.lastRoll = this.programmedRoll[arrayIndex];
69             // System.out.println("Programmed roll's value: " + lastRoll);
70
71             arrayIndex++;
72
73             if (arrayIndex >= this.programmedRoll.length)
74             {
75                 arrayIndex = 0; // reset the array index variable
76             }
77         }
78
79         // -----
80         // changes-----
81     }
82
83     @Override
84     public String toString() // this OVERRIDES the default Object.toString()
85     {
86         return "Die: " + this.getLastRoll();
87     }
88
89 }
+
1 import edu.princeton.cs.introcs.*;
2
3 public class Die
4 {
5     private int lastRoll;
6
7     // -----
7     // changes-----
8     private int[] programmedRoll; // stores user input of preProgrammedDieRolls
9     private boolean isItARandomRoll; // controls alternative execution of the roll method
10    private int arrayIndex;
11
12    // -----
13    // changes-----
14    public Die()
15    {
16        this.isItARandomRoll = true;
17        // this.roll(); // executes the roll method
18    }
19
19    // -----
20    // TP 1.1
21    // Step8: modifying Die to allow it to be initialized with a sequence of
22    // "pre-programmed"
23    // die values returned by repeatedly rolling such a "loaded" die.
24
25    public Die(int[] runTimeArgForProgrammedRollArray)
26    {
27
28        if (runTimeArgForProgrammedRollArray == null)
29        {
30            // creating an instance and throwing it instead calling a method.
31            throw new NullPointerException("Empty Array Initialized");
32        } else // if runTimeArgForProgrammedRollArray == null
33        {
34            this.isItARandomRoll = false;
35            this.programmedRoll = runTimeArgForProgrammedRollArray;
36            this.arrayIndex = 0;
37            // this.roll();
38        }
39    }
40    // -----
41
42    public int getLastRoll() // getter or accessor method
43    {
44        return this.lastRoll;
45    }
46
47    public void roll() // note how this changes Die's state, but doesn't return anything
48    {
49        if (isItARandomRoll == true)
50        {
51            this.lastRoll = (int) (Math.random() * 40 + 1);
52        }
53        // -----
54        // changes-----
55        else // if isItARandomRoll == false
56        {
57
58            this.lastRoll = this.programmedRoll[arrayIndex];
59            // System.out.println("Programmed roll's value: " + lastRoll);
60
61            arrayIndex++;
62
63            if (arrayIndex >= this.programmedRoll.length)
64            {
65                arrayIndex = 0; // reset the array index variable
66            }
67        }
68

```

```

69 +         // -----
70 +         // changes-----
71 +     }
72 +
73 +     @Override
74 +     public String toString() // this OVERIDES the default Object.toString()
75 +     {
76 +         return "Die: " + this.getLastRoll();
77 +     }
78 +
79 + }

```

SkunkProject/src/Player.java

```

@@ -1,37 +1,49 @@
1 - import edu.princeton.cs.introcs.Stdout;
2 -
3 - public class Player {
4 -
5 -     public int playerScore; //uninitialized value is zero
6 -     public String name; //uninitialized value is null
7 -     // public boolean hundred; //uninitialized value is false
8 -     public int chip; //uninitialized value is zero
9 -
10 -
11 -     public Player () {
12 -
13 -     }
14 -
15 -     public Player (String name) {
16 -         this.name = name;
17 -         this.chip = 50;
18 -     }
19 -
20 -
21 -     public int getchip() {
22 -         return chip;
23 -     }
24 -
25 -     public void chipPenalty(int subChip) {
26 -         chip = chip - subChip;
27 -     }
28 -
29 -     public int getScore() {
30 -         return playerScore;
31 -     }
32 -
33 -     public void setScore(int score) {
34 -         playerScore += score;
35 -     }
36 -
37 - }
1 + import edu.princeton.cs.introcs.Stdout;
2 +
3 + public class Player {
4 +
5 +
6 +     public int playerScore; //uninitialized value is zero
7 +     public int turnScore;
8 +     public int roundScore;
9 +     public String name; //uninitialized value is null
10 +    public int chip; //uninitialized value is zero
11 +
12 +
13 +    public Player () {
14 +
15 +    }
16 +
17 +    public Player (String name) {
18 +        this.name = name;
19 +        this.turnScore = 0;
20 +        this.chip = 50;
21 +    }
22 +
23 +
24 +    public int getchip() {
25 +        return chip;
26 +    }
27 +
28 +    public void addSubChip(int chipChange) {
29 +        chip = chip + chipChange;
30 +    }
31 +
32 +
33 +    public int getScore() {
34 +        return playerScore;
35 +    }
36 +
37 +    public void addScore(int score) {
38 +        playerScore += score;
39 +    }
40 +
41 +    public int getTurnScore() {

```

```
42     +         return turnScore;
43     +     }
44     +
45     +     public void setTurnScore(int turnScore) {
46     +         this.turnScore = turnScore;
47     +     }
48     +
49 }
```

```
▼ 20 SkunkProject/src/SkunkApp.java
...
... ... @@ -1,11 +1,11 @@
1 - import java.util.Arrays;
2 -
3 - // -----TP 1.2 changes-----
4 - public class SkunkApp
5 - {
6 -     public static void main(String[] args)
7 -     {
8 -         AppRunner game = new AppRunner();
9 -         game.displayGame();
10 -     }
11 + import java.util.Arrays;
12 +
13 + // -----TP 1.2 changes-----
14 + public class SkunkApp
15 + {
16 +     public static void main(String[] args)
17 +     {
18 +         AppRunner game = new AppRunner();
19 +         game.displayGame();
20 +     }
21 }
```

```
▼ 100 SkunkProject/src/Turn.java
...
... ...
Load diff
This file was deleted.
```

```
▼ 24 SkunkProject/test/AppRunnerTest.java
...
... ... @@ -1,12 +1,12 @@
1 - import static org.junit.Assert.*;
2 -
3 - import org.junit.Test;
4 -
5 - public class AppRunnerTest {
6 -
7 -     @Test
8 -     public void test() {
9 -         fail("Not yet implemented");
10 -     }
11 -
12 - }
13 + import static org.junit.Assert.*;
14 +
15 + import org.junit.Test;
16 +
17 + public class AppRunnerTest {
18 +
19 -     @Test
20 -     public void test() {
21 -         fail("Not yet implemented");
22 -     }
23 + }
```

```
▼ 112 SkunkProject/test/DiceTest.java
...
... ... @@ -1,57 +1,57 @@
1 - import static org.junit.Assert.assertEquals;
2 - import org.junit.Before;
3 - import org.junit.Test;
4 -
5 - //-----TP 1.1 changes-----
6 - public class DiceTest
7 - {
8 -     private Dice dice;
9 -     private Die die1, die2;
10 -     private int[] rollDie1, rollDie2;
11 -
12 -     @Before
13 -     public void setup() throws Exception
14 -     {
15 -         this.rollDie1 = new int[] { 1, 2, 3 };
16 -         this.die1 = new Die(rollDie1);
17 -
18 -         this.rollDie2 = new int[] { 1, 2, 3 };
19 -         this.die2 = new Die(rollDie2);
20 -
21 -         this.dice = new Dice(die1, die2);
22 -     }
23 -
24 -     @Test
25 -     public void test_Init_PredictableDie()
26 -     {
27 -         dice.roll();
28 -         int value = dice.getLastRoll(); // adds rollDie1's position1 + rollDie2's position1
-         assertEquals(2, value);
+         assertEquals(2, value);

```

```

29      }
30
31      @Test
32      public void test_Predictable_Roll_2()
33      {
34          dice.roll();
35          dice.roll();
36          int value = dice.getLastRoll(); // adds rollDie1's position2 + rollDie2's position2
37          assertEquals(4, value);
38      }
39
40      @Test
41      public void test_Predictable_Roll_3()
42      {
43          dice.roll();
44          dice.roll();
45          dice.roll();
46          int value = dice.getLastRoll();
47          assertEquals(6, value); // adds rollDie1's position3 + rollDie2's position3
48      }
49
50      @Test
51      public void test_toString()
52      {
53          dice.toString();
54          assertEquals("Dice with last roll:  =>  + ", "Dice with last roll:  =>  + ");
55      }
56
57      + import static org.junit.Assert.assertEquals;
58      + import org.junit.Before;
59      + import org.junit.Test;
60      + //-----TP 1.1 changes-----/
61      + public class DiceTest
62      + {
63          +     private Dice dice;
64          +     private Die die1, die2;
65          +     private int[] rollDie1, rollDie2;
66          +
67          +     @Before
68          +     public void setUp() throws Exception
69          {
70              +         this.rollDie1 = new int[] { 1, 2, 3 };
71              +         this.die1 = new Die(rollDie1);
72              +
73              +         this.rollDie2 = new int[] { 1, 2, 3 };
74              +         this.die2 = new Die(rollDie2);
75              +
76              +         this.dice = new Dice(die1, die2);
77          }
78
79          +     @Test
80          +     public void test_Init_PredictableDie()
81          {
82              dice.roll();
83              int value = dice.getLastRoll(); // adds rollDie1's position1 + rollDie2's position1
84              assertEquals(2, value);
85          }
86
87          +     @Test
88          +     public void test_Predictable_Roll_2()
89          {
90              dice.roll();
91              dice.roll();
92              int value = dice.getLastRoll(); // adds rollDie1's position2 + rollDie2's position2
93              assertEquals(4, value);
94          }
95
96          +     @Test
97          +     public void test_Predictable_Roll_3()
98          {
99              dice.roll();
100             dice.roll();
101             dice.roll();
102             int value = dice.getLastRoll();
103             assertEquals(6, value); // adds rollDie1's position3 + rollDie2's position3
104         }
105
106         +     @Test
107         +     public void test_toString()
108         {
109             dice.toString();
110             assertEquals("Dice with last roll:  =>  + ", "Dice with last roll:  =>  + ");
111         }
112     }
113 }
```

```

v 132 SkunkProject/test/DieTest.java ...
...
1 ... @@ -1,66 +1,66 @@
2 - import static org.junit.Assert.assertEquals;
3 - import java.util.Arrays;
4 - import org.junit.Assert;
5 - import org.junit.Rule;
6 - import org.junit.Test;
7 - import org.junit.rules.ExpectedException;
8 - //-----TP 1.1 changes-----/
9 - public class DieTest {
10 -
11     @Rule
12     public final ExpectedException exception = ExpectedException.none();
13 }
```

```

14 -     @Test
15 -     //Test that preprogrammed Die object is working as expected
16 -     public void loadedDieTest2() {
17 -         int[] progRoll = {1,2,6,5};
18 -         Die die1 = new Die (progRoll);
19 -         int[] dieTest = new int[10];
20 -         //roll preprogrammed Die object 10 times; should run through the preprogrammed values then start over
21 -         for (int i=0; i<10; i++) {
22 -             die1.roll();
23 -             dieTest[i] = die1.getLastRoll();
24 -             System.out.println(dieTest[i]);
25 -         }
26 -
27 -         String expected = Arrays.toString(new int[] {1,2,6,5,1,2,6,5,1,2});
28 -         String actual = Arrays.toString(dieTest);
29 -
30 -         Assert.assertEquals(expected,actual);
31 -     }
32 -
33 -     @Test
34 -     //test that the nullPointerException is thrown when an empty array is passed as a parameter to Die obj
35 -     public void nullExceptionTest() {
36 -         exception.expect(NullPointerException.class);
37 -         exception.expectMessage("Empty Array Initialized");
38 -         int[] emptyArray = null;
39 -         Die empty = new Die(emptyArray);
40 -     }
41 -
42 -
43 -     @Test
44 -     public void loadedDieTest() {
45 -         int[] progRoll = {1,2,6,5};
46 -         Die die1 = new Die (progRoll);
47 -         int[] dieTest = new int[4];
48 -         int arrayIndex;
49 -
50 -         //for (int arrayIndex=progRoll.length-1; arrayIndex>=1; arrayIndex=arrayIndex - 1) { // forloop for iterating the
51 -         //
52 -         for (arrayIndex=0; arrayIndex<dieTest.length; arrayIndex++) //forloop for iterating the array forwards
53 -         {
54 -             die1.roll();
55 -             dieTest[arrayIndex] = die1.getLastRoll();
56 -             System.out.println(dieTest[arrayIndex]);
57 -         }
58 -
59 -         String actual = Arrays.toString(dieTest );
60 -         String expected = Arrays.toString(new int[] {1,2,6,5});
61 -
62 -         //Assert.assertEquals(actual,expected);
63 -         assertEquals(expected,actual); // rewritten code
64 -     }
65 -
66 - }
1 + import static org.junit.Assert.assertEquals;
2 + import java.util.Arrays;
3 + import org.junit.Assert;
4 + import org.junit.Rule;
5 + import org.junit.Test;
6 + import org.junit.rules.ExpectedException;
7 +
8 + //-----TP 1.1 changes-----/
9 + public class DieTest {
10 +
11 +     @Rule
12 +     public final ExpectedException exception = ExpectedException.none();
13 +
14 +     @Test
15 +     //Test that preprogrammed Die object is working as expected
16 +     public void loadedDieTest2() {
17 +         int[] progRoll = {1,2,6,5};
18 +         Die die1 = new Die (progRoll);
19 +         int[] dieTest = new int[10];
20 +         //roll preprogrammed Die object 10 times; should run through the preprogrammed values then start over
21 +         for (int i=0; i<10; i++) {
22 +             die1.roll();
23 +             dieTest[i] = die1.getLastRoll();
24 +             System.out.println(dieTest[i]);
25 +         }
26 +
27 +         String expected = Arrays.toString(new int[] {1,2,6,5,1,2,6,5,1,2});
28 +         String actual = Arrays.toString(dieTest);
29 +
30 +         Assert.assertEquals(expected,actual);
31 +     }
32 +
33 +     @Test
34 +     //test that the nullPointerException is thrown when an empty array is passed as a parameter to Die obj
35 +     public void nullExceptionTest() {
36 +         exception.expect(NullPointerException.class);
37 +         exception.expectMessage("Empty Array Initialized");
38 +         int[] emptyArray = null;
39 +         Die empty = new Die(emptyArray);
40 +     }
41 +
42 +
43 +     @Test
44 +     public void loadedDieTest() {
45 +         int[] progRoll = {1,2,6,5};
46 +         Die die1 = new Die (progRoll);
47 +         int[] dieTest = new int[4];
48 +         int arrayIndex;
49 +

```

```
50 +
51 +
52 +         //for ( int arrayIndex=progRoll.length-1; arrayIndex>=1; arrayIndex=arrayIndex -1) { // forloop for iterating the
53 +
54 +             for (arrayIndex=0; arrayIndex<dieTest.length; arrayIndex++) //forloop for iterating the array forwards
55 +             {
56 +                 die1.roll();
57 +                 dieTest[arrayIndex] = die1.getLastRoll();
58 +                 System.out.println(dieTest[arrayIndex]);
59 +
60 +             }
61 +
62 +             String actual = Arrays.toString(dieTest );
63 +             String expected = Arrays.toString(new int[] {1,2,6,5});
64 +             //Assert.assertEquals(actual,expected);
65 +             assertEquals(expected,actual); // rewritten code
66 +         }
67 +
68 +     }
```

```
 88 SkunkProject/test/PlayerTest.java
```

... ... @@ -1,40 +1,40 @@

- import org.junit.Assert;

- import org.junit.Before;

- import org.junit.Test;

-

- // Assert.assertEquals (expected, actual)

- // where expected = premethod output

- // actual = post method output

- public class PlayerTest {

- Player p1; Player p2; Player p3;

-

- @Before

- public void setup() throws Exception{

- p1 = new Player("pratiksh"); //calling the overloaded constructor in the Player Class (name = pratiksh, chip = 50)

- p2 = new Player("anna"); //calling the overloaded constructor in the Player Class (name = anna, chip = 50)

- p3 = new Player("khadija"); //calling the overloaded constructor in the Player Class (name = khadija, chip = 50)

- }

-

- @Test

- public void constructorTest() {

- Assert.assertEquals("pratiksh", p1.name); // due to object creation we know p1.name should be pratiksh

- Assert.assertEquals(50, p1.getChip()); //method gets the default value of the chip variable

- Assert.assertEquals(0, p1.getScore()); //method gets the default value of the playerScore variable

- }

-

- @Test

- public void testPenalty() {

- Assert.assertEquals(50, p2.getChip()); //method gets the default value of the chip variable

- p2.chipPenalty(10); // //method sets the updated value of the chip variable based on the argument passed into the function

- Assert.assertEquals(40, p2.getChip()); //method gets the updated value of the chip variable

- }

-

- @Test

- public void setScoreTest() {

- p3.setScore(100); //method sets the updated value of the playerScore variable based on the argument passed into the function

- Assert.assertEquals(100, p3.getScore()); //method gets the updated value of the playerScore variable

- }

- }

+ import org.junit.Assert;

+ import org.junit.Before;

+ import org.junit.Test;

+

+ // Assert.assertEquals (expected, actual)

+ // where expected = premethod output

+ // actual = post method output

+ public class PlayerTest {

+ Player p1; Player p2; Player p3;

+

+ @Before

+ public void setup() throws Exception{

+ p1 = new Player("pratiksh"); //calling the overloaded constructor in the Player Class (name = pratiksh, chip = 50)

+ p2 = new Player("anna"); //calling the overloaded constructor in the Player Class (name = anna, chip = 50)

+ p3 = new Player("khadija"); //calling the overloaded constructor in the Player Class (name = khadija, chip = 50)

+ }

+

+ @Test

+ public void constructorTest() {

+ Assert.assertEquals("pratiksh", p1.name); // due to object creation we know p1.name should be pratiksh

+ Assert.assertEquals(50, p1.getChip()); //method gets the default value of the chip variable

+ Assert.assertEquals(0, p1.getScore()); //method gets the default value of the playerScore variable

+ }

+

+ @Test

+ public void testPenalty() {

+ Assert.assertEquals(50, p2.getChip()); //method gets the default value of the chip variable

+ p2.chipPenalty(10); // //method sets the updated value of the chip variable based on the argument passed into the function

+ Assert.assertEquals(40, p2.getChip()); //method gets the updated value of the chip variable

+ }

+

+ @Test

+ public void setScoreTest() {

+ p3.setScore(100); //method sets the updated value of the playerScore variable based on the argument passed into the function

+ Assert.assertEquals(100, p3.getScore()); //method gets the updated value of the playerScore variable

+ }

+ }

61 SkunkProject/test/TurnTest.java

Load diff  
This file was deleted.

0 comments on commit b6a97f5

Write Preview Lock conversation

Leave a comment

Attach files by dragging & dropping, selecting or pasting them.

Comment on this commit

(A) [Subscribe](#) You're not receiving notifications from this thread.

