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

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 @@
 /AppRunner.class
 /AppRunnerTest.class
+/Controller.class
 /Dice.class
 /DiceTest.class
 /Die.class
@@ -9,7 +10,3 @@
 /SkunkApp.class
 /Turn.class
 /TurnTest.class
-/TestSuite.class
-/TestSuiteRunner.class
-/TestSuite$TestSuiteExample.class
-/Controller.class
```

119 SkunkProject/src/AppRunner.java

```
@@ -1,71 +1,48 @@
-import java.io.BufferedReader;
-import java.io.FileNotFoundException;
-import java.io.FileReader;
-import java.io.IOException;
-import java.util.ArrayList;
-import java.util.Scanner;
-
-import edu.princeton.cs.introcs.*;
-
-//------------------------TP 1.2 changes-----------------------------------------------------//
-public class AppRunner {
-
-	public int numPlayers;
-
-
-	public void displayGame() {
-		Controller controller = new Controller();
-		StdOut.println("Welcome to 635 Skunk project");
-		StdOut.println("Creating the dice object");
-		Scanner scan = new Scanner(System.in);
-		StdOut.println("Would you like to see the rules? (Yes/No)");
-		if (scan.next().equalsIgnoreCase("Yes") ) {
-			controller.showRules();
-		}
-		StdOut.println("Number of players: ");
-		numPlayers = scan.nextInt();
-
-		for (int i = 1; i <= numPlayers; i++) {
-			StdOut.println("Player " + i + " Name: ");
-			controller.createPlayer(scan.next());
-		}
-
-		// Player Turn (For loop was used because we know that a Skunk game has 5 turns
-		// )
-
-//		while (!controller.isHundred) {
-
-			for (Player player : controller.playerList) {
-				StdOut.println(player.name + " start your turn ?(Yes/No)"); // user prompt
-				String decision = scan.next(); // storing user input
-				if (decision.equalsIgnoreCase("No")) { // execute if user input = no
-					StdOut.println(player.name + " skipped turn ");
-				}
-
-				while (decision.equalsIgnoreCase("Yes")) { // execute if user input = yes
-					controller.playerTurn();
-					StdOut.println(player.name + " would you like to roll? (Yes/No)");
-					String rollTurn = scan.next();
-					if (rollTurn.equalsIgnoreCase("Yes")) {
-						StdOut.println(controller.playerTurnContinue(player));
-
-					}
-
-					else if (rollTurn.equalsIgnoreCase("No")) {
-						controller.playerTurnEnd(player);
-						break;
-					}
-				} // end of while loop
-
-			} // end of for loop
-
-			controller.showRules();
```

```
-
-
-                } // end of while loop
-
-  //      } // end of playgame method
-
-         // Updated showRules() method to display info from a text file called SkunkRules for code reusability
-
-  } // end of class
+  import java.io.BufferedReader;
+  import java.io.FileNotFoundException;
+  import java.io.FileReader;
+  import java.io.IOException;
+  import java.util.ArrayList;
+  import java.util.Scanner;
+
+  import edu.princeton.cs.introcs.*;
+
+  //-------------------------TP 1.2 changes-------------------------------------------------//
+  public class AppRunner {
+          boolean turnInProg;
+          boolean roundInProg;
+          Player currentPlayer;
+
+          public void displayGame() {
+                  Controller c = new Controller();
+                  StdOut.println("Welcome to 635 Skunk project");
+                  StdOut.println("Creating the dice object");
+                  Scanner scan = new Scanner(System.in);
+                  StdOut.println("Would you like to see the rules? (Yes/No)");
+                  if (scan.next().equalsIgnoreCase("Yes") ) {
+                          c.showRules();
+                  }
+                  StdOut.println("Number of players: ");
+                  int numPlayers = scan.nextInt();
+                  c.createPlayer(numPlayers);
+                  roundInProg = true;
+
+                  while(roundInProg) {
+                          currentPlayer = c.currentPlayer;
+                          StdOut.println(currentPlayer.name + " start your turn ?(Yes/No)");
+                          String decision = scan.next();
+                          c.playerTurn(decision);
+
+                          while (c.turnInProg) {
+                                  StdOut.println(c.currentPlayer.name + " would you like to roll? (Yes/No)");
+                                  String rollTurn = scan.next();
+                                  if (rollTurn.equalsIgnoreCase("Yes")) {
+                                          StdOut.println(c.playerTurnContinue(currentPlayer));
+                                  }
+                                  else if (rollTurn.equalsIgnoreCase("No")) {
+                                          StdOut.println(c.playerTurnEnd(currentPlayer));
+                                  }
+                          }
+                  }
+          }
+  }// end of class
```

295 SkunkProject/src/Controller.java

```
...  ...  @@ -1,128 +1,167 @@
-  import java.io.BufferedReader;
-  import java.io.FileNotFoundException;
-  import java.io.FileReader;
-  import java.io.IOException;
-  import java.util.ArrayList;
-
-  import edu.princeton.cs.introcs.StdOut;
-
-  public class Controller {
-
-          public ArrayList<Player> playerList = new ArrayList();
-          public Turn turn;
-
-
-
-          public Player createPlayer(String name) {
-                  Player player = new Player(name);
-                  playerList.add(player);
-                  return player;
-          }
-
-          public void playerTurn() {
-                  turn = new Turn();
-          }
-
-          public String playerTurnContinue(Player player) {
-                  turn.turnRoll();
-                  return rollInfo(player);
-          }
-
-          public String rollInfo(Player player) {
-                  String message = "Roll Info: "
-                                  + "\nDie 1: " + getDie1Value() + ";" + " Die 2: " + getDie2Value()
-                                  + "\nRoll total: " + getDiceValue()
-                                  + "\nTurn total: " + getTurnScore()
-                                  + "\nGame total: " + player.getScore()
-                                  + "\n" + checkSkunk(player)
-                                  + "\n" + hundredCheck(player);
-                  return message;
-          }
-
```

```
-         public String playerTurnEnd(Player player) {
-                 player.setScore(getTurnScore());
-                 String message = player.name + "is ending turn \nTurn score was " + getTurnScore() + "\nTotal score is " + player
-                 return message;
-
-         }
-
-         public String checkSkunk(Player player) {
-                 String message = "";
-                 if (turn.isSkunk() || turn.isDoubleSkunk()) {
-                         if (turn.isSkunk()) {
-                                 message = "You rolled a Skunk; end of turn \nTotal score is " + player.getScore();
-                                 player.chipPenalty(1);
-                         }
-                         else if (turn.isDoubleSkunk()) {
-                                 player.setScore(0);
-                                 message = "You rolled a Double Skunk; \nend of turn \nTotal score is " + player.getScore();
-                                 player.chipPenalty(2);
-                         }
-                 }
-                 return message;
-         }
-
-         public  String hundredCheck(Player player) {
-                 String message = "";
-                 if (player.getScore() >= 100) {
-                         message = roundEnd(player);
-                 }
-                 return message;
-
-         }
-
-         public String roundEnd(Player player) {
-                 String message =  player.name + " won the game with " + player.playerScore + " points!";
-                 return message;
-         }
-
-         public void showRules() {
-
-                  // The name of the file to open.
-        String fileName = "SkunkRules";
-
-        // This will reference one line at a time
-        String line = null;
-
-        try {
-            // FileReader reads text files in the default encoding.
-            FileReader fileReader = new FileReader(fileName);
-
-            // Always wrap FileReader in BufferedReader.
-            BufferedReader bufferedReader = new BufferedReader(fileReader);
-
-            //read line by line in the text file until the line is not null
-            while((line = bufferedReader.readLine()) != null) {
-                System.out.println(line);
-            }
-
-            // Always close files.
-            bufferedReader.close();
-        }
-        catch(FileNotFoundException ex) {
-            System.out.println("Unable to open file '" + fileName + "'");
-        }
-        catch(IOException ex) {
-         System.out.println("Error reading file '" + fileName + "'");
-        }
-    }
-
-         public int getDie1Value() {
-                 return turn.getDie1Value();
-         }
-
-         public int getDie2Value() {
-                 return turn.getDie2Value();
-         }
-
-         public int getDiceValue() {
-                 return turn.getDiceValue();
-         }
-
-         public int getTurnScore() {
-                 return turn.getTurnScore();
-         }
-
-
-
- }
+ import java.io.BufferedReader;
+ import java.io.FileNotFoundException;
+ import java.io.FileReader;
+ import java.io.IOException;
+ import java.util.ArrayList;
+ import java.util.Scanner;
+
+ import edu.princeton.cs.introcs.StdOut;
+
+ public class Controller {
+
+         public ArrayList<Player> playerList = new ArrayList();
+         public Dice dice;
+         public int kitty;
+         public int roundGoal;
```

```
+	public Player currentPlayer;
+	public int currentPlayerIndex;
+	public boolean turnInProg;
+
+	public void createPlayer(int numPlayers) {
+		for (int j = 1; j <= numPlayers; j++) {
+			StdOut.println("Player " + j + " Name: ");
+			Scanner scan = new Scanner(System.in);
+			Player player = new Player(scan.next());
+			playerList.add(player);
+		}
+		currentPlayerIndex=0;
+		currentPlayer = playerList.get(currentPlayerIndex);
+
+		StdOut.println("First player: " + currentPlayer.name);
+	}
+
+	public void nextPlayer() {
+		if (currentPlayerIndex == (playerList.size() - 1)) {
+			currentPlayerIndex = 0;
+			currentPlayer = playerList.get(currentPlayerIndex);
+		} else {
+			currentPlayerIndex++;
+			currentPlayer = playerList.get(currentPlayerIndex);
+		}
+	}
+
+	//Questioning if player wants to begin turn or skip
+	public String playerTurn(String decision) {
+		String message = null;
+		if (decision.equalsIgnoreCase("No")) { // execute if user input = no
+			turnInProg = false;
+			message = currentPlayer.name + " skipped turn ";
+			nextPlayer();
+		}
+		else if (decision.equalsIgnoreCase("Yes")) {
+			turnInProg = true;
+			dice = new Dice();
+			message = currentPlayer.name + "'s turn";
+		}
+
+		return message;
+	}
+
+	//Method for player continuing their turn
+	public String playerTurnContinue(Player player) {
+		dice.roll(player);
+		return rollInfo(player);
+	}
+
+	public String rollInfo(Player player) {
+		String message = "Roll Info: "
+				+ "\nDie 1: " + getDie1Value() + ";" + " Die 2: " + getDie2Value()
+				+ "\nRoll total: " + getDiceValue()
+				+ "\nTurn total: " + player.getTurnScore()
+				+ "\nGame total: " + player.getScore()
+				+ "\nKitty total: " + player.getChip()
+				+ "\n" + checkSkunk(player)
+				+ "\n" + checkHundred(player);
+		return message;
+	}
+
+	public String playerTurnEnd(Player player) {
+		int turnScore = player.getTurnScore();
+		player.addScore(turnScore);
+		String message = player.name + " is ending turn \nTurn score was " + player.getTurnScore() + "\nRound score is "
+		turnInProg = false;
+		nextPlayer();
+		return message;
+
+	}
+
+	public String checkSkunk(Player player) {
+		String message = "";
+		if (dice.isSkunk() || dice.isDoubleSkunk()) {
+			if (dice.isSkunk()) {
+				message = "You rolled a Skunk; end of turn \nTotal score is " + player.getScore();
+				player.addSubChip(-1);
+				kitty += 1;
+				playerTurnEnd(currentPlayer);
+			}
+			else if (dice.isDoubleSkunk()) {
+				player.addScore(0);
+				message = "You rolled a Double Skunk; \nend of turn \nTotal score is " + player.getScore();
+				player.addSubChip(-2);
+				kitty += 2;
+				playerTurnEnd(currentPlayer);
+			}
+		}
+		return message;
+	}
+
+	public  String checkHundred (Player player) {
+		String message = "";
+		if (player.getTurnScore() >= 100) {
+			message = player.name + " has reached 100 points.  Continue rolling to raise the round goal, or stop here
+			roundGoal = player.getScore();
+		}
+		return message;
+	}
+
+	public String roundEnd(Player player) {
```

```
+                String message =  player.name + " won the round with " + player.playerScore + " points!";
+                return message;
+        }
+
+
+        public void showRules() {
+
+                 // The name of the file to open.
+       String fileName = "SkunkRules";
+
+       // This will reference one line at a time
+       String line = null;
+
+       try {
+           // FileReader reads text files in the default encoding.
+           FileReader fileReader = new FileReader(fileName);
+
+           // Always wrap FileReader in BufferedReader.
+           BufferedReader bufferedReader = new BufferedReader(fileReader);
+
+           //read line by line in the text file until the line is not null
+           while((line = bufferedReader.readLine()) != null) {
+               System.out.println(line);
+           }
+
+           // Always close files.
+           bufferedReader.close();
+       }
+       catch(FileNotFoundException ex) {
+           System.out.println("Unable to open file '" + fileName + "'");
+       }
+       catch(IOException ex) {
+        System.out.println("Error reading file '" + fileName + "'");
+       }
+   }
+
+        public int getDie1Value() {
+                return dice.getDie1();
+        }
+
+        public int getDie2Value() {
+                return dice.getDie2();
+        }
+
+        public int getDiceValue() {
+                return dice.getDiceValue();
+        }
+
+
+ }
```

192 SkunkProject/src/Dice.java

```
@@ -1,96 +1,96 @@
-/**
- * Dice represents a single pair of rollable Die objects, randomly generating
- * sums of their two values
- *
- * This is a Javadoc comment: add more to your finished class below
- *
- * @author eric
- *
- */
-
-public class Dice
-{
-        // Instance fields (variables) may be declared anywhere in class body
-        // Convention: put at top
-
-        private int lastRoll;
-        private Die die1; //Dice talks to Die
-        private Die die2; //Dice talks to Die
-
-        // Constructors (object initializers) also can be declared anywhere
-        // Convention: after instance fields/variables
-
-        public Dice()
-        {
-                // initialize instance variables die1 and die2 by
-                // creating a new instance of each
-
-                this.die1 = new Die();
-                this.die2 = new Die();
-        }
-
-//      public Dice(int[] programmableroll)
-//      {
-//              //int[] programmableroll= programmableroll;
-//              this.die1 = new Die(programmableroll);
-//              this.die1 = new Die(programmableroll);
-//
-//      }
-
-        public Dice(Die die1, Die die2) // overloaded constructor
-        {
-                this.die1 = die1;
-                this.die2 = die2;
-        }
-
-        // Instance methods can also be declared anywhere
-        // Convention: after constructors
```

```
-
-       public int getLastRoll()
-       {
-               return this.lastRoll;
-       }
-
-       public int getDie1()
-       {
-               return die1.getDie1();
-       }
-
-       public int getDie2()
-       {
-               return die2.getDie2();
-       }
-
-       public void roll()
-       {
-               // roll each of die1, die2, sum their last rolls,
-               // then set Dice.lastRoll to this value
-               die1.roll();
-               die2.roll();
-               this.lastRoll = die1.getLastRoll() + die2.getLastRoll();
-       }
-
-//     public void roll(int[] programmableroll)
-//     {
-//             // roll each of die1, die2, sum their last rolls,
-//             // then set Dice.lastRoll to this value
-//             die1.roll(programmableroll);
-//             die2.roll(programmableroll);
-//             this.lastRoll = die1.getLastRoll() + die2.getLastRoll();
-//     }
-
-
-
-       // the following method converts the internals of
-       // this Dice object, and returns a descriptive String:
-       //
-       // Roll of 7 => 4 + 3
-       //
-       public String toString()
-       {
-               return "Dice with last roll: " + getLastRoll() + " => " + die1.getLastRoll() + " + " + die2.getLastRoll();
-       }
-
-       // static methods can go anywhere - but at end is standard
- }
+
+ public class Dice
+
+ {
+       private boolean isSkunk;
+       private boolean isDoubleSkunk;
+       private Die die1; //Dice talks to Die
+       private Die die2; //Dice talks to Die
+
+       public Dice()
+       {
+               // initialize instance variables die1 and die2 by
+               // creating a new instance of each
+
+               this.die1 = new Die();
+               this.die2 = new Die();
+       }
+
+ //    public Dice(int[] programmableroll)
+ //    {
+ //            //int[] programmableroll= programmableroll;
+ //            this.die1 = new Die(programmableroll);
+ //            this.die1 = new Die(programmableroll);
+ //
+ //    }
+
+       public Dice(Die die1, Die die2) // overloaded constructor
+       {
+               this.die1 = die1;
+               this.die2 = die2;
+       }
+       public int getDie1() {
+               return die1.getLastRoll();
+       }
+
+
+       public int getDie2() {
+               return die2.getLastRoll();
+       }
+       // Instance methods can also be declared anywhere
+       // Convention: after constructors
+
+       public void roll(Player player)
+       {
+               // roll each of die1, die2, sum their last rolls,
+               // then set Dice.lastRoll to this value
+               die1.roll();
+               die2.roll();
+
+
+               if (die1.getLastRoll() == 1 || die2.getLastRoll() == 1 && getDiceValue() != 2)
+               {
```

```
+                player.turnScore = 0;
+                isSkunk = true;
+            }
+
+            else if (getDiceValue() == 2)
+            {
+                player.turnScore = 0;
+                isDoubleSkunk = true;
+            }
+
+            else
+            {
+                player.turnScore += getDiceValue();
+                isSkunk = false;
+            }
+        }
+
+
+        public boolean isSkunk()
+        {
+            return isSkunk;
+        }
+
+        public boolean isDoubleSkunk() {
+            return isDoubleSkunk;
+        }
+
+        public int getDiceValue()
+        {
+            return die1.getLastRoll() + die2.getLastRoll();
+        }
+
+//      public void roll(int[] programmableroll)
+//      {
+//          // roll each of die1, die2, sum their last rolls,
+//          // then set Dice.lastRoll to this value
+//          die1.roll(programmableroll);
+//          die2.roll(programmableroll);
+//          this.lastRoll = die1.getLastRoll() + die2.getLastRoll();
+//      }
+
+
+}
```

168 SkunkProject/src/Die.java

```
@@ -1,89 +1,79 @@
-import edu.princeton.cs.introcs.*;
-
-public class Die
-{
-        private int lastRoll;
-
-        // -------------------------TP 1.1 changes-------------------------------------------------//
-        private int[] programmedRoll; // stores user input of preProgrammedDieRolls
-        private boolean isItARandomRoll; // controls alternative execution of the roll method
-        private int arrayIndex;
-
-        // -------------------------TP 1.1 changes-------------------------------------------------//
-        public Die()
-        {
-                this.isItARandomRoll = true;
-                // this.roll(); // executes the roll method
-        }
-
-        // -------------------------TP 1.1 changes-------------------------------------------------//
-        // TP 1.1
-        // Step8: modifying Die to allow it to be initialized with a sequence of
-        // "pre-programmed"
-        // die values returned by repeatedly rolling such a "loaded" die.
-
-        public Die(int[] runTimeArgForProgrammedRollArray)
-        {
-
-                if (runTimeArgForProgrammedRollArray == null)
-                {
-                        // creating an instance and throwing it instead calling a method.
-                        throw new NullPointerException("Empty Array Initialized");
-                } else // if runTimeArgForProgrammedRollArray == null
-                {
-                        this.isItARandomRoll = false;
-                        this.programmedRoll = runTimeArgForProgrammedRollArray;
-                        this.arrayIndex = 0;
-                        // this.roll();
-                }
-        }
-        // -------------------------TP 1.1 changes-------------------------------------------------//
-
-        public int getLastRoll() // getter or accessor method
-        {
-                return this.lastRoll;
-        }
-
-        public int getDie1()
-        {
-                return this.getLastRoll();
-        }
-
-        public int getDie2()
-        {
-                return this.getLastRoll();
-        }
```

```
-
-	public void roll() // note how this changes Die's state, but doesn't return anything
-	{
-		if (isItARandomRoll == true)
-		{
-			this.lastRoll = (int) (Math.random() * 40 + 1);
-		}
-		// -------------------------TP 1.1
-		// changes---------------------------------------------------//
-		else // if isItARandomRoll == false
-		{
-
-			this.lastRoll = this.programmedRoll[arrayIndex];
-			// System.out.println("Programmed roll's value: " + lastRoll);
-
-			arrayIndex++;
-
-			if (arrayIndex >= this.programmedRoll.length)
-			{
-				arrayIndex = 0; // reset the array index variable
-			}
-		}
-
-		// -------------------------TP 1.1
-		// changes---------------------------------------------------//
-	}
-
-	@Override
-	public String toString() // this OVERRIDES the default Object.toString()
-	{
-		return "Die: " + this.getLastRoll();
-	}
-
- }
+ import edu.princeton.cs.introcs.*;
+
+ public class Die
+ {
+	private int lastRoll;
+
+	// -------------------------TP 1.1 changes---------------------------------------------------//
+	private int[] programmedRoll; // stores user input of preProgrammedDieRolls
+	private boolean isItARandomRoll; // controls alternative execution of the roll method
+	private int arrayIndex;
+
+	// -------------------------TP 1.1 changes---------------------------------------------------//
+	public Die()
+	{
+		this.isItARandomRoll = true;
+		// this.roll(); // executes the roll method
+	}
+
+	// -------------------------TP 1.1 changes---------------------------------------------------//
+	// TP 1.1
+	// Step8: modifying Die to allow it to be initialized with a sequence of
+	// "pre-programmed"
+	// die values returned by repeatedly rolling such a "loaded" die.
+
+	public Die(int[] runTimeArgForProgrammedRollArray)
+	{
+
+		if (runTimeArgForProgrammedRollArray == null)
+		{
+			// creating an instance and throwing it instead calling a method.
+			throw new NullPointerException("Empty Array Initialized");
+		} else // if runTimeArgForProgrammedRollArray == null
+		{
+			this.isItARandomRoll = false;
+			this.programmedRoll = runTimeArgForProgrammedRollArray;
+			this.arrayIndex = 0;
+			// this.roll();
+		}
+	}
+	// -------------------------TP 1.1 changes---------------------------------------------------//
+
+	public int getLastRoll() // getter or accessor method
+	{
+		return this.lastRoll;
+	}
+
+	public void roll() // note how this changes Die's state, but doesn't return anything
+	{
+		if (isItARandomRoll == true)
+		{
+			this.lastRoll = (int) (Math.random() * 40 + 1);
+		}
+		// -------------------------TP 1.1
+		// changes---------------------------------------------------//
+		else // if isItARandomRoll == false
+		{
+
+			this.lastRoll = this.programmedRoll[arrayIndex];
+			// System.out.println("Programmed roll's value: " + lastRoll);
+
+			arrayIndex++;
+
+			if (arrayIndex >= this.programmedRoll.length)
+			{
+				arrayIndex = 0; // reset the array index variable
+			}
+		}
+
```

```
+                 // -------------------------TP 1.1
+                 // changes-------------------------------------------------//
+         }
+
+         @Override
+         public String toString() // this OVERRIDES the default Object.toString()
+         {
+                 return "Die: " + this.getLastRoll();
+         }
+
+ }
```

86 SkunkProject/src/Player.java

```
@@ -1,37 +1,49 @@
- import edu.princeton.cs.introcs.StdOut;
-
- public class Player {
-
-         public int playerScore; //uninitialized value is zero
-         public String name; //uninitialized value is null
- //      public boolean hundred; //uninitialized value is false
-         public int chip; //uninitialized value is zero
-
-
-         public Player () {
-
-         }
-
-         public Player (String name) {
-                 this.name = name;
-                 this.chip = 50;
-         }
-
-
-         public int getChip() {
-                 return chip;
-         }
-
-         public void chipPenalty(int subChip) {
-                 chip = chip - subChip;
-         }
-
-         public int getScore() {
-                 return playerScore;
-         }
-
-         public void setScore(int score) {
-                 playerScore += score;
-         }
-
- }
+ import edu.princeton.cs.introcs.StdOut;
+
+ public class Player {
+
+
+         public int playerScore; //uninitialized value is zero
+         public int turnScore;
+         public int roundScore;
+         public String name; //uninitialized value is null
+         public int chip; //uninitialized value is zero
+
+
+         public Player () {
+
+         }
+
+         public Player (String name) {
+                 this.name = name;
+                 this.turnScore = 0;
+                 this.chip = 50;
+         }
+
+
+         public int getChip() {
+                 return chip;
+         }
+
+         public void addSubChip(int chipChange) {
+                 chip = chip + chipChange;
+         }
+
+
+         public int getScore() {
+                 return playerScore;
+         }
+
+         public void addScore(int score) {
+                 playerScore += score;
+         }
+
+         public int getTurnScore() {
```

```
+                return turnScore;
+        }
+
+        public void setTurnScore(int turnScore) {
+                this.turnScore = turnScore;
+        }
+
+ }
```

20 SkunkProject/src/SkunkApp.java

```
@@ -1,11 +1,11 @@
- import java.util.Arrays;
-
- // ------------------------TP 1.2 changes-------------------------------------------------//
- public class SkunkApp
- {
-         public static void main(String[] args)
-         {
-                 AppRunner game = new AppRunner();
-                 game.displayGame();
-         }
+ import java.util.Arrays;
+
+ // ------------------------TP 1.2 changes-------------------------------------------------//
+ public class SkunkApp
+ {
+         public static void main(String[] args)
+         {
+                 AppRunner game = new AppRunner();
+                 game.displayGame();
+         }
  }
```

100 SkunkProject/src/Turn.java

Load diff

This file was deleted.

24 SkunkProject/test/AppRunnerTest.java

```
@@ -1,12 +1,12 @@
- import static org.junit.Assert.*;
-
- import org.junit.Test;
-
- public class AppRunnerTest {
-
-         @Test
-         public void test() {
-                 fail("Not yet implemented");
-         }
-
- }
+ import static org.junit.Assert.*;
+
+ import org.junit.Test;
+
+ public class AppRunnerTest {
+
+         @Test
+         public void test() {
+                 fail("Not yet implemented");
+         }
+
+ }
```

112 SkunkProject/test/DiceTest.java

```
@@ -1,57 +1,57 @@
- import static org.junit.Assert.assertEquals;
- import org.junit.Before;
- import org.junit.Test;
- //------------------------TP 1.1 changes-------------------------------------------------//
- public class DiceTest
- {
-         private Dice dice;
-         private Die die1, die2;
-         private int[] rollDie1, rollDie2;
-
-         @Before
-         public void setUp() throws Exception
-         {
-                 this.rollDie1 = new int[] { 1, 2, 3 };
-                 this.die1 = new Die(rollDie1);
-
-                 this.rollDie2 = new int[] { 1, 2, 3 };
-                 this.die2 = new Die(rollDie2);
-
-                 this.dice = new Dice(die1, die2);
-         }
-
-         @Test
-         public void test_Init_PredictableDie()
-         {
-                 dice.roll();
-                 int value = dice.getLastRoll();  // adds rollDie1's position1 + rollDie2's position1
-                 assertEquals(2, value);
```

```
-         }
-
-         @Test
-         public void test_Predictable_Roll_2()
-         {
-                 dice.roll();
-                 dice.roll();
-                 int value = dice.getLastRoll();  // adds rollDie1's position2 + rollDie2's position2
-                 assertEquals(4, value);
-         }
-
-         @Test
-         public void test_Predictable_Roll_3()
-         {
-                 dice.roll();
-                 dice.roll();
-                 dice.roll();
-                 int value = dice.getLastRoll();
-                 assertEquals(6, value);  // adds rollDie1's position3 + rollDie2's position3
-         }
-
-         @Test
-         public void test_toString()
-         {
-                 dice.toString();
-                 assertEquals("Dice with last roll:   =>  + ", "Dice with last roll:   =>  + ");
-         }
-
+ import static org.junit.Assert.assertEquals;
+ import org.junit.Before;
+ import org.junit.Test;
+ //------------------------TP 1.1 changes-------------------------------------------------//
+ public class DiceTest
+ {
+         private Dice dice;
+         private Die die1, die2;
+         private int[] rollDie1, rollDie2;
+
+         @Before
+         public void setUp() throws Exception
+         {
+                 this.rollDie1 = new int[] { 1, 2, 3 };
+                 this.die1 = new Die(rollDie1);
+
+                 this.rollDie2 = new int[] { 1, 2, 3 };
+                 this.die2 = new Die(rollDie2);
+
+                 this.dice = new Dice(die1, die2);
+         }
+
+         @Test
+         public void test_Init_PredictableDie()
+         {
+                 dice.roll();
+                 int value = dice.getLastRoll();  // adds rollDie1's position1 + rollDie2's position1
+                 assertEquals(2, value);
+         }
+
+         @Test
+         public void test_Predictable_Roll_2()
+         {
+                 dice.roll();
+                 dice.roll();
+                 int value = dice.getLastRoll();  // adds rollDie1's position2 + rollDie2's position2
+                 assertEquals(4, value);
+         }
+
+         @Test
+         public void test_Predictable_Roll_3()
+         {
+                 dice.roll();
+                 dice.roll();
+                 dice.roll();
+                 int value = dice.getLastRoll();
+                 assertEquals(6, value);  // adds rollDie1's position3 + rollDie2's position3
+         }
+
+         @Test
+         public void test_toString()
+         {
+                 dice.toString();
+                 assertEquals("Dice with last roll:   =>  + ", "Dice with last roll:   =>  + ");
+         }
+
  }⊘↵
```

132 SkunkProject/test/DieTest.java

```
... ... @@ -1,66 +1,66 @@
- import static org.junit.Assert.assertEquals;
- import java.util.Arrays;
- import org.junit.Assert;
- import org.junit.Rule;
- import org.junit.Test;
- import org.junit.rules.ExpectedException;
-
- //------------------------TP 1.1 changes-------------------------------------------------//
- public class DieTest {
-
-         @Rule
-           public final ExpectedException exception = ExpectedException.none();
-
```

```
-	@Test
-	//Test that preprogrammed Die object is working as expected
-	public void loadedDieTest2() {
-		int[] progRoll = {1,2,6,5};
-		Die die1 = new Die (progRoll);
-		int[] dieTest = new int[10];
-		//roll preprogrammed Die object 10 times; should run through the preprogrammed values then start over
-		for (int i=0; i<=9; i++) {
-			die1.roll();
-			dieTest[i] = die1.getLastRoll();
-			System.out.println(dieTest[i]);
-		}
-
-		String expected = Arrays.toString(new int[] {1,2,6,5,1,2,6,5,1,2});
-		String actual = Arrays.toString(dieTest);
-
-		Assert.assertEquals(expected,actual);
-	}
-
-	@Test
-	//test that the nullPointerException is thrown when an empty array is passed as a parameter to Die obj
-	public void nullExceptionTest() {
-		exception.expect(NullPointerException.class);
-		exception.expectMessage("Empty Array Initialized");
-		int[] emptyArray = null;
-		Die empty = new Die(emptyArray);
-	}
-
-
-	@Test
-	public void loadedDieTest() {
-		int[] progRoll = {1,2,6,5};
-		Die die1 = new Die (progRoll);
-		int[] dieTest = new int[4];
-		int arrayIndex;
-
-		//for (int arrayIndex=progRoll.length-1; arrayIndex>=1; arrayIndex=arrayIndex -1) { // forloop for iterating the
-
-		  for (arrayIndex=0; arrayIndex<dieTest.length; arrayIndex++) //forloop for iterating the array forwards
-		  {
-			die1.roll();
-			dieTest[arrayIndex] = die1.getLastRoll();
-			System.out.println(dieTest[arrayIndex]);
-		  }
-
-		String actual = Arrays.toString(dieTest );
-		String expected = Arrays.toString(new int[] {1,2,6,5});
-
-		//Assert.assertEquals(actual,expected);
-		assertEquals(expected,actual); // rewritten code
-	}
-
-}
+import static org.junit.Assert.assertEquals;
+import java.util.Arrays;
+import org.junit.Assert;
+import org.junit.Rule;
+import org.junit.Test;
+import org.junit.rules.ExpectedException;
+
+//-------------------------TP 1.1 changes-------------------------------------------------//
+public class DieTest {
+
+	@Rule
+	  public final ExpectedException exception = ExpectedException.none();
+
+	@Test
+	//Test that preprogrammed Die object is working as expected
+	public void loadedDieTest2() {
+		int[] progRoll = {1,2,6,5};
+		Die die1 = new Die (progRoll);
+		int[] dieTest = new int[10];
+		//roll preprogrammed Die object 10 times; should run through the preprogrammed values then start over
+		for (int i=0; i<=9; i++) {
+			die1.roll();
+			dieTest[i] = die1.getLastRoll();
+			System.out.println(dieTest[i]);
+		}
+
+		String expected = Arrays.toString(new int[] {1,2,6,5,1,2,6,5,1,2});
+		String actual = Arrays.toString(dieTest);
+
+		Assert.assertEquals(expected,actual);
+	}
+
+	@Test
+	//test that the nullPointerException is thrown when an empty array is passed as a parameter to Die obj
+	public void nullExceptionTest() {
+		exception.expect(NullPointerException.class);
+		exception.expectMessage("Empty Array Initialized");
+		int[] emptyArray = null;
+		Die empty = new Die(emptyArray);
+	}
+
+
+	@Test
+	public void loadedDieTest() {
+		int[] progRoll = {1,2,6,5};
+		Die die1 = new Die (progRoll);
+		int[] dieTest = new int[4];
+		int arrayIndex;
+
```

```
+                //for (int arrayIndex=progRoll.length-1; arrayIndex>=1; arrayIndex=arrayIndex -1) { // forloop for iterating the
+
+                  for (arrayIndex=0; arrayIndex<dieTest.length; arrayIndex++) //forloop for iterating the array forwards
+                  {
+                        die1.roll();
+                        dieTest[arrayIndex] = die1.getLastRoll();
+                        System.out.println(dieTest[arrayIndex]);
+                  }
+
+                String actual = Arrays.toString(dieTest );
+                String expected = Arrays.toString(new int[] {1,2,6,5});
+
+                //Assert.assertEquals(actual,expected);
+                assertEquals(expected,actual); // rewritten code
+        }
+
+ }
```

80 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 functio
-          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 funct
-          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 functio
+          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 funct
+          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`

Lock conversation

Write Preview

Leave a comment

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

Comment on this commit

Subscribe You're not receiving notifications from this thread.