CMSC 215

Programming Project 1 – Tallest Basketball Player

Daniel Smolsky

Section 1 – Approach

My approach to this project was to read the design guidelines and produce a plan to implement the requirements. Creating the Player and Height classes was straight forward, they did not have any complicated methods. I planned to have all my logic requirements in the Project1 class, which I implemented last using the methods and variables existing in my Player and Height classes.

Section 2 – Assumptions

The main assumption I made was that the user would enter valid information when prompted for variables. The only source of error catching in my code is an exception thrown in my Height class if the user enters a negative value for height. Otherwise, if the user enters letters when prompted for height or age the program will crash.

Section 3 – Not Implemented

Section 4 – User Guide:

Extract the .zip file and run the Project1.java file. When prompted for inputs, enter the number of players you would like to run the data for, the name of each player, the height of each player in format #FEET #INCHES as integer, and the age of each player as integer.

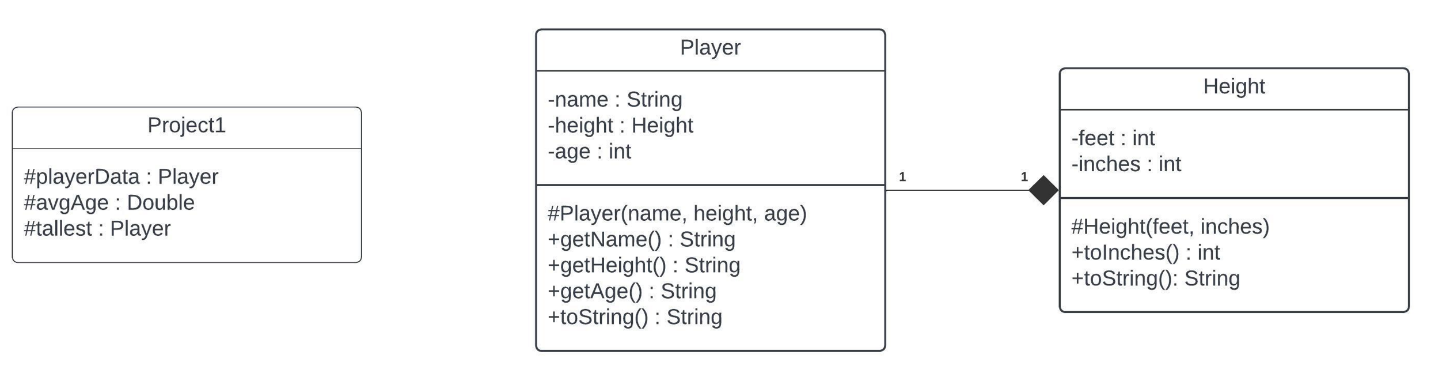
Section 5 – Lessons Learned

Creating this project, I learned how to implement classes and how to reference instance variables using the “this” keyword within the class. I improved my understanding of how to create object instances and how to use their methods from the main method. Additionally, I learned how to throw exceptions when specific conditions are met to safeguard the program from erroneous user inputs.

Section 6 – Possible Improvements

To improve my code further, I could have made Player a subclass of Height, as all players have a Height it follows that they can inherit all height variables and methods. I could not implement this because the project requirements necessitated the Player class to have a Height private variable, which would not be necessary if Player was a subclass of height as it would already be implemented within Height. Additionally, I could have created more ways to ensure correct user input, such as re prompting the user for valid integers if they enter letters for height or age, but this was outside the scope of the project.

Section 7 – UML



Section 8 – Source Code

*/\**

*\* Daniel Smolsky*

*\* Programming Project 1: Tallest Basketball Player*

*\* January 12, 2024*

*\* This class promps the user for information about each basketball player. The information is*

*\* stored in an array, and then later accessed to determine the tallest player below average age.*

*\*/*

package project1;

import java.util.ArrayList;

import java.util.Scanner;

public class Project1 {

public static void **main** (String[] *args*) {

Scanner input = new Scanner(System.in);

System.out.println("How many players will be part of the data set?");

int playerCount = input.nextInt();

ArrayList<Player> playersData = new ArrayList<>();

int combinedAge = 0;

for (int i = 0; i < playerCount; i++) {

System.out.print("Enter player " + (i + 1) + " information:\nName: ");

String tempName = input.next();

System.out.print("Height (Feet Inches): ");

int tempHeightFeet = input.nextInt();

int tempHeightInches = input.nextInt();

System.out.print("Age: ");

int tempAge = input.nextInt();

combinedAge += tempAge;

try {

playersData.add(new Player(tempName, new Height(tempHeightFeet, tempHeightInches), tempAge));

}

catch (IllegalArgumentException *e*) {

System.out.println("invalid height provided, enter information for this player again.");

i--;

}

}

double avgAge = (double) combinedAge / playerCount;

Player tallest = null;

for (Player player : playersData) {

if (player.getAge() <= avgAge) {

if (tallest == null)

tallest = player;

else if (player.getHeight().toInches() > tallest.getHeight().toInches())

tallest = player;

}

}

System.out.println("the tallest player is " + tallest.toString());

input.close();

}

}

*/\**

*\* Daniel Smolsky*

*\* Programming Project 1: Tallest Basketball Player*

*\* January 12, 2024*

*\* This class is used to create player objects. This class has 3 getter methods to access*

*\* all of the stored variables and a toString method to return all of the player information.*

*\*/*

package project1;

public class Player {

private String name;

private Height height;

private int age;

Player(String *name*, Height *height*, int *age*) {

this.name = *name*;

this.height = *height*;

this.age = *age*;

}

public String **getName**() {

return this.name;

}

public Height **getHeight**() {

return this.height;

}

public int **getAge**() {

return this.age;

}

public String **toString**() {

return "name: " + this.name + ", player height: " + this.height.toString() + ", player age: " + this.age;

}

}

*/\**

*\* Daniel Smolsky*

*\* Programming Project 1: Tallest Basketball Player*

*\* January 12, 2024*

*\* This class is used to create player height objects with two getter methods used*

*\* to access the player height in inches and their height as a string.*

*\*/*

package project1;

public class Height {

private int feet;

private int inches;

Height(int *feet*, int *inches*) {

if (*feet* >= 0 && *inches* >= 0) {

this.feet = *feet* + *inches* / 12;

this.inches = *inches* % 12;

}

else

throw new IllegalArgumentException();

}

public int **toInches**() {

return this.feet \* 12 + this.inches;

}

public String **toString**() {

return this.feet + "\' " + this.inches + "\"";

}

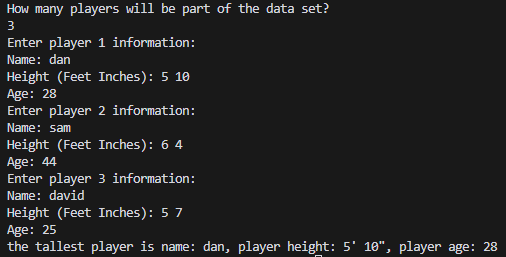
}

Section 9 – Test Plans

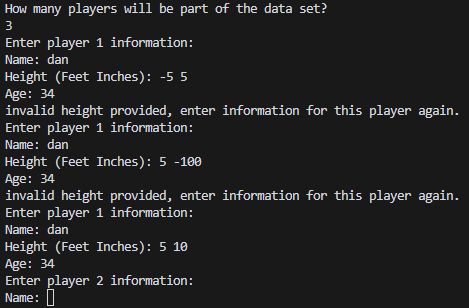
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test # | Purpose | Pos/Neg Test | Input Values | Expected Result | Pass/Fail |
| 1 | Confirm valid inputs produce correct output | Positive | # players: 3  Name: dan  Height: 5 10  Age: 28  Name: sam  Height 6 4  Age: 44  Name: david  Height: 5 7  Age: 25 | the tallest player is name: dan, player height: 5' 10", player age: 28 | Pass |
| 2 | Confirm invalid Heights require data re-entry | Negative | # players: 3  Name: dan  Height: -5 5  Age: 34 | Invalid height provided, enter information for this player again. | pass |
| 3 | Confirm program crashes on character input into height / age field | Negative | # players: 3  Name: dan  Height: 5 feet 10 inches | Input Mismatch Exception | pass |
| 4 | Program gives incorrect output if negative ages are input | Negative | # players: 3  Name: dan  Height: 5 10  Age: -50  Name: sam  Height: 6 3  Age: -10  Name: david  Height: 5 5  Age: 30 | the tallest player is name: sam, player height: 6' 3", player age: -10 | pass |

Section 10 – Screen Shots

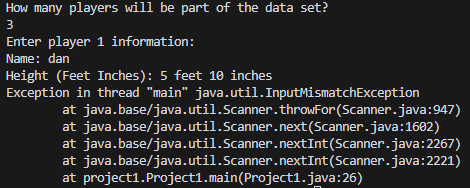
Test 1 -



Test 2 -



Test 3 -



Test 4 -

