Table of contents

Introduction

In this project I have wanted to show that knowing the basics of object-oriented programming, we are able to create a console application.

The application was created with the use:

- loops,
- conditional statements,
- switch statement,
- primitive types,
- arrays,
- lists,
- static methods,
- inheritance,
- encapsulation,
- polymorphism,
- Scanner class.

Description

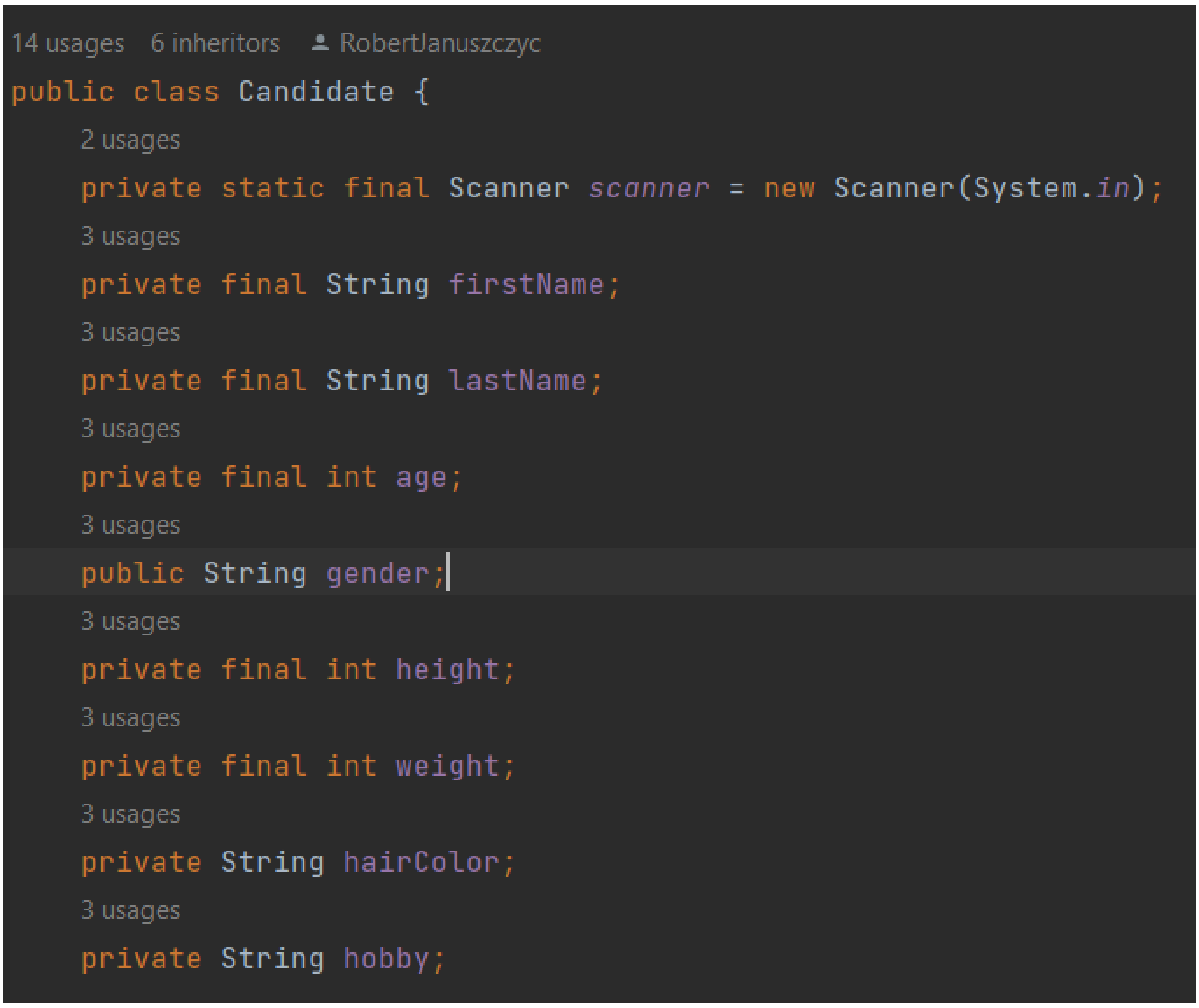
The application enables the user to find the perfect candidate by matching his preferences with the characteristics of the candidates who have applied for the dating program. The candidate who meets the most criteria and scores the most points, wins!!!

Creation Process

Initially I have created class called **Candidate** and fields which describe the characteristic each candidate.

Please find below these fields:

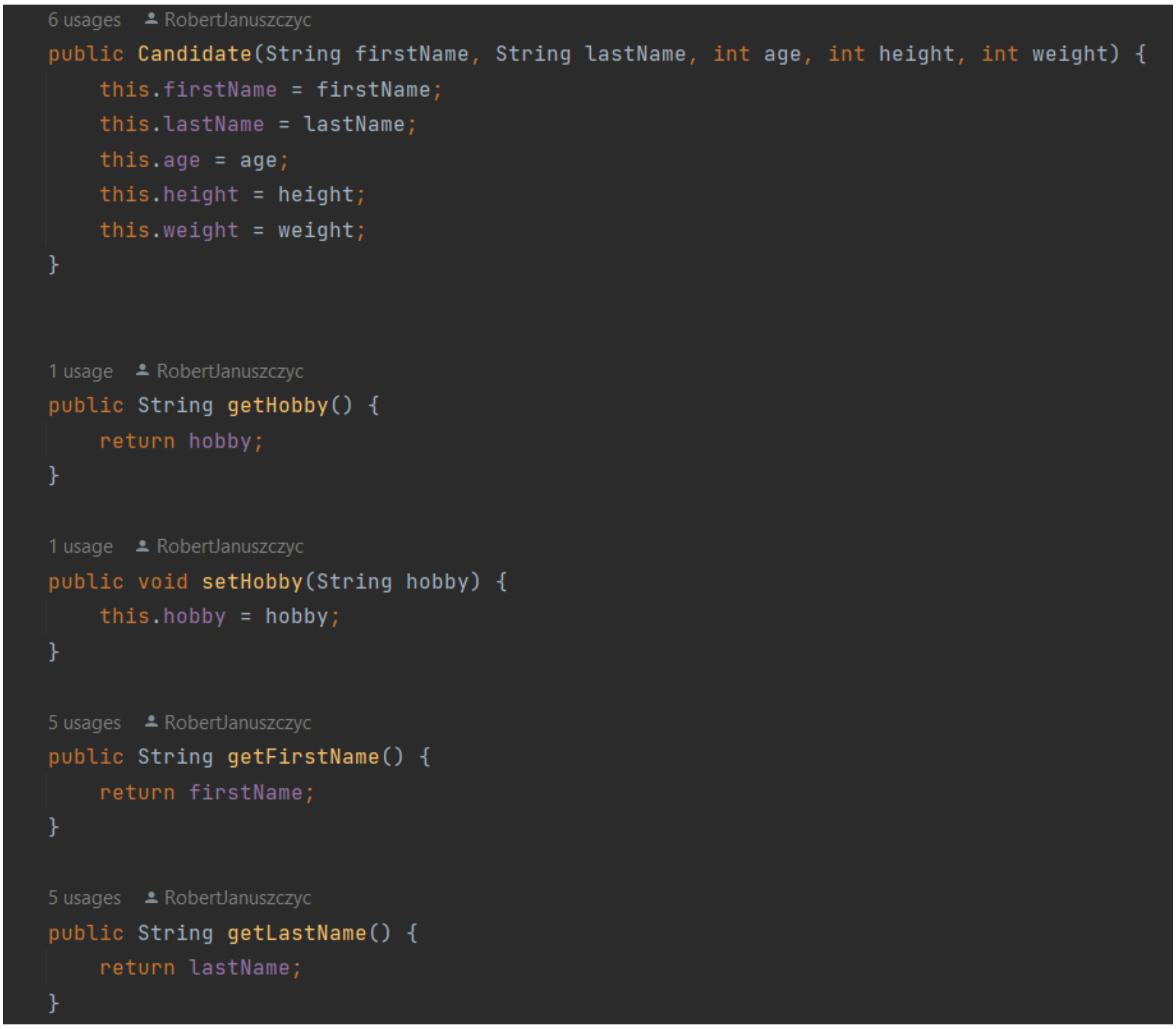
- First name and last name,
- age,
- gender,
- height,
- weight,
- hair color.



```
14 usages   6 inheritors   RobertJanuszczyc
public class Candidate {
    2 usages
    private static final Scanner scanner = new Scanner(System.in);
    3 usages
    private final String firstName;
    3 usages
    private final String lastName;
    3 usages
    private final int age;
    3 usages
    public String gender;
    3 usages
    private final int height;
    3 usages
    private final int weight;
    3 usages
    private String hairColor;
    3 usages
    private String hobby;
```

Figure 1 Candidate class fields

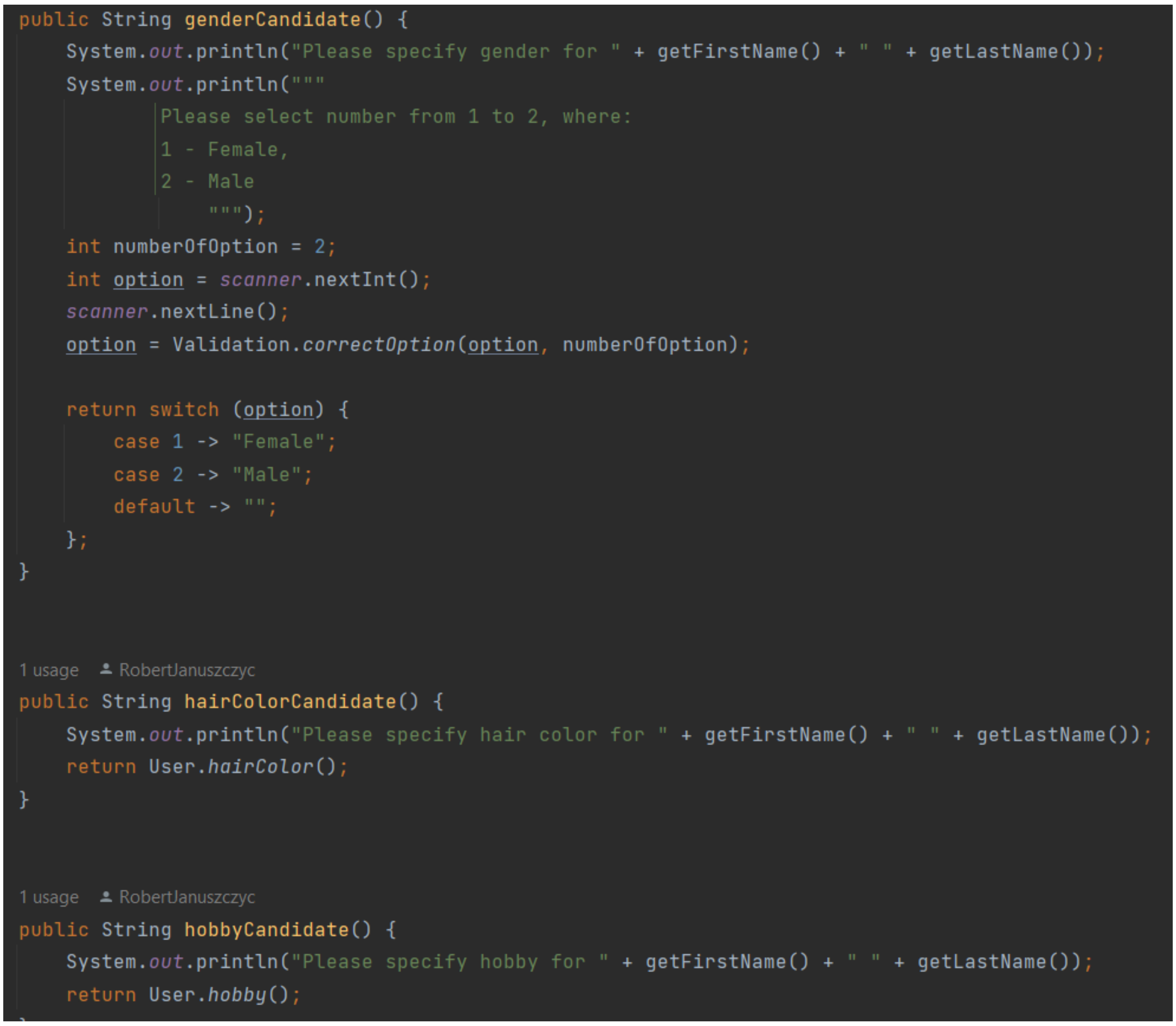
All fields have private access modifier so the next crucial step was to create a constructor as well as getters and setters for the fields.



```
6 usages   RobertJanuszczyc
public Candidate(String firstName, String lastName, int age, int height, int weight) {
    this.firstName = firstName;
    this.lastName = lastName;
    this.age = age;
    this.height = height;
    this.weight = weight;
}


1 usage   RobertJanuszczyc
public String getHobby() {
    return hobby;
}

1 usage   RobertJanuszczyc
public void setHobby(String hobby) {
    this.hobby = hobby;
}

5 usages   RobertJanuszczyc
public String getFirstName() {
    return firstName;
}

5 usages   RobertJanuszczyc
public String getLastName() {
    return lastName;
}
```

Figure 2 Candidate class constructor, getters and setters

Also there are three methods to define gender, hair color and hobby for each candidate.



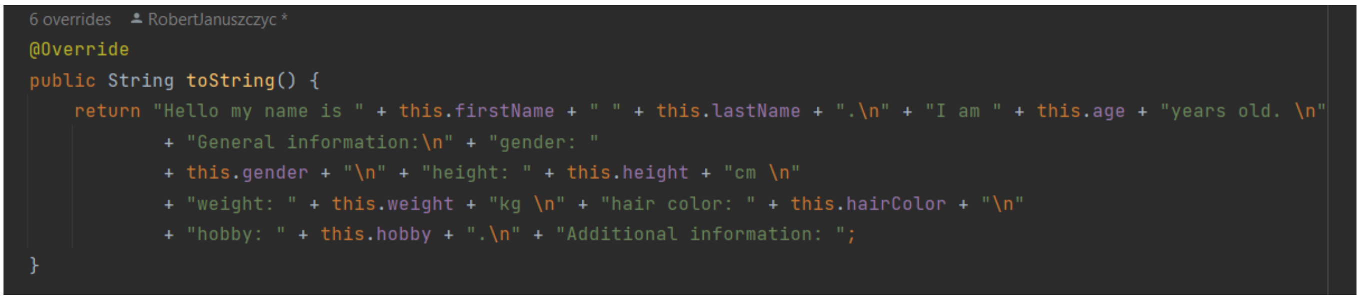
```java
public String genderCandidate() {
    System.out.println("Please specify gender for " + getFirstName() + " " + getLastName());
    System.out.println("""
            Please select number from 1 to 2, where:
            1 - Female,
            2 - Male
              """);
    int numberOfOption = 2;
    int option = scanner.nextInt();
    scanner.nextLine();
    option = Validation.correctOption(option, numberOfOption);

    return switch (option) {
        case 1 -> "Female";
        case 2 -> "Male";
        default -> "";
    };
}


1 usage   RobertJanuszczyc
public String hairColorCandidate() {
    System.out.println("Please specify hair color for " + getFirstName() + " " + getLastName());
    return User.hairColor();
}


1 usage   RobertJanuszczyc
public String hobbyCandidate() {
    System.out.println("Please specify hobby for " + getFirstName() + " " + getLastName());
    return User.hobby();
```

Figure 3 Candidate class methods

The candidates characteristic will be compared to the preferences of the application user. It is a reason why there methods from the **Scanner** class and **switch** statement were used. The introduction of these methods will allow to avoid unnecessary mistakes (typos).

The two methods use a functions from the another **"User"** class because the user is also forced to specify a hobby and a preferred hair color. The similar functionality of these methods allows them to be used in this case. In this way we can avoid unnecessary code duplication.

In the **Candidate** class there is also an override **toString** method, which was created to display an exact description for each of the candidates.



```
6 overrides   RobertJanuszczyc *
@Override
public String toString() {
    return "Hello my name is " + this.firstName + " " + this.lastName + ".\n" + "I am " + this.age + "years old. \n"
            + "General information:\n" + "gender: "
            + this.gender + "\n" + "height: " + this.height + "cm \n"
            + "weight: " + this.weight + "kg \n" + "hair color: " + this.hairColor + "\n"
            + "hobby: " + this.hobby + ".\n" + "Additional information: ";
}
```

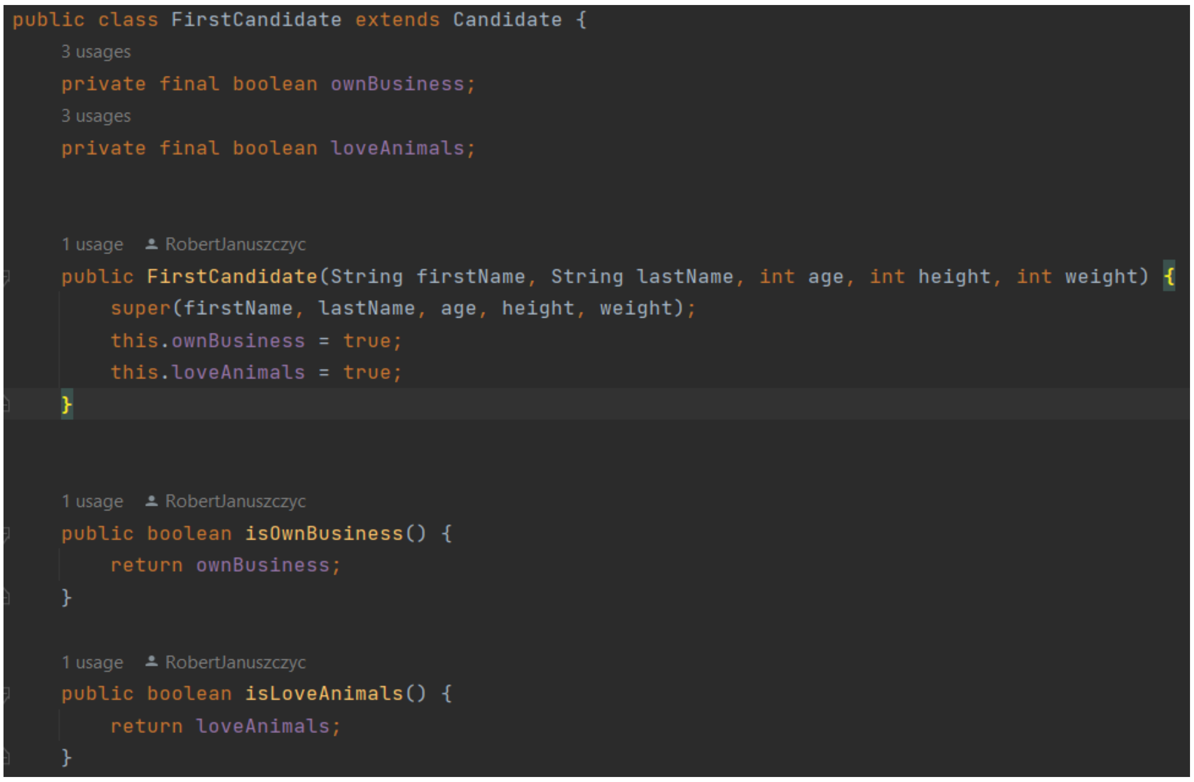
Figure 4 The toString method in Candidate class

The next step was to create six classes representing six candidates extended by the **Candidate** class.

Classes representing candidates:

- FirstCandidate,
- SecondCandidate,
- ThirdCandidate,
- FourthCandidate,
- FifthCandidate,
- SixthCandidate.

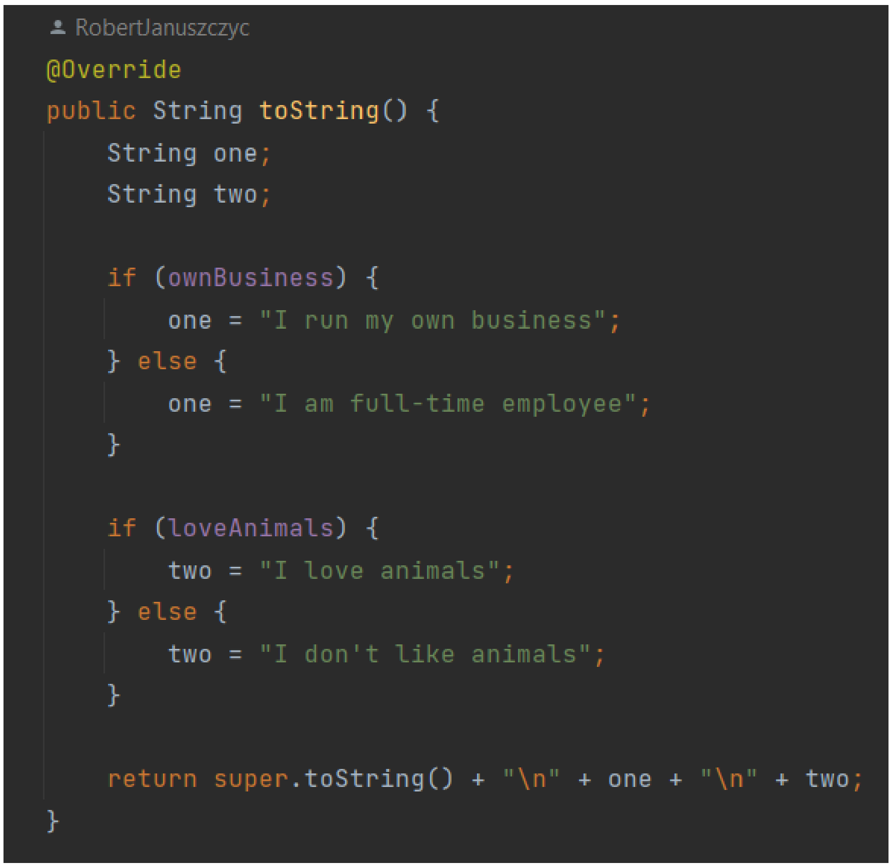
In each class, two additional fields assigned to a concrete candidate were created, as well as a constructor and getters related to the fields visible below.



```
public class FirstCandidate extends Candidate {
    3 usages
    private final boolean ownBusiness;
    3 usages
    private final boolean loveAnimals;


    1 usage   RobertJanuszczyc
    public FirstCandidate(String firstName, String lastName, int age, int height, int weight) {
        super(firstName, lastName, age, height, weight);
        this.ownBusiness = true;
        this.loveAnimals = true;
    }


    1 usage   RobertJanuszczyc
    public boolean isOwnBusiness() {
        return ownBusiness;
    }

    1 usage   RobertJanuszczyc
    public boolean isLoveAnimals() {
        return loveAnimals;
    }
```

Figure 5 Additional fields, constructor, and getters in the class created for the first candidate

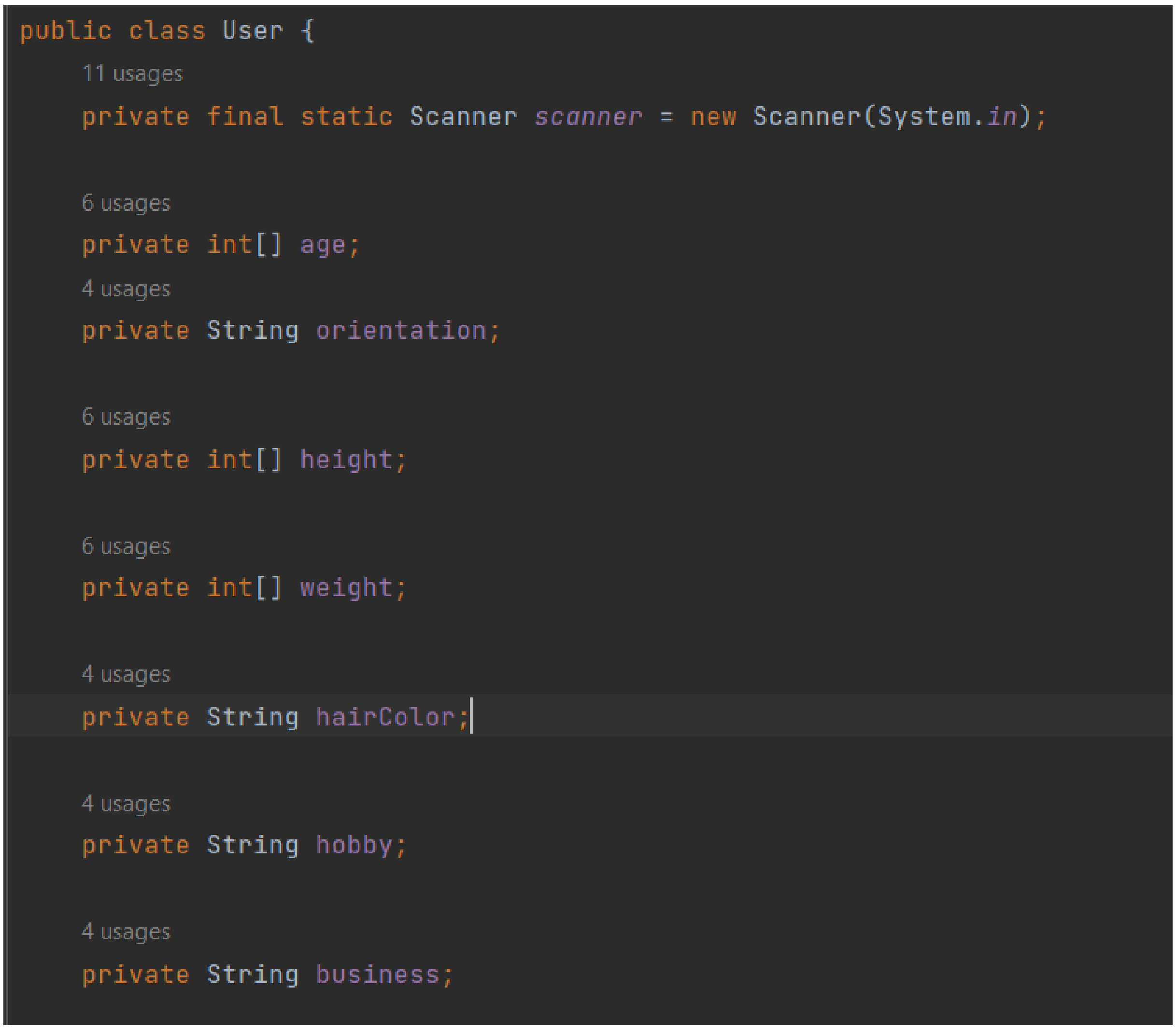
The **toString** method has been overwritten again and supplemented with information directly related to the fields in a specific class describing a specific candidate in order to present its characteristics in more detail.



```
RobertJanuszczyc
@Override
public String toString() {
    String one;
    String two;

    if (ownBusiness) {
        one = "I run my own business";
    } else {
        one = "I am full-time employee";
    }

    if (loveAnimals) {
        two = "I love animals";
    } else {
        two = "I don't like animals";
    }

    return super.toString() + "\n" + one + "\n" + two;
}
```

Figure 6 toString method in the class created for the first candidate

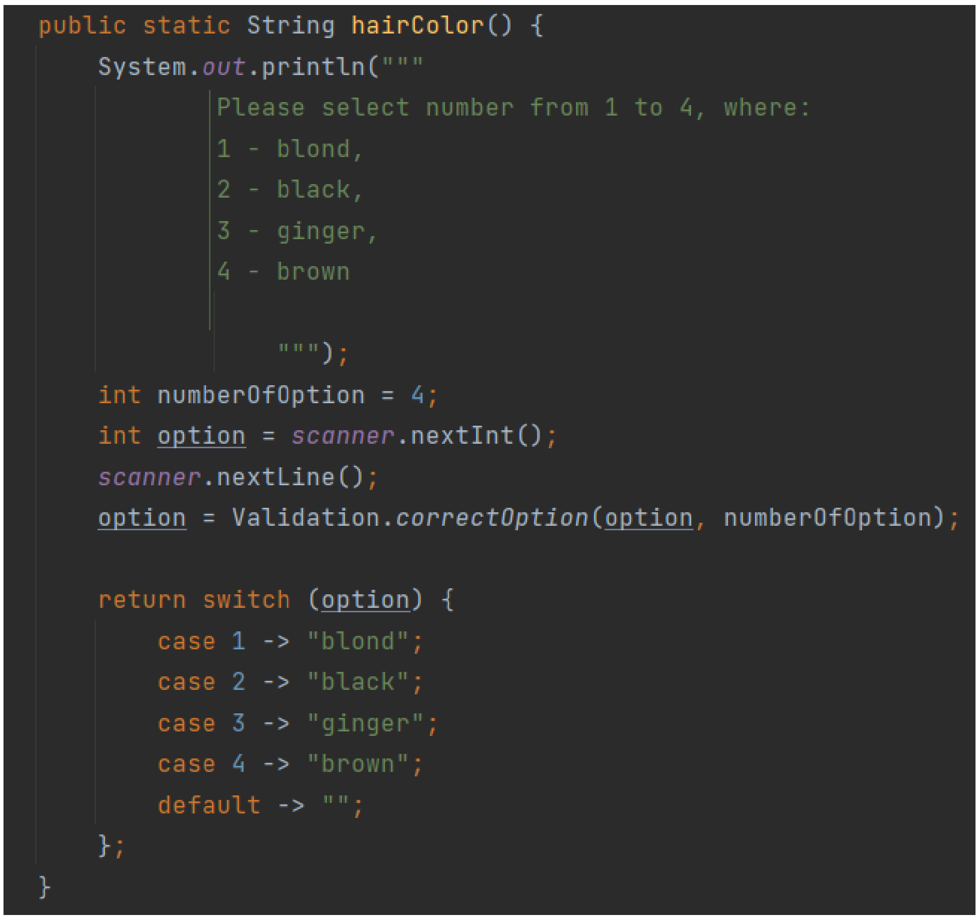
The next stage was to create the **User** class and fields to assign preferences to the user. Getters was created for fields as well.



```
public class User {
    11 usages
    private final static Scanner scanner = new Scanner(System.in);

    6 usages
    private int[] age;
    4 usages
    private String orientation;

    6 usages
    private int[] height;

    6 usages
    private int[] weight;

    4 usages
    private String hairColor;

    4 usages
    private String hobby;

    4 usages
    private String business;
```

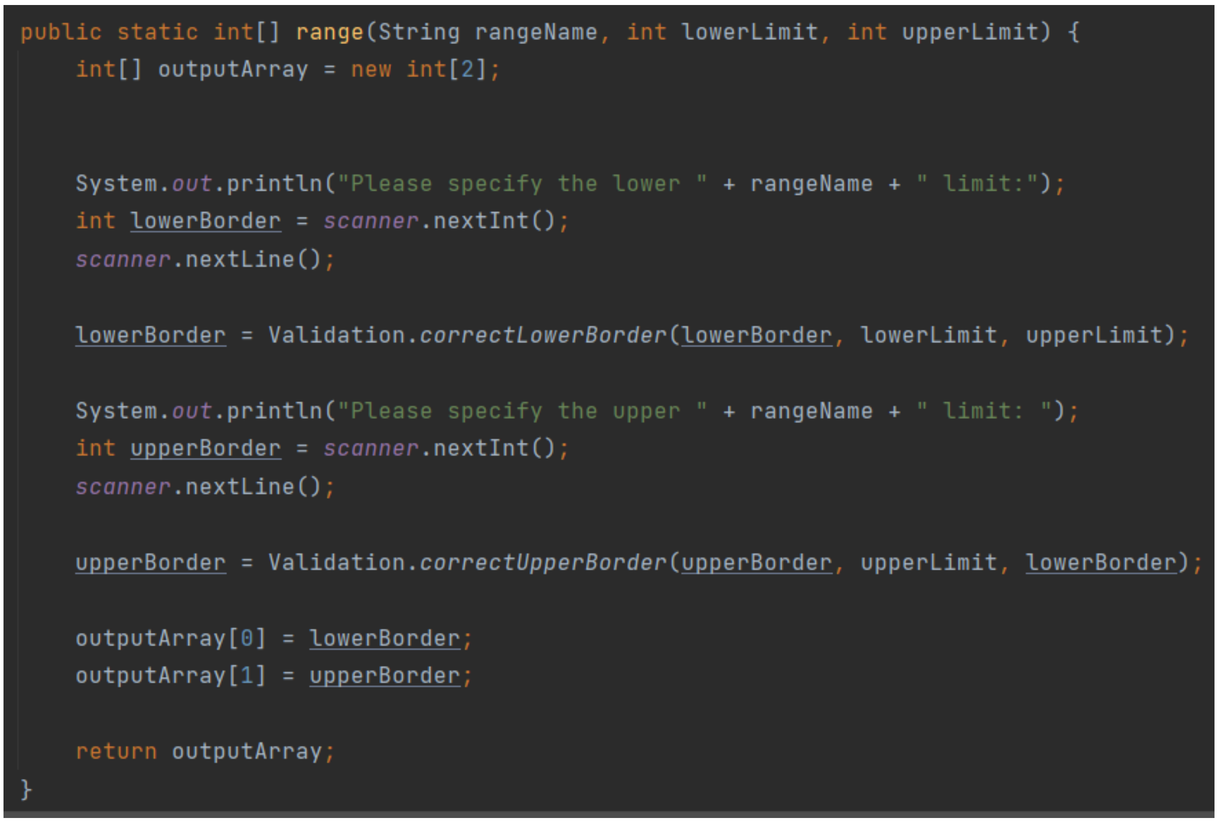
Figure 7 Fields and getters in the User class

Methods **orientation**, **range**, **hairColor**, **hobby** and **additionalPoints** were used to determine user preferences.



```java
public static String hairColor() {
    System.out.println("""
            Please select number from 1 to 4, where:
            1 - blond,
            2 - black,
            3 - ginger,
            4 - brown

                """);
    int numberOfOption = 4;
    int option = scanner.nextInt();
    scanner.nextLine();
    option = Validation.correctOption(option, numberOfOption);

    return switch (option) {
        case 1 -> "blond";
        case 2 -> "black";
        case 3 -> "ginger";
        case 4 -> "brown";
        default -> "";
    };
}
```

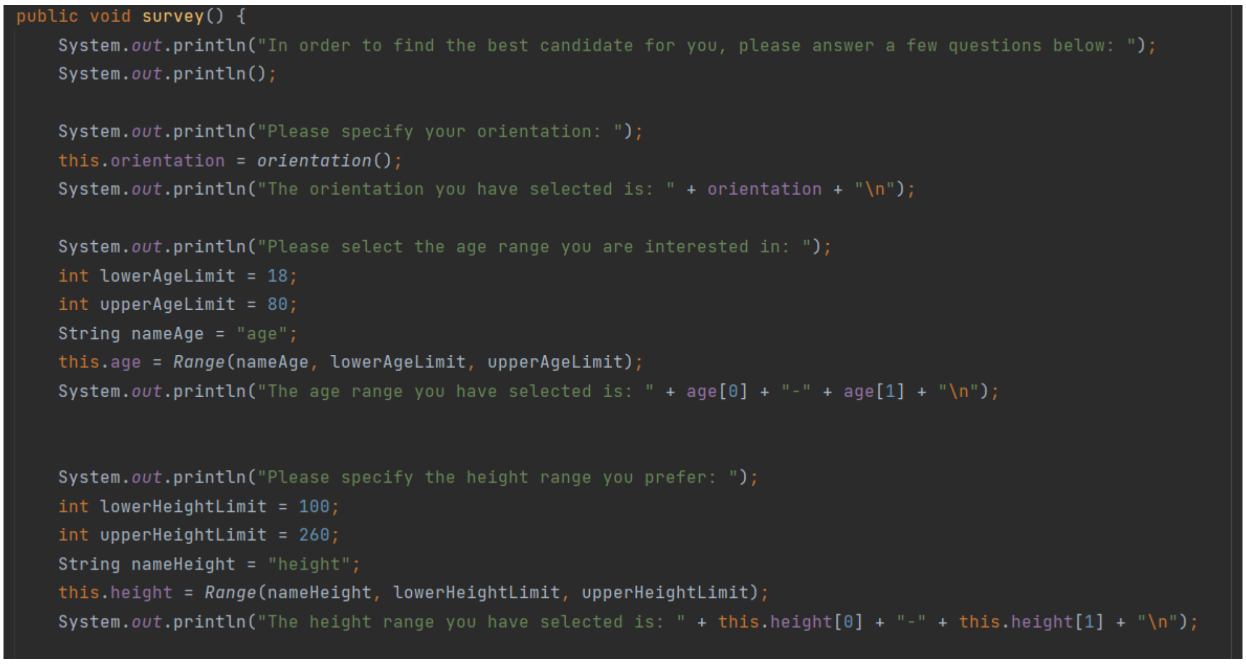
Figure 8 A sample hairColor method in the User class



```java
public static int[] range(String rangeName, int lowerLimit, int upperLimit) {
    int[] outputArray = new int[2];


    System.out.println("Please specify the lower " + rangeName + " limit:");
    int lowerBorder = scanner.nextInt();
    scanner.nextLine();

    lowerBorder = Validation.correctLowerBorder(lowerBorder, lowerLimit, upperLimit);

    System.out.println("Please specify the upper " + rangeName + " limit: ");
    int upperBorder = scanner.nextInt();
    scanner.nextLine();

    upperBorder = Validation.correctUpperBorder(upperBorder, upperLimit, lowerBorder);

    outputArray[0] = lowerBorder;
    outputArray[1] = upperBorder;

    return outputArray;
}
```

Figure 9 A sample range method in the User class

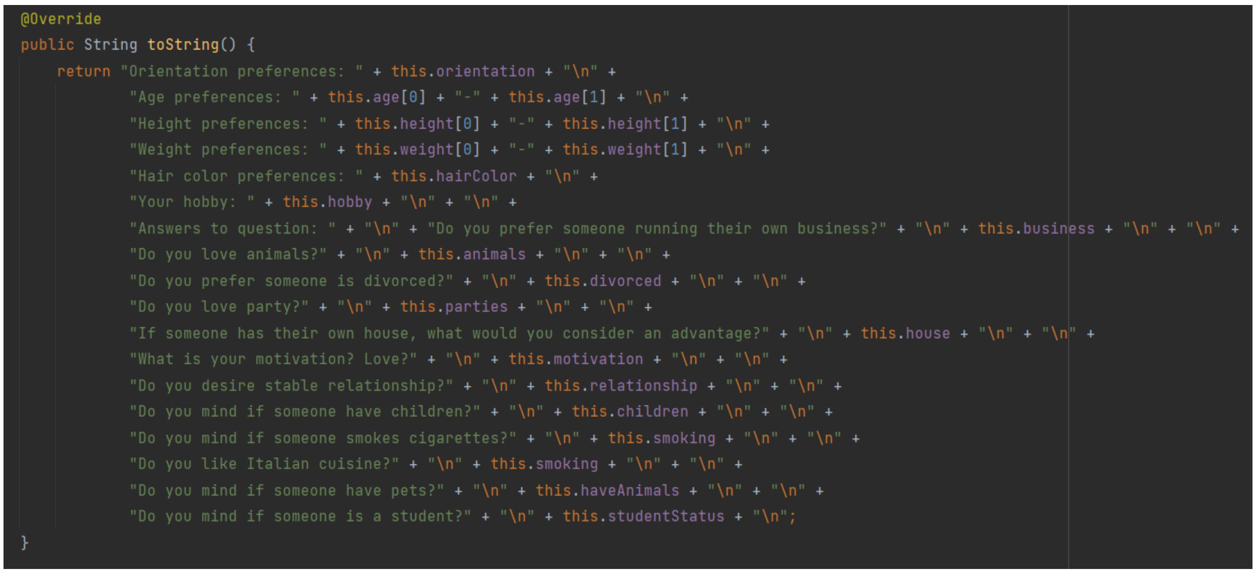
Preferences have been assigned to the fields of the **User** class by a survey method. The method survey allow us for better user understanding.



```java
public void survey() {
    System.out.println("In order to find the best candidate for you, please answer a few questions below: ");
    System.out.println();

    System.out.println("Please specify your orientation: ");
    this.orientation = orientation();
    System.out.println("The orientation you have selected is: " + orientation + "\n");

    System.out.println("Please select the age range you are interested in: ");
    int lowerAgeLimit = 18;
    int upperAgeLimit = 80;
    String nameAge = "age";
    this.age = Range(nameAge, lowerAgeLimit, upperAgeLimit);
    System.out.println("The age range you have selected is: " + age[0] + "-" + age[1] + "\n");


    System.out.println("Please specify the height range you prefer: ");
    int lowerHeightLimit = 100;
    int upperHeightLimit = 260;
    String nameHeight = "height";
    this.height = Range(nameHeight, lowerHeightLimit, upperHeightLimit);
    System.out.println("The height range you have selected is: " + this.height[0] + "-" + this.height[1] + "\n");
```

Figure 10 survey method in the User class

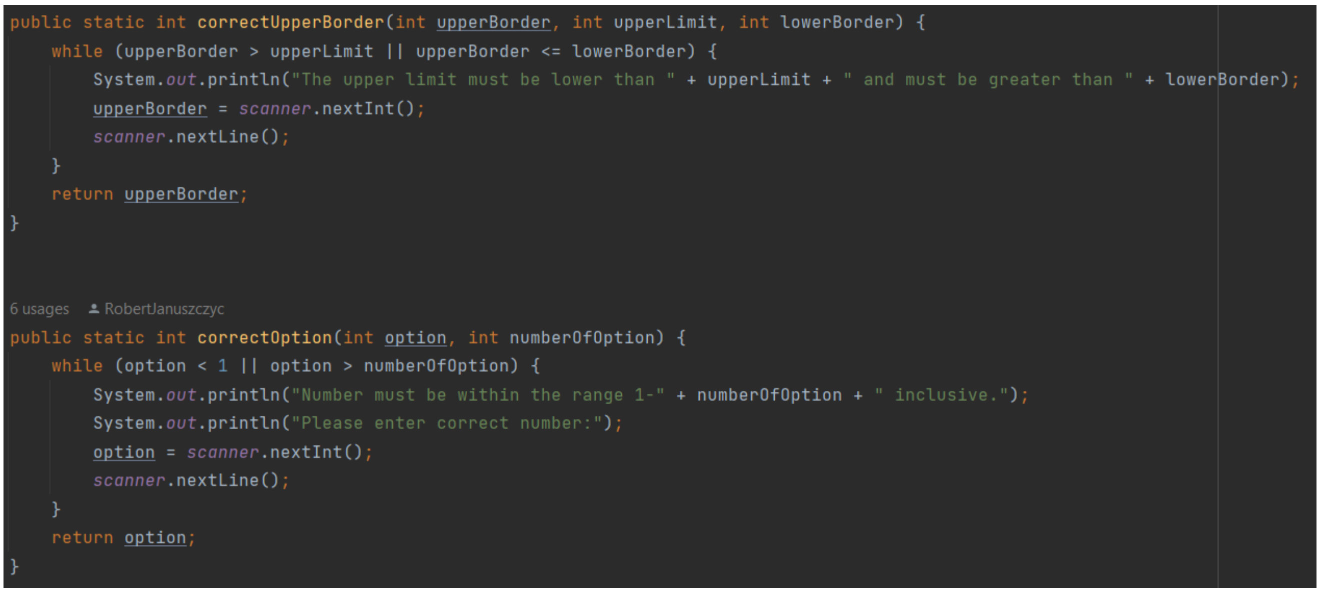
In the **User** class, there is also an overwritten **toString** method from the **Object** class.



```java
@Override
public String toString() {
    return "Orientation preferences: " + this.orientation + "\n" +
            "Age preferences: " + this.age[0] + "-" + this.age[1] + "\n" +
            "Height preferences: " + this.height[0] + "-" + this.height[1] + "\n" +
            "Weight preferences: " + this.weight[0] + "-" + this.weight[1] + "\n" +
            "Hair color preferences: " + this.hairColor + "\n" +
            "Your hobby: " + this.hobby + "\n" + "\n" +
            "Answers to question: " + "\n" + "Do you prefer someone running their own business?" + "\n" + this.business + "\n" + "\n" +
            "Do you love animals?" + "\n" + this.animals + "\n" + "\n" +
            "Do you prefer someone is divorced?" + "\n" + this.divorced + "\n" + "\n" +
            "Do you love party?" + "\n" + this.parties + "\n" + "\n" +
            "If someone has their own house, what would you consider an advantage?" + "\n" + this.house + "\n" + "\n" +
            "What is your motivation? Love?" + "\n" + this.motivation + "\n" + "\n" +
            "Do you desire stable relationship?" + "\n" + this.relationship + "\n" + "\n" +
            "Do you mind if someone have children?" + "\n" + this.children + "\n" + "\n" +
            "Do you mind if someone smokes cigarettes?" + "\n" + this.smoking + "\n" + "\n" +
            "Do you like Italian cuisine?" + "\n" + this.smoking + "\n" + "\n" +
            "Do you mind if someone have pets?" + "\n" + this.haveAnimals + "\n" + "\n" +
            "Do you mind if someone is a student?" + "\n" + this.studentStatus + "\n";
}
```

Figure 11 toString Method in the User class

Validation methods in a separate **Validation** class has been applied to methods such as **range**, **hobby** or **orientation**. These methods have been created to protect against possible user error or exceeding the assumed ranges.

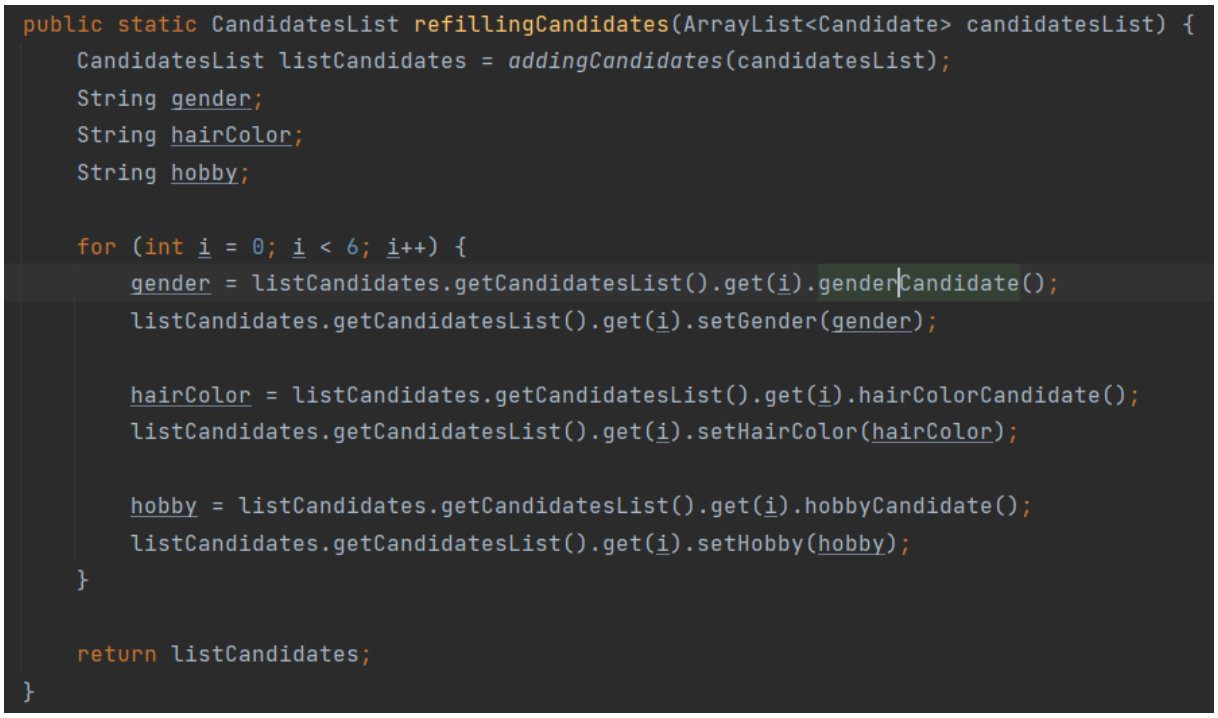


```java
public static int correctUpperBorder(int upperBorder, int upperLimit, int lowerBorder) {
    while (upperBorder > upperLimit || upperBorder <= lowerBorder) {
        System.out.println("The upper limit must be lower than " + upperLimit + " and must be greater than " + lowerBorder);
        upperBorder = scanner.nextInt();
        scanner.nextLine();
    }
    return upperBorder;
}


6 usages   RobertJanuszczyc
public static int correctOption(int option, int numberOfOption) {
    while (option < 1 || option > numberOfOption) {
        System.out.println("Number must be within the range 1-" + numberOfOption + " inclusive.");
        System.out.println("Please enter correct number:");
        option = scanner.nextInt();
        scanner.nextLine();
    }
    return option;
}
```

Figure 12 Validation methods in the Validation class

The most important methods in the **Main** class, which also contains the **main** method, include:

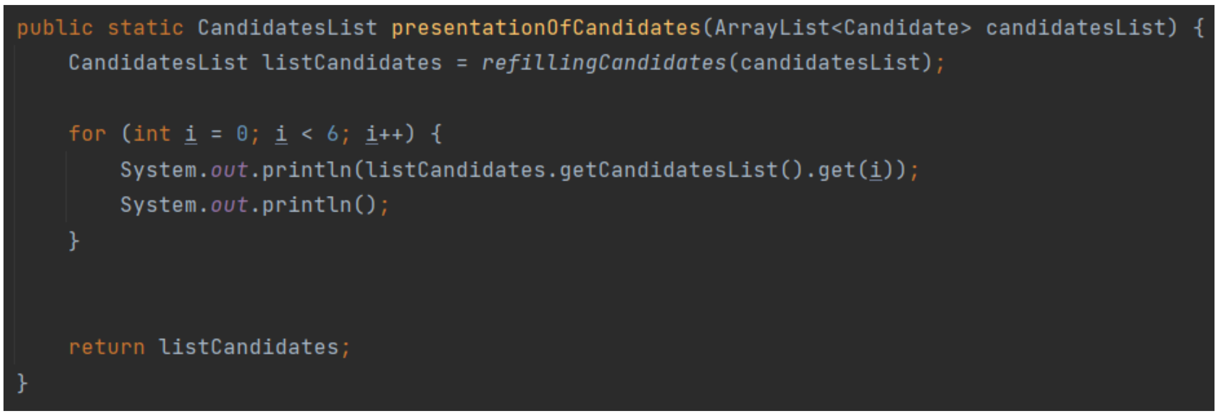
refillingCandidates → A method to complete data on candidates in terms of gender, hobbies and hair color.



```java
public static CandidatesList refillingCandidates(ArrayList<Candidate> candidatesList) {
    CandidatesList listCandidates = addingCandidates(candidatesList);
    String gender;
    String hairColor;
    String hobby;

    for (int i = 0; i < 6; i++) {
        gender = listCandidates.getCandidatesList().get(i).genderCandidate();
        listCandidates.getCandidatesList().get(i).setGender(gender);

        hairColor = listCandidates.getCandidatesList().get(i).hairColorCandidate();
        listCandidates.getCandidatesList().get(i).setHairColor(hairColor);

        hobby = listCandidates.getCandidatesList().get(i).hobbyCandidate();
        listCandidates.getCandidatesList().get(i).setHobby(hobby);
    }

    return listCandidates;
}
```

Figure 13 refillingCandidates method in Main class

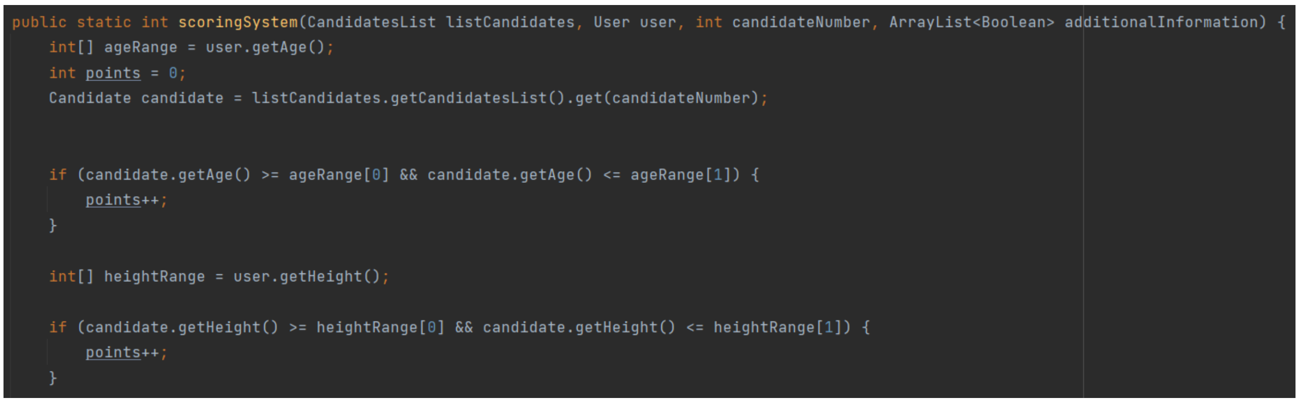
presentationOfCandidates → A method of presenting all candidates.



```java
public static CandidatesList presentationOfCandidates(ArrayList<Candidate> candidatesList) {
    CandidatesList listCandidates = refillingCandidates(candidatesList);

    for (int i = 0; i < 6; i++) {
        System.out.println(listCandidates.getCandidatesList().get(i));
        System.out.println();
    }


    return listCandidates;
}
```

Figure 14 presentationOfCandidates method in Main class

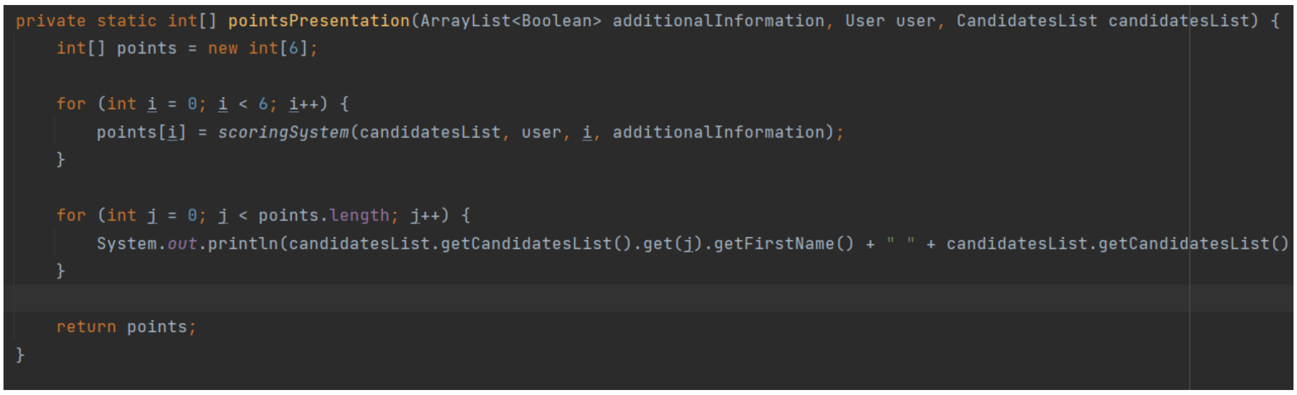
scoringSystem→ A method that assigns points to each candidate based on matching user preferences with candidate characteristics.



```java
public static int scoringSystem(CandidatesList listCandidates, User user, int candidateNumber, ArrayList<Boolean> additionalInformation) {
    int[] ageRange = user.getAge();
    int points = 0;
    Candidate candidate = listCandidates.getCandidatesList().get(candidateNumber);


    if (candidate.getAge() >= ageRange[0] && candidate.getAge() <= ageRange[1]) {
        points++;
    }

    int[] heightRange = user.getHeight();

    if (candidate.getHeight() >= heightRange[0] && candidate.getHeight() <= heightRange[1]) {
        points++;
    }
```

Figure 15 scoringSystem method in Main class

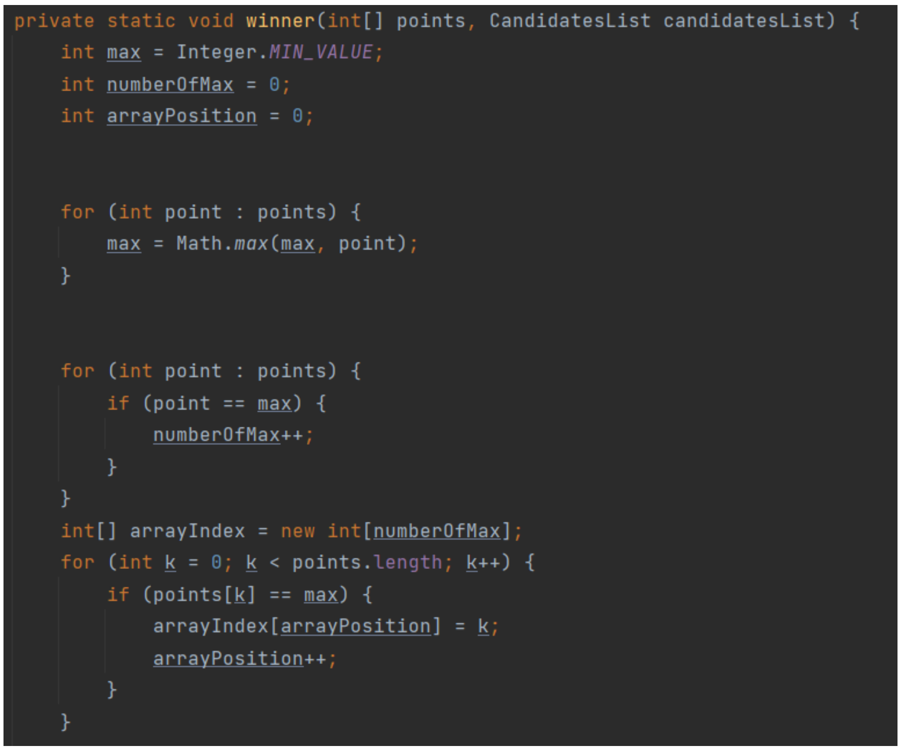
pointPresentation→ A method that presents the number of points that were assigned to each candidate.



```java
private static int[] pointsPresentation(ArrayList<Boolean> additionalInformation, User user, CandidatesList candidatesList) {
    int[] points = new int[6];

    for (int i = 0; i < 6; i++) {
        points[i] = scoringSystem(candidatesList, user, i, additionalInformation);
    }

    for (int j = 0; j < points.length; j++) {
        System.out.println(candidatesList.getCandidatesList().get(j).getFirstName() + " " + candidatesList.getCandidatesList()
    }

    return points;
}
```

Figure 16 pointPresentation method in Main class

winner → A method that determines the winner(s) based on points earned.



```java
private static void winner(int[] points, CandidatesList candidatesList) {
    int max = Integer.MIN_VALUE;
    int numberOfMax = 0;
    int arrayPosition = 0;


    for (int point : points) {
        max = Math.max(max, point);
    }


    for (int point : points) {
        if (point == max) {
            numberOfMax++;
        }
    }
    int[] arrayIndex = new int[numberOfMax];
    for (int k = 0; k < points.length; k++) {
        if (points[k] == max) {
            arrayIndex[arrayPosition] = k;
            arrayPosition++;
        }
    }
```

Figure 17 winner method in Main class

Running application

Prerequisite

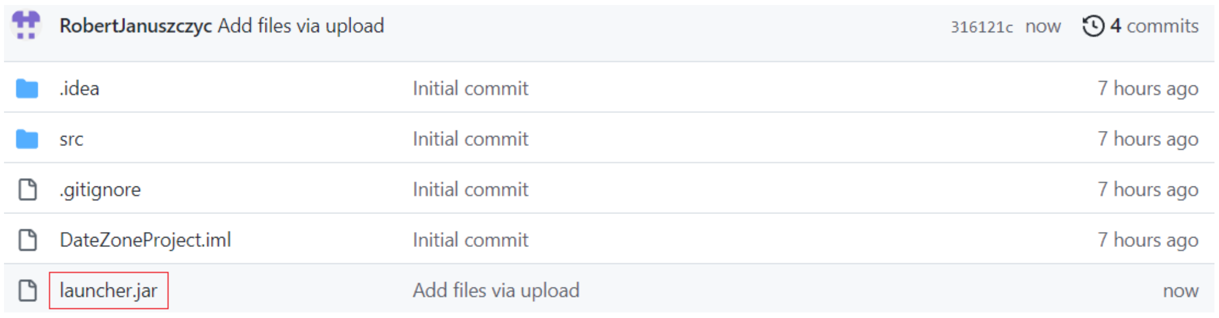
if you want to run the application java 8 or later is required.

To download, follow the link below:

https://www.java.com/pl/

Launch process

To launch the application, download a file named **launcher.jar** from the **DateZoneProject** repository located on **GitHub**.



RobertJanuszczyc Add files via upload		316121c now 4 commits
.idea	Initial commit	7 hours ago
src	Initial commit	7 hours ago
.gitignore	Initial commit	7 hours ago
DateZoneProject.iml	Initial commit	7 hours ago
launcher.jar	Add files via upload	now

Figure 1 File location

Then save it to a specific location on your computer.

On my computer, the file is located in:

C:\Users\Robert\IdeaProjects\DateZoneProject\src

Open the console by typing cmd at the command line.

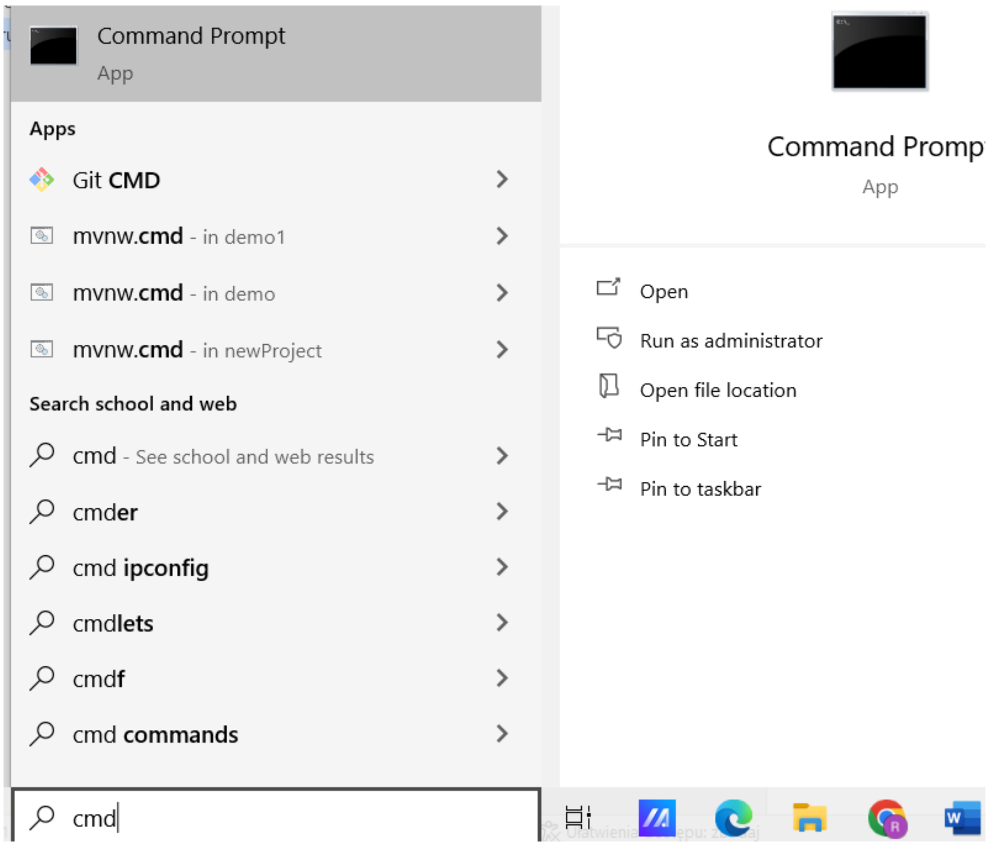
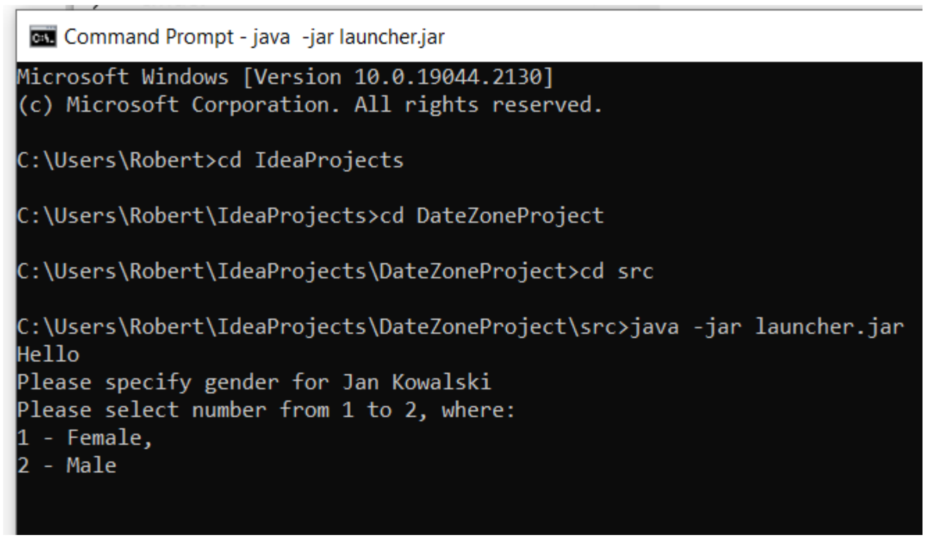


Figure 2 start the console

In the next step, type the following command in the console:

java -jar launcher.jar

After typing the command, the program should start in the console.



Command Prompt - java -jar launcher.jar

```
Microsoft Windows [Version 10.0.19044.2130]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Robert>cd IdeaProjects

C:\Users\Robert\IdeaProjects>cd DateZoneProject

C:\Users\Robert\IdeaProjects\DateZoneProject>cd src

C:\Users\Robert\IdeaProjects\DateZoneProject\src>java -jar launcher.jar
Hello
Please specify gender for Jan Kowalski
Please select number from 1 to 2, where:
1 - Female,
2 - Male
```

Figure 3 launch of the application