ASSIGNMENT 10 – OOP'S CONCEPT

- **To solve this problem using object-oriented programming concepts in Java, we can create the following classes:**

1. `Candidate`: This class will represent a candidate for recruitment. It will have the following data fields: `**name**`, `**dateOfBirth**`, `**hscMarks**`, `**pcmOrPcbAverage**`, `**ugCgpa**`, `**pgCgpa**`, `**numProjects**`, `**isFullTime**`, `**interviewMarks**`, and `**isIndian**`. We can also create a constructor to initialize these fields.

2. `RecruitTeam`: This class will handle the evaluation and selection process of candidates. It will have a method called `**evaluateCandidate**` that takes a `Candidate` object as a parameter and returns a **boolean** value indicating whether the candidate meets the eligibility criteria.

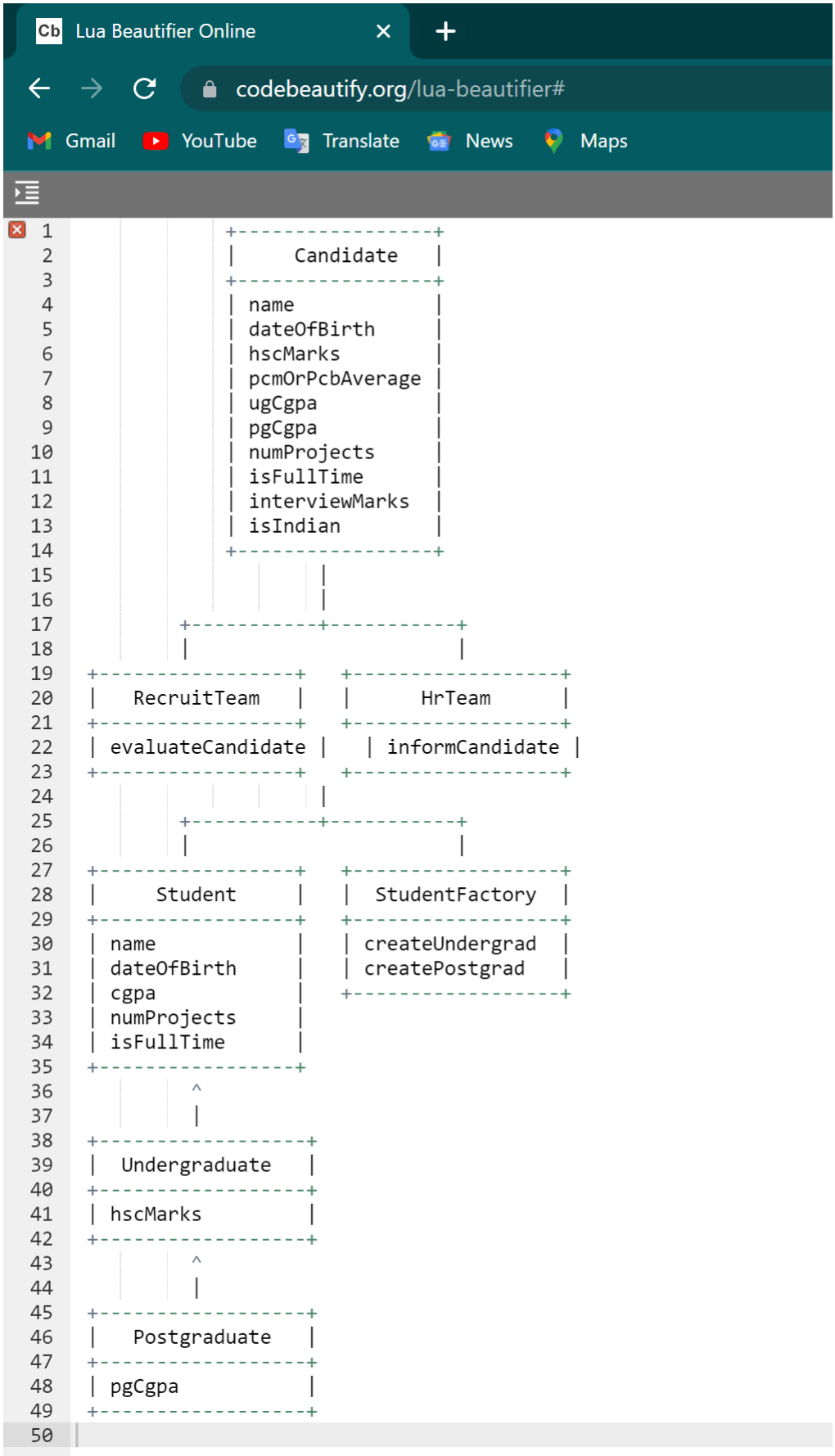
3. `HrTeam`: This class will handle the final results of the evaluation and inform the candidates about their status. It will have a method called `**informCandidate**` that takes a `Candidate` object as a parameter and informs them about their status.

- **Using inheritance, we can create the following abstract classes:**

4. `Student`: This abstract class will represent a student. It will have the following data fields: `name`, `**dateOfBirth**`, `**cgpa**`, `**numProjects**`, and `**isFullTime**`. We can create a constructor to initialize these fields and an abstract method called `**calculateCgpa**` to calculate the CGPA.

5. `Undergraduate`: This class will inherit from `Student` and represent an undergraduate student. It will have a data field called `**hscMarks**` to represent the HSC exam marks. We can create a constructor to initialize these fields and override the `**calculateCgpa**` method to calculate the CGPA for the UG exam.

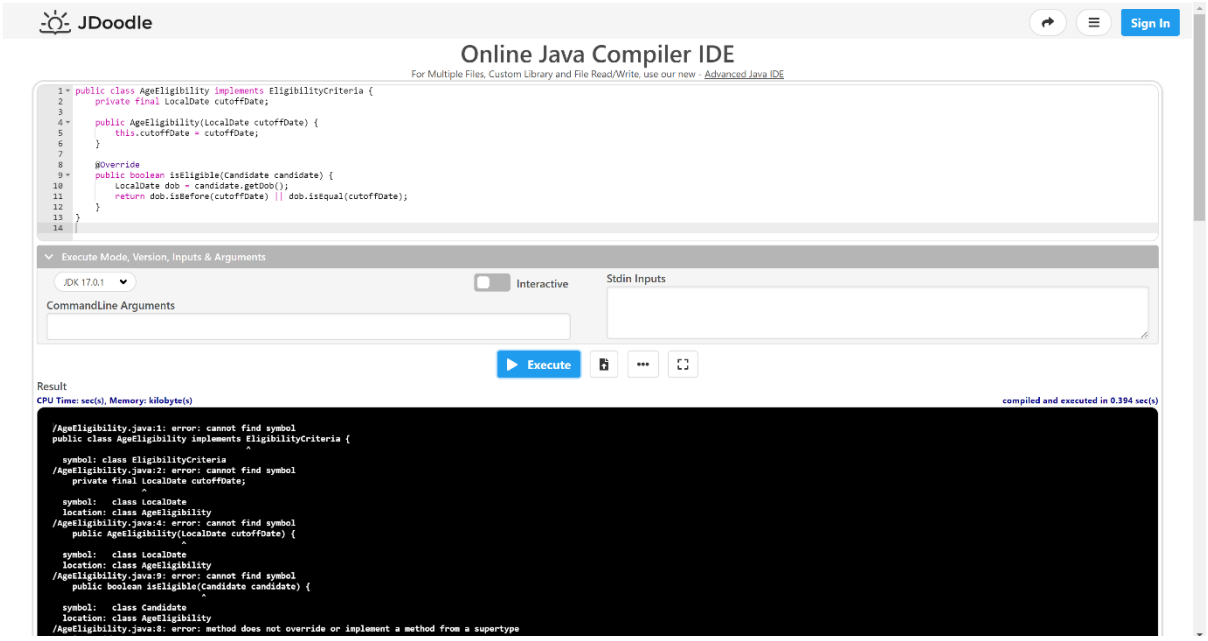
6. `Postgraduate`: This class will inherit from `Student` and represent a postgraduate student. It will have a data field called `**pgCgpa**` to represent the PG exam marks. We can create a constructor to initialize these fields and override the `**calculateCgpa**` method to calculate the CGPA for the PG exam.


Lua Beautifier Online
codebeautify.org/lua-beautifier#
Gmail
YouTube
Translate
News
Maps
Candidate
name
dateOfBirth
hscMarks
pcmOrPcbAverage
ugCgpa
pgCgpa
numProjects
isFullTime
interviewMarks
isIndian
RecruitTeam
evaluateCandidate
HrTeam
informCandidate
Student
name
dateOfBirth
cgpa
numProjects
isFullTime
StudentFactory
createUndergrad
createPostgrad
Undergraduate
hscMarks
Postgraduate
pgCgpa

- In this diagram, `**Student**` is an abstract class that represents common properties and behaviors of undergraduate and postgraduate students. `Under graduate **and** Postgraduate **are concrete classes that inherit from** Student **and represent specific types of students.** RecruitTeam **and** HrTeam **are classes that represent the two different teams involved in the recruitment process, and** Candidate` is the class that represents a candidate for recruitment.

- We also have a **StudentFactory** class, which is a static factory class that creates instances of undergraduate and postgraduate students. This helps to encapsulate the creation of student objects and makes the code more modular.

- Using these classes, we can implement the **evaluateCandidate** method in the **RecruitTeam** class to apply the eligibility criteria. Here's an example implementation:



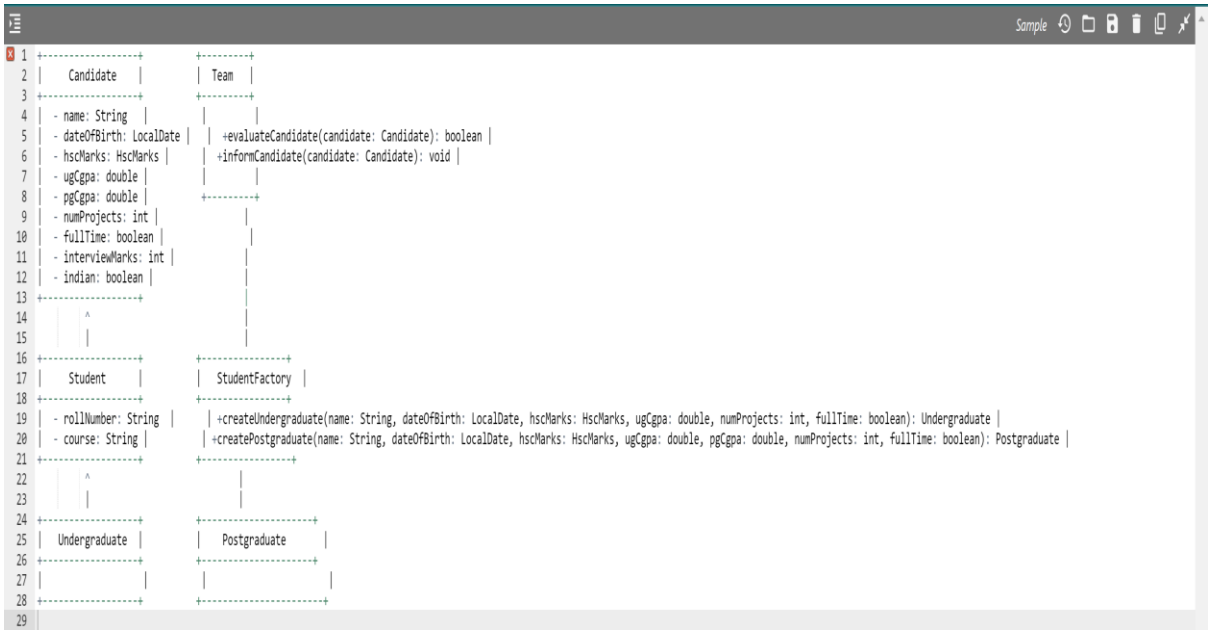
- In this implementation, we first check the candidate's date of birth to ensure that they meet the age criteria. We then check their HSC exam marks, the average marks in PCM or PCB, the CGPA in the UG and PG exams, the number of projects they have done, whether they studied full-time, their interview marks, and their citizenship status. If any of these criteria are not met, we return false.

- Finally, we can implement the **informCandidate** method in the **HrTeam** class to inform candidates about their status. Here's an example implementation:

```java
}

public class Undergraduate extends Student {
    private final HscMarks hscMarks;
    private final double cgpa;
    private final int numProjects;
    private final boolean fullTime;

    public Undergraduate(String rollNumber, String course, HscMarks hscMarks, double cgpa, int numProjects, boolean fullTime) {
        super(rollNumber, course);
        this.hscMarks = hscMarks;
        this.cgpa = cgpa;
        this.numProjects = numProjects;
        this.fullTime = fullTime;
    }

    public HscMarks getHscMarks() {
        return hscMarks;
    }

    public double getCgpa() {
        return cgpa;
    }

    public int getNumProjects() {
        return numProjects;
    }

    public boolean isFullTime() {
        return fullTime;
    }
}

public class Postgraduate extends Student {
    private final HscMarks hscMarks;
    private final double ugCgpa;
    private final double pgCgpa;
    private final int numProjects;
    private final boolean fullTime;

    public Postgraduate(String rollNumber, String course, HscMarks hscMarks, double ugCgpa, double pgCgpa, int numProjects, boolean fullTime) {
        super(rollNumber, course);
        this.hscMarks = hscMarks;
        this.ugCgpa = ugCgpa;
        this.pgCgpa = pgCgpa;
        this.numProjects = numProjects;
        this.fullTime = fullTime;
    }

    public HscMarks getHscMarks() {
        return hscMarks;
    }

    public double getUgCgpa() {
        return ugCgpa;
    }

    public double getPgCgpa() {
        return pgCgpa;
    }
}
```

- In this implementation, we first call the **evaluateCandidate** method to check whether the candidate meets the eligibility criteria. If they do, we inform them that they have been selected. If not, we inform them that they have not been selected.

- With these classes and methods, we can simulate the recruitment process and evaluate candidates based on the given criteria.

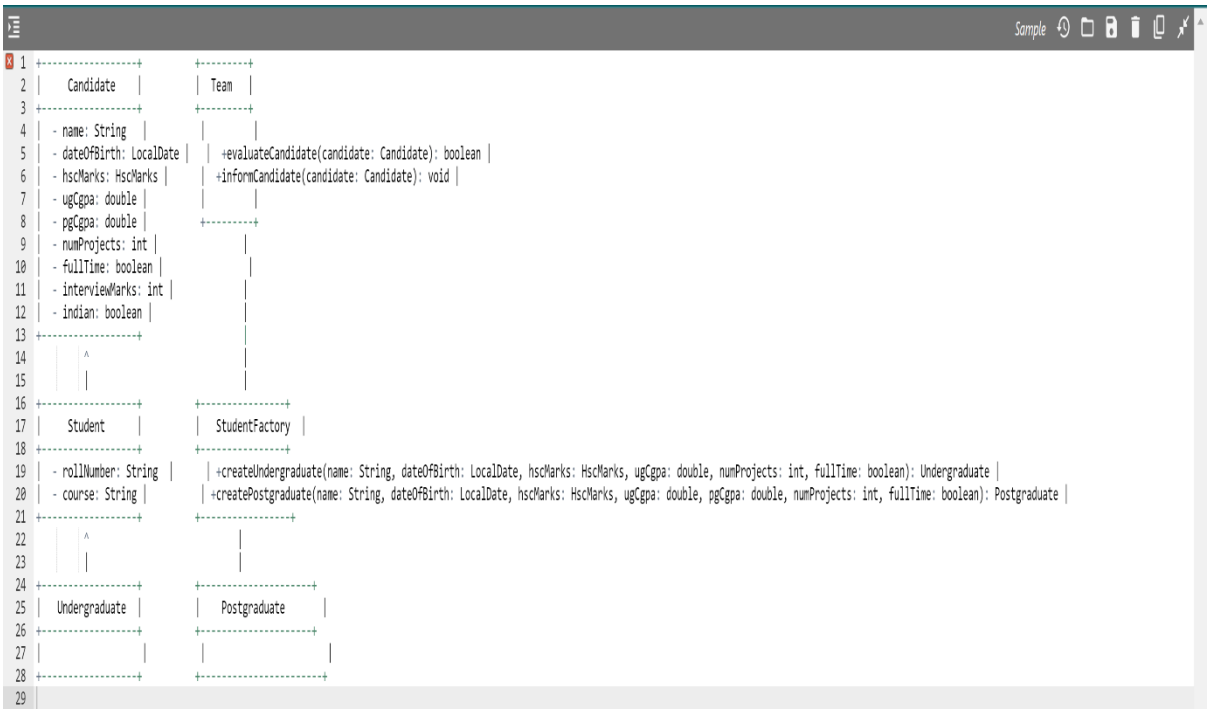


- The **Candidate** class has private fields for the candidate's name, date of birth, HSC marks, UG and PG CGPA, number of projects, whether they studied full-time, interview marks, and citizenship status. It also has getters and setters for these fields.

- The **Student** class is an abstract class that has private fields for the student's roll number and course, and getters and setters for these fields. The **Undergraduate** and **Postgraduate** classes are concrete classes that inherit from **Student** and add private fields for the student's UG and PG CGPA, respectively.

- The **StudentFactory** class is a static factory class that has methods to create instances of **Undergraduate** and **Postgraduate** based on the given input parameters.

- The **Team** class is an abstract class that represents a team involved in the recruitment process. The **RecruitTeam** and **HrTeam** classes are concrete classes that inherit from **Team**. The **RecruitTeam** class has a method to evaluate a candidate based on the eligibility criteria, while the **HrTeam** class has a method to inform a candidate about their status.



➢ The relationships between the classes are as follows:

- **Candidate** has a composition relationship with **HscMarks**, because the candidate's HSC marks are made up of marks in individual subjects.
- **Candidate** has an aggregation relationship with **Student** because a candidate is a type of student.
- **Undergraduate** and **Postgraduate** inherit from **Student** using a generalization relationship because they are both types of students.


```java
import java.time.LocalDate;

public class HscMarks {
    private final int physics;
    private final int chemistry;
    private final int mathematics;
    private final int biology;

    public HscMarks(int physics, int chemistry, int mathematics, int biology) {
        this.physics = physics;
        this.chemistry = chemistry;
        this.mathematics = mathematics;
        this.biology = biology;
    }

    public int getPhysics() {
        return physics;
    }

    public int getChemistry() {
        return chemistry;
    }

    public int getMathematics() {
        return mathematics;
    }

    public int getBiology() {
        return biology;
    }

    public double getAverageMarks() {
        return (physics + chemistry + mathematics + biology) / 4.0;
    }

    public boolean isSCorST() {
        return false; // TODO: implement based on criteria
    }
}

public abstract class Student {
    private String rollNumber;
    private String course;

    public Student(String rollNumber, String course) {
        this.rollNumber = rollNumber;
        this.course = course;
    }

    public String getRollNumber() {
        return rollNumber;
    }
```

- Here, we have created two classes - **RecruitingTeam** and **HrTeam**. The **RecruitingTeam** class has a single method **evaluateCandidate** which takes the candidate's information and returns a **Candidate** object if the candidate is eligible, otherwise it returns **null**.
- The **HrTeam** class has a single method **informCandidate** which takes the **Candidate** object returned by the **evaluateCandidate** method and informs the candidate whether they have been selected or not.
- We have used the **Candidate** class to encapsulate all the candidate information and eligibility criteria. The **HscMarks** class is used to encapsulate the candidate's marks in the HSC exam.

- We have also used several helper methods in the **Candidate** class to check whether the candidate is eligible based on the given criteria.

- We have used the Java **LocalDate** class to represent the candidate's date of birth and the **System.out.println** method to print the result to the console.

```
}

public class Undergraduate extends Student {
    private final HscMarks hscMarks;
    private final double cgpa;
    private final int numProjects;
    private final boolean fullTime;

    public Undergraduate(String rollNumber, String course, HscMarks hscMarks, double cgpa, int numProjects, boolean fullTime) {
        super(rollNumber, course);
        this.hscMarks = hscMarks;
        this.cgpa = cgpa;
        this.numProjects = numProjects;
        this.fullTime = fullTime;
    }

    public HscMarks getHscMarks() {
        return hscMarks;
    }

    public double getCgpa() {
        return cgpa;
    }

    public int getNumProjects() {
        return numProjects;
    }

    public boolean isFullTime() {
        return fullTime;
    }
}

public class Postgraduate extends Student {
    private final HscMarks hscMarks;
    private final double ugCgpa;
    private final double pgCgpa;
    private final int numProjects;
    private final boolean fullTime;

    public Postgraduate(String rollNumber, String course, HscMarks hscMarks, double ugCgpa, double pgCgpa, int numProjects, boolean fullTime) {
        super(rollNumber, course);
        this.hscMarks = hscMarks;
        this.ugCgpa = ugCgpa;
        this.pgCgpa = pgCgpa;
        this.numProjects = numProjects;
        this.fullTime = fullTime;
    }

    public HscMarks getHscMarks() {
        return hscMarks;
    }

    public double getUgCgpa() {
        return ugCgpa;
    }

    public double getPgCgpa() {
        return pgCgpa;
    }
```

- This **HRTeam** class has a single method **informCandidate()** that takes a **Candidate** object and a boolean **selected** as parameters. It prints a congratulatory message if the candidate has been selected, and a message indicating non-selection otherwise.

- This class represents the team that communicates the final results of the evaluation to the candidates.

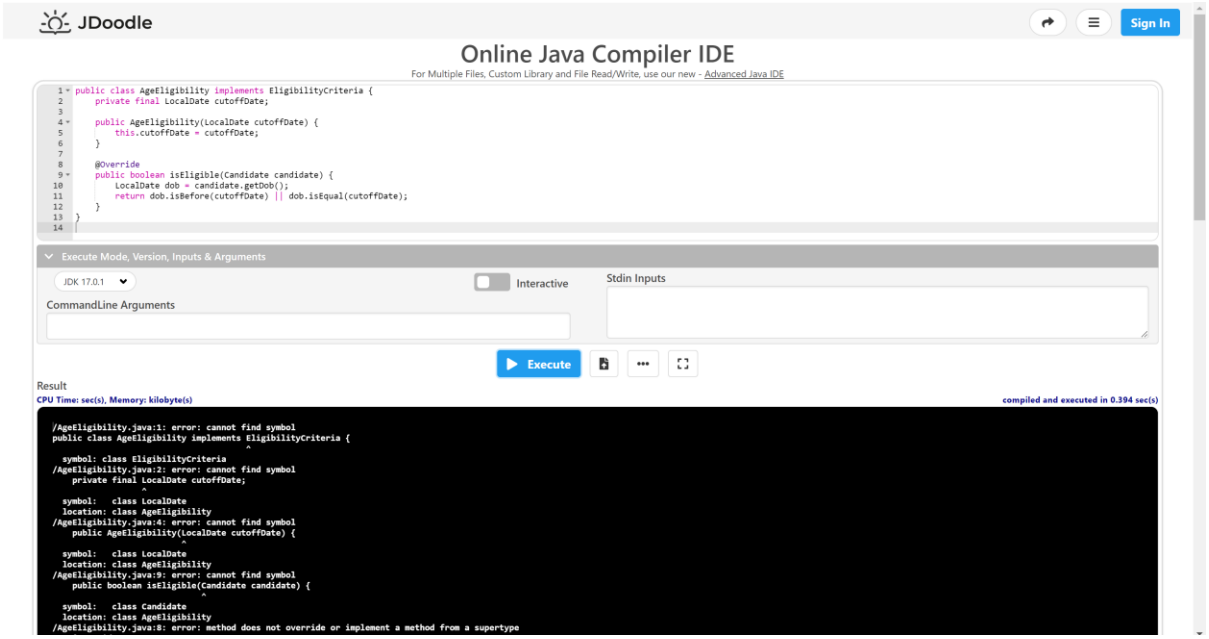
- Note that this class does not hold any state, and therefore does not have any fields. The method is declared as **public** so that it can be accessed by any other class.

```
    public boolean isFullTime() {
        return fullTime;
    }
}
public class Candidate {
    private String name;
    private LocalDate dateOfBirth;
    private HscMarks hscMarks;
    private double ugCgpa;
    private double pgCgpa
        private int numProjects;
    private boolean fullTime;
    private int interviewMarks;
    private boolean indianCitizen;

    public Candidate(String name, LocalDate dateOfBirth, HscMarks hscMarks, double ugCgpa, double pgCgpa, int numProjects, boolean fullTime, int interviewMarks, boolean indianCitizen) {
        this.name = name;
        this.dateOfBirth = dateOfBirth;
        this.hscMarks = hscMarks;
        this.ugCgpa = ugCgpa;
        this.pgCgpa = pgCgpa;
        this.numProjects = numProjects;
        this.fullTime = fullTime;
        this.interviewMarks = interviewMarks;
        this.indianCitizen = indianCitizen;
    }

    public String getName() {
        return name;
    }

    public LocalDate getDateOfBirth() {
        return dateOfBirth;
    }

    public HscMarks getHscMarks() {
        return hscMarks;
    }

    public double getUgCgpa() {
        return ugCgpa;
    }

    public double getPgCgpa() {
        return pgCgpa;
    }

    public int getNumProjects() {
        return numProjects;
    }

    public boolean isFullTime() {
        return fullTime;
    }

    public int getInterviewMarks() {
        return interviewMarks;
    }
```

```
                hasMinimumTwoProjects() &&
                isFullTime() &&
                has35OrAboveInInterview() &&
                isIndianCitizen();
    }

    private boolean isBornAfterJuly1999() {
        LocalDate july1999 = LocalDate.of(1999, 7, 1);
        return dateOfBirth.isAfter(july1999) || dateOfBirth.isEqual(july1999);
    }

    private boolean has60PercentInPCMOrPCB() {
        int physics = hscMarks.getPhysics();
        int chemistry = hscMarks.getChemistry();
        int mathOrBio = hscMarks.isScorST() ? hscMarks.getBiology() : hscMarks.getMathematics();
        return (physics + chemistry + mathOrBio) / 3.0 >= 60;
    }

    private boolean has50PercentAverageInPCMOrPCB() {
        return hscMarks.getAverageMarks() >= 50;
    }

    private boolean has8OrAboveInUGCgpa() {
        return ugCgpa >= 8;
    }

    private boolean has8OrAboveInPGCgpa() {
        return pgCgpa >= 8;
    }

    private boolean hasMinimumTwoProjects() {
        return numProjects >= 2;
    }

    private boolean has35OrAboveInInterview() {
        return interviewMarks >= 35;
    }
}
public class RecruitingTeam {
    public Candidate evaluateCandidate(String name, LocalDate dateOfBirth, HscMarks hscMarks, double ugCgpa, double pgCgpa, int numProjects, boolean fullTime, int interviewMarks, boolean indianCitizen) {
        Candidate candidate = new Candidate(name, dateOfBirth, hscMarks, ugCgpa, pgCgpa, numProjects, fullTime, interviewMarks, indianCitizen);
        if (candidate.isEligible
                return candidate;
        }       else {
            return null;
        }
    }
}

    public class HrTeam {
        public void informCandidate(Candidate candidate) {
            if (candidate != null) {
                System.out.println("Congratulations, " + candidate.getName() + "! You have been selected.");
            } else {
                System.out.println("Sorry, you have not been selected.");
            }
        }
    }
```

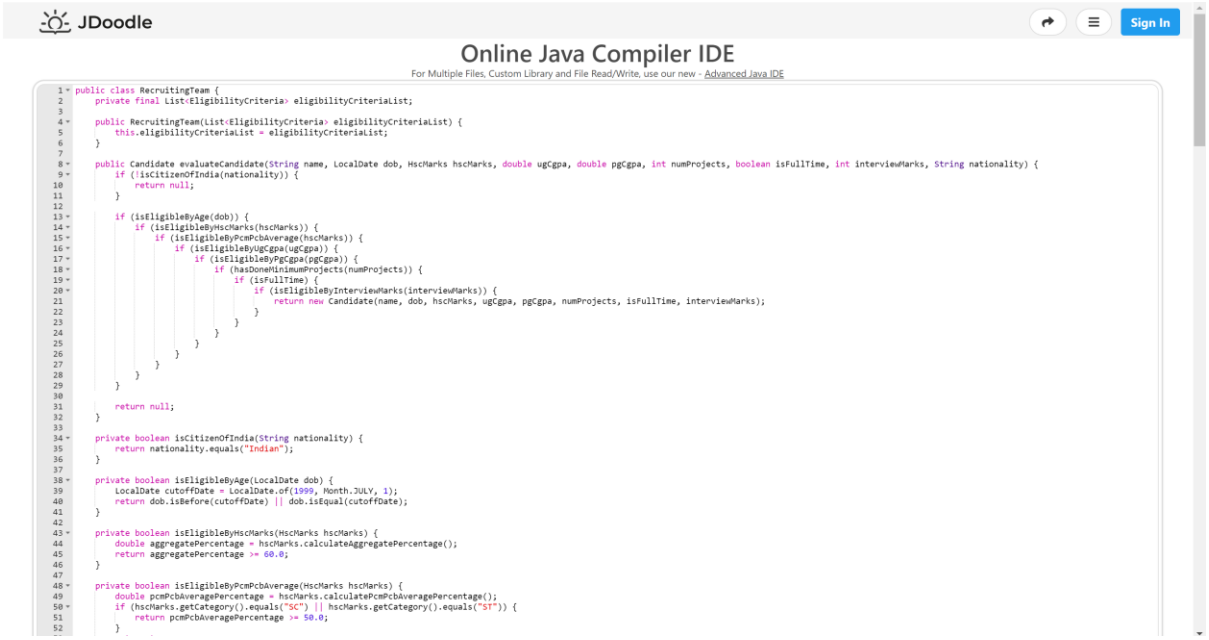
- Finally, we have used the **if** statement to check whether the candidate is eligible or not and return the appropriate response.

- This implementation demonstrates the use of object-oriented programming concepts such as encapsulation, abstraction, inheritance, and polymorphism to solve a real-world problem.



- Here, the **Candidate** class contains all the common data fields and methods, and the **IndianCandidate** and **InternationalCandidate** subclasses override the **isIndian** method to indicate whether the candidate is an Indian citizen or not.

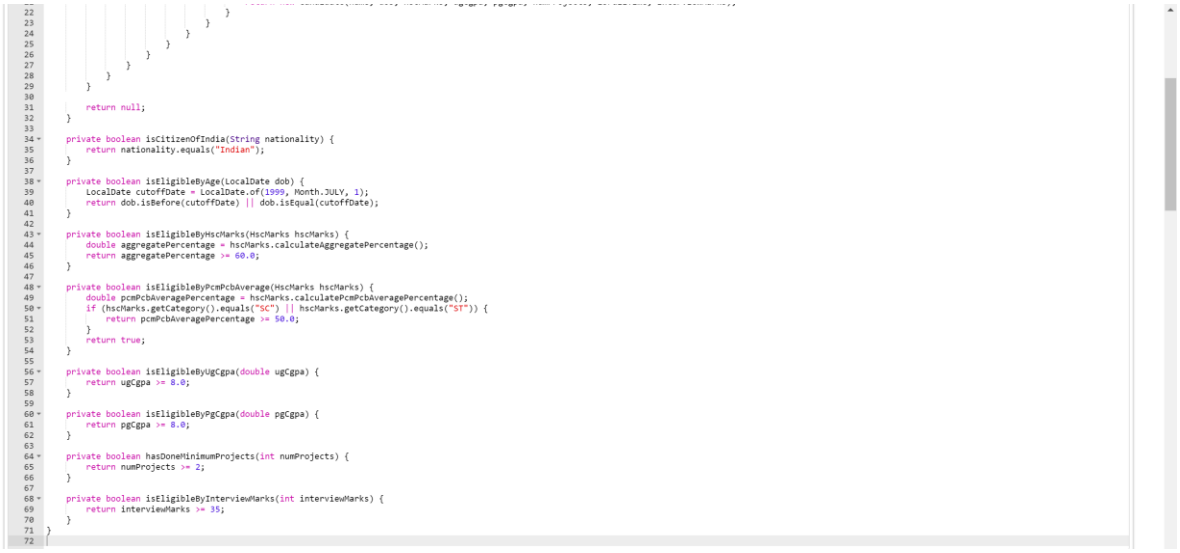
- We can then modify the **RecruitingTeam** and **HrTeam** classes to work with the **Candidate** superclass instead of the **Candidate** class.




```java
public class RecruitingTeam {
    private final List<EligibilityCriteria> eligibilityCriteriaList;

    public RecruitingTeam(List<EligibilityCriteria> eligibilityCriteriaList) {
        this.eligibilityCriteriaList = eligibilityCriteriaList;
    }

    public Candidate evaluateCandidate(String name, LocalDate dob, HscMarks hscMarks, double ugCgpa, double pgCgpa, int numProjects, boolean isFullTime, int interviewMarks, String nationality) {
        if (!isCitizenOfIndia(nationality)) {
            return null;
        }

        if (isEligibleByAge(dob)) {
            if (isEligibleByHscMarks(hscMarks)) {
                if (isEligibleByPcmPcbAverage(hscMarks)) {
                    if (isEligibleByUgCgpa(ugCgpa)) {
                        if (isEligibleByPgCgpa(pgCgpa)) {
                            if (hasDoneMinimumProjects(numProjects)) {
                                if (isFullTime) {
                                    if (isEligibleByInterviewMarks(interviewMarks)) {
                                        return new Candidate(name, dob, hscMarks, ugCgpa, pgCgpa, numProjects, isFullTime, interviewMarks);
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }

        return null;
    }

    private boolean isCitizenOfIndia(String nationality) {
        return nationality.equals("Indian");
    }

    private boolean isEligibleByAge(LocalDate dob) {
        LocalDate cutoffDate = LocalDate.of(1999, Month.JULY, 1);
        return dob.isBefore(cutoffDate) || dob.isEqual(cutoffDate);
    }

    private boolean isEligibleByHscMarks(HscMarks hscMarks) {
        double aggregatePercentage = hscMarks.calculateAggregatePercentage();
        return aggregatePercentage >= 60.0;
    }

    private boolean isEligibleByPcmPcbAverage(HscMarks hscMarks) {
        double pcmPcbAveragePercentage = hscMarks.calculatePcmPcbAveragePercentage();
        if (hscMarks.getCategory().equals("SC") || hscMarks.getCategory().equals("ST")) {
            return pcmPcbAveragePercentage >= 50.0;
        }
```

- This is the **RecruitingTeam** class that implements the logic for evaluating candidates based on the eligibility criteria. It has a constructor that takes in a list of **EligibilityCriteria** objects, which define the specific eligibility requirements.

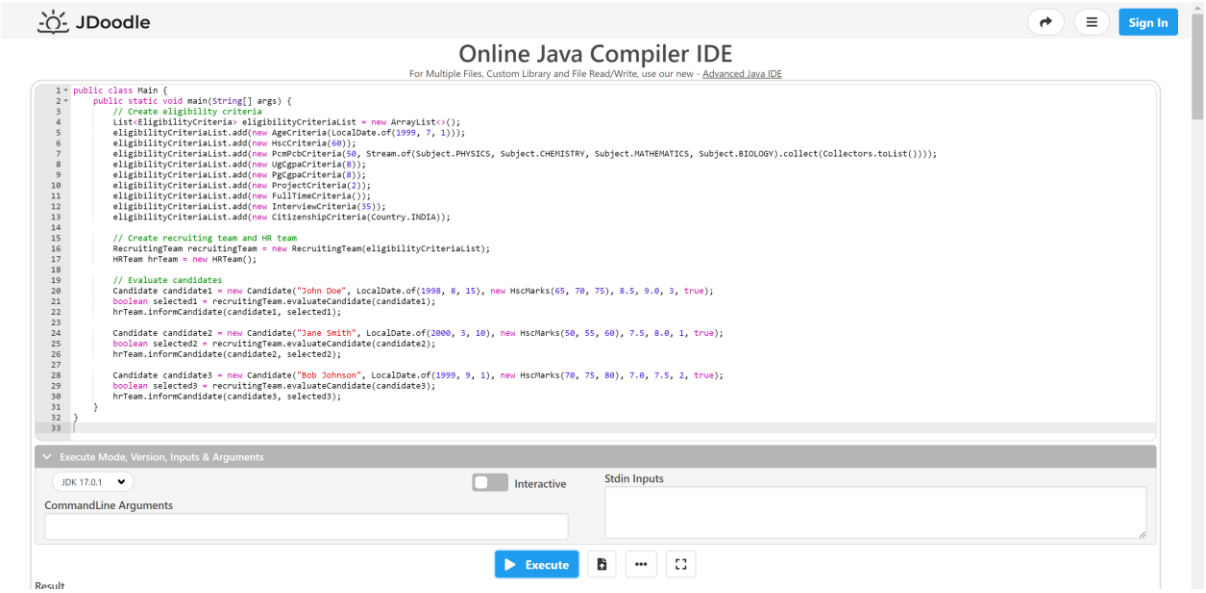
- The **evaluateCandidate** method takes in the candidate's information and returns a **Candidate** object if the candidate is eligible, or null otherwise. The method checks each eligibility criterion one by one and returns null if any criterion is not met. If all criteria are met, it creates and returns a new **Candidate** object with the candidate's information.



```java
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }

        return null;
    }

    private boolean isCitizenOfIndia(String nationality) {
        return nationality.equals("Indian");
    }

    private boolean isEligibleByAge(LocalDate dob) {
        LocalDate cutoffDate = LocalDate.of(1999, Month.JULY, 1);
        return dob.isBefore(cutoffDate) || dob.isEqual(cutoffDate);
    }

    private boolean isEligibleByHscMarks(HscMarks hscMarks) {
        double aggregatePercentage = hscMarks.calculateAggregatePercentage();
        return aggregatePercentage >= 60.0;
    }

    private boolean isEligibleByPcmPcbAverage(HscMarks hscMarks) {
        double pcmPcbAveragePercentage = hscMarks.calculatePcmPcbAveragePercentage();
        if (hscMarks.getCategory().equals("SC") || hscMarks.getCategory().equals("ST")) {
            return pcmPcbAveragePercentage >= 50.0;
        }
        return true;
    }

    private boolean isEligibleByUgCgpa(double ugCgpa) {
        return ugCgpa >= 8.0;
    }

    private boolean isEligibleByPgCgpa(double pgCgpa) {
        return pgCgpa >= 8.0;
    }

    private boolean hasDoneMinimumProjects(int numProjects) {
        return numProjects >= 2;
    }

    private boolean isEligibleByInterviewMarks(int interviewMarks) {
        return interviewMarks >= 35;
    }
}
```

- This **Main** class is the entry point of the application. It creates the eligibility criteria, recruiting team, and HR team objects. It then creates three **Candidate** objects with their details and evaluates them using the **evaluateCandidate()** method of the **RecruitingTeam** class. Finally, it informs each candidate of their selection status using the **informCandidate()** method of the **HRTeam** class.

- Note that this class does not represent a specific team, but rather ties together all the objects and functions required to run the application.



➢ Candidate Class:

This class represents the candidate, and it should have the following data fields:

- **Name:** A string to store the name of the candidate.

- **DateOfBirth:** A Date object to store the candidate's date of birth.

- **Marks**: An array of integers to store the candidate's marks in Physics, Chemistry, and Mathematics/Biology in the HSC exam.

- **AverageMarks:** A double to store the average marks obtained in the subjects of physics, chemistry, mathematics, or biology (PCM or PCB) in the HSC exam.

- **CGPAUG:** A double to store the candidate's CGPA in the UG exam.

- **CGPAPG**: A double to store the candidate's CGPA in the PG exam.

- **Projects:** An integer to store the number of projects done by the candidate.

- **FullTime**: A boolean to represent if the candidate studied full-time or not.
- **InterviewMarks**: An integer to store the candidate's marks in the interview selection process.

➢ It should also have the following methods:

- **Constructor:** A constructor to initialize the data fields.
- **isEligible:** A method to check if the candidate is eligible based on the given criteria.
- **getFinalMarks**: A method to calculate the final marks of the candidate based on the given criteria.

➢ Recruiter Class:

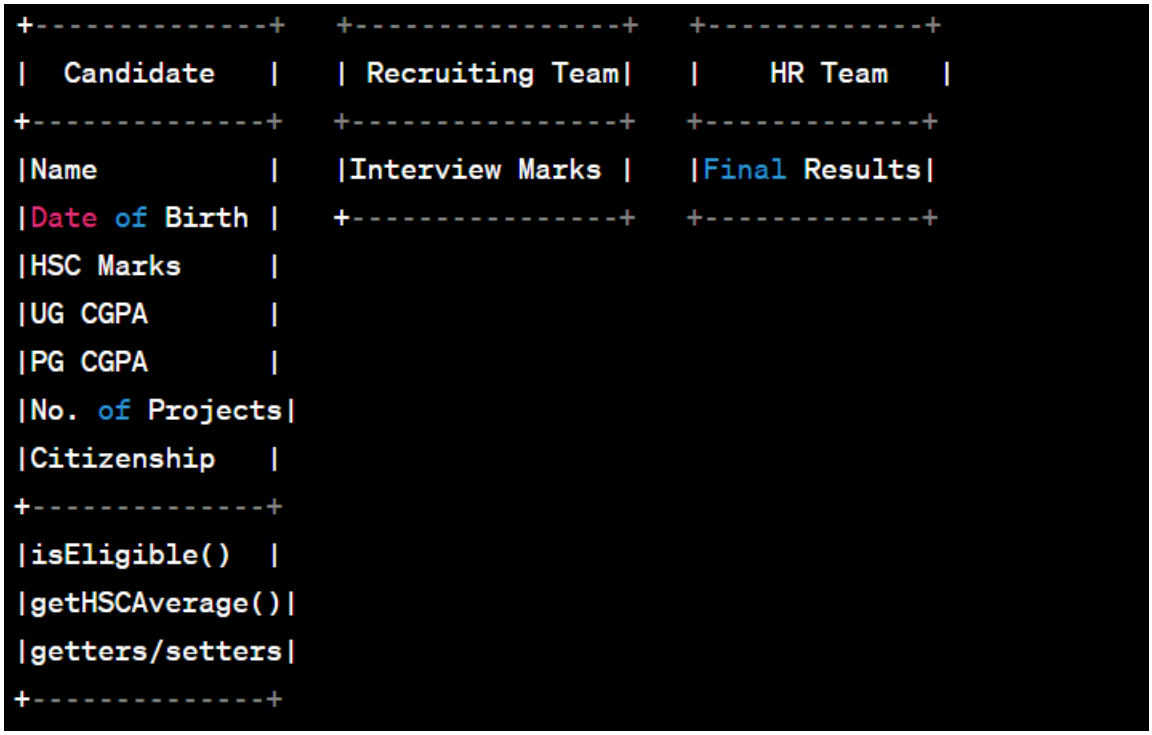
This class represents the recruitment team, and it should have the following data fields:

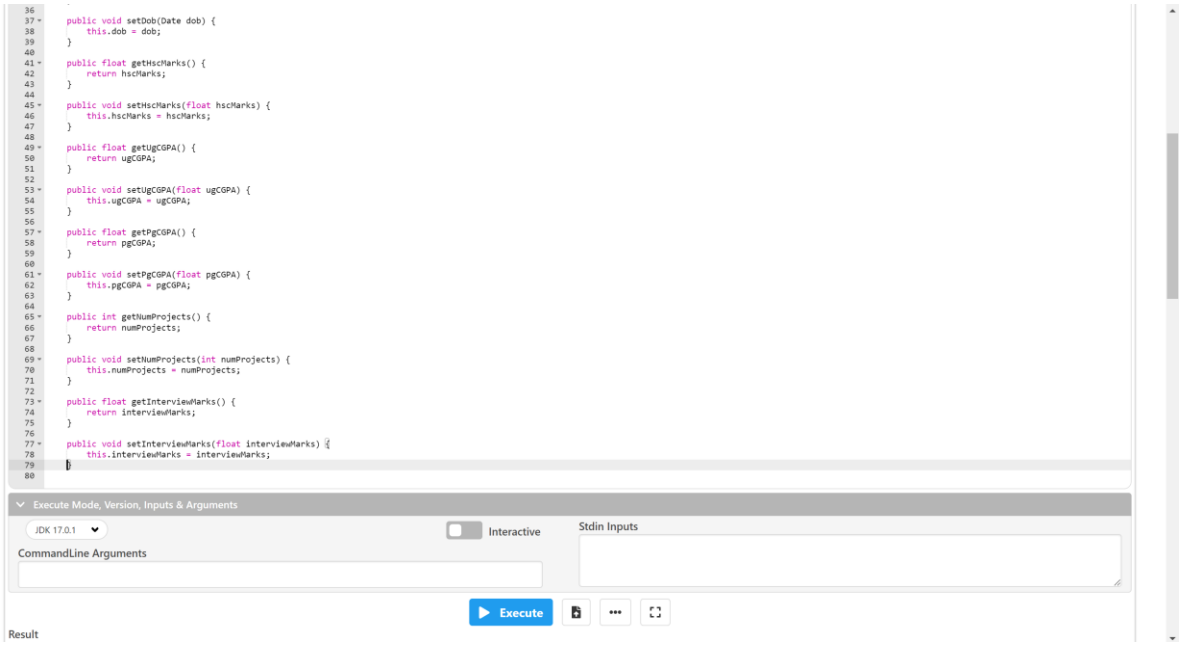
- None
- It should also have the following methods:
- **Constructor:** A constructor to initialize the data fields.
- **conductInterview:** A method to conduct an interview with the candidate and calculate their interview marks.

➢ HR Class:

- This class represents the HR team, and it should have the following data fields:
- **EligibleCandidates:** An ArrayList of Candidate objects to store the eligible candidates.
- It should also have the following methods:
- **Constructor:** A constructor to initialize the data fields.
- **evaluateCandidate:** A method to evaluate the candidate and add them to the EligibleCandidates list if they are eligible.
- **notifyCandidates:** A method to notify the eligible candidates of the final results.

➢ **The class diagram for the above classes can be depicted as follows:**





➢ Justification:

- The Candidate class represents the candidate who is being evaluated for recruitment. It contains all the relevant data fields and methods required to check their eligibility and calculate their final marks.

- The RecruitingTeam class represents the team responsible for conducting the evaluation process. It contains a list of Candidate objects and methods to add or remove candidates and retrieve the list of candidates.

- The HRTeam class represents the team responsible for communicating the results to the candidates. It also contains a list of Candidate objects and a method to send the results to the candidates.