

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.

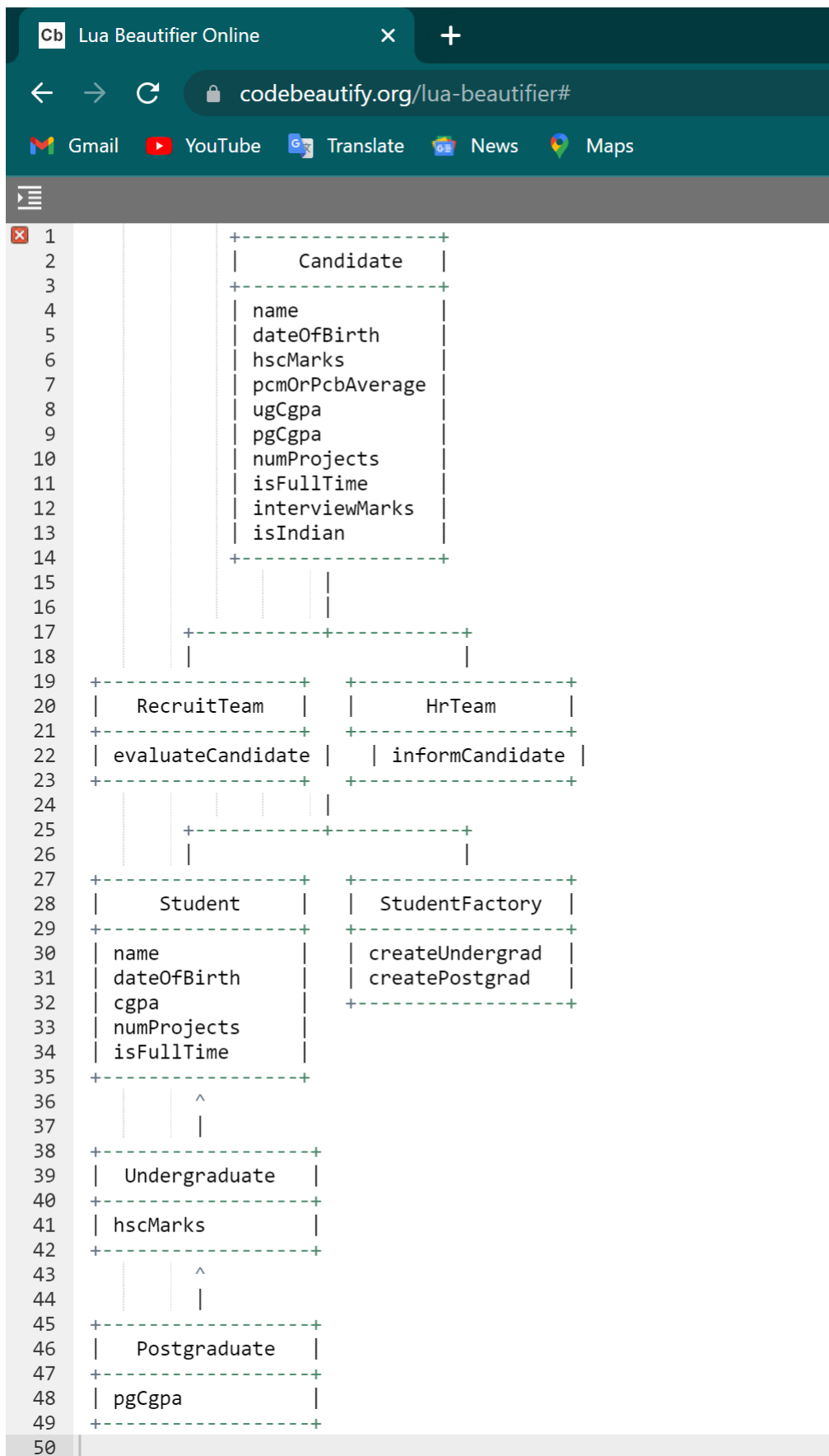
3. **HrTeam**: This class will handle the final results of the evaluation and inform the candidates about their status. It will have a method called `informCandidate` that takes a `Candidate` object as a parameter and informs them about their status.

- 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.



- 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:

The screenshot shows the JDoodle Online Java Compiler IDE. The code editor contains the following Java code:

```

1 public class AgeEligibility implements EligibilityCriteria {
2     private final LocalDate cutoffDate;
3
4     public AgeEligibility(LocalDate cutoffDate) {
5         this.cutoffDate = cutoffDate;
6     }
7
8     @Override
9     public boolean isEligible(Candidate candidate) {
10        LocalDate dob = candidate.getDob();
11        return dob.isBefore(cutoffDate) || dob.isEqual(cutoffDate);
12    }
13 }
14

```

Below the code editor, the 'Execute Mode, Version, Inputs & Arguments' section shows 'JDK 17.0.1' and 'Interactive' mode. The 'Execute' button is visible. The 'Result' section shows the following compilation errors:

```

/Ageligibility.java:1: error: cannot find symbol
public class AgeEligibility implements EligibilityCriteria {
^
  symbol: class EligibilityCriteria
/Ageligibility.java:2: error: cannot find symbol
    private final LocalDate cutoffDate;
^
  symbol: class LocalDate
  location: class AgeEligibility
/Ageligibility.java:4: error: cannot find symbol
    public AgeEligibility(LocalDate cutoffDate) {
^
  symbol: class LocalDate
  location: class AgeEligibility
/Ageligibility.java:9: error: cannot find symbol
    public boolean isEligible(Candidate candidate) {
^
  symbol: class Candidate
  location: class AgeEligibility
/Ageligibility.java:10: error: method does not override or implement a method from a supertype
    }
    ^

```

- 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:

```

65 }
66
67 public class Undergraduate extends Student {
68     private final HscMarks hscMarks;
69     private final double cgpa;
70     private final int numProjects;
71     private final boolean fullTime;
72
73     public Undergraduate(String rollNumber, String course, HscMarks hscMarks, double cgpa, int numProjects, boolean fullTime) {
74         super(rollNumber, course);
75         this.hscMarks = hscMarks;
76         this.cgpa = cgpa;
77         this.numProjects = numProjects;
78         this.fullTime = fullTime;
79     }
80
81     public HscMarks getHscMarks() {
82         return hscMarks;
83     }
84
85     public double getCgpa() {
86         return cgpa;
87     }
88
89     public int getNumProjects() {
90         return numProjects;
91     }
92
93     public boolean isFullTime() {
94         return fullTime;
95     }
96 }
97
98 public class Postgraduate extends Student {
99     private final HscMarks hscMarks;
100     private final double ugCgpa;
101     private final double pgCgpa;
102     private final int numProjects;
103     private final boolean fullTime;
104
105     public Postgraduate(String rollNumber, String course, HscMarks hscMarks, double ugCgpa, double pgCgpa, int numProjects, boolean fullTime) {
106         super(rollNumber, course);
107         this.hscMarks = hscMarks;
108         this.ugCgpa = ugCgpa;
109         this.pgCgpa = pgCgpa;
110         this.numProjects = numProjects;
111         this.fullTime = fullTime;
112     }
113
114     public HscMarks getHscMarks() {
115         return hscMarks;
116     }
117
118     public double getUgCgpa() {
119         return ugCgpa;
120     }
121
122     public double getPgCgpa() {
123         return pgCgpa;
124     }

```

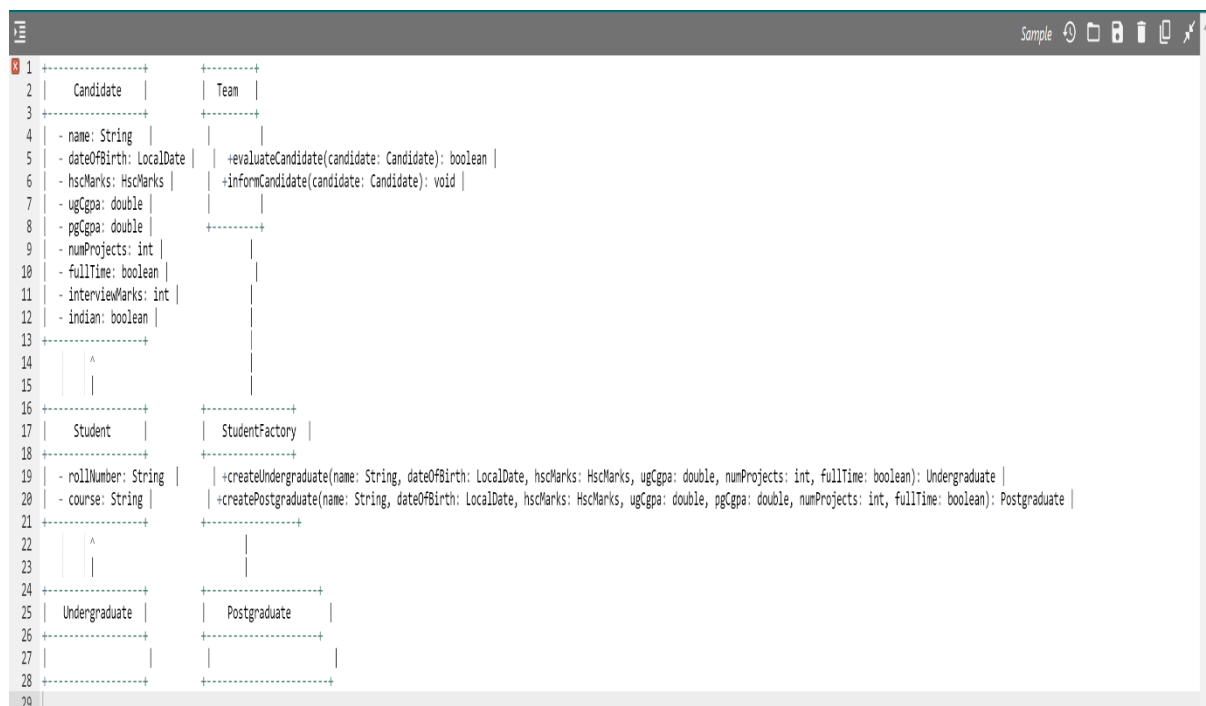
- In this implementation, we first call the **evaluateCandidate** method to check whether the candidate meets the eligibility criteria. If they do, we inform them that they have been selected. If not, we inform them that they have not been selected.
- With these classes and methods, we can simulate the recruitment process and evaluate candidates based on the given criteria.

```

1  +-----+ +-----+
2  | Candidate | | Team |
3  +-----+ +-----+
4  | - name: String | | |
5  | - dateOfBirth: LocalDate | | +evaluateCandidate(candidate: Candidate): boolean |
6  | - hscMarks: HscMarks | | | +informCandidate(candidate: Candidate): void |
7  | - ugCgpa: double | | |
8  | - pgCgpa: double | | |
9  | - numProjects: int | | |
10 | - fullTime: boolean | | |
11 | - interviewMarks: int | | |
12 | - indian: boolean | | |
13 +-----+ +-----+
14 | ^ |
15 | | |
16 +-----+ +-----+
17 | Student | | StudentFactory |
18 +-----+ +-----+
19 | - rollNumber: String | | +createUndergraduate(name: String, dateOfBirth: LocalDate, hscMarks: HscMarks, ugCgpa: double, numProjects: int, fullTime: boolean): Undergraduate |
20 | - course: String | | | +createPostgraduate(name: String, dateOfBirth: LocalDate, hscMarks: HscMarks, ugCgpa: double, pgCgpa: double, numProjects: int, fullTime: boolean): Postgraduate |
21 +-----+ +-----+
22 | ^ |
23 | | |
24 +-----+ +-----+
25 | Undergraduate | | Postgraduate |
26 +-----+ +-----+
27 | | |
28 +-----+ +-----+
29

```

- 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.



```

1 import java.time.LocalDate;
2
3 public class HscMarks {
4     private final int physics;
5     private final int chemistry;
6     private final int mathematics;
7     private final int biology;
8
9     public HscMarks(int physics, int chemistry, int mathematics, int biology) {
10         this.physics = physics;
11         this.chemistry = chemistry;
12         this.mathematics = mathematics;
13         this.biology = biology;
14     }
15
16     public int getPhysics() {
17         return physics;
18     }
19
20     public int getChemistry() {
21         return chemistry;
22     }
23
24     public int getMathematics() {
25         return mathematics;
26     }
27
28     public int getBiology() {
29         return biology;
30     }
31
32     public double getAverageMarks() {
33         return (physics + chemistry + mathematics + biology) / 4.0;
34     }
35
36     public boolean isCorST() {
37         return false; // TODO: Implement based on criteria
38     }
39 }
40
41 public abstract class Student {
42     private String rollNumber;
43     private String course;
44
45     public Student(String rollNumber, String course) {
46         this.rollNumber = rollNumber;
47         this.course = course;
48     }
49
50     public String getRollNumber() {
51         return rollNumber;
52     }
53 }

```

- 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.

```

65 }
66
67 public class Undergraduate extends Student {
68     private final HscMarks hscMarks;
69     private final double cgpa;
70     private final int numProjects;
71     private final boolean fullTime;
72
73     public Undergraduate(String rollNumber, String course, HscMarks hscMarks, double cgpa, int numProjects, boolean fullTime) {
74         super(rollNumber, course);
75         this.hscMarks = hscMarks;
76         this.cgpa = cgpa;
77         this.numProjects = numProjects;
78         this.fullTime = fullTime;
79     }
80
81     public HscMarks getHscMarks() {
82         return hscMarks;
83     }
84
85     public double getCgpa() {
86         return cgpa;
87     }
88
89     public int getNumProjects() {
90         return numProjects;
91     }
92
93     public boolean isFullTime() {
94         return fullTime;
95     }
96 }
97
98 public class Postgraduate extends Student {
99     private final HscMarks hscMarks;
100     private final double ugCgpa;
101     private final double pgCgpa;
102     private final int numProjects;
103     private final boolean fullTime;
104
105     public Postgraduate(String rollNumber, String course, HscMarks hscMarks, double ugCgpa, double pgCgpa, int numProjects, boolean fullTime) {
106         super(rollNumber, course);
107         this.hscMarks = hscMarks;
108         this.ugCgpa = ugCgpa;
109         this.pgCgpa = pgCgpa;
110         this.numProjects = numProjects;
111         this.fullTime = fullTime;
112     }
113
114     public HscMarks getHscMarks() {
115         return hscMarks;
116     }
117
118     public double getUgCgpa() {
119         return ugCgpa;
120     }
121
122     public double getPgCgpa() {
123         return pgCgpa;
124     }

```

- 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.

```

129
130     public boolean isFullTime() {
131         return fullTime;
132     }
133 }
134
135 public class Candidate {
136     private String name;
137     private LocalDate dateOfBirth;
138     private HscMarks hscMarks;
139     private double ugGpa;
140     private double pgGpa;
141     private int numProjects;
142     private boolean fullTime;
143     private int interviewMarks;
144     private boolean indianCitizen;
145
146     public Candidate(String name, LocalDate dateOfBirth, HscMarks hscMarks, double ugGpa, double pgGpa, int numProjects, boolean fullTime, int interviewMarks, boolean indianCitizen) {
147         this.name = name;
148         this.dateOfBirth = dateOfBirth;
149         this.hscMarks = hscMarks;
150         this.ugGpa = ugGpa;
151         this.pgGpa = pgGpa;
152         this.numProjects = numProjects;
153         this.fullTime = fullTime;
154         this.interviewMarks = interviewMarks;
155         this.indianCitizen = indianCitizen;
156     }
157
158     public String getName() {
159         return name;
160     }
161
162     public LocalDate getDateOfBirth() {
163         return dateOfBirth;
164     }
165
166     public HscMarks getHscMarks() {
167         return hscMarks;
168     }
169
170     public double getUgGpa() {
171         return ugGpa;
172     }
173
174     public double getPgGpa() {
175         return pgGpa;
176     }
177
178     public int getNumProjects() {
179         return numProjects;
180     }
181
182     public boolean isFullTime() {
183         return fullTime;
184     }
185
186     public int getInterviewMarks() {
187         return interviewMarks;
188     }

```

```

200         hasMinimumTwoProjects() &&
201         isFullTime() &&
202         has350AboveInInterview() &&
203         isIndianCitizen();
204     }
205
206     private boolean isBornAfterJuly1999() {
207         LocalDate july1999 = LocalDate.of(1999, 7, 1);
208         return dateOfBirth.isAfter(july1999) || dateOfBirth.isEqual(july1999);
209     }
210
211     private boolean has60PercentInPCMO-PCB() {
212         int physics = hscMarks.getPhysics();
213         int chemistry = hscMarks.getChemistry();
214         int mathOrBio = hscMarks.isCorST() ? hscMarks.getBiology() : hscMarks.getMathematics();
215         return (physics + chemistry + mathOrBio) / 3.0 >= 60;
216     }
217
218     private boolean has50PercentAverageInPCMO-PCB() {
219         return hscMarks.getAverageMarks() >= 50;
220     }
221
222     private boolean has80AboveInUGGpa() {
223         return ugGpa >= 8;
224     }
225
226     private boolean has80AboveInPGGpa() {
227         return pgGpa >= 8;
228     }
229
230     private boolean hasMinimumTwoProjects() {
231         return numProjects >= 2;
232     }
233
234     private boolean has350AboveInInterview() {
235         return interviewMarks >= 35;
236     }
237 }
238
239 public class RecruitingTeam {
240     public Candidate evaluateCandidate(String name, LocalDate dateOfBirth, HscMarks hscMarks, double ugGpa, double pgGpa, int numProjects, boolean fullTime, int interviewMarks, boolean indianCitizen) {
241         Candidate candidate = new Candidate(name, dateOfBirth, hscMarks, ugGpa, pgGpa, numProjects, fullTime, interviewMarks, indianCitizen);
242         if (candidate.isEligible)
243             return candidate;
244         else {
245             return null;
246         }
247     }
248 }
249
250 public class HrTeam {
251     public void informCandidate(Candidate candidate) {
252         if (candidate != null) {
253             System.out.println("Congratulations, " + candidate.getName() + "! You have been selected.");
254         }
255         else {
256             System.out.println("Sorry, you have not been selected.");
257         }
258     }
259 }

```

- 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.

The screenshot shows the JDoodle Online Java Compiler IDE. The code defines a class `AgeEligibility` that implements the `EligibilityCriteria` interface. The code includes a private final `LocalDate` `cutoffDate`, a constructor, and an `isEligible` method. The IDE interface shows the code in the editor, the execution mode set to 'Execute', and the command line arguments. The result pane shows compilation errors: 'cannot find symbol' for `EligibilityCriteria`, `LocalDate`, and `Candidate`, and 'method does not override or implement a method from a supertype' for `isEligible`.

```

1 public class AgeEligibility implements EligibilityCriteria {
2     private final LocalDate cutoffDate;
3
4     public AgeEligibility(LocalDate cutoffDate) {
5         this.cutoffDate = cutoffDate;
6     }
7
8     @Override
9     public boolean isEligible(Candidate candidate) {
10        LocalDate dob = candidate.getDob();
11        return dob.isBefore(cutoffDate) || dob.isEqual(cutoffDate);
12    }
13 }
14

```

Result
CPU Time: sec(s). Memory: kilobyte(s) compiled and executed in 0.394 sec(s)

```

/AgEligibility.java:1: error: cannot find symbol
public class AgeEligibility implements EligibilityCriteria {
                                   ^
  symbol: class EligibilityCriteria
/AgEligibility.java:2: error: cannot find symbol
    private final LocalDate cutoffDate;
                   ^
  symbol: class LocalDate
  location: class AgeEligibility
/AgEligibility.java:4: error: cannot find symbol
    public AgeEligibility(LocalDate cutoffDate) {
                   ^
  symbol: class LocalDate
  location: class AgeEligibility
/AgEligibility.java:9: error: error: cannot find symbol
    public boolean isEligible(Candidate candidate) {
                           ^
  symbol: class Candidate
  location: class AgeEligibility
/AgEligibility.java:10: error: method does not override or implement a method from a supertype
    public boolean isEligible(Candidate candidate) {
    ^

```

The screenshot shows the JDoodle Online Java Compiler IDE with a Java class `Candidate` and its subclass `IndianCandidate`. The `Candidate` class is an abstract class with protected fields for `name`, `dob`, `hscMarks`, `ugCgpa`, `pgCgpa`, `numProjects`, `isFullTime`, and `interviewMarks`. It has a constructor and several getter methods. The `IndianCandidate` class extends `Candidate` and overrides the `isIndian` method.

```

1 public abstract class Candidate {
2     protected String name;
3     protected LocalDate dob;
4     protected HscMarks hscMarks;
5     protected double ugCgpa;
6     protected double pgCgpa;
7     protected int numProjects;
8     protected boolean isFullTime;
9     protected int interviewMarks;
10
11     public Candidate(String name, LocalDate dob, HscMarks hscMarks, double ugCgpa, double pgCgpa, int numProjects, boolean isFullTime, int interviewMarks) {
12         this.name = name;
13         this.dob = dob;
14         this.hscMarks = hscMarks;
15         this.ugCgpa = ugCgpa;
16         this.pgCgpa = pgCgpa;
17         this.numProjects = numProjects;
18         this.isFullTime = isFullTime;
19         this.interviewMarks = interviewMarks;
20     }
21
22     public abstract boolean isIndian();
23
24     public String getName() {
25         return name;
26     }
27
28     public LocalDate getDob() {
29         return dob;
30     }
31
32     public HscMarks getHscMarks() {
33         return hscMarks;
34     }
35
36     public double getUgCgpa() {
37         return ugCgpa;
38     }
39
40     public double getPgCgpa() {
41         return pgCgpa;
42     }
43
44     public int getNumProjects() {
45         return numProjects;
46     }
47
48     public boolean isFullTime() {
49         return isFullTime;
50     }
51
52     public int getInterviewMarks() {
53         return interviewMarks;
54     }
55 }
56
57 public class IndianCandidate extends Candidate {
58     public IndianCandidate(String name, LocalDate dob, HscMarks hscMarks, double ugCgpa, double pgCgpa, int numProjects, boolean isFullTime, int interviewMarks) {
59         super(name, dob, hscMarks, ugCgpa, pgCgpa, numProjects, isFullTime, interviewMarks);
60     }
61 }

```

- 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.

```

1 public class RecruitingTeam {
2     private final List<EligibilityCriteria> eligibilityCriteriaList;
3
4     public RecruitingTeam(List<EligibilityCriteria> eligibilityCriteriaList) {
5         this.eligibilityCriteriaList = eligibilityCriteriaList;
6     }
7
8     public Candidate evaluateCandidate(String name, LocalDate dob, HscMarks hscMarks, double ugGpa, double pgGpa, int numProjects, boolean isFullTime, int interviewMarks, String nationality) {
9         if (!isCitizenOfIndia(nationality)) {
10             return null;
11         }
12
13         if (isEligibleByAge(dob)) {
14             if (isEligibleByHscMarks(hscMarks)) {
15                 if (isEligibleByPcmPcbAverage(hscMarks)) {
16                     if (isEligibleByUgGpa(ugGpa)) {
17                         if (isEligibleByPgGpa(pgGpa)) {
18                             if (hasDoneMinimumProjects(numProjects)) {
19                                 if (isFullTime) {
20                                     if (isEligibleByInterviewMarks(interviewMarks)) {
21                                         return new Candidate(name, dob, hscMarks, ugGpa, pgGpa, numProjects, isFullTime, interviewMarks);
22                                     }
23                                 }
24                             }
25                         }
26                     }
27                 }
28             }
29         }
30         return null;
31     }
32
33     private boolean isCitizenOfIndia(String nationality) {
34         return nationality.equals("Indian");
35     }
36
37     private boolean isEligibleByAge(LocalDate dob) {
38         LocalDate cutoffDate = LocalDate.of(1999, Month.JULY, 1);
39         return dob.isBefore(cutoffDate) || dob.isEqual(cutoffDate);
40     }
41
42     private boolean isEligibleByHscMarks(HscMarks hscMarks) {
43         double aggregatePercentage = hscMarks.calculateAggregatePercentage();
44         return aggregatePercentage >= 60.0;
45     }
46
47     private boolean isEligibleByPcmPcbAverage(HscMarks hscMarks) {
48         double pcmPcbAveragePercentage = hscMarks.calculatePcmPcbAveragePercentage();
49         if (hscMarks.getCategory().equals("SC") || hscMarks.getCategory().equals("ST")) {
50             return pcmPcbAveragePercentage >= 50.0;
51         }
52         return pcmPcbAveragePercentage >= 50.0;
53     }
54
55     private boolean isEligibleByUgGpa(double ugGpa) {
56         return ugGpa >= 8.0;
57     }
58
59     private boolean isEligibleByPgGpa(double pgGpa) {
60         return pgGpa >= 8.0;
61     }
62
63     private boolean hasDoneMinimumProjects(int numProjects) {
64         return numProjects >= 2;
65     }
66
67     private boolean isEligibleByInterviewMarks(int interviewMarks) {
68         return interviewMarks >= 35;
69     }
70
71 }
72

```

- 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.

```

22
23
24
25
26
27
28
29
30
31
32
33
34 private boolean isCitizenOfIndia(String nationality) {
35     return nationality.equals("Indian");
36 }
37
38 private boolean isEligibleByAge(LocalDate dob) {
39     LocalDate cutoffDate = LocalDate.of(1999, Month.JULY, 1);
40     return dob.isBefore(cutoffDate) || dob.isEqual(cutoffDate);
41 }
42
43 private boolean isEligibleByHscMarks(HscMarks hscMarks) {
44     double aggregatePercentage = hscMarks.calculateAggregatePercentage();
45     return aggregatePercentage >= 60.0;
46 }
47
48 private boolean isEligibleByPcmPcbAverage(HscMarks hscMarks) {
49     double pcmPcbAveragePercentage = hscMarks.calculatePcmPcbAveragePercentage();
50     if (hscMarks.getCategory().equals("SC") || hscMarks.getCategory().equals("ST")) {
51         return pcmPcbAveragePercentage >= 50.0;
52     }
53     return pcmPcbAveragePercentage >= 50.0;
54 }
55
56 private boolean isEligibleByUgGpa(double ugGpa) {
57     return ugGpa >= 8.0;
58 }
59
60 private boolean isEligibleByPgGpa(double pgGpa) {
61     return pgGpa >= 8.0;
62 }
63
64 private boolean hasDoneMinimumProjects(int numProjects) {
65     return numProjects >= 2;
66 }
67
68 private boolean isEligibleByInterviewMarks(int interviewMarks) {
69     return interviewMarks >= 35;
70 }
71
72

```

- 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.

```

1 public class Main {
2     public static void main(String[] args) {
3         // Create eligibility criteria
4         List<EligibilityCriteria> eligibilityCriteriaList = new ArrayList<>();
5         eligibilityCriteriaList.add(new AgeCriteria(LocalDate.of(1999, 7, 1)));
6         eligibilityCriteriaList.add(new HscCriteria(80));
7         eligibilityCriteriaList.add(new PcmCriteria(50, Stream.of(Subject.PHYSICS, Subject.CHEMISTRY, Subject.MATHEMATICS, Subject.BIOLOGY).collect(Collectors.toList())));
8         eligibilityCriteriaList.add(new UgcpgcCriteria(8));
9         eligibilityCriteriaList.add(new PgcpacCriteria(6));
10        eligibilityCriteriaList.add(new ProjectCriteria(2));
11        eligibilityCriteriaList.add(new FullTimeCriteria());
12        eligibilityCriteriaList.add(new InterviewCriteria(55));
13        eligibilityCriteriaList.add(new CitizenshipCriteria(Country.INDIA));
14
15        // Create recruiting team and HR team
16        RecruitingTeam recruitingTeam = new RecruitingTeam(eligibilityCriteriaList);
17        HRTeam hrTeam = new HRTeam();
18
19        // Evaluate candidates
20        Candidate candidate1 = new Candidate("John Doe", LocalDate.of(1998, 8, 15), new HscMarks(65, 70, 75), 8.5, 9.0, 3, true);
21        boolean selected1 = recruitingTeam.evaluateCandidate(candidate1);
22        hrTeam.informCandidate(candidate1, selected1);
23
24        Candidate candidate2 = new Candidate("Jane Smith", LocalDate.of(2000, 3, 10), new HscMarks(50, 55, 60), 7.5, 8.0, 1, true);
25        boolean selected2 = recruitingTeam.evaluateCandidate(candidate2);
26        hrTeam.informCandidate(candidate2, selected2);
27
28        Candidate candidate3 = new Candidate("Bob Johnson", LocalDate.of(1999, 9, 1), new HscMarks(70, 75, 80), 7.0, 7.5, 2, true);
29        boolean selected3 = recruitingTeam.evaluateCandidate(candidate3);
30        hrTeam.informCandidate(candidate3, selected3);
31    }
32 }
33

```

Execute Mode, Version, Inputs & Arguments

JDK 17.0.1

Interactive

Stdin Inputs

CommandLine Arguments

Execute

Result

➤ 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:**

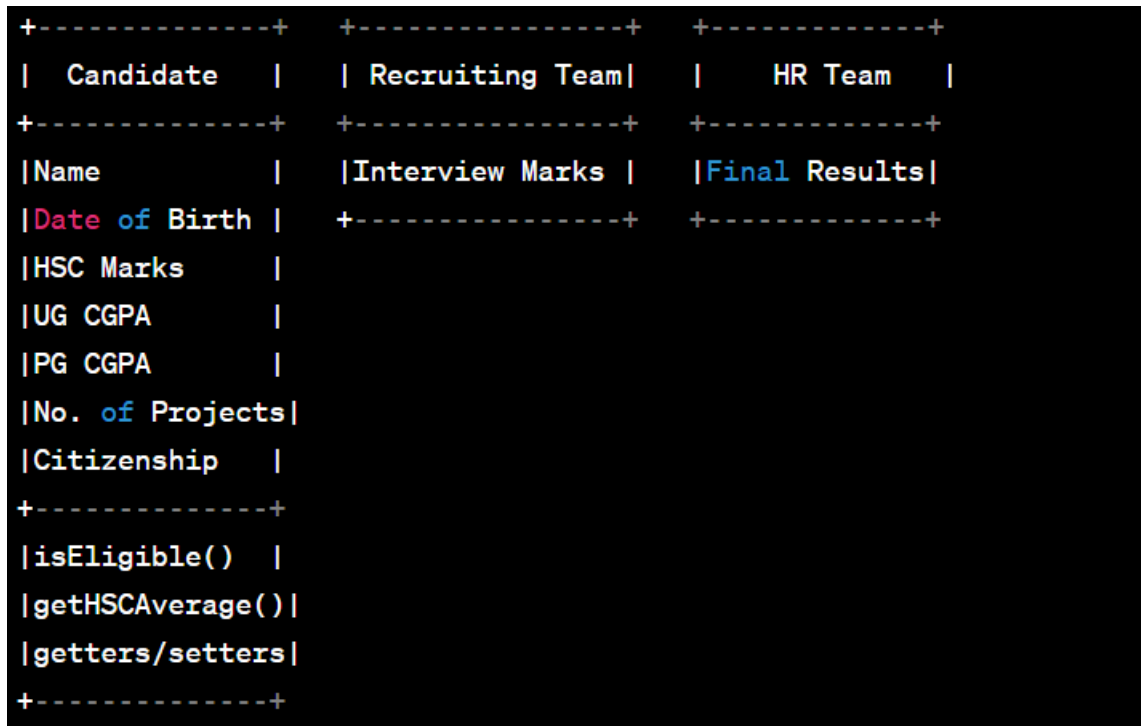
This class represents the recruitment team, and it should have the following data fields:

- 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:



JDoodle

Online Java Compiler IDE

For Multiple Files, Custom Library and File Read/Write, use our new - [Advanced Java IDE](#)

```

1 The code implementation in Java could be as follows:
2 public class Candidate {
3     private String name;
4     private Date dob;
5     private float hscMarks;
6     private float ugCGPA;
7     private float pgCGPA;
8     private int numProjects;
9     private float interviewMarks;
10    private String citizenship;
11
12    // Constructor
13    public Candidate(String name, Date dob, float hscMarks, float ugCGPA, float pgCGPA, int numProjects, float interviewMarks, String citizenship) {
14        this.name = name;
15        this.dob = dob;
16        this.hscMarks = hscMarks;
17        this.ugCGPA = ugCGPA;
18        this.pgCGPA = pgCGPA;
19        this.numProjects = numProjects;
20        this.interviewMarks = interviewMarks;
21        this.citizenship = citizenship;
22    }
23
24    // Getters and Setters
25    public String getName() {
26        return name;
27    }
28
29    public void setName(String name) {
30        this.name = name;
31    }
32
33    public Date getDob() {
34        return dob;
35    }
36
37    public void setDob(Date dob) {
38        this.dob = dob;
39    }
40
41    public float getHscMarks() {
42        return hscMarks;
43    }
44
45    public void setHscMarks(float hscMarks) {
46        this.hscMarks = hscMarks;
47    }
48
49    public float getUgCGPA() {
50        return ugCGPA;
51    }
52
53    public void setUgCGPA(float ugCGPA) {
54
  
```

```

36
37
38 public void setDob(Date dob) {
39     this.dob = dob;
40 }
41
42 public float getHscMarks() {
43     return hscMarks;
44 }
45
46 public void setHscMarks(float hscMarks) {
47     this.hscMarks = hscMarks;
48 }
49
50 public float getUgcGpa() {
51     return ugcGpa;
52 }
53
54 public void setUgcGpa(float ugcGpa) {
55     this.ugcGpa = ugcGpa;
56 }
57
58 public float getPgGpa() {
59     return pgGpa;
60 }
61
62 public void setPgGpa(float pgGpa) {
63     this.pgGpa = pgGpa;
64 }
65
66 public int getNumProjects() {
67     return numProjects;
68 }
69
70 public void setNumProjects(int numProjects) {
71     this.numProjects = numProjects;
72 }
73
74 public float getInterviewMarks() {
75     return interviewMarks;
76 }
77
78 public void setInterviewMarks(float interviewMarks) {
79     this.interviewMarks = interviewMarks;
80 }

```

Execute Mode, Version, Inputs & Arguments

IDK 17.0.1

☐ Interactive

Stdin Inputs

CommandLine Arguments

Execute

Result

➤ 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.