CPS 2231

Lab 6 – Abstract Class and Interface and Recursion

Instructions

Please note:

- **There are 2 tasks in this lab.**
- **Follow the instructions to finish them.**

Submission guidelines:

Task 1 (Gradescope)

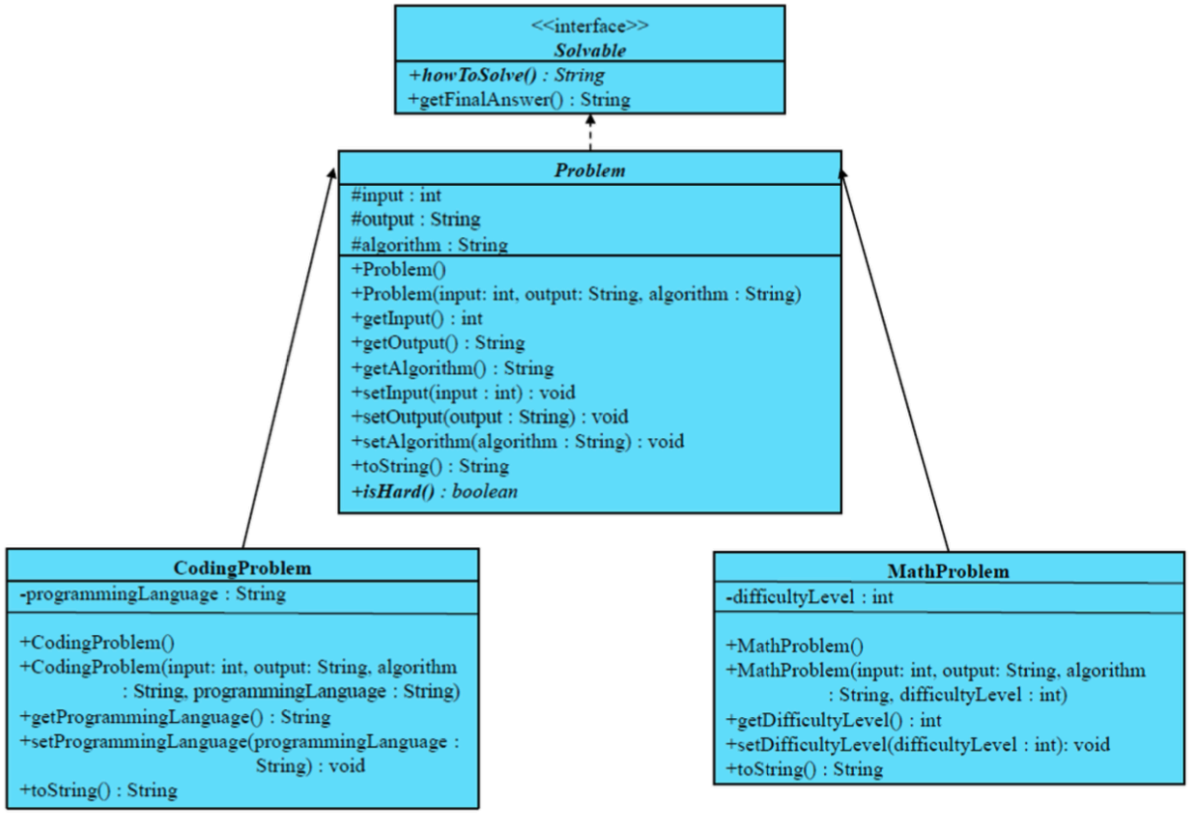
- Using the **SAMPLE FILES PROVIDED**, create a set of functional Java classes to complete the lab, according to the instructions provided below.
- **DO NOT MODIFY THE EXISTING CODE** in the sample files; you may **ONLY ADD CODE** to them.
- **DO NOT RENAME THE SAMPLE FILES.**
- Submit **ALL** of the following files (i.e., the files included as sample code) to the **GRADESCOPE** assignment titled **(TASK 1) Lab 6 – Abstract Class and Interface and Recursion**:
    - **Solvable.java**
    - **Problem.java**
    - **CodingProblem.java**
    - **MathProblem.java**
- Review the feedback (test case results) of your code in Gradescope.

Task 2 (Canvas)

- Write down your answers in a Word/Google Docs document.
- Convert it to a pdf file.
- Submit the pdf file to the assignment in CANVAS titled **(TASK 2) Lab 6 – Abstract Class and Interface and Recursion**

Task 1: Abstract Class and Interface

An Inheritance principle is used to explore and implement the ***Solvable*** interface below and the classes implementing it. The ***UML*** diagram of the inheritance hierarchy is represented below:



Part 1.1. Implement the *Solvable* inheritance structure strictly according to its UML diagram. Add comments stating where data fields, constructors, toString(), and other methods are (if any). **Neither method should have an empty body unless abstract methods.**

Note: You should implement the *Solvable interface, abstract class Problem, regular class CodingProblem, and regular class MathProblem*.

1. *Solvable interface Problem*

a. Two abstract methods:
   i. String **howToSolve()**
   ii. String **getFinalAnswer()**

2. *Abstract class Problem:*

a. Two constructors:
   i. default constructor
   ii. constructor with all fields
b. getters and setters for all fields
c. **toString()** method:
   i. Note: modify by yourself to match the example output
d. Abstract method **isHard()**

3. *CodingProblem class:*

a. One additional field
   i. String ***programmingLanguage***
b. Two constructors
   i. default constructor
   ii. constructor with all fields
      1. including ***programmingLanguage*** and the fields in the superclass ***Problem***
c. getters and setters for ***programmingLanguage***
d. **toString()** method:
   i. Note: modify by yourself to match the example output
e. Implement **isHard()**:
   i. Note: always return true
f. Implement **howToSolve()**:
   i. return a string: “Write a xxx program.”
      1. xxx should be ***programmingLanguage.***
g. Implement **getFinalAnswer()**:
   i. return a string: “Eureka!”

4. *MathProblem class:*

a. One additional field:
   i. int ***difficultyLevel.***
b. two constructors:
   i. *default constructor*
   ii. *constructor with all fields*
      1. *including* ***difficultyLevel*** and the fields in the superclass ***Problem.***
c. getters and setters for ***difficultyLevel***
d. **toString()** method:

    i. modify by yourself to match the example output
e. Implement **isHard():**
    i. if ***difficultyLevel*** > 50: return true
    ii. else: return False
f. Implement **howToSolve()**:
    i. return a string: “Multiply to itself.”
g. Implement **getFinalAnswer()**:
    i. return a string: “Eureka!”

Task 2: Recursion Tracing:

We have learned the binary search algorithm this semester, but we implement binary search using a while loop. In this exercise, we are going to see binary search implemented by recursion and trace the recursion.

Tracing the Recursion. Observe the recursive solution provided below and answer the following questions:

1. Which line(s) of this program define(s) ***the base case*** of the **binary**() method?
2. Which line(s) of this program include recursive call(s)?
3. Trace the recursion below.
    a. **You must show the tracing step by step (write them down); otherwise – little to no credit!**
4. At what step of your recursion tracing did you hit the base case?
5. What is the final output of this code?

```
1  public class binarySearch {
2
3      public static int binary(int[] arr, int target, int left, int right) {
4          if (left > right) {
5              return -1;
6          }
7          int mid = (left + right) / 2;
8          if (arr[mid] == target) {
9              return mid;
10         } else if (arr[mid] > target) {
11             return binary(arr, target, left, mid - 1);
12         } else {
13             return binary(arr, target, mid + 1, right);
14         }
15     }
16
17     public static void main(String[] args) {
18         int[] arr = {1, 2, 3, 4, 5, 6, 7};
19         int target = 2;
20
21         System.out.println(binary(arr, target, 0, arr.length - 1));
22
23
24     }
25
26 }
```