COMP 132: Advanced Programming
Spring 2016, Homework 3

In this homework, you are asked to write Java classes, abstract classes and interfaces to build type hierarchies.

Part 1. Organize the types described below into a type hierarchy, making use of interfaces and abstract classes as appropriate. Identify the abstract classes and interfaces. Write code for them.

- **File**: A File has the following members (fields and methods)
  - A field
    private long size
    which is the size of the file in bytes
  - A field
    private String name
    with no spaces in it
  - A field
    private Date dateCreated;
  - A method
    public (abstract?) void onClick()
    that is activated when the file icon is clicked on the operating system graphical user interface (GUI). Every file type implements this method differently.
  - Getters and setters (abstract?) for all appropriate fields, a **toString()** method
  - Write a static method that takes as argument a **File** array, visits each element of the array, and calls all polymorphic **File** methods on the element.

- **TextFile**: Every text file is a File
  - A field
    private String encoding; // can be "ASCII", "Unicode", "DOS", etc.
  - A field
    private long numChars; // the number of characters in the file
    Getters and setters for all appropriate fields
  - A method
    public (abstract?) void clear()
    that every kind of TextFile implements differently

- **HTMLFile:** Every HTMLFile is a TextFile
  - A field
    ExecutableFile browserCompatibleWith;
    that contains a reference to the executable (see below) of the browsers and version numbers that this HTML file is compatible with.
  - A field
    private int htmlVersion
  - A field
    private List<HTMLFile> linksTo;
    that contains links to the other HTML files this file has links to.
  - The **onClick()** method for an HTMLFile named "index.html" whose **browserCompatibleWith** refers to an **ExecutableFile** with name "**iexplore.exe**" writes "**iexplore.exe index.html**" on the console (pretend that this is a system call that starts a browser on this file).
  - The **clear()** method clears the "**linksTo**" list and sets "**browserCompatibleWith**" to "**iexplore.exe**"

- **PlainTextFile:**
  - **private String[] lines;**
  - The **onClick()** method for a **TextFile** prints the lines array on the output console.

- The **clear()** method sets the lines array to an array of size 10 where each line is an empty string (“”)
- Constructors, getters and setters as needed.
- A **toString()** method.

- **BinaryFile**:
  - **boolean canCopy;** //Digital rights management.
  - **public (abstract?) BinaryFile getCopy()**
    Creates and returns an independent copy of the file if **canCopy** is true. Throws the “**CopyingNotAllowedException**” otherwise.
  - Getters, setters, constructors and a **toString()** method as needed.

- **ExecutableFile**: Every ExecutableFile is a BinaryFile.
  - **private String platform**
    that contains the platform (“Mac OS on Intel” or “Android on ARM”) that the file can execute on.
  - **private byte[] contents;**
  - the **onClick()** method prints “winamp” on the console (pretend this is a
  - system call) if the name of the ExecutableFile is “winamp”
  - Getters, setters, constructors and a **toString()** method as needed.

- **DataFile**: Every **DataFile** is a **BinaryFile**.
  - **private String compressionFormat**
  - **private ExecutableFile opensWith**
  - **private boolean[] bits;**
    true stands for binary 1, false stands for binary 0
  - the **onClick()** method prints “winamp song.mp3” on the console (pretend this is a system call) if the name of the **ExecutableFile** openWith is “winamp” and the name of the **DataFile** is “song.mp3”

- **Openable**: **DataFile**s and **HTMLFile**s are **Openable**. They all should implement the **public String opensWith()** method which returns the name of the program (e.g., “iexplore.exe” or “winamp”) that opens the **DataFile** or **HTMLFile**.

Part 2. Build a type hierarchy representing different kinds of questions (and associated answers) in an examination. Write code for the following. Decide whether each should be a concrete class, abstract class or an interface. Write also a Test class to test your code.

- **Question**
  - **WrittenQuestion**: A kind of Question
  - **MultipleChoiceQuestion**: A kind of Question
    - **PickOneChoiceQuestion**: A kind of MultipleChoiceQuestion
    - **OrderAllChoicesQuestion**: A kind of MultipleChoiceQuestion
- **PartialCreditGiveable**
  - **PartialCreditGiveable** objects are required to have the following method:
    - **double getPartialCredit()**
  - **WrittenQuestion** objects and **OrderAllChoicesQuestion** objects are **PartialCreditGiveable** objects.

The fields and methods of the types above are described below.
- All **Question** objects have
  - a field **questionText** of type **String**.
  - A field **points** of type **double**
  - a constructor **Question(String text, double points)**
    The answer to the question is graded over “**points**” points.
  - getters for **questionText** and **points** (but no setters)

- a method double **getGrade()** that is computed differently for different kinds of questions.

- All **WrittenQuestion** objects have,
  - two fields **responseWords** and **correctAnswerWords**, both of type **String[]**
  - a setter and getter for **responseWords**
  - a constructor
    WrittenQuestion(String questionText, String[] responseWords, String[] correctAnswerWords, double points)
    In a **WritttenQuestion**, each element of the **correctAnswerWords[]** array is a single word. Each element of the array **responseWords[]** is also a single word.
  - **getGrade()** should return "**points**" if these two arrays match exactly. Otherwise, **getGrade()** should return 0.
  - **getPartialCredit()** should count how many of the words in the **correctAnswerWords** array appear in the **responseWords** array. The student gets partial credit proportional to the number of the words from **correctAnswerWords** he got right. For instance, if **correctAnswerWords** contains 10 words and only three of them are elements of **responseWords**, then the student should get 30% of "**points**" as partial credit.

- All **MultipleChoice** objects have,
  - a field choices of type **String[]**, which lists the choices,
  - a constructor with appropriate arguments,
  - a method **boolean isAnsweredCompletelyCorrectly()**,
    that **PickOneChoiceQuestion** and **OrderAllChoicesQuestion** implement differently.

- **PickOneChoiceQuestion** objects have
  - a field int **choiceIndex** which is the index of the choice the student picked.
  - a field int **correctIndex** which is the index of the correct choice.
  - an implementation of **isAnsweredCompletelyCorrectly()** that returns true iff **choiceIndex** is the same as **correctIndex**.
  - an implementation of **getGrade()** that returns "**points**" if the choice picked is correct; otherwise, **getGrade()** returns 0.

- **OrderAllChoicesQuestion** objects have
  - a field **correctOrdering** of type **int[]**, representing the correct ordering of the choices.
  - a field **studentsOrdering** of type **int[]**, representing the student's ordering of the choices
  - an implementation of **isAnsweredCompletelyCorrectly()** that returns true iff **correctOrdering** and **studentsOrdering** contain the same integers in the same order.
  - an implementation of **getGrade()** that returns "**points**" if **isAnsweredCompletelyCorrectly()** returns true; otherwise, **getGrade()** returns 0.

  - an implementation of **getPartialCredit()** that works as follows:
    Suppose that the following is the case

    Correct ordering: 2, 4, 5, 1, 3
    Students ordering: 1, 4, 5, 2, 3

    In this case, the student's ordering and the correct ordering match in three places. The student gets 3/5 = 60% of "**points**" as partial credit.