Programming Assignments on Java

Software Engineering Lab
Dept. of CSE, IIT Kharagpur

Instructions

1. Create a separate directory for each assignment.
2. The source files must contain documentation. At the end, use `javadoc` to generate the API documentation for your code. You may follow this tutorial[1] on the usage of `javadoc`. In particular, you should document the
   - Classes
   - All public fields, if any
   - All public methods (what does a method do, what parameters it takes, and what does it return)
3. All classes should be declared as *public*.
4. Unless otherwise specified, all fields should be declared as *private*. Use getters and setters wherever appropriate.
5. Any input to your program should be passed as command line argument(s), for example, `java TextProcessorTest "This is a string"`.
6. Compress all the directories (containing source code and documentation), and upload them to Moodle.
7. You may only access your books/notes, Moodle, the Java API documentation[2], and the aforementioned tutorial on `javadoc`. Use of any other online resource is not allowed.
8. Marks division:
   - Coding: 80%
   - API documentation: 20%

[1] `https://newcircle.com/bookshelf/java_fundamentals_tutorial/javadoc`
[2] `https://docs.oracle.com/javase/8/docs/api/`

A1: Text Processing with Java

Define a *singleton* class named `TextProcessor` that provides the following methods.

- `String[] getWords(String string)`: Return an array containing all the words from the input string.
- `int getWordCount(String string)`: Return the number of words present in the input string. This method **must** use the `getWords()` method.
- `int[] getWordLengths(String string)`: Return an array containing the length of each word from the input string.

Define a class named `TextProcessorTest` containing the `main` method, and illustrate the usage of the methods from the `TextProcessor` class.

Note: Consider that the input string contains only a-z (lower/upper cases) and spaces.

Hint: Read the API documentation of the String class.

A2: Bibliography Manager

A bibliography consists of various resources that are consulted in preparing a document. A bibliography manager (e.g., JabRef) allows to store details of all such resources, and automatically insert references in the document whenever required.

Figure 1 shows the class diagram of different resources maintained in a bibliography manager. Note that `print()` in the `Resource` class is an abstract method. Provide an implementation of `print()` in the `Book`, `Journal`, and `OnlineResource` classes so that every attribute of each resource is printed. The unfilled squares and filled circles, respectively, indicate private and public access specifiers.

Define another class, `BibliographyManager`, containing the `main` method. Inside `main()`, you should create at least one resource of each type, and print their details. Some examples are given below.

- Online resource:
  "Overview (Java Platform SE 8)", https://docs.oracle.com/javase/8/docs/api/
- Book:
  Ken Arnold, James Gosling, and David Holmes, The Java Programming Language. Addison Wesley, p. 928, 2005, ISBN: 978-0321349804.
- Journal article:
  B. K. Saha and S. Misra, "Named Content Searching in Opportunistic Mobile Networks," IEEE Communications Letters, vol. 20, no. 10, pp. 2067–2070, 2016.

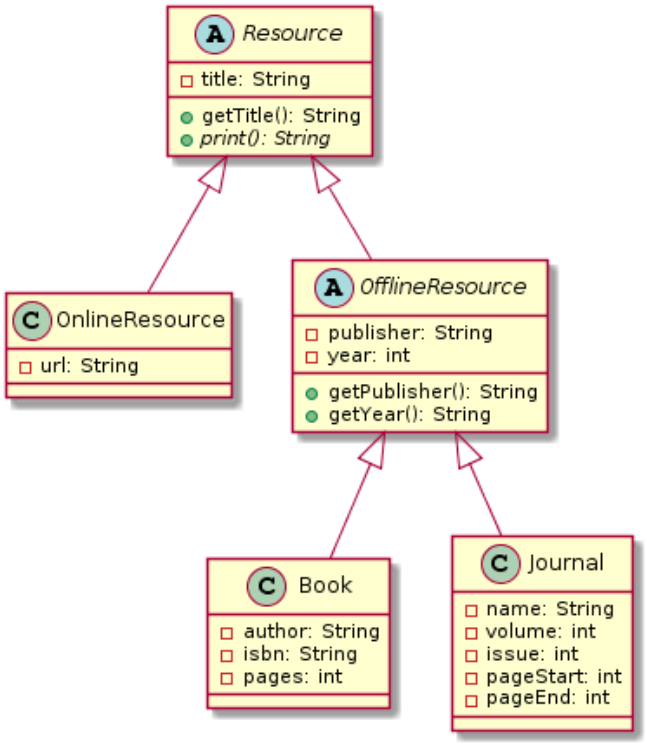

Figure 1: Class diagram of various resources included in bibliography.

You may refer to Amazon for other examples on books, and search IEEE Xplore[3] for journal articles.

A3: Employee Management

Company X was launched two years back, and now consists of a dozen of employees in different roles. It was decided to create a Java-based software to digitize the employee records for better management. In particular, the following aspects need to be addressed.

1. Each employee should be abstracted by the `Employee` class, and should be assigned a unique ID starting from 1001. The ID should be automatically incremented by unity as a new employee is added. The ID value is never decreased. Employees, of course, should have a name field along with salary. The class should have a `getSalary()` method to know the salary of a given employee. The salary of an employee is fixed at Rs 30000.

[3] `http://ieeexplore.ieee.org/Xplore/home.jsp`

2. Manager are special type of employees. Apart from inheriting the states and behaviors of `Employee`, a manager maintains a count of the number of employees supervised. Moreover, a manager's salary is determined by using the following formula: $\log(1+nrofSubordinates) \times employeeSalary$, where $employeeSalary$ is the salary obtained by an ordinary employee; $nrofSubordinates > 0$;

3. Onsite managers are special type of managers who get additional renumeration based on the formula: $log(1+nrofMonthsAtOnsite) \times managerSalary$, where $managerSalary$ is the typical salary obtained by a manager; $nrofMonthsAtOnsite > 0$.

4. Finally, there are technical architects, whose salary is determined by the formula $1.5 \times managerSalary$.

One of the goals of developing this software is to keep track of the cost-to-company (CTC) for all the employees. It is proposed to determine the total expenses using the following line of Java code:

```
for (Employee employee :  employees) {
total += employee.getSalary();
}
```

Apply the concepts of inheritance and polymorphism to complete the rest of the code required to develop this software. Provide appropriate constructors for each class. Inside `main()`, you should create at least two employees of each type, and store them in an array called `employees`.

A4: Polymorphic Virus

Computer viruses can inflict wide range of damages ranging from service interruption and spying to theft and permanent loss of data. Polymorphic viruses are particularly hard to detect due to the reason that they continuously keep changing their code, but still resulting in the same effect. For example, 5×3, $14 + 1$, and $21 - 6$ are different codes, but all produce the same output, 15.

Figure 2 shows the class diagram of a hypothetical polymorphic virus implementation. In particular, the virus has two integer fields, `number1` and `number2`, that must be taken as arguments from the user, and passed to the constructors. The `execute` method of `PolymorphicVirus` should simply print the two numbers.

Let $sum = number1 + number2$. The `AdditivePolymorphic` class should generate two numbers say, $a1$ and $a2$, such that $a1 + a2 = sum$. The `execute` method of this class should print $a1$ and $a2$. Similarly, the `execute` method of the `SubstractivePolymorphicVirus` should print two numbers whose subtraction results into sum; the `MultiplicativePolymorphicVirus` should provide a similar multiplication operation.

Notes:

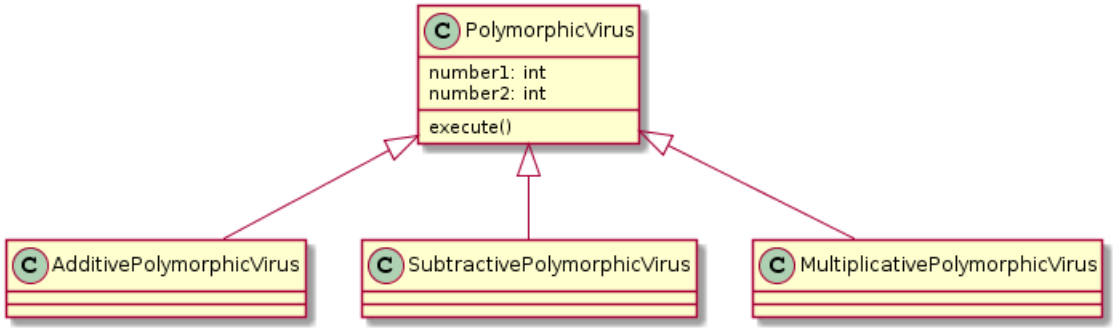

Figure 2: Class diagram of polymorphic viruses.

- Assume that `number1` and `number2` are non-prime and greater than or equal to 200. However, no number should be hard-coded.
- Repeated call to `execute()` of any polymorphic virus object should print (or return as a string) different numbers, in general. For example, 100×2, 40×5, and 2×200. Use the `Math.random()` method with appropriate scaling for this purpose.
- The code inside the `main` method should look as shown in Figure 3.

```
1  public class PolymorphicVirusTest {
2      public static void main(String[] args) {
3          int x1;
4          int x2;
5          // TODO: Read two integers from the command line, and assign
6          // them to x1 and x2
7
8          // Create the viruses
9          PolymorphicVirus[] viruses = new PolymorphicVirus[4];
10         viruses[0] = new PolymorphicVirus(x1, x2);
11         viruses[1] = new AdditivePolymorphicVirus(x1, x2);
12         viruses[2] = new SubtractivePolymorphicVirus(x1, x2);
13         viruses[3] = new MultiplicativePolymorphicVirus(x1, x2);
14
15         for (PolymorphicVirus virus : viruses) {
16             for (int i = 1; i <= 5; i++) {
17                 System.out.println(virus.execute());
18             }
19         }
20     }
21 }
```

Figure 3: The `main` method of the polymorphic virus test class.