SE433/333 Software Testing and Quality Assurance

Individual Assignment #2

Part 1 (Practice)

Description:

Fault prediction is necessary in software development life cycle in order to reduce the probable software failure and is carried out mostly during initial planning to identify fault-prone modules. Fault prediction not only gives an insight to the need for increased quality of monitoring during software development but also provides necessary tips to undertake suitable verification and validation approaches that eventually lead to improvement of efficiency and effectiveness of fault correction.

DesigniteJava is a tool for quality assessment of code written in Java.
It detects numerous architecture, design, and implementation smells that show maintainability issues present in the analyzed code. It also computes many commonly used object-oriented metrics.

Downloads:

- You can find DesigniteJava with the description in: http://www.designite-tools.com/designitejava/enterprise
    - You can request for academic license from here (you should receive the License key in 24-48 hours, so do not wait for the last day before deadline to request a license) : http://www.designite-tools.com/acad-lic-request/
    - To execute DesigniteJava follow the instruction in Usage section (in the website)

Tasks:

- Search for an open source project (Java) (a minimum of 50 000 LOC)). (You can use the same project from assignment . (only one version (release) is needed)
- Compare between DesigniteJava and Understand (the tool from assignment 1) : based on the CK metric results (for example, if the results of metrics between the tools for the same class are different, try to explain why and which tool provides accurate measures according to you), Compare between the features of the tools (advantage/drawbacks of using each tool)
- Recommend a set of useful refactoring operations to fix two types of code smells (antipatterns) identified by DesigniteJava. It is not required to provide a source code implementing the refactoring operations. You can provide the name of the sequence of refactoring operations along with their parameters such as *movemethod(m1, Class A, Class B)* as a solution to fix an antipattern.

- Add a screenshot from the command prompt (batch) and also a screenshot from the output directory in the report

Part 2

Exercise 1 : Write a Java code to implement the following class diagrams

Diagram 1

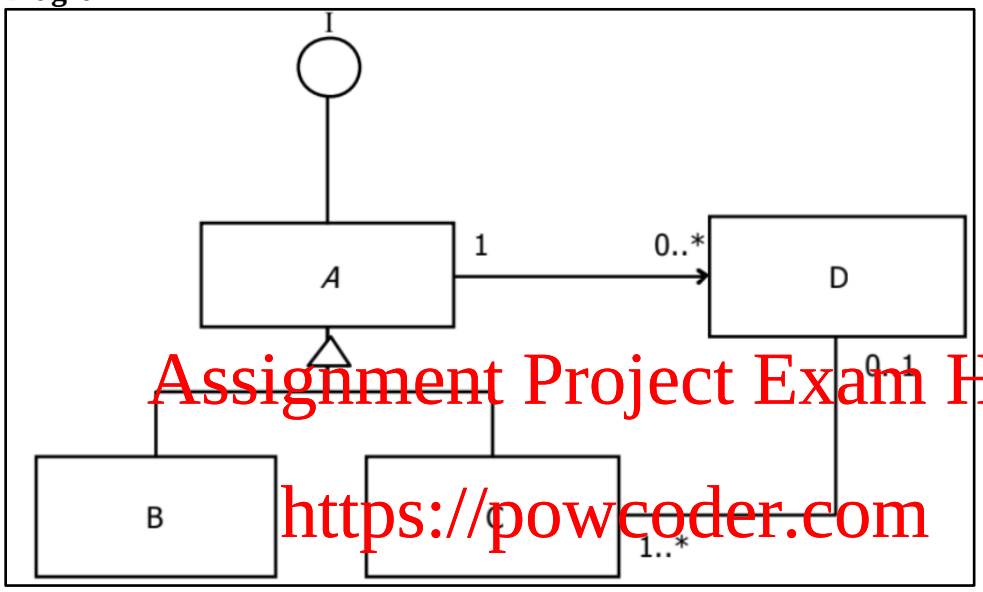
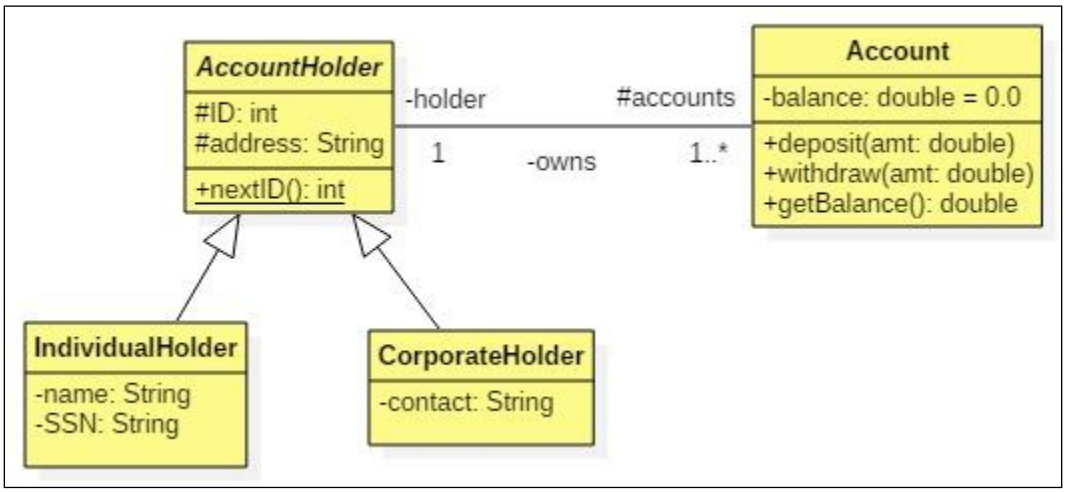
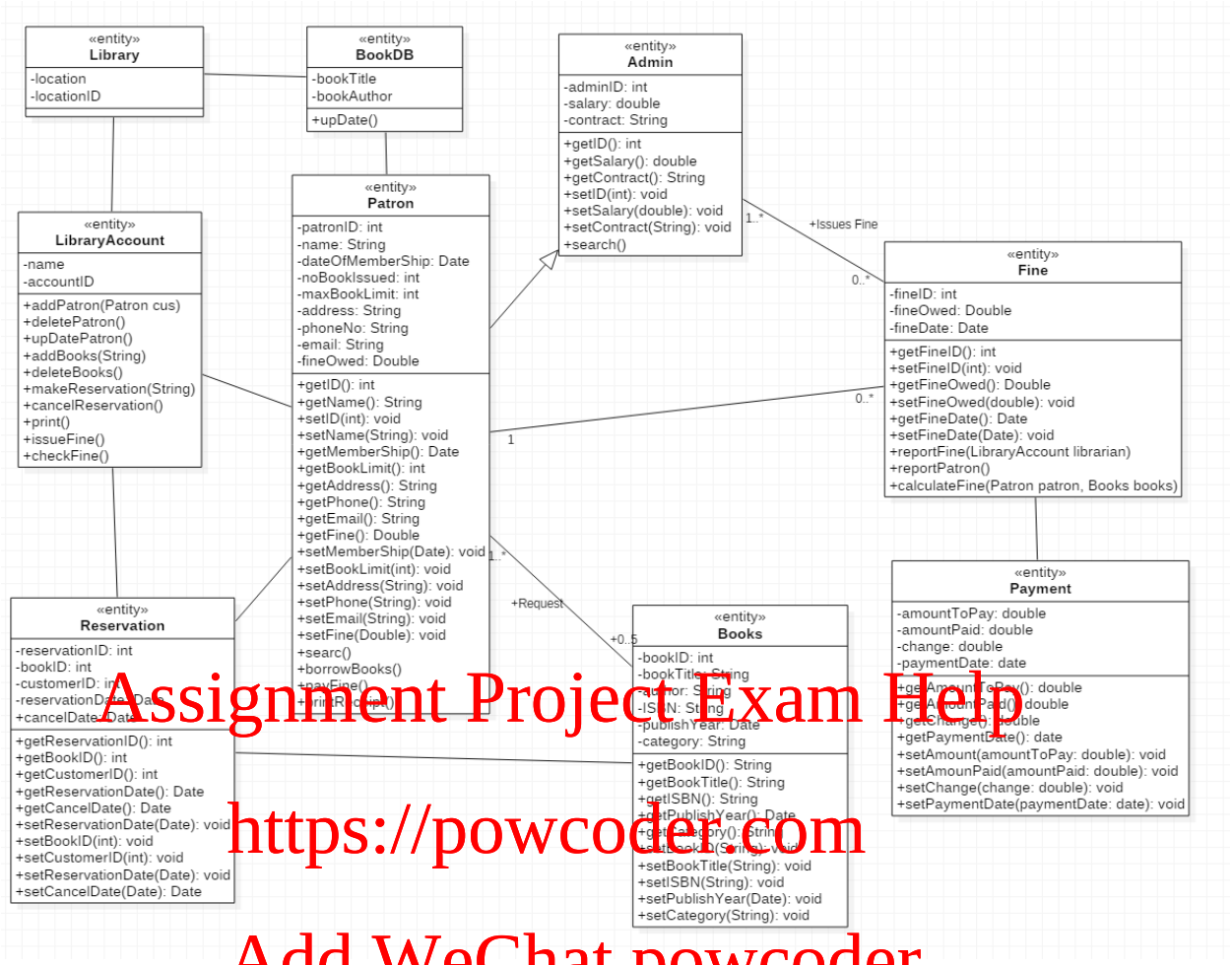


Diagram2



Exercise 2:

A- Identify two code smells (antipatterns) in the following diagram



B-

C- Recommend a set of useful refactoring to fix the two types of design antipatterns identified in Exercise 2-A. It is not required to provide a source code implementing the refactoring operations. You can provide the name of the sequence of refactoring operations along with their parameters such as movemethod(*m1*, Class A, Class B) as a solution to fix the design antipatterns.

Submission:

You need to prepare a pdf document that contains your work (part 1 and part 2). Upload the pdf document in the appropriate folder on D2L. (There is no need to upload the excel documents generated by the tool)

Deadline : Feb 13, 2021 11:59 P.M