Syarif Hidayatullah
Unibas-Sweng

Exercise 1:

- Examples of separately defined Interfaces in GanttProject:
  There are a lot of separately defined interfaces in GanttProject. I picked two Interfaces as examples. The first Interface is the UIFacade Interface which is implemented –among others– in UIFacadeImpl class. The second Interface is the Document interface which is implanted – among others– in FileDocument class.

- Short description of how this separation works
  As explained in the Java doc, interface form a contract between the class and the outside world. If a class implemented an Interface, it is enforced to implements all methods defined in that interface. It means that your code will not be compiled successfully otherwise.

  Example: The method getFileName is defined in Document Interface. The class FileDocument implemented this Interface. Therefore all the method defined in the Document Interface are implemented in this class, including the getFileName method. The implementation method is marked by the @override annotation (not necessary).

  To use the getFileName method, we need a reference of the implementing class. Either as an Object which is declared globally like in ProxyDocument class or as a Paramter like in open method in GanttProject class.

- Possible reasons why it is useful to do so
  It is easier to maintain your code. If you have to change the implementation of a function. You can do it in one place (implementing class). The end user of the implementing class does not have to know how your new implementation looks like. What they should interested in, is only what functionality this class has to offer, and they can have this information simply by reading the interface.

Exercise 2:

- Pros & cons of using Optional
  - No more NullPointerException at run-time
  - No more Null checks, therefore less boilerplate code.
  - Catching an exception is generally an expensive process, therefore it should only be used under exceptional circumstances
  - Your code can be harder to read.
  - You have to write more code (as a price of less documentation)

- Pros & cons of using Exception
  - I think the only benefit of using exception to handle nullpointerexception is that your code more readable for average java coder.

- When to use which method
  Use Exception only in an exceptional circumstances

- Which method is better in this particular case
  My answer can be biased here. I am not very familiar with this new feature but in my opinion try catch block suits better in this particular case. The main reason here is that you have to put BufferedReader in a try catch block anyway, with or without Optional.

Exercise 3:

- What is hidden in ADT
  The detail implementation of the functionalities are encapsulated.

- State management of each implementation
  In OOP the state of the stack is managed by the attribute of the stack class.

- Why the OO implementation looks so natural for us?
  Because it is easier for us to think of a Stack as an Object.

- Advantages of OOP vs functional programming and vice versa
  OOP is good when you have a fixed set of functionalities on an Object. You can add new Object by implement existing functionality and add some special functionality to the new Object, without changing the existing Objects.

  The functional approach is good if you have a fixed set of Objects. When your application grows, you can add functionalities to an existing object, without changing the existing functionalities of this object.

Exercise 4:

- Which problems are solved by the new module system of Java 9
  - Modular JDK. Now small devices with limited access of RAM can also use Java.
  - REPL (Read Eval Print and Loop) within a JShell. Now you can write one line of code in a console and directly evaluate it. Similar like what we have in Python. Now it is easier to teach Java. You can code “hello World” without explaining what is public static void main.
  - Enhanced modular programming to improve security. You can now specify an internal module which is only accessible to privileged classes.

- What should be specified in requires clause
  In requires clause, we have to specify which modules we need to be able to run our code. This is similar to require syntax in JavaScript.

- What should be specified in export clause
  Like in JavaScript a module is only available to another module if we explicitly export that module. In java 9 the export clause is used to specify, which module we want to make available for the outer world.

- When is it useful to specify use and provides clauses
  It is useful if we want to let an Implementation of certain module be available for another module at the start time. Each “uses” muss be handled by at least one “provides” otherwise the JVM will not started.