Week 4

Refactoring in IntelliJ (lecture)

1. What is refactoring?
   Refactoring consists of improving the internal structure of an existing program's source code, while preserving its external behaviour.
2. Give at least three examples of things that we might refactor in the case study code. For each, make note of how you can accomplish this in IntelliJ.
   i. Change the constants in class Constants to private and use a getter to get them: Refactor → Encapsulate Fields...
   ii. Rename some variable to another in the whole project: Refactor → Rename...
   iii. Extract some lengthy code into a function: Refactor → Extract Method

Java Recap

3. Make sure you study for the Java graded quiz this week. If you have completed all of the readings, quizzes, and exercises then you should find it straightforward. Take notes on anything from the slides this week during lecture about Java that you may have not seen previously.

Generics

4. What does the syntax mean?
   The type should be a child class that inherits from parent class Parent.
5. Give an example of a possible method signature which demonstrates how generics can enforce a relationship between method parameters.

```
class myClass<T>{
   public void addItemToList(T item, List<T> list){
       // in the method, we can only add item of type T or its children
       // into the list that only contain type T objects
   }
}
```

Exceptions

6. What class is at the top of the hierarchy of all Error and Exception classes in Java?

Throwable

7. What is the difference between an Exception and a RunTimeException?
   RuntimeException inherits from class Exception, so it is a specific type of exception in the Exception class. A RuntimeException usually occurs when the given input causes unexpected behaviours in the program. e.g a division by zero.
8. Find an example in the case study code where an exception is thrown and an example where an exception is caught.
   In the History class, an exception is thrown when the number of commands is less than zero. It is caught only in a test testExecuteCommandHistory() since the exception is too general (class Exception).

Packages

9. What is the purpose of using packages?
   A package in Java is used to group related classes. Think of it as a folder in a file directory. We use packages to avoid name conflicts, and to write a better maintainable code.
10. Describe at least two ways you might use packages when you work on your course project this term.
    i. Say we want to implement different controllers for Phase 0 and Phase 2. We can name those controller classes with a same name and place them into two different packages. Whenever we need to reference the class, we can specify the import with the correct package name and will not cause any conflicts. I am worried that might not be a good naming convention though.
    ii. To better organize our code, we can place all related components into a same package. That way the readers will be able to understand our code more easily.