Below are key points of what was discussed in the lesson, which was based on the principles outlined in Items 58 & 65 of Effective Java book.

Note: It is ***highly recommended*** to buy the latest edition of **Effective Java book by Joshua Bloch**. It is one of the must-have books for any serious Java programmer!

Item 58: Use checked exceptions for recoverable conditions and runtime exceptions for programming errors

Key Points:

- **Should I use Checked or Unchecked?** An important rule in deciding to go with a checked exception would be:

    a) that it happened due to some *exceptional situation and not programming error* and

    b) caller can be expected to reasonably recover from it

- We know that unchecked exceptions can be either runtime exceptions (subclasses of RuntimeException) or errors (subclass of Error). Runtime exceptions should be generated in the event of programming errors. Mostly it would be precondition violations, which would be failure of the API client to follow the API contract. Generally, runtime exceptions should not be caught. Errors are JVM related like resource deficiencies like running out of memory. Such errors are expected to be generated by JVM and hence there is a very strong convention not to subclass Error class. You should ***almost never*** catch an error. Hence, any unchecked exceptions we create should be subclasses of RuntimeException class or one of its subclasses

- Note that checked exceptions need not always indicate recovery in a programmatic fashion from the callers end. Sometimes, the recovery can be done by the end user who is using the system. For instance, let us say a user is trying to purchase something with his credit card and the credit card could not be processed successfully. It could be due to any number of reasons, but assuming the API that processes credit card information gives us the specific reason, then we can generate a checked exception and encapsulate that information within

the exception itself (capturing failure reason will be discussed in next lesson). This was calling code can extract the failure reason (e.g., incorrect expiry date) and display it to the user in UI, which could allow the user to correct the information provided.

Item 65: Don’t ignore exceptions

Key Points:

- A very common mistake is to ignore exceptions by using empty catch blocks. If the API designer is throwing a checked exception, then catching it with an empty catch block implies that you are ignoring the exceptional situation, which the item relates to a fire alarm.

- Similarly, if an empty catch block is catching an unchecked exception, then it is equally dangerous as the program might continue to work silently, but might result in some other very serious errors.

- At the very least, it is recommended to include a comment indicating why it is okay to ignore the exception.