Course 4 - Sprint 2: Create and Implement User Defined Exception

Learning Consolidation

1. Following are some of the best practices related to Java Exception handling:
   a. Always Use Specific Exceptions for ease of debugging.
   b. Throw Exceptions Early (Fail-Fast) in the program.
   c. Catch Exceptions late in the program, let the caller handle the exception.
   d. Always log exception messages in Loggers for debugging purposes.
   e. Use multi-catch block for cleaner code.
   f. Use custom exceptions to throw a single type of exception from your application API.
   g. Use proper naming convention, exception object should end with Exception keyword.
   h. Exceptions are costly, so throw it only when it make sense. Else you can catch them and provide a null or empty response.
   i. Use Java 7 ARM feature to make sure resources are closed or use finally block to close them properly.

2. What is the difference between the throw and throws keyword in Java?
   a. throws keyword is used with method signature to declare the exceptions that the method might throw whereas throw keyword is used to disrupt the flow of the program and handing over the exception object to runtime to handle it.

3. What is the Problem with the below code?

```
public class ExceptionDemo
{
        public static void main(String[] args)
        {
                try
                {
                    demoExceptions();
                }
        catch  (FileNotFoundException|IOException e)
                {
                    e.printStackTrace();
                }
        }
    public static void demoExceptions() throws IOEx
ception, FileNotFoundException
    {
    }
}
```

4. What is the problem with the below code?

```
public class Parent
{
    public void startCode() throws IOException{}
    public void foo() throws NullPointerException{}
}
public class Child extends Parent
{
    public void startCode() throws Exception{}
    public void foo() throws RuntimeException{}
}
```

5. What does JVM do when an exception occurs in a program?
    a. When JVM faces an exception in a program, it creates an exception object and throws it to program informing us that an error has occurred. If the exception object is not caught and handled properly, JVM will display an error message and will terminate the rest of the program abnormally.