2. Checked Exceptions: checked at compile-time. Examples: ‘FileNotFoundException’ when opening a file. Unchecked Exceptions: checked at runtime. Examples: ‘Stack OverFlowException’ in a loop.

3. No, there can be only one final block. Final block is executed after all the catch block checks the exception.

4. The final block contains the final return value, regardless of the return value from the catch block.

5. Runtime exceptions are checked during runtime. The compiler doesn’t detect the errors during compile time. It’s usually something wrong with the logic of the code. For example, null pointer execution, array index out of bound exception.

Compile-time exceptions are checked at compile-time. It’s usually more critical because the code wouldn’t start running. For example, file not found exceptions, class not found exception.

6. Throw is used inside the function, whereas throws is used in the function signature. Throw can only be used in one specific exceptions. But throws can be used for multiple exceptions.

7. In the first block of the code, ArithmeticException is caught because you can’t divide anything by zero. In the second block of the code, Null pointer exception is caught because string s is null. In the third block of code, runtime exception is thrown cause it’s in the try block, and nothing else is there, so it will always happens. ‘Exception’ is put at the end due to exceptions hierarchy. The most general types of executions should always be put into the end.

8. Optional is a container object in Java 8. It gives the options that the object can be null or non-null. Example:

Public Optional<String> getValue(String key) {

Return Optional.ofNullable(Data.get(key));

}// it’ll return a value if present, or an empty ‘Optional’ if the key is not in the map.

8. The finally block is designed to be always executed. After the try-catch blocks, finally block will always run no mater what.

10. Java 8 new features: Lambda expressions(anonymous functions). Optional. Streams API(filter, map, reduce, allowing faster process of a collections of elements), etc.

11: design patterns in java: aggregation: the lifetime of the owned object does not depend on the lifetime of the owner, the whole can exist without the parts. Composition: the lifecycle of the owned object depends on the lifetime of the owner. Singleton: Ensures a class has only one instance and provides a global point of access to that instance, the advantage is that it’s thread safe and resource efficient since it’s loaded only during usage.

12. SOLID: single responsibility principle: a class should have only one reason to change. Open/closed Principle: explained later. Liskov Substitution Principle(LSP): subtypes must be substitutable. Interface Segregation Principle: Clients should not be forced to depend on interfaces they do not use. Dependency Inversion Principle: High-level modules should not depend on low-level modules. Both should depend on abstractions.

13. To achieve thread safe singleton pattern, we can do Eager Load by creating private static variable, getters and private constructor. Or we can do a lazy load, creating a static inner class, ensuring the instance is created only when it’s accessed. Or we can just use the enum keyword. This guarantee the instance is thread safe.

14. Open/Closed Principle: open for extension, you can add new code on top of the existing code. Closed for Modifications, you shouldn’t modify the source code provided to you.

15. 1. “If object A can do something, object B could also do the same thing”. If B object can substitute A object, then it should be simply able to perform all the actions of type A.