EXCEPTIONS HANDLING

Exception is an exceptional event that occurs during execution of program, that disturbs the normal flow of program.

Exception is an abnormal behavior of an application, occurs when the execution time and terminates the flow of execution if not handled. Exceptions may be database connection errors, network connection errors, file not found error, file corrupted error and so many.

1. What is an exception object and throwing an exception?

When an error occurs within the method, the method creates an object and handles it to the run time to handle, the object is called exception object. Which contains information about the exception.

The object is created and handles (delegates) it to the run time system to handle is called throwing an exception.

After the method throws an exception the run time system looks for the caller of the method to handle, those methods are called call stack.

2. How to handle exceptions?

Exceptions handled by try, catch, and finally blocks.

Try: Try block is used to enclose the code that might throw an exception.

Catch: The block catches all the exceptions that have occurred in the try block. Each catch block is an exception handler that handles the exceptions. It can contain specific type of exception to handle that occur in the try block or subclasses of the exception type.

Finally*:* this block always performs irrespective of exception is catching or not.

3. Difference between error and exception?

Errors typically happen when an application is running while the exception is occurred by the application.

Exception	***Error***
Belongs to java.lang package	Belongs to java.lang
Exceptions are created by application itself.	Errors are occurred when application is running.
Compiler will check for checked exceptions whereas it will not care about the un-checked exceptions.	Compiler will not check errors because they are occurring at run-time.

4. What is the use of exception handling?

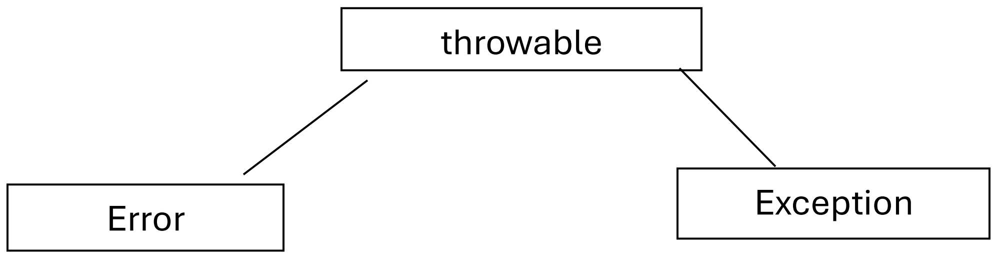
If there is no exception handling, then the application execution terminates. To avoid this, exception handling is necessary.

- Avoids the unexpected termination of program.
- Provides flexibility to the programmers to handle exceptions thereby making application more robust and immune to unexpected errors.
- Programmers can understand the reason for the termination of applications.

5. Types of exceptions in java?

There are two types of exception are present checked and un-checked exceptions.

The parent most of the exception is throwable.



1)Checked exceptions:

Occurs during compilation time, compiler will check whether the exception is handled, if not then throws an error accordingly.

2)Unchecked exceptions:

Occurs during application execution time. Compiler will not check whether the exception is handled or not. It is compiled and throws exception at the run time. All the subclasses of java.lang.RuntimeException class and java.lang.Error classes belong to runtime exceptions.

6. Briefly explain the usage of throw and throws keys?

Both throw and throws are the keys of exception handling.

throw: The throw key is used to explicitly throw an exception inside the method or block of code. The exception is handed over to the run time to handle it.

throws: The throws key is used to delegate or declare which exceptions are thrown by the method. Is used with method signature specifies that method may throw an exception.

7. What is the difference between final, finally and finalize keywords?

FINAL:

- Final is a keyword used with variables, methods, and classes, when variable is marked as final it is like a constant it cannot modify in future.

- If the method is final, then it cannot be inherited by the subclass means final methods are not overridden.
- If the class is final then it does not allow inheritance, classes cannot inherit the final classes.

FINALLY:

- The finally is a keyword used in the exceptions.
- It executes the statements irrespective of exception is handled or not.

FINALIZE:

- The finalize method is called by the java garbage collector when the object is no longer needed.

8. What are the kinds to handle exceptions?

There are multiple kinds of catching the exceptions.

- try with catch
- try only
- try with finally
- try, catch, and finally
- try with multiple catch
- try with multi catch
- try with resource

9. How to handle checked exceptions?

Checked exceptions are also called compile time exceptions; compiler will check whether the exception is handled or not. If not handled properly the program will not be compiled. Can be handled by try-catch block or throws clause in the method declaration.

10. Differentiate between checked and unchecked exception?

Checked exceptions	***Unchecked exceptions***
They are checked by the compiler.	They are not checked by the compile.
Until and unless exception is handling the compilation of the program is not possible.	Compilation may possible but exception may arise during runtime if exception is not handled properly.
Checked exceptions are subclasses of Exception class.	They are subclasses of RuntimeException class.
Ex: IOExceptions, FileNotFoundException etc..	Ex: ArithmeticException, NullPointerException etc..

11. What is stack trace and how it is related to exception?

Stack trace is information consisting of names of classes and methods that were invoked right from the start of the program execution to the point where an exception occurred. This is helpful to know where the exception occurred and reason for exception.

12. What are the methods of the Exception class?

Exception class several methods below they are:

getMessage() :- returns a string that describes the exception.

printStackTrace() :- prints stack trace of the exception to the console.

equals() :- compares two exceptions for equality.

toString() :- returns string representation of the exception.

clone() :- creates clone or copy the exception.

hashCode() :- returns hash code for the exception.

getCause() :- returns cause of the exception if available.

getStackTrace() :- returns an array of StackTarceElement objects that represents the stack trace of the exception.

initCause() :- initializes the cause of the exception.

13. What is exception chaining?

One exception is thrown due to another exception called exception chaining. This helps developers to identify what situations an exception was thrown that in turn cause another exception.

14. Explain unreachable catch block?

This error is thrown by the compiler when multiple catch blocks and keep parent classes first and subclasses later. The catch block should follow the order, the specific ones (subclasses) are at the top and general ones (parent classes) are at the bottom. If not followed, an unreachable catch block error is thrown during compile time.

15. Explain try with resource block?

The try-with-resource statement is used for resource management, the resource that is opened in the try block will be automatically closed when try block finishes.

In earlier versions resources are usually closed in the finally block.

```
try (resource initialization) {

}

catch () {

} // finally block in not needed.
```

The resource that is declared inside the parenthesis of the try block must implement the AutoCloseable interface or Closeable interface.

16. NullPointerException?

- It is a *Runtime Exception* (extends Runtime Exception).
- A special null value can be assigned to an object reference.
- When using the reference of that object throw a ***NullPointerException***.
- These include.
    - Calling the instance method of a null object.
    - Accessing or modifying the field (variable) of the null object.
    - Taking length of null as if it were an array.
    - Accessing or modifying the slots of null as if it were an array.
    - Throwing null as if it were throwable value.

17. How to resolve NullPointerException?

To avoid *NullPointerException*, we must ensure that all *objects are initialized properly* before their usage. When declaring a reference variable, we must ensure that object is not null before we access a field or method of the object.

18. Explain *Errors* in java?

Error is an ***illegal operation*** performed by the developer which leads to ***abnormal*** working of program. Errors remains undetected until the program is compiled or executed. Some of the errors will restrict the program getting compiled until they are not removed.

19. What are the types of errors?

There are 3 types of errors are there:

1. *Run-time Errors*
2. *Compile time Error*
3. *Logical Errors*

20. Explain Run-time Errors?

Runtime errors occur during the execution of the program. Sometimes these are discovered when the user enters the invalid data.

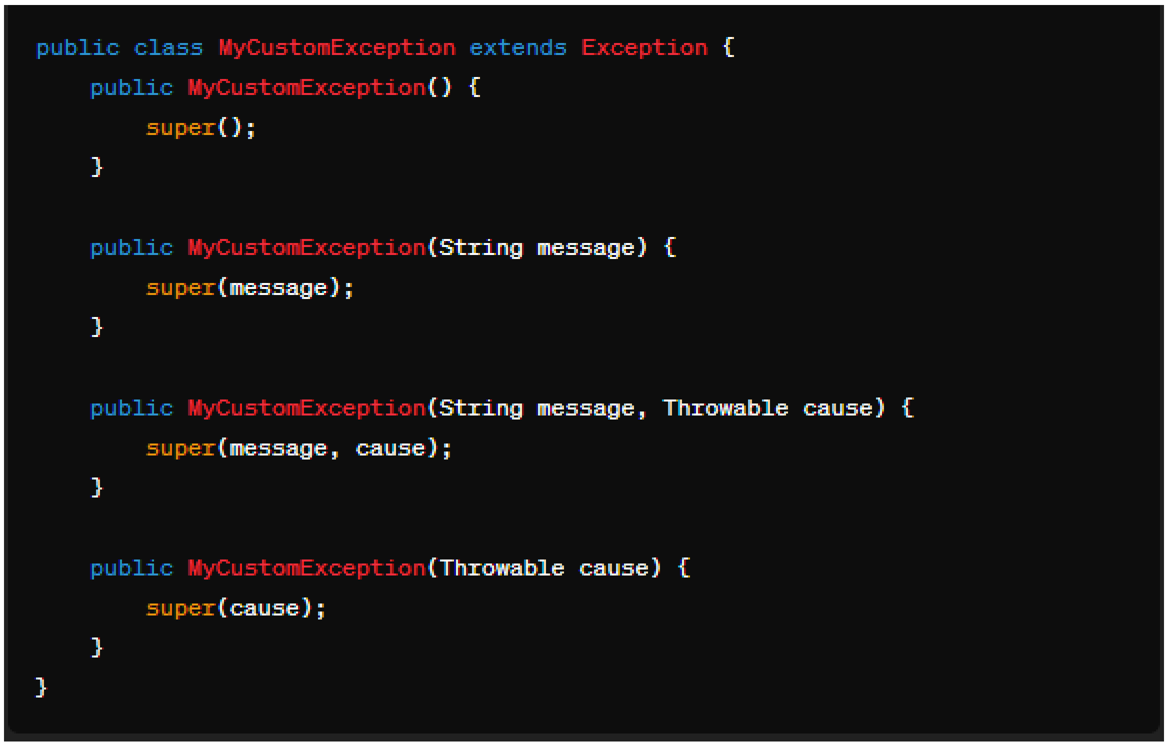
21. Explain *CloneNotSupportedException*?

If the object on which clone() method invoke implement the Cloneable interface. If the object does not clone, then is arises the exception called *CloneNotSupportedException*.

22. What are custom exceptions?

The custom exceptions are defined by the developer, rather than using the exception that are provided by the Java.

- Custom exceptions allow us to specify more convenient information for the application.
- Which can improve code readability, maintainability, and error handling.
- To create a custom exception, we typically define a class that must extends either Exception class or sub-classes of Exception class like RuntimeException class.
- Once the custom exception is created then it can be used to throw an exception.



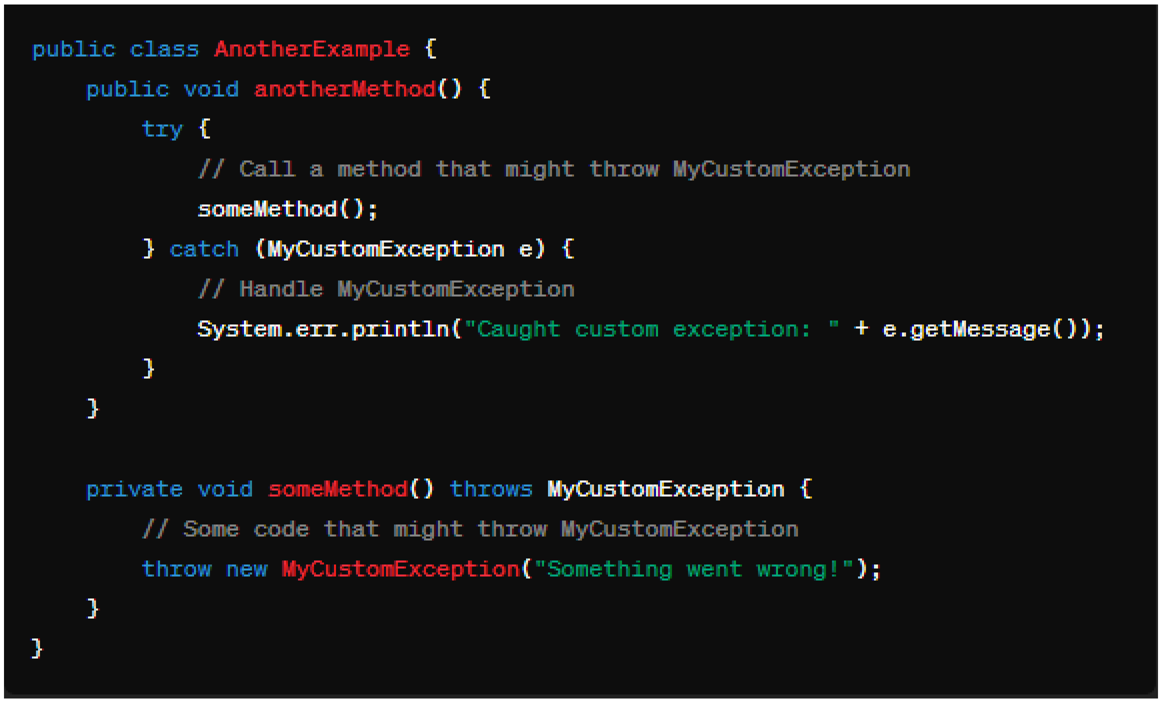
```java
public class MyCustomException extends Exception {
    public MyCustomException() {
        super();
    }

    public MyCustomException(String message) {
        super(message);
    }

    public MyCustomException(String message, Throwable cause) {
        super(message, cause);
    }

    public MyCustomException(Throwable cause) {
        super(cause);
    }
}
```

Creating a custom exception, that can contain the information about the exception or cause for the exception.



```java
public class AnotherExample {
    public void anotherMethod() {
        try {
            // Call a method that might throw MyCustomException
            someMethod();
        } catch (MyCustomException e) {
            // Handle MyCustomException
            System.err.println("Caught custom exception: " + e.getMessage());
        }
    }

    private void someMethod() throws MyCustomException {
        // Some code that might throw MyCustomException
        throw new MyCustomException("Something went wrong!");
    }
}
```

Throwing an custom exception inside the method body.

23. What are the best practices for handling exception?

- Catch specific exceptions, rather than general ones i.e. catching the specific exception than using Exception class.
- Catch exception at an appropriate level.
- Provide more informative error messages.
- Use finally block to release the resources that are used during the execution.