5 keywords

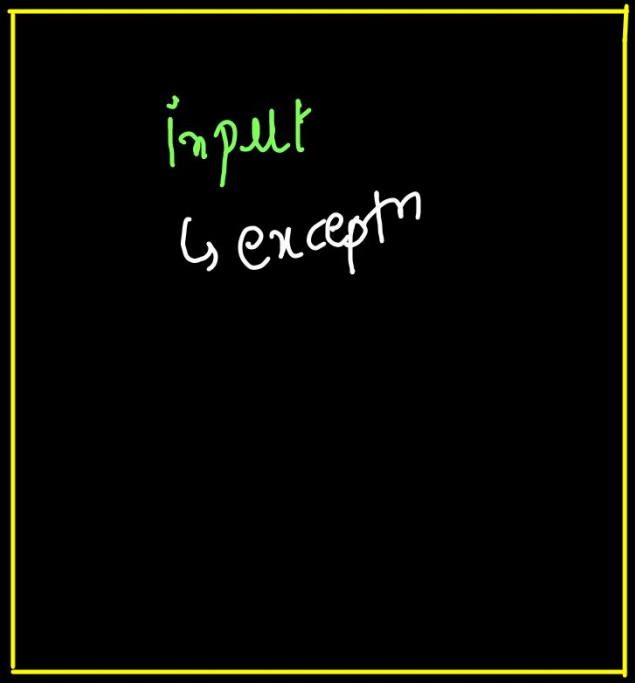
(1) try ⟶ risky code (chances of Excepn)

(2) catch ⟶ alternate flow (handling the exception)

(3) finally ⟶ cleanup code (FileInputOutput, I/O stream close, memory release)

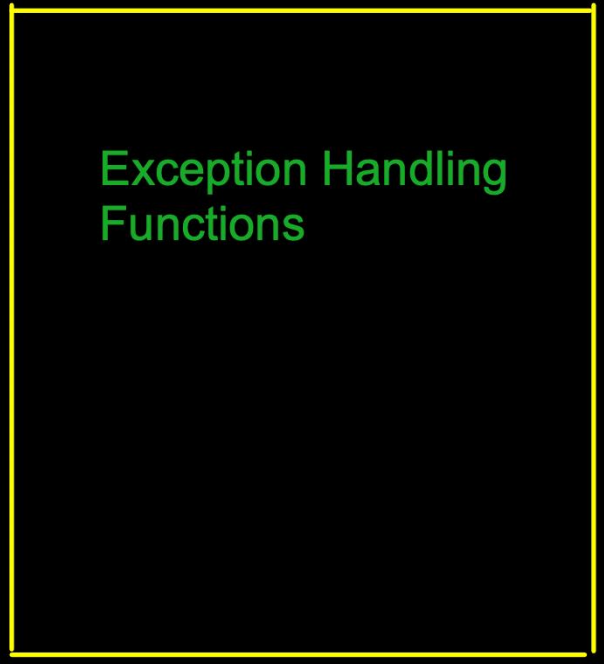
(4) throw ⟶ throw custom Excepn from the current fn to the calling fn!
↳ Runtime

(5) throws ⟶ calling fn gets to know called fn might throw Exception!
↳ checked Exception



Driver
(main)

Calculator



Exception Handler

```java
class Calculator {
    public int add(int a, int b) {
        return a + b;
    }

    public int subtract(int a, int b) {
        return a - b;
    }

    public int divide(int a, int b) {
        return a / b;
    }

    public int multiply(int a, int b) {
        return a * b;
    }
}
```

Business Logic (App)

throws ArithmeticException (Optional)

```java
class Driver {
    Run | Debug
    public static void main(String[] args) {
        Calculator calc = new Calculator();

        Scanner scn = new Scanner(System.in);
        int a = 0, b = 0;
        try {
            a = scn.nextInt();
            b = scn.nextInt();

            try {
                int res = calc.divide(a, b);
                System.out.println(res);
            } catch (ArithmeticException e) {
                ExceptionHandling.calculatorException();
            }

        } catch (Exception e) {
            ExceptionHandling.inputOutputException();
        }

    }
}
```

Driver Code (Client side)

```java
class ExceptionHandling {
    public static void inputOutputException() {
        System.out.println(x: "Please Provide Input Again");
    }

    public static void calculatorException() {
        System.out.println(x: "Division by 0 Not Allowed");
    }
}
```

Except Handler Servic

If your funcn where Runtime/Unchecked Excepn is occured does not handle it, it will be automatically thrown to the calling function.

If your funcn where checked/compiletime Excepn is occured does not handle it, it have to explicitly tell the caller funcn that I can throw a Excepn by using throws keyword.

checked / compile-time

```java
class FileInputOutput {
    public static void fileRead(String path) throws FileNotFoundException {
        // Checked / Compile Time Exception
        FileInputStream file = new FileInputStream(path);
        fileWrite(file);
    }

    public static void fileWrite(FileInputStream file) {
        System.out.println(x: "Performs Some Task on the File");
    }
}
```

```java
class Driver2 {
    Run | Debug
    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        String path = scn.nextLine();

        try {
            FileInputOutput.fileRead(path);
        } catch (FileNotFoundException e) {
            System.out.println(x: "Wrong Path Passed");
        }
    }
}
```

```
architaggarwal@Archits-MacBook-Air Java Advanced % javac A_06_ExceptionHandlingPart2.java
architaggarwal@Archits-MacBook-Air Java Advanced % java Driver2
d://abc.txt
Wrong Path Passed
architaggarwal@Archits-MacBook-Air Java Advanced % javac A_06_ExceptionHandlingPart2.java
architaggarwal@Archits-MacBook-Air Java Advanced % java Driver2
/Users/architaggarwal/Documents/Demo.txt
Performs Some Task on the File
architaggarwal@Archits-MacBook-Air Java Advanced %
```

```
vote(int age){
    if(age < 18)
        throw Excep^n

    Sys o("voted successfully");
}
```

Voting app

```java
// Custom Exception: Unchecked
class AgeInvalidException extends RuntimeException {
    public AgeInvalidException() {
        super(message: "Age Is Invalid");
    }

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

class VotingApp {
    public static void vote(int age) {
        if (age < 18)
            throw new AgeInvalidException();

        System.out.println(x: "Voted Successfully");
    }
}
```

```java
class Driver3 {
    Run | Debug
    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        int age = scn.nextInt();

        try {
            VotingApp.vote(age);
        } catch (AgeInvalidException e) {
            System.out.println(e);
        }
    }
}
```

```
architaggarwal@Archits-MacBook-Air Java Advanced % javac A_06_ExceptionHandlingPart2.java
architaggarwal@Archits-MacBook-Air Java Advanced % java Driver3
15
AgeInvalidException: Age Is Invalid
architaggarwal@Archits-MacBook-Air Java Advanced % java Driver3
25
Voted Successfully
```

Difference Between Checked Exception and Unchecked Exception

Checked Exception / Compile Time Exception	Unchecked Exception / Runtime Exception
1. Checked Exceptions are the exceptions that are checked and handled at compile time.	1. Unchecked Exceptions are the exceptions that are not checked at compiled time.
2. The program gives a compilation error if a method throws a checked exception.	2. The program compiles fine because the compiler is not able to check the exception.
3. If some code within a method throws a checked exception, then the method must either handle the exception or it must specify the exception using throws keyword.	3. A method is not forced by compiler to declare the unchecked exceptions thrown by its implementation. Generally, such methods almost always do not declare them, as well.
4. A checked exceptions occur when the chances of failure are too high.	4. Unchecked exception occurs mostly due to programming mistakes.
5. They are direct subclass of Exception class but do not inherit from RuntimeException.	5. They are direct subclass of RuntimeException class.

final

keyword

1. const variable
2. final class cannot be extend
3. final method cannot be overided.

finally

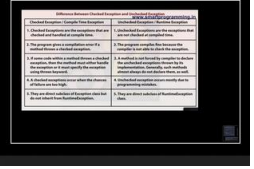
block

clean up code
(memory release, I/o stream close)

finalize

method (of object class)

Destructor
(release resources from an object to be deallocate)



Difference between throw and throws keyword

throw keyword	throws keyword
1. throw keyword is used to create an exception object manually i.e. by programmer (otherwise by default method is responsible to create exception object)	1. throws keyword is used to declare the exceptions i.e. it indicate the caller method that given type of exception can occur so you have to handle it while calling.
2. throw keyword is mainly used for runtime exceptions or unchecked exceptions	2. throws keyword is mainly used for compile time exceptions or checked exceptions
3. In case of throw keyword we can throw only single exception	3. In case of throws keyword we can declare multiple exceptions i.e. void readFile() throws FileNotFoundException, NullPointerException, etc.
4. throw keyword is used within the method	4. throws keyword is used with method signature
5. throw keyword is followed by new instance	5. throws keyword is followed by class
6. We cannot write any statement after throw keyword and thus it can be used to break the statement	6. throws keyword does not have any such rule