Exception Handling

What is **Exception**?: An unwanted unexpected event that disturbs normal flow of the program is called exception.

What is the meaning of exception handling?

- Exception handling doesn't mean repairing an exception.
- We have to define alternative ways to continue the rest of the program normally.
- This way of "Defining alternatives is nothing but exception handling".

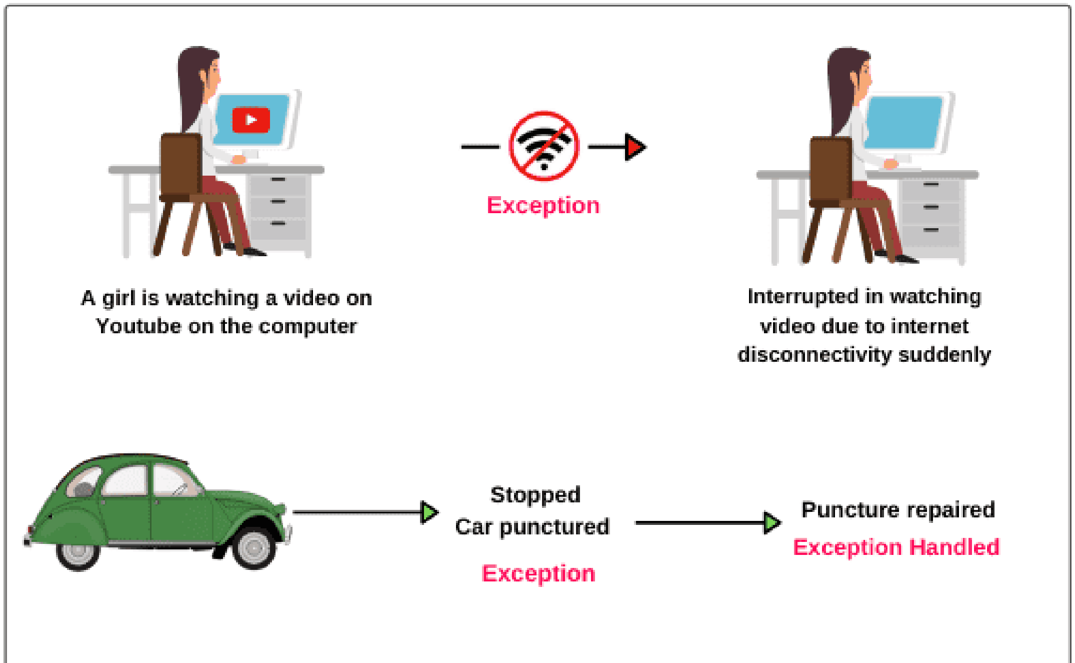
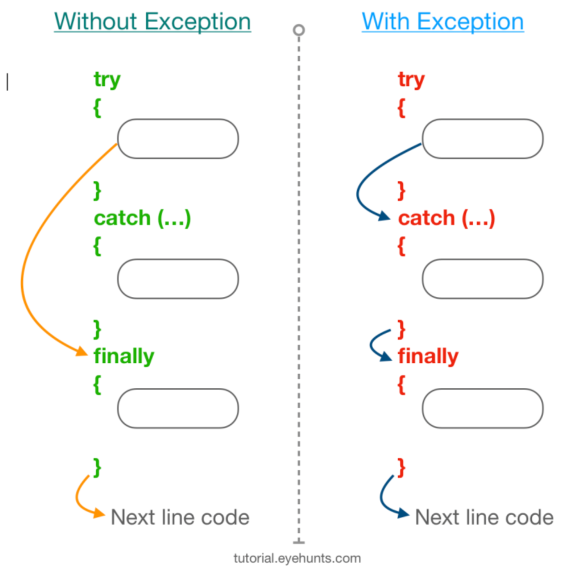

A girl is watching a video on Youtube on the computer
Exception
Interrupted in watching video due to internet disconnectivity suddenly
Stopped
Car punctured
Exception
Puncture repaired
Exception Handled

Fig: Realtime Example of Exception Handling

```
public class ExceptionHandlingExample {
   public static void main(String[] args) {
       try{
          // Open abc.txt file from C drive
       } catch (Exception e) {
           // Open abc.txt file from D drive
       }
   }
}
```

- **try block** – It contains the application code like reading a file, writing to databases, or performing complex business operations.
- **catch block** – It handles the checked exceptions thrown by try block as well as any possible unchecked exceptions.
- **finally block** – It is an optional and typically used for closing files, network connections, etc.



Internal flow when Exception occurred:

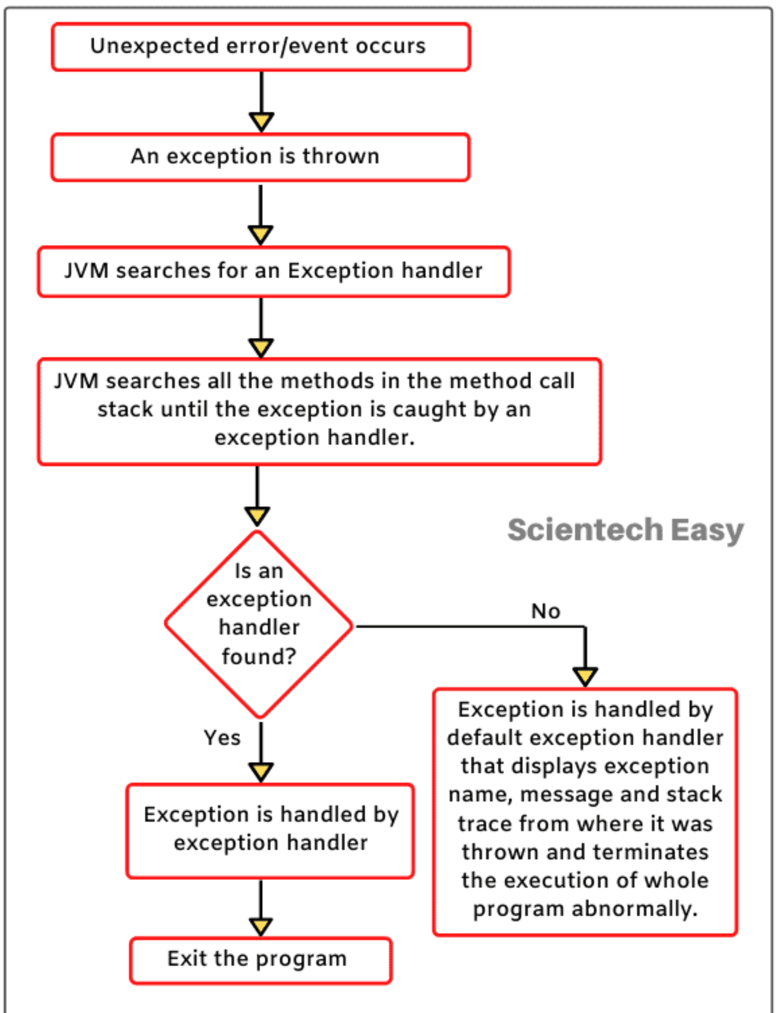
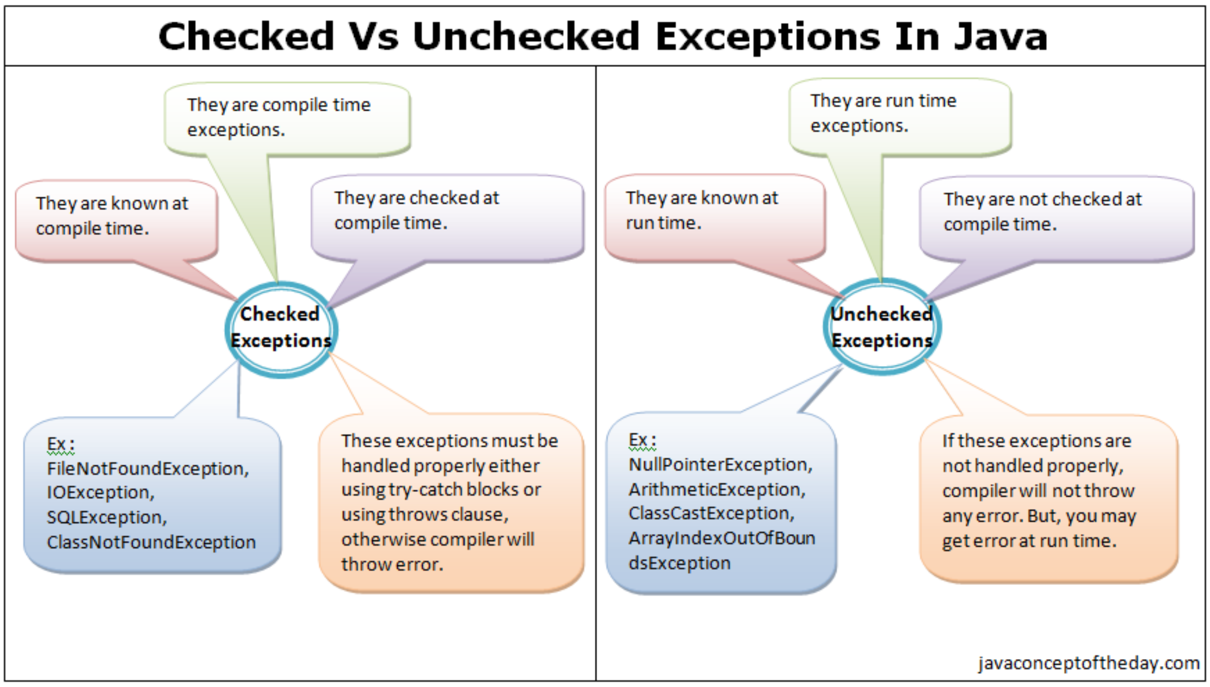
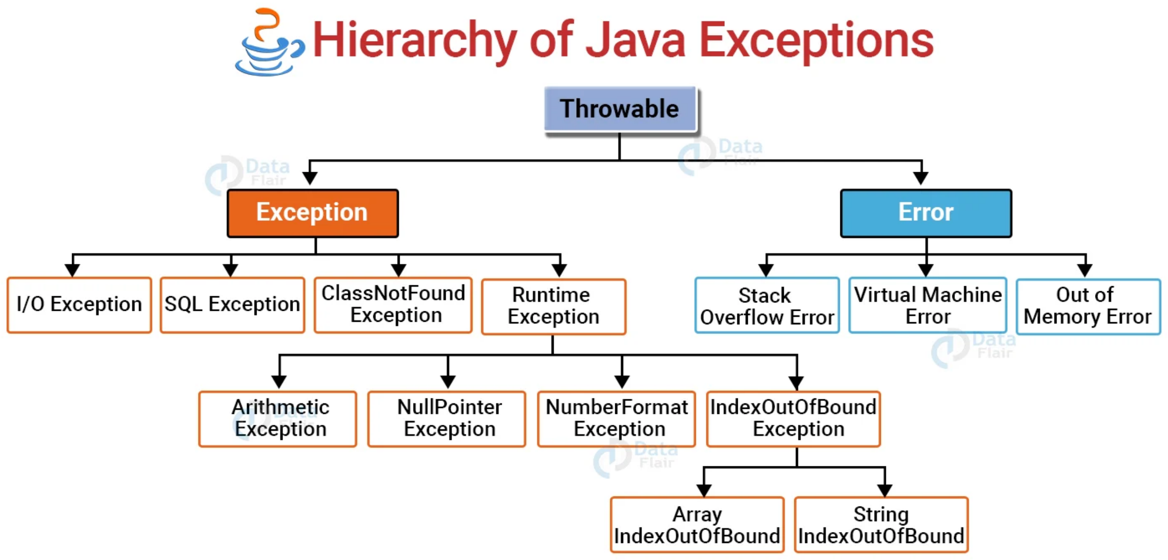

Unexpected error/event occurs
An exception is thrown
JVM searches for an Exception handler
JVM searches all the methods in the method call stack until the exception is caught by an exception handler.
Scientech Easy
Is an exception handler found?
No
Yes
Exception is handled by exception handler
Exit the program
Exception is handled by default exception handler that displays exception name, message and stack trace from where it was thrown and terminates the execution of whole program abnormally.

Fig: Exception handling mechanism

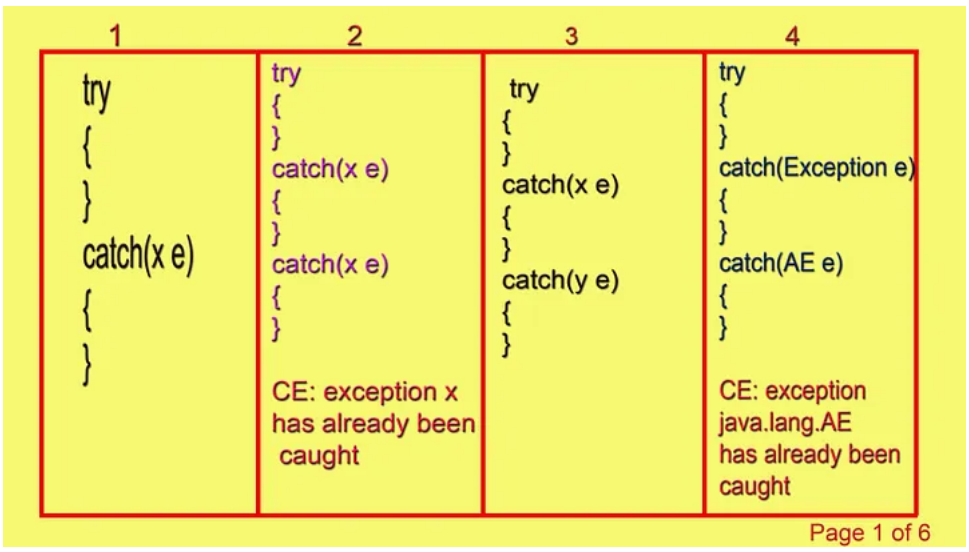
Checked vs Unchecked Exception:


Checked Vs Unchecked Exceptions In Java
They are compile time exceptions.
They are known at compile time.
They are checked at compile time.
Checked Exceptions
Ex: FileNotFoundException, IOException, SQLException, ClassNotFoundException
These exceptions must be handled properly either using try-catch blocks or using throws clause, otherwise compiler will throw error.
They are run time exceptions.
They are known at run time.
They are not checked at compile time.
Unchecked Exceptions
Ex: NullPointerException, ArithmeticException, ClassCastException, ArrayIndexOutOfBoundsException
If these exceptions are not handled properly, compiler will not throw any error. But, you may get error at run time.
javaconceptoftheday.com

Exception Hierarchy in Java:



try-catch-finally combinations possible in Java:



1

```
try
{
}
catch(x e)
{
}
```

2

```
try
{
}
catch(x e)
{
}
catch(x e)
{
}
```

CE: exception x has already been caught

3

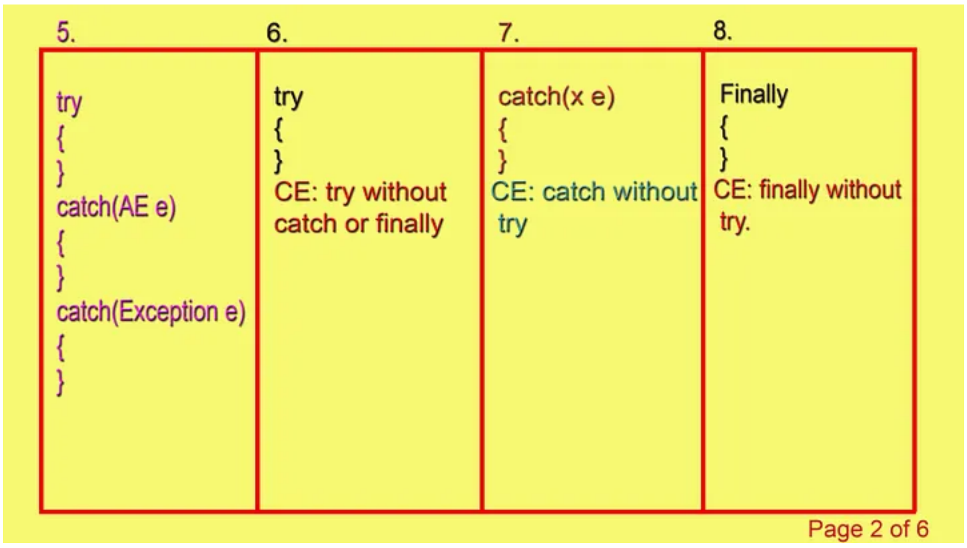
```
try
{
}
catch(x e)
{
}
catch(y e)
{
}
```

4

```
try
{
}
catch(Exception e)
{
}
catch(AE e)
{
}
```

CE: exception java.lang.AE has already been caught

Page 1 of 6



5.

```
try
{
}
catch(AE e)
{
}
catch(Exception e)
{
}
```

6.

```
try
{
}
```

CE: try without catch or finally

7.

```
catch(x e)
{
}
```

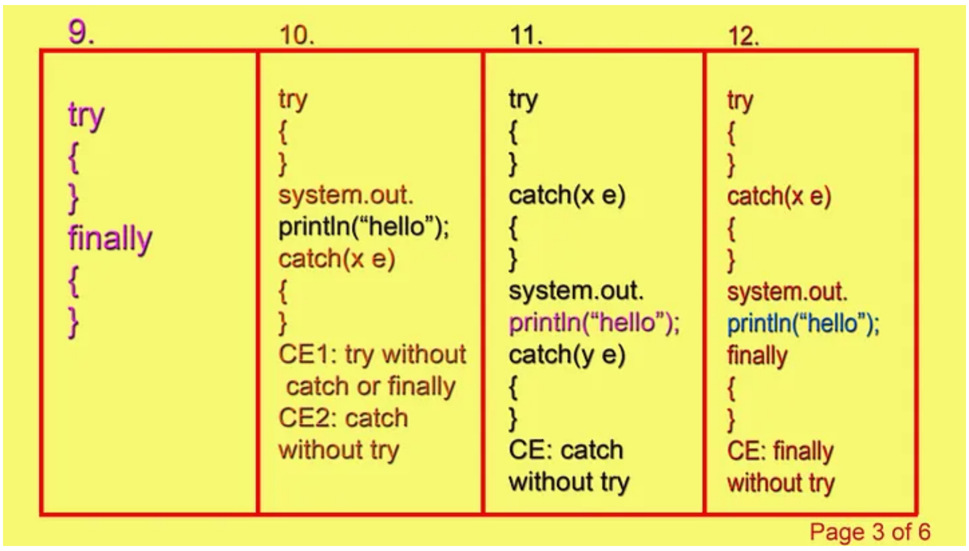
CE: catch without try

8.

```
Finally
{
}
```

CE: finally without try.

Page 2 of 6



9.

```
try
{
}
finally
{
}
```

10.

```
try
{
}
system.out.
println("hello");
catch(x e)
{
}
```

CE1: try without catch or finally
CE2: catch without try

11.

```
try
{
}
catch(x e)
{
}
system.out.
println("hello");
catch(y e)
{
}
```

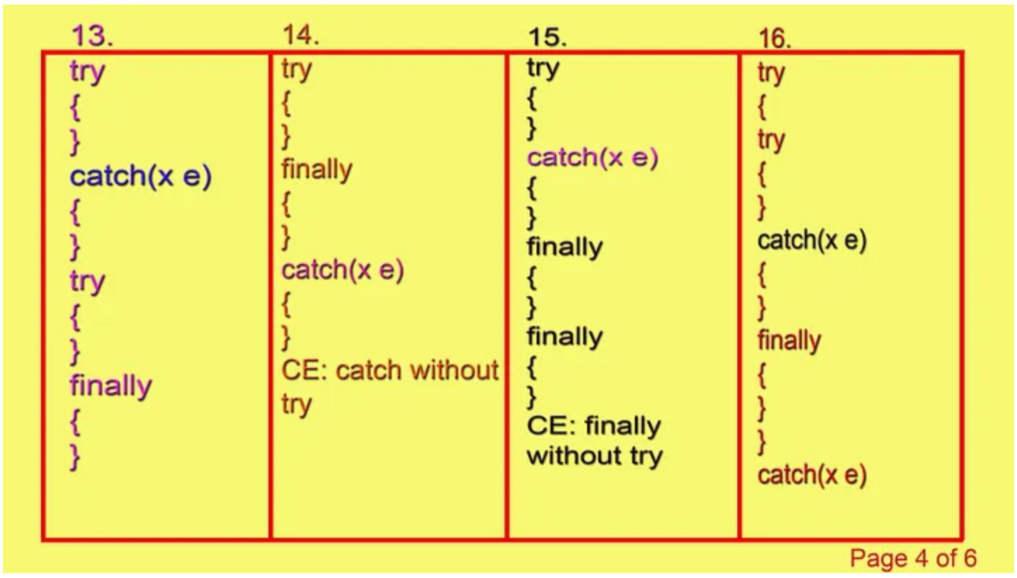
CE: catch without try

12.

```
try
{
}
catch(x e)
{
}
system.out.
println("hello");
finally
{
}
```

CE: finally without try

Page 3 of 6



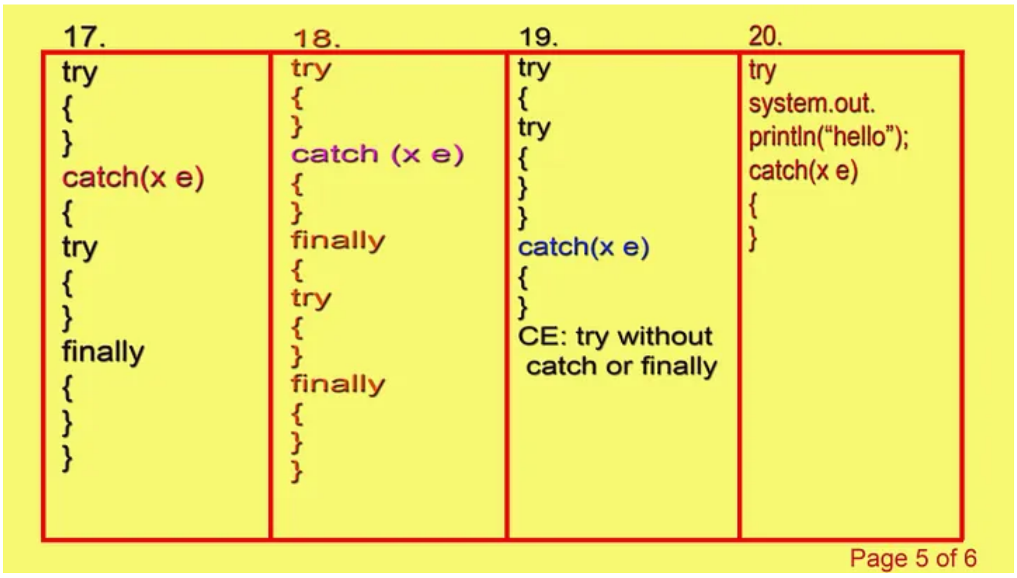
13.
```
try
{
}
catch(x e)
{
}
try
{
}
finally
{
}
```

14.
```
try
{
}
finally
{
}
catch(x e)
{
}
CE: catch without
try
```

15.
```
try
{
}
catch(x e)
{
}
finally
{
}
finally
{
}
CE: finally
without try
```

16.
```
try
{
try
{
}
catch(x e)
{
}
finally
{
}
}
catch(x e)
```

Page 4 of 6



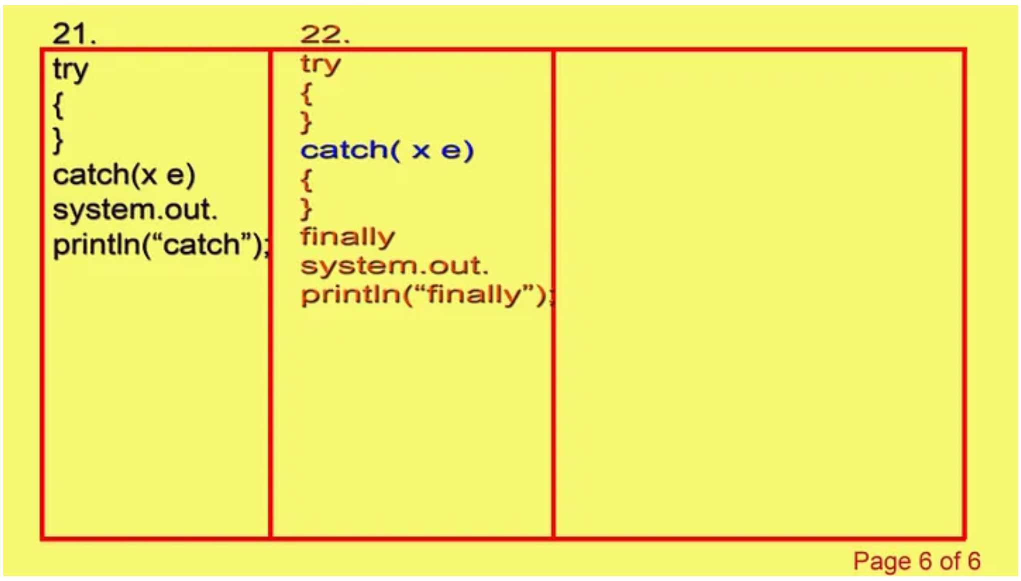
17.
```
try
{
}
catch(x e)
{
try
{
}
finally
{
}
}
```

18.
```
try
{
}
catch (x e)
{
}
finally
{
try
{
}
finally
{
}
}
```

19.
```
try
{
try
{
}
}
catch(x e)
{
}
CE: try without
 catch or finally
```

20.
```
try
system.out.
println("hello");
catch(x e)
{
}
```

Page 5 of 6

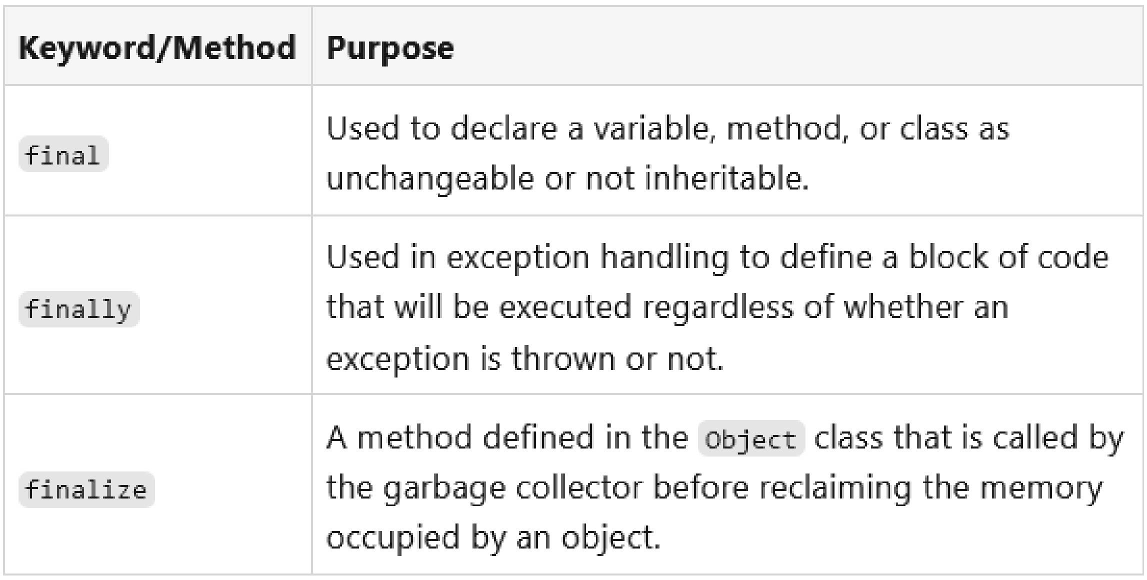


21.
```
try
{
}
catch(x e)
system.out.
println("catch");
```

22.
```
try
{
}
catch( x e)
{
}
finally
system.out.
println("finally");
```

Page 6 of 6

Final vs Finally vs Finalise in java:



Keyword/Method	Purpose
`final`	Used to declare a variable, method, or class as unchangeable or not inheritable.
`finally`	Used in exception handling to define a block of code that will be executed regardless of whether an exception is thrown or not.
`finalize`	A method defined in the `Object` class that is called by the garbage collector before reclaiming the memory occupied by an object.

```java
public class ExceptionHandlingFileNotFound {
   public static void main(String[] args) {
       File file = new File("scores.dat");
       System.out.println(file.exists());
       try {
           Scanner scan = new Scanner(file);//Critical section of code.
       } catch (FileNotFoundException e) {
           System.out.println("File not found!");
       }
   }
}
```

throws keyword:

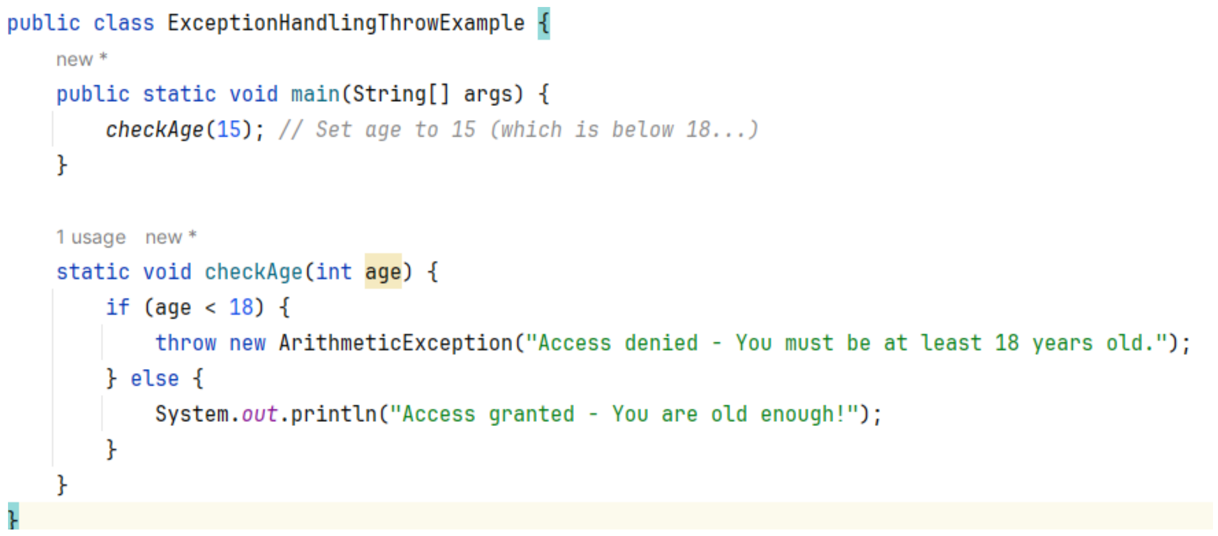
Throws statement: in our program if there is any chance of raising a checked exception, we should handle either by try catch or by throws keyword otherwise the code won't compile.

Hence the main objective of the “**throws**” keyword is to delegate the responsibility of exception handling to the caller method.
“**throws**” keyword required only checked exceptions. Usage of throws for unchecked exceptions is of no use.
“throws” keyword required only to convene the compiler. Usage of throws keyword doesn't prevent abnormal termination of the program.

```java
public class ExceptionHandlingFileNotFoundThrows {
   public static void main(String[] args) throws FileNotFoundException {
       File file = new File("scores.dat");
       System.out.println(file.exists());
       Scanner scan = new Scanner(file);
   }
}
```

Java throw keyword

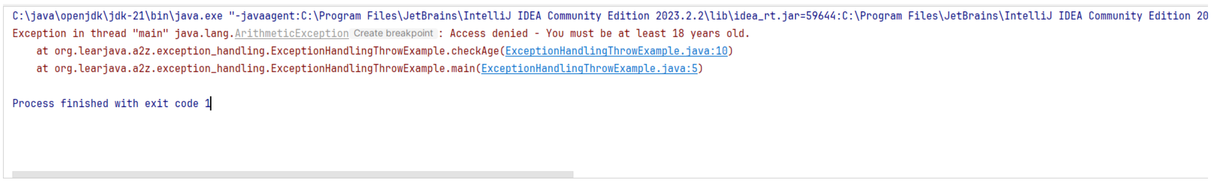
- The Java throw keyword is used to throw an exception explicitly.
- We specify the exception object which is to be thrown. The Exception has some message with it that provides the error description. These exceptions may be related to user inputs, server, etc.
- We can throw either checked or unchecked exceptions in Java by throw keyword. It is mainly used to throw a custom exception.



```java
public class ExceptionHandlingThrowExample {
    new *
    public static void main(String[] args) {
        checkAge(15); // Set age to 15 (which is below 18...)
    }

    1 usage   new *
    static void checkAge(int age) {
        if (age < 18) {
            throw new ArithmeticException("Access denied - You must be at least 18 years old.");
        } else {
            System.out.println("Access granted - You are old enough!");
        }
    }
}
```

Output:



```
C:\java\openjdk\jdk-21\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2023.2.2\lib\idea_rt.jar=59644:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 20
Exception in thread "main" java.lang.ArithmeticException Create breakpoint : Access denied - You must be at least 18 years old.
    at org.learjava.a2z.exception_handling.ExceptionHandlingThrowExample.checkAge(ExceptionHandlingThrowExample.java:10)
    at org.learjava.a2z.exception_handling.ExceptionHandlingThrowExample.main(ExceptionHandlingThrowExample.java:5)

Process finished with exit code 1
```

.User defined Exception:

```java
class InvalidAgeException extends Exception {
   public InvalidAgeException(String str) {
       // calling the constructor of parent Exception
       super(str);
    }
}
```

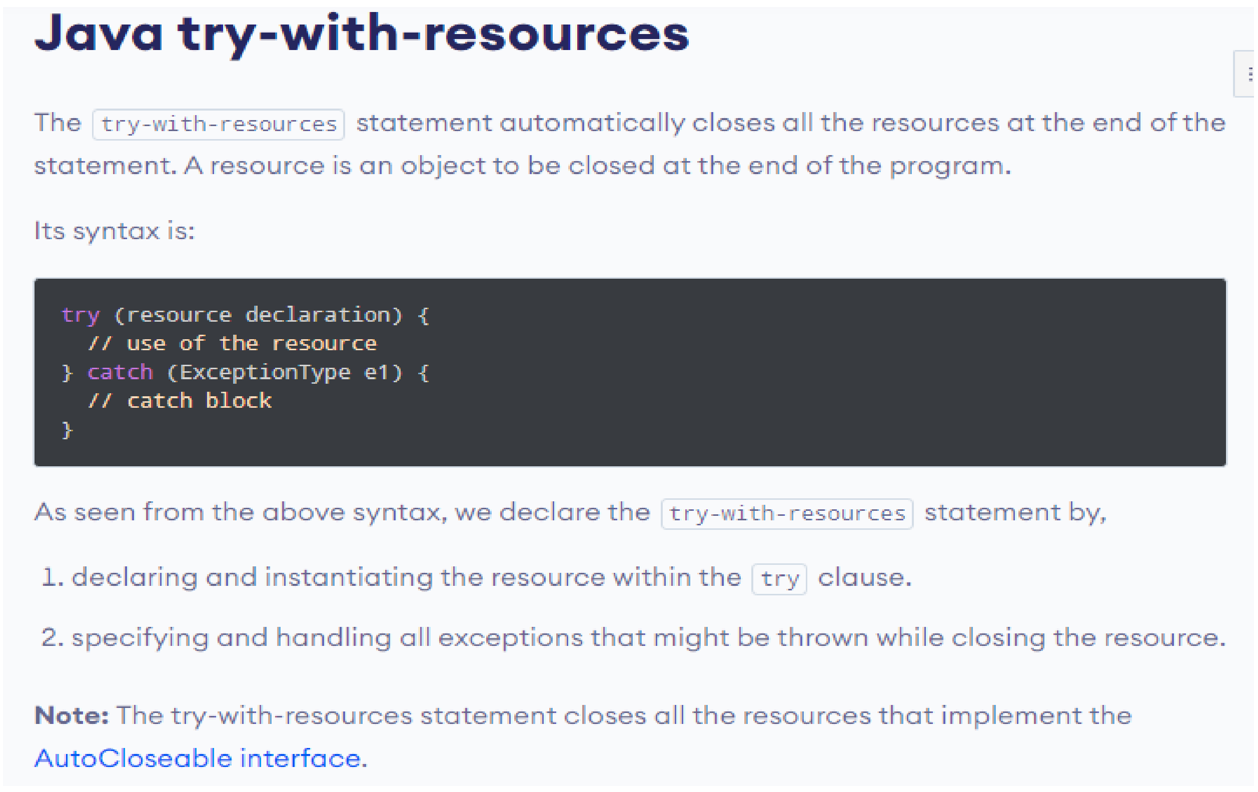
Test Class:

```
public class TestCustomException {
   static void validate(int age) throws InvalidAgeException {
       if (age < 18) {
           throw new InvalidAgeException("age is not valid to vote");
       } else {
           System.out.println("welcome to vote");
       }
   }

   public static void main(String args[]) {
       try {
           validate(13);
       } catch (InvalidAgeException ex) {
           System.out.println("Caught the exception");
           System.out.println("Exception occured: " + ex);
       }
       System.out.println("rest of the code...");
   }
}
```

Output:

```
Caught the exception
Exception occured: org.learjava.a2z.exception_handling.InvalidAgeException: age is not
valid to vote
rest of the code…
```



Java try-with-resources

The `try-with-resources` statement automatically closes all the resources at the end of the statement. A resource is an object to be closed at the end of the program.

Its syntax is:

```
try (resource declaration) {
  // use of the resource
} catch (ExceptionType e1) {
  // catch block
}
```

As seen from the above syntax, we declare the `try-with-resources` statement by,

1. declaring and instantiating the resource within the `try` clause.
2. specifying and handling all exceptions that might be thrown while closing the resource.

Note: The try-with-resources statement closes all the resources that implement the AutoCloseable interface.

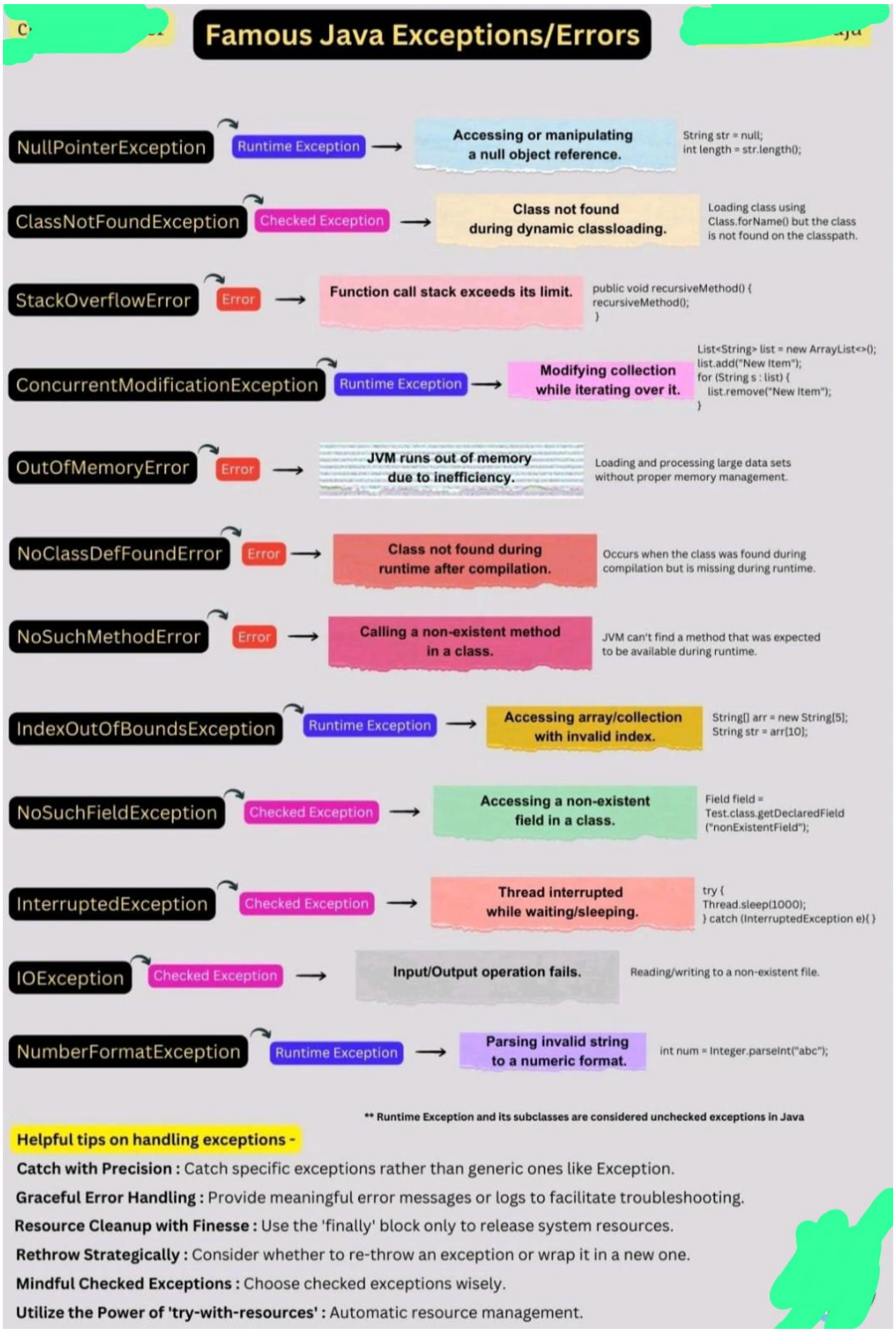
```
public class TryWithResources {
   public static void main(String[] args) {
       String line;
       try (BufferedReader br = new BufferedReader(new FileReader("test.txt"))) {
           while ((line = br.readLine()) != null) {
               System.out.println("Line =>" + line);
           }
       } catch (IOException e) {
           System.out.println("IOException in try block =>" + e.getMessage());
       }
   }
}
```

Output:

```
IOException in try block =>test.txt (The system cannot find the file specified)
```

Exception Handling keyword summary:

Keyword	Description
try	The "try" keyword is used to specify a block where we should place an exception code. It means we can't use try block alone. The try block must be followed by either catch or finally.
catch	The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later.
finally	The "finally" block is used to execute the necessary code of the program. It is executed whether an exception is handled or not.
throw	The "throw" keyword is used to throw an exception.
throws	The "throws" keyword is used to declare exceptions. It specifies that there may occur an exception in the method. It doesn't throw an exception. It is always used with method signature.



[redacted]

Famous Java Exceptions/Errors

[redacted]

NullPointerException	Runtime Exception	Accessing or manipulating a null object reference.	`String str = null;` `int length = str.length();`
ClassNotFoundException	Checked Exception	Class not found during dynamic classloading.	Loading class using Class.forName() but the class is not found on the classpath.
StackOverflowError	Error	Function call stack exceeds its limit.	`public void recursiveMethod() {` `recursiveMethod();` `}`
ConcurrentModificationException	Runtime Exception	Modifying collection while iterating over it.	`List<String> list = new ArrayList<>();` `list.add("New Item");` `for (String s : list) {` `list.remove("New Item");` `}`
OutOfMemoryError	Error	JVM runs out of memory due to inefficiency.	Loading and processing large data sets without proper memory management.
NoClassDefFoundError	Error	Class not found during runtime after compilation.	Occurs when the class was found during compilation but is missing during runtime.
NoSuchMethodError	Error	Calling a non-existent method in a class.	JVM can't find a method that was expected to be available during runtime.
IndexOutOfBoundsException	Runtime Exception	Accessing array/collection with invalid index.	`String[] arr = new String[5];` `String str = arr[10];`
NoSuchFieldException	Checked Exception	Accessing a non-existent field in a class.	`Field field =` `Test.class.getDeclaredField` `("nonExistentField");`
InterruptedException	Checked Exception	Thread interrupted while waiting/sleeping.	`try {` `Thread.sleep(1000);` `} catch (InterruptedException e){ }`
IOException	Checked Exception	Input/Output operation fails.	Reading/writing to a non-existent file.
NumberFormatException	Runtime Exception	Parsing invalid string to a numeric format.	`int num = Integer.parseInt("abc");`

** Runtime Exception and its subclasses are considered unchecked exceptions in Java

Helpful tips on handling exceptions -

Catch with Precision : Catch specific exceptions rather than generic ones like Exception.

Graceful Error Handling : Provide meaningful error messages or logs to facilitate troubleshooting.

Resource Cleanup with Finesse : Use the 'finally' block only to release system resources.

Rethrow Strategically : Consider whether to re-throw an exception or wrap it in a new one.

Mindful Checked Exceptions : Choose checked exceptions wisely.

Utilize the Power of 'try-with-resources' : Automatic resource management.

[redacted]