25th Jan 2024

EXCEPTION HANDLING BEST PRACTICES

Expert Connect Session

Speaker

Kusum Chaurasia

Technical Architect (CCA-Europe)
Certified L1 Architect
Mercedes Benz EPDM

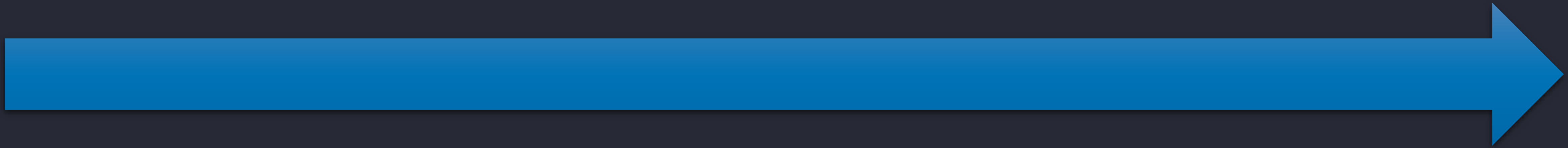
Exp: 11.8 years
Key SkillSets:

- Java/J2EE
- Springboot
- Microservices
- Azure Cloud

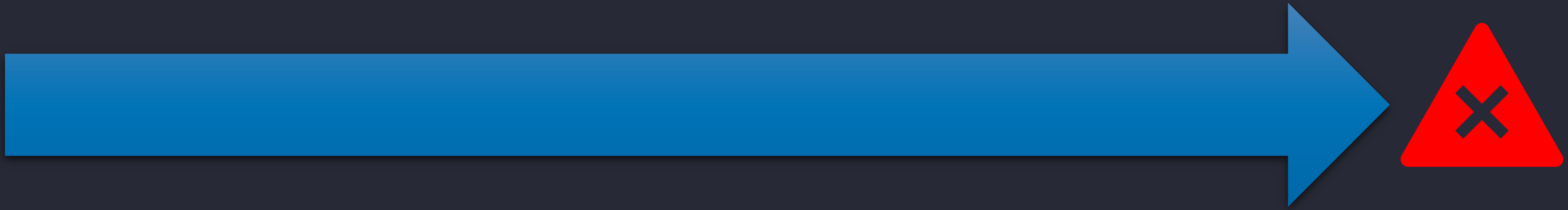
AGENDA

- Overview of Exception
- Exception hierarchy
- Exception Handling
  - How it works?
  - Operations
  - Finally, rethrowing and chained exception
  - User-Defined/Custom Exception
- Best Practices/Usual Issues
- Static Code Analysis tool
- Key Takeaways

Exception ??



Exception ??

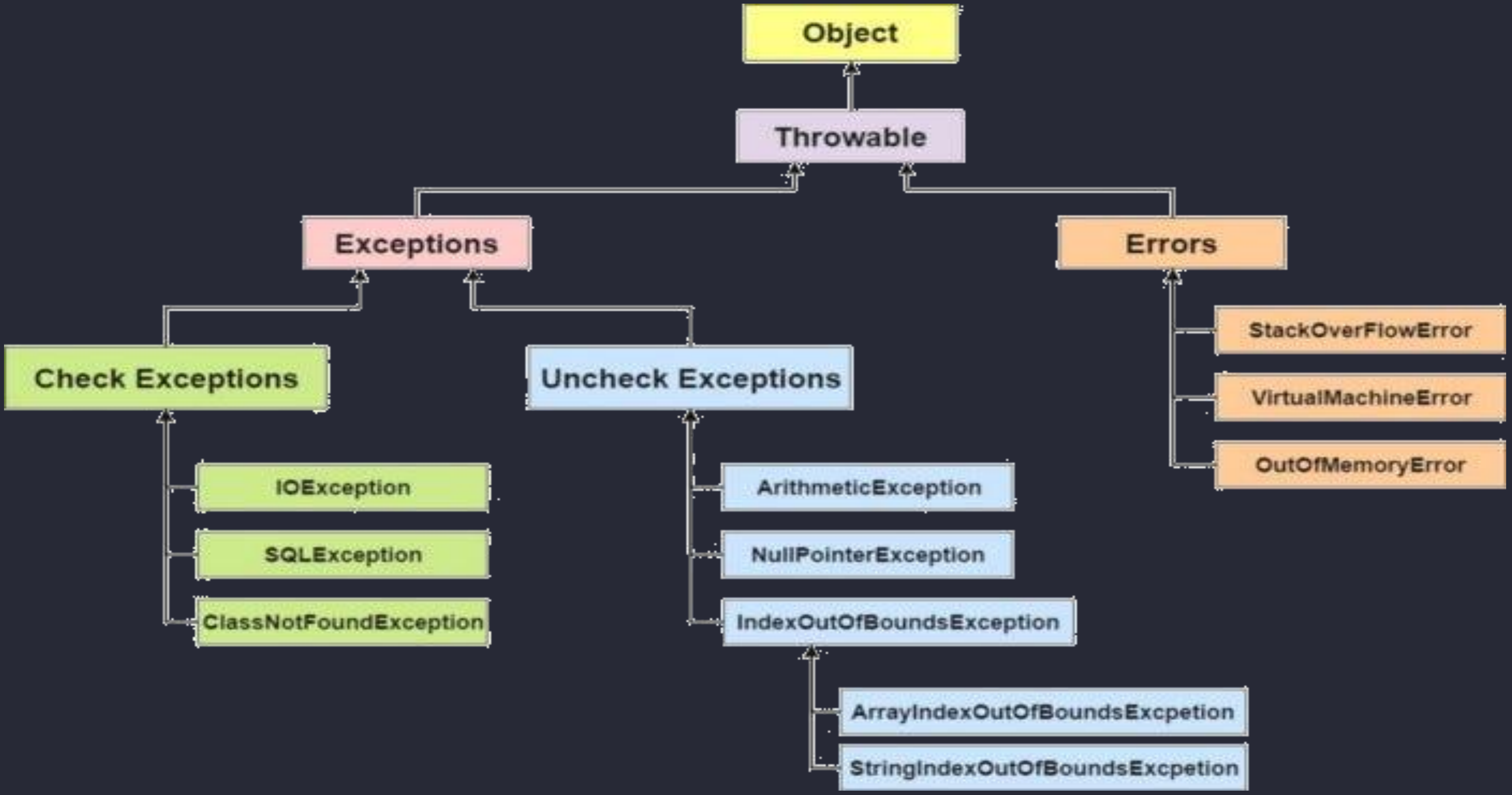


OVERVIEW OF EXCEPTION

- What is an Exception?
  - Exception represent **abnormal conditions** or errors that occur during the execution of a Java program.
  - Event that **disrupts** the normal flow of the program
  - It is an object which is thrown at runtime.

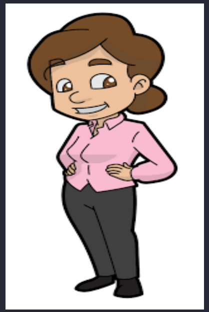
- Why we need to care about Exceptions?
  - Enables a program to **deal with exceptional situations** and continue its normal execution.

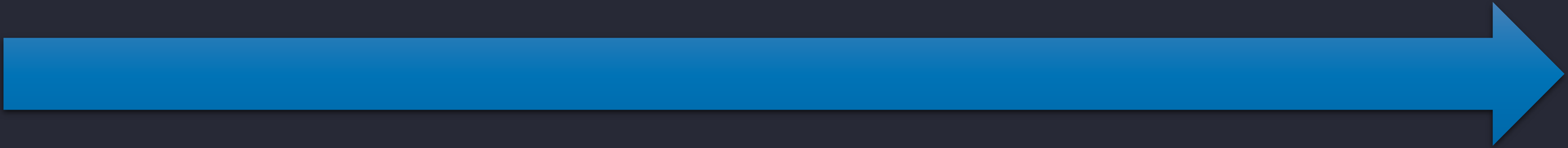
EXCEPTION HIERARCHY



CHECKED & UNCHECKED EXCEPTION??

Wife/Mother





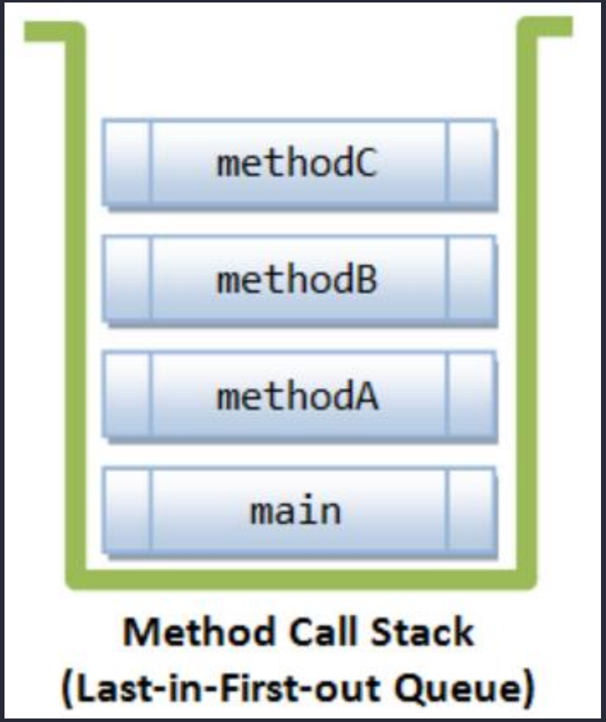
CHECKED & UNCHECKED EXCEPTION??

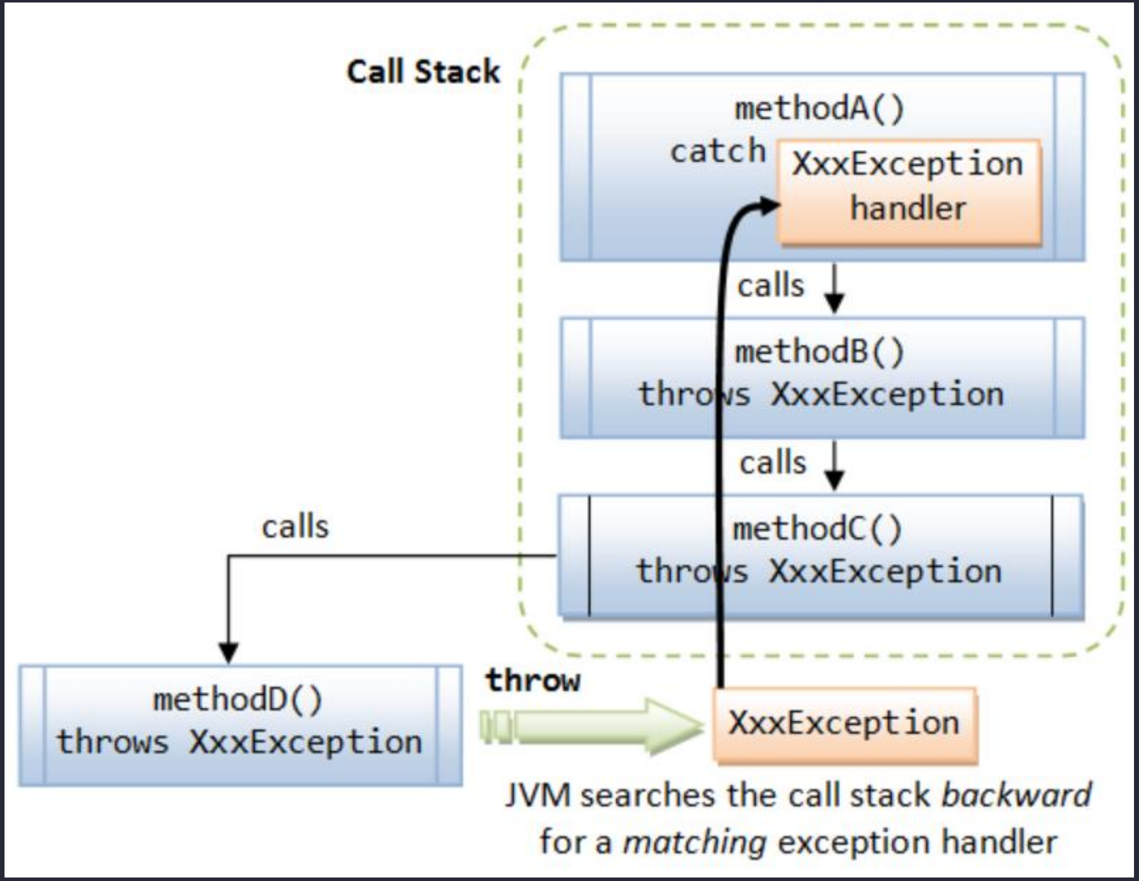
Compiler

WalletNotFoundExcetion
IDNotFoundException
LunchNotFoundException
….

TirePunctureException
AccidentException
…

EXCEPTION HANDLING (HOW IT WORKS ?)





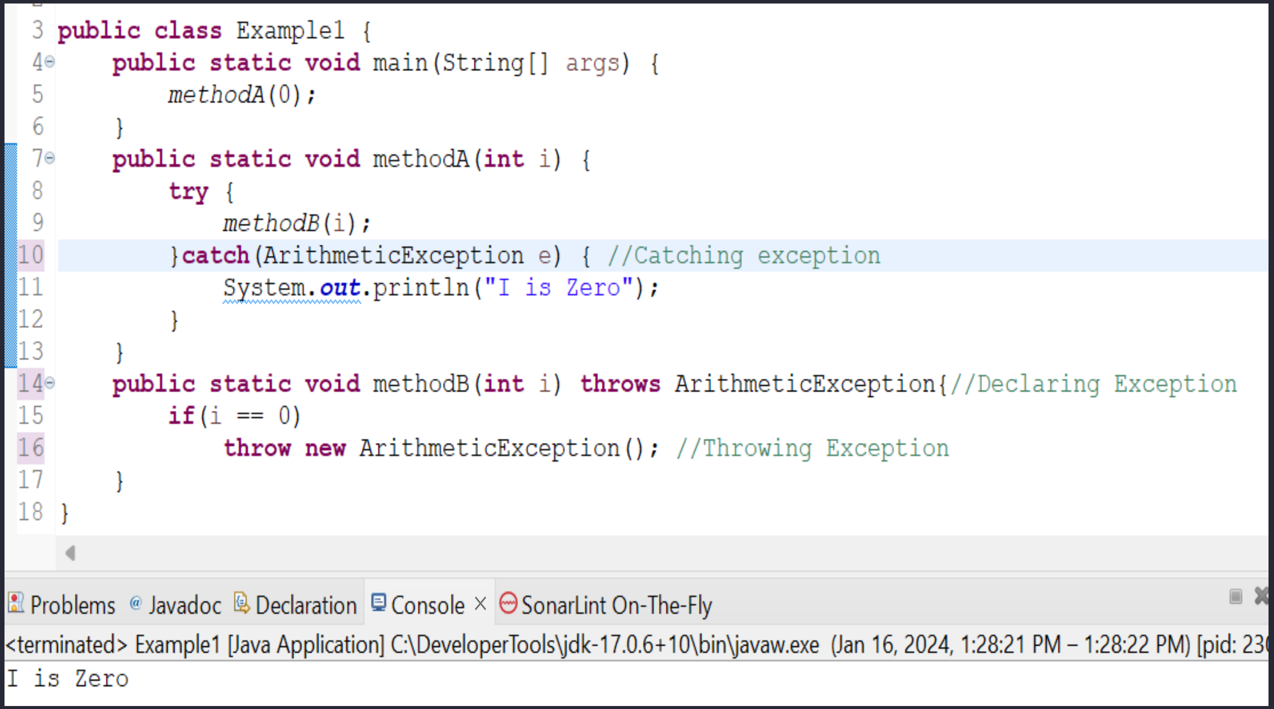
EXCEPTION HANDLING OPERATIONS

- Five keywords are used in exception handling: **try**, **catch**, **finally**, **throws** and **throw**.
- Java's exception handling consists of three operations:

Declare

Throw

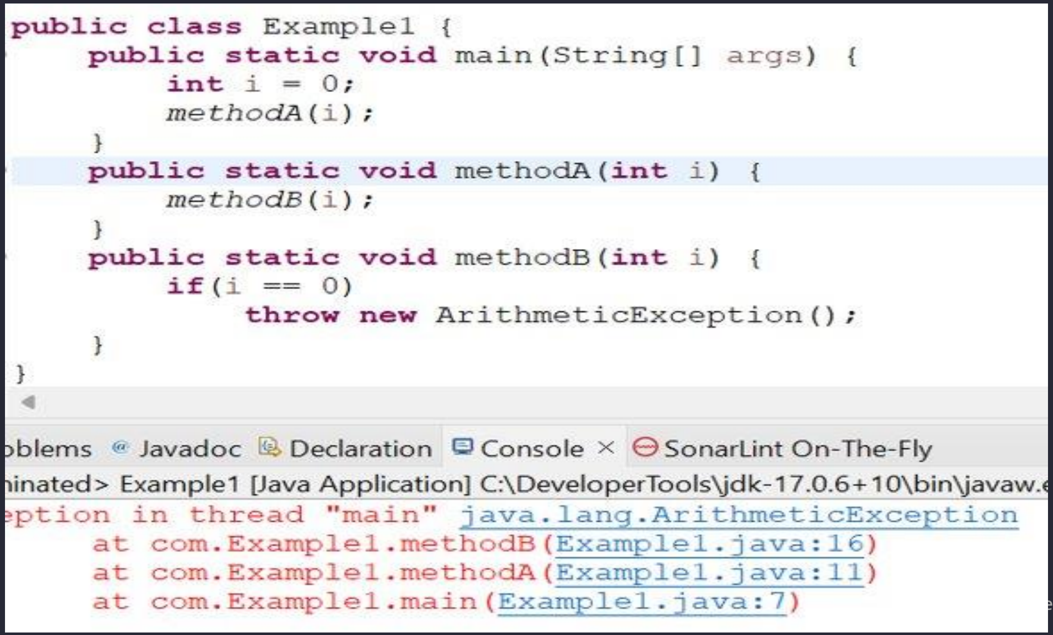
Catch



```
public class Example1 {
    public static void main(String[] args) {
        methodA(0);
    }
    public static void methodA(int i) {
        try {
            methodB(i);
        }catch(ArithmeticException e) { //Catching exception
            System.out.println("I is Zero");
        }
    }
    public static void methodB(int i) throws ArithmeticException{//Declaring Exception
        if(i == 0)
            throw new ArithmeticException(); //Throwing Exception
    }
}
```

```
<terminated> Example1 [Java Application] C:\DeveloperTools\jdk-17.0.6+10\bin\javaw.exe  (Jan 16, 2024, 1:28:21 PM – 1:28:22 PM) [pid: 23
I is Zero
```

If no exception handler is found in the call stack, the program terminates.



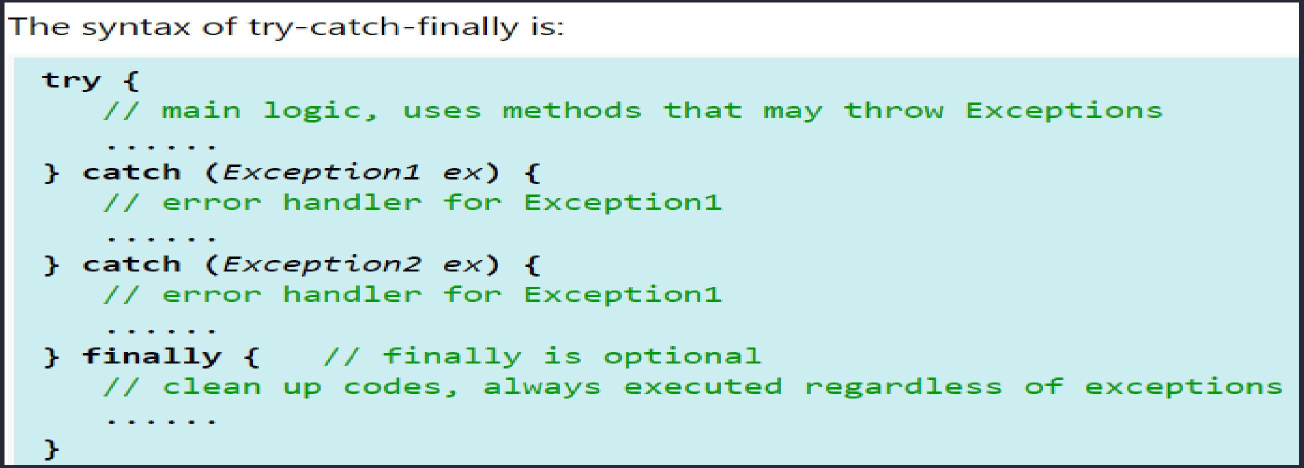
```
public class Example1 {
    public static void main(String[] args) {
        int i = 0;
        methodA(i);
    }
    public static void methodA(int i) {
        methodB(i);
    }
    public static void methodB(int i) {
        if(i == 0)
            throw new ArithmeticException();
    }
}
```

```
inated> Example1 [Java Application] C:\DeveloperTools\jdk-17.0.6+10\bin\javaw.e
eption in thread "main" java.lang.ArithmeticException
	at com.Example1.methodB(Example1.java:16)
	at com.Example1.methodA(Example1.java:11)
	at com.Example1.main(Example1.java:7)
```

FINALLY, RETHROWING AND CHAINED EXCEPTION

- **finally Clause**

  The finally block is almost certain to be executed, regardless of whether or not exception occurs (unless JVM encountered a severe error or a System.exit() is called in the catch block).

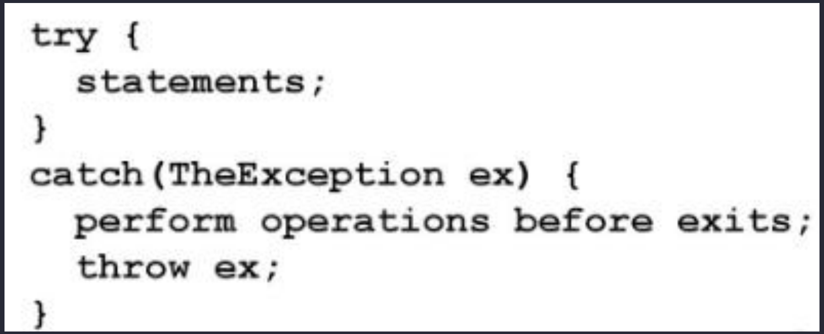


The syntax of try-catch-finally is:

```
try {
   // main logic, uses methods that may throw Exceptions
   ......
} catch (Exception1 ex) {
   // error handler for Exception1
   ......
} catch (Exception2 ex) {
   // error handler for Exception1
   ......
} finally {   // finally is optional
   // clean up codes, always executed regardless of exceptions
   ......
}
```

- **Rethrowing Exception**

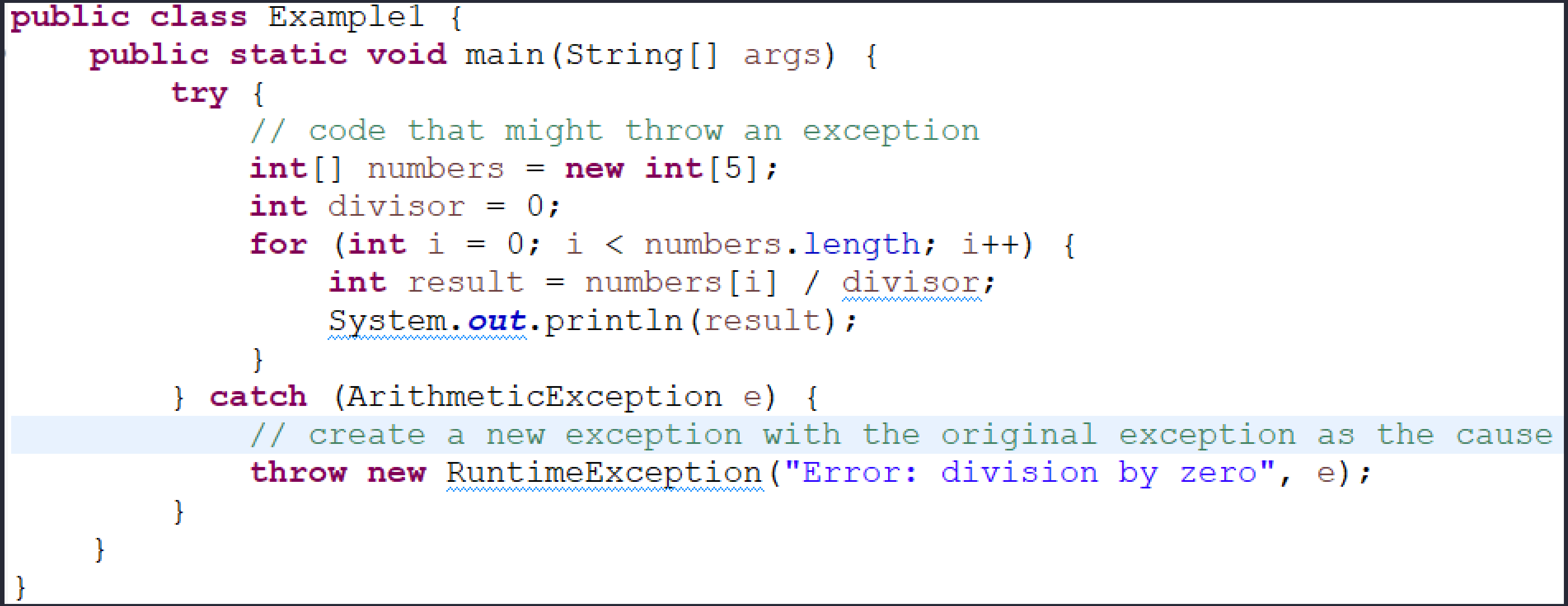
  Java allows an exception handler to rethrow the exception if the handler cannot process the execution or simply wants to let its caller to be notified of the exception. **Used in catch block.**



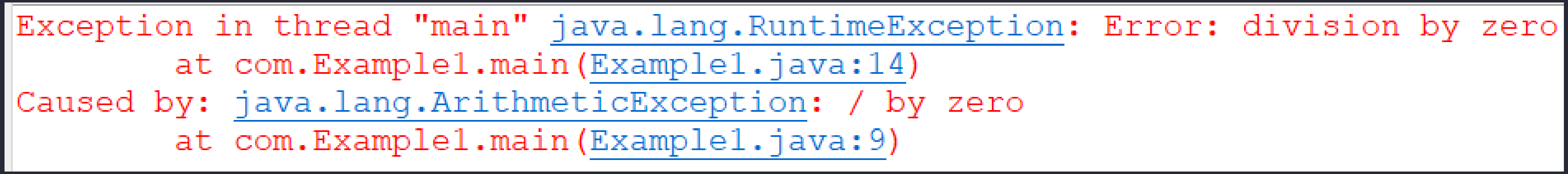
```
try {
  statements;
}
catch(TheException ex) {
  perform operations before exits;
  throw ex;
}
```

➢ Chained Exception

Throwing an exception along with another exception forms a chained exception.



```java
public class Example1 {
    public static void main(String[] args) {
        try {
            // code that might throw an exception
            int[] numbers = new int[5];
            int divisor = 0;
            for (int i = 0; i < numbers.length; i++) {
                int result = numbers[i] / divisor;
                System.out.println(result);
            }
        } catch (ArithmeticException e) {
            // create a new exception with the original exception as the cause
            throw new RuntimeException("Error: division by zero", e);
        }
    }
}
```



```
Exception in thread "main" java.lang.RuntimeException: Error: division by zero
	at com.Example1.main(Example1.java:14)
Caused by: java.lang.ArithmeticException: / by zero
	at com.Example1.main(Example1.java:9)
```

Caution When Creating Exception

Exception handling usually requires more time and resources because it requires instantiating a new exception object, rolling back the call stack and propagating the errors to the calling methods.

When to Throw Exception

- If you want the **exception to be processed by its caller,** you should create an exception object and throw it.
- If you can handle the **exception in the method where it occurs**, there is no need to throw it (simply use try-catch block).

When to use Exception

- When should you use the try-catch block in the code? you should use it **to deal with unexpected error conditions.** Do not use it to deal with simple, expected situations.

```
try {
    System.out.println(str.toString());
}catch(NullPointerException ex){
    System.out.println("str is null");
}
```

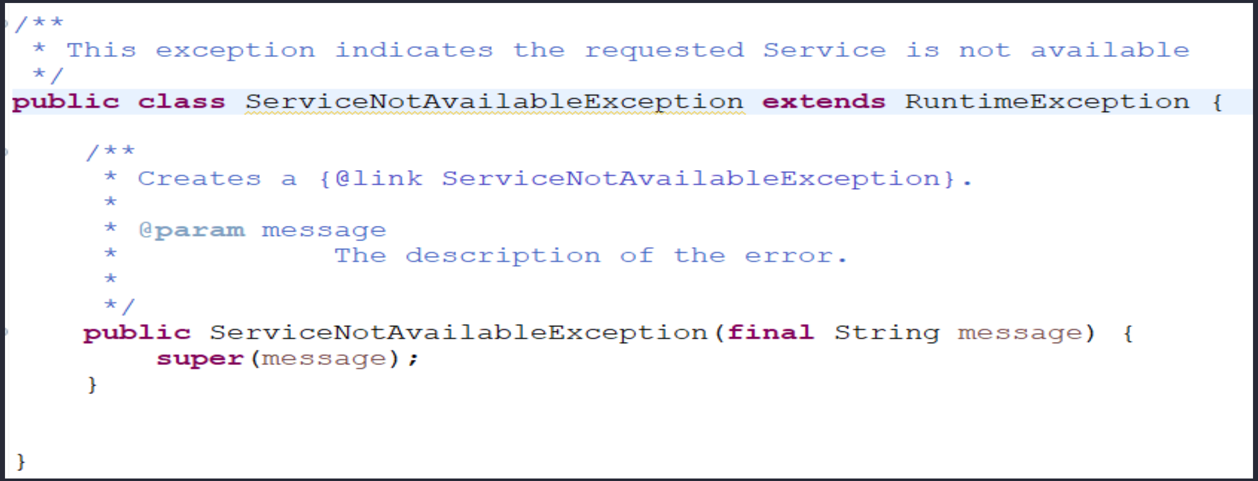
Better to replace with

```
if(str != null) {
    System.out.println(str.toString());
}
```

USER DEFINED/CUSTOM EXCEPTION

- Create your own `Exception` classes by extending from the class `Exception` or one of its subclasses.

- WalletNotFoundException
- LunchNotFoundExceptio
- IDNotFoundException
- ……
- TirePunctureException
- AccidentException

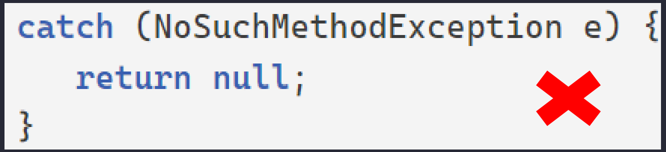


```
/**
 * This exception indicates the requested Service is not available
 */
public class ServiceNotAvailableException extends RuntimeException {

    /**
     * Creates a {@link ServiceNotAvailableException}.
     *
     * @param message
     *            The description of the error.
     *
     */
    public ServiceNotAvailableException(final String message) {
        super(message);
    }


}
```

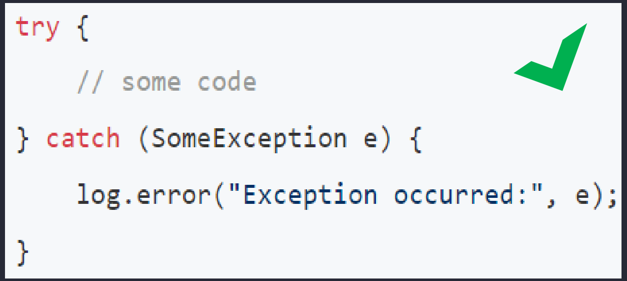
BEST PRACTICES

1. Never swallow the exception in the *catch* block

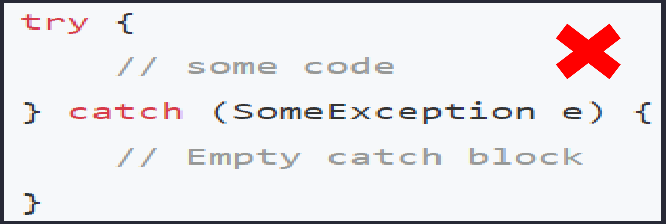
Never leave an empty catch block or returns "null" instead of handling or re-throwing the exception, it totally swallows the exception, losing the original cause of the error forever. And when you don't know the reason for failure, how would you prevent it in the future? Never do this !!.



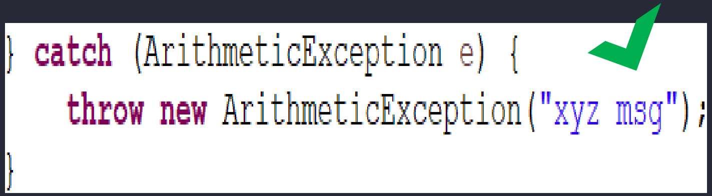
```java
catch (NoSuchMethodException e) {
   return null;
}
```



```java
try {
    // some code
} catch (SomeException e) {
    log.error("Exception occurred:", e);
}
```



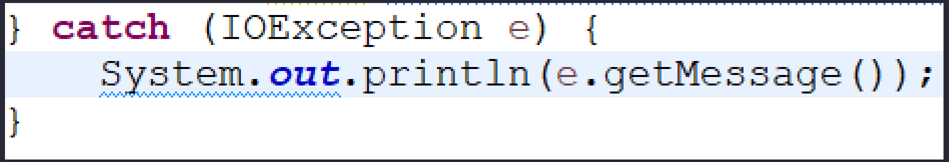
```java
try {
    // some code
} catch (SomeException e) {
    // Empty catch block
}
```

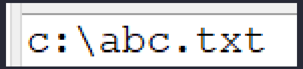


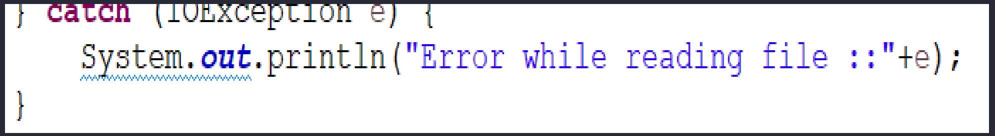
```java
} catch (ArithmeticException e) {
    throw new ArithmeticException("xyz msg");
}
```

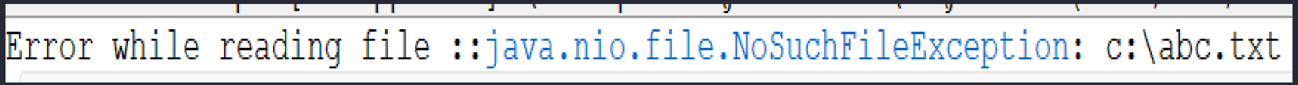
2. Use meaningful exception messages

- When you catch an exception, always use meaningful and informative messages. This helps to identify the root cause of the exception quickly and take appropriate actions.


```
} catch (IOException e) {
    System.out.println(e.getMessage());
}
```

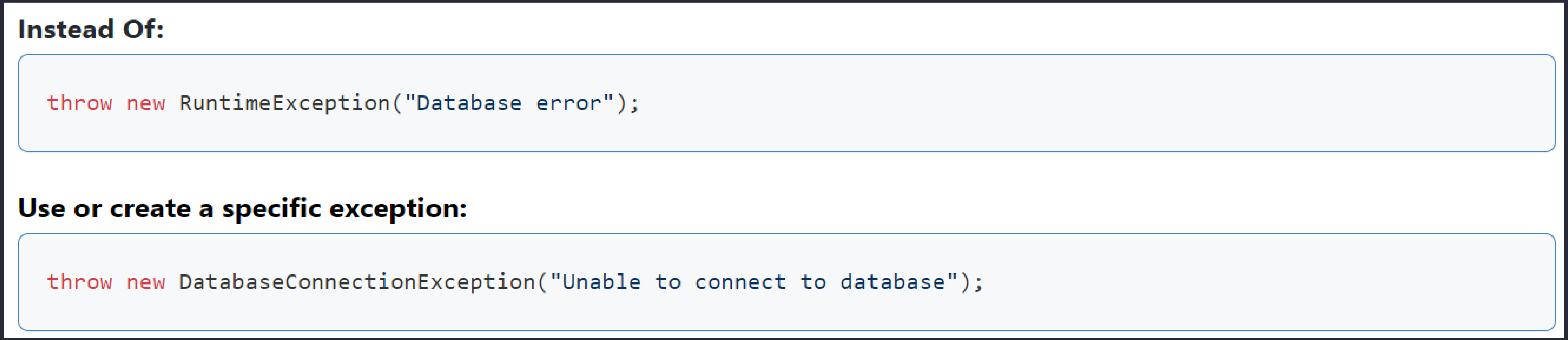

```
c:\abc.txt
```


```
} catch (IOException e) {
    System.out.println("Error while reading file ::"+e);
}
```


```
Error while reading file ::java.nio.file.NoSuchFileException: c:\abc.txt
```

3. Avoid Throwing Raw Exception Types

Avoid throwing raw exception types like RuntimeException, Exception, or Throwable. It’s always best to throw specific exception types, whether they're from the Java Standard Library or custom ones you've defined.



Instead Of:

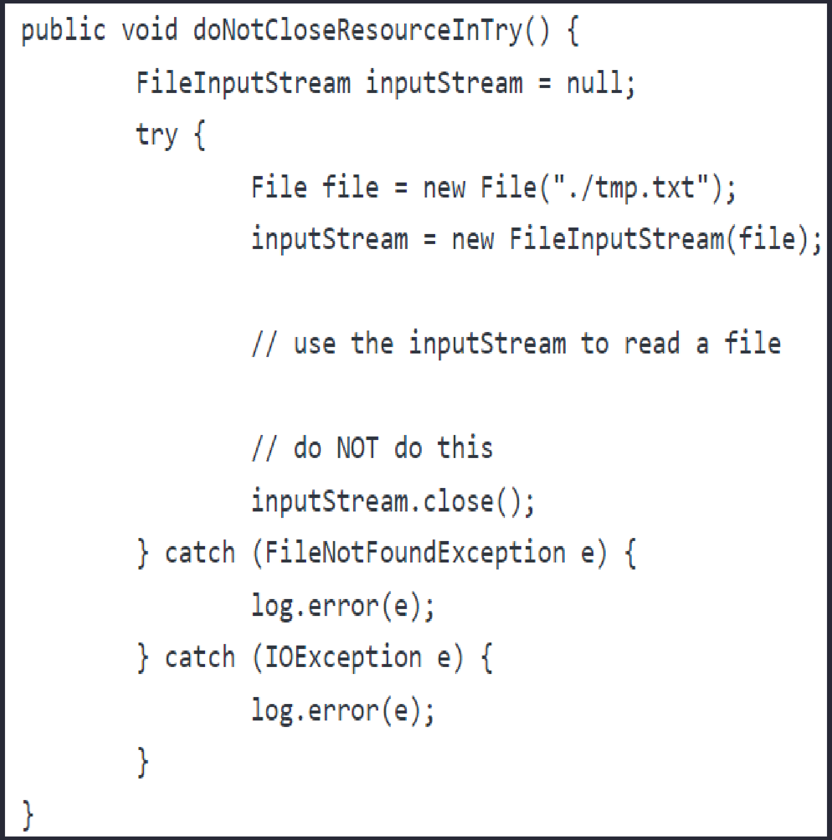
```
throw new RuntimeException("Database error");
```

Use or create a specific exception:

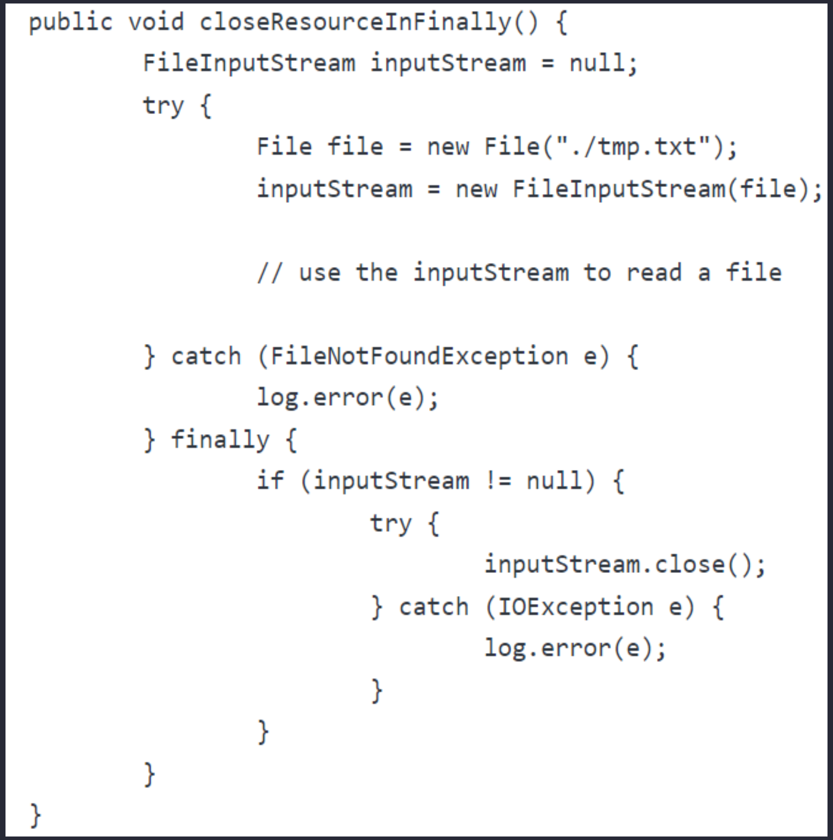
```
throw new DatabaseConnectionException("Unable to connect to database");
```

4. Clean up Resources in a Finally Block or Use a Try-With-Resource Statement

Ensure that resources (like streams or database connections) are closed in the finally block or utilize the try-with-resources statement for auto-closable resources.

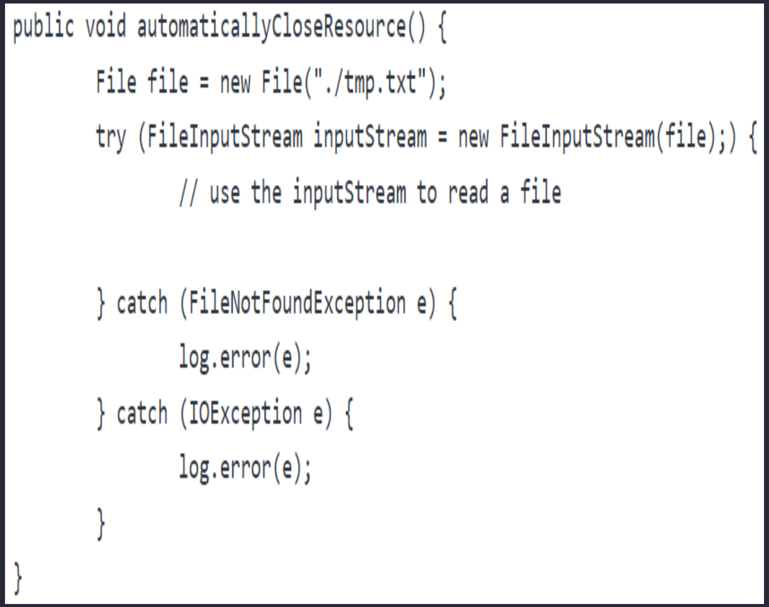


```java
public void doNotCloseResourceInTry() {
	FileInputStream inputStream = null;
	try {
		File file = new File("./tmp.txt");
		inputStream = new FileInputStream(file);

		// use the inputStream to read a file

		// do NOT do this
		inputStream.close();
	} catch (FileNotFoundException e) {
		log.error(e);
	} catch (IOException e) {
		log.error(e);
	}
}
```



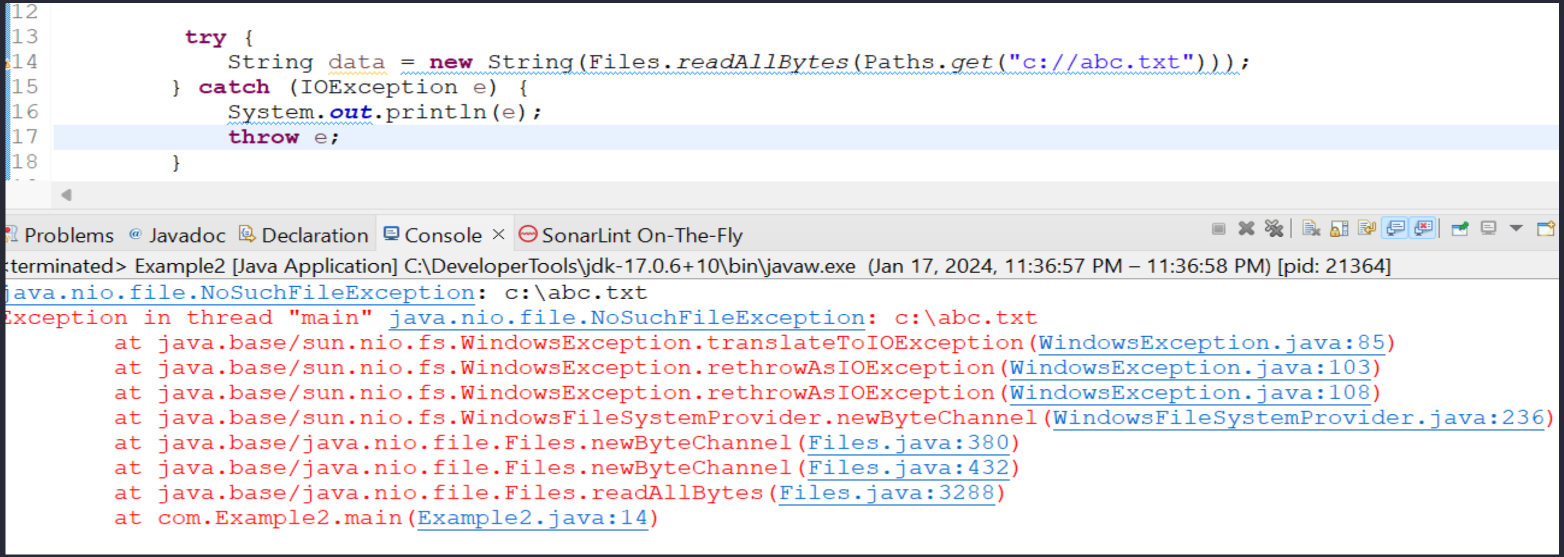
```java
public void closeResourceInFinally() {
	FileInputStream inputStream = null;
	try {
		File file = new File("./tmp.txt");
		inputStream = new FileInputStream(file);

		// use the inputStream to read a file

	} catch (FileNotFoundException e) {
		log.error(e);
	} finally {
		if (inputStream != null) {
			try {
				inputStream.close();
			} catch (IOException e) {
				log.error(e);
			}
		}
	}
}
```

Try with Resource statement



```java
public void automaticallyCloseResource() {
	File file = new File("./tmp.txt");
	try (FileInputStream inputStream = new FileInputStream(file);) {
		// use the inputStream to read a file

	} catch (FileNotFoundException e) {
		log.error(e);
	} catch (IOException e) {
		log.error(e);
	}
}
```

5. Log the exception or throw it but never do both



```
        try {
            String data = new String(Files.readAllBytes(Paths.get("c://abc.txt")));
        } catch (IOException e) {
            System.out.println(e);
            throw e;
        }
```

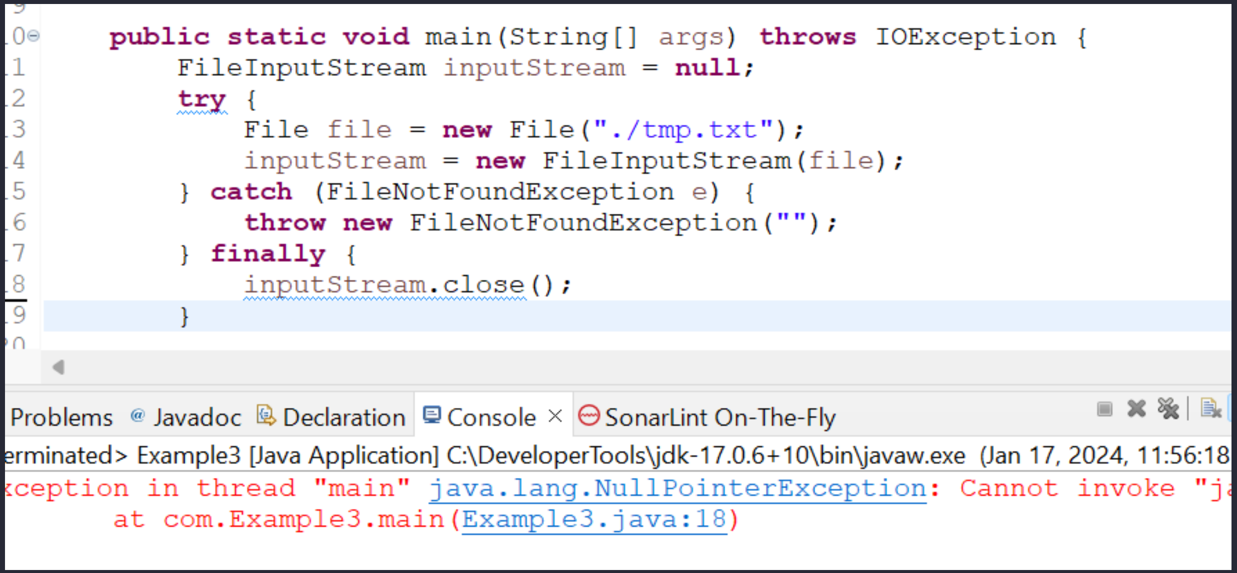
Problems Javadoc Declaration Console × SonarLint On-The-Fly

```
terminated> Example2 [Java Application] C:\DeveloperTools\jdk-17.0.6+10\bin\javaw.exe  (Jan 17, 2024, 11:36:57 PM – 11:36:58 PM) [pid: 21364]
java.nio.file.NoSuchFileException: c:\abc.txt
Exception in thread "main" java.nio.file.NoSuchFileException: c:\abc.txt
	at java.base/sun.nio.fs.WindowsException.translateToIOException(WindowsException.java:85)
	at java.base/sun.nio.fs.WindowsException.rethrowAsIOException(WindowsException.java:103)
	at java.base/sun.nio.fs.WindowsException.rethrowAsIOException(WindowsException.java:108)
	at java.base/sun.nio.fs.WindowsFileSystemProvider.newByteChannel(WindowsFileSystemProvider.java:236)
	at java.base/java.nio.file.Files.newByteChannel(Files.java:380)
	at java.base/java.nio.file.Files.newByteChannel(Files.java:432)
	at java.base/java.nio.file.Files.readAllBytes(Files.java:3288)
	at com.Example2.main(Example2.java:14)
```

As it is shown in the above code that throwing and logging can result in the multiple log messages in the log files. The **single issue in the code can create worst circumstances** for the developers who are trying to go through various logs.

6. Never throw an exception from finally block

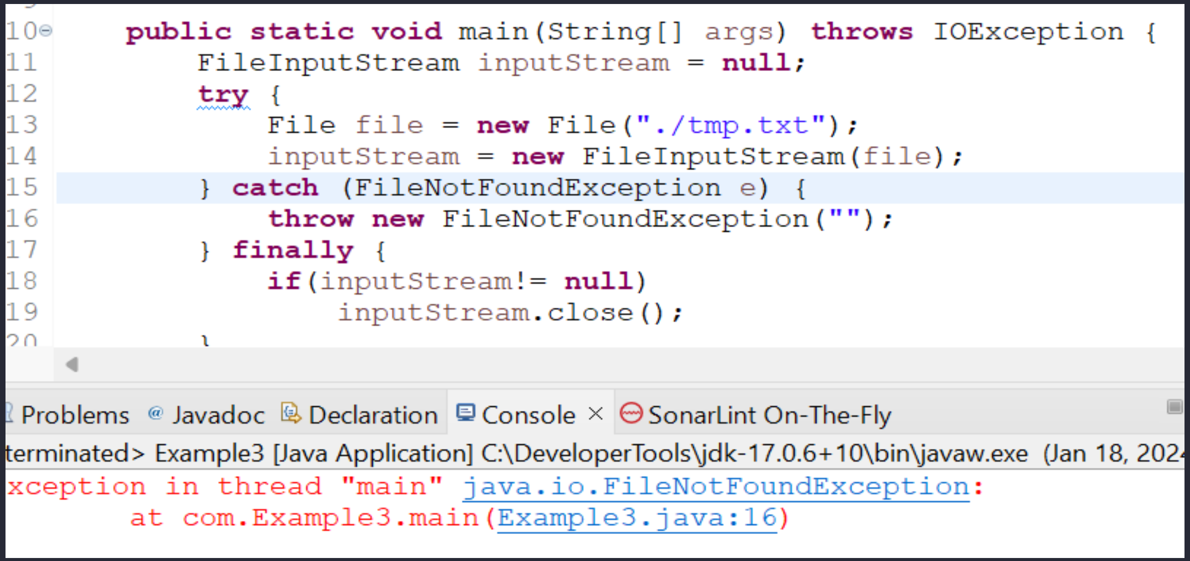
Other exception will grow out of the method and the original first exception **(right reason) will be lost forever**.



```
    public static void main(String[] args) throws IOException {
        FileInputStream inputStream = null;
        try {
            File file = new File("./tmp.txt");
            inputStream = new FileInputStream(file);
        } catch (FileNotFoundException e) {
            throw new FileNotFoundException("");
        } finally {
            inputStream.close();
        }
```

Problems Javadoc Declaration Console × SonarLint On-The-Fly

```
erminated> Example3 [Java Application] C:\DeveloperTools\jdk-17.0.6+10\bin\javaw.exe (Jan 17, 2024, 11:56:18
xception in thread "main" java.lang.NullPointerException: Cannot invoke "ja
        at com.Example3.main(Example3.java:18)
```



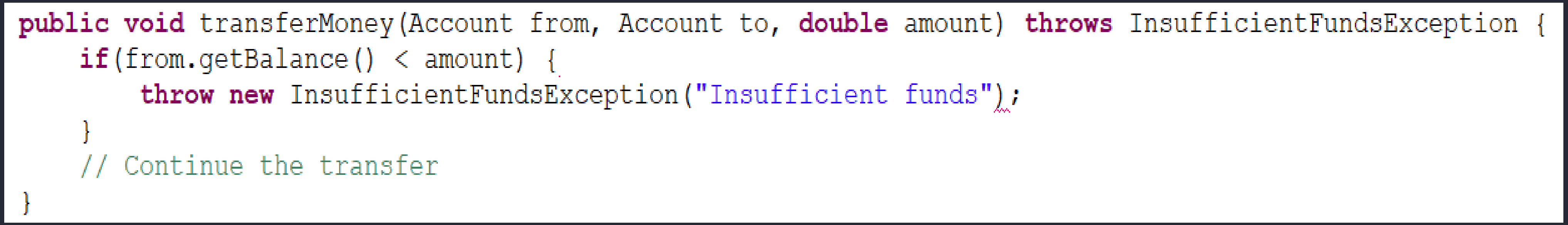
```
    public static void main(String[] args) throws IOException {
        FileInputStream inputStream = null;
        try {
            File file = new File("./tmp.txt");
            inputStream = new FileInputStream(file);
        } catch (FileNotFoundException e) {
            throw new FileNotFoundException("");
        } finally {
            if(inputStream!= null)
                inputStream.close();
```

Problems Javadoc Declaration Console × SonarLint On-The-Fly

```
terminated> Example3 [Java Application] C:\DeveloperTools\jdk-17.0.6+10\bin\javaw.exe (Jan 18, 2024
xception in thread "main" java.io.FileNotFoundException:
        at com.Example3.main(Example3.java:16)
```

7. Use Checked Exceptions for Recoverable Errors

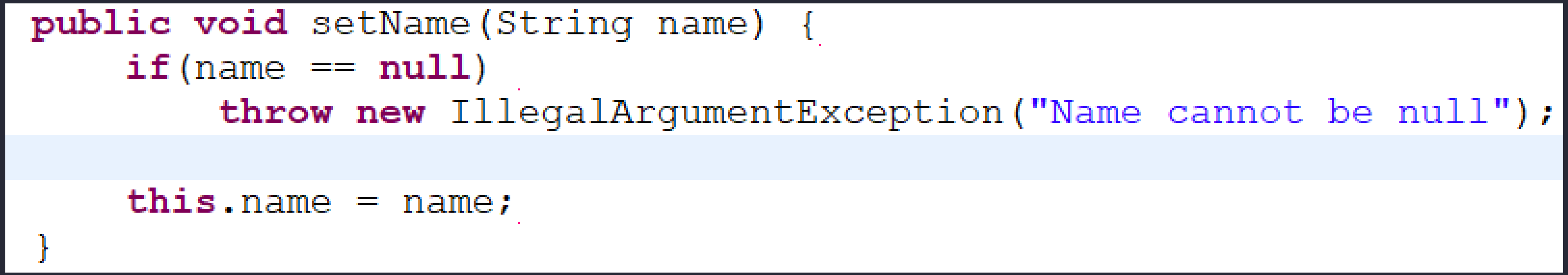
Checked exceptions (those that extend Exception but not RuntimeException) should be used for conditions from which the caller can reasonably be **expected to recover**.



```java
public void transferMoney(Account from, Account to, double amount) throws InsufficientFundsException {
    if(from.getBalance() < amount) {
        throw new InsufficientFundsException("Insufficient funds");
    }
    // Continue the transfer
}
```

8. Use Runtime Exceptions for Programming Errors

Unchecked exceptions (those that extend RuntimeException) **should indicate programming errors**.



```
public void setName(String name) {
    if(name == null)
        throw new IllegalArgumentException("Name cannot be null");

    this.name = name;
}
```

STATIC CODE ANALYSIS TOOLS

sonarqube

Self-managed

Self-managed static analysis tool for continuous codebase inspection

sonarlint

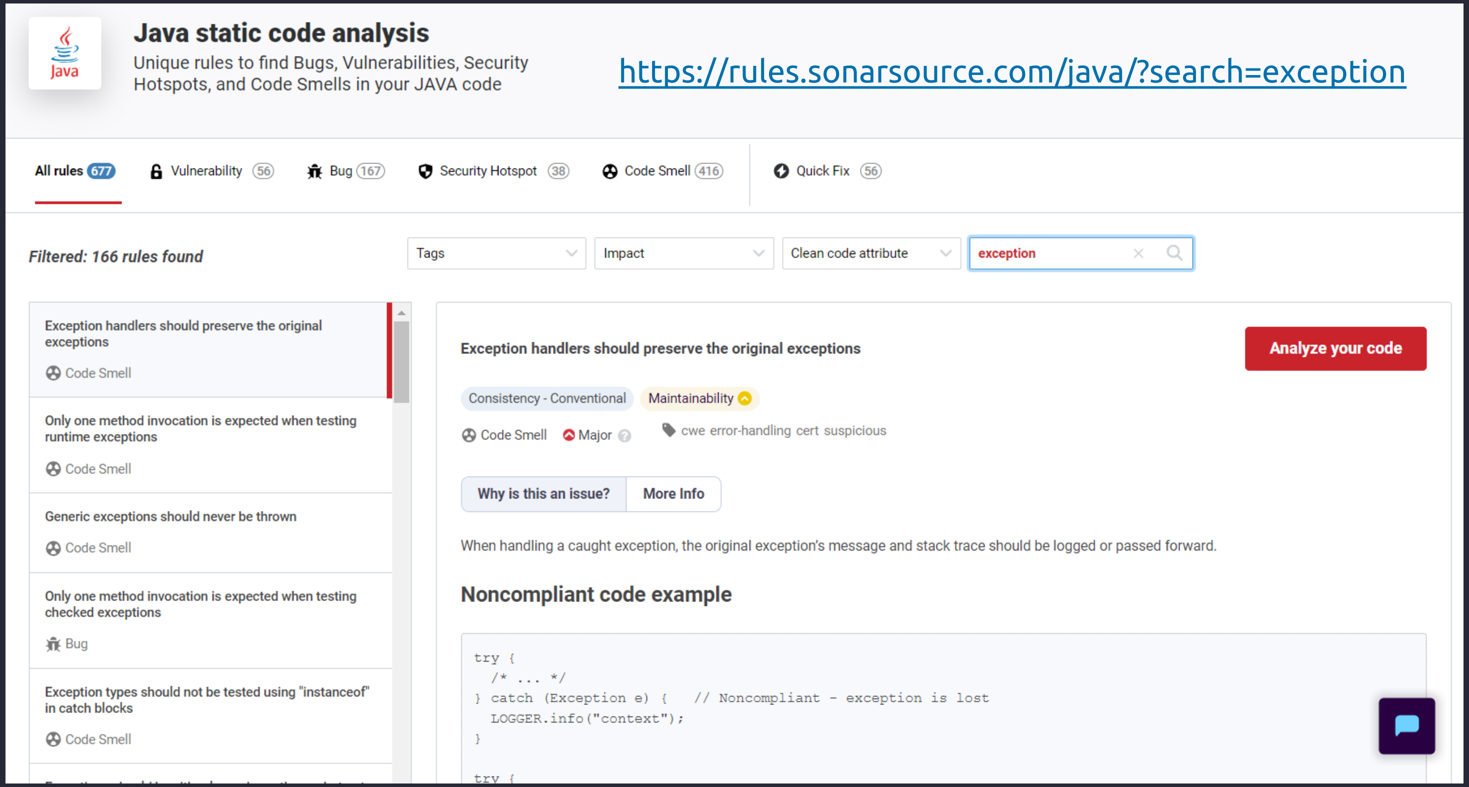
In your IDE

Free IDE extension that provides on-the-fly analysis and coding guidance

FindBugs

because it's easy

{PMD}



Java static code analysis

Unique rules to find Bugs, Vulnerabilities, Security Hotspots, and Code Smells in your JAVA code

https://rules.sonarsource.com/java/?search=exception

All rules 677 | Vulnerability 56 | Bug 167 | Security Hotspot 38 | Code Smell 416 | Quick Fix 56

Filtered: 166 rules found

Tags | Impact | Clean code attribute | exception

- Exception handlers should preserve the original exceptions — Code Smell
- Only one method invocation is expected when testing runtime exceptions — Code Smell
- Generic exceptions should never be thrown — Code Smell
- Only one method invocation is expected when testing checked exceptions — Bug
- Exception types should not be tested using "instanceof" in catch blocks — Code Smell

Exception handlers should preserve the original exceptions

Analyze your code

Consistency - Conventional | Maintainability

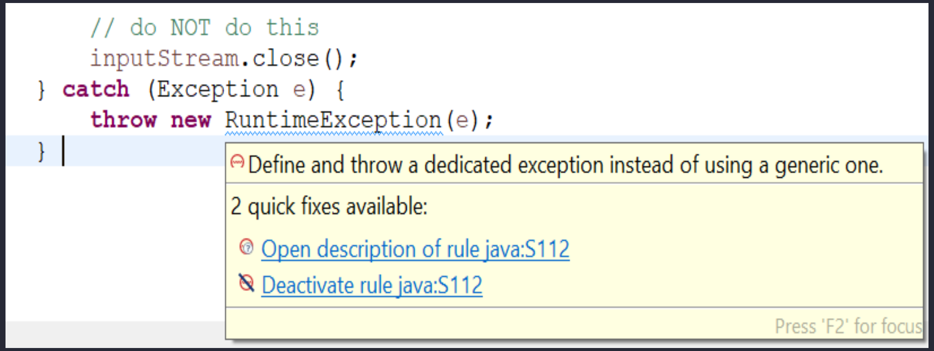
Code Smell | Major | cwe error-handling cert suspicious

Why is this an issue? | More Info

When handling a caught exception, the original exception's message and stack trace should be logged or passed forward.

Noncompliant code example

```
try {
  /* ... */
} catch (Exception e) {   // Noncompliant - exception is lost
  LOGGER.info("context");
}

try {
```



```
    // do NOT do this
    inputStream.close();
} catch (Exception e) {
    throw new RuntimeException(e);
}
```

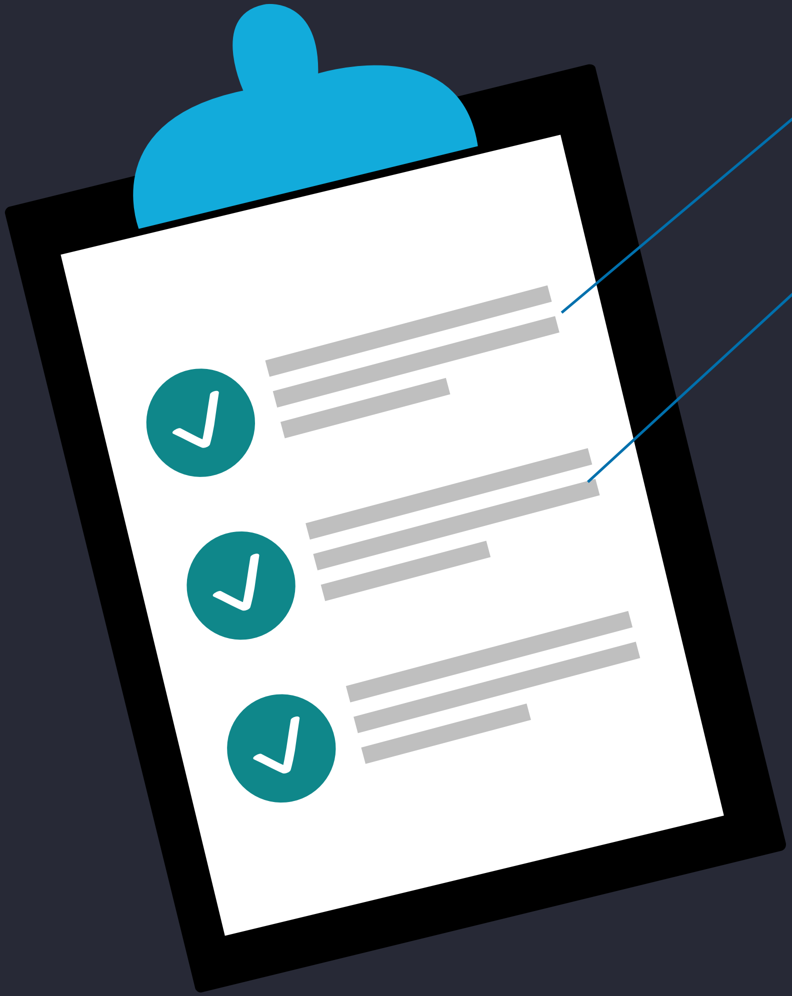
Define and throw a dedicated exception instead of using a generic one.

2 quick fixes available:

- Open description of rule java:S112
- Deactivate rule java:S112

Press 'F2' for focus

KEY TAKEAWAYS



- Simplified and streamlined flow of code.
- Ensures the Continuity and Robustness of the system.
- Improves the Readability & Maintainability of the Code and reduces cost of issue.
- Enables the use of error-recovery Mechanisms
- Improves the Scalability and Performance of the Program
- Improves end user experience

Capgemini

THANK YOU

Quiz?

When should you use the "try-with-resources" statement in Java?

- A) When handling checked exceptions
- B) When a method may throw unchecked exceptions
- [✓ C) When working with resources that need to be closed]
- D) When throwing custom exceptions

What is the primary purpose of the "finally" block in Java Exception Handling?

- A) To catch exceptions
- B) To throw exceptions
- [✓ C) To ensure code execution regardless of exceptions]
- D) To ignore exceptions

Why should you avoid catching generic exceptions like "catch (Exception e)"?

- A) It simplifies the try-catch structure.
- B) It improves code readability.
- [✓ C) It catches all exceptions, making it harder to identify specific issues.]
- D) It ensures better performance.

What is the significance of using "throws" in a method signature?

- A) It is used to throw exceptions explicitly.
- [✓ B) It indicates that the method may throw checked exceptions.]
- C) It is used for catching multiple exceptions.
- D) It is used to declare custom exceptions.

What is the purpose of the "Error" class in Java?

- A) To handle runtime exceptions
- [✓ B) To represent non-recoverable errors]
- C) To catch checked exceptions
- D) To throw custom exceptions

Which keyword is used to explicitly throw a custom exception in Java?

- [✓ A) throw]
- B) throws
- C) try
- D) catch

What is the primary purpose of logging in Java Exception Handling?

- A) To hide exceptions from users
- [✓ B) To record exception details for debugging]
- C) To replace catch blocks
- D) To throw custom exceptions

Why is it recommended to avoid empty catch blocks in Java?

- A) It reduces code complexity
- B) It allows for more efficient exception handling
- C) It helps in resource management
- [✓ D) It can mask errors and hinder debugging]

What is the role of the "try" block in Java Exception Handling?

- A) To throw exceptions
- B) To catch exceptions
- C) To ensure code execution regardless of exceptions
- [✓ D) To enclose code that may throw exceptions for handling]

In Java, which exception type must be either caught or declared to be thrown in the method signature?

- A) Unchecked Exception
- [✓ B) Checked Exception]
- C) RuntimeException
- D) Error

Q&A

Feedback

https://forms.office.com/e/XPGGbGJ5dr