

HANDLING ERROR/EXCEPTIONS

Unish Rajak

Handling Error/Exceptions

- ❑ Basic Exceptions
- ❑ Proper use of exceptions
- ❑ User defined Exceptions
- ❑ Catching Exception: try, catch
- ❑ Throwing and re-throwing: throw, throws
- ❑ Cleaning up using the finally clause.



What is Error Handling?

Error handling is the process of anticipating, detecting, and resolving unexpected issues (errors or exceptions) that occur during the execution of a program. It ensures the program doesn't crash and behaves predictably when something goes wrong.



Why is Error Handling Important?

1. Prevents program crashes and unexpected behavior.
2. Helps maintain a smooth user experience.
3. Allows developers to locate and fix bugs more easily.
4. Makes applications **robust, reliable, and professional**.

Real-World Examples of Errors

1. ATM shows “Insufficient Balance” when needed.
2. Website displays “404 Page Not Found”.
3. Login form shows “Invalid Username or Password”.



Goal of This Presentation

1. Understand different types of errors and exceptions in Java.
2. Learn how Java handles them using `try`, `catch`, `finally`, `throw`, and `throws`.
3. Be able to write code that handles errors gracefully.
4. Know best practices to avoid common mistakes.

Types of Problems in Java

Errors

1. Serious issues related to the Java Virtual Machine (JVM).
2. Cannot be handled by the program.
3. Usually caused by hardware or environment failure.

Examples:

- OutOfMemoryError
- StackOverflowError
- VirtualMachineError

Use Case:

When system runs out of memory due to infinite recursion.

Exceptions

1. Issues that occur **during program execution**.
2. Can be **handled** using error handling techniques.

• Two sub-types:

- **Checked Exceptions** 
- **Unchecked Exceptions** 

**Technically, exceptions are Java's way of saying:
"something went wrong!", and if you don't respond
to that properly, it ends in a crash.**

Checked Exceptions

1. Checked at compile time.
2. Must be handled using try-catch or declared using throws.

Examples:

- IOException
- SQLException
- FileNotFoundException

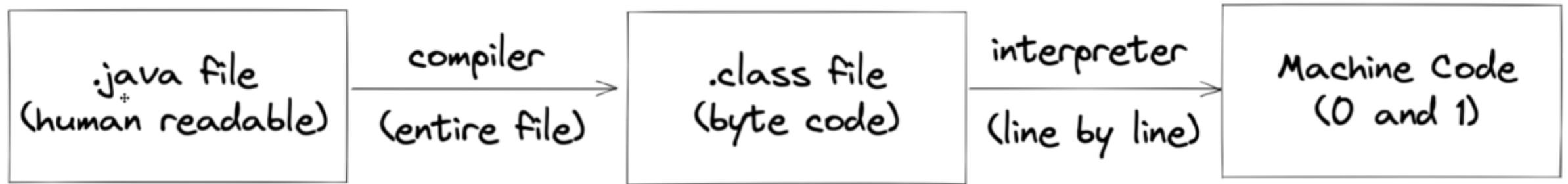
Unchecked Exceptions

1. Checked at runtime.
2. Usually due to programming mistakes.

Examples:

- ArithmeticException
- NullPointerException
- ArrayIndexOutOfBoundsException

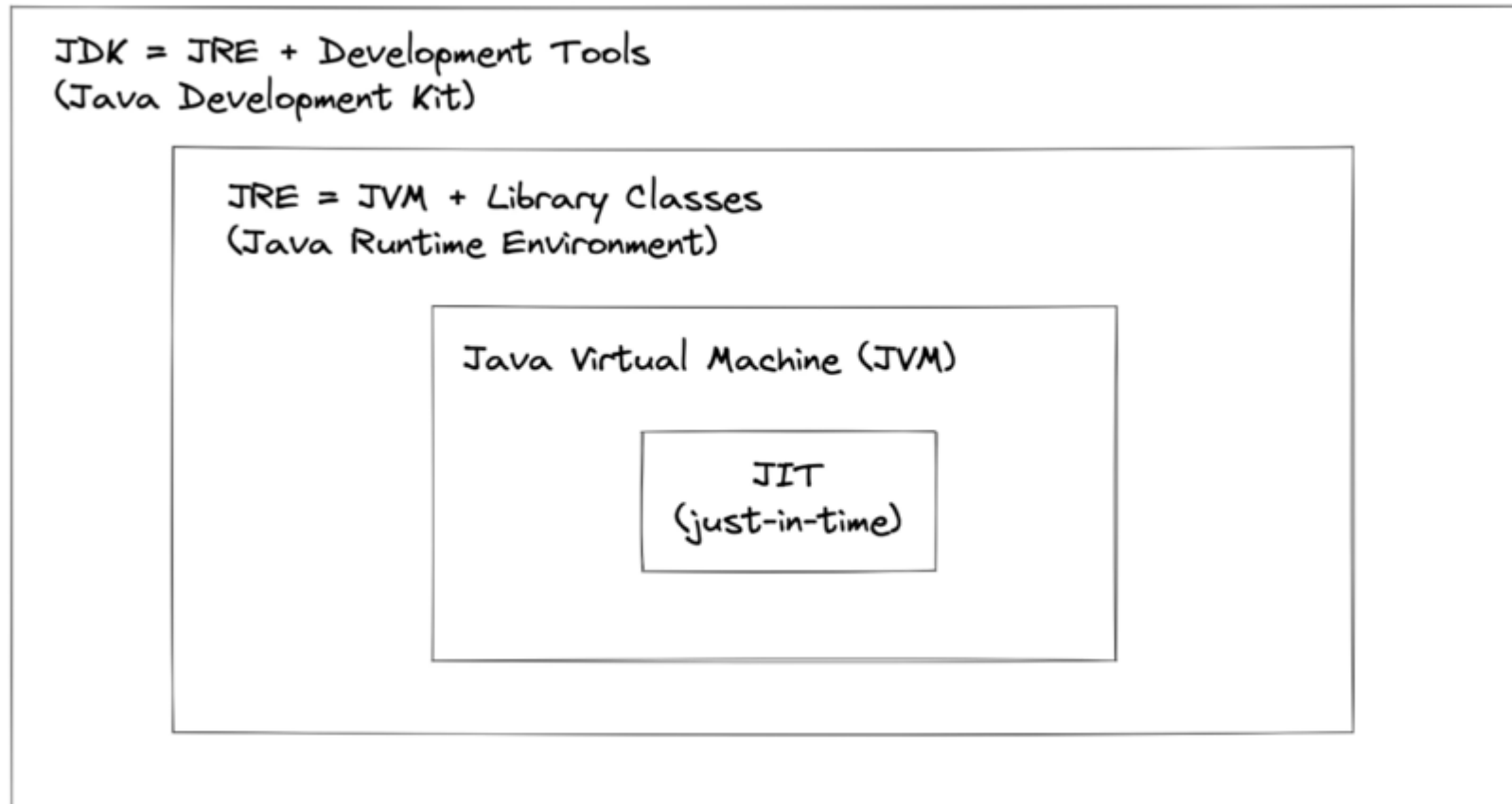
How Java code executes

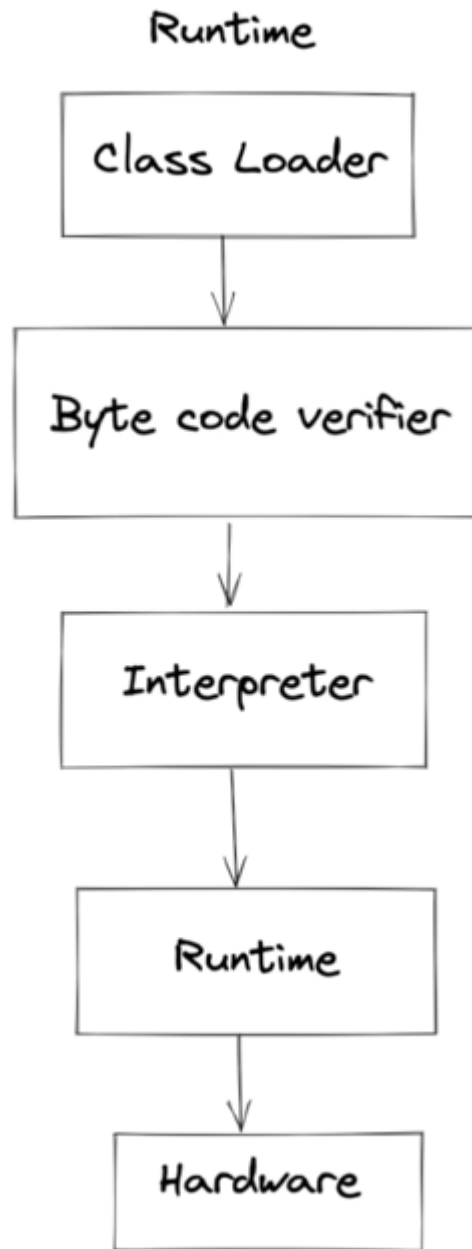


this is the source code

- this code will not directly run on a system
- we need JVM to run this
- Reason why Java is platform independent

Architecture of JAVA





(How JVM works) Class Loader

- Loading:
 - reads .class file and generate binary data
 - an object of this class is created in heap
- Linking
 - JVM verifies the .class file
 - allocates memory for class variables & default values
 - replace symbolic references from the type with direct references
- Initialization
 - all static variables are assigned with their values defined in the code and static block

JVM contains the Stack and Heap memory allocations.

Java Virtual Machine (JVM)

JIT
(just-in-time)

JVM Execution

Interpreter:

- Line by line execution
- when one method is called many times, it will interpret again and again

JIT:

- those methods that are repeated, JIT provides direct machine code so re-interpretation is⁺not required.
- makes execution faster

Garbage collector

Error/Exception	Description
ArithmeticError	Occurs when a numeric calculation goes wrong
ArrayIndexOutOfBoundsException	Occurs when trying to access an index number that does not exist in an array
ClassFormatError	Occurs when a class file cannot be accessed
ClassNotFoundException	Occurs when trying to access a class that does not exist
ConcurrentModificationException	Occurs when an element is added or removed from iterables
FileNotFoundException	Occurs when a file cannot be accessed
IncompatibleClassChangeError	Occurs when there's been a change in a base class after a child class has already been initialized
InputMismatchException	Occurs when entering wrong input (e.g. text in a numerical input)
InterruptedException	Occurs when a Thread is interrupted while waiting/sleeping
InvalidClassException	Occurs when the Serialization runtime observes a problem with a class
IOException	Occurs when an input or output operation fails

NegativeArraySizeException	Occurs when trying to create an array with negative size
NoClassDefFoundError	Occurs when the class is not found at runtime
NoSuchFieldException	Occurs when trying to access a class field/variable that does not exist
NoSuchMethodException	Occurs when trying to access a class method that does not exist
NullPointerException	Occurs when trying to access an object referece that is null
NumberFormatException	Occurs when it is not possible to convert a specified string to a numeric type
RuntimeException	Occurs when an exception occurs at runtime
StringIndexOutOfBoundsException	Occurs when trying to access a character in a String that does not exist
TypeNotPresentException	Occurs when a type cannot be found
IllegalArgumentException	Occurs when when an illegal argument is passed to a method
IllegalStateException	Occurs when when a method is called at an illegal time



Java Provides 5 Main Tools for Exception Handling

Keyword	Purpose
try	Wraps code that might throw an exception
catch	Catches and handles the exception
finally	Executes code after try/catch, always runs (e.g., cleanup code)
throw	Used to manually throw an exception
throws	Declares exceptions in method signature (used with checked exceptions)

Basic Syntax

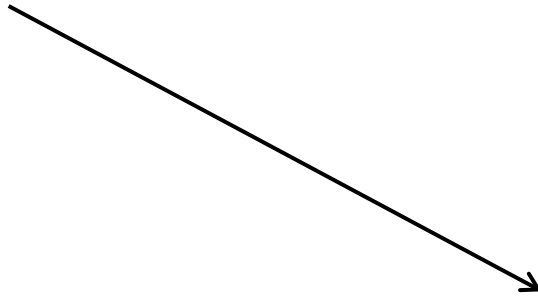
```
try {  
    // Code that might cause an exception  
} catch (ExceptionType e) {  
    // Code to handle the exception  
} finally {  
    // Code that will always run (optional)  
}
```

In Java, **every exception is an object**



Example Code

```
int result = 10 / 0;
```

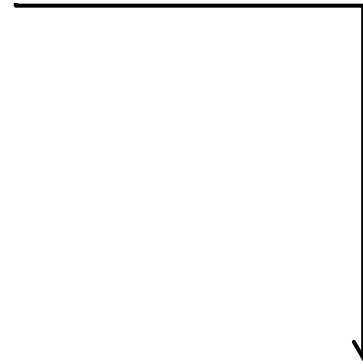


```
C:\Users\User\.jdk\openjdk-23.0.1\bin\java.exe "-javaagent:C:\Program Files\JetBra  
Exception in thread "main" java.lang.ArithmeticException: / by zero  
    at Main.main(Main.java:5)
```



Example Code

```
try {  
    int result = 10 / 0;  
} catch (ArithmeticException e) {  
    System.out.println("Can't divide by zero!");  
} finally {  
    System.out.println("Always executed!");  
}
```



e is the variable name that holds the caught exception object.

```
C:\Users\User\.jdk\openjdk-23.0.1\bin\java.exe  
Can't divide by zero  
Always executed  
|  
Process finished with exit code 0
```



Example Code

```
try {  
    int result = 10 / 1;  
} catch (ArithmeticException e) {  
    System.out.println("Can't divide by zero!");  
} finally {  
    System.out.println("Always executed!");  
}
```

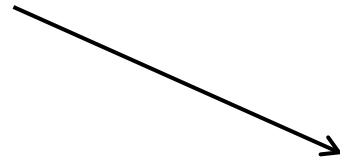


```
C:\Users\User\.jdk\openjdk-23.0.1\bin\java.exe  
Always executed  
  
Process finished with exit code 0
```



Example Code

```
try {  
    int result = sc.nextInt();  
} catch (ArithmeticException e) {  
    System.out.println("Can't divide by zero!");  
} finally {  
    System.out.println("Always executed!");  
}
```

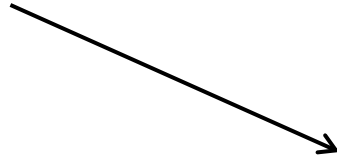


```
C:\Users\User\.jdk\openjdk-23.0.1\bin\java.exe "-javaagent:C:\Progr  
w  
Always executed  
Exception in thread "main" java.util.InputMismatchException Create brea  
    at java.base/java.util.Scanner.throwFor(Scanner.java:964)  
    at java.base/java.util.Scanner.next(Scanner.java:1619)  
    at java.base/java.util.Scanner.nextInt(Scanner.java:2284)  
    at java.base/java.util.Scanner.nextInt(Scanner.java:2238)  
    at ExampleCode.main(ExampleCode.java:10)  
  
Process finished with exit code 1
```



Example Code

```
try{
    int result = sc.nextInt();
} catch (Exception err){
    System.out.println("SOmething went wrong");
} finally {
    System.out.println("Always executed");
}
```



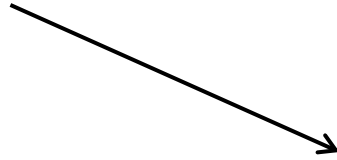
```
C:\Users\User\.jdk\openjdk-23.0.1\bin
W
SOmething went wrong
Always executed

Process finished with exit code 0
|
```



Example Code

```
try{
    int result = 10/0;
} catch (ArithmeticException err){
    System.out.println(err);
} finally {
    System.out.println("Always executed");
}
```



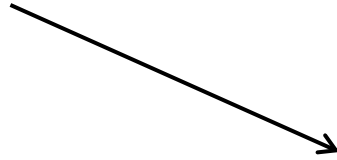
```
C:\Users\User\.jdk\openjdk-23.0.1\bin\jav
java.lang.ArithmeticException: / by zero
Always executed

Process finished with exit code 0
```



Example Code

```
try{
    int result = 10/0;
} catch (ArithmeticException err){
    System.out.println(err.getMessage());
} finally {
    System.out.println("Always executed");
}
```



```
C:\Users\User\.jdk\openjdk-23.0.1\
/ by zero
Always executed

Process finished with exit code 0
```

Use Case

Imagine reading a file. If the file isn't found, Java can catch that issue using these tools and prevent a crash.



Using Finally Block

```
Scanner scanner = new Scanner(System.in);
int secretNumber = 7; // fixed number for demo

try {
    System.out.println("Welcome to the Guessing Game!");
    System.out.print("Guess a number between 1 and 10: ");
    int guess = scanner.nextInt(); // risky line

    if (guess == secretNumber) {
        System.out.println("🎉 Correct! You win!");
    } else {
        System.out.println("❌ Wrong guess. Try again next time.");
    }
} catch (Exception e) {
    System.out.println("⚠️ Invalid input! Please enter a number.");
} finally {
    System.out.println("Thanks for playing the game! 🙌");
}

scanner.close();
```

Explanation

- try: Tests a block of code for errors.
- catch: Handles the specific error.
- finally: Runs regardless of whether an exception occurred.
- throw: Used to raise an exception manually.
- throws: Declares that a method can throw exceptions.

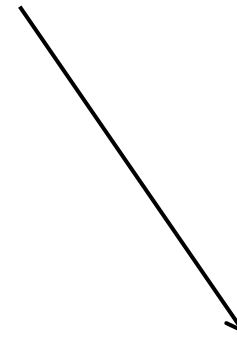
Using throw

```
Scanner scanner = new Scanner(System.in);

System.out.print("Enter your age: ");
int age = scanner.nextInt();

if (age < 18) {
    // Throwing an exception manually
    throw new ArithmeticException();
} else {
    System.out.println("Access granted - You are old enough!");
}

scanner.close();
```



```
C:\Users\User\.jdk\openjdk-23.0.1\bin\java.exe "-javaagent
Enter your age: 15
Exception in thread "main" java.lang.ArithmeticException C
    at Throw.main(Throw.java:12)

Process finished with exit code 1
```

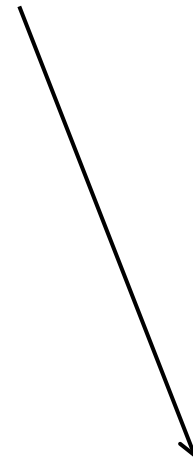
Using throw

```
Scanner scanner = new Scanner(System.in);

System.out.print("Enter your age: ");
int age = scanner.nextInt();

if (age < 18) {
    // Throwing an exception manually
    throw new ArithmeticException("Access denied – You are not old enough");
} else {
    System.out.println("Access granted - You are old enough!");
}

scanner.close();
```

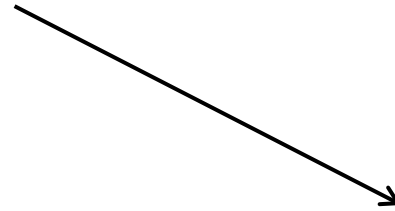


```
C:\Users\User\.jdk\openjdk-23.0.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community E
Enter your age: 17
Exception in thread "main" java.lang.ArithmeticException Create breakpoint : Access denied - You are not old enough
    at Throw.main(Throw.java:12)
```

Using throw with catch

```
try {
    System.out.print("Enter your age: ");
    int age = scanner.nextInt();

    if (age < 18) {
        throw new ArithmeticException();
    } else {
        System.out.println("Access granted - You are old enough!");
    }
} catch (ArithmeticException e) {
    System.out.println(" ⚠ Exception caught: " + e);
} finally {
    System.out.println("Program ended.");
    scanner.close();
}
```



```
C:\Users\User\.jdk\openjdk-23.0.1\bin\java.exe "-j
Enter your age: 17
⚠ Exception caught: java.lang.ArithmeticException
Program ended.

Process finished with exit code 0
```

Why use throws?

```
public class Demo {  
    public static void main(String[] args) {  
        A obj = new A();  
        obj.show();  
    }  
}
```

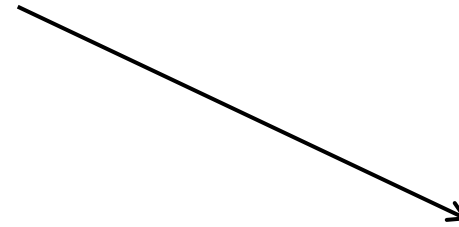


```
C:\Users\User\.jdk\openjdk-23.0.1\bin\java.exe  
Not able to find the class  
  
Process finished with exit code 0
```

```
class A{  
    public void show(){  
        try{  
            Class.forName("Throw");  
        }  
        catch (ClassNotFoundException e){  
            System.out.println("Not able to find the class");  
        }  
    }  
}
```

Throws Example

```
class A {  
    public void show() throws ClassNotFoundException {  
        Class.forName("Playground");  
    }  
}  
  
public class Demo {  
    public static void main(String[] args) {  
        A obj = new A();  
  
        try {  
            obj.show();  
        } catch (ClassNotFoundException e) {  
            System.out.println("Not able to find the class");  
        }  
    }  
}
```

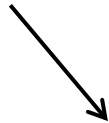


```
C:\Users\User\.jdk\openjdk-23.0.1\bin  
Not able to find the class  
  
Process finished with exit code 0
```

Custom Exception

```
try {
    System.out.print("Enter your age: ");
    int age = scanner.nextInt();

    if (age < 18) {
        throw new RuntimeException("Access denied - You must be at least 18 years old.");
    } else {
        System.out.println("Access granted - You are old enough!");
    }
} catch (RuntimeException e) {
    System.out.println(" ⚠ Exception caught: " + e);
}
```



```
Enter your age: 15
⚠ Exception caught: java.lang.RuntimeException: Access denied - You must be at least 18 years old.
```


Custom Exception

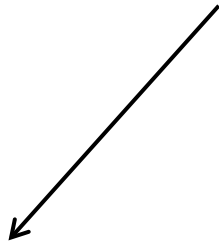
```
class AafnoException {  
    public AafnoException() {  
  
    }  
}
```

```
public class CustomException {  
    public static void main(String[] args) {  
        Scanner scanner = new Scanner(System.in);  
  
        try {  
            System.out.print("Enter your age: ");  
            int age = scanner.nextInt();  
  
            if (age < 18) {  
                throw new RuntimeException("Access denied - You must be at least 18 years old.");  
            } else {  
                System.out.println("Access granted - You are old enough!");  
            }  
        } catch (RuntimeException e) {  
            System.out.println("⚠ Exception caught: " + e);  
        }  
    }  
}
```

Custom Exception

```
class AafnoException {  
    public AafnoException() {  
  
    }  
}
```

```
public class CustomException {  
    public static void main(String[] args) {  
        Scanner scanner = new Scanner(System.in);  
  
        try {  
            System.out.print("Enter your age: ");  
            int age = scanner.nextInt();  
  
            if (age < 18) {  
                throw new AafnoException("Access denied - You must be at least 18 years old.");  
            } else {  
                System.out.println("Access granted - You are old enough!");  
            }  
        } catch (AafnoException e) {  
            System.out.println("⚠ Exception caught: " + e);  
        }  
    }  
}
```

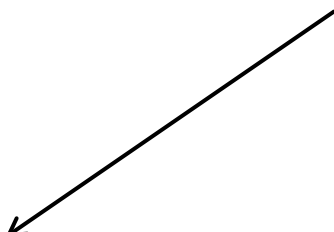


```
java: constructor AafnoException in class AafnoException cannot be applied to given types;  
  required: no arguments  
  found:    java.lang.String  
  reason: actual and formal argument lists differ in length
```

Custom Exception

```
class AafnoException extends RuntimeException {  
    public AafnoException() {  
    }  
}
```

```
public class CustomException {  
    public static void main(String[] args) {  
        Scanner scanner = new Scanner(System.in);  
  
        try {  
            System.out.print("Enter your age: ");  
            int age = scanner.nextInt();  
  
            if (age < 18) {  
                throw new AafnoException();  
            } else {  
                System.out.println("Access granted - You are old enough!");  
            }  
        } catch (AafnoException e) {  
            System.out.println("⚠ Exception caught: " + e);  
        }  
    }  
}
```



```
C:\Users\User\.jdk\openjdk-23.0.1\bin  
Enter your age: 14  
⚠ Exception caught: AafnoException  
  
Process finished with exit code 0
```

Custom Exception



```
RuntimeException.java x ExampleCode.java GuessGame.java Throw.java

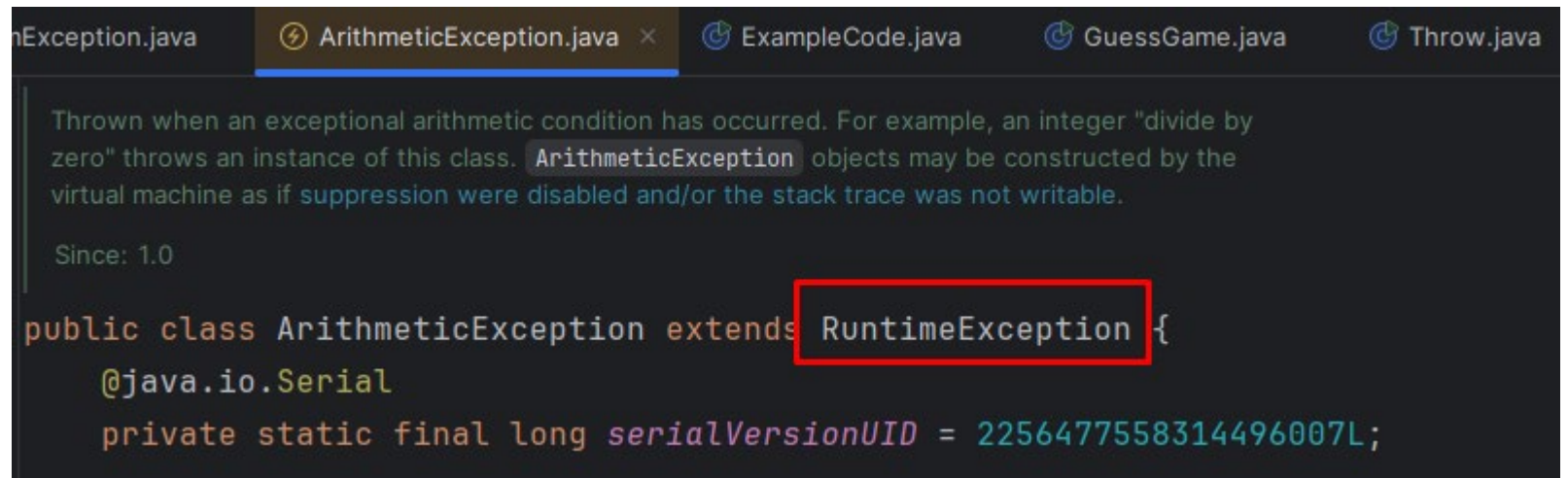
11.2 Compile-Time Checking of Exceptions

public class RuntimeException extends Exception {
    @java.io.Serial
    static final long serialVersionUID = -7034897190745766939L;

    Constructs a new runtime exception with null as its detail message. The cause is not
    initialized, and may subsequently be initialized by a call to initCause().

    public RuntimeException() { super(); }

    Constructs a new runtime exception with the specified detail message. The cause is not
```



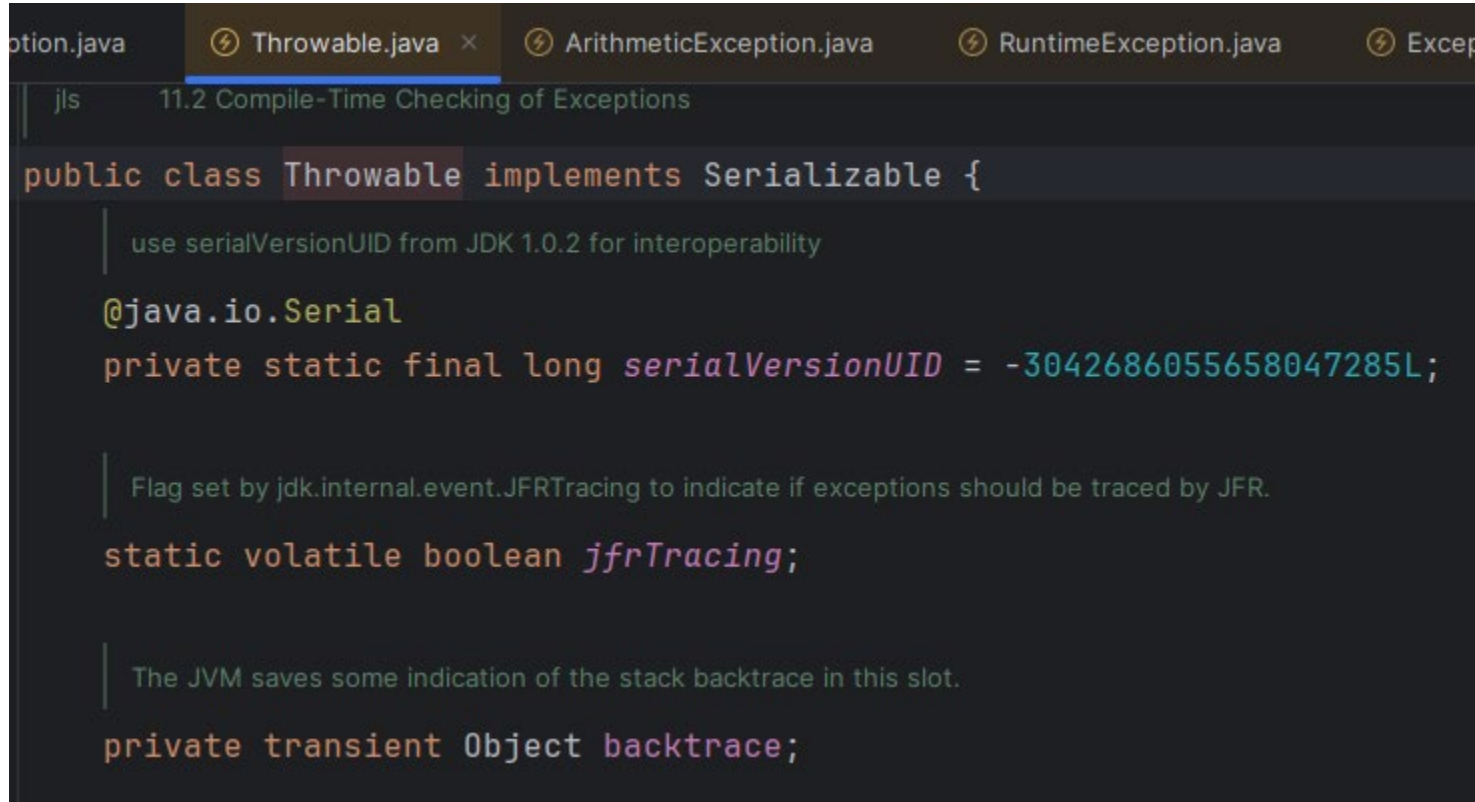
```
RuntimeException.java x ArithmeticException.java x ExampleCode.java GuessGame.java Throw.java

Thrown when an exceptional arithmetic condition has occurred. For example, an integer "divide by
zero" throws an instance of this class. ArithmeticException objects may be constructed by the
virtual machine as if suppression were disabled and/or the stack trace was not writable.

Since: 1.0

public class ArithmeticException extends RuntimeException {
    @java.io.Serial
    private static final long serialVersionUID = 2256477558314496007L;
```

Custom Exception



```
Throwable.java x ArithmeticException.java RuntimeException.java Excep
jls 11.2 Compile-Time Checking of Exceptions

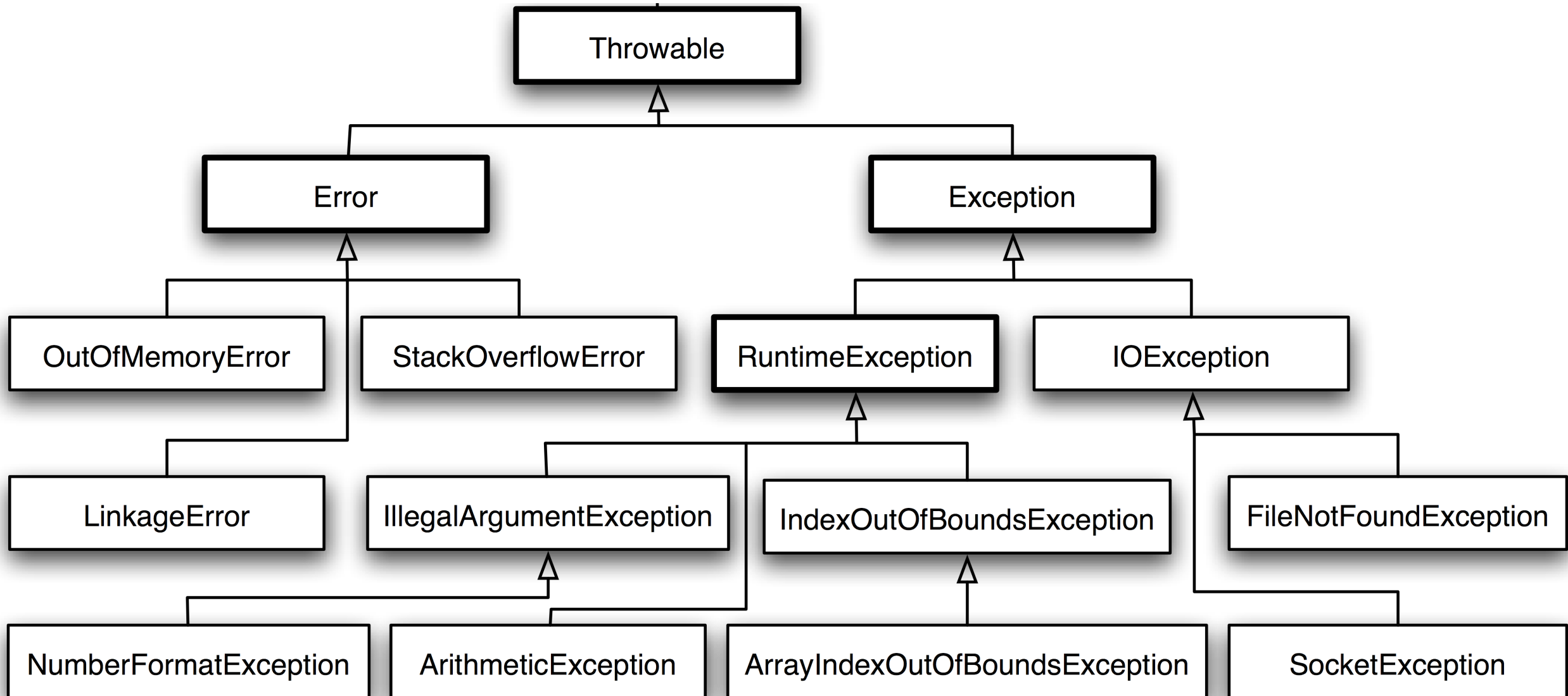
public class Throwable implements Serializable {
    use serialVersionUID from JDK 1.0.2 for interoperability

    @java.io.Serial
    private static final long serialVersionUID = -3042686055658047285L;

    Flag set by jdk.internal.event.JFRTracing to indicate if exceptions should be traced by JFR.
    static volatile boolean jfrTracing;

    The JVM saves some indication of the stack backtrace in this slot.
    private transient Object backtrace;
```

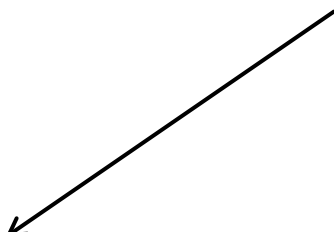
inheritance



Custom Exception

```
class AafnoException extends Exception {  
    public AafnoException() {  
    }  
}
```

```
public class CustomException {  
    public static void main(String[] args) {  
        Scanner scanner = new Scanner(System.in);  
  
        try {  
            System.out.print("Enter your age: ");  
            int age = scanner.nextInt();  
  
            if (age < 18) {  
                throw new AafnoException();  
            } else {  
                System.out.println("Access granted - You are old enough!");  
            }  
  
        } catch (AafnoException e) {  
            System.out.println("⚠ Exception caught: " + e);  
        }  
    }  
}
```

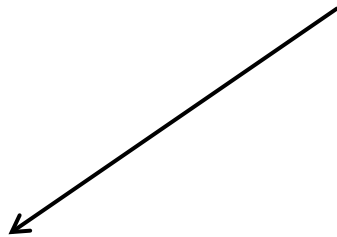


```
C:\Users\User\.jdk\openjdk-23.0.1\bin  
Enter your age: 14  
⚠ Exception caught: AafnoException  
  
Process finished with exit code 0
```

Custom Exception

```
class AafnoException extends Exception {  
    public AafnoException(String message) {  
        super(message);  
    }  
}
```

```
public class CustomException {  
    public static void main(String[] args) {  
        Scanner scanner = new Scanner(System.in);  
  
        try {  
            System.out.print("Enter your age: ");  
            int age = scanner.nextInt();  
  
            if (age < 18) {  
                throw new AafnoException();  
            } else {  
                System.out.println("Access denied - You must be at least 18 years old.");  
            }  
        } catch (AafnoException e) {  
            System.out.println(" ⚠ Exception caught: " + e);  
        }  
    }  
}
```



```
C:\Users\User\.jdk\openjdk-23.0.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA\bin\idea-agent.jar" -jar C:\Users\User\IdeaProjects\CustomException\build\libs\CustomException.jar  
Enter your age: 4  
⚠ Exception caught: AafnoException: Access denied - You must be at least 18 years old.  
  
Process finished with exit code 0
```


The End



Let's simulate a turn-based battle game (like an RPG fight) where

- The player attacks an enemy.
- The attack might fail (e.g., due to a wrong input or a simulated error).
- The game must always print the status of the game turn (no matter what happens).

What Makes This a Good Game-Based Example?

Block	What it does
try	Simulates a game turn (user chooses a move and attacks)
catch	Handles invalid moves or bad input (e.g., typing a letter instead of number)
finally	Shows game status after every turn — like a turn report

```
import java.util.Scanner;

public class BattleGame {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int playerHealth = 100;
        int enemyHealth = 80;

        try {
            System.out.println("✂️ Battle Start!");
            System.out.println("Enemy Health: " + enemyHealth);
            System.out.print("Choose your move (1 = Slash, 2 = Fireball): ");
            int move = scanner.nextInt(); // risky input

            int damage = 0;

            if (move == 1) {
                damage = 20;
                System.out.println("You used SLASH! ✂️");
            } else if (move == 2) {
                damage = 30;
                System.out.println("You cast FIREBALL! 🔥");
            } else {
                throw new IllegalArgumentException("Invalid move chosen!");
            }
        }
```

```
        enemyHealth -= damage;
        System.out.println("You dealt " + damage + " damage!");

        } catch (Exception e) {
            System.out.println("⚠️ Error during your turn: " + e);
            System.out.println("⚠️ Make sure to choose a valid move next turn.");
        } finally {
            System.out.println("🔄 Turn ended. Checking game state...");
            System.out.println("Player Health: " + playerHealth);
            System.out.println("Enemy Health: " + enemyHealth);
        }

        scanner.close();
    }
}
```

