customize
methods
keywords
handle
types
exceptions

Exceptions in Java

Advanced Programming
Dr. Mojtaba Vahidi Asl
Sara Shiri
Spring 1404

customize

methods

keywords

handle

types

Exception in Programming:

- An exception is an unwanted or unexpected event that occurs during the execution of a program at runtime.

- Exceptions disrupt the normal flow of a program’s instructions.

exceptions

customize

methods

keywords

handle

types

Nature of Exceptions:

- Errors encountered during runtime create exception objects.

- These objects contain information about the exception, such as:

  - Name of the exception.
  - Description of the exception.
  - The state of the program when the exception occurred.

exceptions

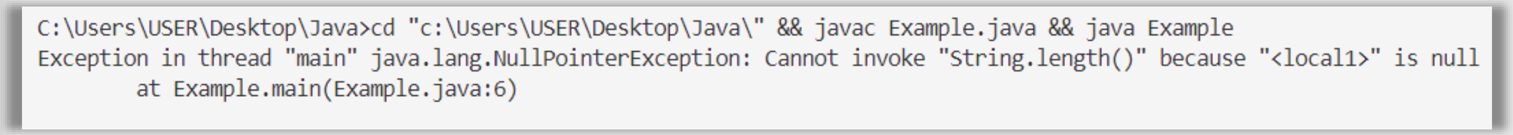
customize

methods

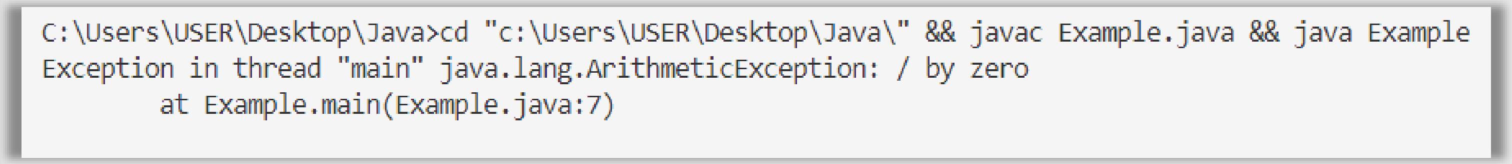
keywords

handle

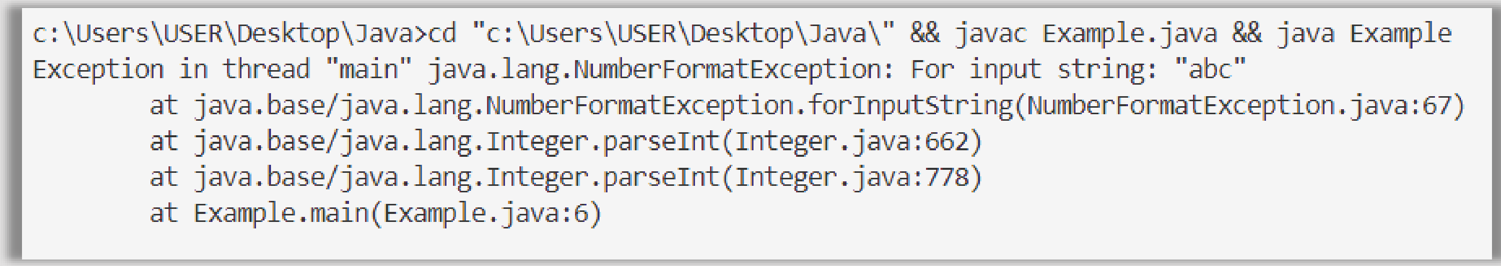
types



```
C:\Users\USER\Desktop\Java>cd "c:\Users\USER\Desktop\Java\" && javac Example.java && java Example
Exception in thread "main" java.lang.NullPointerException: Cannot invoke "String.length()" because "<local1>" is null
        at Example.main(Example.java:6)
```



```
C:\Users\USER\Desktop\Java>cd "c:\Users\USER\Desktop\Java\" && javac Example.java && java Example
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at Example.main(Example.java:7)
```



```
c:\Users\USER\Desktop\Java>cd "c:\Users\USER\Desktop\Java\" && javac Example.java && java Example
Exception in thread "main" java.lang.NumberFormatException: For input string: "abc"
        at java.base/java.lang.NumberFormatException.forInputString(NumberFormatException.java:67)
        at java.base/java.lang.Integer.parseInt(Integer.java:662)
        at java.base/java.lang.Integer.parseInt(Integer.java:778)
        at Example.main(Example.java:6)
```

exceptions

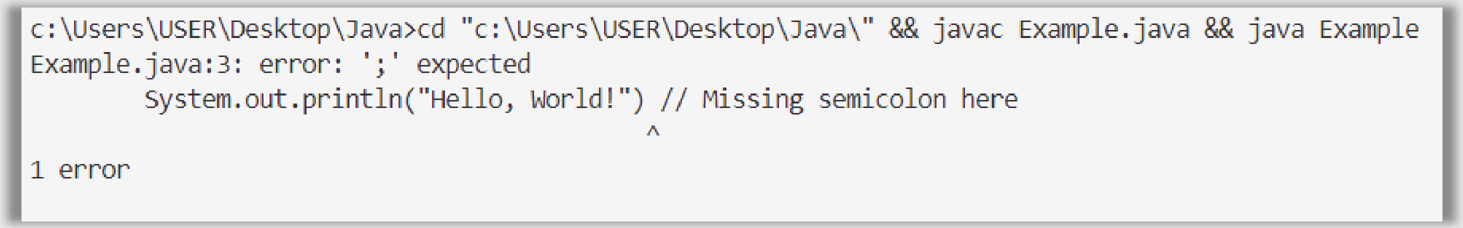
customize

methods

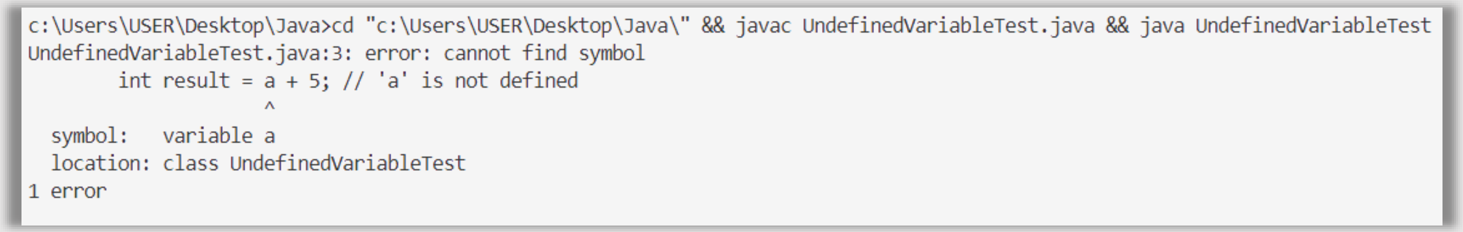
keywords

handle

types



```
c:\Users\USER\Desktop\Java>cd "c:\Users\USER\Desktop\Java\" && javac Example.java && java Example
Example.java:3: error: ';' expected
        System.out.println("Hello, World!") // Missing semicolon here
                                           ^
1 error
```



```
c:\Users\USER\Desktop\Java>cd "c:\Users\USER\Desktop\Java\" && javac UndefinedVariableTest.java && java UndefinedVariableTest
UndefinedVariableTest.java:3: error: cannot find symbol
        int result = a + 5; // 'a' is not defined
                     ^
  symbol:   variable a
  location: class UndefinedVariableTest
1 error
```

These are Errors and not Exceptions!

exceptions

customize

methods

keywords

handle

types

Errors Vs. Exceptions:

Errors:

- Indicate unrecoverable system issues.
- Are beyond the control of the program.

Exceptions:

- Represent unexpected events within the program.
- Can often be handled gracefully by the program.

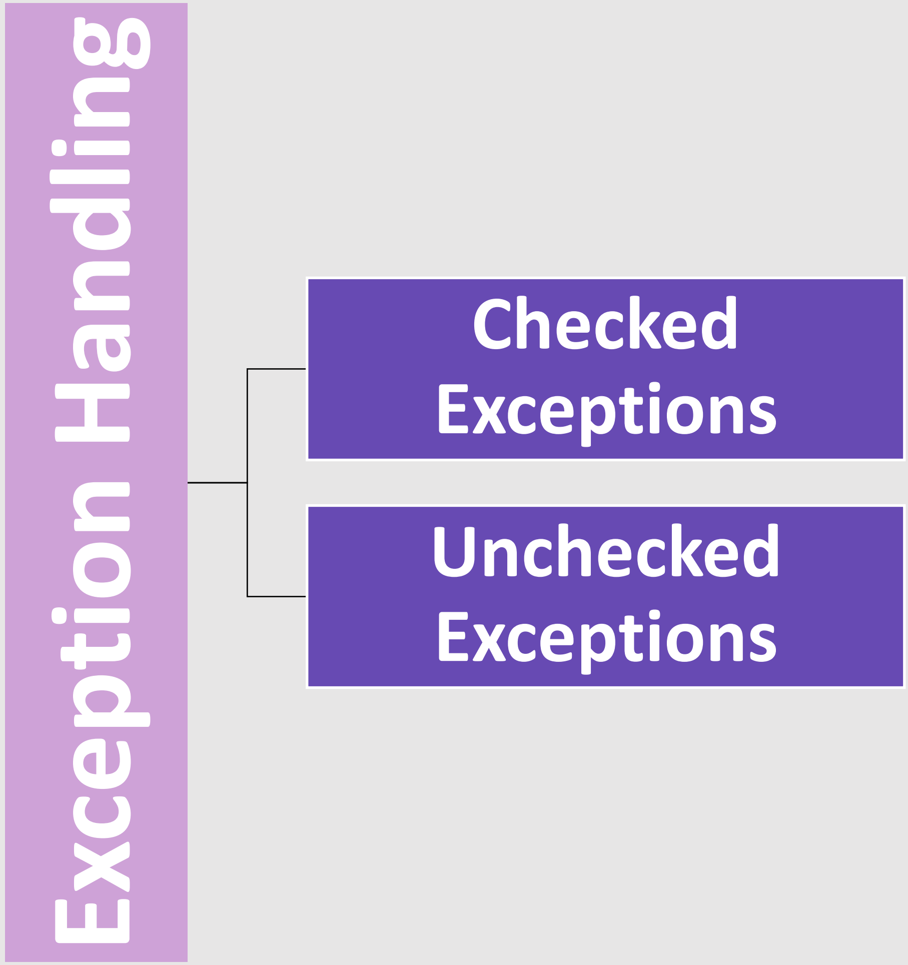
exceptions

customize

methods

keywords

handle



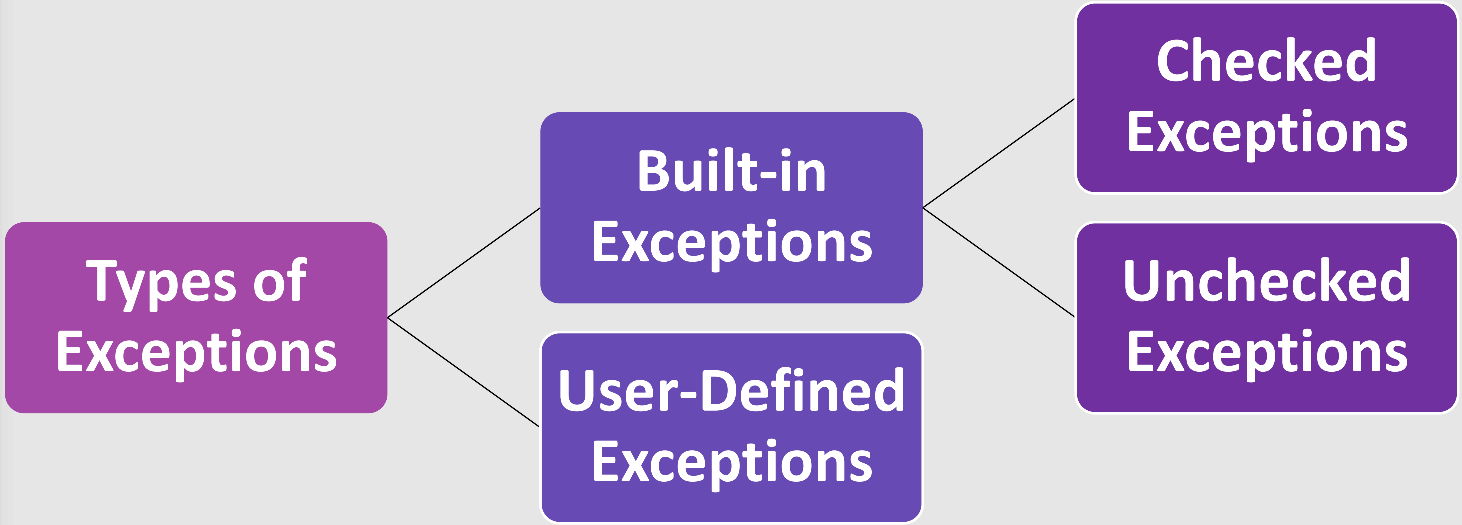
Exception Handling

Checked Exceptions

Unchecked Exceptions

types

exceptions



Types of Exceptions

Built-in Exceptions

User-Defined Exceptions

Checked Exceptions

Unchecked Exceptions

We’ll find out about User-Defined Exceptions later.

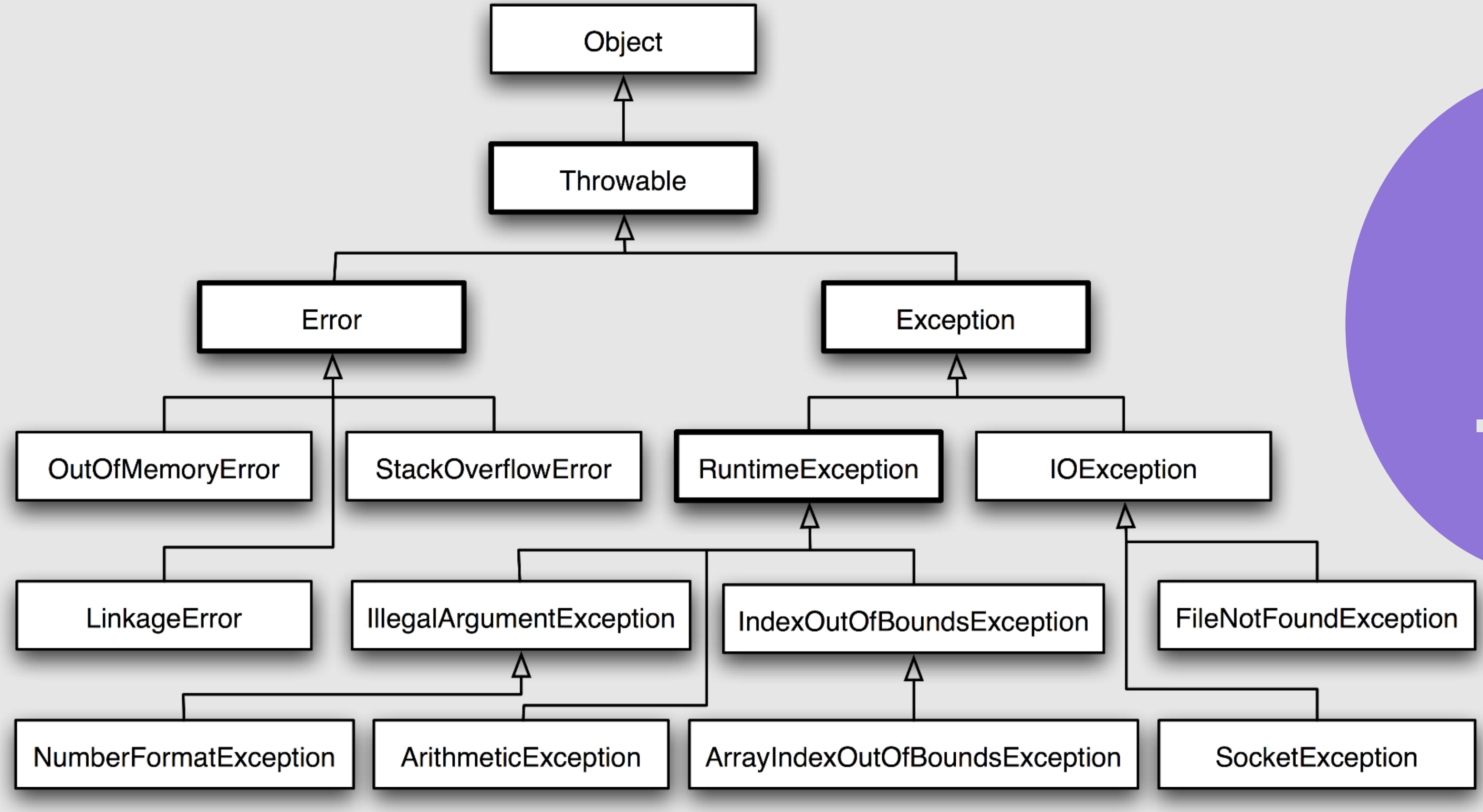
customize

methods

keywords

handle

Exceptions Hierarchy



types

exceptions

customize
methods
keywords
handle

Checked Exception:

- These are exceptions where the classes directly inherit the Throwable class, excluding RuntimeException and Error.
- Checked exceptions are verified at compile-time.
- Examples: IOException, SQLException, etc.

types
exceptions

customize

methods

keywords

handle

Unchecked Exception:

- These are exceptions where the classes inherit the RuntimeException.
- Unchecked exceptions are not checked at compile-time, but are checked at runtime.
- Examples: ArithmeticException, NullPointerException, etc.

types

exceptions

How the JVM Handles Exceptions?

- The JVM plays a critical role in exception handling.
- Here’s how it manages exceptions:

1. **Throwing an Exception**
2. **Searching for an Exception Handler**
3. **Finding the Handler**
4. **Uncaught Exceptions**

customize

methods

keywords

handle

types

exceptions

customize

methods

keywords

1. Throwing an Exception:

- When an exception occurs inside a method, the method creates an exception object.

- The exception object contains details about the error, such as:

  - The type of error.
  - The state of the program when the error occurred.

handle

types

exceptions

customize

methods

keywords

2. Searching for an Exception Handler:

- Once an exception is thrown, the JVM starts searching for an appropriate exception handler.
- The search begins in the method where the exception occurred.
- The search continues up the call stack, going through the methods that were called to reach the current method.

handle

types

exceptions

customize

methods

keywords

3. Finding the Handler:

- If the JVM finds a method in the call stack with a matching catch block, it hands over the exception to that handler.

- The exception handler processes the exception.

handle

types

exceptions

customize

methods

keywords

4. Uncaught Exceptions:

- If the JVM doesn’t find a matching handler, it uses the default exception handler.

- The default handler:

  - Prints the exception’s stack trace, listing all the method calls in progress when the exception was thrown.
  - Terminates the program.

handle

types

exceptions

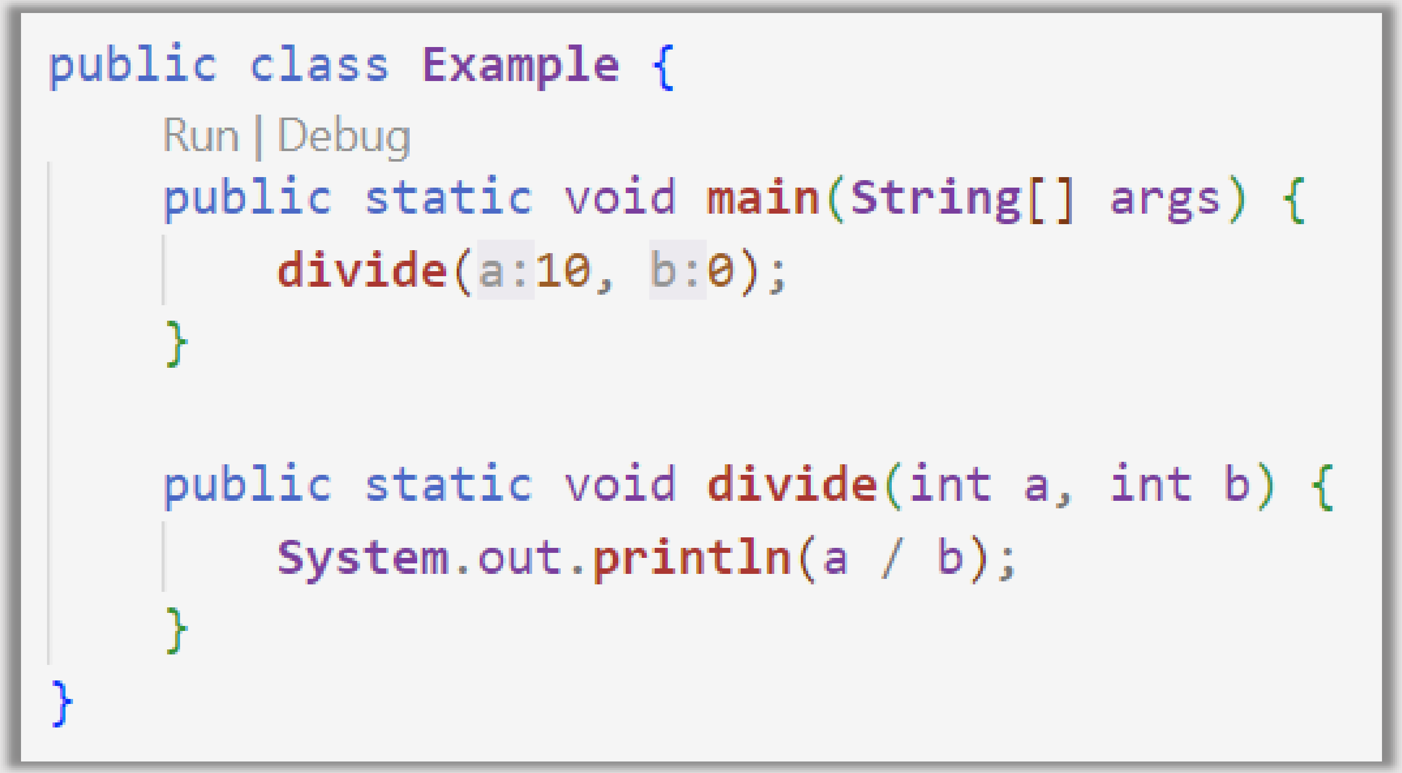
customize

methods

keywords

Example of JVM Handling an Exception

Suppose you have a program where a method tries to divide by zero:



```
public class Example {
          Run | Debug
          public static void main(String[] args) {
              divide(a:10, b:0);
          }

          public static void divide(int a, int b) {
              System.out.println(a / b);
          }
}
```

handle

types

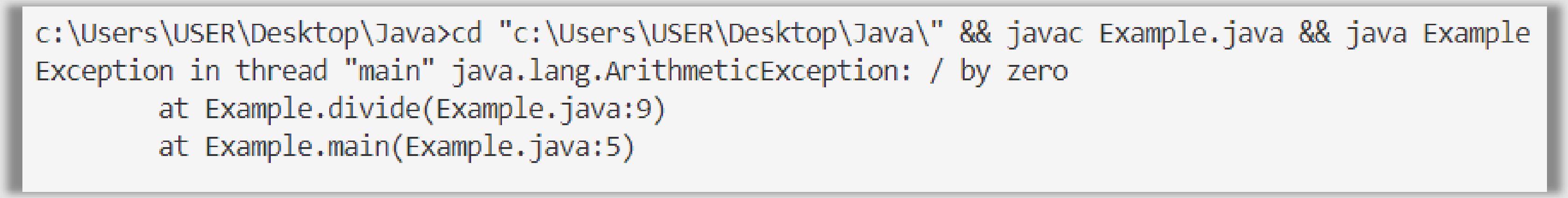
exceptions

- The divide method tries to execute a / b, which throws an ArithmeticException.

- The JVM searches for an appropriate handler in the divide method

- It then checks the main method.

- Finally, the JVM invokes the default exception handler, which prints something like this:



```
c:\Users\USER\Desktop\Java>cd "c:\Users\USER\Desktop\Java\" && javac Example.java && java Example
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at Example.divide(Example.java:9)
        at Example.main(Example.java:5)
```

customize

methods

How Programmers Handle Exceptions?

In Java, handling exceptions is done using five key keywords:

1. **try**
2. **catch**
3. **finally**
4. **throw**
5. **throws**

keywords

handle

types

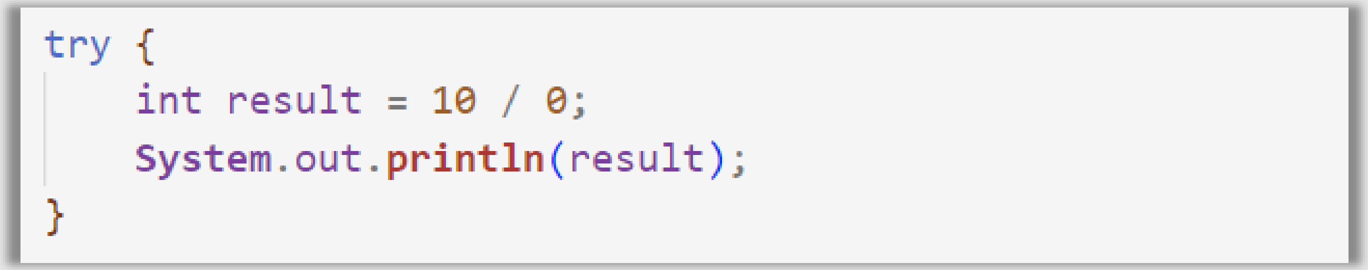
exceptions

customize

methods

1. The try Block

- The try block is used to enclose code that might throw an exception.
- Potentially problematic code is wrapped within a try block.
- This allows the program to catch and handle exceptions if they occur.



```
try {
    int result = 10 / 0;
    System.out.println(result);
}
```

- The try block tells Java to "try" running this code, but be prepared in case something goes wrong.

keyword

handle

types

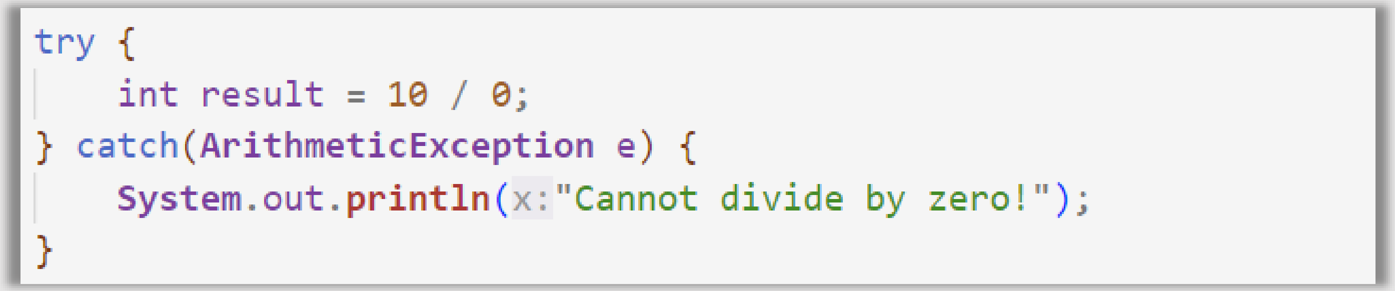
exceptions

customize

methods

2. The catch Block

- If an exception occurs within the try block, control is immediately transferred to the corresponding catch block.
- The catch block is where the exception is handled.
- You can have multiple catch blocks to handle different types of exceptions separately.



```
try {
    int result = 10 / 0;
} catch(ArithmeticException e) {
    System.out.println(x:"Cannot divide by zero!");
}
```

- Here, if an exception occurs, the catch block will execute, allowing you to handle the error gracefully instead of crashing the program.

keyword

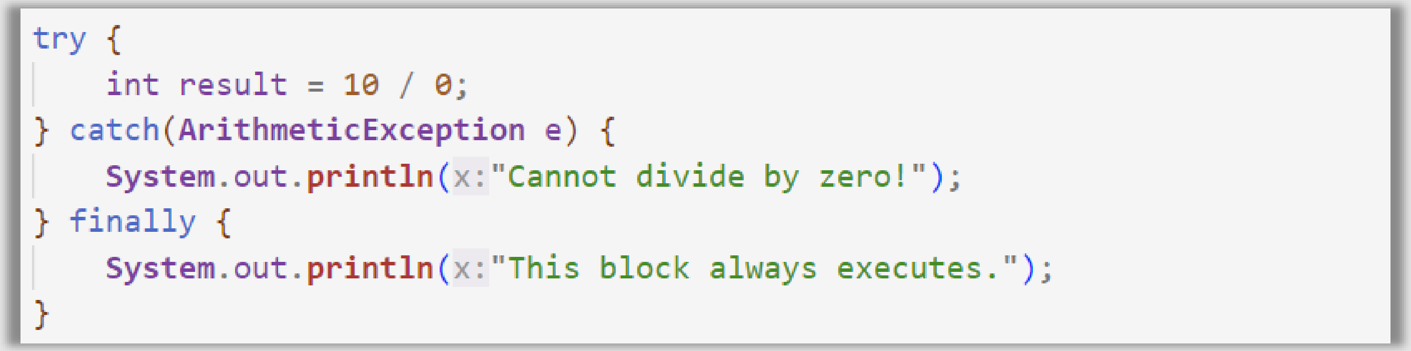
handle

types

exceptions

3. The finally Block

- The finally block is optional in a try-catch-finally structure.
- It is executed regardless of whether an exception is thrown or not.
- Commonly used for cleanup operations.



```
try {
    int result = 10 / 0;
} catch(ArithmeticException e) {
    System.out.println(x:"Cannot divide by zero!");
} finally {
    System.out.println(x:"This block always executes.");
}
```

- The finally block will execute regardless of whether an exception was caught or not.

customize

methods

keyword

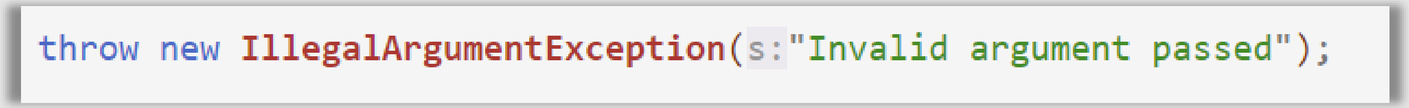
handle

types

exceptions

4. The throw Keyword

- You can throw an exception manually, even if one hasn’t occurred yet.
- This is useful for enforcing specific conditions in your program.
- The throw keyword is used to manually trigger an exception.

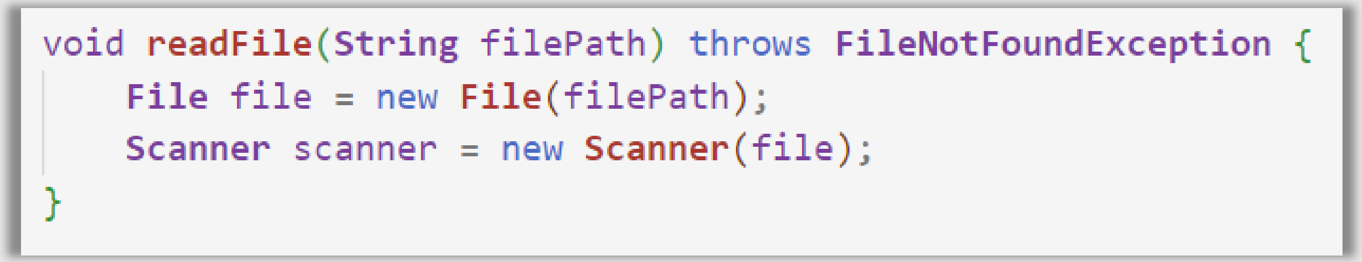


```
throw new IllegalArgumentException(s:"Invalid argument passed");
```

- This code creates and throws a new exception, which you can catch elsewhere in your program.

5. The throws Keyword

- If a method can throw an exception that it doesn’t handle, it must declare this using the throws keyword.
- This declaration informs the caller of the method that it needs to handle the potential exception.



```
void readFile(String filePath) throws FileNotFoundException {
    File file = new File(filePath);
    Scanner scanner = new Scanner(file);
}
```

- Here, the readFile method declares that it might throw a FileNotFoundException.
- The caller of this method needs to either handle this exception or declare it using throws as well.

customize

methods

keyword

handle

types

exceptions

customize

Exception Methods in Java

- These methods help you retrieve useful information about the exception, such as the error message, the cause, and the stack trace.
- Here’s a rundown of the commonly used exception methods:

1. **getMessage()**
2. **toString()**
3. **printStackTrace()**
4. **getStackTrace()**
5. **getCause()**
6. **initCause()**

methods

keyword

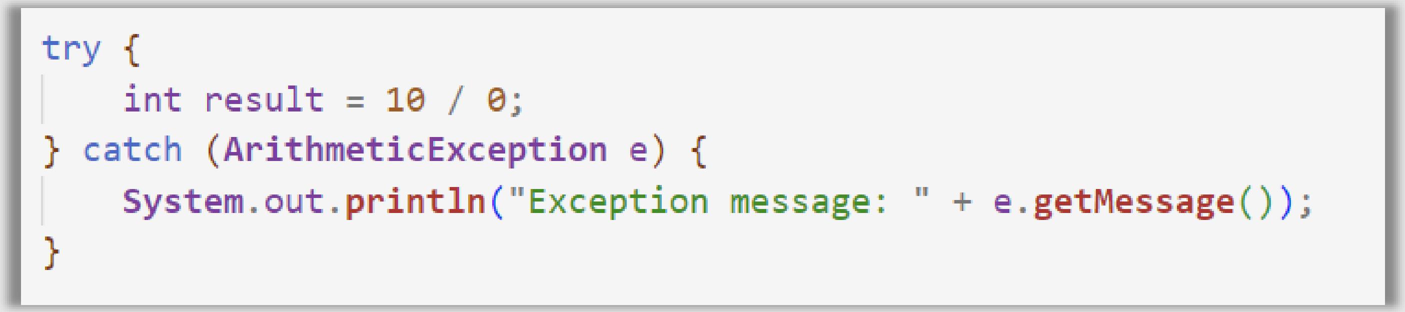
handle

types

exceptions

customize

getMessage() :

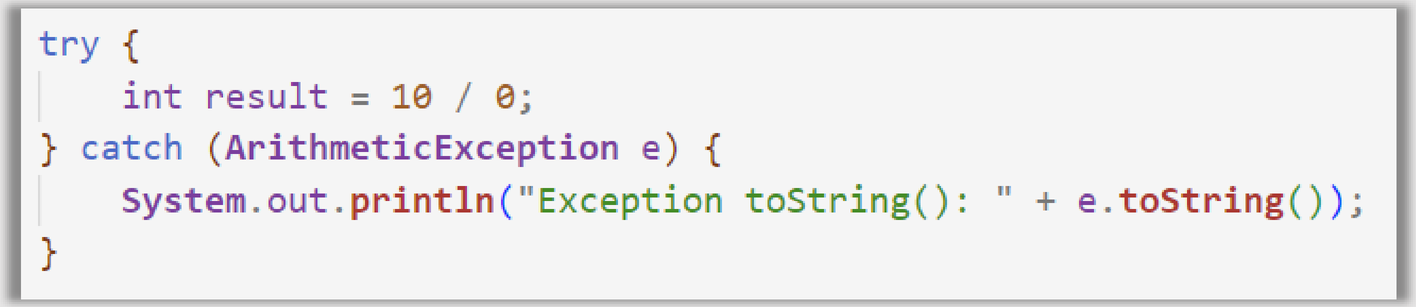


```
try {
    int result = 10 / 0;
} catch (ArithmeticException e) {
    System.out.println("Exception message: " + e.getMessage());
}
```

Output:
Exception message: / by zero

methods
keyword
handle
types
exceptions

toString() :

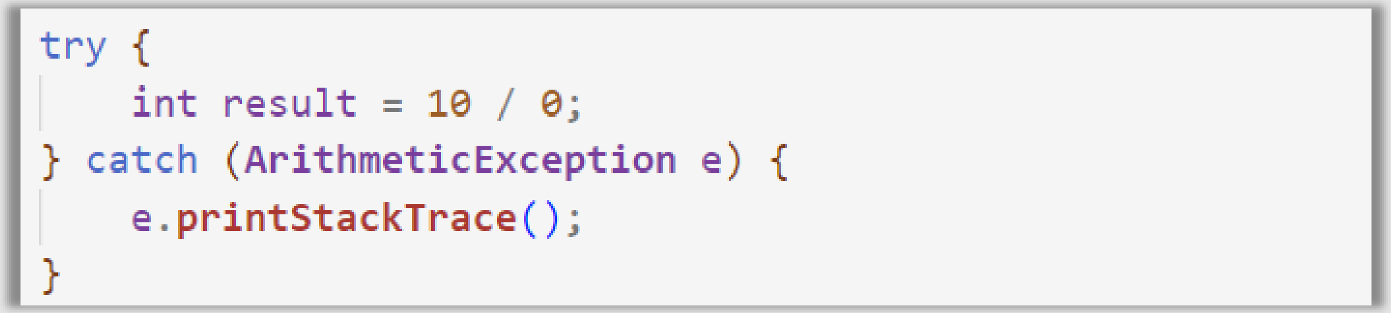


```
try {
    int result = 10 / 0;
} catch (ArithmeticException e) {
    System.out.println("Exception toString(): " + e.toString());
}
```

Output:
Exception toString():
java.lang.ArithmeticException: / by zero

customize

printStackTrace() :



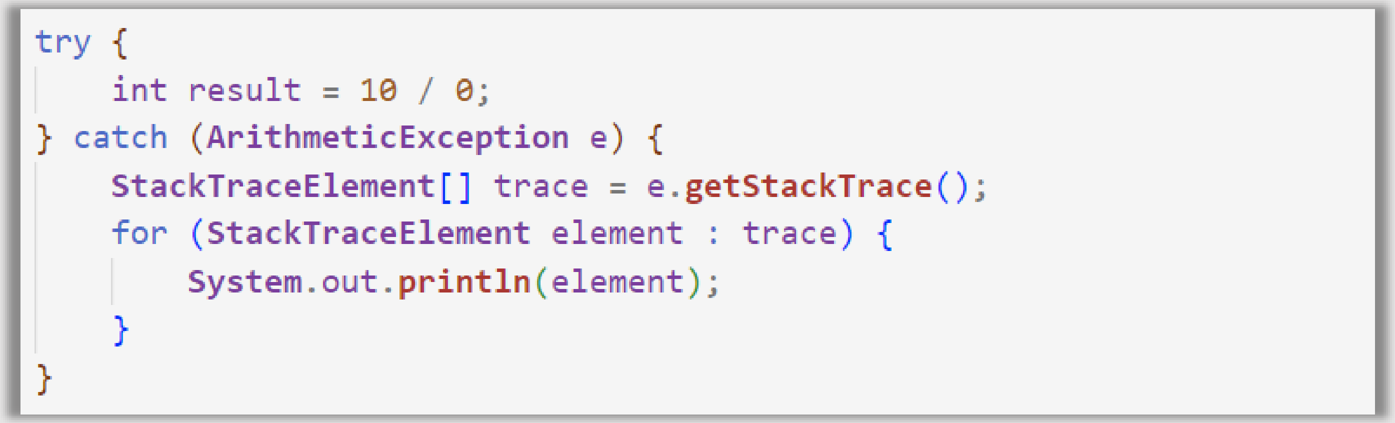
```
try {
    int result = 10 / 0;
} catch (ArithmeticException e) {
    e.printStackTrace();
}
```

Output:
java.lang.ArithmeticException: / by zero at Main.main(Main.java:4)

methods
keyword
handle
types
exceptions

customize

getStackTrace() :



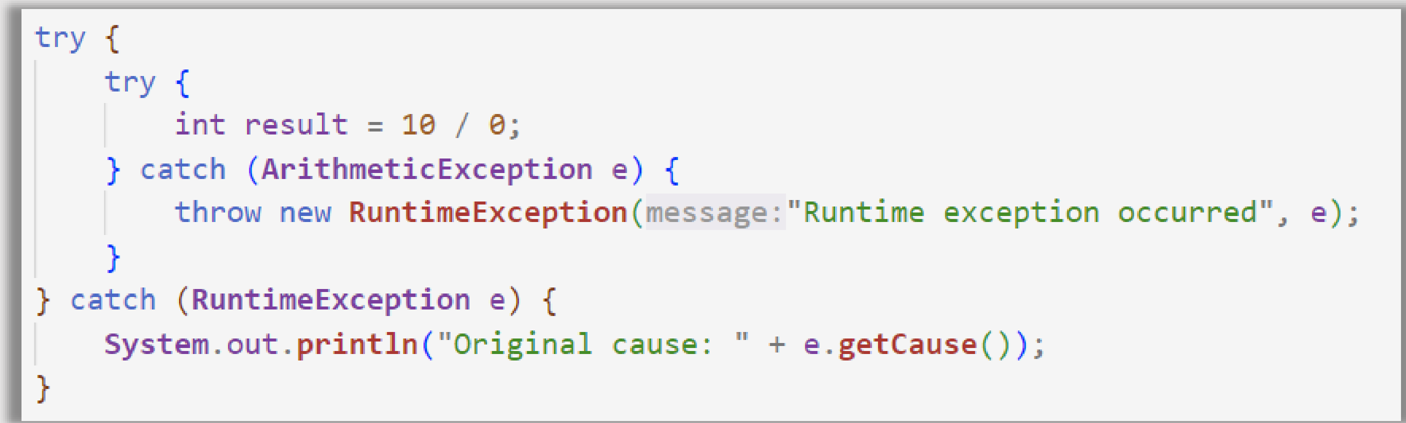
```
try {
    int result = 10 / 0;
} catch (ArithmeticException e) {
    StackTraceElement[] trace = e.getStackTrace();
    for (StackTraceElement element : trace) {
        System.out.println(element);
    }
}
```

Output:
Main.main(Main.java:4)

methods
keyword
handle
types
exceptions

customize

getCause() :



```
try {
    try {
        int result = 10 / 0;
    } catch (ArithmeticException e) {
        throw new RuntimeException(message:"Runtime exception occurred", e);
    }
} catch (RuntimeException e) {
    System.out.println("Original cause: " + e.getCause());
}
```

Output:

Original cause: java.lang.ArithmeticException: / by zero

methods

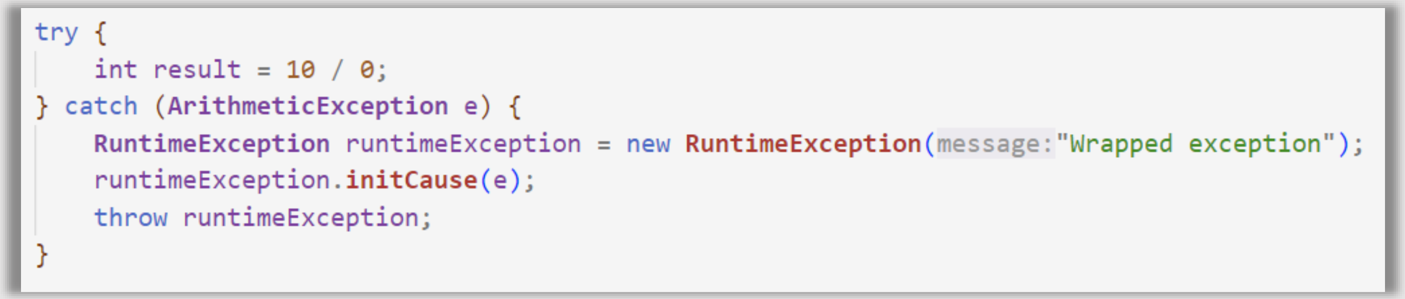
keyword

handle

types

exceptions

initCause() :



```
try {
    int result = 10 / 0;
} catch (ArithmeticException e) {
    RuntimeException runtimeException = new RuntimeException(message:"Wrapped exception");
    runtimeException.initCause(e);
    throw runtimeException;
}
```

Try this code and see what is happening.

Customize Exceptions in Java:

- You can create custom exceptions to handle specific situations not covered by standard Java exceptions.
- Here's how you can define, throw, and use a custom exception in Java:

1. Creating a Custom Exception
2. Checked vs. Unchecked Custom Exceptions
3. Adding Custom Fields or Methods

Creating a Custom Exception:

- The first step is to create a new class that represents your custom exception.

- This class should extend:
  - Exception for a checked exception.
  - RuntimeException for an unchecked exception.

- You can throw the custom exception in your code under specific conditions.
- The custom exception can be caught and handled just like any other exception in Java.

customize

methods

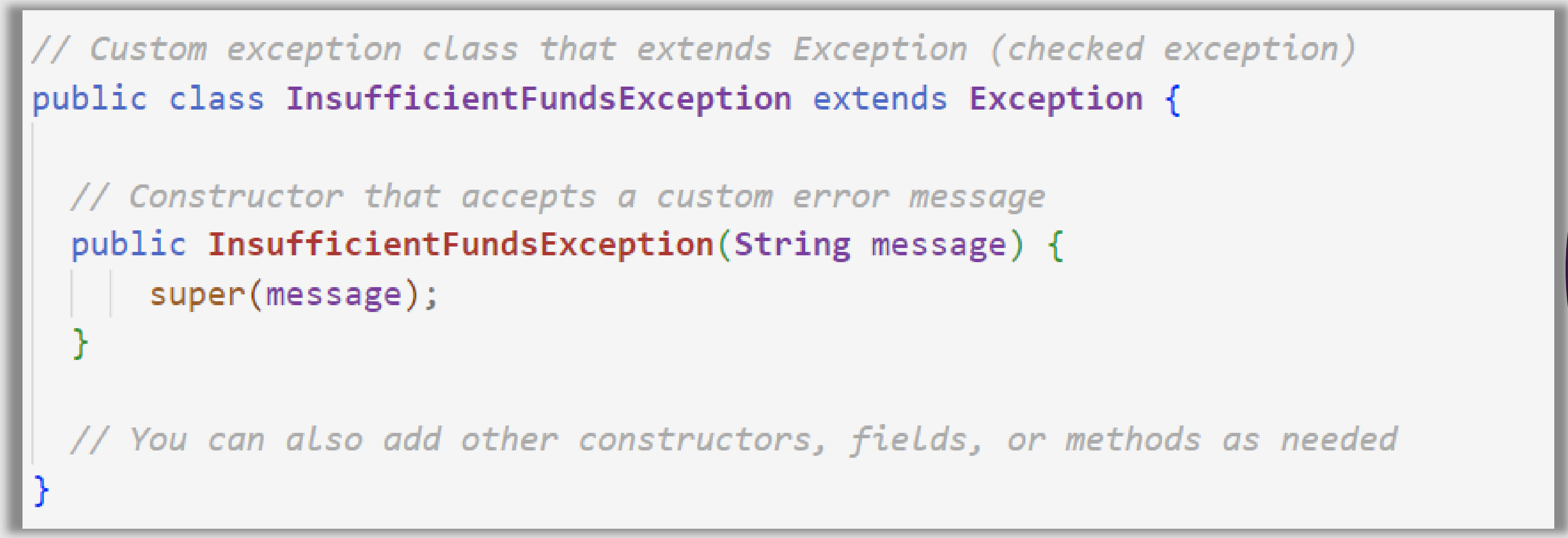
keywords

handle

types

exceptions

1. Define the Custom Exception Class



```
// Custom exception class that extends Exception (checked exception)
public class InsufficientFundsException extends Exception {

  // Constructor that accepts a custom error message
  public InsufficientFundsException(String message) {
      super(message);
  }

  // You can also add other constructors, fields, or methods as needed
}
```

customize

methods

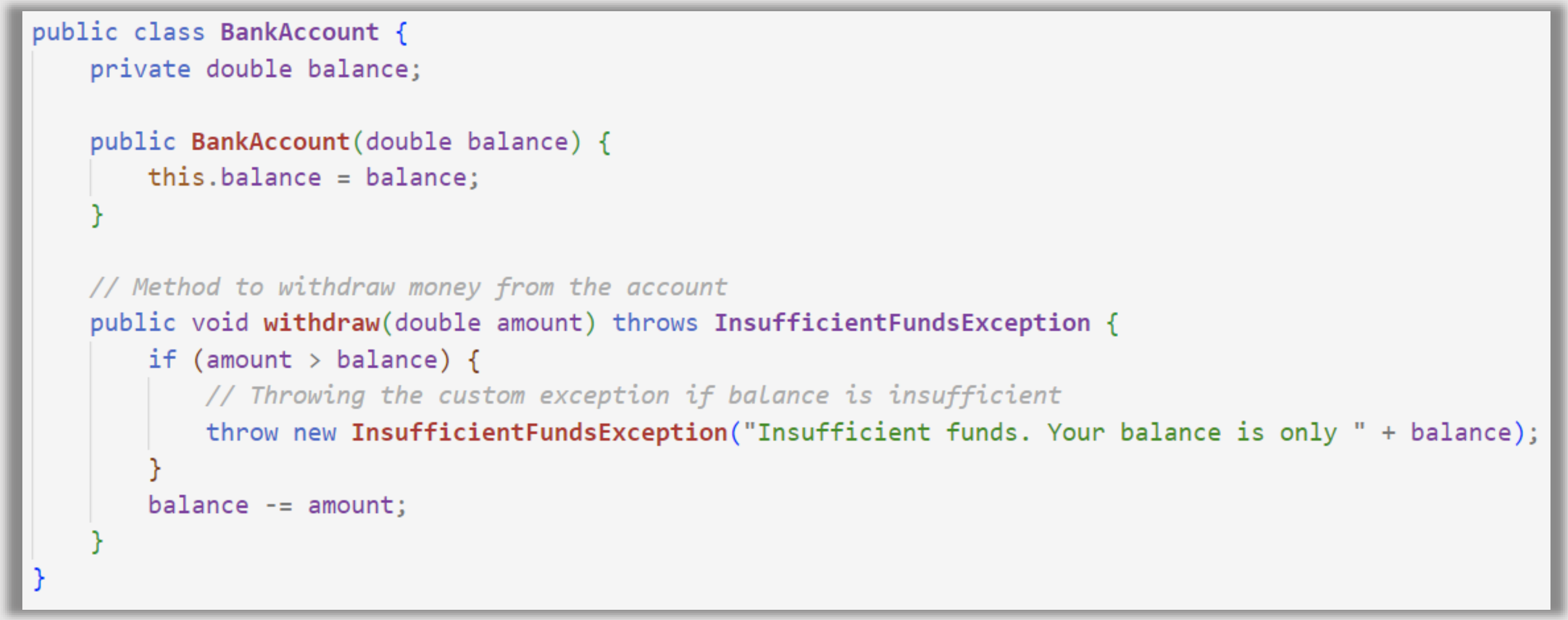
keyword

handle

types

exceptions

2. Throw the Custom Exception

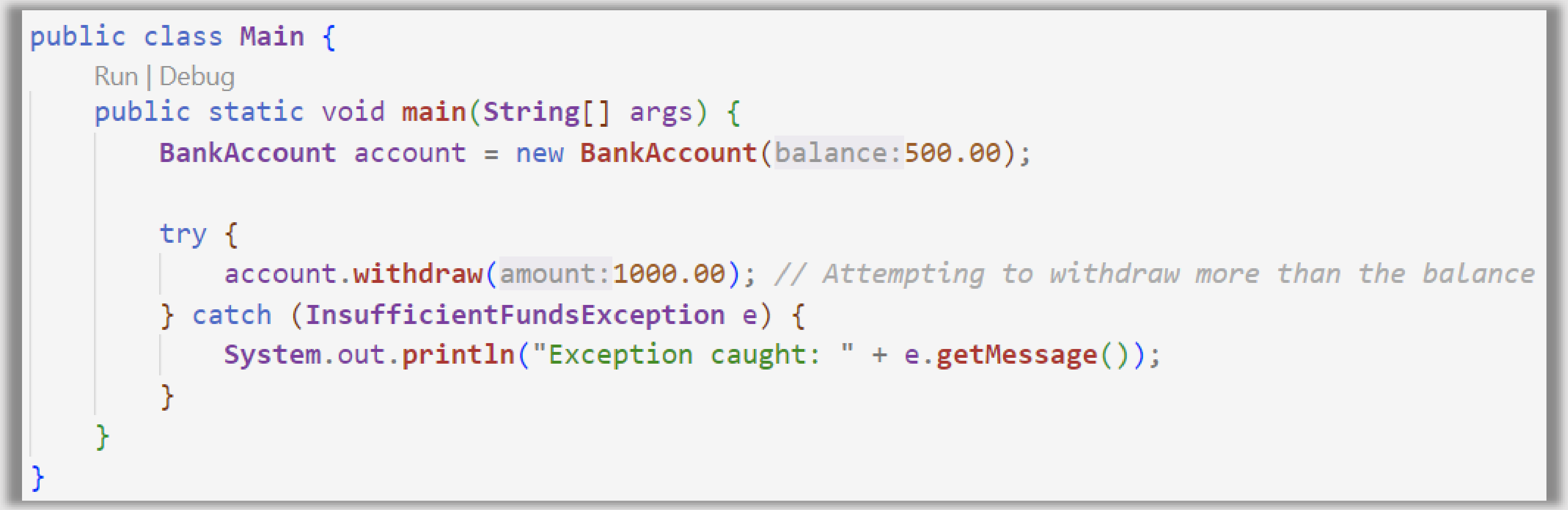


```java
public class BankAccount {
    private double balance;

    public BankAccount(double balance) {
        this.balance = balance;
    }

    // Method to withdraw money from the account
    public void withdraw(double amount) throws InsufficientFundsException {
        if (amount > balance) {
            // Throwing the custom exception if balance is insufficient
            throw new InsufficientFundsException("Insufficient funds. Your balance is only " + balance);
        }
        balance -= amount;
    }
}
```

3. Handle the Custom Exception



```
public class Main {
    Run | Debug
    public static void main(String[] args) {
        BankAccount account = new BankAccount(balance:500.00);

        try {
            account.withdraw(amount:1000.00); // Attempting to withdraw more than the balance
        } catch (InsufficientFundsException e) {
            System.out.println("Exception caught: " + e.getMessage());
        }
    }
}
```

customize

methods

keyword

handle

types

exceptions

Checked vs. Unchecked Custom Exceptions

Checked Custom Exceptions:

- Extends the Exception class.
- Becomes a checked exception.
- Any method that might throw this exception must:
  - Declare it using the throws keyword.
  - Ensure it is either caught or declared by any method that calls it.

customize

methods

keyword

handle

types

exceptions

Checked vs. Unchecked Custom Exceptions

Unchecked Custom Exceptions:

- Extends the RuntimeException class.
- Becomes an unchecked exception.
- Does not need to be declared in a method's throws clause.
- Can be thrown at any time during runtime without being explicitly caught.

customize
methods
keyword
handle
types
exceptions

Resources:

- https://www.geeksforgeeks.org/exceptions-in-java/
- https://www.w3schools.com/java/java_try_catch.asp
- https://www.javatpoint.com/exception-handling-in-java
- https://www.tutorialspoint.com/java/java_exceptions.htm

Any questions?

Thanks for your Attention.

FOLLOW