Unit 5: Exception Handling

Introduction to Exception

A Java exception is an object that describes an exceptional (that is, error) condition that has occurred in a piece of code.

When an exceptional condition arises, an object representing that exception is created and thrown in the method that caused the error.

That method may choose to handle the exception itself, or pass it on. Either way, at some point, the exception is caught and processed.

One of the powerful mechanism to handle the runtime errors so that the normal flow of the application can be maintained.

Terminologies

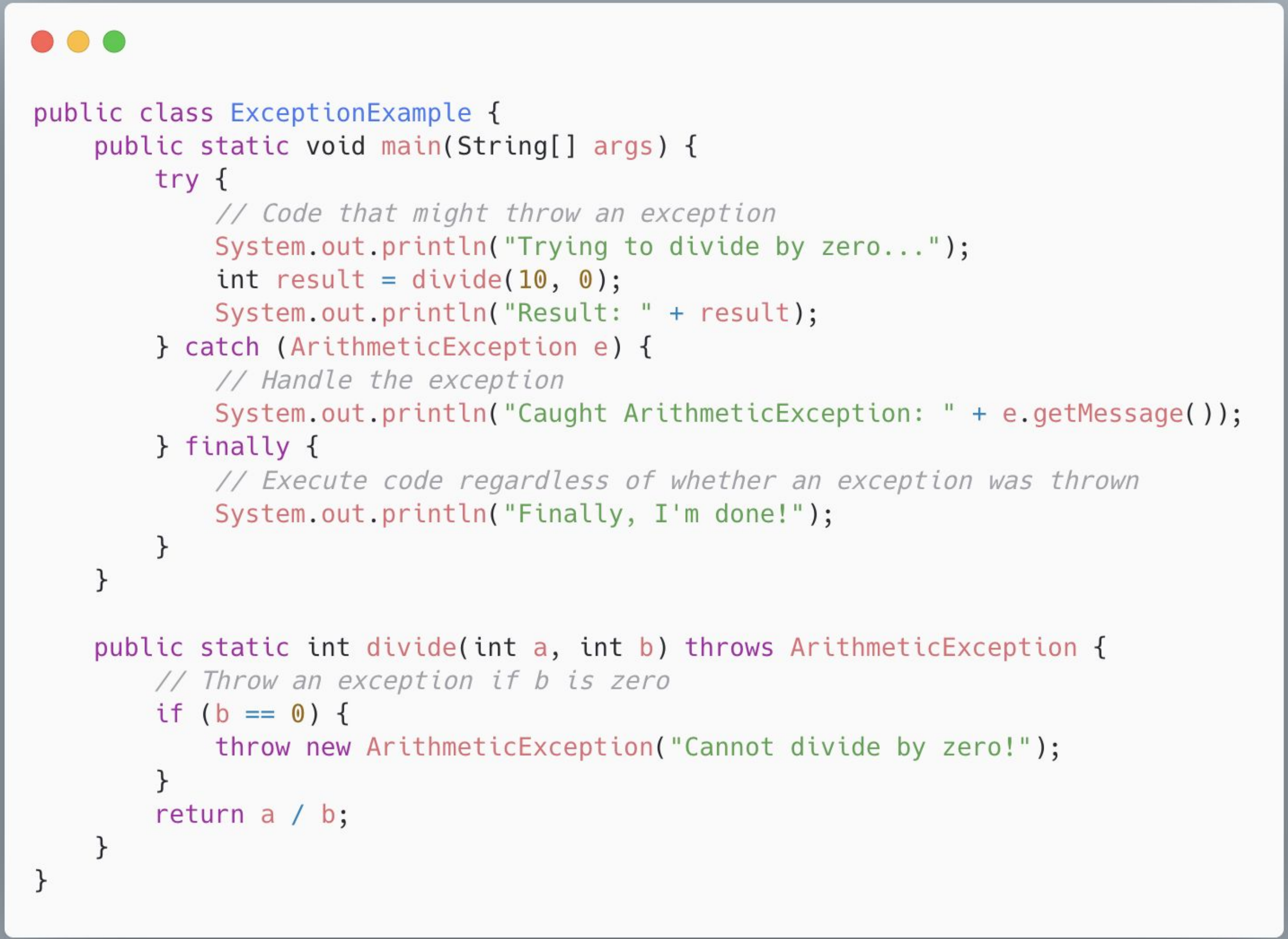
Try: The try block is used to enclose the code that might throw an exception. It's like a safety net to catch any unexpected errors.

Catch: The catch block is used to handle the exception thrown by the try block. It's like a rescue team that takes care of the error.

Throw: The throw keyword is used to explicitly throw an exception. It's like raising an alarm to signal that something has gone wrong.

Throws: The throws keyword is used to declare that a method might throw an exception. It's like a warning sign that says, "Be careful, I might throw an exception!"

Finally: The finally block is used to execute code regardless of whether an exception was thrown or not. It's like a cleanup crew that makes sure everything is tidy, no matter what happened.



```java
public class ExceptionExample {
    public static void main(String[] args) {
        try {
            // Code that might throw an exception
            System.out.println("Trying to divide by zero...");
            int result = divide(10, 0);
            System.out.println("Result: " + result);
        } catch (ArithmeticException e) {
            // Handle the exception
            System.out.println("Caught ArithmeticException: " + e.getMessage());
        } finally {
            // Execute code regardless of whether an exception was thrown
            System.out.println("Finally, I'm done!");
        }
    }

    public static int divide(int a, int b) throws ArithmeticException {
        // Throw an exception if b is zero
        if (b == 0) {
            throw new ArithmeticException("Cannot divide by zero!");
        }
        return a / b;
    }
}
```

Errors in Java Program

An error defines a reasonable issue that is stopping the execution of the program.

In Java, an error is a subclass of Throwable that tells that something serious problem is existing and a reasonable Java application should not try to catch that error.

Error and its subclass are referred to as Unchecked Exceptions.

Errors cannot be solved by any handling techniques, whereas we can solve exceptions using some logic implementations.

Errors are serious issues that are usually outside the control of the program. They indicate issues with the environment in which the application is running. Common examples in Java include OutOfMemoryError and StackOverflowError.

References: https://www.javatpoint.com/java-error

Exceptions

Exceptions are issues that occur during the execution of a program. They are usually conditions that a well-designed program can anticipate and handle, like FileNotFoundException or ArrayIndexOutOfBoundsException.

It is an object which is thrown at runtime.

These can and should be handled using try-catch blocks to allow the program to recover gracefully from these issues.

References: https://www.javatpoint.com/exception-handling-in-java

Types of Exceptions

1. Checked Exception
2. Unchecked Exception

References:
https://www.geeksforgeeks.org/checked-vs-unchecked-exceptions-in-java/

Checked Exceptions

1. Are checked by the compiler during compilation
2. Must be handled by the programmer or declared in the method signature
3. Examples: SQLException, IOException, ClassNotFoundException, etc.
4. Are subclasses of Exception

Unchecked Exception

1. Are not checked by the compiler during compilation
2. Do not need to be handled by the programmer or declared in the method signature
3. Examples: RuntimeException, NullPointerException, ArrayIndexOutOfBoundsException, etc.
4. Are subclasses of RuntimeException

Built in exceptions in Java

Inside the standard package java.lang, Java defines several exception classes.

These exceptions are subclasses of the standard type RuntimeException.

Exception	**Meaning**
ArithmeticException	Arithmetic error, such as divide-by-zero.
ArrayIndexOutOfBoundsException	Array index is out-of-bounds.
ArrayStoreException	Assignment to an array element of an incompatible type.
ClassCastException	Invalid cast.
EnumConstantNotPresentException	An attempt is made to use an undefined enumeration value.
IllegalArgumentException	Illegal argument used to invoke a method.
IllegalMonitorStateException	Illegal monitor operation, such as waiting on an unlocked thread.
IllegalStateException	Environment or application is in incorrect state.
IllegalThreadStateException	Requested operation not compatible with current thread state.
IndexOutOfBoundsException	Some type of index is out-of-bounds.
NegativeArraySizeException	Array created with a negative size.
NullPointerException	Invalid use of a null reference.
NumberFormatException	Invalid conversion of a string to a numeric format.
SecurityException	Attempt to violate security.
StringIndexOutOfBounds	Attempt to index outside the bounds of a string.
TypeNotPresentException	Type not found.
UnsupportedOperationException	An unsupported operation was encountered.

TABLE 10-1 Java's Unchecked **RuntimeException** Subclasses Defined in **java.lang**

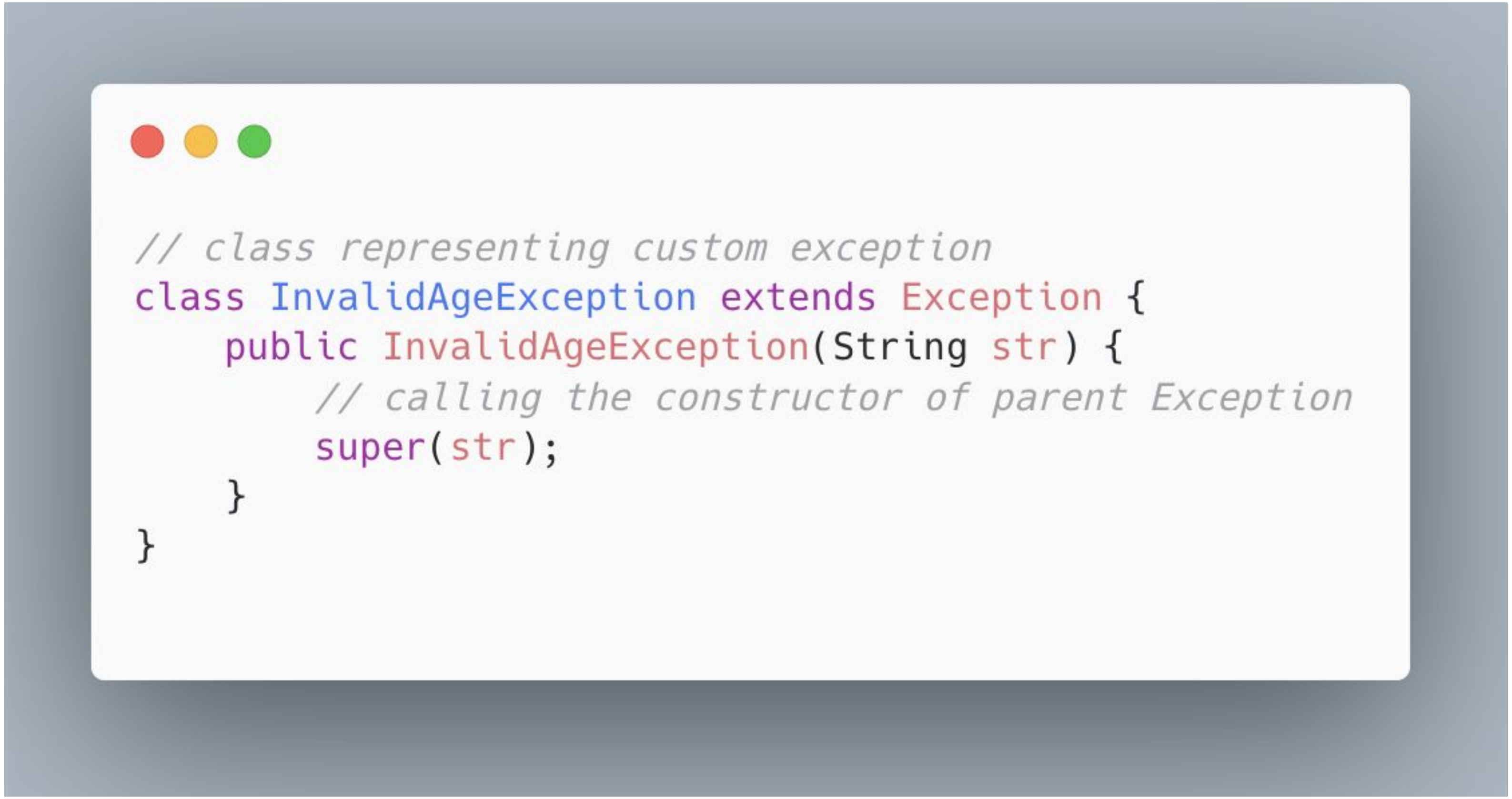
Exception	**Meaning**
ClassNotFoundException	Class not found.
CloneNotSupportedException	Attempt to clone an object that does not implement the **Cloneable** interface.
IllegalAccessException	Access to a class is denied.
InstantiationException	Attempt to create an object of an abstract class or interface.
InterruptedException	One thread has been interrupted by another thread.
NoSuchFieldException	A requested field does not exist.
NoSuchMethodException	A requested method does not exist.

TABLE 10-2 Java's Checked Exceptions Defined in **java.lang**

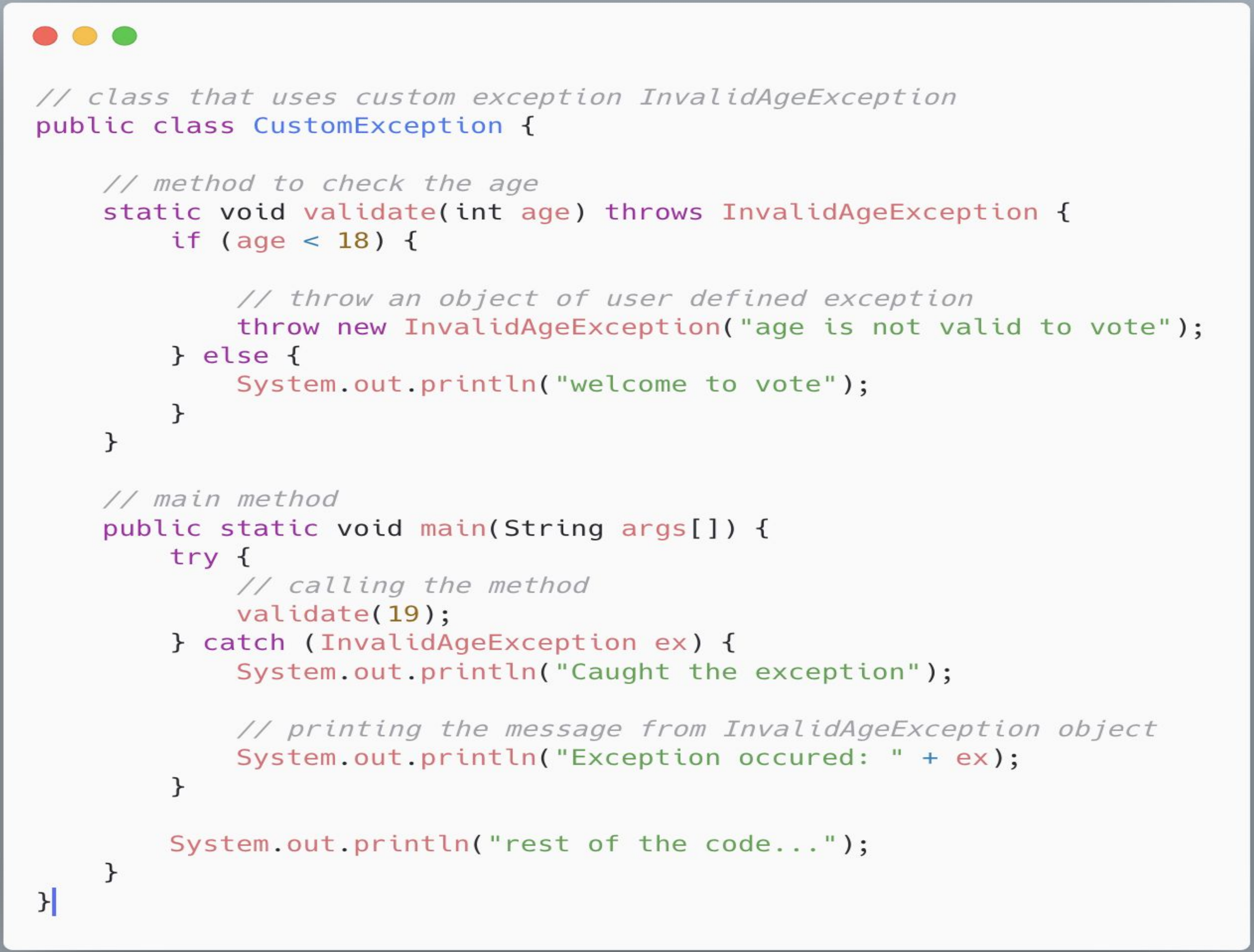
User defined exception in Java

In Java, we can create our own exceptions that are derived classes of the Exception class. Creating our own Exception is known as custom exception or user-defined exception. Basically, Java custom exceptions are used to customize the exception according to user need.

Example



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



```java
// class that uses custom exception InvalidAgeException
public class CustomException {

    // method to check the age
    static void validate(int age) throws InvalidAgeException {
        if (age < 18) {

            // throw an object of user defined exception
            throw new InvalidAgeException("age is not valid to vote");
        } else {
            System.out.println("welcome to vote");
        }
    }

    // main method
    public static void main(String args[]) {
        try {
            // calling the method
            validate(19);
        } catch (InvalidAgeException ex) {
            System.out.println("Caught the exception");

            // printing the message from InvalidAgeException object
            System.out.println("Exception occured: " + ex);
        }

        System.out.println("rest of the code...");
    }
}
```

Practical

1. Write a Java program that demonstrates the basic concept of an exception. Create an array of size 5 and try to access the 10th element. Catch the exception and print an appropriate message.
2. Write a Java program that intentionally causes a NullPointerException. Catch the exception and print a message indicating the error.
3. Write a Java program that demonstrates the difference between a checked and an unchecked exception. Use FileNotFoundException for the checked exception and ArithmeticException for the unchecked exception.
4. Write a Java program that demonstrates the use of try, catch, finally, and throw keywords. Create a method that throws an IllegalArgumentException if a negative number is passed to it.
5. Write a Java program that handles multiple built-in exceptions. Use NumberFormatException and ArrayIndexOutOfBoundsException in your program.
6. Write a Java program that creates a user-defined exception called InvalidAgeException. This exception should be thrown if the age provided is less than 18. Create a method that validates the age and throws this exception if the condition is met.