

# CSI1102: Introduction to Software Design

## Chapter 8: Exceptions and I/O Streams

## Learning objectives: Exceptions and I/O Streams

- Understand what the following are:
  - the try-catch statement
  - exception propagation
  - creating and throwing exceptions
  - types of I/O streams
- Study section 8.0 for the final examination

2

## What are Exceptions?

- An *exception* is an object that describes an unusual or erroneous situation; e.g. Misuse
- Exceptions are *thrown* by a program, and may be *caught* and *handled* by another part of the program
- A program can be separated into
  - a normal execution flow and
  - an *exception execution flow*
- An *error* is also represented as an object in Java, but usually represents a unrecoverable situation and should not be caught



3

## Java Exception Handling

- Java has a predefined set of exceptions and errors that can occur during execution
- A program can deal with an exception in one of three ways:
  - ignore it
  - handle it where it occurs
  - handle it in another place in the program
- The manner in which an exception is processed is an **important design consideration**



4

## Java Exception Handling

- The message includes a *call stack trace* that indicates the line on which the exception occurred
- The call stack trace also shows the method call trail that lead to the attempted execution of the offending line
  - The `getMessage` method returns a string explaining why the exception was thrown
  - The `printStackTrace` method prints the call stack trace
- See [Zero.java](#) (p.449)

5

## Zero.java

```
public class Zero
{
    // Deliberately divides by zero to produce an exception.
    public static void main (String[] args)
    {
        int numerator = 10;
        int denominator = 0;

        System.out.println (numerator / denominator);

        System.out.println ("This text will not be printed.");
    }
}
```

```
java.lang.ArithmeticException: / by zero
    at Zero.main(Zero.java:17)
Exception in thread "main" Exit code: 1
There were errors
```

6

## Handling Exceptions: The `try` Statement



- To process an exception when it occurs, the line that throws the exception is executed within a *try block*
- A try block is followed by one or more *catch* clauses, which contain code to process an exception
- Each catch clause has an associated exception type and is called an *exception handler*
- When an exception occurs, processing continues at the first catch clause that matches the exception type
- See [ProductCodes.java](#) (page 451)

7

## ProductCodes.java

```
import cs1.Keyboard;

public class ProductCodes
{
    // Counts the number of product codes that are entered with a
    // zone of R and district greater than 2000.
    //-----
    public static void main (String[] args)
    {
        String code;
        char zone;
        int district, valid = 0, banned = 0;

        System.out.print ("Enter product code (XXX to quit): ");
        code = Keyboard.readString();
    }
}
```

Continued...

8

## ProductCodes.java

```
while (!code.equals ("XXX"))
{
    try
    {
        zone = code.charAt(9);
        district = Integer.parseInt(code.substring(3, 7));
        valid++;
        if (zone == 'R' && district > 2000)
            banned++;
    }
    catch (StringIndexOutOfBoundsException e)
    {
        System.out.println ("Improper code length: " + code);
    }
    catch (NumberFormatException e)
    {
        System.out.println ("District is not numeric: " + code);
    }
    System.out.print ("Enter product code (XXX to quit): ");
    code = Keyboard.readString();
}

System.out.println ("# of valid codes entered: " + valid);
System.out.println ("# of banned codes entered: " + banned); } }
```

9

## The `finally` Clause

- A try statement can have an optional clause following the catch clauses, designated by the reserved word *finally*
- The statements in the finally clause **are always executed**
  - If no exception is generated, the statements in the finally clause are executed after the statements in the try block are completed
  - If an exception is generated, the statements in the finally clause are executed after the statements in the appropriate catch clause are completed

10

## Exception Propagation

- An exception can be handled at a higher level if it is not appropriate to handle it where it occurs
- Exceptions *propagate* up through the method calling hierarchy until they are caught and handled or until they reach the level of the `main` method
- A try block that contains a call to a method in which an exception is thrown can be used to catch that exception
- See [Propagation.java](#) (page 455)
- See [ExceptionScope.java](#) (page 456)

11

## Propagation.java

```
public class Propagation
{
    // Invokes the level1 method to begin the exception demonstration.

    static public void main (String[] args)
    {
        ExceptionScope demo = new ExceptionScope();

        System.out.println("Program beginning.");
        demo.level1();
        System.out.println("Program ending.");
    }
}
```

12

## Propagation: The output

Program beginning.  
Level 1 beginning.  
Level 2 beginning.  
Level 3 beginning.

The exception message is: / by zero

The call stack trace:  
java.lang.ArithmeticException: / by zero  
at ExceptionScope.level3(ExceptionScope.java:54)  
at ExceptionScope.level2(ExceptionScope.java:41)  
at ExceptionScope.level1(ExceptionScope.java:18)  
at Propagation.main(Propagation.java:17)

Level 1 ending.  
Program ending.

13

## ExceptionScope.java

```
public class ExceptionScope
{
    //-----
    // Catches and handles the exception that is thrown in level3.
    //-----
    public void level1()
    {
        System.out.println("Level 1 beginning.");

        try
        {
            level2();
        }
        catch (ArithmeticException problem)
        {
            System.out.println ();
            System.out.println ("The exception message is: " +
                problem.getMessage());
            System.out.println ();
            System.out.println ("The call stack trace:");
            problem.printStackTrace();
            System.out.println ();
        }

        System.out.println("Level 1 ending."); }
}
```

Continued<sup>4</sup>

## ExceptionScope.java

```
public void level2()
{
    System.out.println("Level 2 beginning.");
    level3 ();
    System.out.println("Level 2 ending.");
}

//-----
// Performs a calculation to produce an exception. It is not
// caught and handled at this level.
//-----
public void level3 ()
{
    int numerator = 10, denominator = 0;

    System.out.println("Level 3 beginning.");
    int result = numerator / denominator;
    System.out.println("Level 3 ending.");
}
}
```

15

## The Exception Class Hierarchy

- Figure 8.1 p.458
- Object
- Throwable
- Error
- AWTError
- VirtualMachineError
- Exception
- RuntimeException
- ArithmeticException
- IndexOutOfBoundsException
- NullPointerException
- IOException
- NoSuchElementException
- ...

16

## Exception handling: The `throw` Statement

- A programmer can define an exception by extending the `Exception` class or one of its descendants
- Exceptions are thrown using the `throw` statement
- Usually a `throw` statement is nested inside an `if` statement that evaluates the condition to see if the exception should be thrown
- See [CreatingExceptions.java](#) (page 459)
- See [OutOfRangeException.java](#) (page 460)

17

## CreatingExceptions.java

```
import cs1.Keyboard;

public class CreatingExceptions
{
    //-----
    // Creates an exception object and possibly throws it.
    //-----
    public static void main (String[] args) throws OutOfRangeException
    {
        final int MIN = 25, MAX = 40;

        OutOfRangeException problem =
            new OutOfRangeException ("Input value is out of range.");

        System.out.print ("Enter an integer value between " + MIN +
            " and " + MAX + ", inclusive: ");
        int value = Keyboard.readInt();

        // Determines if the exception should be thrown
        if (value < MIN || value > MAX)
            throw problem;

        System.out.println ("End of main method."); // may never reach
    }
}
```

18

## Throwing an exception

Enter an integer value between 25 and 40, inclusive: 3  
OutOfRangeException: Input value is out of range.  
At CreatingExceptions.main  
(CreatingExceptions.java:18)

Enter an integer value between 25 and 40, inclusive: 27  
End of main method

19

## OutOfRangeException.java

```
public class OutOutOfRangeException extends Exception
{
    //-----
    // Sets up the exception object with a particular message.
    //-----
    OutOutOfRangeException (String message)
    {
        super (message);
    }
}
```

- Whether to use an exception, a conditional, or a loop is an important design decision

20

## Checked Exceptions

- An exception is either *checked* or *unchecked*
- A *checked exception* either must be caught by a method, or must be listed in the throws clause of any method that may throw or propagate it
  - a throws clause is appended to the method header
- The compiler will complain if a checked exception is not handled appropriately

21

## Unchecked Exceptions

- An unchecked exception does not require explicit handling, though it could be processed that way
- The only unchecked exceptions in Java are objects of type `RuntimeException` or any of its descendants
- Errors are similar to `RuntimeException` and its descendants
  - Errors should not be caught
  - Errors do not require a throws clause

22

## Standard I/O


- There are three standard I/O streams:
  - standard input* – defined by `System.in`
  - standard output* – defined by `System.out`
  - standard error* – defined by `System.err`
- `System.in` typically represents keyboard input
- `System.out` and `System.err` typically represent a particular window on the monitor screen
- We use `System.out` when we execute `println` statements
- See p.461+ of text book

23

## The IOException Class

- Operations performed by the I/O classes may throw an `IOException`
  - A file intended for reading or writing might not exist
  - Even if the file exists, a program may not be able to find it
  - The file might not contain the kind of data we expect
- An `IOException` is a checked exception

24



## Chapter 8: Summary

---

- Study only section 8.0 for the examination
- Understand what the following are:
  - the try-catch statement
  - exception propagation
  - creating and throwing exceptions
  - (I/O streams)