CSC220 (CSI)
Computational Problem Solving

Exceptions

The College of New Jersey

Please turn off your cell phone!

Exceptions

- An exception is an object that describes an unusual or erroneous situation
- 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

Exception Handling

- The Java API has a predefined set of exceptions 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 an another place in the program
- The manner in which an exception is processed is an important design consideration

Exception Handling

- If an exception is ignored (not caught) by the program, the program will terminate and produce an appropriate message
- The message includes a call stack trace that:
  - indicates the line on which the exception occurred
  - shows the method call trail that lead to the attempted execution of the offending line
- See `Zero.java`

```
//********************************************************************
//  Zero.java       Author: Lewis/Loftus
//  Demonstrates an uncaught exception.
//********************************************************************

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.");
   }
}
```

Output (when program terminates)

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at Zero.main(Zero.java:17)
```

The try Statement

- To handle an exception in a program, use a try-catch statement
- A try block is followed by one or more catch clauses
- Each catch clause has an associated exception type and is called an exception handler
- When an exception occurs within the try block, processing immediately jumps to the first catch clause that matches the exception type
- See `ProductCodes.java`

```
//********************************************************************
//  ProductCodes.java       Author: Lewis/Loftus
//
//  Demonstrates the use of a try-catch block.
//********************************************************************

import java.util.Scanner;

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

      Scanner scan = new Scanner(System.in);

      System.out.print("Enter product code (XXX to quit): ");
      code = scan.nextLine();
```

continue

continue

```
        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 exception)
            {
                System.out.println("Improper code length: " + code);
            }
            catch (NumberFormatException exception)
            {
                System.out.println("District is not numeric: " + code);
            }

            System.out.print("Enter product code (XXX to quit): ");
            code = scan.nextLine();
        }

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

continue

```
        wh
        {
```

Sample Run

```
Enter product code (XXX to quit): TRV2475A5R-14
Enter product code (XXX to quit): TRD1704A7R-12
Enter product code (XXX to quit): TRL2k74A5R-11
District is not numeric: TRL2k74A5R-11
Enter product code (XXX to quit): TRQ2949A6M-04
Enter product code (XXX to quit): TRV2105A2
Improper code length: TRV2105A2
Enter product code (XXX to quit): TRQ2778A7R-19
Enter product code (XXX to quit): XXX
# of valid codes entered: 4
# of banned codes entered: 2
```

```
            }
            catch (NumberFormatException exception)
            {
               System.out.println("District is not numeric: " + code);
            }

            System.out.print ("Enter product code (XXX to quit): ");
            code = scan.nextLine();
        }

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

The finally Clause

- A try statement can have an optional `finally` clause, which is always executed
- If no exception is generated, the statements in the finally clause are executed after the statements in the try block finish
- If an exception is generated, the statements in the finally clause are executed after the statements in the appropriate catch clause finish

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
- See `Propagation.java`
- See `ExceptionScope.java`

```
//********************************************************************
//  Propagation.java       Author: Lewis/Loftus
//
//  Demonstrates exception propagation.
//********************************************************************

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.");
   }
}
```

```
//********************************************************************
//  ExceptionScope.java       Author: Lewis/Loftus
//
//  Demonstrates exception propagation.
//********************************************************************

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();
```

continue

continue

```
            System.out.println("The call stack trace:");
            problem.printStackTrace();
            System.out.println();
        }
        System.out.println("Level 1 ending.");
    }

    //  Serves as an intermediate level.  The exception propagates
    //  through this method back to level1.
    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.");
    }
}
```

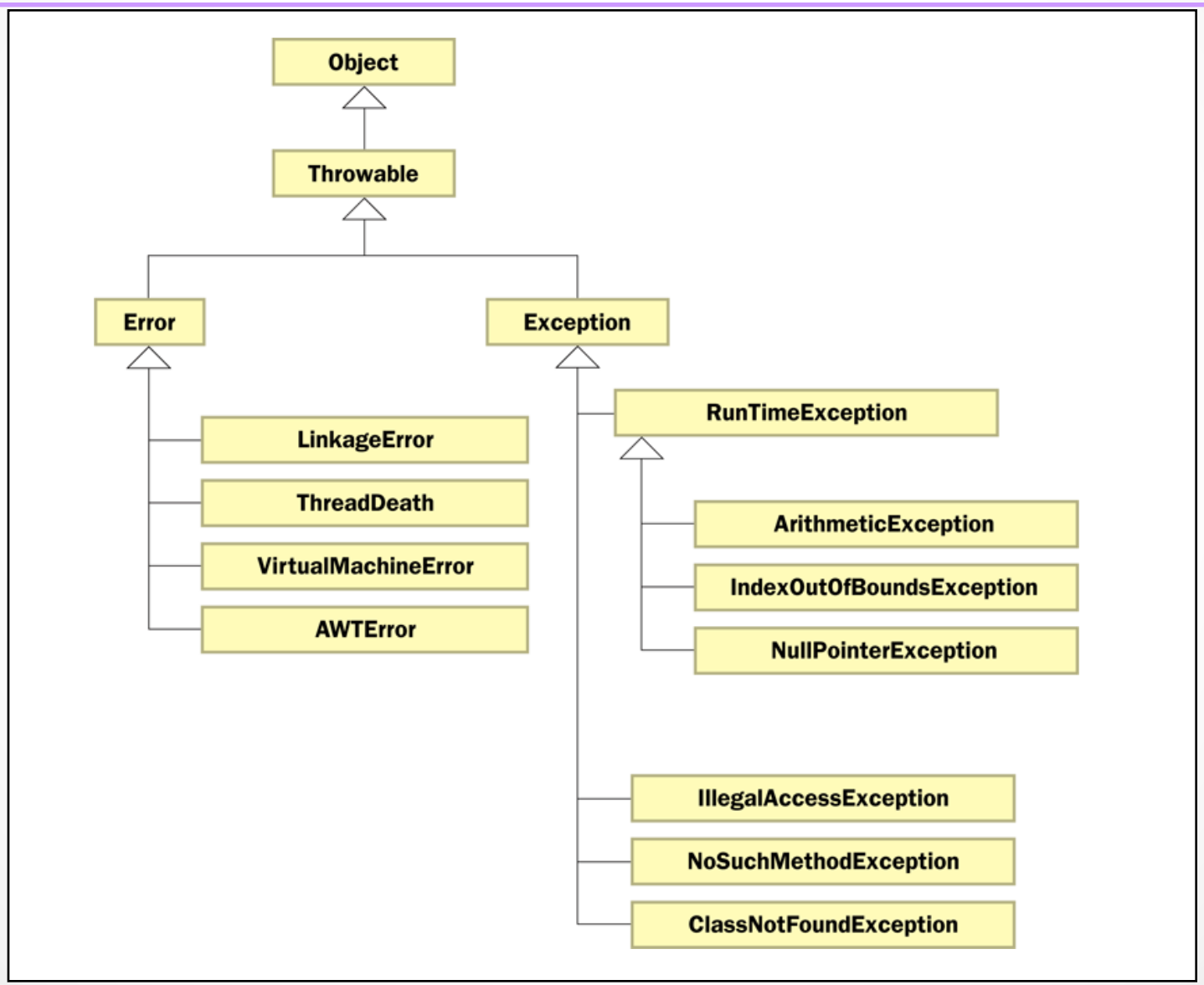
Output

```
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.
```

The Exception Class Hierarchy

- Exception classes in the Java API are related by inheritance, forming an exception class hierarchy
- All error and exception classes are descendants of the `Throwable` class
- A programmer can define an exception by extending the `Exception` class or one of its descendants
- The parent class used depends on how the new exception will be used

The Exception Class Hierarchy



Checked Exceptions

- An exception is either checked or unchecked
- A checked exception must either be caught 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 issue an error if a checked exception is not caught or listed in a throws clause

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 in that:
  - Errors should not be caught
  - Errors do not require a throws clause

Quick Check

Which of these exceptions are checked and which are unchecked?

`NullPointerException`	Unchecked
`IndexOutOfBoundsException`	Unchecked
`ClassNotFoundException`	Checked
`NoSuchMethodException`	Checked
`ArithmeticException`	Unchecked

The throw Statement

- Exceptions are thrown using the *throw* statement
- Usually a throw statement is executed inside an if statement that evaluates a condition to see if the exception should be thrown
- See `CreatingExceptions.java`
- See `OutOfRangeException.java`

```
//********************************************************************
//  CreatingExceptions.java       Author: Lewis/Loftus
//  Demonstrates the ability to define an exception via inheritance.
//********************************************************************
import java.util.Scanner;
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;
      Scanner scan = new Scanner(System.in);

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

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

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

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

```
//********************************************************************
//  OutOfRangeException.java       Author: Lewis/Loftus
//
//  Represents an exceptional condition in which a value is out of
//  some particular range.
//********************************************************************

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

Sample Run

```
Enter an integer value between 25 and 40, inclusive:  69
Exception in thread "main" OutOfRangeException:
    Input value is out of range.
    at CreatingExceptions.main(CreatingExceptions.java:20)
```

Quick Check

What is the matter with this code?

```
System.out.println("Before throw");
throw new OutOfRangeException("Too High");
System.out.println("After throw");
```

The throw is not conditional and therefore always occurs. The second `println` statement can never be reached.

I/O Exceptions

- Let's examine issues related to exceptions and I/O
- A *stream* is a sequence of bytes that flow from a source to a destination
- In a program, we read information from an input stream and write information to an output stream
- A program can manage multiple streams simultaneously

Standard I/O

- There are three standard I/O streams:
  - *standard output* – defined by `System.out`
  - *standard input* – defined by `System.in`
  - *standard error* – defined by `System.err`
- We use `System.out` when we execute `println` statements
- `System.out` and `System.err` typically represent the console window
- `System.in` typically represents keyboard input, which we've used many times with `Scanner`

The IOException Class

- Operations performed by some I/O classes may throw an `IOException`
  - A file 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

Writing Text Files

- In Chapter 5 we explored the use of the `Scanner` class to read input from a text file
- Let's now explore writing data to a text file
- The `PrintWriter` class represents a text output file
- Output streams should be closed explicitly
- See `TestData.java`

```java
//********************************************************************
//  TestData.java       Author: Lewis/Loftus
//  Demonstrates I/O exceptions and the use of a character file
//  output stream.
//********************************************************************
import java.util.Random;
import java.io.*;
public class TestData{
   //  Creates a file of test data that consists of ten lines each
   //  containing ten integer values in the range 10 to 99.
   public static void main(String[] args) throws IOException {
      final int MAX = 10;
      int value;
      String fileName = "test.txt";
      PrintWriter outFile = new PrintWriter(fileName);
      Random rand = new Random();

      for (int line=1; line <= MAX; line++) {
         for (int num=1; num <= MAX; num++) {
            value = rand.nextInt(90) + 10;
            outFile.print(value + "   ");
         }
         outFile.println();
      }
      outFile.close();
      System.out.println("Output file has been created: " + fileName);
   }
}
```