Murdoch
UNIVERSITY

Topic 8
Exception Handling

ICT167 Principles of
Computer Science

Murdoch UNIVERSITY

Objectives

- Explain what is an ***exception***
- Explain how Java supports ***exception handling***
- Describe a simple use of **try**, **throw** and **catch**
- Know the syntax for throwing exceptions
- Be able to recognize common built-in exception classes
- Explain the process of catching exceptions when multiple catch blocks are present

Objectives

- Be able to create your own exception classes
- Know how and when to declare exceptions, and be able to trace the flow of control after an exception has been thrown in such a program

- **Reading**

  Savitch: Chapter 9

Java Exceptions

- When a Java program violates the semantic constraints of the Java language, the JVM (Java Virtual Machine) signals this error to the program as an ***exception***
- This is also referred to as an unusual situation which the programmer may not want to be handled by the usual code

Java Exceptions

- Eg:
  - dividing by zero
  - index position in String is out of bounds
  - a name is too long for the spacing allocated for it on the screen
- Note that exceptions may arise in built-in Java, in library classes, or in your own code

Java Exceptions

- Exceptions are **`thrown`** by a program, and may be **`caught`** and **`handled`** by another part of the (same) program
- A program can therefore be separated into **`a normal execution flow`** and an **`exception execution flow`**

Exception Terminology

- **`Throwing an exception`**: either Java itself or your code signals when something unusual happens – involves creating the exception object
- **`Handling an exception`**: responding to an exception by executing a part of the program specifically written for the exception
  - Also called **`catching an exception`**

Exception Terminology

- The normal case is handled in a **`try`** block – the code where something could *possibly* go wrong
- The exceptional case is handled in a **`catch`** block
- The **`catch`** block takes a parameter of type **`Exception`**
  - It is called the **`catch-block parameter`**
  - **`e`** is a commonly used name for it

Exception Terminology

- If an exception is **`thrown`** execution in the **`try`** block ends and control passes to the **`catch`** block(s) after the **`try`** block

Exception Handling

- Exception handling is the code to deal with the special cases (without the program crashing)
- Usually, this is an afterthought by programmers but in many programming languages it may involve making big changes to the usual code
- A program can deal with an exception in one of three ways:
  - Ignore it
  - Handle it where it occurs
  - Handle it at another place within the program

Exception Handling

- In many programming languages you use the normal features of the language
  - Eg: add *if-then* statements etc., inside the usual code to deal with the exceptions
- You can do that in Java too
- But in Java there are also special constructs designed so that you can add on the exception handling separately from the usual code
- Look at these three versions of an example from Savitch …

Example: DodgyStodgy

```
// DodgyStodgy.java: Code capable of crashing
import java.util.Scanner;
public class DodgyStodgy {
   public static void main(String[] args) {
      int donutCount, milkCount;
      double donutsPerGlass;
      Scanner keyboard = new Scanner(System.in);
      System.out.println("Number of donuts?:");
      donutCount = keyboard.nextInt();
      System.out.println("Glasses of milk?:");
      milkCount = keyboard.nextInt();
```

Example: DodgyStodgy

```
        donutsPerGlass =
                        donutCount/(double)milkCount;
        System.out.println(donutCount+" donuts.");
        System.out.println(milkCount+" glasses.");
        System.out.println("You have " +
                           donutsPerGlass + " donuts
                           for each glass of milk.");
    } // end main
} // end class DodgyStodgy
```

Example: DodgyStodgy

```
/* Sample Output 1
Enter number of donuts:
12
Enter glasses of milk:
4
12 donuts.
4 glasses.

You have 3.0 donuts for each glass.
*/
```

Example: DodgyStodgy

```
/* Sample Output 2
Enter number of donuts:
6
Enter glasses of milk:
0
6 donuts.
0 glasses.

You have Infinity donuts for each glass.
*/
```

Example: GotMilk

```
/** GotMilk.java: A better version of DodgyStodgy
    with a programmer's own exception handling
    routine added in the middle. */
import java.util.Scanner;
public class GotMilk {
  public static void main(String[] args) {
     int donutCount, milkCount;
     double donutsPerGlass;
     Scanner keyboard = new Scanner(System.in);
     System.out.println("Number of donuts?:");
     donutCount = keyboard.nextInt();
```

Example: GotMilk

```
System.out.println("Glasses of milk?:");
milkCount = keyboard.nextInt();
if (milkCount < 1) {
   System.out.println("No Milk!");
   System.out.println("Go buy some milk.");
} else {
   donutsPerGlass =
                donutCount/(double)milkCount;
   System.out.println(donutCount + "
                                   donuts.");
```

Example: GotMilk

```
            System.out.println(milkCount + "
                                              glasses.");
            System.out.println("You have " +
                              donutsPerGlass +" donuts
                              for each glass of milk.");
        }
        System.out.println("End of program.");
    } // end main
} // end class GotMilk
```

Example: GotMilk

```
/* Sample Output
Enter number of donuts:
6
Enter glasses of milk:
0
No Milk!
Go buy some milk.
End of program.
*/
```

Example: Exceptions

```
/** ExceptionDemo.java: An even better version of
    DodgyStodgy - making use of Java's built-in
    exception handling. */
import java.util.Scanner;
public class ExceptionDemo {
  public static void main(String[] args) {
     int donutCount, milkCount;
     double donutsPerGlass;
     Scanner keyboard = new Scanner(System.in);
     try{      //start of try block
        System.out.println("Number donuts?:");
        donutCount = keyboard.nextInt();
```

Example: Exceptions

```
System.out.println("Glasses of milk?:");
milkCount = keyboard.nextInt();
if (milkCount < 1)
   throw new Exception("Exception: No
                                    Milk!");
donutsPerGlass =
            donutCount/(double)milkCount;
System.out.println(donutCount + "
                                  donuts.");
System.out.println(milkCount + "
                                  glasses.");
```

Murdoch UNIVERSITY

Example: Exceptions

```
            System.out.println("You have " +
                                donutsPerGlass + " donuts
                                for each glass of milk.");
    } //end of try block
    catch(Exception e){ //start catch block
                System.out.println(e.getMessage());
                System.out.println(”Go buy some milk!");
    } //end of catch block
    System.out.println("End of program.");
    } // end main
} // end class ExceptionDemo
```

Example: Exceptions

```
/* Sample Output
Enter number of donuts:
6
Enter glasses of milk:
0

Exception: No Milk!
Go buy some milk.
End of program.

*/
```

The `try` Statement

- To process an exception when it occurs, the line that **`throws`** the exception is executed within a **`try`** block
- Your code may **`throw`** different types of exceptions which you may want to treat differently

The `try` Statement

- A **`try`** block is followed by one or more **`catch`** blocks (clauses), which contain code to process an exception
- Each **`catch`** block has an associated exception type
- When an exception occurs, processing continues at the first catch block that matches the exception type

Throwing an Exception

- Put the normal code (for unexceptional cases) inside a **`try`** block
- Inside the **`try`** block, an exception can be **`thrown`** either
  - By some Java construct, or
  - Within some method (yours, in a library class or someone else's) which is invoked from within the **`try`** block, or
  - By the code explicitly throwing an exception by:

```
throw new Exception("optional
message");
```

Throwing an Exception

- If an exception is **`thrown`**, the following actions are taken:
  - A new exception object is created which just contains the optional message (if there is one)
  - The interpreter (JVM) leaves the try block immediately and goes in search of a catch block immediately after the try block
  - If a matching catch block exists (see later) then the code inside the catch block is executed
  - After that, execution continues after the end of the catch block(s)

Throwing an Exception

- If there is no exception **`thrown`** in the **`try`** block then the **`catch`** block is just skipped

Built-in/Library Exception Classes

- The predefined class **Exception** (to which all exception objects belong) is the top of a complicated hierarchy of classes which contain built-in exceptions, library class exceptions and many programmer-defined exceptions
- Eg:
  - **IOException**
  - **ClassNotFoundException**
  - **FileNotFoundException**
  - **IndexOutOfBoundsException**
  - **NumberFormatException**

Built-in/Library Exception Classes

- In documentation you will see such exceptions as being possibly **`thrown`** by methods
- Eg:

```
public char charAt(int index)

    Parameters: ...

    Returns: ...

     Throws:

         IndexOutOfBoundsException - if the
         index argument is negative or not
         less than the length of this string
```

Catching Exceptions

- Your program may throw different types of exceptions which you may want to treat differently
- Then use multiple catch blocks ...

```
try {
  // normal code including code which may
  // throw exceptions
}
```

Catching Exceptions

```
catch(IOException e) {
  // code to deal with an IOException probably
  // including the following:
  System.out.println(e.getMessage);
}
catch(IndexOutOfBoundsException e) {
  // code to deal with this exception as above
}
```

Catching Exceptions

```
catch(Exception e) {
  // deal with all other exceptions
  System.out.println(e.getMessage();
  System.out.println("Program " +
       "aborted");
  System.exit(0);
}
// rest of the method code
// ...
```

Catching Exceptions

- If an exception is thrown then it will be caught by the first matching catch block (if there is one following on from the try block)
- Note that any exception will match the last catch block above
- The string returned by the method call `e.getMessage()` ought to provide enough information to identify the source of exception

Example: Method to read an `int`


```
// assumes java.util.Scanner is imported
public static int readLineInt( ) {
   Scanner keyboard = new Scanner(System.in);
   String inputString = null;
   int number = -9999; // Keep compiler happy
   boolean done = false;
   while (! done) {
      try {
         inputString = keyboard.nextLine( );
         inputString = inputString.trim( );
         number = Integer.parseInt(inputString);
```

Example: Method to read an `int`


```
    done = true;
} // end try block
catch (NumberFormatException e) {
    System.out.println("Error: incorrect input.");
    System.out.println("Input number must be a ");
    System.out.println("whole number written ");
    System.out.println("as an ordinary numeral,");
    System.out.println(" such as 21.");
    System.out.println("Minus signs are OK, " +
                       "but do not use a plus sign.");
```

Example: Method to read an `int`


```
            System.out.println("Please try again.");
            System.out.println("Enter a whole number:");
        } // end catch block
    } // end while
    return number;
} // end method
```

Sequence of execution after the exception has been thrown for the last example

- In the Try block:
  1. inputString = keyboard.nextLine( );
  2. inputString = inputString.trim( );
  3. number = Integer.parseInt(inputString);
  4. exception is thrown
- catch (NumberFormatException e) block will be executed
- After the above catch block has been executed, it will NOT return to the the try block and continue (`done = true;`), but continue to execute the code after, i.e. entering the next round of while loop

Defining Your Own Exception Classes

- If you have your own type of exception which you want to be handled separately then define your own
- Eg: in its own file create a class such as

Defining Your Own Exception Classes

```
//DivideByZeroException.java
public class DivideByZeroException extends
                                        Exception {
   //default constructor
   public DivideByZeroException() {
     super("Dividing by Zero!");
     // via Exception constructor
   }
   //constructor with message
   public DivideByZeroException(String message) {
     super(message);
   }
} // end class DivideByZeroException
```

Defining Your Own Exception Classes

- You can now `throw` and `catch` `DivideByZeroExceptions` as shown in the following `DoDivision` class:

Defining Your Own Exception Classes

```
import java.util.Scanner;
public class DoDivision {
   private int numerator, denominator;
   private double quotient;
   public static void main(String[] args) {
      DoDivision doIt = new DoDivision();
      try
      {
         doIt.doNormalCase();
      }
```

Defining Your Own Exception Classes

```
    catch(DivideByZeroException e)
    {
        System.out.println(e.getMessage());
        doIt.giveSecondChance();
    }
    System.out.println("End of Program.");
} // end main
```

Defining Your Own Exception Classes

```
public void doNormalCase() throws
                              DivideByZeroException {
   Scanner keyboard = new Scanner(System.in);
   System.out.println("Enter numerator:");
   numerator = keyboard.nextInt();
   System.out.println("Enter denominator:");
   denominator = keyboard.nextInt();
   if (denominator == 0)
      throw new DivideByZeroException();
   quotient = numerator/(double)denominator;
   System.out.println(numerator + "/" +
                      denominator + " = " + quotient);
}// end doNormalCase
```

Defining Your Own Exception Classes

```
public void giveSecondChance() {
    System.out.println("Try Again:");
    Scanner keyboard = new Scanner(System.in);
    System.out.println("Enter numerator:");
    numerator = keyboard.nextInt();
    System.out.println("Enter denominator:");
    System.out.println("Be sure the denominator
                                    is not zero.");
    denominator = keyboard.nextInt();
```

Defining Your Own Exception Classes

```
        if (denominator == 0) {
            System.out.println("Can't divide by 0.");
            System.out.println("Since what you want
                                        cannot be done, ");
            System.out.println("program will end.");
            System.exit(0);
        } // end if
        quotient = ((double)numerator)/denominator;
        System.out.println(numerator + "/"
                  + denominator + " = " + quotient);
    }
}// end class DoDivision
```

Declaring Exceptions

- Sometimes it is useful not to `catch` exceptions within the method which `threw` them, but to pass the problem back to the method which called that method, or the method which called that method, etc.
- In that case, declare that a method `throws` an exception and don't bother handling the exception within the method

Declaring Exceptions

- Eg:

```
public double evaluate(double n1, double n2)   throws
   DivideByZeroException, UnknownOpException
{
   //body of method which may throw
     //exceptions
}
```

Declaring Exceptions

- The method which invokes `evaluate` must `catch` these exceptions or declare that it `throws` them itself, etc.
- If the main method fails to `catch` exceptions then the program may crash
- Note: some common errors do not need to be declared (`RuntimeExceptions`).

Example: Calculator

```
// Calculator.java (from Savitch)
/** Simple line-oriented calculator program. Class
    can also be used to create other calculator
    programs. */
import java.util.Scanner;
public class Calculator {
  private double result;
  private double precision = 0.0001;
  // Numbers this close to zero are treated as if
  // they are equal to zero.
  public static void main(String[] args) {
     Calculator clerk = new Calculator();
```

Example: Calculator

```
try {
   System.out.println("Calculator is on.");
   System.out.print("Format of each line: "
                              + "operator number");
   System.out.println("For example: + 3");
   System.out.println("Enter 'e' to end.");
   clerk.doCalculation();
} // end try block
```

Example: Calculator

```
    catch(UnknownOpException e) {
       clerk.handleUnknownOpException(e);
    }
    catch(DivideByZeroException e) {
       clerk.handleDivideByZeroException(e);
    }
    System.out.println("The final result is: "
                          + clerk.resultValue());
    System.out.println("Calculator ending.");
} // end main
```

Example: Calculator

```
public Calculator() {   // constructor
    result = 0;
}
public void reset() {
    result = 0;
}
public void setResult(double newResult) {
    result = newResult;
}
public double resultValue() {
    return result;
}
```

Example: Calculator

```
/** The heart of a calculator. Does not give
 instructions. Input errors throw exceptions. */
public void doCalculation() throws
    DivideByZeroException, UnknownOpException {
    char nextOp;
    double nextNumber;
    Scanner keyboard = new Scanner(System.in);
    boolean done = false;
    result = 0;
    System.out.println("result = " + result);
```

Example: Calculator

```
    while (! done) {
       nextOp = keyboard.next().trim().charAt(0);
       if ((nextOp == 'e') || (nextOp == 'E'))
          done = true;
       else {
          nextNumber = keyboard.nextDouble();
          result = evaluate(nextOp, result,
                                          nextNumber);
          System.out.println("result " + nextOp
                +" "+ nextNumber +" = "+ result);
          System.out.println("updated = " +result);
    } // end if
 } // end while
} // end doCalculation method
```

Example: Calculator

```
/** Returns n1 op n2, provided op is one of '+',
 '-', '*',or '/'. Any other value of op throws
 UnknownOpException. */
public double evaluate(char op, double n1,
   double n2) throws DivideByZeroException,
                                    UnknownOpException {
   double answer;
   switch (op) {
      case '+':
         answer = n1 + n2;
         break;
```

Example: Calculator

```
case '-':
   answer = n1 - n2;
   break;
case '*':
   answer = n1 * n2;
   break;
case '/':
   if ((-precision<n2)&&(n2<precision))
      throw new DivideByZeroException();
   answer = n1 / n2;
   break;
```

Example: Calculator

```
            default:
                throw new UnknownOpException(op);
        } // end switch
        return answer;
    } // end evaluate method
```

Example: Calculator

```
public void handleDivideByZeroException(
                        DivideByZeroException e) {
    System.out.println("Dividing by zero.");
    System.out.println("Program aborted.");
    System.exit(0);
}
```

Example: Calculator

```
public void handleUnknownOpException(
                              UnknownOpException e) {
   System.out.println(e.getMessage());
   System.out.println("Try again from the

   beginning:");
   try {
      System.out.println("Format of each line: "
                                 + "operator number");
      System.out.println("For example: +3");
      System.out.println("To end, enter the
                                          letter e.");

      doCalculation();
   } // end try block
```

Example: Calculator

```
        catch(UnknownOpException e2) {
            System.out.println(e2.getMessage());
            System.out.println("Try again at some
                                        other time.");
            System.out.println("Program ending.");
            System.exit(0);
        }
        catch(DivideByZeroException e3) {
            handleDivideByZeroException(e3);
        }
    } // end handleUnknownOpException method
} // end class Calculator
```

Example: Calculator

```
/* Sample test run …
Calculator is on.
Format of each line: operator number
For example: + 3
To end, enter the letter e.
result = 0.0
+  10
result + 10.0 = 10.0
updated result = 10.0
```

Example: Calculator

```
/* Sample test run …
result - 3.0 = 7.0
updated result = 7.0
*  4
result * 4.0 = 28.0
updated result = 28.0
/  2
result / 2.0 = 14.0
updated result = 14.0
/  0
Dividing by zero.
Program aborted. */
```

Example: Calculator

```
// UnknownOpException.java: for use with
   Calculator.java
public class UnknownOpException extends
                                        Exception {
  public UnknownOpException() {
     super("UnknownOpException");
  }
  public UnknownOpException(char op) {
     super(op + " is an unknown operator.");
  }
  public UnknownOpException(String message) {
     super(message);
  }
} // end class UnknownOpException
```

End of Topic 8

Murdoch UNIVERSITY