

COMP 202 Exceptions

CONTENTS:

- Exceptions and Errors
- The try-catch statement
- The try-catch-finally statement
- Exception propagation

Exceptions

- An *exception* is an object that describes an unusual or erroneous situation
 - division by zero
 - reading the wrong data type from a Scanner
 - accessing a non existing array-element
 - out of bound
 - accessing a null object
 - ...

Exceptions

- When such an unusual situation occurs
 - the program *throws* an exception
 - it does not continue with the next statement in the program
 - so far, the program actually terminates
- Instead of letting the program terminate
 - an exception can be *caught* and *handled* by another part of the program
 - that is, the programmer writes special code that is executed whenever an exception is thrown
- A program can therefore 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

Exceptions

- Exceptions:
 - `java.lang.ArrayIndexOutOfBoundsException`
 - `java.lang.StringIndexOutOfBoundsException`
 - `java.lang.NullPointerException`
- Errors:
 - `java.lang.OutOfMemoryError`
 - `java.lang.ClassFormatError`
 - `java.lang.InternalError`
 - `java.lang.VirtualMachineError`

Exception Handling

- If an exception is ignored by the program, the program will terminate and produce an appropriate message
- The message includes a *call stack trace* that indicates on which line the exception occurred
- The call stack trace also shows the method call trail that lead to the execution of the offending line

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 ("Will this line be printed?");
    }
}
```

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
- When an exception occurs, processing continues at the first catch clause that matches the exception type

Using `try-catch`

General format:

```
try {
    // code which may throw an exception
} catch (AException ae) {
    // control goes here if an AException occurs
} catch (BException be) {
    // control goes here if a BException occurs
} ...etc
```

ZeroException

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

        try {
            System.out.println (numerator / denominator);
        }
        catch (ArithmeticException ex)
        {
            System.out.println("Arithmetic error: "+ex.getMessage());
        }
        System.out.println ("Will this line be printed?");
    }
}
```

The finally Clause

- A try statement can have an optional clause designated by the reserved word `finally`
- If no exception is generated, the statements in the finally clause are executed after the statements in the try block complete
- Also, if an exception is generated, the statements in the finally clause are executed after the statements in the appropriate catch clause complete

Using try-catch-finally

General format:

```
try {
    // code which may throw an exception
} catch (AException ae) {
    // control goes here if an AException occurs
} catch (BException be) {
    // control goes here if a BException occurs
} finally {
    // this code is always executed before
    // control flow leaves the try or any catch
}
```

Exception Propagation

- If it is not appropriate to handle the exception where it occurs, it can be handled at a higher level
- Exceptions *propagate* up through the method calling hierarchy until they are caught and handled or until they reach the outermost level
- A try block that contains a call to a method in which an exception is thrown can be used to catch that exception

Zero2.java

```
public class Zero2
{
    //-----
    //  Deliberately divides by zero to produce an exception.
    //-----
    public static void main (String[] args)
    {
        int numerator = 10;
        int denominator = 0;

        divide(numerator, denominator);

        System.out.println ("Will this line of main be printed?");
    }

    public static void divide (int num, int den)
    {
        System.out.println (num / den);
        System.out.println ("Will this line of divide be printed?");
    }
}
```

Zero2Exception.java

```
public class Zero2
{
    //-----
    //  Deliberately divides by zero to produce an exception.
    //-----
    public static void main (String[] args)
    {
        int numerator = 10;
        int denominator = 0;

        try {
            divide(numerator, denominator);
        }
        catch (ArithmeticException ex) {
            System.out.println("Arithmetic Error: "+ex.getMessage());
        }
        System.out.println ("Will this line of main be printed?");
    }

    public static void divide (int num, int den) {
        System.out.println (num / den);
        System.out.println ("Will this line of divide be printed?");
    }
}
```

Three ways to handle Exceptions

- ignore the exception
 - the program terminates
- handle the exception where it occurs
 - the exception handling code resides in the method that throws the exception
- handle the exception at another place in the program
 - the exception handling code resides somewhere in the calling hierarchy (method calls method that calls method... that calls method that throws the exception)

WildernessIndex.java

```
public class WildernessIndex
{
    static public void main (String[] args)
    {
        WorldZoom wildIndex = new WorldZoom();

        System.out.println("Picking a country...");
        wildIndex.theUS();

        System.out.println("\nPicking another country...");
        wildIndex.canada();

        System.out.println("\nDone.");
    }
}
```

WorldZoom.java (1/3)

```
class WorldZoom {
    public void canada() {
        System.out.println("Zooming in to Canada.");

        try
        {
            quebec();
        }
        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("Zooming out of Canada.");
    }
}
```

```
public void quebec() {
    System.out.println("Zooming in to Quebec.");
    montreal ();
    System.out.println("Zooming out of quebec.");
}

public void montreal(){
    int numPeople = 3000000, numBears = 0;

    System.out.println("Zooming in to Montreal.");
    int result = numPeople / numBears;
    System.out.println("The wilderness index is: " + result);
    System.out.println("Zooming out of Montreal.");
}

public void alaska() {
    System.out.println("Zooming in to Alaska.");
    kodiak ();
    System.out.println("Zooming out of Alaska.");
}

public void kodiak(){
    int numPeople = 13000, numBears = 3000;

    System.out.println("Zooming in to kodiak island.");
    int result = numPeople / numBears;
    System.out.println("The wilderness index is: " + result);
    System.out.println("Zooming out of kodiak island.");
}
}
```

```
public void theUS() {
    System.out.println("Zooming in to the US.");

    try
    {
        alaska();
    }
    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("Zooming out of the US.");
}
}
```

Checked Exceptions

- An exception is either *checked* or *unchecked*
- So far unchecked exceptions
 - they are the default handling procedure
 - can but do not need to be caught or propagated but
 - if not caught anywhere then program simply terminates
- A checked exception
 - must be caught within within a try/catch block within the method in which it occurs
 - can be propagated to the outer method
 - but then the method that throws the exception must declare this
 - A *throws* clause must be appended to the header of the method
 - We will see the throws clause when we handle files
 - The compiler will complain if a checked exception is not handled or declared appropriately