

Exceptions - Chapter 11

- Exceptions are errors or unexpected actions.
- Some examples are:
 - IndexOutOfBoundsException
 - NullPointerException
 - NumberFormatException
 - ArithmeticException
 - FileNotFoundException
- In Java we can "catch" them and try to recover.

1

Robust Programs

- A robust program deals gracefully with unexpected input (among other things).

```
int myData = Console.in.readInt();
```

- How can we make this more robust? More specifically, what happens if the user doesn't enter an integer?

2

```
class Status {  
    boolean flag;  
}  
  
Status status = new Status();  
int myData = Console.in.readInt(status);  
if (!status.flag) {  
    // put error handling code here  
}
```

3

Problems with this approach

- Had to modify readInt().
- Needed to declare status in our code.
- Needed to test **status** even though we "expect" it to always be true.
- Even worse, what if the preceding code was inside another method?

4

This example just passes the status up the line, returning a bogus value to keep the compiler happy.

```
int processInput(..., Status status) {  
    ...  
    int myData = Console.in.readInt(status);  
    if (!status.flag) {  
        // have to return something  
        return 0; // assume return value is ignored  
    }  
    // go on with normal processing  
    ...  
}
```

5

try-catch

- Exception handling is language support for the previous scenario.
- When something goes wrong an exception is "thrown".
- The code that wants to deal with the exception can "catch" it.

6

```

try {
    // some code here that might throw an exception
}
catch (ExceptionType Identifier) {
    // some code here to recover from the problem
}

```

7

```

int myData;
try {
    myData = Console.in.readInt();
}
catch (NumberFormatException e) {
    // some code here to recover from the problem
}

int processInput(...) {
    ...
    int myData = Console.in.readInt();
    // go on with normal processing
}

```

8

```

import tio.*;
public class ExceptionExample {
    public static void main(String[] args) {
        int aNumber = 0;
        boolean success = false;
        String inputString = "";
        System.out.println("Type an integer.");
        while (!success) {
            try {
                aNumber = Console.in.readInt();
                success = true;
            }
            catch (NumberFormatException exception) {
                inputString = Console.in.readWord();
                System.out.println(inputString +
                    " is not an integer. Try again!");
            }
        }
        System.out.println("You typed " + aNumber);
        // continue with code to process aNumber
    }
}

```

9

```

import java.io.*;
class BinaryInput {
    public static void main(String[] args)
        throws IOException
    {
        DataInputStream input = null;
        if (args.length != 1) {
            System.out.println("Usage: " +
                "java BinaryInput filename");
            System.exit(1);
        }
        try {
            input = new DataInputStream(
                new FileInputStream(args[0]));
        }
        catch (IOException e) {
            System.out.println("Could not open " + args[0]);
            System.out.println(e);
            System.exit(1);
        }
    }
}

```

10

```

// count is used to print 4 values per line
int count = 0;
try {
    while (true) {
        int myData = input.readInt();
        count++;
        System.out.print(myData + " ");
        // print a newline every 4th value
        if (count % 4 == 0)
            System.out.println();
    }
}
catch (EOFException e)
{
    // just catch the exception and discard it
}
// add a newline after the last partial line
// if necessary
if (count % 4 != 0)
    System.out.println();
}
}

```

11

```

// ExceptionExampleTwo.java - show control flow when
// an exception occurs during nested method calls
import tio.*;
class ExceptionExampleTwo {
    public static void main(String[] args) {
        int x = 0;
        System.out.println("main starting");
        try {
            x = callOne();
            System.out.println("callOne OK x = " + x);
        }
        catch (ArithmeticException e) {
            System.out.println("callOne not OK: " + e);
            x = -1;
        }
        System.out.println("main exiting x = " + x);
    }
}

```

12

```

static int callOne() {
    System.out.println("callOne starting");
    int result = callTwo();
    System.out.println("callOne returning result = "
        + result);

    return result;
}
static int callTwo() {
    int num = 0;
    System.out.println("type a number");
    int input = Console.in.readInt();
    num = 1000 / input;
    System.out.println("callTwo returning num = "
        + num);

    return num;
}
}

```

13

Nested Exceptions

- Run ExceptionExampleTwo first giving it 10 as the input and the give it 0.

14

```

// TwoCatchExample.java - use two catch clauses
...
while (!success) {
    try {
        aNumber = Console.in.readInt();
        success = true;
        System.out.println("You typed " + aNumber);
    }
    catch (NumberFormatException exception) {
        inputString = Console.in.readWord();
        System.out.println(inputString +
            " is not an integer. Try again!");
    }
    catch (IOException exception) {
        System.out.println(
            "Continuing with default value 0.");
        aNumber = 0;
        success = true;
    }
}
// continue with code to process a_number

```

15

```

// BinaryInput2.java - read some integers from
// a binary file
import java.io.*;

class BinaryInput2 {
    public static int readBinaryInput(String filename,
        int howMany)
        throws IOException
    {
        DataInputStream input = null;
        try {
            input = new DataInputStream(
                new FileInputStream(filename));
        }
        catch (IOException e) {
            System.out.println("Could not open " +filename);
            System.out.println(e);
            throw e;
        }
    }
}

```

16

```

int count = 0;
try {
    while (count < howMany) {
        int myData = input.readInt();
        System.out.print(myData + " ");
        // print a newline every 4th value
        if (++count % 4 == 0)
            System.out.println();
    }
}
catch (EOFException e) { /* ignore */}
finally {
    if (count % 4 != 0)
        System.out.println();
    if (input != null)
        input.close();
}
return count;
}
}

```

17

Throwing Exceptions

- Some exceptions are thrown "automatically" by the Java Virtual Machine. E.g. IndexOutOfBoundsException
- You can also throw them yourself.

18

```

public class Counter {
    //constructors
    public Counter() {}

    public Counter(int v) {
        if (v < 0 || v >= MAX)
            throw new Exception("Invalid intial value.");
        else
            value = v % MAX;
    }
    ...
}

```

19

The throws clause

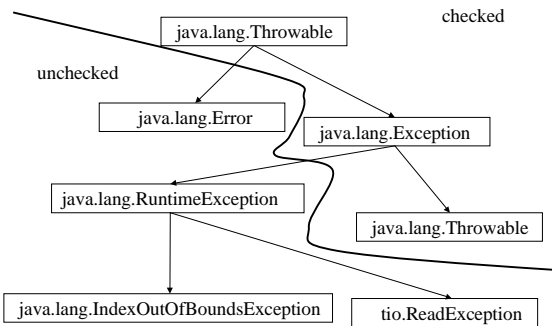
- When do you need a throws?
- There are two types of exceptions in Java
 - checked exceptions, and
 - unchecked exceptions.
- Checked exceptions require a throws clause whenever they might be thrown.
- Unchecked exceptions are things like `NullPointerException`, and `IndexOutOfBoundsException`.

20

Checked vs Unchecked

- Unchecked exceptions are exceptions that are instances of `java.lang.RuntimeException`, `java.lang.Error`, or one of their subclasses.
- Everything else is a checked exception.

21



22