Exceptions (JFS Chapter 16)

- In this section:
  - What an "exception" is
  - Approaches to handling the problem
  - The Java exception-handling mechanism
  - Exception-handling example

What is an "exception"?

- An "exception" occurs when a program is running and something "goes wrong"
- Examples:
  - The program tried to do a division sum
    but it was division by zero
  - The program tried to open a file
    but it doesn't exist (files later)
- Note: These may *not* be due to *logic errors* in the program
  - Sometimes the problem *cannot be detected in advance*
  - Sometimes it is appropriate or necessary to allow an exception to occur, and to then take remedial action
- The following slides show what might happen if we do *not* take remedial action…

Demo (AreaDialogs)

System Messages – BlueJ pops up an extra window

```
Exception occurred during event dispatching:
java.lang.ArithmeticException: / by zero
    at Excep.actionPerformed(Excep.java:24)
    at java.awt.TextField.processActionEvent(TextField.java:513)
    at java.awt.TextField.processEvent(TextField.java:486)
    at java.awt.Component.dispatchEventImpl(Component.java:2595)
    at java.awt.Component.dispatchEvent(Component.java:2499)
    at java.awt.EventQueue.dispatchEvent(EventQueue.java:319)
    at java.awt.EventDispatchThread.pumpOneEvent(EventDispatchThread.java:103)
    at java.awt.EventDispatchThread.pumpEvents(EventDispatchThread.java:93)
    at java.awt.EventDispatchThread.run(EventDispatchThread.java:84)
```

Note: This gives a description of the fault (/ by zero) and an indication of which step in our program was executing when the fault occurred:

In class **Excep**, in method **actionPerformed**, in file **Excep.java** at line 24

System Messages

```
Exception occurred during event dispatching:
java.lang.NumberFormatException: abc
    at java.lang.Integer.parseInt(Integer.java:405)
    at java.lang.Integer.parseInt(Integer.java:454)
    at Excep.actionPerformed(Excep.java:23)
    at java.awt.TextField.processActionEvent(TextField.java:513)
    at java.awt.TextField.processEvent(TextField.java:486)
    at java.awt.Component.dispatchEventImpl(Component.java:2595)
    at java.awt.Component.dispatchEvent(Component.java:2499)
    at java.awt.EventQueue.dispatchEvent(EventQueue.java:319)
    at java.awt.EventDispatchThread.pumpOneEvent(EventDispatchThread.java:103)
    at java.awt.EventDispatchThread.pumpEvents(EventDispatchThread.java:93)
    at java.awt.EventDispatchThread.run(EventDispatchThread.java:84)
```

The exception occurred in method **Integer.parseInt**, trying to convert "abc" to an **int**, but that method was called by our **actionPerformed**, in file **Excep.java** at line 23

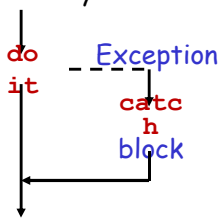
Solutions

- The simplest solution to exceptions is just to ignore them! Maybe the program will be able to continue ...
  - This is unprofessional - real software should not stop unpredictably, nor continue with unreliable data
  - Also, if an un-dealt with exception occurs, then the JVM ***abandons*** the current event handling and waits for the next event – *so some work may not have been done!* (See demo again)
- So we ought to deal with exceptions explicitly
- We could use `if` statements
  - This can get awkward
  - And sometimes may not be possible
- Java has a special feature to help us…

The Java Exception-handling Mechanism

- We place the problematical code `do it;` inside a try-catch:

```
try {
  do it;
}
catch (ExceptionType ex) {
  handle the problem;
}
```



- We note that the code `do it;` might "throw an exception"
  - and we provide a section of program that "catches" the exception
- If `do it;` does *not* throw an exception, then the `catch` section (the "catch block") is ignored
- If any part of `do it;` *does* throw a recognised exception, execution jumps *immediately* to the catch block, and the rest of `do it;` is *ignored*

Different Kinds of Exception

- Most exceptions are thrown by library methods that detect a problem
  - For example, in class `Integer` we find:

```
public static int parseInt(String s)
            throws NumberFormatException
```

  - It throws the exception if we apply it to a string which is not all digits
  - If we wish to catch and handle an exception thrown by `parseInt` then we must call it like this:

```
try {
  n = Integer.parseInt(s);
}
catch (NumberFormatException nfe) {
  n = 99;    // (say)
}
```

  - Note that the `catch` indicates the kind of exception it will recognise

- Another example is an attempt to divide by zero, which would cause an `ArithmeticException`
- We are not obliged to catch exceptions like
  `ArithmeticException` and
  `NumberFormatException`
  - If not caught these exceptions cause the current event-handler to be terminated immediately - but the program remains "alive"
- But for some kinds of exceptions the compiler *forces us to include a try-catch*, e.g. `IOException`
  - *We* decide what is appropriate in the catch block, eg:

```
try {
  // open an input file
}
catch (IOException ioe) {
  // Report problem/Take some recovery action
}
```

Exception handling: Example 1

```
public void actionPerformed(ActionEvent e) {
  try {
    inValue = Integer.parseInt(forNum.getText());
    quotient = 10/inValue;
    ... some drawing ...
  }
  catch (NumberFormatException nfe) {
    JOptionPane.showMessageDialog(null,
                          "Error: Enter a number");
    forNum.setText("");
  }
  catch (ArithmeticException ae) {
    JOptionPane.showMessageDialog(null,
                        "Error: Cannot divide by 0");
    forNum.setText("");
  }
}
```

Multiple catches, one try only

Exception handling: Example 2

```
public void actionPerformed(ActionEvent event) {
  int length = 0; ...
  boolean lengthOK;
  do {
    try {
      lengthString =
             JOptionPane.showInputDialog("Length:");
      length = Integer.parseInt(lengthString);
      lengthOK = true;
    }
    catch (NumberFormatException exc) {
      JOptionPane.showMessageDialog(null,
        "Length should be entered as a valid integer");
      lengthOK = false;
    }
  }
  while (!lengthOK);
  ...
```

AreaDialogs:
Consider `length` only
Repeated tries

Demo (AreaDialogsSafe)

Throwing Exceptions

- Exceptions do not need to be caught immediately at the program level at which they occur
- An exception may occur inside one of *our* methods. A method (method A, say) has a choice:
  - Deal with the exception on the spot (with a try-catch), or
  - Pass the buck to whatever method (method B, say) has called method A
- In the latter case, method A will have no try-catch, but it will **throw** the exception (pass the buck). Its header will be like this:

```
private void inputOp() throws IOException {
```

- Any method that calls this method will *itself* have the responsibility of calling it within a try-catch

End of Section