Exceptions

© Copyright IBM Corporation 2008
Course materials may not be reproduced in whole or in part without the prior written permission of IBM.

Topics

- iSeries exception model
- OPM versus ILE exception model
- RPG (language) exception handling
- Java exception model
- Sending exceptions in Java
- Monitoring for exceptions in Java
- Throws clause
- Rethrowing exceptions
- Unhandled exceptions

© Copyright IBM Corporation 2008

Unit objectives

After completing this unit, you should be able to:

- Describe the Java exception model and how it differs from the RPG model
- Write Java code that defines and manages Java exceptions

© Copyright IBM Corporation 2008

OS/400 exceptions

- On iSeries when something 'exceptional' or unexpected happens, a message is sent.

- Messages imbed
  - Error message text with substitution variables
  - Message severity

- Messages have a unique 7-digit number.

- In your CL program you use:
  - MONMSG to monitor for specific number
  - SNDPGMMSG to send a message

© Copyright IBM Corporation 2008

iSeries exception model

OPM

- Program call stack entry handle exception?
  - If Yes, you are finished
  - If No, then generate a function check (msg CPF9999)
- Program call stack entry handle CPF9999?
  - If Yes, you are finished
  - If No, terminate pgm and send CPF9999 to previous entry.

ILE

- Exc passed up call stack until handler found
  - **If nobody handles, converted to CPF9999 and process repeated**
  - **If someone handles it, entries above it are all terminated**
- Each entry on call stack that does not handle function check is removed.
  - **Depending on user answer to an inquiry message**

© Copyright IBM Corporation 2008

RPG exception handling

- RPG divides exceptions into two camps:
  - File errors: Occurs when processing files such as record not found
  - Program errors: Programming errors such as divide by zero

- Four ways to handle exceptions:
  - Error indicators on many op-codes
  - %ERROR built-in function
  - INFSR error subroutine for file errors
  - *PSSR error subroutine for program errors

© Copyright IBM Corporation 2008

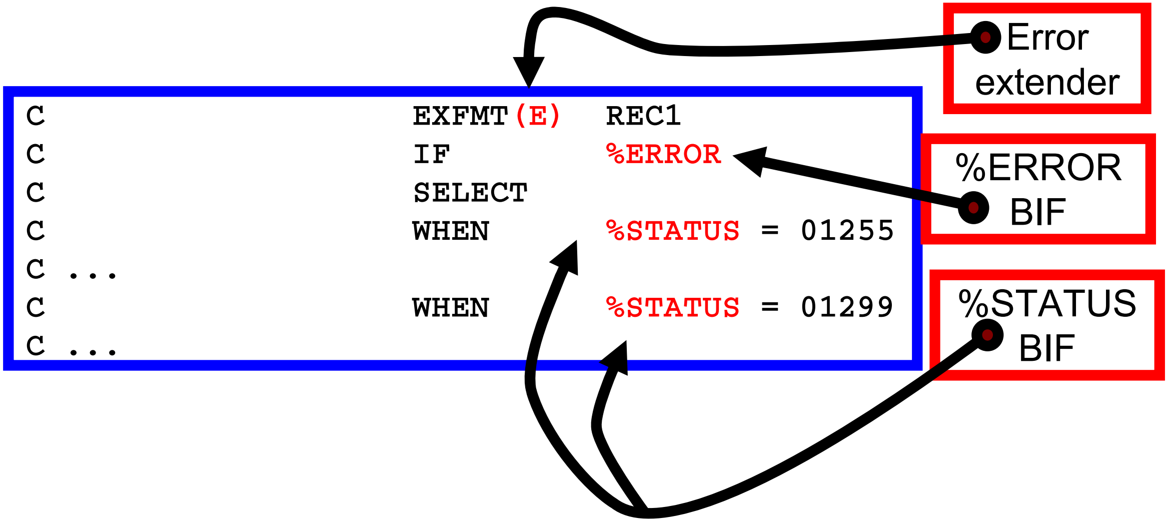
RPG error built-in functions

- RPG has new built-in functions as of V4R2, replacing need for error indicators:
  - %ERROR: Returns one if most recent operation resulted in an error
    - Operation must have specified error ('E') extender
  - %FOUND(file): Returns '1' if most recent relevant operation found a record (CHAIN, DELETE, SETGT, SETLL), an element (LOOKUP), or match (CHECK, CHECKR, SCAN)
  - %EQUAL(<file>): one if SETLL / LOOKUP match

- Other new related BIFs:
  - %EOF(file), %STATUS(<file>), %OPEN(file)

© Copyright IBM Corporation 2008

RPG error built-in functions example



```
C                   EXFMT(E)  REC1
C                   IF        %ERROR
C                   SELECT
C                   WHEN      %STATUS = 01255
C ...
C                   WHEN      %STATUS = 01299
C ...
```

© Copyright IBM Corporation 2008

RPG MONITOR group

- RPG has new MONITOR support as of V5R1:
  - Place all statements that might result in error between MONITOR and ENDMON statements.
    - New op-codes
  - Follow sensitive statements with ON-ERROR statements, specifying the status code that this ON-ERROR block handles.
    - The ON-ERROR blocks go before the ENDMON statement
    - The code to handle each status code error goes between the ON-ERROR statements
    - To handle all program-errors, use *PROGRAM versus a status code
    - To handle all file-errors, use *FILE versus a status code
    - To handle all errors, use *ALL or nothing versus a status code
    - When an error occurs, the first applicable ON-ERROR block of statements are executed.

© Copyright IBM Corporation 2008

RPG MONITOR example

```
 * The MONITOR block consists of the READ statement and the IF group.
 * - The first ON-ERROR block handles status 1211 which
 *   is issued for the READ operation if the file is not open.
 * - The second ON-ERROR block handles all other file errors.
 * - The third ON-ERROR block handles the string-operation status
 *   code 00100 and array index status code 00121.
 * - The fourth ON-ERROR block (which could have had a factor 2
 *   of *ALL) handles errors not handled by the specific ON-ERROR
 *   operations.
 * If no error occurs in the MONITOR block, control passes from the
 * ENDIF to the ENDMON.
C                  MONITOR
C                  READ       FILE1
C                  IF         NOT %EOF
C                  EVAL       Line = %SUBST(Line(i) :
C                                    %SCAN('***': Line(i)) + 1)
C                  ENDIF
C                  ON-ERROR   1211
C                   ... handle file-not-open
C                  ON-ERROR   *FILE
C                   ... handle other file errors
C                  ON-ERROR   00100 : 00121
C                   ... handle string error and array-index error
C                  ON-ERROR
C                   ... handle all other errors
C                  ENDMON
```

© Copyright IBM Corporation 2008

Full RPG example

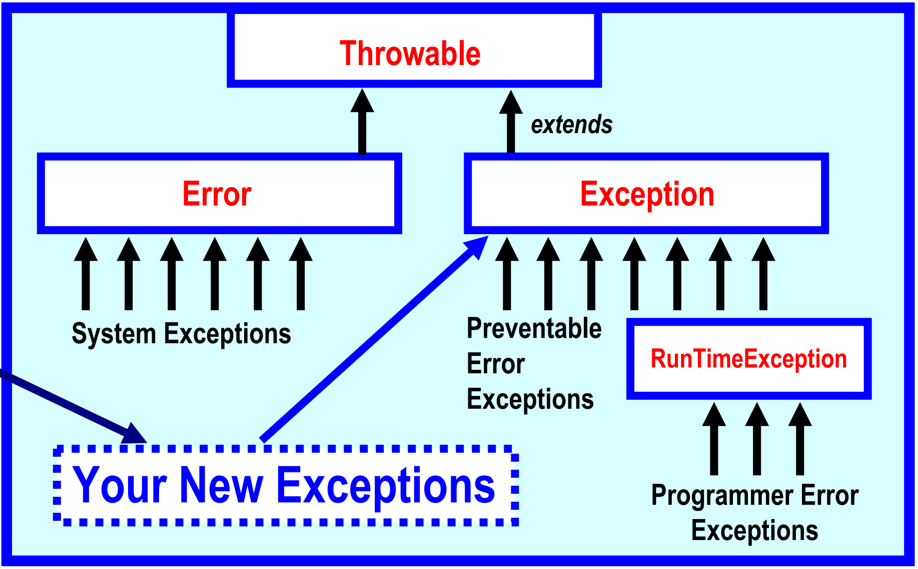
```
FMASTER     UF    E                K DISK
F                                            INFSR(FILEINF)
F                                            INFDS(FILEDS)
D FILEDS             DS
D  MASTERSTS           *STATUS
D*error 1221 - update without previous read
D*-----------------------------------------------------
C      Number           CHAIN     MASTER                    99
C                       IF        *IN99
C*   do error processing
C                       ELSE
C*                      No errors
C                       UPDATE     MASREC
C*                      :
C      20               DIV        0         Result   5 0
C*                      :
C                       ENDIF
C      FILEINF          BEGSR
C*     *******HANDLE FILE ERRORs and exceptions ********
C                       ENDSR
C     *PSSR             BEGSR
C*     *******      Handle Program errors        ********
C                       ENDSR
```

Exception handling in Java (1 of 2)

- Java has exceptions
  - These are Java objects
- All Java exceptions inherit from class Throwable.
  - Class Throwable is in package java.lang
- Any class that extends Throwable is an exception in Java.
- Throwable objects contain a string describing the exception.
  - Use method getMessage() to get it
- Another useful method is printStackTrace()
- The primary subclasses of Throwable are:
  - Error class
  - Exception class
- You can create your own subclasses of Exception.

© Copyright IBM Corporation 2008

Exception handling in Java (2 of 2)



You can define your own!!!

Example:

```
public class BadZipCode extends Exception
{
    ...
}
```

© Copyright IBM Corporation 2008

Example

```
public class BadZipCode extends Exception
 {
     private String badzip;
     private String method;

     public BadZipCode(String badzip, String method)
     {
         super("Bad zipcode '" + badzip + "' given.");
         this.badzip = badzip;
         this.method = method;
     }
     public String getBadZip()
     {
        return badzip;
     }
     public String getMethod()
     {
        return method;
     }
 } // end class BadZipCode
```

© Copyright IBM Corporation 2008

Sending exceptions

1. Instantiate an instance of the Exception child class
2. Use throw operator to signal an exception
3. Specify "throws exception-name" in method definition

```
public void myMethod(ZipCode zipcode) throws BadZipCode 3
{
  if (zipcode.isbad())
    {
      BadZipCode exc = new BadZipCode(zipcode.asString(), 1
                                       "myMethod");
      throw(exc); 2
      // or:
      // throw(new BadZipCode(zipcode.asString,"myMethod"));
    }
  ...
}
```

Example

- Throw operator is similar to CL's SNDPGMMSG.
- Code after the throw operator is not executed.

© Copyright IBM Corporation 2008

What to throw

- Use exceptions instead of return codes for truly exceptional situations:
  - Invalid input
  - File not found
  - Unexpected communications error
- But use return codes for normal situations
  - End of file
- What exception to throw?
  - Use Java supplied exception if possible:
    - java.lang.Exception (generic)
    - java.io.IOException (invalid input)
    - java.io.FileNotFoundException (file not found)
    - java.io.ObjectNotFoundException (object not found)
  - Create your own when necessary

© Copyright IBM Corporation 2008

Handling exceptions

How do you *monitor* for exceptions?

Use the **try-catch** statement

Java

```
try
{
  // try-block: one or more statements of code
}
catch (Exception exc)
{
  // catch-block: code to handle the exception
}
```

try-block

catch-block

- Place any method call that may throw exceptions inside a try-block
  - If an exception occurs, the catch-block will get control

© Copyright IBM Corporation 2008

Try and catch (1 of 2)

- Catch-block defines a parameter
  - The exception it will handle
- At runtime, the exception object that was thrown will be passed.

```
try
{
    ...
    myObject.myMethod(zipcode);
    ...
}
catch (BadZipCode exc)
{
    System.out.println(exc.getMessage());
    System.out.println("... in method " + exc.getMethod());
}
```

© Copyright IBM Corporation 2008

Try and catch (2 of 2)

- Catch block is equivalent to CL MONMSG.
  - Tells Java the exception type to monitor for:
    - catch (MyException exc)
- You can specify an explicit exception class to monitor, or a parent class.
  - Any exceptions of a child class are also caught
  - Thus catch (Exception exc) will catch all exceptions
- Many Java methods throw exceptions, too
  - You only monitor for Exception subclasses
  - Error and RunTimeException subclasses do not need to be monitored: system errors!

© Copyright IBM Corporation 2008

Multiple catches (1 of 2)

- You can specify more than one catch block:
  - `catch (MyException exc) { ... }`
  - `catch (YourException exc) { ... }`

- Upon receipt of an exception, Java will match the actual exception class type to the declared exception type and run that catch block
  - Remember. it will match the exception to the first matching explicit type or a parent type

- You can also specify an ending finally-block
  - `finally () { ... }`
    - Be careful! `finally` always run if it exists, regardless of whether the run-time exception is caught.
      - Your opportunity to do "always needed" cleanup, such as closing open files

© Copyright IBM Corporation 2008

Multiple catches (2 of 2)

Example:

```
try
{
   ...
   myObject.myMethod(zipcode);
   ...
}
catch (BadZipCode exc)
{
   System.out.println(exc.getMessage());
   System.out.println("... in method " + exc.getMethod());
}
catch (IOException exc)
{
   System.out.println("Bad input - naughty, naughty!");
}
finally
{
   // always required clean-up code
}
```

```
try
  block
catch (exception-1 identifier)
  block
catch (exception-2 identifier)
  block
finally
  block
```

Java

© Copyright IBM Corporation 2008

Throws clause (1 of 2)

- javac compiler forces you to identify all exceptions that your method throws in the **throws** clause of your method...

```
public static void myMethod2()
{
    throw (new Exception("testing"));
} // end of method myMethod2
```

```
C:\>javac TestTryCatch.java
TestTryCatch.java:22: Exception
java.lang.Exception must be caught, or it must be
declared in the throws clause of this method.
```

© Copyright IBM Corporation 2008

Throws clause (2 of 2)

- The throws clause is a form of forced documentation for users of your method
  - They know what to monitor for!

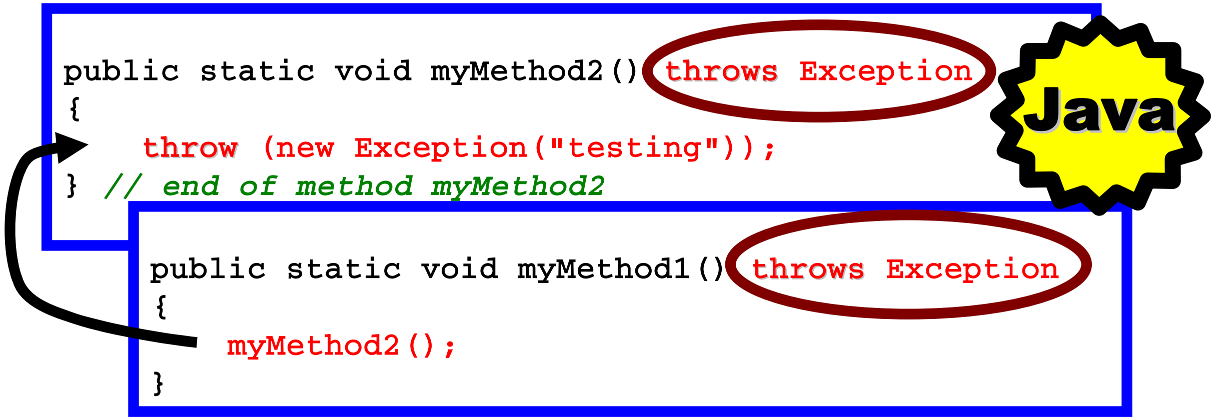
```
public static void myMethod2() throws Exception
{
     throw (new Exception("testing"));
} // end of method myMethod2
```

```
public static void myMethod1()
{
     try {
        myMethod2();
     }
     catch (Exception exc) {
        System.out.println("Error: " + exc.getMessage());
     }
}
```

© Copyright IBM Corporation 2008

Rethrowing

- You can choose not to monitor for an exception
  - Instead you can "send it up the call stack"
  - Simply specify "throws XXX" on your method instead of monitoring via try-catch



```
public static void myMethod2() throws Exception
{
    throw (new Exception("testing"));
} // end of method myMethod2
```

```
public static void myMethod1() throws Exception
{
    myMethod2();
}
```

© Copyright IBM Corporation 2008

Unhandled exceptions

- What happens if you call a method that throws an exception you do not catch; for example, MyException?
  - Does calling method specify "throws MyException"?
    - No: Your code will not compile!!!
    - Yes: Exception is sent to method that called you
- If nobody catches the exception:
  - Compile will fail unless all calling methods specify the exception on the throws clause
  - If they all do, program will end!! Somebody has to "catch" the exception eventually, to prevent this

```
java.lang.Exception: testing
        at TestTryCatch.myMethod2(TestTryCatch.java:24)
        at TestTryCatch.myMethod1(TestTryCatch.java:17)
        at TestTryCatch.main(TestTryCatch.java:11)
```

© Copyright IBM Corporation 2008

Full Java example: Part 1

```
void openFile(String fileName) throws NotFoundException,
                                      ReadOnlyException
{
    if (!findFile(fileName))
      throw (new NotFoundException(fileName));
    else if (isFileReadOnly(fileName))
      throw (new ReadOnlyException(fileName));
    else
      // open the file
}
```

```
String getAndOpenFile() throws NotFoundException,
                               ReadOnlyException
{
    String fileName = console.askForFileName();
    openFile(fileName);
    return fileName;
}
```

. . . next page

© Copyright IBM Corporation 2008

Full Java example: Part 2

```
public void mainMethod()
{
    String fileName = null;
    try
    {
      fileName = getAndOpenFile();
      // do application processing...
    }
    catch (NotFoundException)
    {
      System.out.println("File not found");
    }
    catch (ReadOnlyException)
    {
      System.out.println("File is read only");
    }
    finally
    {
      closeFile(fileName);
    }
}
```

Java

© Copyright IBM Corporation 2008

Topics covered

- iSeries exception model
- OPM versus ILE exception model
- RPG (language) exception handling
- Java exception model
- Sending exceptions in Java
- Monitoring for exceptions in Java
- Throws clause
- Rethrowing exceptions
- Unhandled exceptions

© Copyright IBM Corporation 2008

Unit summary

Having completed this unit, you should be able to:

- Describe the Java exception model and how it differs from the RPG model
- Write Java code that defines and manages Java exceptions

© Copyright IBM Corporation 2008