Exceptions Handling

In Java programming

JOHN BRYCE
Leading in IT Education

JOHN BRYCE
Leading in IT Education
a matrix company

Objectives

JOHN BRYCE
Leading in IT Education
matrix company

1 Java Technology

2 Object Oriented Analysis and Design

3 Identifiers Keywords and Types

4 Expressions and Flow Control

5 Arrays

6 Class Design

7 Advanced Class Features

8 Exceptions

9 Collections

10 Lasest Syntax Enh.

11 IO

12 Threads

13 JDBC

Objectives

JOHN BRYCE
Leading in IT Education
matrix company

By the end of this session

- You'll be able to use runtime error mechanism
- You'll be able to create custom exceptions

Exceptions

JOHN BRYCE
Leading in IT Education
matrix company

- Are java objects
- Represent many types of problems that may occur during program execution
- Can be handled in different ways
- May stop program flow if not handled

Type of Exceptions

JOHN BRYCE
Leading in IT Education
matrix company

Error | defines serious error conditions

Exception | – program errors
Run-time Exception
Others - I/O
- SQL
- other customized application exception

Errors

JOHN BRYCE
Leading in IT Education
matrix company

- Used to indicated problems that mostly cannot be fixed in runtime

- AWTError
- VirtualMachineError

  StackOverflowError

  OutOfMemoryError

  InternalError (unexpected problem in the VM)

Exceptions

JOHN BRYCE
Leading in IT Education
matrix company

There are two kinds of Exceptions:

1- RuntimeException

- any exception that extends RuntimeException
- counted as bugs and must be fixed to complete app
- unchecked by the compiler – developer responsibility

2- Application Exceptions

- any exception that doesn't extend RuntimeException
- user defined exceptions
- are NOT bugs !! And therefore checked by the compilers

RuntimeException

JOHN BRYCE
Leading in IT Education
matrix company

- Any error raised due to the application logic or miscalculation
- Programmer does not have to handle it – but…

  The consequences will be stopping the program
- Some RuntimeExceptions:

  ArithmeticException
  NullPointerException
  NegativeArraySizeException
  ArrayIndexOutOfBoundsException
  SecurityException
  NumberFormatException
  ClassCastException
- Since we don't want to provide

  applications with bugs,

  developers will eliminate these exceptions from occurring

Application Exception

JOHN BRYCE
Leading in IT Education
matrix company

- Usually thrown when any external implementation is involved – such as:

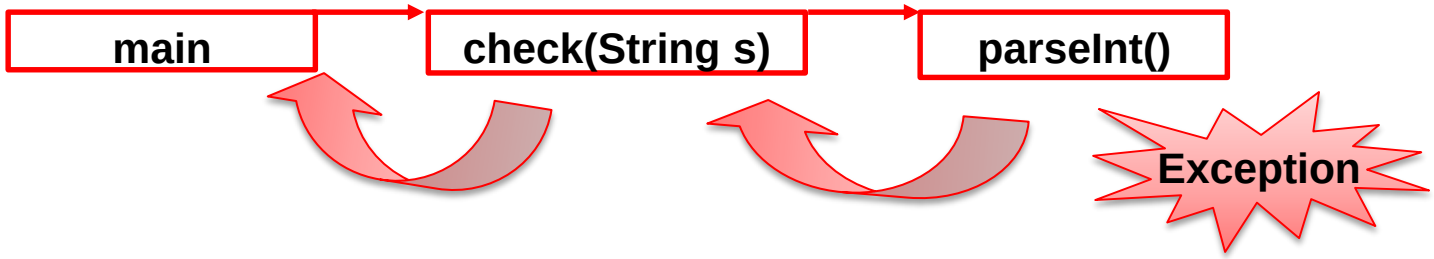
  File System
  JDBC Drivers
  XML Parsers

- Programmer must handle it (compilation error)
- Some ApplicationExceptions:

  IOException [EOFException, FileNotFoundException…]
  SQLException
  DOMException, SAXException
  ClassNotFoundException
  RemoteException
  AWTException

Call Stack Mechanism

- both exceptions and errors extend Throwable
- It allows a method that generates exception to 'throw' it up the stack
- If the thrown exception reaches main() without being handled – the program stops



Handling Exceptions

JOHN BRYCE
Leading in IT Education
matrix company

- Done in two ways:

  Catching exceptions

  Or

  Throwing Exceptions

- Application Exception must be caught or thrown
- Runtime Exception may be caught and can be thrown

Catching Exception

JOHN BRYCE
Leading in IT Education
matrix company

Writing problematic code in a try & catch block

try

- opens a block to 'try' and execute

catch

- catches the exception if thrown
- written for each Exception [& its subclasses]
- closes the try block and opens a new one to be executed when exception is caught
- catching order should be considered

finally

- a "do it anyway" block
- is optional

Catching Exception

JOHN BRYCE
Leading in IT Education
matrix company

Example – multi-catch-blocks

```
public void check(String fileName, String value) {
    try{
        FileInputStream in=new FileInputStream(fileName);
        int data=in.read();
        …
    }catch(FileNotFoundException e){
        //handle I/O problem…
    }catch(EOFException ex){
        //handle end of file exception…
    }
    System.println("Done !");
}
```

Catching Exception

JOHN BRYCE
Leading in IT Education
matrix company

Example – super catch

```
public void check(String fileName, String value) {
    try{
        FileInputStream in=new FileInputStream(fileName);
        if(Integer.parseInt(value) >= 100) {
                return;
        }
        …
    }catch(Exception){
        //handle I/O & runtime problems…
    }
    System.println("Done !");
}
```

Catching Exception

JOHN BRYCE
Leading in IT Education
matrix company

Example – finally

```
public void check(String fileName, String value) {
    try{
        FileInputStream in=new FileInputStream(fileName);
        if(Integer.parseInt(value) >= 100) {
                return;
        }
        …
    }catch(FileNotFoundException){
        //handle I/O problem…
        return;
    }catch(NumberFormatException){
        //handle runtime exception…
    }finally{
        System.println("This is printed in any case….");
    }
    System.println("Done !");
}
```

Catching Exception

JOHN BRYCE
Leading in IT Education
matrix company

- Should be written as close to the origin throwing point as possible
- Catching `java.lang.Exception` will catch all types of exceptions
- Use `java.lang.Exception` methods to get information:

Exception

• getMessage()	Returns a message describes this exception
• printStackTrace (out)	Prints the stack trace – good for debugging

```
}catch(Exception e){  System.out.println(e.getMessage());
        e.printStackTrace(System.out);
}
```

Exception Throwing

JOHN BRYCE
Leading in IT Education
matrix company

- Any method can delegate exceptions to the caller
- A method must declare any thrown Exception as part of its signature
- Throwing Runtime Exceptions is allowed but not always necessary
- **throws** - declares all thrown exceptions
- **throw** - actually creates an exception and throw it

Exception Throwing

JOHN BRYCE
Leading in IT Education
matrix company

Example

```
public class Check{

 public static int check(String s) throws NumberFormatException{
        return Integer.parseInt(s);
 }

 public static void main(String[] args) {
        int num=check(args[0]);
        System.out.println(num+1);
 }
}
```

Exception Throwing

JOHN BRYCE
Leading in IT Education
matrix company

Example

```
public class Check{

  public static int check(String s) throws NumberFormatException{
          int x=Integer.parseInt(s);
          if(x>100)
               throw new NumberFormatException("Number is too big");
           return x ;
  }

  public static void main(String[] args) {
         int num=check(args[0]);
         System.out.println(num+1);
  }
 }
```

Method Overriding and Exceptions

JOHN BRYCE
Leading in IT Education
matrix company

- Must throw the same or less Exceptions
- May throw subclasses of the super method exceptions

```
public class A{
 public void methodA () throws RuntimeException{
        ...
 }
}
```

```
public class B extends A{
 public void methodA () throws NumberFormatException{
        ...
 }
}
```

Method Overriding and Exceptions

JOHN BRYCE
Leading in IT Education
matrix company

More examples

```
public class A{
 public void methodA () throws RuntimeException{
        ...
 }
}
```

```
public class B extends A{
 public void methodA () throws NumberFormatException, SecurityException{
        ...
 }
}
```

Method Overriding and Exceptions

JOHN BRYCE
Leading in IT Education
matrix company

More examples

```
public class A{
 public void methodA () throws RuntimeException, IOException{
        ...
 }
}
```

```
public class B extends A{
 public void methodA () throws EOFException{
        ...
 }
}
```

Creating Your Own Exceptions

JOHN BRYCE
Leading in IT Education
matrix company

- Class must be a subclass of Exception
- May hold more methods and fields

Exception – Constructors

• Exception()	Empty constructor
• Exception (String msg)	Exception with a message
• Exception (String msg, Throwable cause)	Exception with a message and a root cause
• Exception (Throwable cause)	Exception with a root cause

Exception – Main Methods

• getMessage()	Returns the Exception's message
• toString ()	Calls getMessage() method
• getCause()	Returns the root cause as Throwable

Creating Your Own Exceptions

JOHN BRYCE
Leading in IT Education
matrix company

Example

```
public class NumberOutOfLimitsException extends Exception{

        private int num=0;

 public NumberOutOfLimitsException (String msg, int num){
     super(msg);
     this.num=num;
 }

 public int getNum(){
    return num;
 }
}
```

Creating Your Own Exceptions

JOHN BRYCE
Leading in IT Education
matrix company

Example

```
public class NumChecker{

    public void check (int num) throws NumberOutOfLimitsException {
        if (num<0 || num>100)
            throw new NumberOutOfLimitsException ("Wrong value",num);
    }
```

```
public class TestChecker{

    public static void main (String[] args) {
        NumChecker nc=new NumChecker();
        try{
            nc.check(Integer.parseInt(args[0]));
            System.out.println(args[0]+" is OK");
        }catch (NumberOutOfLimitsException e){
            System.out.println(e.getMessage()+" "+e.getNum());
        }
    }
```

Syntax Enhancements- Java 7 & up

JOHN BRYCE
Leading in IT Education
matrix company

ARM – Automatic Resource Management

- Open/close resource connection is not part of the try-catch block
- Instead of:

```
public void doIO() throws IOException{
    FileInputStream in=null;
    try{
        in=new FileInputStream ("file");
        int data = in.read();
    }catch(FileNotFoundException e){
        in.close();
    }
}
```

- We use:

```
public void doIO() throws IOException{
    try(FileInputStream in= new FileInputStream ("file")){
        int data = in.read();
    }
}
```

- Forces the resource to be "Auto Closable"

Syntax Enhancements

JOHN BRYCE
Leading in IT Education
matrix company

ARM – Automatic Resource Management

- Closable.close() method throws IO exception
- In order to use ARM for other APIs as well – an AutoClosable super interface was created

AutoCloseable close() method throws a generat Exception

Closeable now extends it

```
public  interface AutoClosable {
            public void close () throws Exception;
}
```

```
public  interface Closable extends AutoClosable {
            public void close () throws IOException;
}
```

Syntax Enhancements

JOHN BRYCE
Leading in IT Education
matrix company

More on ARM

- Manages "AutoCloseable" implementations only(!)
- Whether try block pass or fails – close() will be invoked
- Can declare and use more than one resource:

```
try(FileInputStream in= new FileInputStream ("file1");
    FileOutputStream out= new FileOutputStream("file2") ){
        int data = in.read();
        out.write(data);
}
```

- Close() method is called according to resource declaration order in the try clause

Syntax Enhancements

JOHN BRYCE
Leading in IT Education
matrix company

Multi-catch

- Relating to different exceptions in a single catch block

Instead of:

```
try{
        FileInputStream in=new FileInputStream ("file");
        Connection con = DriverManager.getConnection(....);
        ....in.read();
        ....con.createStatement();
}catch(IOException e){
        ....
}catch(SQLException e){
        ....
}
```

We use:

```
try{
        .....
}catch(IOException | SQLException e){ .... }
```

Summary

JOHN BRYCE
Leading in IT Education
matrix company

- Exceptions type
  - Runtime
  - Exceptions
- Handling exceptions
  - Catching
  - Throwing
- Method override and exceptions
- Create exception
- Use ARM & Multicatch

Exceptions

References

- http://java.sun.com

- SUN Educational Services SL-275