


Java Exception

Wang Yang
wyang@njnet.edu.cn


- Last Chapter Review
- A Notion of Exception
- Java Exceptions
- Exception Handling
- How to Use Exception
- User-defined Exceptions

Last Chapter Review

Last Chapter Review

- java.io.*
- File : A Path in a file system
- I/O Manner
  - Text-based (char) / Data-based(byte)
  - Sequential / Random Access

Last Chapter Review

- Data-based IO
  - InputStream/OutputStream
  - FileInputStream/FileOutputStream
  - DataInputStream/DataOutputStream
- Text-based IO
  - Reader/Writer
  - FileReader/FileWriter
- RandomAccessFile

A Notion of Exception


Remember File Stream

```
public class IODemo {
   public void demo(String filename){
      File f = new File(filename);
      try {
         FileInputStream fis = new FileInputStream(f);
         int b = fis.read();
         fis.close();
      } catch (IOException e){
         System.out.println(e);
      }
   }
}
```

Remember File Stream

- filename object is not initialized
- File Exist : ?
- File Not Exist : ?
- File is a Directory : ?
- File has no Content

Remember File Stream

- filename object is not initialized: java.lang.NullPointerException
- File Not Exist : java.io.FileNotFoundException: FileNotExist.txt (No such file or directory)
- File is a Directory : java.io.FileNotFoundException: /usr (/usr Is a directory))
- File have no content: return -1
- File is exist and have content: OK

Another Example

```
public class ATMDemo {
   static Account[] users = new Account[20];

   public Account login(){
      System.out.println("Please input Account Name: ");
      Scanner s = new Scanner(System.in);
      int inputID = Integer.parseInt(s.nextLine());
      Account account = users[inputID];
      System.out.println("User " + inputID + "login");
      return account;
   }
}
```

Another Example

- Input 0~19 ?
- Input -1 ?
- Input 20 ?
- Input abcd ?

Another Example

- Input 0~19 : NullPointerException
- Input -1 : ArrayIndexOutOfBoundsException
- Input 20 : ArrayIndexOutOfBoundsException
- Input abcd : InputMismatchException

What we met is Exceptional Condition

- Exceptional Condition
  - No input or output file
  - Visit a Null reference
  - Access array out of bound
- A Sound Program Should
  - Declare the possible exceptional condition
  - Handle the exceptions at right time and in right place

Exception Case

- Normal Condition
- Not Normal, But you know it may happen, and you allow it happen
- Not Normal, and you don't allow it happen


```
public static void demo(String filename){
    if (filename==null){
        System.out.println("filename is null");
        return;
    }
    File f = new File(filename);
    try {
        if (!f.exists()){
            System.out.println("No Such File or Directory");
            return;
        }
        if(f.isDirectory()){
            System.out.println(filename + " is a Directory");
            return;
        }
        FileInputStream fis = new FileInputStream(f);
        int b = fis.read();
        while(b != -1){
            System.out.println(b);
            b = fis.read();
        }
        fis.close();
    } catch (IOException e){
        System.out.println(e);
    }
}
```

Too Much Code in main logic

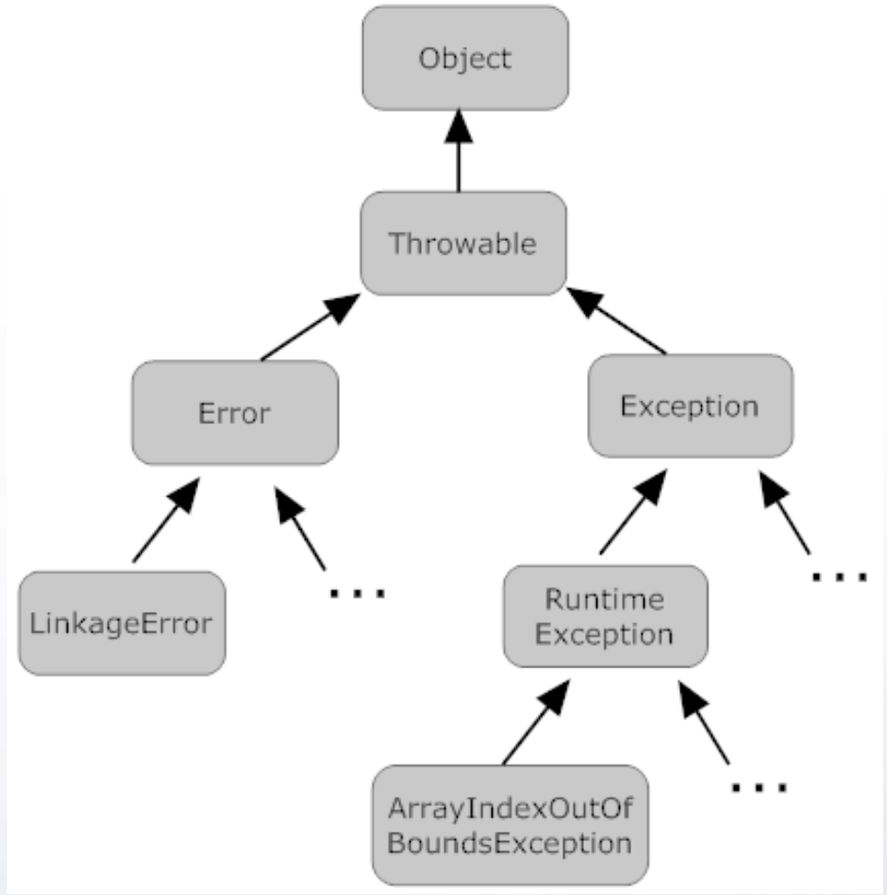
Java Exception

- Java Exception
- Offering a clear grammar for handling program correctness
- Balancing between Clarity and Correctness

Java Exception

What is Java Exception

- An Object
- A tree of classes
- Key word:
  - Throw
  - Exception
  - Try/Catch/Finally



Java Exceptions

Exception	Superclass of all exceptions
DataFormatException	Error data format
ClassNotFoundException	Exception in class loading
IOException	IO operation error
SQLException	Database operation error
TimeoutException	Timeout error
SocketException	Socket operation error
ArrayIndexOutOfBounsException	Exception in visiting array
NullPointerException	Visiting null reference

Example

```
public void demo(String filename){
      File file = new File(filename);
      try {
         FileInputStream fis = new FileInputStream(f);
         int b = fis.read();
         fis.close();
      } catch (IOException e){
         System.out.println(e);
      }
}
```

IOException e = new FileNotFindException (filename);
throw e;

Examples Con't

```
public void demo(String filename){
      File file = new File(filename);
      FileInputStream fis = null;
      try {
         FileInputStream fis = new FileInputStream(f);
         int b = fis.read();
      } catch (IOException e){
         System.out.println(e);
      } finally {
         try {
            if(fis != null){fis.close();}
         } catch (IOException e){
            System.out.println(e);
         }
      }
   }
```

File Exist

jump into finally

File not Exist

jump into catch

jump into finally

Exception Flow

- try, catch and finally
- try is used to monitor method invocation
- catch is used to catch thrown exceptions
- finally is used for execute essential code – whether there are exceptions or not
- Multiple catch clause
- At least one catch or finally clause

Examples : Multi Catch

```
public void test(File file){
      try{
         checkFile(file);
      } catch (IOException e){
         System.out.println(e.getMessage());
      } catch (IllegalArgumentException e){
         System.out.println("Please Provide a File");
      } catch (Exception e){
         System.out.println("Other Exception occurs");
      } finally {
         delete(file);
      }
}
```

Declare Exception

- throw and throws – Declare Possible Exceptions
- throw throws exceptions in method body
- throws defines Exception Specification

```
public void checkFile(File file) throws
IOException,IllegalArgumentException{
      if(!file.exists()){
         throw new IOException("File Don't exist!");
      } else  if(file.isDirectory()){
         throw new IllegalArgumentException("File is a Directory");
      } else {
         ……
         throw new Exception("Blabla");
      }
   }
```

‘s Family

- ‘s family
  - extends
  - implements
  - throws
  - all with declaration

Exception in Overriding

- Exception in Overriding
- Which is allowed?

```
public class A {
   public void test() throws IOException, IllegalArgumentException{
   }
}
public class B extends A{
   public void test() throws IOException{
   }
}
public class C extends A{
   public void test() throws Exception{
   }
}
```

```
public class A {
    public void test() throws IOException, IllegalArgumentException{
    }
}
public class B extends A{
    public void test() throws IOException{
    }
}
public class C extends A{
    public void test() throws Exception{
    }
}
```

Right

Wrong

Examples

- Exception in Overriding
- Which is allowed?
- Remember the compatibility
  - the return type is reduced
  - the access control is enlarged
  - the exception type is reduced

Exception Handling

Catch and Handling of Exceptions

```
pre();
try {
        // actions;
} catch (ExceptionType1  e) {
} catch (ExceptionType2 e) {
}  finally {
        post();
}
```

Catch and Handling of Exceptions

```
Object val = null;
try {
        val = pre();
        // actions;
} catch (ExceptionType1  e) {
} catch (ExceptionType2 e) {
}  finally {
        post();
}
```

Rethrow Exceptions

```
public void function () throws Exception1, Exception2 {
        pre();
        try {
            // actions;
        }  finally {
            post();
        }
    }
```

About Return Value

```
public Type function () throws Exception2 {
        Type ret = default value // like false or true for boolean,
        pre();
        try {
            // actions;
            ret = success value // like true
        }  catch (Exception1 e){
            ret = failure value // like false
        } finally {
            post();
            return ret;
        }
}
```

Define Exception

- Message
  - new Exception(String message)
  - getMessage()
- Cause
  - initCause()
  - new Exception(Exception cause)
  - getCause()
- StackTrace
  - printStackTrace()

User Defined Exception

```
public class BadObjectException extends Exception {
   private Object badObj;
   public BadObjectException(Object object, String msg){
      super(msg);
      this.badObj = object;
   }

   public Object getBadObj(){
      return badObj;
   }
}
```

Runtime Exception

RuntimeException is the superclass of those exceptions that can be thrown during the normal operation of the Java Virtual Machine.

- Throwable Exceptions in Normal Run-time
- Difference: DO NOT HAVE TO Declare
- Example:
  - NullPointerException
  - ArithmeticException
  - BufferOverflowException

It's HARM to run program continuously

How to Use Exception


Exception Idioms

- Use exceptions only for exceptional conditions
- Use checked exceptions for recoverable conditions and runtime exceptions for programming errors
- Avoid unnecessary use of checked exceptions
- Favor the use of standard Exceptions
- Throw exceptions appropriate to the abstraction
- Document all exceptions thrown by each method
- Include failure-capture information in detail messages
- Strive for failure atomicity
- Don’t ignore exceptions

Avoid unnecessary use of checked exceptions

- Checked Exception : the program maybe recovered, not in normal control logic
- Runtime Exception: the program has bugs and should be avoid
- Error: the vm/program have fatal errors, the program should be terminated immediately
- Normal Condition: should be in normal control logic

Exception Conditions

- Exceptional Condition VS. Normal Problems
  - End of file is a normal problem
  - Normal problem does not lead to program halt
  - Exceptional conditions lead to program halt
- Exceptional Condition VS. Error Condition
  - JVM crash is an error condition
  - Error conditions cannot be handled by program
  - Exceptional conditions can and should be handled by program

Wrong Exception Usage

```
public void demo(String filename){
      File file = new File(filename);
      FileInputStream fis = null;
      try {
         FileInputStream fis = new FileInputStream(f);
         int b = fis.read();
         while(true) {
            b = fis.read();
         }
      } catch (EOFException e){
         System.out.println(e);
      } finally {
         try {
            if(fis != null){fis.close();}
         } catch (IOException e){
            System.out.println(e);
         }
      }
}
```

Wrong Usage


Favor the use of standard Exceptions

- IllegalArgumentException
- IllegalStateException
- UnsupportOperationException
- ConcurrentModificationException

Throw exceptions appropriate to the abstraction

- Exception Message should reflect the semantic of the method
- Exception Chain Handling

```
public void deposit( int id, int moneyVal){
      users[id].money += moneyVal;
   }
public void menu (){
      ……
      deposit(-1,200);
      ……
   }
```

ArrayIndexOutofBound

IllegalArgument

Strive for failure atomicity

- Object state should not change when Exceptions occur
- But How?

```
public void AddUser( Account account){
        ATMDemo.userCount += 1;
        users[ATMDemo.userCount] = account
}
```

- Change the process logic
- verify the params before invoke the method
- recovery code
- backup data

Self-Study

- Assert
- Exception Chaining