Chapter 3 Java Exception

javaseu@163.com

Content

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

Exceptional Condition

- Exceptional Condition
  - Divided by 0
  - No input or output file
  - Visit a Null reference
- A Sound Program Should
  - Declare the possible exceptional condition
  - Handle the exceptions at right time and in right place

Exceptional Condition

- Exceptional Condition VS. Normal Condition
  - End of file is a normal condition
  - Normal condition 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

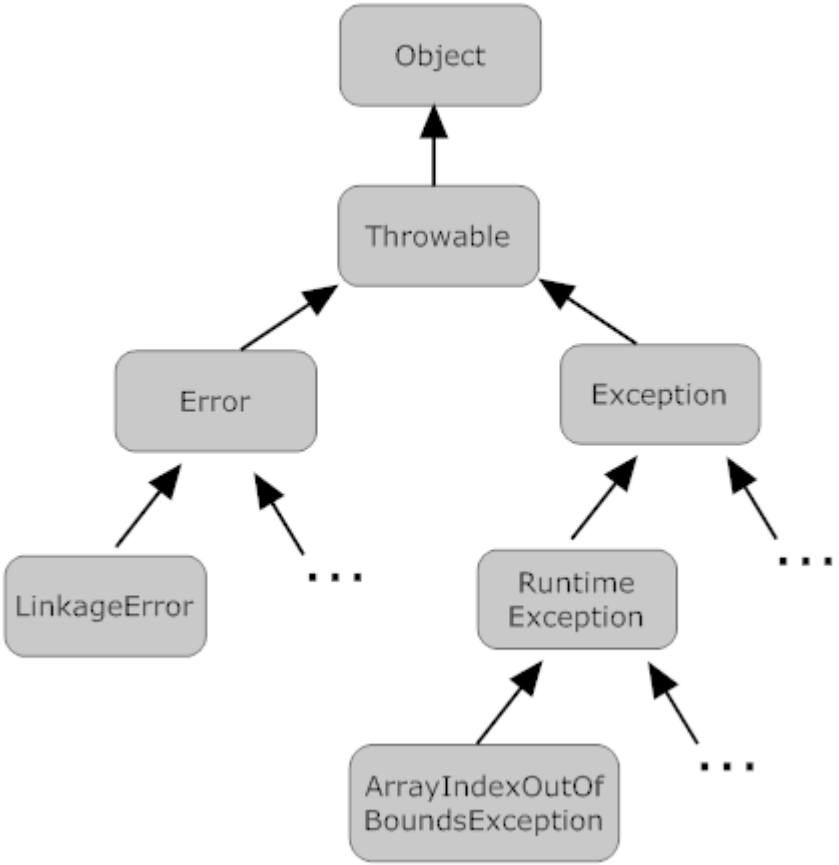
Significance of Exception

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

```
public void getShape(Person p){
    if(p==null){
        ...
    }else if(p.weight==0){
        ...
    }else{
        double ratio = p.height / p.weight;
        // Calculate and print the body shape of Person p
    }
}

public void getShape(Person p)
    throws NullPointerException, ArithmeticException{
    double ratio = p.height / p.weight;
    // Calculate and print the body shape of Person p
}
```

Java Exception



Runtime Exception

- Throwable Exceptions in Normal Run-time
- Related to Execution of Program
- Not Related to the Business Logic
- Difference: DO NOT HAVE TO Declare
- Example:
  - NullPointerException
  - ArithmeticException

Java Exception Handling

- Java Use Following Exception Handling Process:
  - Block A meets an exceptional condition
  - Block A halt
  - One or more exception objects generated
  - Exception objects are thrown
  - JVM looks for proper codes to catch these objects
  - Exceptional condition is handled
  - Program continues from exceptional handling code

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

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 doesn't exist!");
        }else if(file.isDirectory()){
            throw new IllegalArgumentException("Not a file!");
        }
}
```

Catch and Handling of Exceptions

- 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

```
public void test(File file){
        try{
            this.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 exceptions occur");
        }finally{
            file.delete();
        }
}
```

Deep Into FINALLY

The finally block always executes when the try block exits. This ensures that the finally block is executed even if an unexpected exception occurs. But finally is useful for more than just exception handling — it allows the programmer to avoid having cleanup code accidentally bypassed by a return, continue, or break. Putting cleanup code in a finally block is always a good practice, even when no exceptions are anticipated.

Note: If the JVM exits while the try or catch code is being executed, then the finally block may not execute. Likewise, if the thread executing the try or catch code is interrupted or killed, the finally block may not execute even though the application as a whole continues.

- No catch
- Return in catch

Deep Into FINALLY

- Guess the result:

```
public void test(){
      try{
           System.out.println("try block");
           return;
          }finally{
           System.out.println("finally block");
          }
     }
}
```

Deep Into FINALLY

- Guess the result:

```
public int test(){
    try{
        System.out.println("try block");
        int i = 1 / 0;
        return 1;
    }
    catch(Exception e){
        System.out.println("catch block");
        return 2;
    }
    finally{
        System.out.println("finally block");
    }
}
```

Deep Into FINALLY

- http://www.ibm.com/developerworks/cn/java/j-lo-finally/

Rethrow Exceptions


```
public void test(File file)throws IOException{
        try{
            this.checkFile(file);
        }catch(IOException e){
            throw e;
        }catch(IllegalArgumentException e){
            throw e;
        }catch(Exception e){
            System.out.println("Other exceptions occur");
        }finally{
            file.delete();
        }
}
```

Is there something wrong?

Exception Handling

- 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 badObject;

    public BadObjectException(Object object, String msg){
        super(msg);
        this.badObject = object;
    }

    public Object getBadObject(){
        return this.badObject;
    }
}
```

Lab Work

- hashcode() + cloneable + exception

```
Person p1 = new Person("tom", 18);
Person p2 = new Person("tom", 18);
Person p3 = new Person("jack", 18);
Person p4 = p1;
Person p5 = null;
try{
    p5 = (Person)p1.clone();
}catch(Exception e){
    e.printStackTrace();
}
Person source = p1;
Person target = p3;
try{
    if(source.hashCode()!=target.hashCode()) throw(new BadObjectException(target));
    else System.out.println("Same Hashcode.");
}catch(BadObjectException e){
    System.out.println("The Souce Hashcode: " + source.hashCode());
    System.out.println("Bad Object Hashcode: " + ((Person)e.getBadObject()).hashCode());
}
```

We want to let the hashcode of p1 = hashcode of p2 p4 p5, but different with p3, how to write the code?

Think

- Exception in Overriding
- Which is allowed?

```
public class A{
    public void test() throws IOException, SQLException{
    ...
    }
}

public class B extends A{
    public void test() throws Exception{
    ...
    }
}

public class C extends A{
    public void test() throws SQLException{
    ...
    }
}
```

Self-study

- Assertions(断言)
  - Used for Software Testing

```
i++;
assert i < max;
```

- Regular Expression(正则表达式)
  - java.util.regex

```
Pattern p = Pattern.compile("a*b");
Matcher m = p.matcher("aaaaab");
boolean b = m.matches();
```

Forecast

- Java I/O Introduction
- File and Directory
- Byte-stream and Character-stream
- Bridge between b-s and c-s
- Random Access File
- Standard I/O
  - System.in
  - System.out
- java.nio Pilot