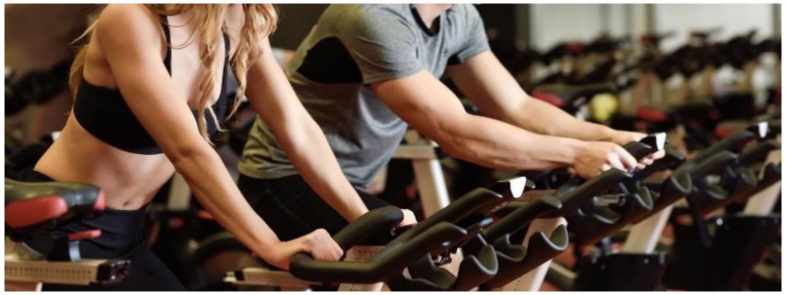
IECS273/274 [1111 3670/3671]
物件導向設計與實習

08#1 Exception in Java



01 Java Program Exceptions

Java Program Exceptions

- An **exception** refers to an **unexpected event** that occurs when a *Java Virtual Machine (JVM)* executes a Java program.
- Examples of Java program exceptions:
  - Divisor 0 in a division operation, e.g., a = b; x = 100 / (a - b);
  - Array subscript out of range, e.g., array A is declared as **int** A[10], program code is m = n + 1; A[m-n] = …; or **for** (i=0; i<10; i++) A[i*2] = …;
  - The program cannot read a file due to inappropriate permissions or wrong filename, e.g., use the following statement to read the file abc.txt, InputStream ins=**new** FileInputStream("abc.text"); filename error is detected during execution time.
  - The program wants to read an integer, but input a string of numbers and letters during execution, e.g.,, the program code is a = in.nextInt();, but the input is 1oo.
  - Addition/subtraction of two matrices with different size, e.g.,

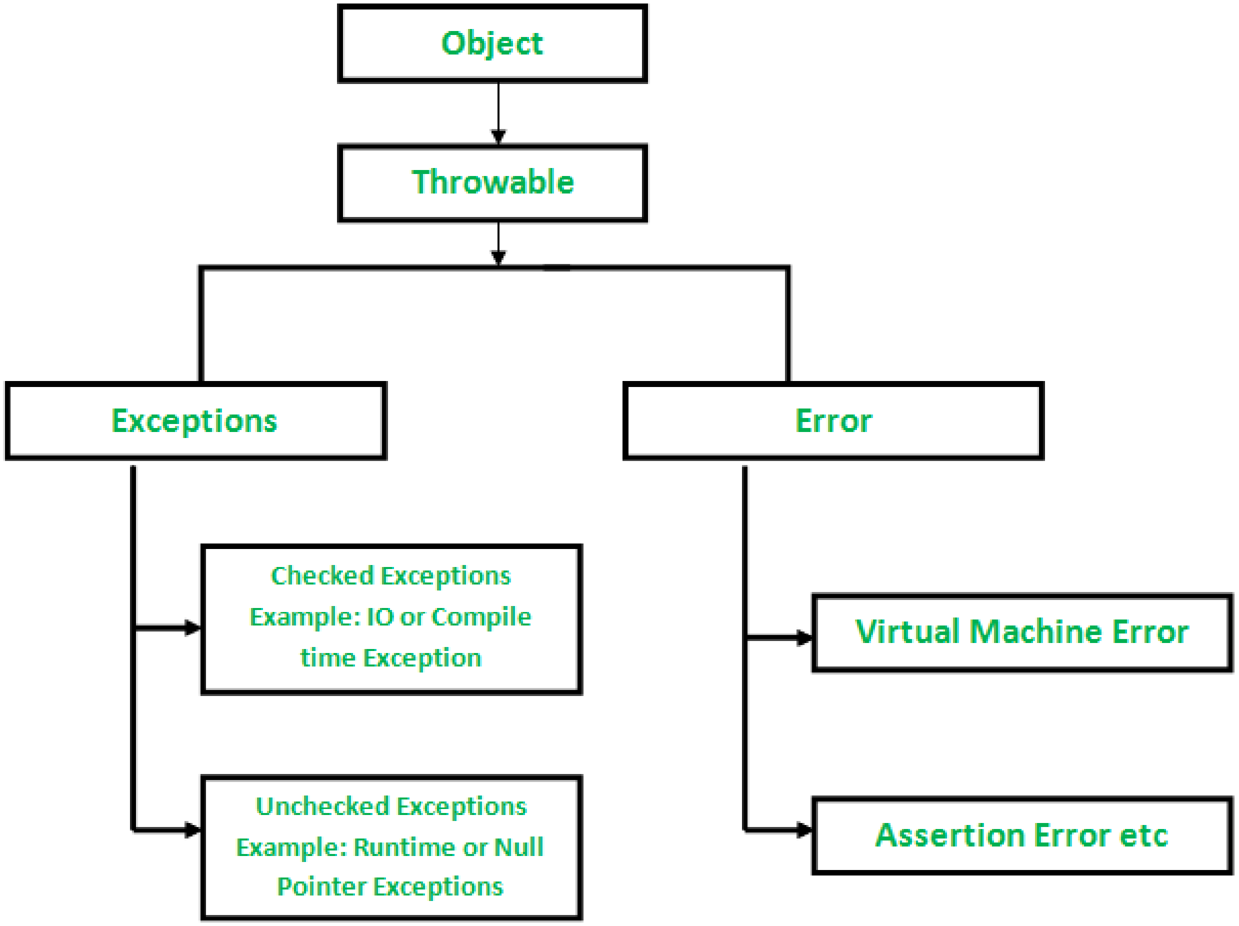
```
Matrix a = new Matrix(3, 5);
Matrix b = new Matrix(5, 3);
Matrix c = a.add(b);
```

Exceptions vs. Errors in Java Programs

- Exceptions vs. Errors
  - Both are *abnormal behavior* that occurs in *program execution*.
  - Errors means serious problems that *cannot be resolved or recovered* by programs
    - ✓ Program execution errors must cause execution to *terminate*.
    - ✓ For example, virtual machine error, logical assertion error.
  - Exceptions are *unwanted* or *unexpected* events.
    - ✓ Program execution *can* **try** and **catch** exception events. After catching, it can make appropriate remedies to *handle exceptions*.
    - ✓ If the exception is not handled, the program execution will *terminate.*
    - ✓ For example, *calling a method with* **null** *value* or *divided by* **0**.

Exception Hierarchy in Java

- The top-level class of the Java programming language is called **Object**, and all exceptions and errors are subclasses of the **class Throwable**.


Object
Throwable
Exceptions
Error
Checked Exceptions Example: IO or Compile time Exception
Unchecked Exceptions Example: Runtime or Null Pointer Exceptions
Virtual Machine Error
Assertion Error etc

Materials from： https://www.geeksforgeeks.org/exceptions-in-java/

Exception Handling in JVM

- How the Java Virtual Machine Handles Exceptions?
  - When an exception occurs inside a method, the method will create an object called an **exception object** and hand it over to *the run-time system* (*JVM*).
  - The exception object contains the *name* and *description* of the exception, and the *current state* of the program where the exception occurred.
  - After the exception object is created, its handler is "*thrown*" to the execution system, called **throwing an exception**.
  - An exception may occur through a *series* of method calls, which are placed on the *call stack* in the order in which they were called.
  - The execution system searchs of the call stack for a *block of code that can handle an exception* that occurs. This block of code is called an **exception handler**。
  - The execution system starts searching from the method where the exception occurred and *walks the call stack in reverse order* of calling methods.
  - If the execution system searches all methods on the call stack and cannot find an appropriate handler, the runtime system hands over the exception object to the execution system's *default exception handler*.
  - This handler will print the following exception information and *abort the program*.

```
Exception in thread "xxx" Name of Exception : Description
... ...... ..  // Call Stack
```

Exception1.java

```
public class Exception1 {

  public static void main(String[] args) {
    String str = null;
    System.out.println(str.length());
  }
}
```

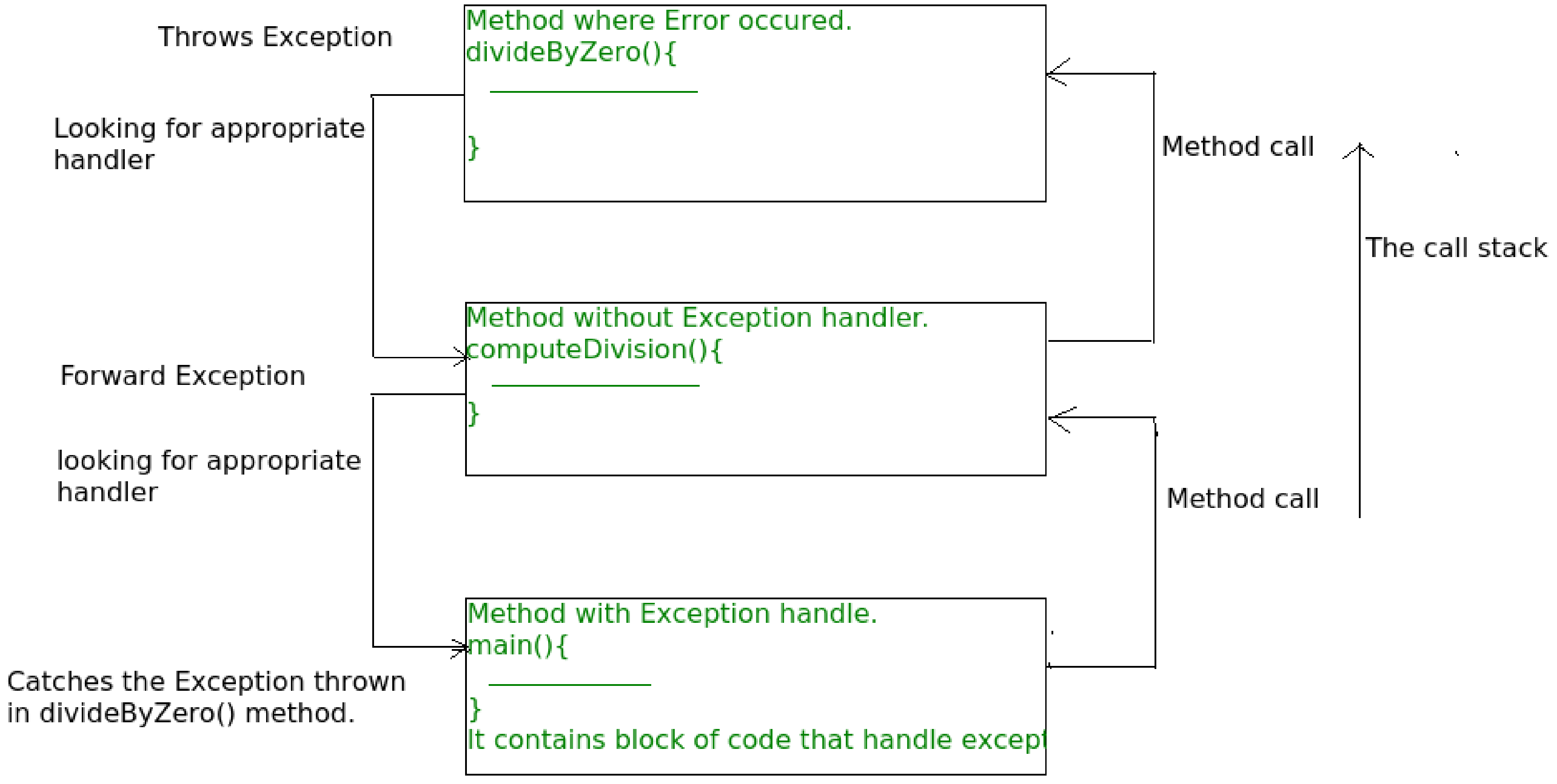
str is an empty string pointer
Call the string length method str.length()

Exception1.class

```
Exception in thread "main" java.lang.NullPointerException
        at Exception1.main(Exception1.java:5)
```

Exception message during execution,
no exception handling,
Program execution terminates.

Exception Handling in JVM


Throws Exception
Looking for appropriate handler
Forward Exception
looking for appropriate handler
Catches the Exception thrown in divideByZero() method.
Method where Error occured.
divideByZero(){
}
Method without Exception handler.
computeDivision(){
}
Method with Exception handle.
main(){
}
It contains block of code that handle except
Method call
Method call
The call stack

The call stack and searching the call stack for exception handler.

Exception2.java

```
public class Exception2 {

  static int divideByZero(int a, int b) {
    int i = a / b;

    return i;
  }


  static int computeDivision(int a, int b) {
    int result = 0;

    try {
      result = divideByZero(a, b);
    }
    catch (NumberFormatException ex) {
      System.out.println("NumberFormatException is occured.");
    }
    return result;
  }
```

Call this method (a==5, b==0).
An exception will occur.

The execution system passes the exception to this calling method

The execution system did not find an **ArithmeticException**, Exception inconsistent!
Pass the exception to the previous calling method.

Exception2.java

```
public static void main(String[] args) {
   int m = 5;
   int n = 0;

   try {
      int i = computeDivision(m, n);
   }
   catch (ArithmeticException ex) {
      System.out.println(ex.getMessage());
   }
}
}
```

The execution system passes the exception to this calling method

The execution system finds a handler that matches the exception **ArithmeticException**;
Execute this handler.

Exception2.class

```
/ by zero
```

02

Raising and Handling Exceptions in Java

Raising and Handling Exceptions in Java

- The **raising** and **handling of exceptions** in Java uses five keywords: **try, catch, throw, throws, finally.**
  - If one or more program statements could *throw* an exception, place those statements in a **try** *block statement*.
  - If an exception occurs in a try block, the exception is *thrown*.
  - Once an exception is *thrown*, program code can use a **catch** *block statement* to catch the exception and handle it in some reasonable way.
    - ✓ System-generated exceptions are *automatically thrown* by the execution system.
    - ✓ To throw an exception *manually*, use the keyword **throw**.
    - ✓ Any exception *thrown from a method* must be specified by the **throws** clause.
    - ✓ Any code that must be executed after *completion of a try block* is placed in a **finally** *block statement*.

Raising and Handling Exceptions in Java

<try statement>::= **try***<block>* *<catches>* |

try *<block>* *<catches>*? *<finally>*

<catches> ::=*<catch clause>* |

<catches> *<catch clause>*

<catch clause>::= **catch (** *<formal parameter>* **)** *<block>*

<finally > ::= **finally** *<block>*

Raising and Handling Exceptions in Java

- try-catch-finally combination
  (control flow in try-catch clause or try-catch-finally clause)
  - Exceptions occur in **try** blocks and are handled in **catch** blocks:
    - ✓ If a statement within a **try** block throws an exception, the rest of the **try** block does not execute; control passes to the corresponding **catch** block.
    - ✓ After the **catch** block is executed, control will be transferred to the **finally** block (if present), then the rest of the program will be executed. 3
  - The exception occurred in the **try** block, but was not handled by the **catch** block:
    - ✓ In this case, the default handling mechanism will be followed. 4
    - ✓ If the code has a **finally** block, it will be executed, followed by the default processing mechanism. 5
  - The exception did not occur during the execution of the **try** block:
    - ✓ In this case, the **catch** blocks are not executed because they only run when an exception occurs.
    - ✓ If the code has a **finally** block, execute this code; then execute the rest of the program. 6

Exception3.java

```
public class Exception3 {

  public static void main(String[] args) {
    int[] arr = new int[4];
```

try block

```
    try {
      int i = arr[4];
      System.out.println("Inside try block.");
    }
```

An exception occurs when the array s ubscript is out of range.

This statement will not be executed after the exception.

catch block

```
    catch (ArrayIndexOutOfBoundsException ex) {
      System.out.println("Exception caught in catch block.");
    }
```

finally block

```
    finally {
      System.out.println("Finally block is executed.");
    }
```

rest of the code executed at the end

```
    System.out.println("Outside try-catch-finally clause.");
  }
}
```

Exception3.class

```
Exception caught in catch block.
Finally block is executed.
Outside try-catch-finally clause.
```

Exception4.java

```java
public class Exception4 {

  public static void main(String[] args) {
    int[] arr = new int[4];
```

try block

```java
    try {
      int i = arr[4];
      System.out.println("Inside try block.")
    }
```

An exception occurs when the array s ubscript is out of range.
This statement will not be executed after the exception.

catch block

```java
    catch (NullPointerException ex) {
      System.out.println("Exception caught in catch block.");
    }
```

Unmatched exception.
This statement will not be executed.

The rest of the code will not be executed

```java
    System.out.println("Outside try-catch clause.");
  }
}
```

Exception4.class

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 4 out of bounds for length 4
        at Exception4.main(Exception4.java:10)
```

Exception5.java

```java
public class Exception5 {

  public static void main(String[] args) {
    int[] arr = new int[4];
```

try block

```java
    try {
      int i = arr[4];
      System.out.println("Inside try block.");
    }
```

An exception occurs when the array s
ubscript is out of range.
This statement will not be executed
after the exception.

catch block

```java
    catch (NullPointerException ex) {
      System.out.println("Exception caught in catch block.");
    }
```

Unmatched exception.
This statement will not be executed.

finally block

```java
    finally { // Finally block statement.
      System.out.println("Finally block is executed");
     }
```

The rest of the code will not be executed

```java
    System.out.println("Outside try-catch clause.");
  }
}
```

Exception5.class

```
Finally block is executed.
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 4 out of bounds for length 4
        at Exception5.main(Exception5.java:10)
```

Output of the **finally** clause.

Exception6.java

```
public class Exception6 {

  public static void main(String[] args) {
```

try block

```
    try {
       String str = "123";
       int num = Integer.parseInt(str);

       System.out.println("Inside try block.");
    }
```

No exception, "Inside try block" will be printed.

catch block

```
    catch (NumberFormatException ex) {
       System.out.println("catch block executed.");
    }
```

No exception. This statement will not be executed.

finally block

```
    finally { // Finally block statement.
     System.out.println("Finally block is executed.");
    }
```

This statement will be executed with or without exception.

The rest of the code will not be executed

```
    System.out.println("Outside try-catch-finally clause.");
}
```

Exception6.class

```
Inside try block.
Finally block is executed.
Outside try-catch-finally clause.
```

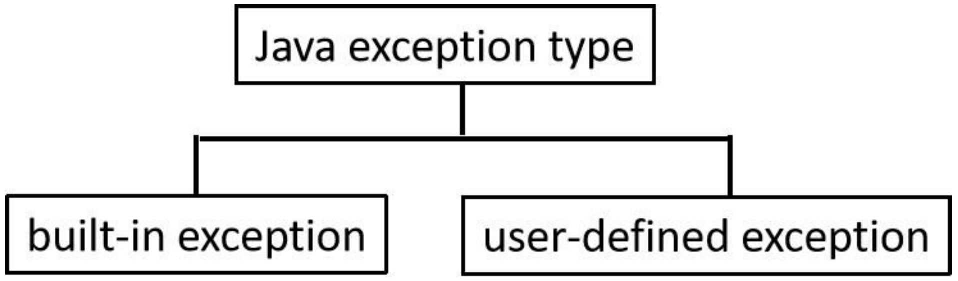
Output of the **catch** clause.
Output of the **finally** clause.
Output of the rest of the code.

03

Exception Types in Java

Exception Types in Java

- Java defines several types of exceptions that are associated with its various class libraries. Java also allows users to define their own exceptions.



- **Built-in exceptions** are exceptions available in Java libraries. These exceptions are suitable for explaining certain error conditions.
- Sometimes built-in exceptions in Java cannot describe a specific situation. In this case, the user can create their own exceptions, known as "**user-defined exceptions**".

Built-in Exceptions in Java

- Following is a list of important built-in exceptions in Java:
  - **ArithmeticException**

    This exception is thrown when an exception condition occurs in an arithmetic operation.
  - **ArrayIndexOutOfBoundsException**

    This exception is thrown when an illegal array subscript is used, the subscript is negative or greater than or equal to the size of the array.
  - **ClassNotFoundException**

    This exception is thrown when a program tries to access a class whose definition was not found.
  - **FileNotFoundException**

    This exception is thrown when the file is inaccessible or not opened.
  - **IOException**

    This exception is thrown when an I/O operation fails or is interrupted.
  - **InterruptedException**

    This exception is thrown when a thread is interrupted while waiting, sleeping, or doing some processing.

Built-in Exceptions in Java

- **NoSuchFieldException**

  This exception is thrown when the class does not contain the specified field (or variable).

- **NoSuchMethodException**

  This exception is thrown when a method not found is accessed.

- **NullPointerException**

  This exception is thrown when referencing a member of a null object; **NULL** does not represent any object.

- **NumberFormatException**

  This exception is thrown when the method cannot convert the string to numeric format.

- **RuntimeException**

  This indicates any exceptions that occurred while the system was executing.

- **StringIndexOutOfBoundsException**

  It is raised by **String** class methods to indicate that the subscript ratio exceeds the string size or is negative.

Exception7.java

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;

public class Exception7 {
   public static void main(String[] args) {
      try {
            int a = 30, b = 0;
            int c = a/b;  // Cannot be divided by zero.
            System.out.println ("Result = " + c);
      }
      catch (ArithmeticException e) {
            System.out.println ("ArithmeticException: Cannot divide a number by 0.");
      }

      try {
            String a = null; // Null value.
            System.out.println(a.charAt(0));
      }
      catch (NullPointerException e) {
            System.out.println("NullPointerException: String is null.");
```

File processing package.

Exception7.java

```
    }

    try {
        String a = "Minjiang University"; // Length is 19.
        char c = a.charAt(19); // Accessing 20th element.
        System.out.println(c);
    }
    catch (StringIndexOutOfBoundsException e) {
        System.out.println("StringIndexOutOfBoundsException: String index out of bound.");
    }

    try {
        File file = new File("D://file.txt"); // This file does not exist.
        FileReader fr = new FileReader(file);
    }
    catch (FileNotFoundException e) {
        System.out.println("FileNotFoundException: File does not exist.");
    }

    try {
        int num = Integer.parseInt ("lol") ;  // "lol" is not a number.
```

Exception7.java

```
            System.out.println(num);
        }
        catch (NumberFormatException e) {
            System.out.println("NumberFormatException: lol is not 101.");
        }

        try {
            int arr[] = new int[5];
            arr[6] = 9; // Accessing 7th element in an array of size 5.
        }
        catch (ArrayIndexOutOfBoundsException e){
            System.out.println ("ArrayIndexOutOfBoundsException: Array index is out of bound.");
        }
    }
}
```

Exception7.class

```
ArithmeticException: Cannot divide a number by 0.
NullPointerException: String is null.
StringIndexOutOfBoundsException: String index out of bound.
FileNotFoundException: File does not exist.
NumberFormatException: lol is not 101.
ArrayIndexOutOfBoundsException: Array index is out of bound.
```

User-defined Exceptions in Java

- There are many specific situations that cannot be described by built-in exceptions in Java; in such situations, users can also create exceptions called **user-defined exceptions**.
- Following are the steps to *define* and *create* a user-defined exception:
  - Because all exceptions are subclasses of the **Exception** class, first define a specific exception class as a subclass of the exception class. Example:

    ```
    public class ScoreException extends Exception {…}
    ```

  - Default constructors can be defined for specific exception classes. Example:

    ```
    ScoreException () { }
    ```

  - Define a constructor with parameters, a program can also *call the constructor of the superclass* and *pass the string parameters in the past*. Example:

    ```
    ScoreException(String str) { super(str); }
    ```

  - To *throw* a user-defined exception, we need to create an *object* of its exception class and then *throw* it using the **throw** clause. Example:

    ```
    ScoreException sEx = new ScoreException(“Must be between 0 and 100.”);
    throw sEx;
    ```

ScoreException.java

```
public class ScoreException extends Exception { // A user defined exception.

  public ScoreException() { }; // Default constructor.

  public ScoreException(String str) { super(str); } // A constructor with a message.
}
```

user-defined exception

default constructor

parameter constructor

Exception8.java

```
public class Exception8 {

  private static String name[] = {"John", "Peter", "Susan", "David", "Mary"};
  private static int oo[] = {85, 52, 110, 70, 82};
  private static int db[] = {72, 68, 80, 90, 88};

  public static void main(String[] args) {
    int i;
    boolean exceptionOccurred = false;

    System.out.println("Name\tComputer Programming\tData Structure");
    for (i=0; i<5; i++) {
      try {
        if (oo[i]>=0 & oo[i]<=100 & db[i]>=0 & db[i]<=100) {
          System.out.println(name[i] + "\t  " + oo[i] + "\t\t\t  " + db[i]);
          exceptionOccurred =  false;
        }
        else {
          ScoreException sEx = new ScoreException(name[i] + ", " + oo[i] +
                                          ", " + db[i] + ": score must be between 0 and 100.");
          throw sEx;
```

try block

creat and **throw**
user-defined exeception

Exception8.java

```
            }
        }
        catch (ScoreException sEx) {
            exceptionOccurred = true;
            sEx.printStackTrace();
        }
        finally {
            if (exceptionOccurred) System.out.println("**** An invalid score has been detected.");
        }
      }
   }
}
```

try block

catch block

finally block

Exception8.class

```
Name        Computer Programming                Data Structure
John          85                                  72
Peter         52                                  68
exception8.ScoreException: Susan, 110, 80: score must be between 0 and 100.
            at exception8.Exception8.main(Exception8.java:21)
**** An inalid score has been detected.
David         70                                  90
Mary          82                                  88
```

04

Exception Propagation in Java

Exception Propagation in Java

- **Exception propagation**: first throw an exception *from the top of the call stack*; if **uncaught**, it will try to call a method below the stack until the call stack becomes empty.
- Two types Java exceptions: **checked** and **unchecked**.
  - **Checked exceptions**: exceptions that are **caught** at *compile time*.
    - If some code in a method throws an exception, the method ***must*** handle the exception, or the exception must be propagated using the **throws** keyword.
    - User-defined exceptions in Java are all ***checked*** exceptions.
    - If, a method with a *checked exception*, does not **catch** nor *propagate* (**throws**) the exception, the compiler will generate a compilation error.
  - **Unchecked exception**: an exception that is not checked at compile time.
    - An exception that the *compiler* does **not** *force or require to propagate*; that is, at *execution time*, if the exception is not caught, it will **automatically propagate**.
    - Java's built-in exceptions are unchecked exceptions.

ScoreException.java

```
public class ScoreException extends Exception { // A user defined exception.

  public ScoreException() { }; // Default constructor.

  public ScoreException(String str) { super(str); } // A constructor with a message.
}
```

user-defined exception

default constructor

paremeter constructor

Exception9.java

```
public class Exception9 {

   private static String name[] = {"John", "Peter", "Susan", "David", "Mary"};
   private static int oo[] = {85, 52, 110, 70, 82};
   private static int db[] = {72, 68, 80, 90, 88};

   public static void checkScore(String name, String course, int score) throws ScoreException {
      if (score<0 || score>100) {
         ScoreException sEx = new ScoreException(name + ", " + course + " " +
               score + ": score must be between 0 and 100.");
         throw sEx;
      }
   }

   public static void checkRecord(String name, int ooScore, int dbScore) throws ScoreException {
      checkScore(name, "Computer Programming", ooScore);
      checkScore(name, "Data Structure", dbScore);
   }

   public static void main(String[] args) {
      int i;
```

throws
propagate exception

try block

create exception object sEx,
throw user-defined execption

throws
propagate exception

Call the method checkScore()

Exception9.java

```
      boolean exceptionOccurred = false;

      System.out.println("Name\tComputer Programming\tData Structure");
      for (i=0; i<5; i++) {
        try {
          checkRecord(name[i], oo[i], db[i]);
          System.out.println(name[i] + "\t  " + oo[i] + "\t\t  " + db[i]);
          exceptionOccurred = false;
        }
        catch (ScoreException sEx) {
          exceptionOccurred = true;
          sEx.printStackTrace();
        }
        finally {
          if (exceptionOccurred) System.out.println("**** An invalid score has been detected.");
        }
      }
    }
  }
```

try-catch-finally in the for loop,
Execute once per loop.

try block

catch block

finally block

Exception9.class

```
Name        Computer Programming                  Data Structure
John          85                                    72
Peter         52                                    68
exception9.ScoreException: Susan, Computer Programming 110: score must be between 0 and 100.
            at exception9.Exception9.checkScore(Exception9.java:11)
            at exception9.Exception9.checkRecord(Exception9.java:18)
            at exception9.Exception9.main(Exception9.java:28)
**** An invalid score has been detected.
David         70                                    90
Mary          82                                    88
```

Exception handling output message

Exception propagation, calling three methods

finally block, output message

05 ▶Programming Practice

Practice 1：Matrix Interface & Exception

An m×n matrix is a two-dimensional structure with m rows and n columns. Matrices have two special structures, vector and square matrix. The vector can be divided into a 1×n row vector and an n×1 column vector; a square matrix is an n×n matrix. The basic operations on two matrices are addition, subtraction, and multiplication. To add (A+B) and subtract (A-B) two matrices A and B, the two matrices must have the same number of rows and columns. To multiply two matrices (A×B), the number of columns of A must be equal to the number of rows of B.

Use the Java programming language to define an **interface** for a two-dimensional structure, this interface has four abstract methods addition, subtraction, multiplication, and matrix transposition; and use object-oriented class **inheritance** to define a concrete class Matrix (matrices) and three sub-classes VectorRow (row vectors), VectorCol (column vectors), and SMatrix (square matrix). Assume that the value of each matrix element is a positive floating point number less than 1.0, and use a random number generator to generate the values of the matrix elements.

Practice 1：Matrix Interface & Exception

The **inner product** of two row vectors or two column vectors of equal length is defined as the products of two pairwise vector elements and then added together. Let U and V be two row or column vectors of length n, the inner product of U and V is $U \bullet V = \sum_{\mathrm{i}-0}^{n-1} U_i \times V_i$. For an *n*×*n* matrix A, calculate its **determinant** |A|. The recursive definition of the determinant, expanded by row *i*, is:

$$|A| = \begin{cases} a_{0,0} & \text{if } n = 1 \\ \sum_{j=0}^{n-1} (-1)^{i+j} a_{i,j} \times |cofactor(A, i, j)| & \text{if } n > 1\text{'} \end{cases}$$

cofactor(A,i,j) is the (n-1)×(n-1) square matrix after removing the i-th row and j-th column of A.

At the same time, define a **matrix exception**, MatrixException, if the operand(s) of addition, subtraction, multiplication, matrix transposition, vector inner product, and square matrix determinant do not meet the requirements, then raise, propagate, and handle matrix exceptions.

Practice 1：Matrix Interface & Exception

In the application program, perform the following steps, and process any exception :

① Declare and create three matrices A, B, and C of size 6×4, 6×4, and 4×6, respectively. Test expressions A+B, A-B, A×C, C×A, $(B-A)^T$, and C-A. Print the matrices A, B, and C and the resulting matrices of the above expressions.

② Declare row vector vR. Set each of row 0 to row 6 of matrix A to vR and print each row.

③ Declare row vector vC. Set each of column 0 to column 4 of matrix A to vC and print each column.

④ Compute and print $A_{*,0} \bullet B_{*,0}$, $A_{*,0} \times C_{0,*}$, and $A_{*,1} \bullet C_{*,1}$.

⑤ Declare and create square matrix S. Compute and print |S|, |B×C|, and |A|.