Exceptions

(http://docs.oracle.com/javase/tutorial/essential/exceptions/index.html)

Objectives

- Exception
- Kinds of Exception
- Exception Handling
- Examples

Exception

- **Exception** is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions.
- For example:
  - a user might type an invalid filename;
  - An accessed file does not exist of might contain corrupted data;
  - a network link could fail;
  - …
- When an error occurs within a method, the method creates an **exception object** and hands it off to the runtime system
- Creating an exception object and handing it to the runtime system is called ***throwing an exception*.**

Exception

- The following program causes an exception.

ExceptionDemo_1.java *

```
1   public class ExceptionDemo_1 {
2     public static void main (String[] args)
3     {   int x=5, y=0;
4         System.out.println(x/y);
5         System.out.println("Hello");
6     }
7   }
```

Exceptions are pre-defined data (Exception classes) thrown by JVM and they can be caught by code in the program

Output - Chapter04 (run)

```
run:
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at ExceptionDemo_1.main(ExceptionDemo_1.java:4)
Java Result: 1
BUILD SUCCESSFUL (total time: 2 seconds)
```

Kinds of Exception

- java.lang.Throwable (implements java.io.Serializable)
  - java.lang.Error
  - java.lang.Exception
    - java.lang.RuntimeException

Checked Exceptions (We must use the try catch blocks or throw)

Unchecked- Exceptions Program Bugs (We may not use the try catch blocks)

Refer to the Java.lang documentation for more information.

```
1   public class ExceptionDemo_1 {
2       public static void main (String[] args)
3     {  int[] a= { 1,2,3,4,5};
4        int n=10;
5        for (int i=0;i<n;i++)
6            System.out.print("" + a[i] + ",");
7     }
8   }
9
```

Output - Chapter04 (run)

```
run:
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5
1,2,3,4,5,        at ExceptionDemo_1.main(ExceptionDemo_1.java:6)
Java Result: 1
BUILD SUCCESSFUL (total time: 1 second)
```

```
1   public class ExceptionDemo_1 {
2       public static void main (String[] args)
3     {  int[] a= { 1,2,3,4,5};
4        int n=10;
5        try
6        { for (int i=0;i<n;i++)
7            System.out.print("" + a[i] + ",");
8        }
9        catch(Exception e) // general exception
10       {  System.out.println(e);
11       }
12    }
```

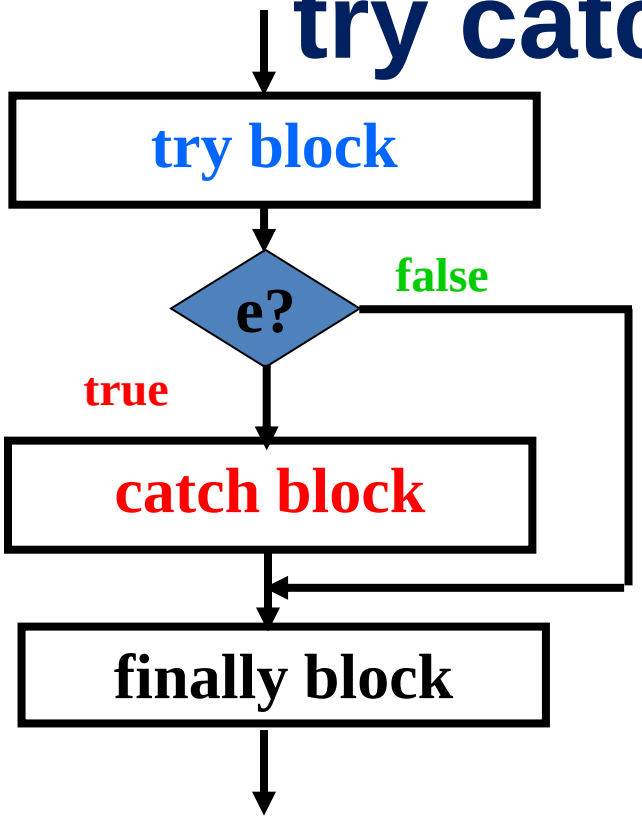
Output - Chapter04 (run)

```
run:
1,2,3,4,5,java.lang.ArrayIndexOutOfBoundsException: 5
BUILD SUCCESSFUL (total time: 0 seconds)
```

Kinds of Exception

- ***Checked exception***
  - Must be handled by either the try-catch mechanism or the throws-declaration mechanism.
- **Runtime exception**
  - The right time to deal with runtime exceptions is when you're designing, developing, and debugging your code. Since runtime exceptions <u>should never be thrown in finished code.</u>

Catching exceptions: try catch finally mechanism



If no exception is thrown in the try *block, all* catch *blocks are bypassed*

```
try {
    < statements may cause exceptions >
}

catch ( ExceptionType1 e1 ) {
    < statements handle the situation 1>
}

catch ( ExceptionType2 e2) {
    < statements handle the situation 2>
}

finally {
    < statements are always executed >
}
```

If an exception arises, the first matching catch *block, if any, is executed, and the others are skipped*

Catching specific/general-level exception

ExceptionDemo_1.java

```java
public class ExceptionDemo_1 {
  public static void main (String[] args)
  {  int x=6, y=0;
     try
     { System.out.println(x/y);
       // other statements
     }
     catch( ArithmeticException e)
     {  System.out.println(e);
        y=2;
     }
     finally
     { System.out.println("Hello");
       System.out.println(x/y);
     }
  }
}
```

Output - Chapter04 (run)

```
run:
java.lang.ArithmeticException: / by zero
Hello
3
BUILD SUCCESSFUL (total time: 0 seconds)
```

ExceptionDemo_1.java *

```java
public class ExceptionDemo_1 {
  public static void main (String[] args)
  {  int x=6, y=0;
     try
     { System.out.println(x/y);
       // other statements
     }
     catch(Exception e) // general exception
     {  e.printStackTrace();
        y=2;
     }
     finally
     { System.out.println("Hello");
       System.out.println(x/y);
     }
  }
}
```

Type conformity: father=son;

Output - Chapter04 (run)

```
run:
Hello
java.lang.ArithmeticException: / by zero
3
        at ExceptionDemo_1.main(ExceptionDemo_1.java:5)
BUILD SUCCESSFUL (total time: 0 seconds)
```

The *finally* block (1)

- A try block may optionally have a finally block associated with it.
- The code within a finally block is *guaranteed* to execute no matter what happens in the try/catch code that precedes it.
  - The try block executes to completion without throwing any exceptions whatsoever.
  - The try block throws an exception that is handled by one of the catch blocks.
  - The try block throws an exception that is ***not* handled by *any* of the catch blocks**

Nesting of try/catch Blocks

- A try statement may be nested inside either the try or catch block of another try statement.

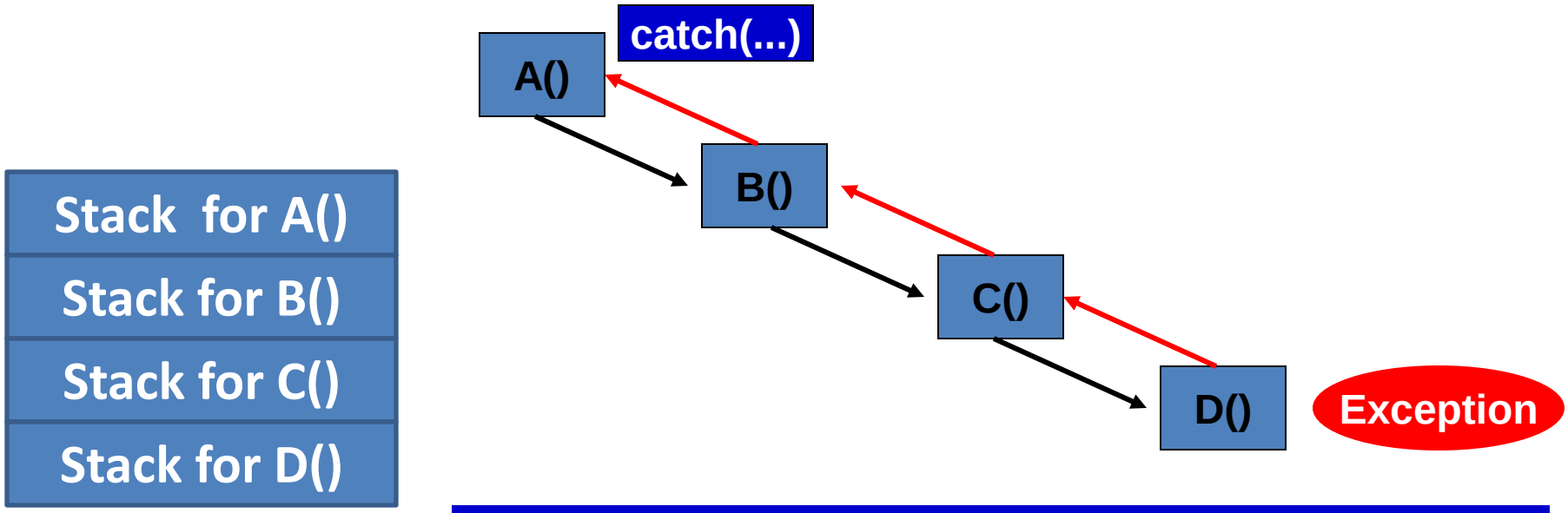
```
try {
   // Pseudo code.
   open a user-specified file
   }
   catch (FileNotFoundException e) {
       try {
           // Pseudo code.
           open a DEFAULT file instead ...
       }
       catch (FileNotFoundException e2) {
          // Pseudo code.
          attempt to recover ...
       }
}
```

Catching exceptions: Throws mechanism

The throws keyword indicates what exception type may be thrown by a method.

```
8   import java.io.FileNotFoundException;
9   import java.io.FileReader;
10  public class DemoException2 {
11      public static void demoReadFile() throws FileNotFoundException
12      {
13          FileReader f=new FileReader("computer.txt");
14      }
15      public static void main(String[] args){
16          try{
17              demoReadFile();
18          }catch(FileNotFoundException e)
19          {
20              System.out.println("somthing are wrong");
21          }
22      }
```

Exception Propagations



Stack trace

When an exception occurs at a method, program stack is containing running methods (method A calls method B,….). So, we can trace statements related to this exception.

Exception Propagations

```java
1  public class ExceptionPropagate {
2      public void mA()
3      {
4          mB();
5      }
6      public void mB()
7      {
8          mC();
9      }
10     public void mC()
11     {
12         System.out.println(5/0);
13     }
14     public static void main(String[]args){
15         ExceptionPropagate obj= new ExceptionPropagate();
16         obj.mA();
17     }
18 }
```

Output - FirstPrj (run)

```
run:
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at ExceptionPropagate.mC(ExceptionPropagate.java:12)
        at ExceptionPropagate.mB(ExceptionPropagate.java:8)
        at ExceptionPropagate.mA(ExceptionPropagate.java:4)
        at ExceptionPropagate.main(ExceptionPropagate.java:16)
Java Result: 1
```

Example:Catching Exceptions

Using try…catch to input an integer ,10<=n<=50

```
public static int inputInteger(){
     Scanner in = new Scanner(System.in);
     boolean cont = true;
     int n;
     do {
        try {
                System.out.print("Enter a whole number: ");
                n = Integer.parseInt(in.nextLine());
                cont = false;
        } catch (Exception e) {
               System.out.println("Required integer!");
        }
     } while (cont == true|| n<10 || n>50);
     return n;
}
public static void main(String[] args){
    int n= inputInteger();
    System.out.print("number:" + n);
}
```

Example:Catching Exceptions

Using try…catch to input an integer ,10<=n<=50

```
public static int inputInteger(){
    Scanner in = new Scanner(System.in);
    boolean cont = true;
    int n;
    do {
       try {
               System.out.print("Enter a whole number: ");
               n = Integer.parseInt(in.nextLine());
               if( n<10 || n>50) throw new Exception();
               cont = false;
       } catch (Exception e) {
              System.out.println("Required integer!");
       }
    } while (cont==true);
    return n;
}
public static void main(String[] args){
    int n= inputInteger();
    System.out.print("number:" + n);
}
```

Example:Catching Exceptions

Using try…catch to input an integer ,10<=n<=50

```
public static int inputInteger() throws Exception{
    Scanner in = new Scanner(System.in);
    System.out.print("Enter a whole number: ");
    int n = Integer.parseInt(in.nextLine());
    if( n<10 || n>50) throw new Exception();
    return n;
}
public static void main(String[] args){
     boolean cont=true; int n=0;
     do{  try{ n= inputInteger();
                cont=false;
          }catch(Exception e){
             System.out.println("Required integer!"); }
     }while(cont==true);
     System.out.print("number:"+ n);
}
```

Summary

- We can use **try-catch mechanism** or **throws mechanism** to handle to avoid errors.
- A single try block can have multiple catch blocks associated with it
- Code Finalization and Cleaning Up (finally block)