Object Oriented Programming IT-1201

Dr. Jesmin Akhter
Associate Professor, IIT, JU

Exception Handling-2

Multiple catch Clauses

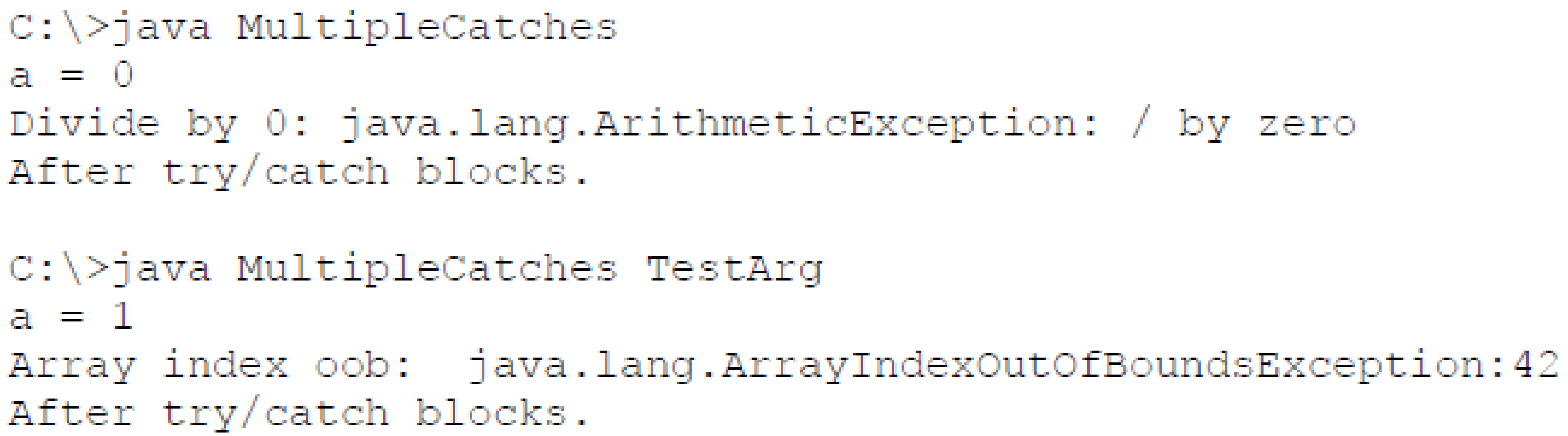
- In some cases, more than one exception could be raised by a single piece of code.
- To handle this type of situation, you can specify two or more **catch clauses, each catching a** different type of exception.
- When an exception is thrown, each **catch statement is** inspected in order, and the first one whose type matches that of the exception is executed.
- After one **catch statement executes, the others are bypassed, and execution** continues after the **try / catch block.**

Multiple catch Clauses : Example 1

```
// Demonstrate multiple catch statements.
class MultipleCatches {
  public static void main(String args[]) {
    try {
       int a = args.length;
      System.out.println("a = " + a);
      int b = 42 / a;
      int c[] = { 1 };
      c[42] = 99;
    } catch(ArithmeticException e) {
      System.out.println("Divide by 0: " + e);
    } catch(ArrayIndexOutOfBoundsException e) {
      System.out.println("Array index oob: " + e);
    }
    System.out.println("After try/catch blocks.");
  }
}
```

Multiple catch Clauses

- The output generated by running it both ways:



```
C:\>java MultipleCatches
a = 0
Divide by 0: java.lang.ArithmeticException: / by zero
After try/catch blocks.

C:\>java MultipleCatches TestArg
a = 1
Array index oob:  java.lang.ArrayIndexOutOfBoundsException:42
After try/catch blocks.
```

Multiple catch Clauses: Example 2

```
public class TestMultipleCatchBlock{
  public static void main(String args[]){
   try{
    int a[]=new int[5];
    a[5]=30/0;
   }
   catch(ArithmeticException e){System.out.println("task1 is completed");}
   catch(ArrayIndexOutOfBoundsException e){System.out.println("task 2 completed");}
   catch(Exception e){System.out.println("common task completed");}

   System.out.println("rest of the code...");
 }
}
```

```
Output:
task1 completed
rest of the code...
```

Multiple catch Clauses

- **Rule 1: At a time only one Exception is occurred and at a time only one catch block is executed.**

- **Rule 2:** When you use multiple **catch statements, it is important to remember that exception** subclasses must come before any of their super-classes.

Multiple catch Clauses

```
/* This program contains an error.

   A subclass must come before its superclass in
   a series of catch statements. If not,
   unreachable code will be created and a
   compile-time error will result.
*/
class SuperSubCatch {
  public static void main(String args[]) {
    try {
      int a = 0;
      int b = 42 / a;
     } catch(Exception e) {
       System.out.println("Generic Exception catch.");
    }
    /* This catch is never reached because
       ArithmeticException is a subclass of Exception. */
     catch(ArithmeticException e) { // ERROR - unreachable
       System.out.println("This is never reached.");
     }
  }
}
```

Multiple catch Clauses

- If you try to compile this program, you will receive an error message stating that the second **catch statement is unreachable because the exception has already been caught.**

- Since **ArithmeticException is a subclass of Exception,** the first catch statement will handle all **Exception-based errors, including ArithmeticException.**

- This means that the second **catch statement will never execute**. To fix the problem, reverse the order of the catch statements.

Multiple catch Clauses

```
class TestMultipleCatchBlock1{
 public static void main(String args[]){
  try{
   int a[]=new int[5];
   a[5]=30/0;
  }
  catch(Exception e){System.out.println("common task completed");}
  catch(ArithmeticException e){System.out.println("task1 is completed");}
  catch(ArrayIndexOutOfBoundsException e){System.out.println("task 2 completed");}
  System.out.println("rest of the code...");
 }
}
```

Output:
Compile-time error

Because ArithmeticException and ArrayIndexOutOfBoundsException is a sub class of **Exception**

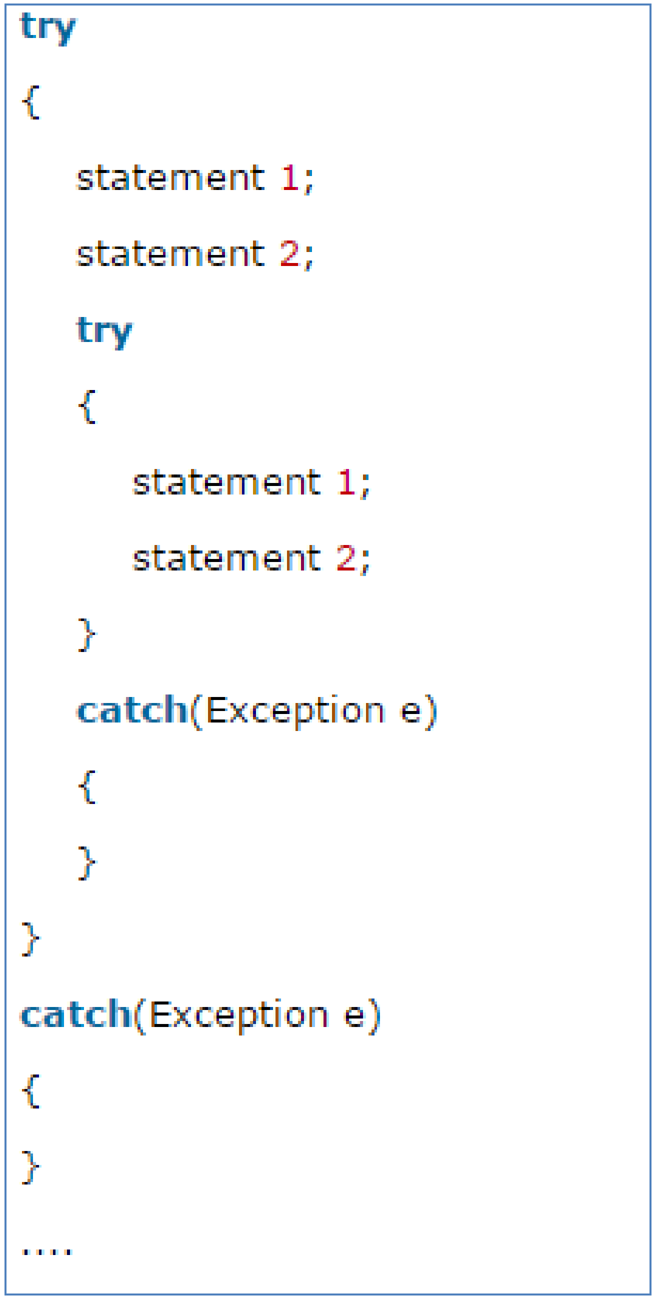
Nested try Statements

- The try block within a try block is known as nested try block in java.

Why use nested try block:

- Sometimes a situation may arise where a part of a block may cause one error and the entire block itself may cause another error. In such cases, exception handlers have to be nested.

- Syntax:



```
try
{
    statement 1;
    statement 2;
    try
    {
        statement 1;
        statement 2;
    }
    catch(Exception e)
    {
    }
}
catch(Exception e)
{
}
....
```

```
class Excep6{
 public static void main(String args[]){
  try{
    try{
     System.out.println("going to divide");
     int b =39/0;
    }catch(ArithmeticException e){System.out.println(e);}

    try{
    int a[]=new int[5];
    a[5]=4;
    }catch(ArrayIndexOutOfBoundsException e){System.out.println(e);}
   System.out.println("other statement);
  }catch(Exception e){System.out.println("handeled");}

  System.out.println("normal flow..");
 }
}
```

Thank you