OBJECT ORIENTED PROGRAMMING

Exception Handling

by
Dr. Nor Saradatul Akmar Zulkifli
Faculty of Computer Systems & Software Engineering
saradatulakmar@ump.edu.my

Content Overview

- Definition
- Catching an Exception (try and catch)
- Multiple Catch Block
- The `finally` block
- Types of Exception

Learning Objective

Students should be able to

❑ Enhance the reliability of code by combining exception-handling and assertion mechanism.

❑ Utilizing the try-catch blocks for catching and handling exceptions

❑ Write programmer-defined exception classes

WHAT IS AN EXCEPTION?

- An event or an error action which can occur during the normal process of a program execution and disrupts its normal flow.
- When this happen, or is `thrown`, the normal flow is terminated – Execute the exception-handling routine, which is thrown exception (known as `caught`)
- By catching the exception using special error recovery routines that has been develop – increase the program's reliability.
- It can be done by wrapping the statements with the `try-catch` control statement

CATCHING AN EXCEPTION?

```
String inputStr;
int age;

inputStr = JOptionPane.showInputDialog (null,”Age:”);
age      = Integer.parseInt (inputStr);
```

By entering “ten”, an error message for invalid input shown as below:

```
Java.lang.NumberFormatException: ten
       at java.lang.Integer.parseInt (Integer.java:405)
       at java.lang.Integer.parseInt (Integer.java:454)
       at Ch8Sample1.main (Ch8Sample1.java:20)
```

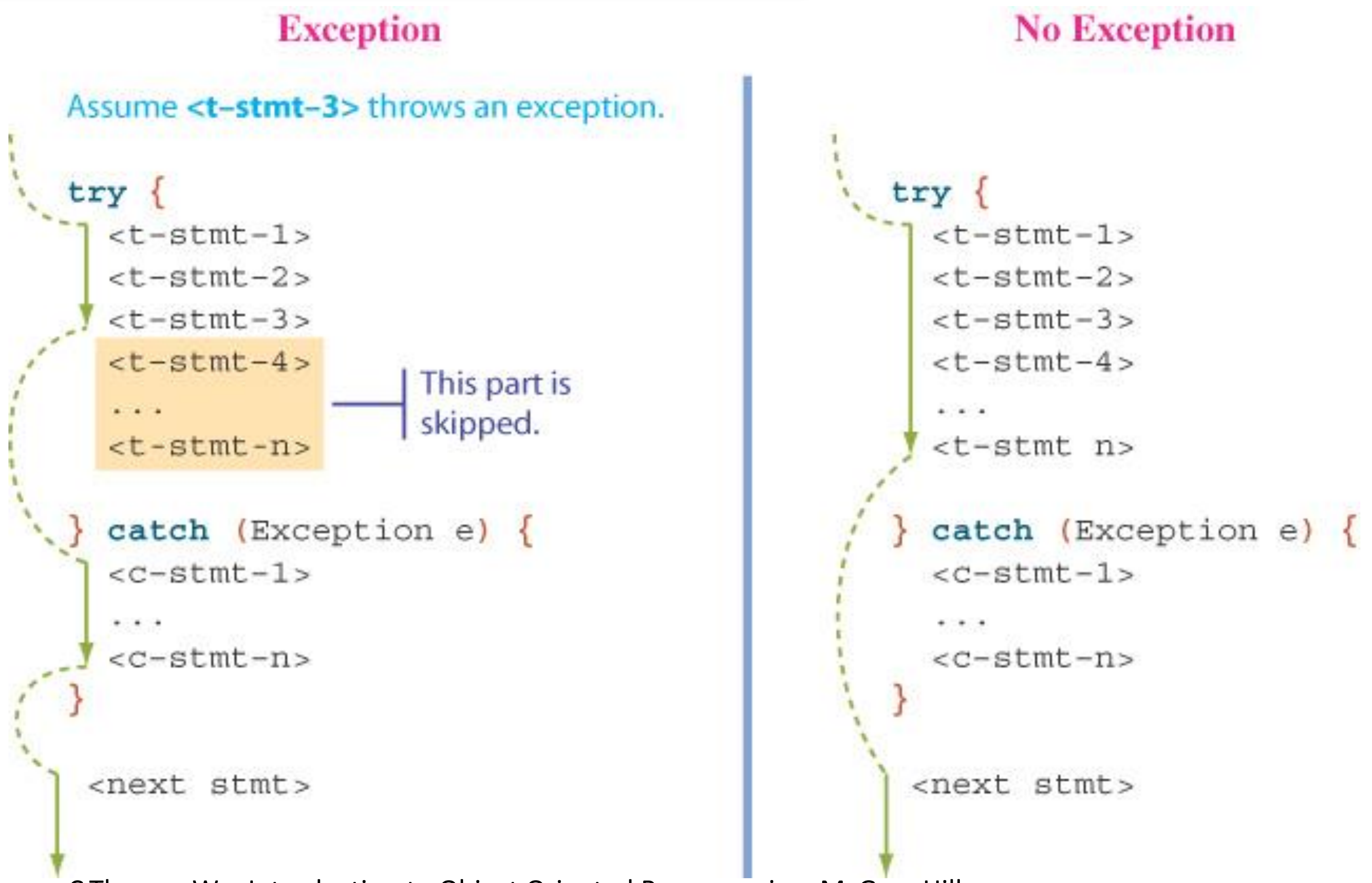
CATCHING AN EXCEPTION? ✓

```
inputStr = JOptionPane.showInputDialog(null, "Age:");

try {

    age = Integer.parseInt (inputStr);

} catch (NumberFormatException e) {

    JOptionPane.showMessageDialog (null, "'" +
            inputStr + "' is invalid\n"
            + "Please enter digits only");
}
```

try
executed in sequence

catch
When one of the statements throws an exception, control is passed to the matching `catch` block and execute statements inside the `catch` block

try-catch CONTROL FLOW



```
try {
   <t-stmt-1>
   <t-stmt-2>
   <t-stmt-3>
   <t-stmt-4>
   . . .
   <t-stmt-n>

} catch (Exception e) {
   <c-stmt-1>
   . . .
   <c-stmt-n>
}

  <next stmt>
```

Source : C.Thomas Wu, Introduction to Object Oriented Programming, McGrawHill

try-catch CONTROL FLOW

Execute statements in the **try** block in sequence.

One statements throws an exception, then pass the control to the matching **catch** block and execute statement inside the **catch** block

The execution continues to the statement following the **try-catch** block statement, ignoring any remaining statements in the try block

If no statements throw an exception in the try block, the catch block is ignored. Execution continues with the try-catch statement.

MULTIPLE CATCH BLOCKS

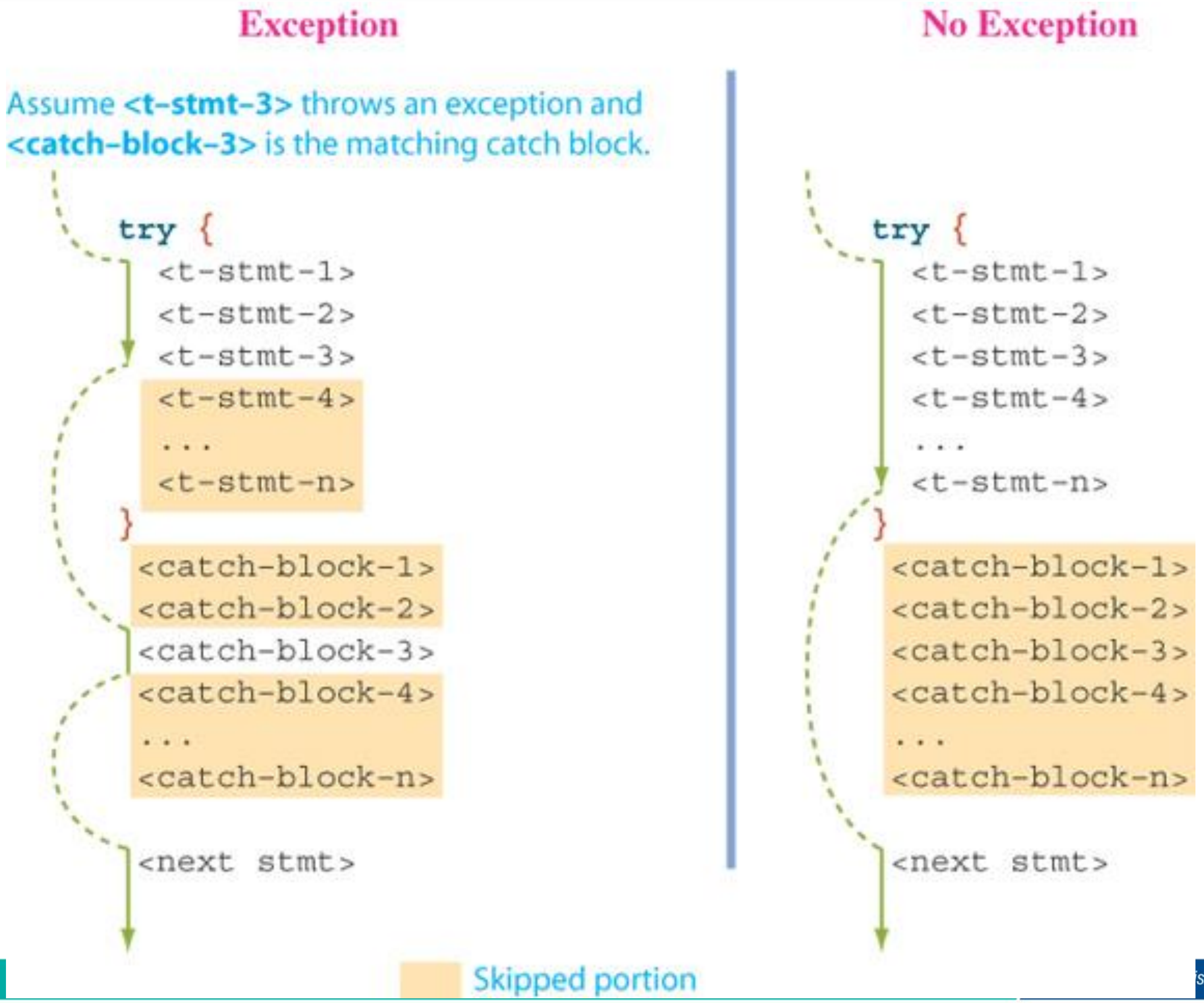
- A single try-catch block can include multiple catch blocks, one for each type of exception

```
try {
    age = Integer.parseInt (inputStr);
    if (age < 10) { //directly throw an exception
    throw new Exception ("Negative age is invalid");
    }
    return age;
} catch (NumberFormatException e) {
    System.out.println (e.getMessage());
} catch (Exception e) {
    System.out.println (e.getMessage());
```

MULTIPLE CATCH BLOCK

- Multiple `catch` blocks in a `try-catch` statements are check in sequence
- Start with more specialized exception classes before general exception classes.
- When an exception is thrown, its matching `catch` block is executed and the other `catch` blocks are ignored.

MULTIPLE CATCH BLOCK : CONTROL FLOW



THE `finally` BLOCK

1. In certain situation, whether an exception is thrown or not, action need to be taken

use **`finally`**

2. Place the statements that must be executed regardless of exceptions in the **`finally`** blocks

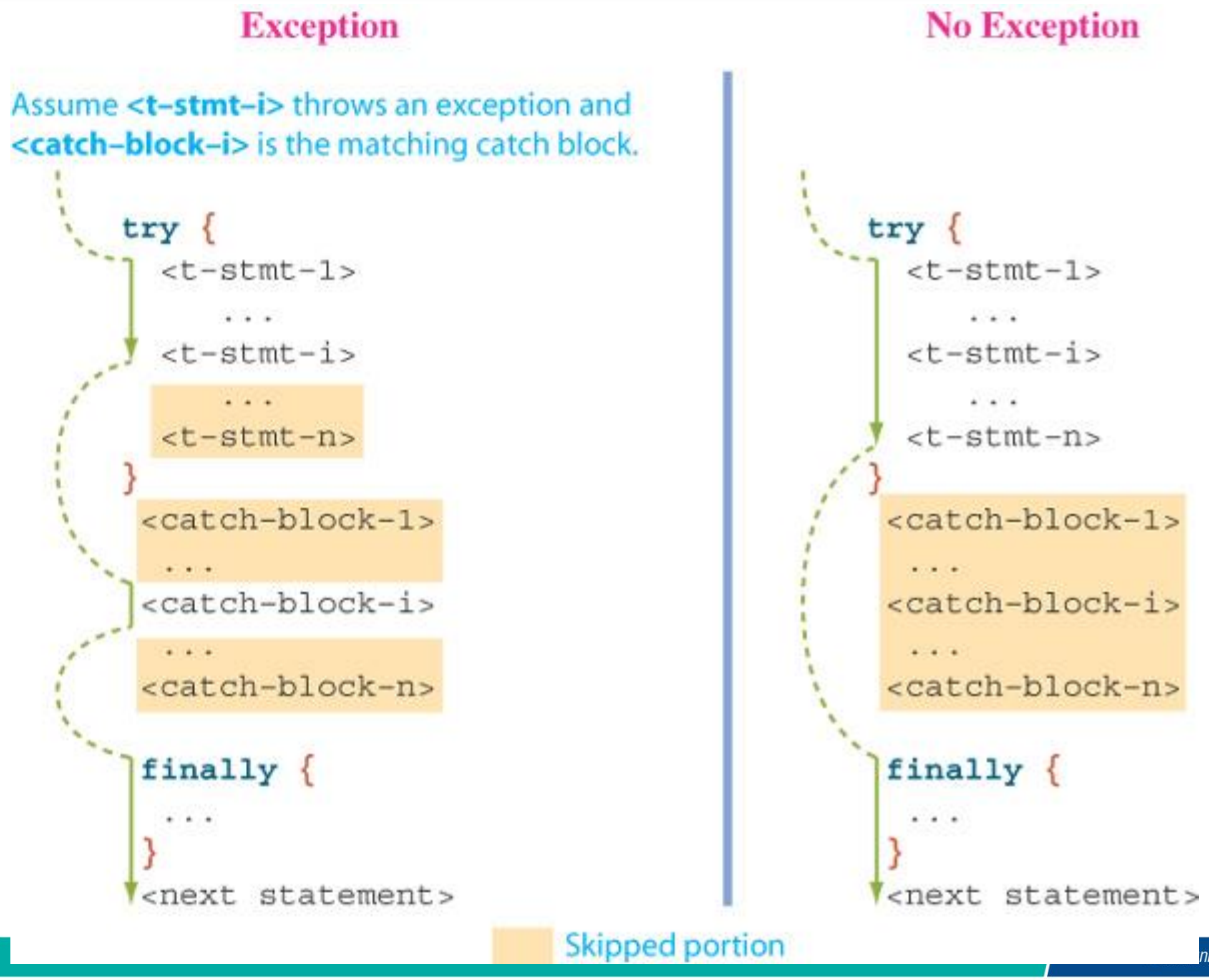
3. The **`finally`** block is executed even if there is a **`return`** statement inside the **`try`** block

4. When encountered with **`return`** statement in the **`try`** block, execute the **`finally`** block

THE finally BLOCK : EXAMPLE

```
try {
   age = Integer.parseInt (inputStr);
   if (age < 0) {
   throw new Exception (“Negative age is invalid”);
   }
   return age;
} catch (NumberFormatException e) {
  . . .
} catch (Exception e) {
  . . .
} finally {
       System.out.println (“DONE”);
}
```

try-catch-finally CONTROL FLOW



PROPAGATING EXCEPTIONS

Exception thrower is when a method throws an exception either directly (using `throw`) or indirectly (error in the program)

Exception thrower is one of two types:

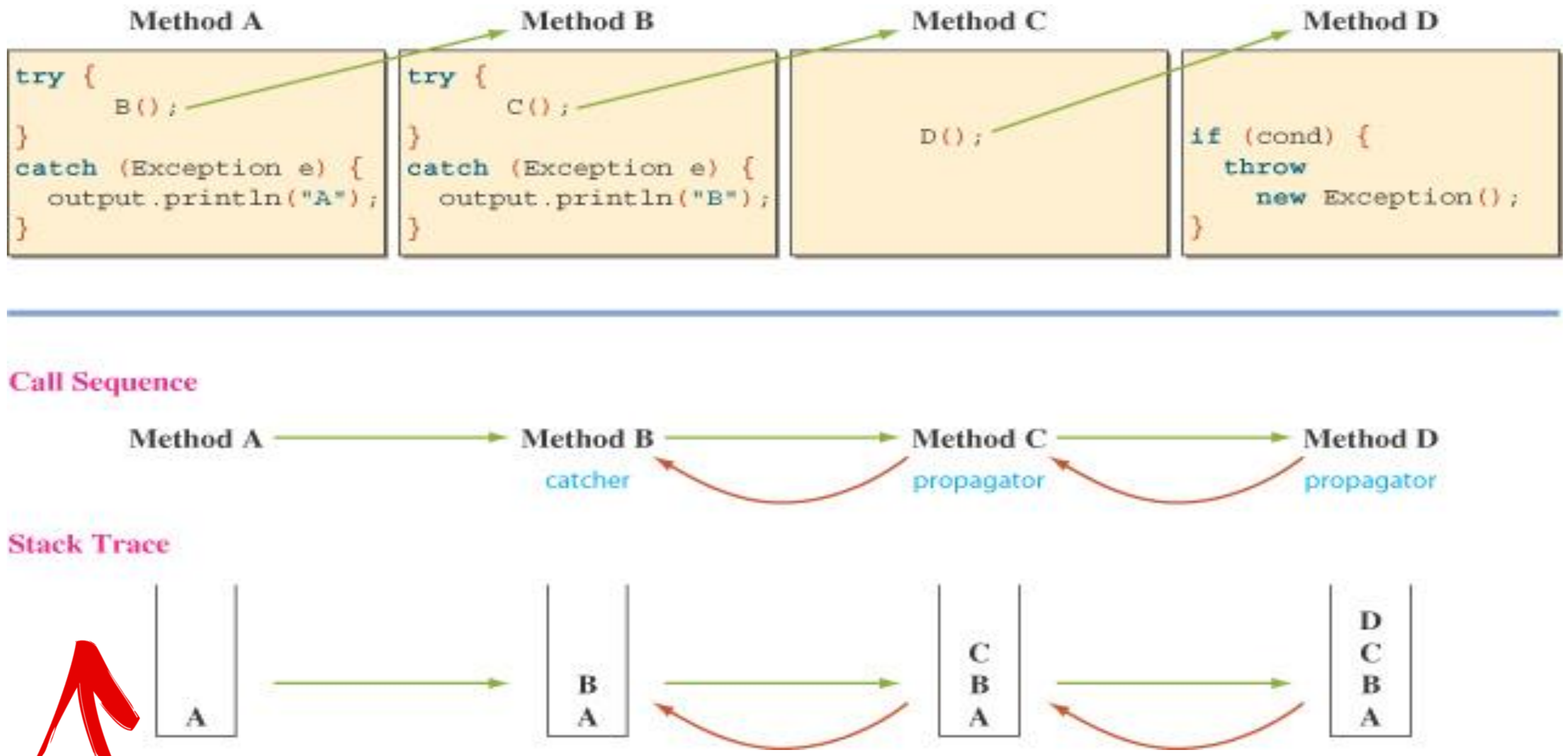
Catcher or **Propagator**

a. **Exception Catcher** – An exception thrower that **includes a matching `catch` block** for the thrown exception.

OR

b. **Exception Propagator** – An exception thrower that **does NOT contain a matching `catch` block**

CALL SEQUENCE : EXAMPLE



Method A
```
try {
      B();
}
catch (Exception e) {
   output.println("A");
}
```

Method B
```
try {
      C();
}
catch (Exception e) {
   output.println("B");
}
```

Method C
```
      D();
```

Method D
```
if (cond) {
   throw
      new Exception();
}
```

A sequence of method calls among the exception throwers

Source : C.Thomas Wu, Introduction to Object Oriented Programming, McGrawHill

PROPAGATING EXCEPTIONS

Method D throws an instance of **Exception**. **Green** arrows indicate the **direction of calls**, **Red** shows the **reversing of call sequence** looking for matching catcher, which is Method B.

The call sequence is traced using a stack

Instead of catching a thrown exception using the try-catch statement, **propagate the thrown exception back to the caller**

Method header includes the reserved word `throws`

THROWING EXCEPTION

```
public int getAge ( ) throws NumberFormatException {
    . . .
    int age = Integer.parseInt (inputStr);
    . . .
    return age;
}
```

Programmer can write a method that throws an exception directly (this method is the origin of the exception)

Use the `throw` reserved to create a new instance of the exception or its subclasses.

```
public void doWork (int num) throws Exception {
     . . .
     if (num != val) throw new Exception (“Invalid val”);
     . . .
}
```

TYPES OF EXCEPTIONS

Universiti Malaysia PAHANG
Engineering • Technology • Creativity

Checked

- An exception that is checked at compile time

Unchecked

- Detected only at runtime
- All other exceptions or runtime exception

Common Runtime Exceptions:

✓ NumberFormatException – an attempt to **parse a string into number** that has an illegal number format.

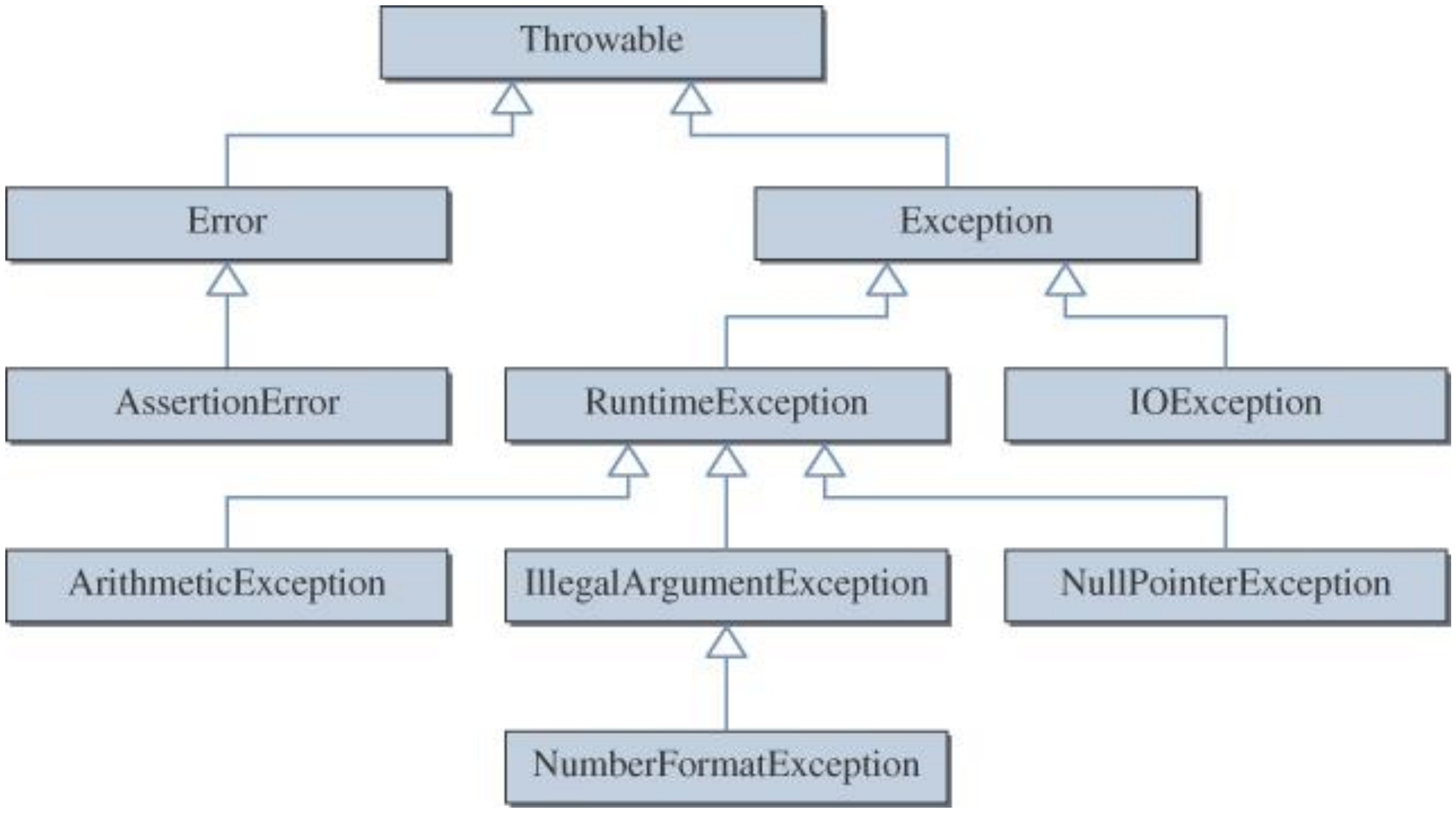
✓ ArithmeticException – the **result of a divide-by-zero operation** for integer

✓ ArrayIndexOutOfBoundException – An attempt to access an element of an array **beyond the array's size**

✓ FileNotFoundException – An attempt to read from **file that does not exist**

THROWABLE HIERARCHY

Over 60 classes in the hierarchy



Source : C.Thomas Wu, Introduction to Object Oriented Programming, McGrawHill

PROGRAMMER-DEFINED EXCEPTIONS

We can **pack more useful information** by defining our own exception class

It's created by **extending the Exception class**

Ex: AgeInputException – defined as a **subclass of Exception** and include public methods to access three pieces of information it carries (lower and upper bound of valid age input)

Author Information

Dr. Nor Saradatul Akmar Binti Zulkifli

Senior Lecturer
Faculty of Computer Systems & Software Engineering
Universiti Malaysia Pahang