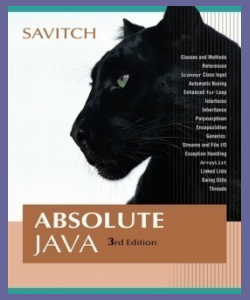
Comp 6481
Programming & Problem Solving
Chapter 9 – *Exception Handling*

Dr. Aiman Hanna
Department of Computer Science & Software Engineering
Concordia University, Montreal, Canada

These slides have been extracted, modified and updated from original slides of Absolute Java 3rd Edition by Savitch; which has originally been prepared by Rose Williams of Binghamton University. Absolute Java is published by Pearson Education / Addison-Wesley.



UNIVERSITÉ Concordia UNIVERSITY

PEARSON Addison Wesley

Introduction to Exception Handling

- Sometimes the best outcome can be when nothing unusual happens
- However, the case where exceptional things happen must also be prepared for
  - Java exception handling facilities are used when the invocation of a method may cause something exceptional to occur

Introduction to Exception Handling

- Java library software (or programmer-defined code) provides a mechanism that signals when something unusual happens
  - This is called *throwing an exception*
- In another place in the program, the programmer must provide code that deals with the exceptional case
  - This is called *handling the exception*

try-throw-catch Mechanism

- The basic way of handling exceptions in Java consists of the ***try-throw-catch*** trio

- The **try** block contains the code for the basic algorithm
  - It tells what to do when everything goes smoothly
  - It can also contain code that throws an exception if something unusual happens

```
try
{
   CodeThatMayThrowAnException
}
```

try-throw-catch Mechanism

- If something goes wrong within the **try** block, execution of the block is stopped and an exception is thrown
  - The thrown exception is always an object of some exception class

- A **throw** statement would look like that:

```
throw new
   ExceptionClassName(PossiblySomeArguments);
```

  - The execution of a **throw** statement is called *throwing an exception*

- Normally, following the exception throwing, the flow of control is transferred to another portion of code known as the **catch** block

try-throw-catch Mechanism

- When an exception is thrown, the **catch** block begins execution
  - The **catch** block has one parameter, which is the object thrown by the exception

```
catch(Exception e)
{
   ExceptionHandlingCode
}
```

  - Note: The identifier **e** is often used by convention, but any non-keyword identifier can be used

`try-throw-catch` Mechanism

- The execution of the **`catch`** block is called *catching the exception*, or *handling the exception*
  - Whenever an exception is thrown, it should ultimately be handled (or caught) by some **`catch`** block

- If no exception occurs, nothing is thrown and the **`catch`** block is skipped

See ExceptionHandling1.java
See ExceptionHandling2.java
See ExceptionHandling3.java

Exception Classes

- There are more exception classes than just the single class **`Exception`**
  - New exception classes can also be defined like any other class
- All predefined exception classes have the following two properties:
  - There is a constructor that takes a single argument of type **`String`**
  - The class has an accessor method **`getMessage`** that returns the string given as an argument to the constructor when the exception object was created
- All programmer-defined classes *should* have the above two properties

Exception Classes from Standard Packages

- Numerous predefined exception classes are included in the standard packages that come with Java
  - For example:

    `IOException`
    `NoSuchMethodException`
    `FileNotFoundException`

  - Many exception classes must be imported in order to use them

    ```
    import java.io.IOException;
    ```

  - The class `Exception` is in the `java.lang` package, and so requires no `import` statement

Exception Classes from Standard Packages

- The predefined exception class **`Exception`** is the root class for all exceptions
  - Every exception class is a derived, or descendent, class of the class **`Exception`**
  - Although the **`Exception`** class can be used directly in a class or program, it is most often used to define a derived class

Defining Exception Classes

- A **`throw`** statement can throw an exception object of any exception class

- Instead of using a predefined class, programmers can define their own exception classes
  - These can be tailored to carry the precise kinds of information needed in the **`catch`** block
  - Different types of exception can be defined to identify different exceptional situations

Defining Exception Classes

- Every exception class to be defined must be a derived class of some already defined exception class
  - It can be a derived class of any exception class in the standard Java libraries, or of any programmer defined exception class
- Constructors are the most important members to define in an exception class
  - They must behave appropriately with respect to the variables and methods inherited from the base class
  - Often, there are no other members, except those inherited from the base class
- The following exception class performs these basic tasks only

An Example of a Programmer-Defined Exception Class

Display 9.3 A Programmer-Defined Exception Class

```
1  public class DivisionByZeroException extends Exception
2  {
3      public DivisionByZeroException()
4      {
5          super("Division by Zero!");
6      }

7      public DivisionByZeroException(String message)
8      {
9          super(message);
10     }
11 }
```

You can do more in an exception constructor, but this form is common.

super is an invocation of the constructor for the base class Exception.

See ExceptionHandling4.java

Tip: An Exception Class Can Carry a Message of Any Type: int Message

- An exception class constructor can be defined that takes an argument of another type
  - It would stores its value in an instance variable
  - It would need to define accessor methods for this instance variable

An Exception Class with an `int` Message

Display 9.5 An Exception Class with an `int` Message

```
1  public class BadNumberException extends Exception
2  {
3      private int badNumber;

4      public BadNumberException(int number)
5      {
6          super("BadNumberException");
7          badNumber = number;
8      }

9      public BadNumberException()
10     {
11         super("BadNumberException");
12     }

13     public BadNumberException(String message)
14     {
15         super(message);
16     }

17     public int getBadNumber()
18     {
19         return badNumber;
20     }
21 }
```

Throwing an Exception in a Method

- Sometimes it makes sense to throw an exception in a method, but not catch it in the same method
  - Some programs that use a method should just end if an exception is thrown, and other programs should do something else
  - In such cases, the program using the method should enclose the method invocation in a **`try`** block, and catch the exception in a **`catch`** block that follows
- In this case, the method itself would not include **`try`** and **`catch`** blocks
- In such a case, the method must provide a warning
  - This warning is called a ***`throws`*** *clause*

Declaring Exceptions in a `throws` Clause

- The process of including an exception class in a throws clause is called *declaring the exception*

```
throws AnException  //throws clause
```

  - The following states that an invocation of **`aMethod`** could throw **`AnException`**

```
public void aMethod() throws AnException
```

See ExceptionHandling5.java

Declaring Exceptions in a `throws` Clause

- If a method can throw more than one type of exception, then the exception types are separated by commas

```
public void aMethod() throws
    AnException, AnotherException
```

- If a method throws an exception and the catch clause is inside the method, then throwing the exception does not terminate the method (i.e. executes the catch and what follows it, if any)
- However if the catch is not inside the method, then throwing the exception ends the method immediately

The Catch or Declare Rule

- Most ordinary exceptions that might be thrown within a method must be accounted for in one of two ways:

1. The code that can throw an exception is placed within a **`try`** block, and the possible exception is caught in a **`catch`** block within the same method

2. The possible exception can be declared at the start of the method definition by placing the exception class name in a **`throws`** clause

The Catch or Declare Rule

- The first technique handles an exception in a `catch` block
- The second technique is a way to shift the exception handling responsibility to the method that invoked the exception throwing method
- The invoking method must handle the exception, unless it too uses the same technique to "pass the buck“
- Ultimately, every exception that is thrown should eventually be caught by a `catch` block in some method that does not just declare the exception class in a `throws` clause

The Catch or Declare Rule

- In any one method, both techniques can be mixed
  - Some exceptions may be caught, and others may be declared in a `throws` clause
- However, these techniques must be used consistently with a given exception
  - If an exception is not declared, then it must be handled within the method
  - If an exception is declared, then the responsibility for handling it is shifted to some other calling method
  - Note that if a method definition encloses an invocation of a second method, and the second method can throw an exception and does not catch it, then the first method must catch or declare it

Multiple `catch` Blocks

- A **`try`** block can potentially throw any number of exception values, and they can be of different types
  - In any one execution of a **`try`** block, at most one exception can be thrown (since a throw statement ends the execution of the **`try`** block)

- Different types of exceptions can be caught by placing more than one **`catch`** block after a **`try`** block
  - Any number of **`catch`** blocks can be included, but they must be placed in the correct order

Pitfall: Catch the More Specific Exception First

- When catching multiple exceptions, the order of the `catch` blocks is important
  - When an exception is thrown in a `try` block, the `catch` blocks are examined in order
  - The first one that matches the type of the exception thrown is the one that is executed

See ExceptionHandling6.java

Pitfall: Catch the More Specific Exception First

Example:

```
catch (Exception e)
{ . . . }
catch (NegativeNumberException e)
{ . . . }
```

- Because a **NegativeNumberException** is a type of **Exception**, all **NegativeNumberExceptions** will be caught by the first **catch** block before ever reaching the second block
  - The catch block for **NegativeNumberException** will never be used!
- For the correct ordering, simply reverse the two blocks

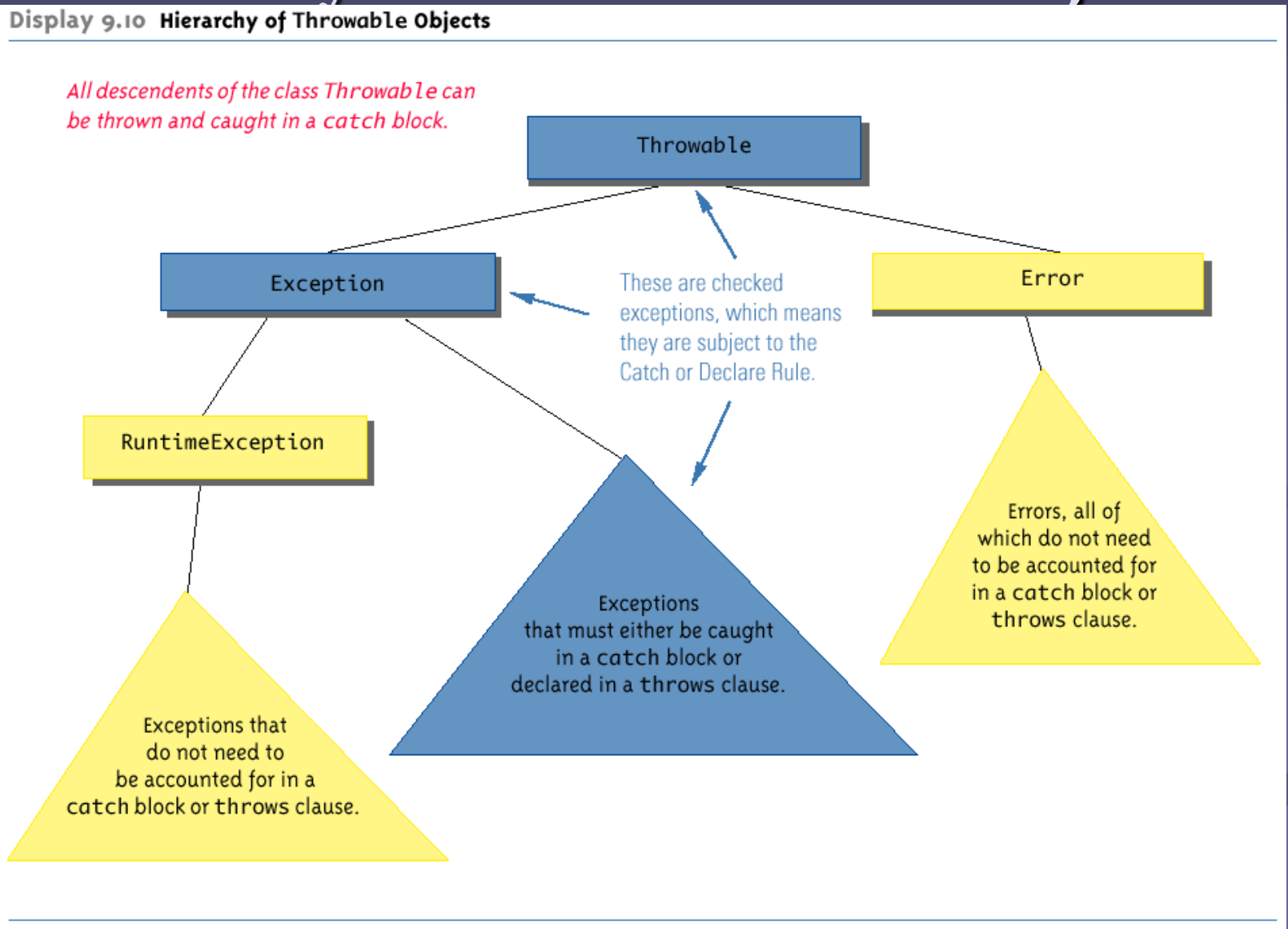
Checked and Unchecked Exceptions

- Exceptions that are subject to the catch or declare rule are called *checked* exceptions
  - The compiler checks to see if they are accounted for with either a catch block or a throws clause
  - The classes **`Throwable`**, which is the superclass of all errors and exceptions in Java, **`Exception`**, and all descendants of the class **`Exception`** are checked exceptions
- All other exceptions are *unchecked* exceptions
- The class **`Error`** and all its descendant classes are called *error classes*
  - Error classes are *not* subject to the Catch or Declare Rule

Exceptions to the Catch or Declare Rule

- Checked exceptions must follow the Catch or Declare Rule
  - Programs in which these exceptions can be thrown will not compile until they are handled properly
- Unchecked exceptions are exempt from the Catch or Declare Rule
  - Programs in which these exceptions are thrown simply need to be corrected, as they result from some sort of error

Hierarchy of Throwable Objects



The `throws` Clause in Derived Classes

- When a method in a derived class is overridden, it should have the same exception classes listed in its **`throws`** clause that it had in the base class
  - Or it should have a subset of them
- A derived class may not add any exceptions to the **`throws`** clause
  - But it can delete some

What Happens If an Exception is Never Caught?

- If every method up to and including the main method simply includes a **`throws`** clause for an exception, that exception may be thrown but never caught
  - In a GUI program (i.e., a program with a windowing interface), nothing happens - but the user may be left in an unexplained situation, and the program may no longer be reliable
  - In non-GUI programs, this causes the program to terminate with an error message giving the name of the exception class
- Every well-written program should eventually catch every exception by a **`catch`** block in some method

When to Use Exceptions

- Exceptions should be reserved for situations where a method encounters *an unusual or unexpected case that cannot be handled easily in some other way*

- When exception handling must be used, some basic guidelines should be followed:
  - Include **`throw`** statements and list the exception classes in a **`throws`** clause within a method definition

  - Place the **`try`** and **`catch`** blocks in a different method

Event Driven Programming

- Exception handling is an example of a programming methodology known as *event-driven programming*
- When using event-driven programming, objects are defined so that they send events to other objects that handle the events
  - An event is an object also
  - Sending an event is called *firing an event*

Event Driven Programming

- In exception handling, the event objects are the exception objects
  - They are fired (thrown) by an object when the object invokes a method that throws the exception
  - An exception event is sent to a **`catch`** block, where it is handled

Pitfall: Nested `try-catch` Blocks

- It is possible to place a **`try`** block and its following catch blocks inside a larger **`try`** block, or inside a larger **`catch`** block
  - If a set of **`try-catch`** blocks are placed inside a larger **`catch`** block, different names must be used for the **`catch`** block parameters in the inner and outer blocks, just like any other set of nested blocks
  - If a set of **`try-catch`** blocks are placed inside a larger **`try`** block, and an exception is thrown in the inner **`try`** block that is not caught, then the exception is thrown to the outer **`try`** block for processing, and may be caught in one of its **`catch`** blocks

See ExceptionHandling7.java

The `finally` Block

- The **`finally`** block contains code to be executed whether or not an exception is thrown in a **`try`** block
  - If it is used, a **`finally`** block is placed after a **`try`** block and its following **`catch`** blocks

```
try
{   . . .   }
catch(ExceptionClass1 e)
{   . . .   }
   . . .
catch(ExceptionClassN e)
{   . . .   }
finally
{
   CodeToBeExecutedInAllCases
}
```

See ExceptionHandling8.java

The `finally` Block

- If the **`try-catch-finally`** blocks are inside a method definition, there are three possibilities when the code is run:
  1. The **`try`** block runs to the end, no exception is thrown, and the finally block is executed
  2. An exception is thrown in the **`try`** block, caught in one of the **`catch`** blocks, and the **`finally`** block is executed
  3. An exception is thrown in the **`try`** block, there is no matching **`catch`** block in the method, the **`finally`** block is executed, and then the method invocation ends and the exception object is thrown to the enclosing method

Note: Of course, if the execution of any code that proceeds the finally block results in flow control being driven away (i.e. execute System.exit(0);), the finally block will not execute

Rethrowing an Exception

- A `catch` block can contain code that throws an exception
  - Sometimes it is useful to catch an exception and then, depending on the string produced by `getMessage` (or perhaps something else), throw the same or a different exception for handling further up the chain of exception handling blocks

See ExceptionHandling9.java

The `AssertionError` Class

- When a program contains an assertion check, and the assertion check fails, an object of the class **`AssertionError`** is thrown
  - This causes the program to end with an error message
- The class **`AssertionError`** is derived from the class **`Error`**, and therefore is an unchecked exception
  - In order to prevent the program from ending, it could be handled, but this is not required

Tip: Exception Controlled Loops

- Sometimes it is better to simply loop through an action again when an exception is thrown, as follows:

```
boolean done = false;
while (! done)
{
   try
   {
      CodeThatMayThrowAnException
      done = true;
   }
   catch (SomeExceptionClass e)
   {
      SomeMoreCode
   }
}
```

Exception Handling with the `Scanner` Class

- The **`nextInt`** method of the **`Scanner`** class can be used to read **`int`** values from the keyboard

- However, if a user enters something other than a well-formed **`int`** value, an **`InputMismatchException`** will be thrown
  - Unless this exception is caught, the program will end with an error message
  - If the exception is caught, the **`catch`** block can give code for some alternative action, such as asking the user to reenter the input

See ExceptionHandling10.java

The `InputMismatchException`

- The **`InputMismatchException`** is in the standard Java package **`java.util`**
  - A program that refers to it must use an **`import`** statement, such as the following:

```
import java.util.InputMismatchException;
```

- It is a descendent class of **`RuntimeException`**
  - Therefore, it is an unchecked exception and does not have to be caught in a **`catch`** block or declared in a **`throws`** clause
  - However, catching it in a **`catch`** block is allowed, and can sometimes be useful

ArrayIndexOutOfBoundsException

- An **ArrayIndexOutOfBoundsException** is thrown whenever a program attempts to use an array index that is out of bounds
  - This normally causes the program to end
- Like all other descendents of the class **RuntimeException**, it is an unchecked exception
  - There is no requirement to handle it
- When this exception is thrown, it is an indication that the program contains an error
  - Instead of attempting to handle the exception, the program should simply be fixed