Exceptions in Java

Thumrongsak Kosiyatrakul
tkosiyat@cs.pitt.edu

Exceptions in Java

- An **Exception** is an occurrence of an **erroneous**, **unusual or unexpected** event in a program execution
  - Cannot open a file
  - Index of an array is out-of-bound
  - Parse a string into an integer but the string is not a string representation of an integer
- How do we handle it so far
  - Nothing – the program simply terminated
    - Example: the `ArrayIndexOutOfBondsException`
  - Throw it to the caller (Operating System)
    - Example: the `IOException` or `FileNotFoundException`

Exceptions in Java

- Exceptions are handled using **try-catch** blocks

```
try
{
    // code that will normally execute
}
catch (ExceptionType1 e)
{
    // code to "handle" this ExceptionType1
}
catch (ExceptionType2 e)
{
    // code to "handle" this ExceptionType2
}
:
finally
{
    // code to "clean up" before leaving try block
}
```

Exceptions in Java

- If all goes well (no exceptions occur)
  - Code in **try** block is executed, followed by code in (optional) **finally** block
- If an **exception occurs** anywhere in the try block
  - Execution immediately jumps out of the try block (i.e. the try block does not complete its execution)
  - An **exception handler** is sought in a catch block
    - If execution **is handled** in a catch block, that block executes followed by the (optional) finally block
    - If the exception **is not handled** in a catch block, the (optional) finally block is executed and then the exception is **propagated**
- Note that in all cases the **finally block** is executed if it is present

Exceptions in Java

- If an exception is **handled**
  - Execution resumes immediately **After** try/catch block in which it was **handled**, and does not return to throw point
  - **termination model** of exception handling
    - As opposed to a resumption model, where execution resumes from where the exception occurred
- If an exception is **propagated**
  - A handler is searched for by backing up through the call chain on the run-time stack
  - This is **dynamic exception propagation**
  - If no handler is ever found
    - Console applications crash and report exception

Simple Example

- Consider the following program:

```
public class Ex1
{
    public static void main(String[] args)
    {
        int[] x = new int[5];
        x[10] = 123;
        System.out.println("Done!!!");
    }
}
```

output

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 10
       at Ex1.main(Ex1.java:6)
```

- The above program **terminates** immediately because of the exception

Simple Example

- Consider the following program:

```
public class Ex2
{
    public static void main(String[] args)
    {
        int[] x = new int[5];
        try
        {
            x[10] = 123;
        }
        catch (ArrayIndexOutOfBoundsException e)
        {
            System.out.println("Index is out-of-bound");
        }
        System.out.println("Done!!!");
    }
}
```

output

```
Index is out-of-bound
Done!!!
```

- `ArrayIndexOutOfBoundsException` is handled

Checked vs Unchecked Exceptions

- **Checked** Exceptions
  - If a method does not handle these, the method MUST state that it throws them
    - Done in a **throws clause** in the method header
    - Remember? `IOException` or `FileNotFoundException` when we try to open a file
- **Unchecked** Exeptions
  - Method not required to explicitly "throw" these
  - These include `RunTimeException`
    - `ArrayIndexOutOfBoundsException`
    - `NumberFormatException`

Catching Exceptions

- Catching a **superclass** of an exception will catch **subclass** exception object
  - For example, `FileNotFoundException` is a subclass of the class `IOException`

```
try
{
    :
}
catch (IOException e)
{
    // This block will be executed if the
    // FileNotFoundException occurs in the try block
}
```

- The class `Exception` is the superclass of all Java exceptions

```
:
catch (Exception e)
{
    // Catch all if no other exceptions match
}
```

Catching Exceptions

- Because of superclass and subclass exceptions
  - You should list (catch) exceptions in order of most specific to most general
    - If `catch (Exception e)` is the first, **NO OTHER** catches in the block could ever execute
- It is better style to be as specific as possible with the exceptions that are caught
  - See `ex23.java`