Chapter 12.

Exception Handling

2020-2021
COMP2396 Object-Oriented Programming and Java
Dr. T.W. Chim (E-mail: twchim@cs.hku.hk)
Department of Computer Science, The University of Hong Kong

Risky Behavior

- Suppose you want to call a method in a class not written by you.
- That method does something risky, something that might not work at runtime (e.g., opening a file that does not exist).
- You need to know that the method you are calling is risky.
- You can then write code which can handle the exceptional situation if it does happen.

```
public class Cow {
   void moo() {
      if (serverDown()) {
         explode();
      }
   }
}
```

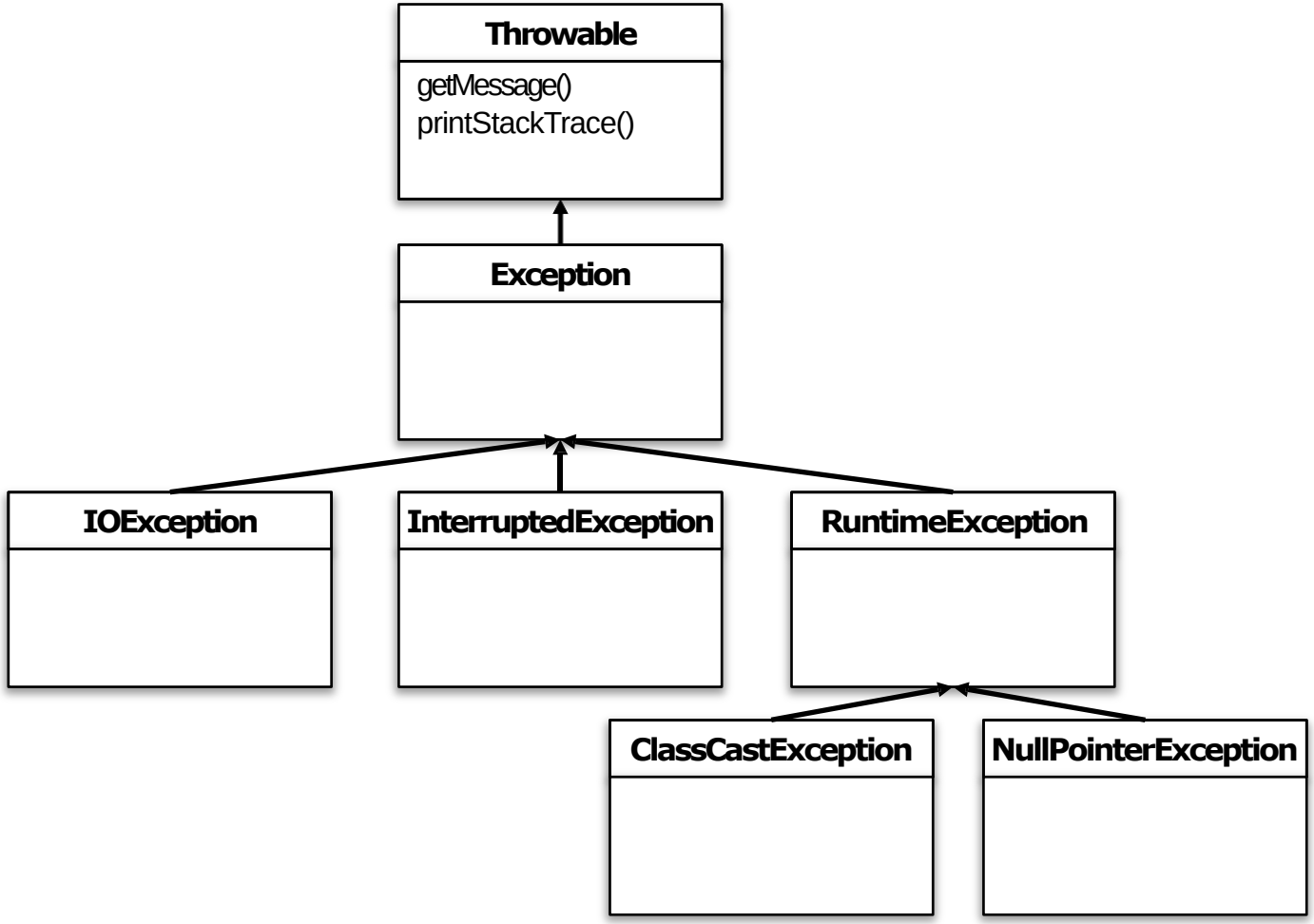
Exceptions in Java

- In Java, a method can throw an exception when something fails at runtime.
- An exception is an object of type Exception (or any of its subclasses)
- A risky method (i.e., one that may throw an exception)
  - Declares the type of exception it might throw using the keyword throws
  - Throws an exception using the keyword throw

```
public void takeRisk() throws BadException {
   if (abandonAllHope()) {
      throw new BadException();
   }
}
```

Exception Class Hierarchy

- Part of the Exception class hierarchy



Handling Exceptions

- Use a try/catch block when calling a risky method to tell the compiler that
  - You know an exceptional thing could happen in the method you are calling
  - You are prepared to handle it

try block

catch block

```
public void crossFinger() {
   try {
      anObject.takeRisk();
   } catch (BadException ex) {
      System.out.println("Aaargh!");
      ex.printStackTrace();
   }
}
```

If you cannot recover from the exception, at least get a stack trace using the printStackTrace() method that all exceptions inherit

- The compiler does not care how you handle the exception. All it cares is that you are taking care of it

Flow of Control in Try/Catch Blocks

- When you call a risky method, one of the 2 cases can happen
  - The risky method succeeds
    - The try block completes
    - The code in the catch block never runs
    - The method continues with the code below the catch block

```
try {
    x.doRiskyThing ();
    System.out.println("We did it!");
} catch (Exception ex) {
    System.out.println("Bummer...");
}
System.out.println("Out of the risky zone");
```

Flow of Control in Try/Catch Blocks

- When you call a risky method, one of the 2 cases can happen
  - The risky method throws an exception
    - The rest of the try block never runs
    - The catch block runs
    - The method continues with the code below the catch block

```
try {
    x.doRiskyThing ();
    System.out.println("We did it!");
} catch (Exception ex) {
    System.out.println("Bummer...");
}
System.out.println("Out of the risky zone");
```

Finally Block

- A finally block is where you put code that must run regardless of an exception

```
try {
   turnOnOven();
   x.bake(); // bake() throws BakingException
} catch (BakingException ex) {
   ex.printStackTrace();
} finally {
   turnOffOven();
}
```

finally block

- When the try or catch block completes, the finally block runs
- When the finally block completes, the rest of the method continues

Finally Block

- Even if the try or catch block has a return statement, the finally block will still run!
- Flow jumps to the finally block, then back to the return statement
- Example

```
try {
   turnOnOven();
   x.bake(); // bake() throws BakingException
} catch (BakingException ex) {
   ex.printStackTrace();
   return; // the finally block runs before the return statement
} finally {
   turnOffOven();
}
```

Unchecked Exceptions

- RuntimeExceptions are not checked by the compiler, and are known as "unchecked exceptions"
- Most RuntimeExceptions come from a problem in code logic (e.g., casting a reference into an incompatible type), rather than from a condition that fails at runtime in a way that one cannot predict or prevent
- The compiler does not bother checking whether a method declares that it throws a RuntimeException, or whether the caller acknowledges that it might get that exception at runtime (i.e., by using a try/catch block)
- A try/catch block is for handling exceptional situations. Use it to recover from situations you cannot guarantee will succeed, not flaws in your code
- Examples: NullPointerException, ArrayIndexOutOfBoundsException, ArithmeticException, IllegalArgumentException, NumberFormatException

Checked Exceptions

- All exceptions that are not a subclass of RuntimeException are checked by the compiler, and are known as "checked exceptions"
- Methods that might throw a checked exception must announce it with a throws-exception declaration
- If your code calls a checked-exception-throwing method, it must reassure the compiler that precautions have been taken
  - If you are prepared to handle the exception, wrap the call in a try block, and put your exception handling code in the catch block
  - Alternatively, you can make the compiler happy by officially 'ducking' the exception
- Examples: SQLException, IOException, ClassNotFoundException, InvocationTargetException

Ducking an Exception

- If you do not want to handle an exception, you can duck it by declaring it

```
public void foo throws ClothingException {
   // doLaundry() throws ClothingException
   laundry.doLaundry();
}
```

- When a method throws an exception, that method is popped off the stack immediately, and the exception is thrown to the next method down the stack (i.e., the caller)
- If the caller is a ducker, the ducker is popped off the stack immediately and the exception is thrown to the next method and so on…
- This repeats until some method in the calling chain handles the exception.

Ducking an Exception

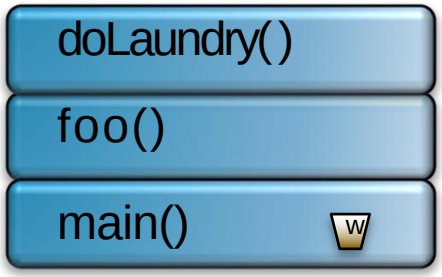
Example

```
public class Washer {
   Laundry laundry = new Laundry();

   public void foo()
               throws ClothingException {
      laundary.doLaundry();
   }

   public static void main(String[] args)
               throws ClothingException {
      Washer w = new Washer();
      w.foo();
   }
}
```

1. main() calls foo(), and foo() calls doLaundry(). doLaundry() is running and throws a ClothingException



Ducking an Exception

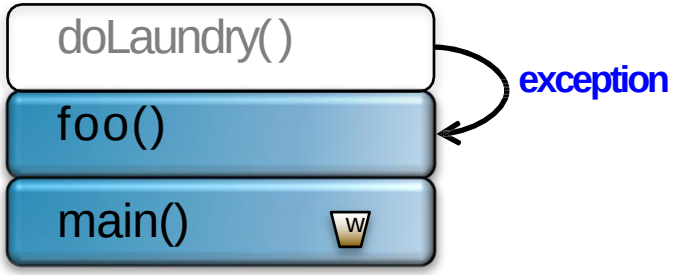
Example

```
public class Washer {
   Laundry laundry = new Laundry();

   public void foo()
               throws ClothingException {
      laundary.doLaundry();
   }

   public static void main(String[] args)
               throws ClothingException {
      Washer w = new Washer();
      w.foo();
   }
}
```

2 doLaundry() is popped off the stack immediately and the exception is thrown back to foo(). foo(), however, is a ducker and does not have a try/catch block



Ducking an Exception

— Example

```
public class Washer {
   Laundry laundry = new Laundry();

   public void foo()
               throws ClothingException {
      laundary.doLaundry();
   }

   public static void main(String[] args)
               throws ClothingException {
      Washer w = new Washer();
      w.foo();
   }
}
```

❸ foo() is popped off the stack and the exception is thrown back to main(). main() is also a ducker and does not have a try/catch block

Ducking an Exception

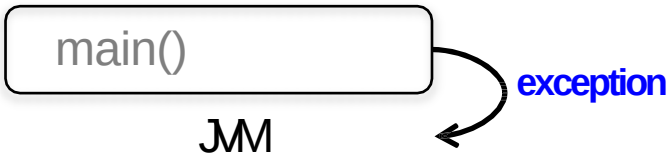
Example

```
public class Washer {
   Laundry laundry = new Laundry();

   public void foo()
               throws ClothingException {
      laundary.doLaundry();
   }

   public static void main(String[] args)
               throws ClothingException {
      Washer w = new Washer();
      w.foo();
   }
}
```

4 main() is popped off the stack and the exception is thrown back to the JVM. The JVM shuts down



Exception Rules

- You cannot have a catch or finally block without a try block

```
void go() {
   Foo f = new Foo();
   f.foof();
   catch (FooException ex) { }
}
```

This won't compile!

- You cannot put code between the try block and the catch block

```
try {
   x.doStuff();
}
int y = 43;
catch (Exception ex) { }
```

This won't compile!

Exception Rules

- A try block must be followed by either a catch or a finally block, or both

```
try {
   x.doStuff();
} finally {
   // cleanup
}
```

- A try block with only a finally block must still declare the exception

```
void go() throws FooException
   {
   try {
      x.doStuff();
   } finally { }
}
```

Catching Multiple Exceptions

- A method can throw multiple exceptions, and it must declare all the checked exceptions it can throw

```
public void doLaundry() throws PantsException, ShirtException {
   // code that could throw either exception
}
```

- The compiler will make sure that you have handled all the checked exceptions thrown by the method you are calling
- To handle multiple exceptions, simply stack the catch blocks under the try block, one after another

```
try {
   laundry.doLaundry();
} catch (PantsException pex) {
   // recovery from PantsException
} catch (ShirtException lex) {
   // recovery from ShirtException
}
```

Catching Multiple Exceptions

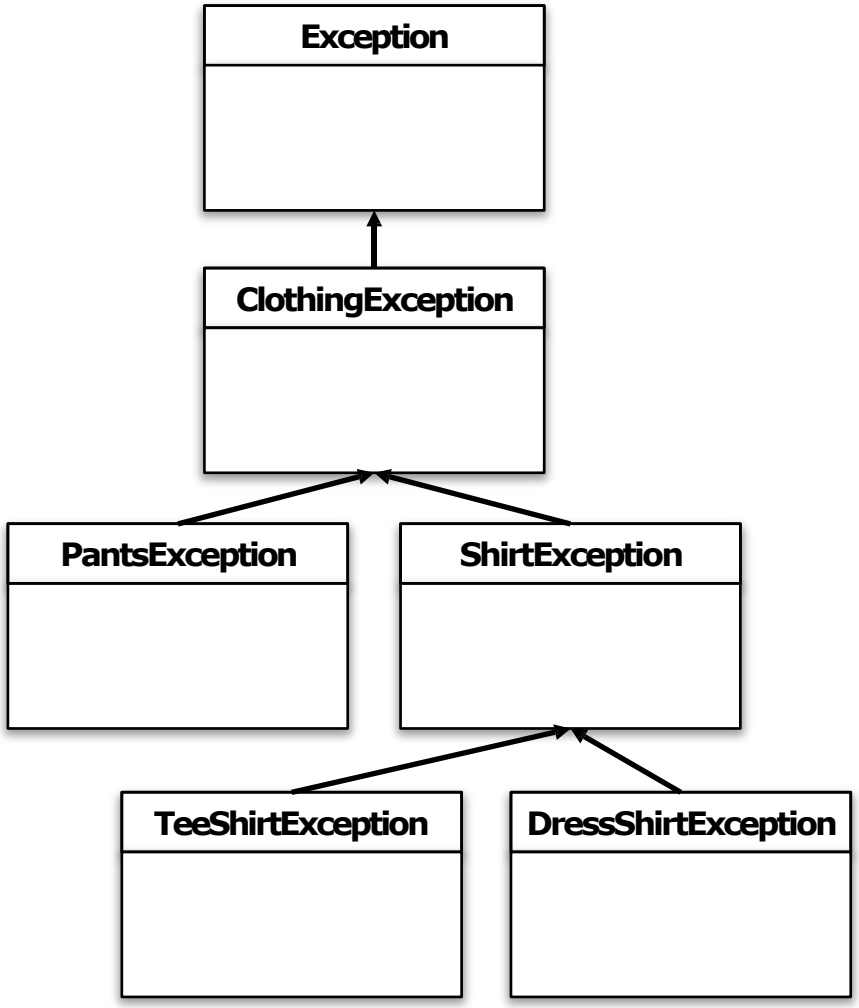
- When an exception is thrown by a method in the try block, the JVM simply starts at the first catch block and works its way down until it finds the first catch block that can handle the exception

```
try {
   anObject.riskyMethod();
} catch (Exception1 ex1) {
    // 1st catch block to check
} catch (Exception2 ex2) {
  // 2nd catch block to check
} catch (Exception3 ex3) {
  // 3rd catch block to check
} catch (Exception4 ex4) {
  // 4th catch block to check
}
```

Catching Multiple Exceptions

- If 2 or more exceptions have a common superclass, the method can declare just the superclass (i.e., polymorphic type)

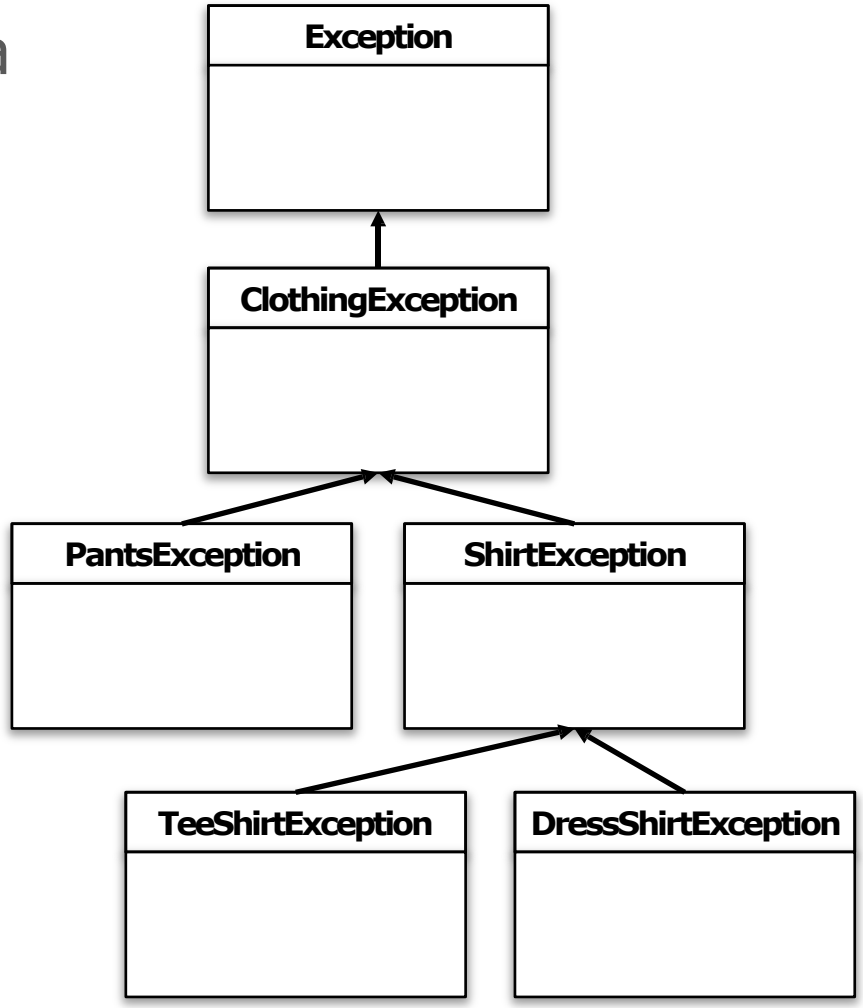
```
public void doLaundry()
            throws ClothingException {
   // code that could throw any
   // subclass of ClothingException
}
```



Catching Multiple Exceptions

- A catch block for catching a particular exception can also catch all subclasses of that exception (i.e., polymorphic type)

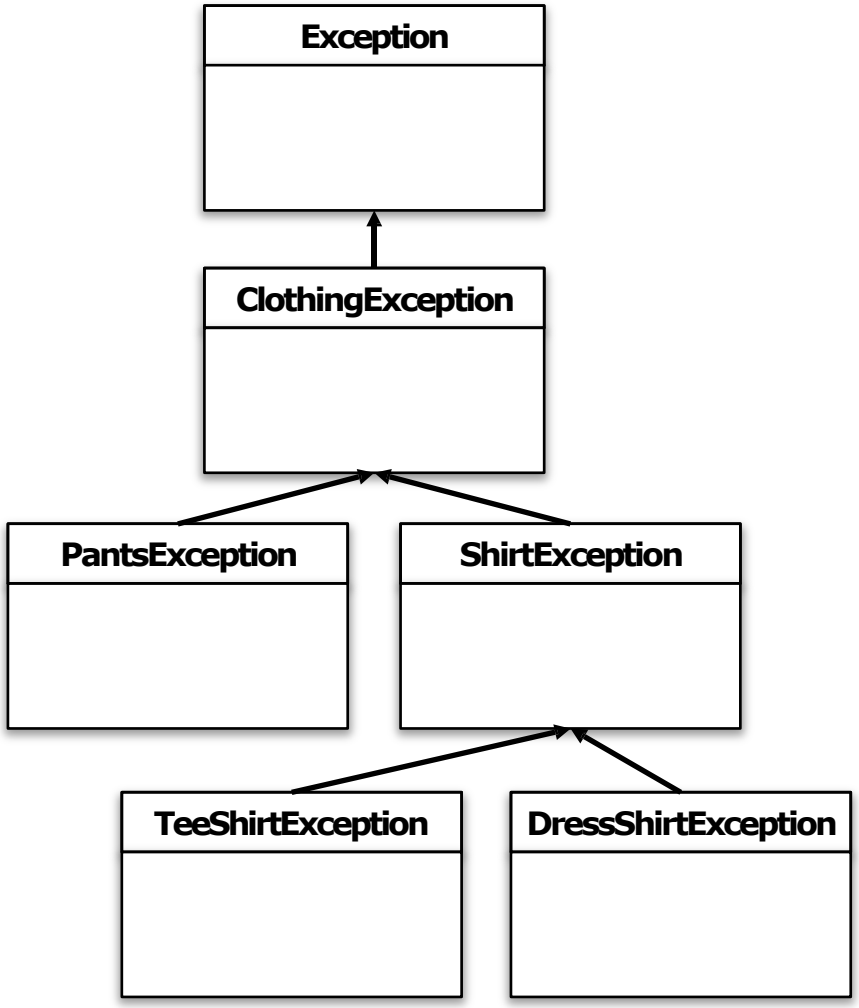
```
try {
   laundry.doLaundry();
} catch (ClothingException ex) {
   // recovery from any subclass
   // of ClothingException
}
```



Catching Multiple Exceptions

- Multiple catch blocks must therefore be ordered from catching the most specific exception to the most generic exception

```
try {
   laundry.doLaundry();
} catch (TeeShirtException tex) {
   // recovery from TeeShirtException
} catch (ShirtException pex) {
   // recovery from ShirtException
} catch (ClothingException ex) {
   // recovery from all others
}
```



Any problems?

If you encounter any problems in understanding this set of materials, **please feel free to contact me or my TAs**.
We are always with you!

Additional reference: https://howtodoinjava.com/java/exception-handling/checked-vs-unchecked-exceptions-in-java/

Chapter 12.

End

2020-2021
COMP2396 Object-Oriented Programming and Java
Dr. T.W. Chim (E-mail: twchim@cs.hku.hk)
Department of Computer Science, The University of Hong Kong