Head First Java

Chapter-11:
Risky Behavior

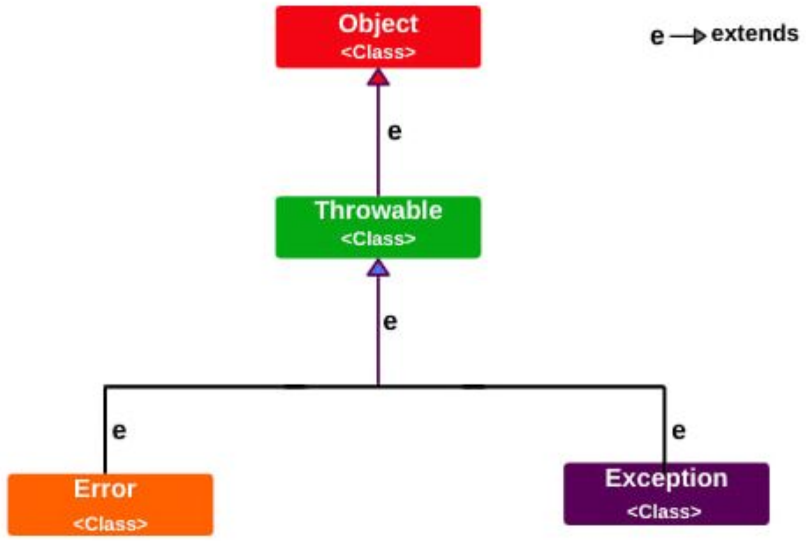
Upcode Software
Engineer Team

-2023-

CONTENT

What is Exception (1/n)

- In Java, Exception is an **unwanted** or **unexpected** event, which occurs during the execution of a program, i.e. at run time, that disrupts the normal flow of the program's instructions
- Exceptions **can be caught** and **handled** by the program.



Exception Handling in Java (1/n)

- **Exception Handling** is a mechanism to handle runtime errors
- **It is** an object which is thrown at runtime

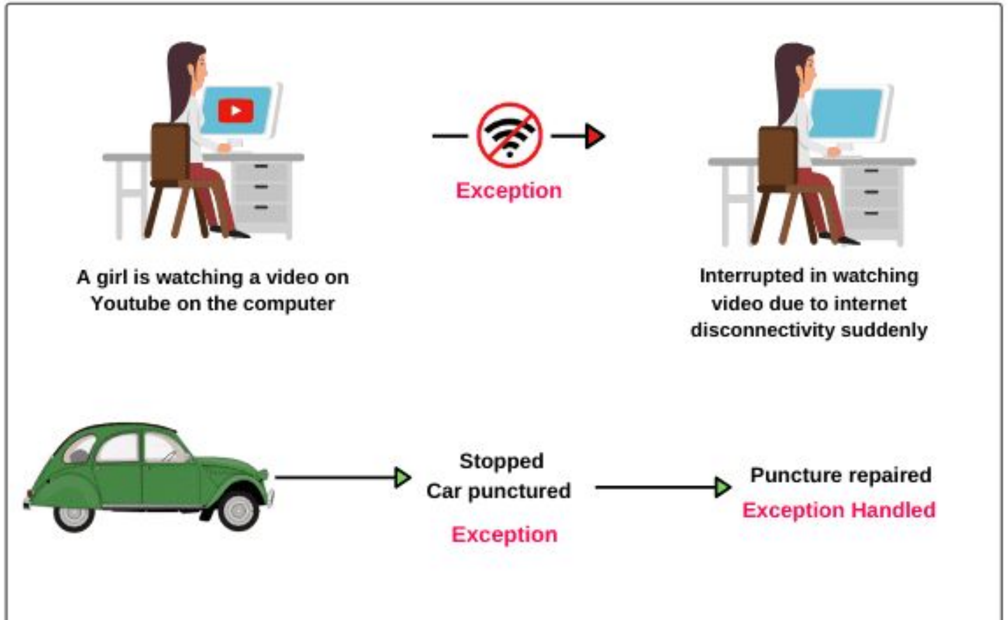
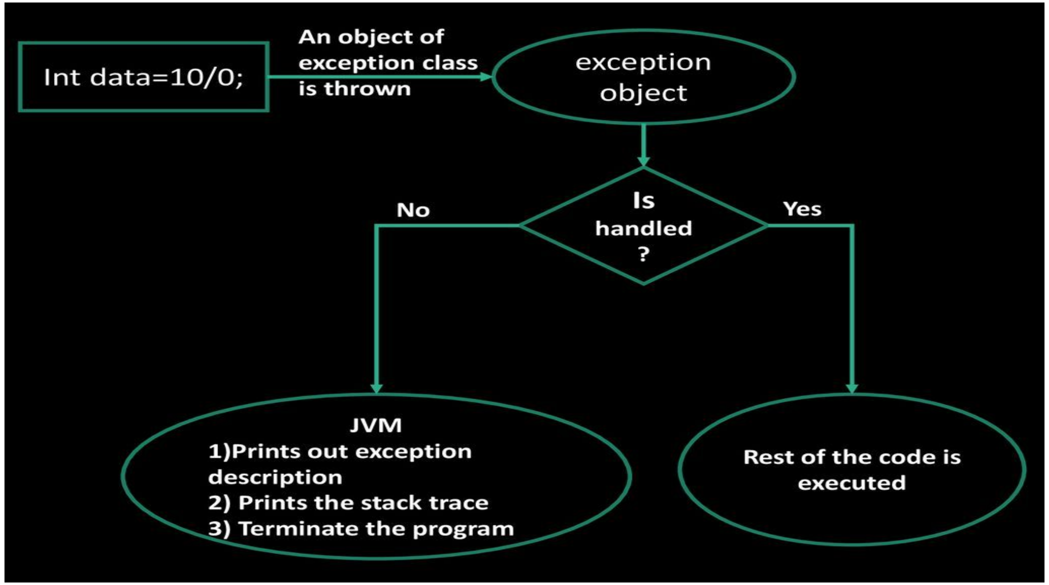

A girl is watching a video on Youtube on the computer
Exception
Interrupted in watching video due to internet disconnectivity suddenly
Stopped
Car punctured
Exception
Puncture repaired
Exception Handled

Fig: Realtime Example of Exception Handling

Exception Handling in Java (2/n)

- Exception handling done with the **exception object**

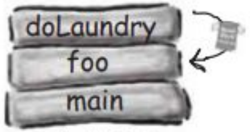

Int data=10/0;
An object of exception class is thrown
exception object
Is handled ?
No
Yes
JVM
1)Prints out exception description
2) Prints the stack trace
3) Terminate the program
Rest of the code is executed

Exception Handling in Java (3/n)

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

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

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

Both methods duck the exception (by declaring it) so there's nobody to handle it! This compiles just fine.

1 doLaundry() throws a ClothingException

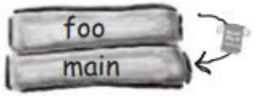


main() calls foo()

foo() calls doLaundry()

doLaundry() is running and throws a ClothingException

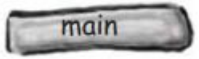
2 foo() ducks the exception



doLaundry() pops off the stack immediately and the exception is thrown back to foo().

But foo() doesn't have a try/catch, so...

3 main() ducks the exception



foo() pops off the stack immediately and the exception is thrown back to... who? What? There's nobody left but the JVM, and it's thinking, "Don't expect ME to get you out of this."

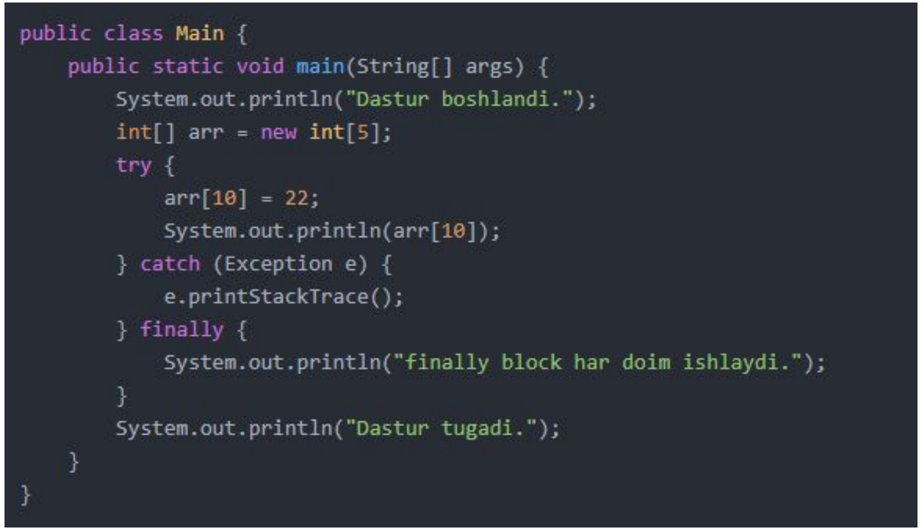
4 The JVM shuts down

Exception Handling in Java (3/n)

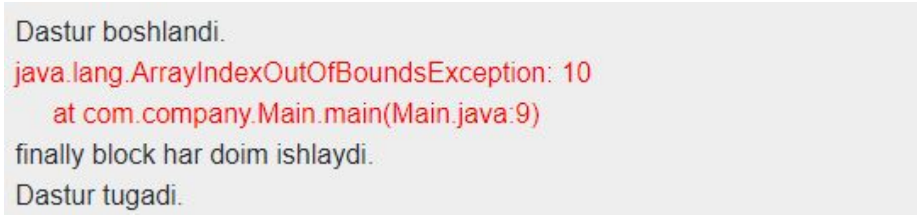
Exception Handling Terms

- **try** - used to enclose a segment of code that may produce a exception
- **catch** - placed directly after the try block to handle one or more exception types
- **finally** - optional statement used after a try-catch block to run a segment of code regardless if a exception is generated

Exception Handling in Java (4/n)

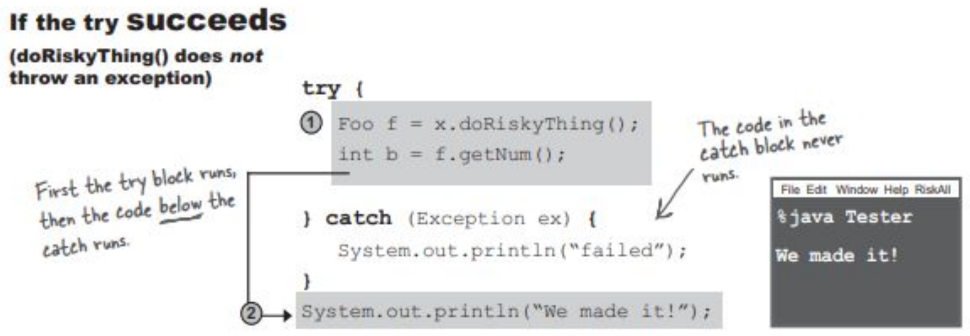


```java
public class Main {
    public static void main(String[] args) {
        System.out.println("Dastur boshlandi.");
        int[] arr = new int[5];
        try {
            arr[10] = 22;
            System.out.println(arr[10]);
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            System.out.println("finally block har doim ishlaydi.");
        }
        System.out.println("Dastur tugadi.");
    }
}
```



```
Dastur boshlandi.
java.lang.ArrayIndexOutOfBoundsException: 10
    at com.company.Main.main(Main.java:9)
finally block har doim ishlaydi.
Dastur tugadi.
```

Exception Handling in Java (5/n)



If the try succeeds

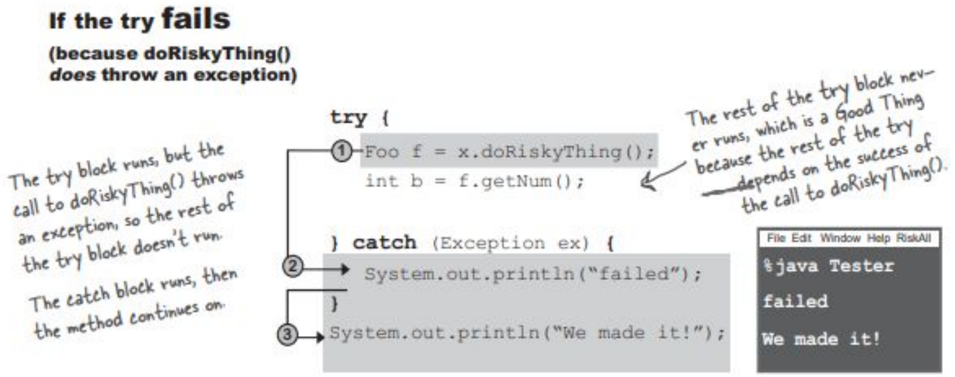
(doRiskyThing() does *not* throw an exception)

```
try {
①  Foo f = x.doRiskyThing();
   int b = f.getNum();

} catch (Exception ex) {
   System.out.println("failed");
}
② System.out.println("We made it!");
```

First the try block runs, then the code below the catch runs.

The code in the catch block never runs.

```
File Edit Window Help RiskAll
%java Tester
We made it!
```



If the try fails

(because doRiskyThing() *does* throw an exception)

```
try {
①  Foo f = x.doRiskyThing();
   int b = f.getNum();

} catch (Exception ex) {
②  System.out.println("failed");
}
③ System.out.println("We made it!");
```

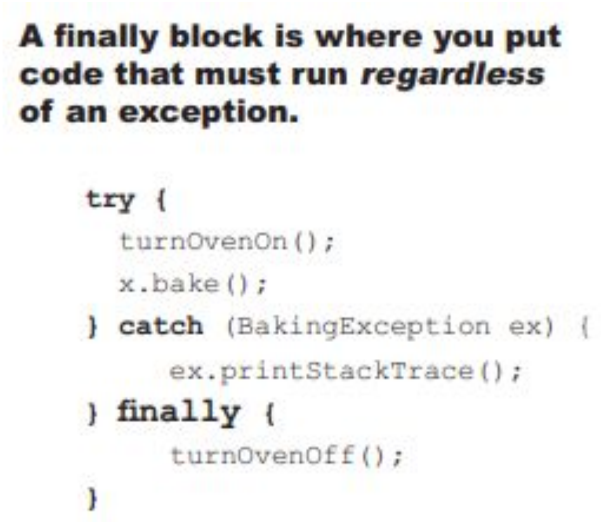
The try block runs, but the call to doRiskyThing() throws an exception, so the rest of the try block doesn't run.

The catch block runs, then the method continues on.

The rest of the try block never runs, which is a Good Thing because the rest of the try depends on the success of the call to doRiskyThing().

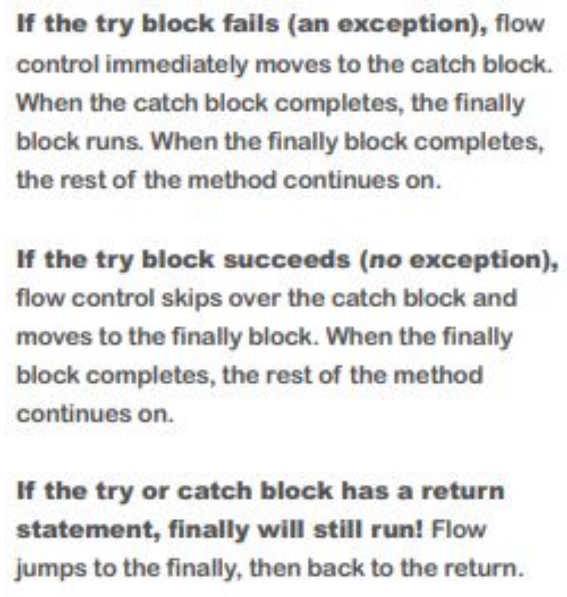
```
File Edit Window Help RiskAll
%java Tester
failed
We made it!
```

Exception Handling in Java (6/n)



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

```
try {
  turnOvenOn();
  x.bake();
} catch (BakingException ex) {
     ex.printStackTrace();
} finally {
     turnOvenOff();
}
```

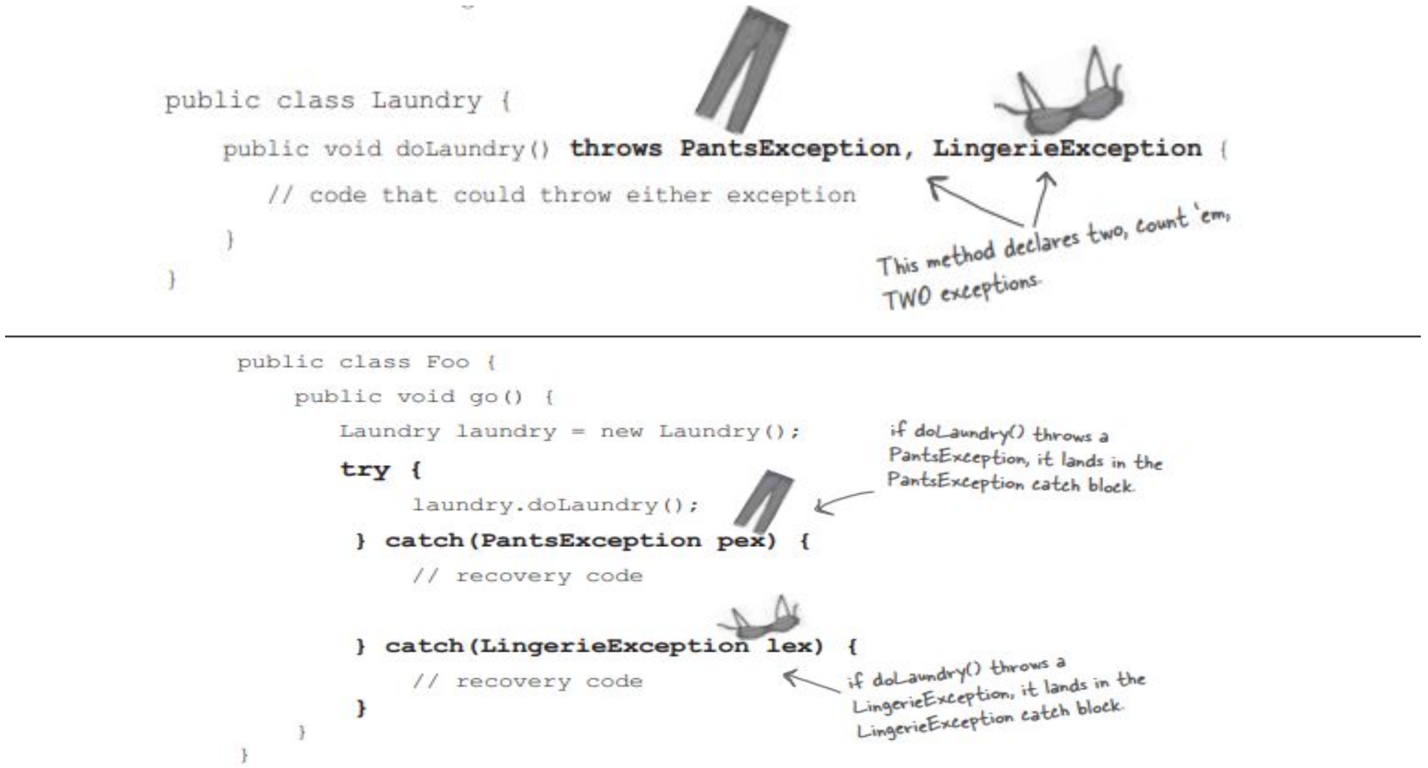


If the try block fails (an exception), flow control immediately moves to the catch block. When the catch block completes, the finally block runs. When the finally block completes, the rest of the method continues on.

If the try block succeeds (*no* exception), flow control skips over the catch block and moves to the finally block. When the finally block completes, the rest of the method continues on.

If the try or catch block has a return statement, finally will still run! Flow jumps to the finally, then back to the return.

Exception Handling in Java (7/n)



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

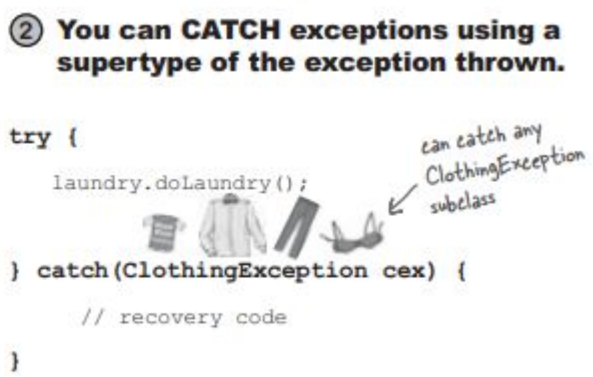
This method declares two, count 'em, TWO exceptions.

```
public class Foo {
    public void go() {
        Laundry laundry = new Laundry();
        try {
            laundry.doLaundry();
         } catch(PantsException pex) {
            // recovery code
         } catch(LingerieException lex) {
            // recovery code
         }
    }
}
```

if doLaundry() throws a PantsException, it lands in the PantsException catch block.

if doLaundry() throws a LingerieException, it lands in the LingerieException catch block.

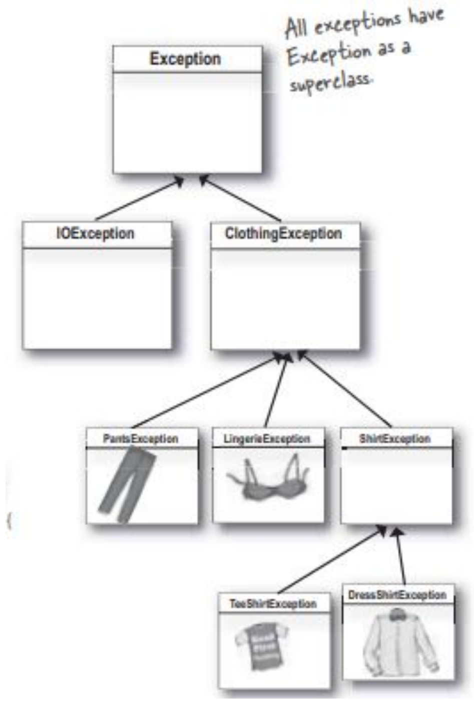
Exception Handling in Java (8/n)



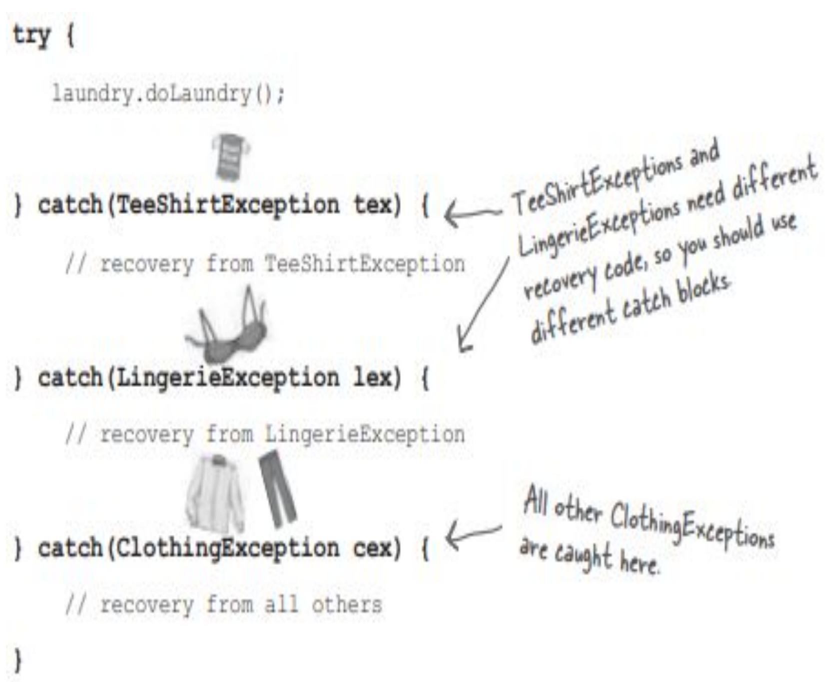
② **You can CATCH exceptions using a supertype of the exception thrown.**

```
try {

    laundry.doLaundry();

} catch(ClothingException cex) {

      // recovery code

}
```

can catch any ClothingException subclass



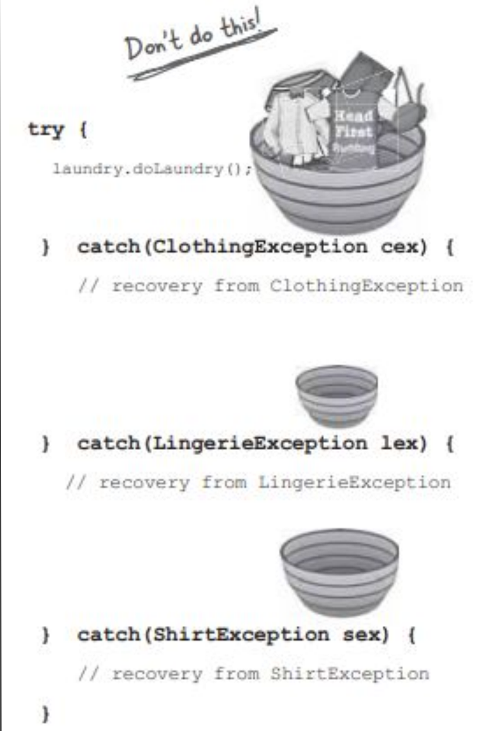
Exception Handling in Java (9/n)



```
try {
   laundry.doLaundry();

} catch(TeeShirtException tex) {
   // recovery from TeeShirtException

} catch(LingerieException lex) {
   // recovery from LingerieException

} catch(ClothingException cex) {
   // recovery from all others
}
```

TeeShirtExceptions and LingerieExceptions need different recovery code, so you should use different catch blocks.

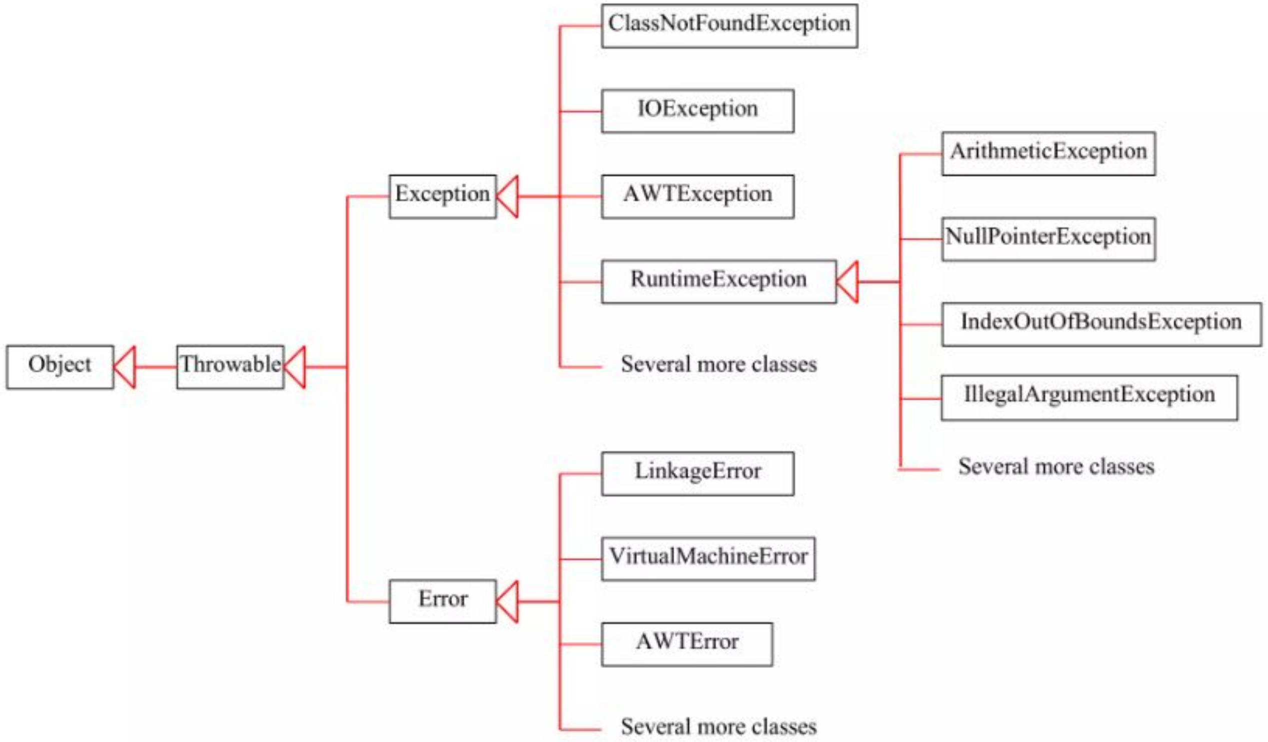
All other ClothingExceptions are caught here.



Don't do this!

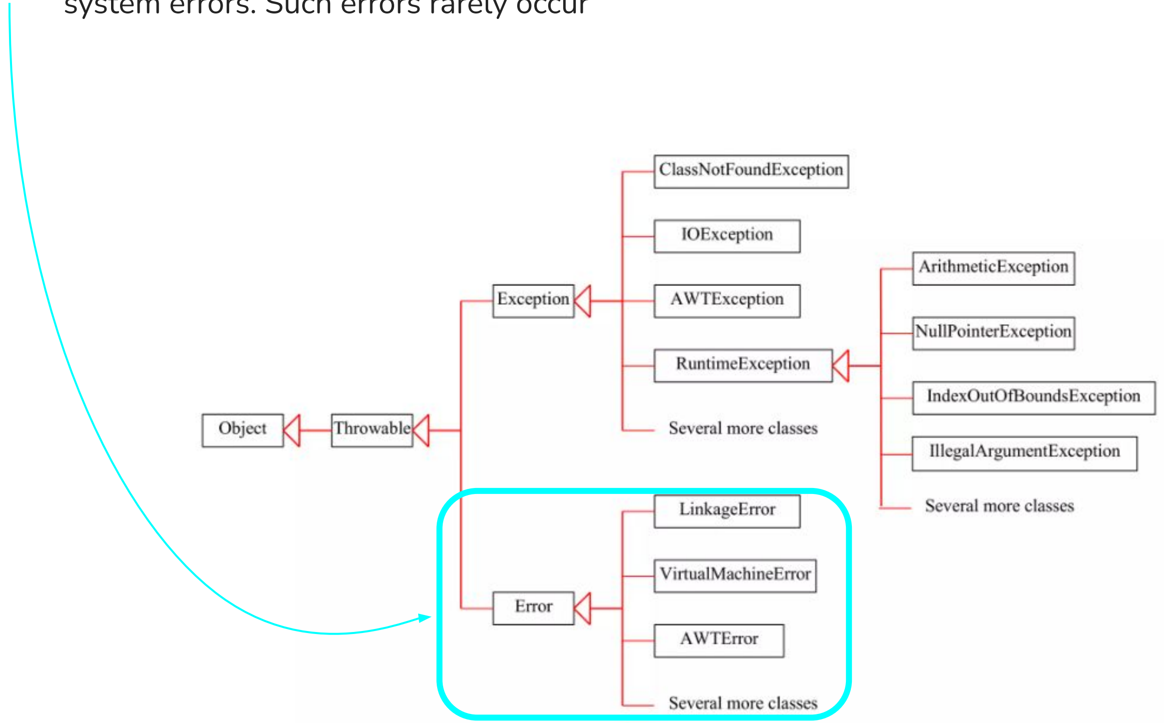
```
try {
  laundry.doLaundry();

 }  catch(ClothingException cex) {
    // recovery from ClothingException

 }  catch(LingerieException lex) {
   // recovery from LingerieException

 }  catch(ShirtException sex) {
    // recovery from ShirtException
 }
```

Java Exception Hierarchy(1/n)



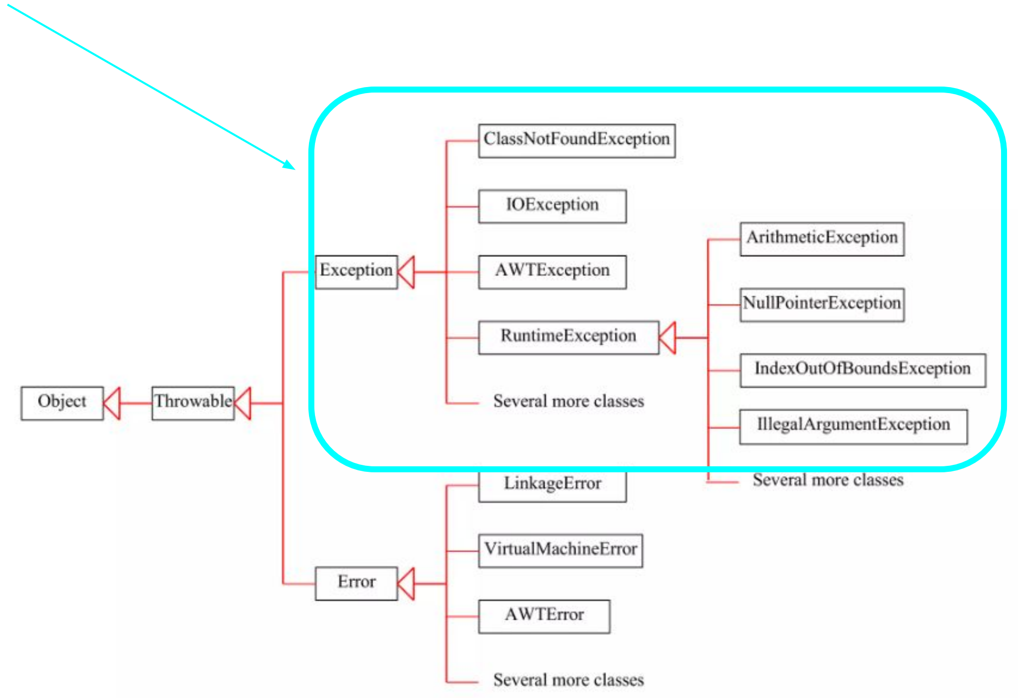
Java Exception Hierarchy(2/n)

- **System errors** are thrown by JVM and represented in the **Error** class. The **Error** class describes internal system errors. Such errors rarely occur



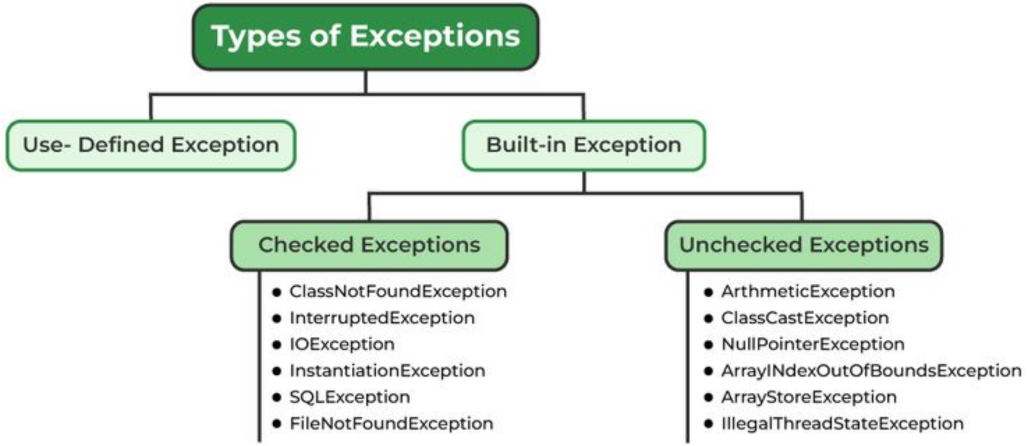
Java Exception Hierarchy(3/n)

- **Exception** describes errors caused by your program and external circumstances. These errors can be caught and handled by your program.



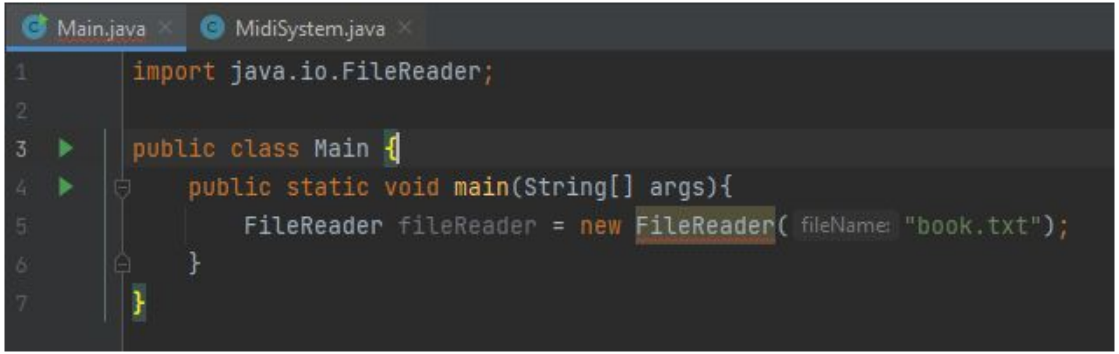
What are the Types of Exception?(1/n)

- **Checked exceptions:** These are the exceptions that are checked by the compiler at compile time. If a method throws a checked exception, then the caller of the method must either handle the exception or declare it in the throws clause.
- **Unchecked exceptions:** These are the exceptions that are not checked by the compiler at compile time. They include runtime exceptions and errors.



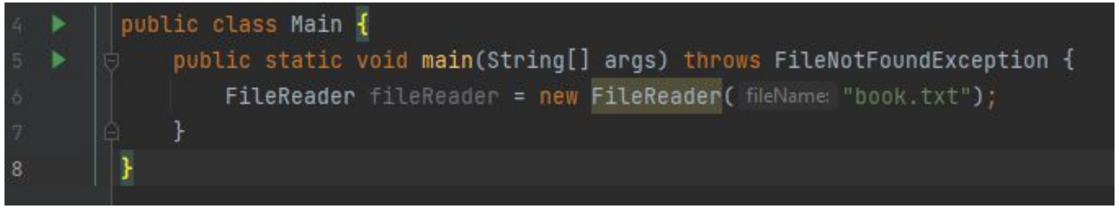
What are the Types of Exception?(2/n)

example of checked exception:



Main.java MidiSystem.java

```
import java.io.FileReader;

public class Main {
    public static void main(String[] args){
        FileReader fileReader = new FileReader( fileName: "book.txt");
    }
}
```

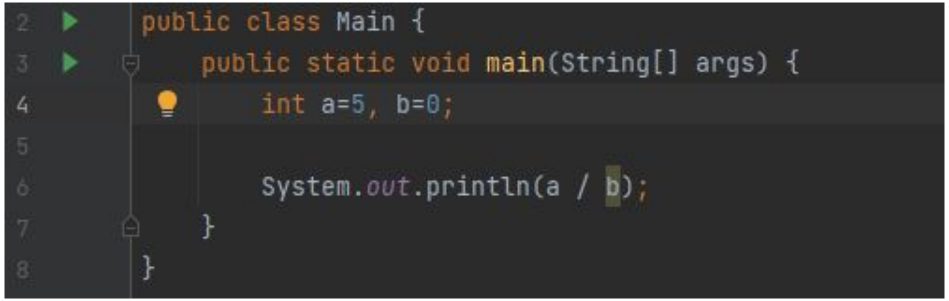


```
public class Main {
    public static void main(String[] args) throws FileNotFoundException {
        FileReader fileReader = new FileReader( fileName: "book.txt");
    }
}
```

What are the Types of Exception?(3/n)

example of unchecked exception:

- This code will be **passed compile successfully.** The compiler does not force the code to **try-catch** because the expression a/b may **throw an Exception at runtime**

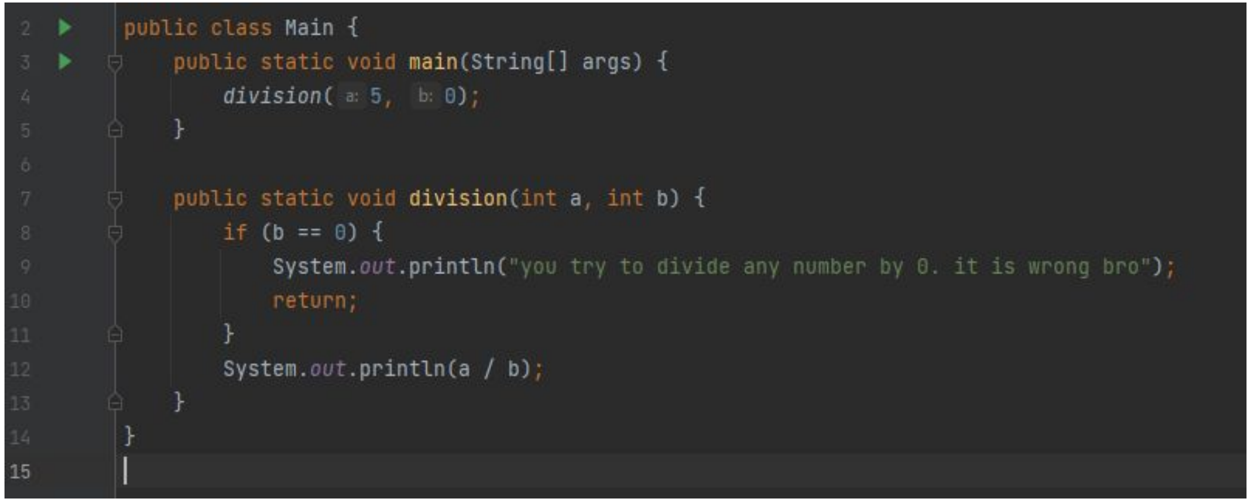


```java
public class Main {
    public static void main(String[] args) {
        int a=5, b=0;

        System.out.println(a / b);
    }
}
```

What are the Types of Exception?(4/n)

example of unchecked exception:

- **Unchecked Exceptions** are exceptions that **can be avoided**. That is, the programmer can prevent errors by checking the **incoming parameters** while writing the code. For example, dividing a by b. A possible ArithmeticException can be avoided by checking that b is not equal to 0.



```java
public class Main {
    public static void main(String[] args) {
        division( a: 5,  b: 0);
    }

    public static void division(int a, int b) {
        if (b == 0) {
            System.out.println("you try to divide any number by 0. it is wrong bro");
            return;
        }
        System.out.println(a / b);
    }
}
```

Reference Resource?

1. **Head First book**
2. Java Exception in uzbek
3. Exception Handling Spring Boot REST API

Thank you!

Presented by

Jamshid Erkinov

(jamshiderkinov19992206@gmial.com)