Lecture # 4

Imran Ahsan

Outline

- **Exception Handling**
- Exception Types
- try, catch, throw, throws, finally
- Java's Built-in Exceptions
- User Defined Exceptions

Exception Handling

Exception Handling

- A Java exception is an object that describes an exceptional (that is, error) condition that has occurred in a piece of code.

- Exceptions can be generated by the Java run-time system, or they can be manually generated by your code.
- Exceptions thrown by Java relate to fundamental errors that violate the rules of the Java language or the constraints of the Java execution environment.
- Manually generated exceptions are typically used to report some error condition to the caller of a method.

- Java exception handling is managed via five keywords: **try, catch, throw, throws, and finally.**

Exception Handling

- Program statements that you want to monitor for exceptions are contained within a **try** block.
- If an exception occurs within the **try** block, it is thrown.
- Your code can catch this exception (using **catch**) and handle it in some rational manner.

- System-generated exceptions are automatically thrown by the Java run-time system.
- To manually throw an exception, use the keyword **throw** .
- Any exception that is thrown out of a method must be specified as such by a **throws** clause.
- Any code that absolutely must be executed before a method returns is put in a **finally** block.

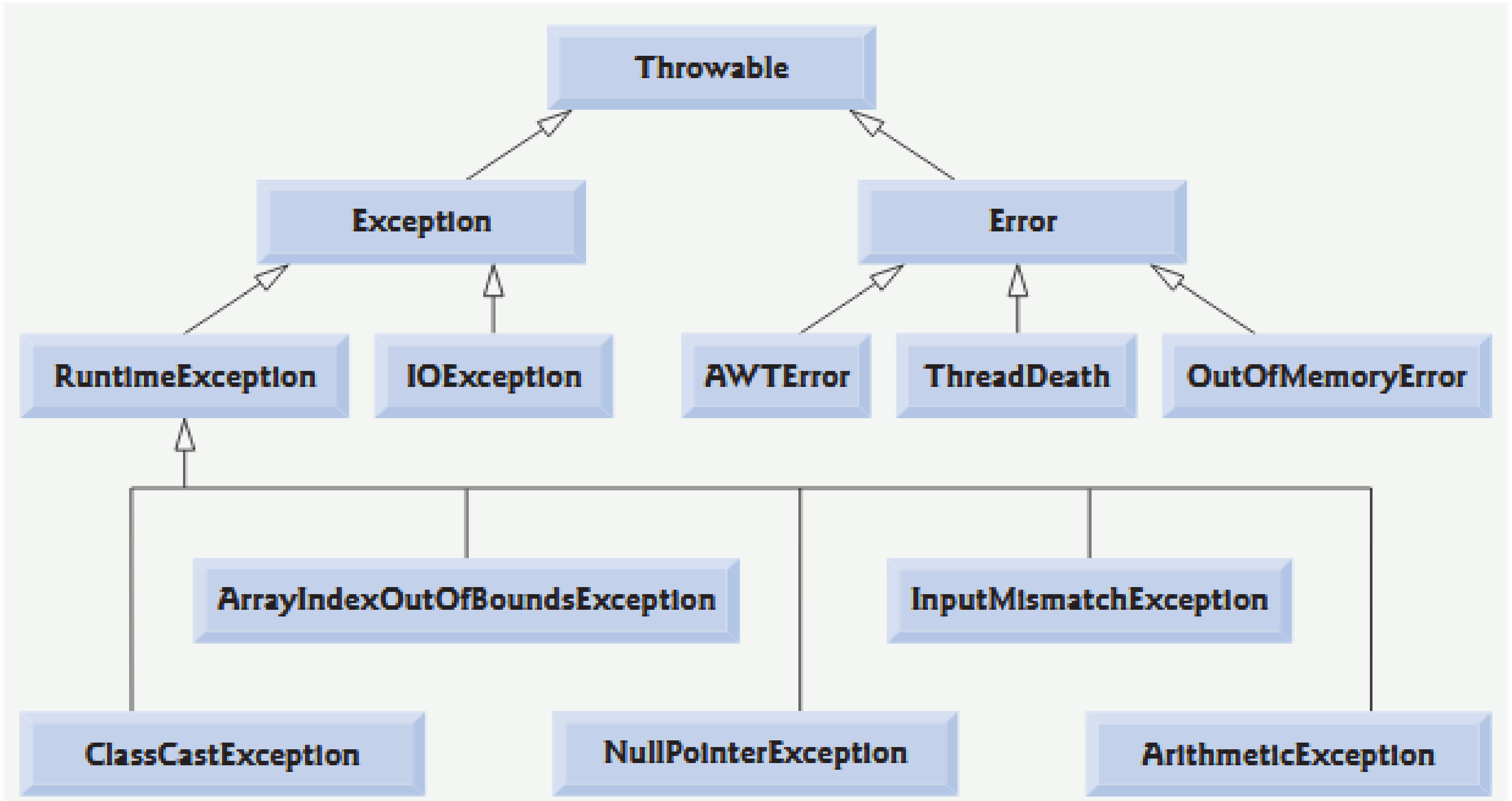
Exception Handling

```
Student {

   try {


   }
   catch  (ExceptionType1 exOb)  {   doSomething;  }
   catch  (ExceptionType2 exOb)  {   doSomething;  }
   catch  (ExceptionType3 exOb)  {   doSomething;  }
   catch  (ExceptionType4 exOb)  {   doSomething;  }
   catch  (ExceptionType5 exOb)  {   doSomething;  }
}
```

Exception Handling



Exception Types

- All exception types are subclasses of the built-in class Throwable. Thus, Throwable is at the top of the exception class hierarchy. Immediately below Throwable are two subclasses that partition exceptions into two distinct branches.
- One branch is headed by **Exception** . This class is used for exceptional conditions that user programs should catch. This is also the class that you will subclass to create your own custom exception types.
- There is an important subclass of Exception , called RuntimeException. Exceptions of this type are automatically defined for the programs that you write and include things such as division by zero and invalid array indexing.
- The other branch is topped by **Error**, which defines exceptions that are not expected to be caught under normal circumstances by your program. Exceptions of type Error are used by the Java run-time system to indicate errors having to do with the run-time environment, itself. Stack overflow is an example of such an error.

Why Exception Handling?

```
class NewClass{
public static void main(String args[]) {
int d = 0;
int a = 42 / d;
}
}
```

Output:

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
at javaapplication1.NewClass.main(NewClass.java:46)
```

try and **catch** block

- The `try` block
  - is used to wrap or enclose code where exceptions may occur
- The `catch` block
  - A method can catch an exception by providing an exception handler for that type of exception
  - Typically, a catch block is created for each type of exception that can occur.

Using **try** and **catch** block

```
class NewClass {
public static void main(String args[]) {
int d, a;
try { // monitor a block of code.
        d = 0;
        a = 42 / d;
        System.out.println("This will not be printed.");
}
catch (ArithmeticException e) { // catch divide-by-zero error
        System.out.println("Division by zero.");
}
System.out.println("After catch statement.");
}
}
```

Multiple **catch** Clauses

- In some cases, more than one exception could be raised by a single piece of code.
- To handle this type of situation, you can specify two or more catch clauses, each catching a different type of exception.
- When an exception is thrown, each catch statement is inspected in order, and the first one whose type matches that of the exception is executed.
- After one catch statement executes, the others are bypassed, and execution continues after the try/catch block.

Multiple **catch** Clauses

```
class NewClass {
public static void main(String args[]) {
try {
        int a = 1;
        System.out.println("a = " + a);
        int b = 42 / a;
        int c[] = { 1 };
        c[42] = 99;
} catch(ArithmeticException e) {
        System.out.println("Divide by 0: " + e);
} catch(ArrayIndexOutOfBoundsException e) {
        System.out.println("Array index out of bound: " + e);
}
System.out.println("After try/catch blocks.");
}
}
```

throw

- So far, you have only been catching exceptions that are thrown by the Java run-time system. However, it is possible for your program to throw an exception explicitly, using the throw statement. The general form of throw is shown here:

```
throw ThrowableInstance;
```

- ThrowableInstance must be an object of type Throwable or a subclass of Throwable.

throw

```
// Demonstrate throw.
class ThrowDemo {
static void demoproc() {
try {
        throw new NullPointerException("demo");
} catch(NullPointerException e) {
        System.out.println("Caught inside demoproc.");
}
}
public static void main(String args[]) {
try {
        demoproc();
} catch(NullPointerException e) {
        System.out.println("Recaught: " + e);
}
}
}
```

Using Exceptions

- The **`throw`** clause
  - Exceptions originate within a method or block of code using the *`throw`* clause

```
throw someThrowableObject;
```

- The **`throws`** clause
  - Specifies that the method can throw Exception(s)

```
public Object pop()
   throws EmptyStackException {}
```

throws

```
// Demonstrate throw.
class ThrowDemo {
static void throwOne() throws IllegalAccessException {
System.out.println("Inside throwOne.");
throw new IllegalAccessException("demo");
}
public static void main(String args[]) {
try {
throwOne();
} catch (IllegalAccessException e) {
System.out.println("Caught " + e);
}
}
}
```

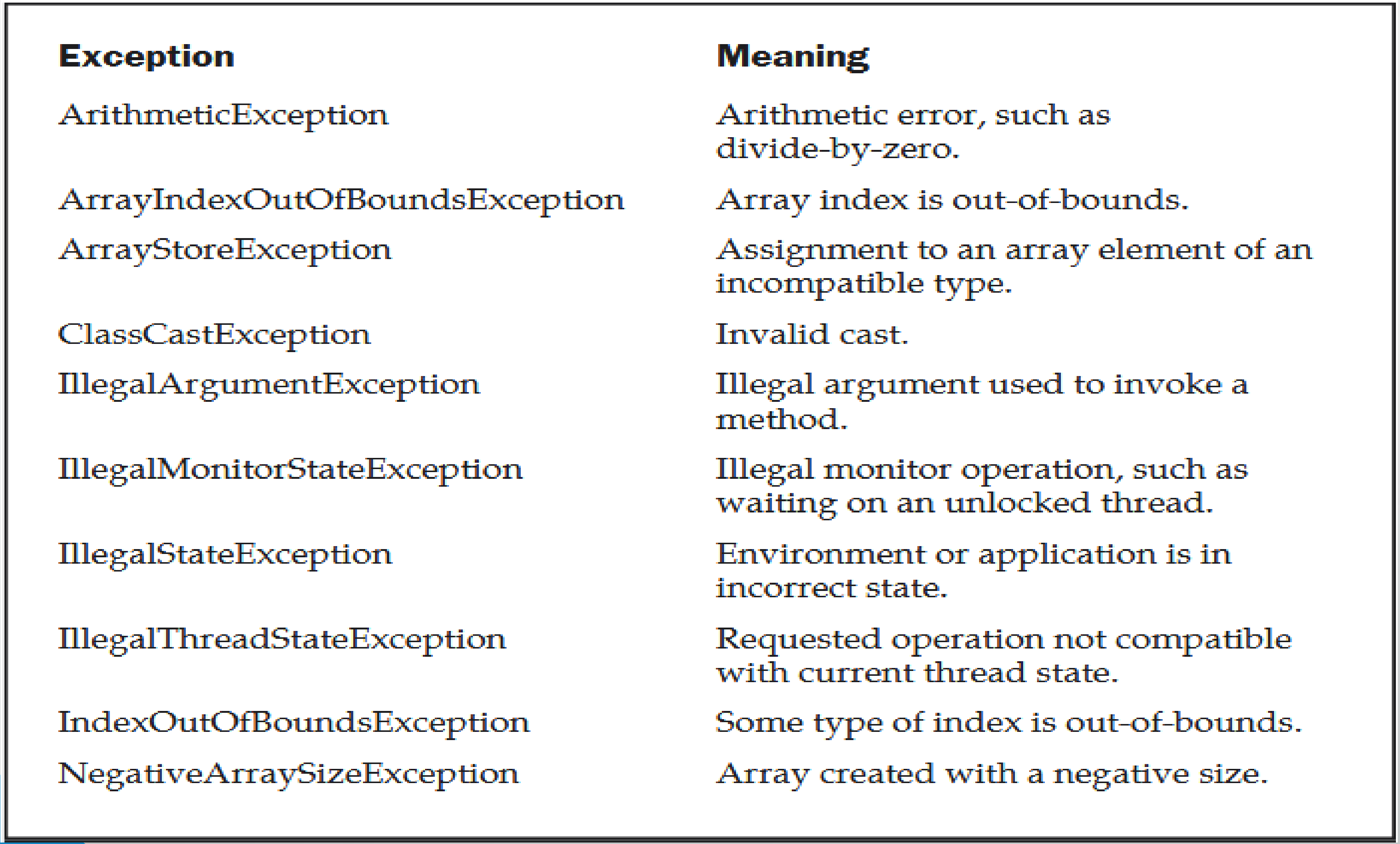
finally

- When exceptions are thrown, execution in a method takes a rather abrupt, nonlinear path that alters the normal flow through the method. Depending upon how the method is coded, it is even possible for an exception to cause the method to return prematurely. This could be a problem in some methods. For example, if a method opens a file upon entry and closes it upon exit, then you will not want the code that closes the file to be bypassed by the exception-handling mechanism. The finally keyword is designed to address this contingency.

finally

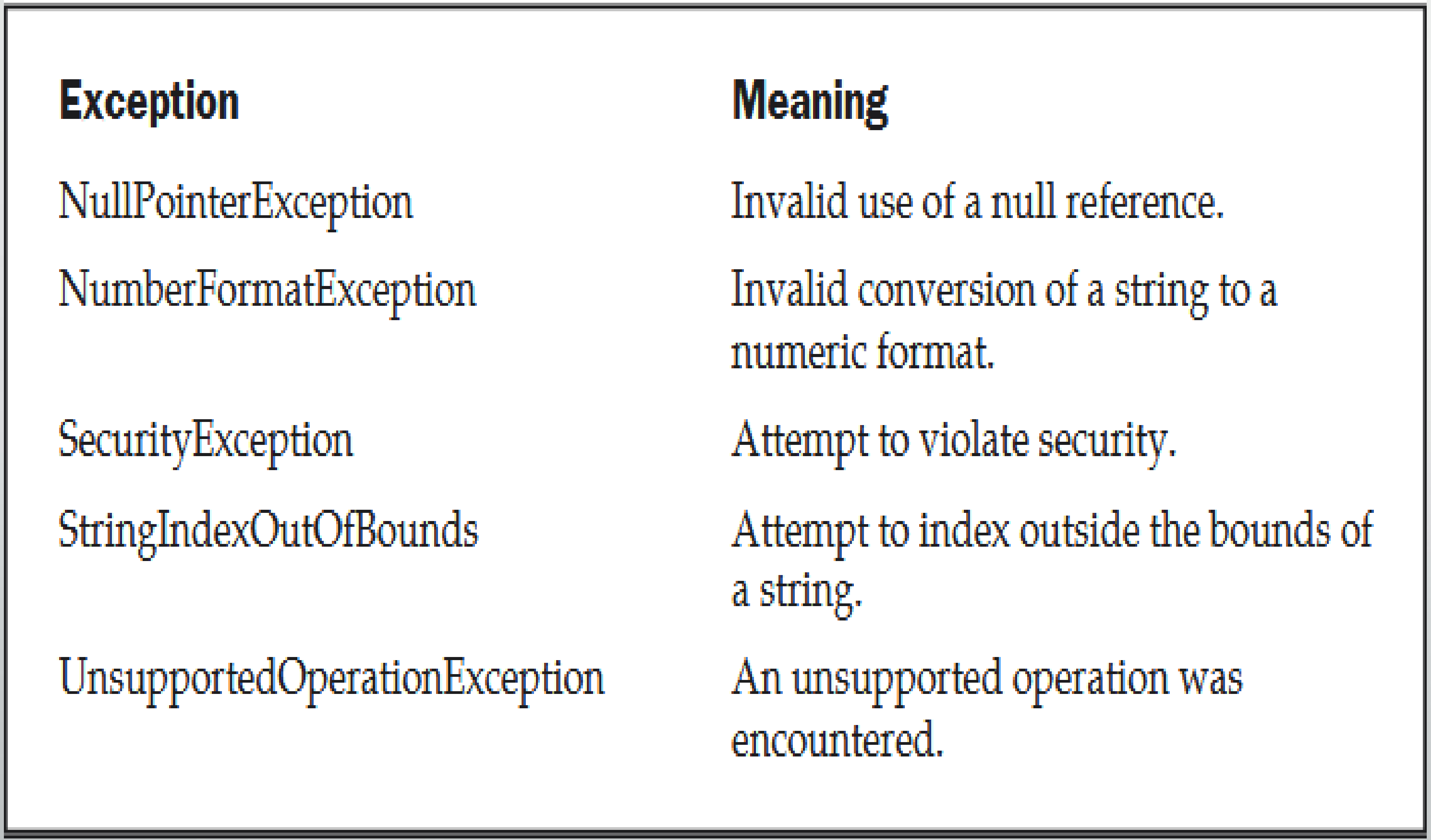
- Finally creates a block of code that will be executed after a try / catch block has completed and before the code following the try/catch block. The finally block will execute whether or not an exception is thrown. If an exception is thrown, the finally block will execute even if no catch statement matches the exception. Any time a method is about to return to the caller from inside a try/catch block, via an uncaughtexception or an explicit return statement, the finally clause is also executed just before the method returns. This can be useful for closing file handles and freeing up any other resources that might have been allocated at the beginning of a method with the intent of disposing of them before returning. The finally clause is optional. However, each try statement requires at least one catch or a finally clause.

Java's Built-in Exceptions



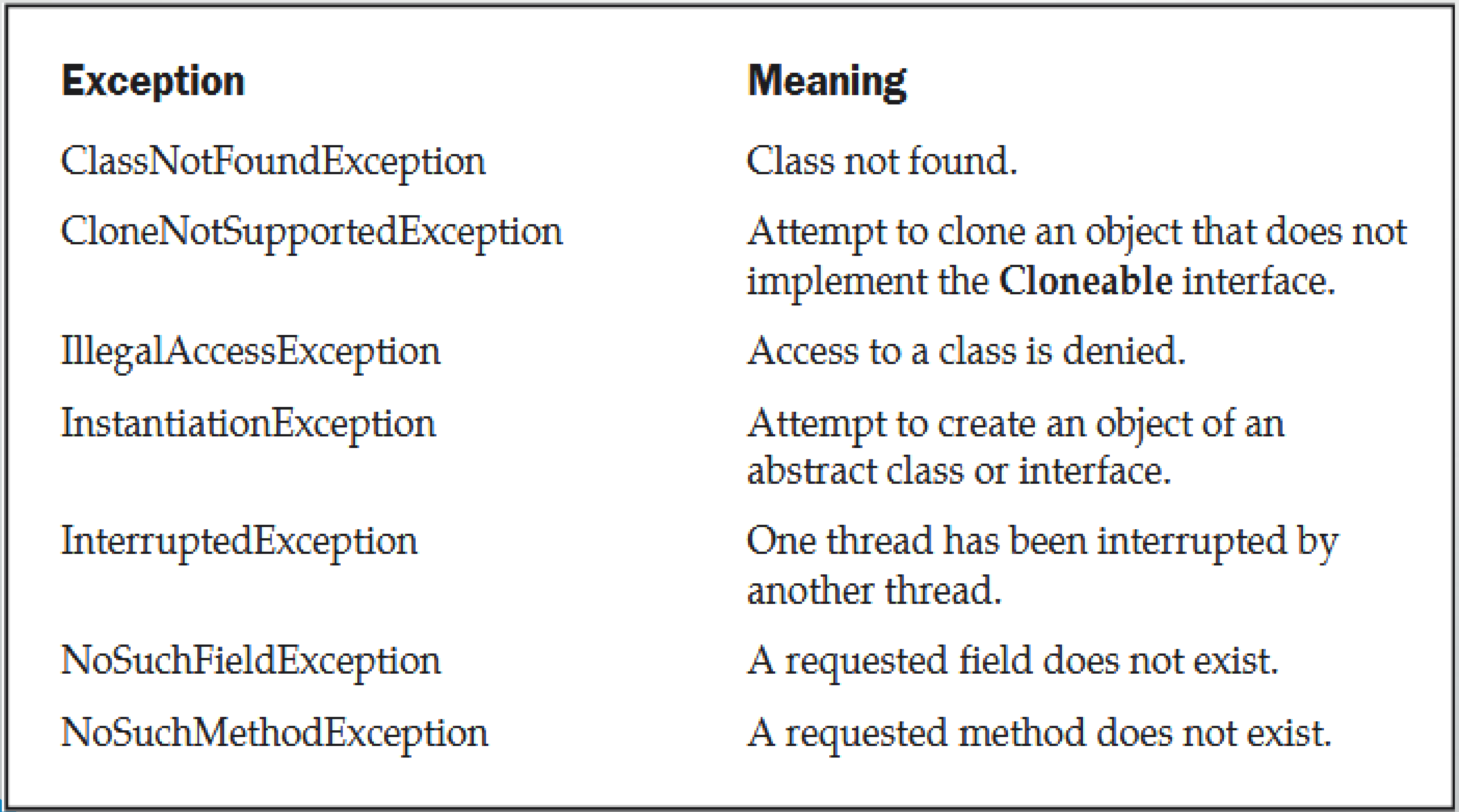
Exception	Meaning
ArithmeticException	Arithmetic error, such as divide-by-zero.
ArrayIndexOutOfBoundsException	Array index is out-of-bounds.
ArrayStoreException	Assignment to an array element of an incompatible type.
ClassCastException	Invalid cast.
IllegalArgumentException	Illegal argument used to invoke a method.
IllegalMonitorStateException	Illegal monitor operation, such as waiting on an unlocked thread.
IllegalStateException	Environment or application is in incorrect state.
IllegalThreadStateException	Requested operation not compatible with current thread state.
IndexOutOfBoundsException	Some type of index is out-of-bounds.
NegativeArraySizeException	Array created with a negative size.

Java’s Built-in Exceptions



Exception	Meaning
NullPointerException	Invalid use of a null reference.
NumberFormatException	Invalid conversion of a string to a numeric format.
SecurityException	Attempt to violate security.
StringIndexOutOfBounds	Attempt to index outside the bounds of a string.
UnsupportedOperationException	An unsupported operation was encountered.

Java's Built-in Exceptions



Exception	Meaning
ClassNotFoundException	Class not found.
CloneNotSupportedException	Attempt to clone an object that does not implement the **Cloneable** interface.
IllegalAccessException	Access to a class is denied.
InstantiationException	Attempt to create an object of an abstract class or interface.
InterruptedException	One thread has been interrupted by another thread.
NoSuchFieldException	A requested field does not exist.
NoSuchMethodException	A requested method does not exist.

End.