Java Enterprise Application Development

Lecture 6
Exceptions

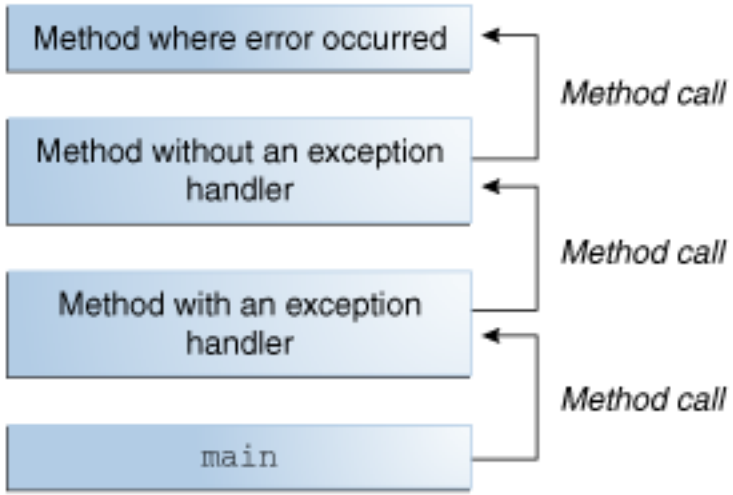
Dr. Fan Hongfei

15 October 2025

Introduction

- **Exception**: an event, which occurs during the execution of a program, that **disrupts** the normal flow of the program's instructions
- **Exception object**: containing **information** about the error, including its type and the state of the program when the error occurred
- **Throwing an exception**: **creating** an exception object and **handing** it to the runtime system

Introduction (cont.)



- The runtime system searches the **call stack** for a method that contains a block of code that can handle the exception: **Exception Handler**

- The exception handler chosen is said to **catch** the exception
- If the runtime system exhaustively searches all the methods on the call stack without finding an appropriate exception handler, the runtime system **terminates**

The Catch or Specify Requirement

- Code that might throw certain exceptions must be enclosed by either of the following
  - A **try statement** that catches the exception
    - Providing a handler for the exception
  - A method that **specifies that it can throw the exception**
    - Providing a throws clause that lists the exception
- **Not all exceptions** are subject to the Catch or Specify Requirement

Three Categories of Exceptions

- **Checked exception**: exceptional conditions that a well-written application should anticipate and recover from
  - Example: java.io.FileNotFoundException
  - **Subject to the Catch or Specify Requirement**
- **Error**: exceptional conditions that are external to the application, and that the application usually cannot anticipate or recover from
  - Example: java.io.IOError
- **Runtime exception**: exceptional conditions that are internal to the application, and that the application usually cannot anticipate or recover from
  - Example: NullPointerException
- Errors and runtime exceptions are known as unchecked exceptions

Catching and Handling Exceptions

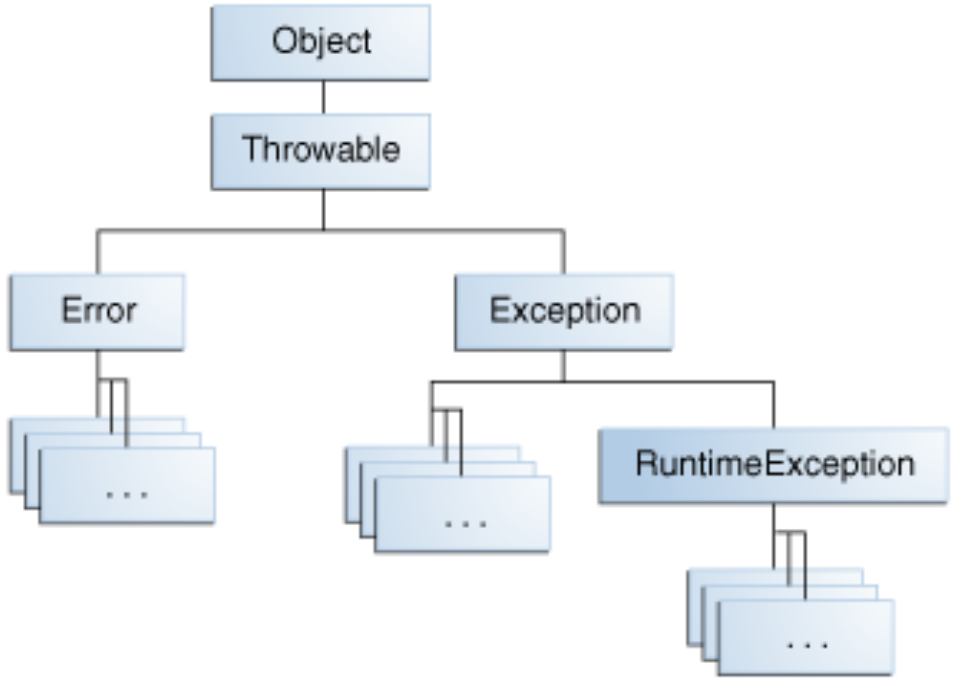
- An example: writing a list of numbers into a text file
  - A checked exception and an unchecked exception
- The **try** block
- The **catch** blocks
  - The **argument type** must be the name of a class that inherits from the Throwable class
  - The runtime system invokes the exception handler when the handler is the first one in the call stack whose ExceptionType **matches** the type of the exception thrown
  - A single catch block can handle more than one type of exception
- The **finally** block

Specifying Exceptions Thrown by Methods

- In some cases, it is better to let a method further up the call stack handle the exception
- The **throws** clause comprises the throws keyword followed by a comma-separated list of all the exceptions thrown by that method
- Example

Throwing Exceptions

- The throw statement
  - Requiring a single argument: a Throwable object
  - Example: the Stack's pop method
- The Throwable class and its subclasses



Chained Exceptions

- An application often responds to an exception by throwing another exception

```
try {
    //...
} catch (IOException e) {
    throw new SampleException("Other IOException", e);
}
```

- Methods and constructors in Throwable that support chained exceptions
  - Throwable getCause()
  - Throwable initCause(Throwable)
  - Throwable(String, Throwable)
  - Throwable(Throwable)

Creating Exception Classes

- Do you need an exception type that isn't represented by those in the Java platform?
- Would it help users if they could differentiate your exceptions from those thrown by classes written by other vendors?
- Does your code throw more than one related exception?
- If you use someone else's exceptions, will users have access to those exceptions? A similar question is, should your package be independent and self-contained?

Advantages of Exceptions

- Separating error-handling code from regular code
- Propagating errors up the call stack
- Grouping and differentiating error types