



Chapter 14:

Exception Handling

Yepang LIU (刘烨庞)

liuyp1@sustech.edu.cn



Objectives

- ▶ What exceptions are
- ▶ How exception handling works
- ▶ Exception class hierarchy
- ▶ Checked/unchecked exceptions
- ▶ Stack traces and chained exceptions



Exception

- ▶ An *exception* is an indication of a problem that occurs during a program's execution. It would disrupt the normal flow of instructions.

```
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    System.out.print("Enter an integer numerator: ");
    int numerator = scanner.nextInt();
    System.out.print("Enter an integer denominator: ");
    int denominator = scanner.nextInt();
    int result = quotient(numerator, denominator);
    System.out.printf("Result: %d / %d = %d\n", numerator, denominator, result);
    scanner.close();
}

public static int quotient(int numerator, int denominator) {
    return numerator / denominator;
}
```



Three executions of the program

```
Enter an integer numerator: 3
Enter an integer denominator: 2
Result: 3 / 2 = 1
```

A normal execution, where the result is calculated correctly.

```
Enter an integer numerator: 3
Enter an integer denominator: 0
Exception in thread "main" java.lang.ArithmeticException: / by zero
at ExceptionExample.quotient(ExceptionExample.java:15)
at ExceptionExample.main(ExceptionExample.java:10)
```

An execution where the “/ by zero” exception is thrown and the program terminates



Three executions of the program

```
Enter an integer numerator: 3
Enter an integer denominator: a
Exception in thread "main" java.util.InputMismatchException
at java.util.Scanner.throwFor(Unknown Source)
at java.util.Scanner.next(Unknown Source)
at java.util.Scanner.nextInt(Unknown Source)
at java.util.Scanner.nextInt(Unknown Source)
at ExceptionExample.main(ExceptionExample.java:9)
```

An execution where the “InputMismatch” exception is thrown and the program terminates

Exception (Cont.)

The name (type) of the exception

Stack Trace

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at ExceptionExample.quotient(ExceptionExample.java:15)
    at ExceptionExample.main(ExceptionExample.java:10)
```

The method call stack when the exception occurs

Stack trace contains the path of execution that led to the exception!!!



Exception (Cont.)

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
at ExceptionExample.quotient(ExceptionExample.java:15)
at ExceptionExample.main(ExceptionExample.java:10)
```

```
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    System.out.print("Enter an integer numerator: ");
    int numerator = scanner.nextInt();
    System.out.print("Enter an integer denominator: ");
    int denominator = scanner.nextInt();
    int result = quotient(numerator, denominator);
    ...
}
```

```
public static int quotient(int numerator, int denominator) {
    return numerator / denominator;
}
```

**The execution path
information in the stack
trace helps debugging**

Line 10

Line 15

Exception throw point: top row of the call stack



Exceptions we've seen so far

- ▶ `ArithmeticException (3/0)`
- ▶ `ArrayIndexOutOfBoundsException (array[-1])`
- ▶ `ClassCastException ((String) obj)`
- ▶ `NullPointerException (obj.foo())`
- ▶ `FileNotFoundException`
- ▶



Objectives

- ▶ What exceptions are
- ▶ How exception handling works
- ▶ Exception class hierarchy
- ▶ Checked/unchecked exceptions
- ▶ Stack traces and chained exceptions



Exception handling

- ▶ An exception would disrupt program execution flows (for example, causing crashes).
- ▶ **Exception handling** is a nice feature of the Java language that can help you write **robust** and **fault-tolerant** programs.
- ▶ With exception handling, a program can **continue executing (rather than terminating)** after dealing with a problem. It is very useful in **mission-critical** or **business-critical** computing.



try-catch statement syntax

```
try {  
    // code that might throw an exception  
} catch( ExceptionType1 e1 ) {  
    // code that handles type1 exception  
} catch( ExceptionType2 e2 ) {  
    // code that handles type2 exception  
} catch( ExceptionType3 e3 ) {  
    // code that handles type3 exception  
} ...
```

Exception parameter

e1 is a **local variable** in the catch block

At least one catch block or a finally block must **immediately** follow the try block (“immediately” means no content in between)



Handling the two exceptions

```
public static void main(String[] args) {  
    Scanner scanner = new Scanner(System.in);  
    boolean continueLoop = true;  
    do {
```

try { Enclose the code that might throw an exception in a try block

```
        System.out.print("Enter an integer numerator: ");  
        int numerator = scanner.nextInt();  
        System.out.print("Enter an integer denominator: ");  
        int denominator = scanner.nextInt();  
        int result = quotient(numerator, denominator);  
        System.out.printf("Result: %d / %d = %d\n", numerator, denominator, result);  
        scanner.close();  
        continueLoop = false;
```

} catch(InputMismatchException inputMismatchException) {

```
        System.err.printf("Exception: %s\n", inputMismatchException);  
        scanner.nextLine(); // discard input so user can try again
```

} catch(ArithmeticException arithmeticException) {

```
        System.err.printf("Exception: %s\n", arithmeticException);
```

}

```
} while(continueLoop);
```

```
}
```

Each catch block (exception handler) handles a certain type of exception.

The type is specified in the exception parameter.



Handling the two exceptions

```
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    boolean continueLoop = true;
    do {
        try {
            System.out.print("Enter an integer numerator: ");
            int numerator = scanner.nextInt();
            System.out.print("Enter an integer denominator: ");
            int denominator = scanner.nextInt();
            int result = quotient(numerator, denominator);
            System.out.printf("Result: %d / %d = %d\n", numerator, denominator, result);
            scanner.close();
            continueLoop = false;
        } catch (InputMismatchException inputMismatchException) {
            System.err.printf("Exception: %s\n", inputMismatchException);
            scanner.nextLine(); // discard input so user can try again
        } catch (ArithmeticException arithmeticException) {
            System.err.printf("Exception: %s\n", arithmeticException);
        }
    } while(continueLoop);
}
```

What if we remove
this statement?



Handling the two exceptions

```
public static void main(String[] args) {  
    Scanner scanner = new Scanner(System.in);  
    boolean continueLoop = true;  
    do {  
        try {  
            System.out.print("Enter an integer numerator: ");  
            int numerator = scanner.nextInt();  
            System.out.print("Enter an integer denominator: ");  
            int denominator = scanner.nextInt();  
            int result = quotient(numerator, denominator);  
            System.out.printf("Result: %d / %d = %d\n", numerator, denominator, result);  
            scanner.close();  
            continueLoop = false;  
        } catch (InputMismatchException inputMismatchException) {  
            System.err.printf("Exception: %s\n", inputMismatchException);  
            scanner.nextLine(); // discard input so user can try again  
        } catch (ArithmeticException arithmeticException) {  
            System.err.printf("Exception: %s\n", arithmeticException);  
        }  
    } while(continueLoop);  
}
```

Loop until all inputs are valid



Let's examine the control flow
of a typical case



```
public static void main(String[] args) {  
1 Scanner scanner = new Scanner(System.in);  
    boolean continueLoop = true;  
    do {  
        try {  
            System.out.print("Enter an integer numerator: ");  
            int numerator = scanner.nextInt();  
            System.out.print("Enter an integer denominator: ");  
            int denominator = scanner.nextInt();  
            int result = quotient(numerator, denominator);  
            System.out.printf("Result: %d / %d = %d\n", numerator, denominator, result);  
            scanner.close();  
            continueLoop = false;  
        } catch (InputMismatchException inputMismatchException) {  
            System.err.printf("Exception: %s\n", inputMismatchException);  
            scanner.nextLine(); // discard input so user can try again  
        } catch (ArithmeticException arithmeticException) {  
            System.err.printf("Exception: %s\n", arithmeticException);  
        }  
    } while(continueLoop);  
}  
  
public static int quotient(int numerator, int denominator) {  
    return numerator / denominator;  
}
```




```
public static void main(String[] args) {  
1 Scanner scanner = new Scanner(System.in);  
2 boolean continueLoop = true;  
    do {  
        try {  
            System.out.print("Enter an integer numerator: ");  
            int numerator = scanner.nextInt();  
            System.out.print("Enter an integer denominator: ");  
            int denominator = scanner.nextInt();  
            int result = quotient(numerator, denominator);  
            System.out.printf("Result: %d / %d = %d\n", numerator, denominator, result);  
            scanner.close();  
            continueLoop = false;  
        } catch (InputMismatchException inputMismatchException) {  
            System.err.printf("Exception: %s\n", inputMismatchException);  
            scanner.nextLine(); // discard input so user can try again  
        } catch (ArithmeticException arithmeticException) {  
            System.err.printf("Exception: %s\n", arithmeticException);  
        }  
    } while(continueLoop);  
}  
  
public static int quotient(int numerator, int denominator) {  
    return numerator / denominator;  
}
```



```
public static void main(String[] args) {  
    1 Scanner scanner = new Scanner(System.in);  
    2 boolean continueLoop = true;  
    do {  
        try {  
            3 System.out.print("Enter an integer numerator: ");  
            int numerator = scanner.nextInt();  
            System.out.print("Enter an integer denominator: ");  
            int denominator = scanner.nextInt();  
            int result = quotient(numerator, denominator);  
            System.out.printf("Result: %d / %d = %d\n", numerator, denominator, result);  
            scanner.close();  
            continueLoop = false;  
        } catch (InputMismatchException inputMismatchException) {  
            System.err.printf("Exception: %s\n", inputMismatchException);  
            scanner.nextLine(); // discard input so user can try again  
        } catch (ArithmeticException arithmeticException) {  
            System.err.printf("Exception: %s\n", arithmeticException);  
        }  
    } while(continueLoop);  
}  
  
public static int quotient(int numerator, int denominator) {  
    return numerator / denominator;  
}
```

Enter an integer numerator:



```
public static void main(String[] args) {  
    1 Scanner scanner = new Scanner(System.in);  
    2 boolean continueLoop = true;  
    do {  
        try {  
            3 System.out.print("Enter an integer numerator: ");  
            4 int numerator = scanner.nextInt();  
            System.out.print("Enter an integer denominator: ");  
            int denominator = scanner.nextInt();  
            int result = quotient(numerator, denominator);  
            System.out.printf("Result: %d / %d = %d\n", numerator, denominator, result);  
            scanner.close();  
            continueLoop = false;  
        } catch (InputMismatchException inputMismatchException) {  
            System.err.printf("Exception: %s\n", inputMismatchException);  
            scanner.nextLine(); // discard input so user can try again  
        } catch (ArithmeticException arithmeticException) {  
            System.err.printf("Exception: %s\n", arithmeticException);  
        }  
    } while(continueLoop);  
}  
  
public static int quotient(int numerator, int denominator) {  
    return numerator / denominator;  
}
```

Enter an integer numerator: 3



```
public static void main(String[] args) {  
    1 Scanner scanner = new Scanner(System.in);  
    2 boolean continueLoop = true;  
    do {  
        try {  
            3 System.out.print("Enter an integer numerator: ");  
            4 int numerator = scanner.nextInt();  
            5 System.out.print("Enter an integer denominator: ");  
            int denominator = scanner.nextInt();  
            int result = quotient(numerator, denominator);  
            System.out.printf("Result: %d / %d = %d\n", numerator, denominator, result);  
            scanner.close();  
            continueLoop = false;  
        } catch (InputMismatchException inputMismatchException) {  
            System.err.printf("Exception: %s\n", inputMismatchException);  
            scanner.nextLine(); // discard input so user can try again  
        } catch (ArithmeticException arithmeticException) {  
            System.err.printf("Exception: %s\n", arithmeticException);  
        }  
    } while(continueLoop);  
}  
  
public static int quotient(int numerator, int denominator) {  
    return numerator / denominator;  
}
```

Enter an integer denominator:



```
public static void main(String[] args) {  
  1 Scanner scanner = new Scanner(System.in);  
  2 boolean continueLoop = true;  
  do {  
    try {  
      3 System.out.print("Enter an integer numerator: ");  
      4 int numerator = scanner.nextInt();  
      5 System.out.print("Enter an integer denominator: ");  
      6 int denominator = scanner.nextInt();  
      int result = quotient(numerator, denominator);  
      System.out.printf("Result: %d / %d = %d\n", numerator, denominator, result);  
      scanner.close();  
      continueLoop = false;  
    } catch (InputMismatchException inputMismatchException) {  
      System.err.printf("Exception: %s\n", inputMismatchException);  
      scanner.nextLine(); // discard input so user can try again  
    } catch (ArithmeticException arithmeticException) {  
      System.err.printf("Exception: %s\n", arithmeticException);  
    }  
  } while(continueLoop);  
}  
  
public static int quotient(int numerator, int denominator) {  
  return numerator / denominator;  
}
```

Enter an integer denominator: a



Exception occurs!



```
public static void main(String[] args) {  
  1 Scanner scanner = new Scanner(System.in);  
  2 boolean continueLoop = true;  
  do {  
    try {  
      3 System.out.print("Enter an integer numerator: ");  
      4 int numerator = scanner.nextInt();  
      5 System.out.print("Enter an integer denominator: ");  
      6 int denominator = scanner.nextInt();  
      int result = quotient(numerator, denominator);  
      System.out.printf("Result: %d / %d = %d\n", numerator, denominator, result);  
      scanner.close();  
      continueLoop = false;  
    } catch (InputMismatchException inputMismatchException) {  
      7 System.err.printf("Exception: %s\n", inputMismatchException);  
      scanner.nextLine(); // discard input so user can try again  
    } catch (ArithmeticException arithmeticException) {  
      System.err.printf("Exception: %s\n", arithmeticException);  
    }  
  } while(continueLoop);  
}  
  
public static int quotient(int numerator, int denominator) {  
  return numerator / denominator;  
}
```

**Skip the remaining
statements in the try
block (termination
model)**

**The first catch block
whose exception type
matches the thrown one
gets executed**

Exception: java.util.InputMismatchException



```
public static void main(String[] args) {  
1 Scanner scanner = new Scanner(System.in);  
2 boolean continueLoop = true;  
    do {  
        try {  
3 System.out.print("Enter an integer numerator: ");  
4 int numerator = scanner.nextInt();  
5 System.out.print("Enter an integer denominator: ");  
6 int denominator = scanner.nextInt();  
        int result = quotient(numerator, denominator);  
        System.out.printf("Result: %d / %d = %d\n", numerator, denominator, result);  
        scanner.close();  
        continueLoop = false;  
    } catch (InputMismatchException inputMismatchException) {  
7 System.err.printf("Exception: %s\n", inputMismatchException);  
8 scanner.nextLine(); // discard input so user can try again  
    } catch (ArithmeticException arithmeticException) {  
        System.err.printf("Exception: %s\n", arithmeticException);  
    }  
    } while(continueLoop);  
}  
  
public static int quotient(int numerator, int denominator) {  
    return numerator / denominator;  
}
```

What happens
if we delete this
line?



```
public static void main(String[] args) {  
    1 Scanner scanner = new Scanner(System.in);  
    2 boolean continueLoop = true;  
    do {  
        try {  
            3 System.out.print("Enter an integer numerator: ");  
            4 int numerator = scanner.nextInt();  
            5 System.out.print("Enter an integer denominator: ");  
            6 int denominator = scanner.nextInt();  
            int result = quotient(numerator, denominator);  
            System.out.printf("Result: %d / %d = %d\n", numerator, denominator, result);  
            scanner.close();  
            continueLoop = false;  
        } catch (InputMismatchException inputMismatchException) {  
            7 System.err.printf("Exception: %s\n", inputMismatchException);  
            8 scanner.nextLine(); // discard input so user can try again  
        } catch (ArithmeticException arithmeticException) {  
            System.err.printf("Exception: %s\n", arithmeticException);  
        }  
    } while(continueLoop); 9  
}  
  
public static int quotient(int numerator, int denominator) {  
    return numerator / denominator;  
}
```

**Exit the try-catch
statement and continue
with the loop condition
test**



```
public static void main(String[] args) {  
1 Scanner scanner = new Scanner(System.in);  
2 boolean continueLoop = true;  
    do {  
        try {  
10 3 System.out.print("Enter an integer numerator: ");  
4 int numerator = scanner.nextInt();  
5 System.out.print("Enter an integer denominator: ");  
6 int denominator = scanner.nextInt();  
        int result = quotient(numerator, denominator);  
        System.out.printf("Result: %d / %d = %d\n", numerator, denominator, result);  
        scanner.close();  
        continueLoop = false;  
    } catch (InputMismatchException inputMismatchException) {  
7 System.err.printf("Exception: %s\n", inputMismatchException);  
8 scanner.nextLine(); // discard input so user can try again  
    } catch (ArithmeticException arithmeticException) {  
        System.err.printf("Exception: %s\n", arithmeticException);  
    }  
    } while(continueLoop); 9  
}  
  
public static int quotient(int numerator, int denominator) {  
    return numerator / denominator;  
}
```

Enter an integer numerator:



```
public static void main(String[] args) {  
1 Scanner scanner = new Scanner(System.in);  
2 boolean continueLoop = true;  
    do {  
        try {  
10 3 System.out.print("Enter an integer numerator: ");  
11 4 int numerator = scanner.nextInt();  
5 System.out.print("Enter an integer denominator: ");  
6 int denominator = scanner.nextInt();  
        int result = quotient(numerator, denominator);  
        System.out.printf("Result: %d / %d = %d\n", numerator, denominator, result);  
        scanner.close();  
        continueLoop = false;  
    } catch (InputMismatchException inputMismatchException) {  
7 System.err.printf("Exception: %s\n", inputMismatchException);  
8 scanner.nextLine(); // discard input so user can try again  
    } catch (ArithmeticException arithmeticException) {  
        System.err.printf("Exception: %s\n", arithmeticException);  
    }  
    } while(continueLoop); 9  
}  
  
public static int quotient(int numerator, int denominator) {  
    return numerator / denominator;  
}
```

Enter an integer numerator: 3



```
public static void main(String[] args) {  
1 Scanner scanner = new Scanner(System.in);  
2 boolean continueLoop = true;  
    do {  
        try {  
10 3 System.out.print("Enter an integer numerator: ");  
11 4 int numerator = scanner.nextInt();  
12 5 System.out.print("Enter an integer denominator: ");  
        6 int denominator = scanner.nextInt();  
        int result = quotient(numerator, denominator);  
        System.out.printf("Result: %d / %d = %d\n", numerator, denominator, result);  
        scanner.close();  
        continueLoop = false;  
    } catch (InputMismatchException inputMismatchException) {  
7 System.err.printf("Exception: %s\n", inputMismatchException);  
8 scanner.nextLine(); // discard input so user can try again  
    } catch (ArithmeticException arithmeticException) {  
        System.err.printf("Exception: %s\n", arithmeticException);  
    }  
    } while(continueLoop); 9  
}  
  
public static int quotient(int numerator, int denominator) {  
    return numerator / denominator;  
}
```

Enter an integer denominator:



```
public static void main(String[] args) {  
1 Scanner scanner = new Scanner(System.in);  
2 boolean continueLoop = true;  
    do {  
        try {  
10 3 System.out.print("Enter an integer numerator: ");  
11 4 int numerator = scanner.nextInt();  
12 5 System.out.print("Enter an integer denominator: ");  
13 6 int denominator = scanner.nextInt();  
        int result = quotient(numerator, denominator);  
        System.out.printf("Result: %d / %d = %d\n", numerator, denominator, result);  
        scanner.close();  
        continueLoop = false;  
    } catch (InputMismatchException inputMismatchException) {  
7 System.err.printf("Exception: %s\n", inputMismatchException);  
8 scanner.nextLine(); // discard input so user can try again  
    } catch (ArithmeticException arithmeticException) {  
        System.err.printf("Exception: %s\n", arithmeticException);  
    }  
    } while(continueLoop); 9  
}  
  
public static int quotient(int numerator, int denominator) {  
    return numerator / denominator;  
}
```

```
Enter an integer denominator: 0
```



```
public static void main(String[] args) {  
1 Scanner scanner = new Scanner(System.in);  
2 boolean continueLoop = true;  
    do {  
        try {  
10 3 System.out.print("Enter an integer numerator: ");  
11 4 int numerator = scanner.nextInt();  
12 5 System.out.print("Enter an integer denominator: ");  
13 6 int denominator = scanner.nextInt();  
14 int result = quotient(numerator, denominator);  
        System.out.printf("Result: %d / %d = %d\n", numerator, denominator, result);  
        scanner.close();  
        continueLoop = false;  
    } catch (InputMismatchException inputMismatchException) {  
7 System.err.printf("Exception: %s\n", inputMismatchException);  
8 scanner.nextLine(); // discard input so user can try again  
    } catch (ArithmeticException arithmeticException) {  
        System.err.printf("Exception: %s\n", arithmeticException);  
    }  
    } while(continueLoop); 9  
}  
  
public static int quotient(int numerator, int denominator) {  
    return numerator / denominator;  
}
```



```
public static void main(String[] args) {  
1 Scanner scanner = new Scanner(System.in);  
2 boolean continueLoop = true;  
    do {  
        try {  
10 3 System.out.print("Enter an integer numerator: ");  
11 4 int numerator = scanner.nextInt();  
12 5 System.out.print("Enter an integer denominator: ");  
13 6 int denominator = scanner.nextInt();  
14 int result = quotient(numerator, denominator);  
        System.out.printf("Result: %d / %d = %d\n", numerator, denominator, result);  
        scanner.close();  
        continueLoop = false;  
    } catch(InputMismatchException inputMismatchException) {  
7 System.err.printf("Exception: %s\n", inputMismatchException);  
8 scanner.nextLine(); // discard input so user can try again  
    } catch(ArithmeticException arithmeticException) {  
        System.err.printf("Exception: %s\n", arithmeticException);  
    }  
    } while(continueLoop); 9  
}  
  
public static int quotient(int numerator, int denominator) {  
15 return numerator / denominator;  
}
```



Exception occurs!



```
public static void main(String[] args) {  
1 Scanner scanner = new Scanner(System.in);  
2 boolean continueLoop = true;  
    do {  
        try {  
10 3 System.out.print("Enter an integer numerator: ");  
11 4 int numerator = scanner.nextInt();  
12 5 System.out.print("Enter an integer denominator: ");  
13 6 int denominator = scanner.nextInt();  
14 int result = quotient(numerator, denominator);  
        System.out.printf("Result: %d / %d = %d\n", numerator, denominator, result);  
        scanner.close();  
        continueLoop = false;  
    } catch (InputMismatchException inputMismatchException) {  
7 System.err.printf("Exception: %s\n", inputMismatchException);  
8 scanner.nextLine(); // discard input so user can try again  
    } catch (ArithmeticException arithmeticException) {  
16 System.err.printf("Exception: %s\n", arithmeticException);  
    }  
    } while(continueLoop); 9  
}  
  
public static int quotient(int numerator, int denominator) {  
15 return numerator / denominator;  
}
```

**Skip the remaining
statements in the try
block**

**The first catch block
whose exception type
matches the thrown one
gets executed**

Exception: java.lang.ArithmeticException: / by zero



```
public static void main(String[] args) {  
1 Scanner scanner = new Scanner(System.in);  
2 boolean continueLoop = true;  
    do {  
        try {  
10 3 System.out.print("Enter an integer numerator: ");  
11 4 int numerator = scanner.nextInt();  
12 5 System.out.print("Enter an integer denominator: ");  
13 6 int denominator = scanner.nextInt();  
14 int result = quotient(numerator, denominator);  
        System.out.printf("Result: %d / %d = %d\n", numerator, denominator, result);  
        scanner.close();  
        continueLoop = false;  
    } catch (InputMismatchException inputMismatchException) {  
7 System.err.printf("Exception: %s\n", inputMismatchException);  
8 scanner.nextLine(); // discard input so user can try again  
    } catch (ArithmeticException arithmeticException) {  
16 System.err.printf("Exception: %s\n", arithmeticException);  
    }  
    } while(continueLoop); 9 17  
}  
  
public static int quotient(int numerator, int denominator) {  
15 return numerator / denominator;  
}
```

**Exit the try-catch
statement and continue
with the loop condition
test**



```
public static void main(String[] args) {
1  Scanner scanner = new Scanner(System.in);
2  boolean continueLoop = true;
    do {
        try {
18 10 3  System.out.print("Enter an integer numerator: ");
11 4   int numerator = scanner.nextInt();
12 5   System.out.print("Enter an integer denominator: ");
13 6   int denominator = scanner.nextInt();
14    int result = quotient(numerator, denominator);
        System.out.printf("Result: %d / %d = %d\n", numerator, denominator, result);
        scanner.close();
        continueLoop = false;
    } catch(InputMismatchException inputMismatchException) {
7     System.err.printf("Exception: %s\n", inputMismatchException);
8     scanner.nextLine(); // discard input so user can try again
    } catch(ArithmeticException arithmeticException) {
16    System.err.printf("Exception: %s\n", arithmeticException);
    }
    } while(continueLoop); 9 17
}

public static int quotient(int numerator, int denominator) {
15  return numerator / denominator;
}
```

Enter an integer numerator:



```
public static void main(String[] args) {  
1 Scanner scanner = new Scanner(System.in);  
2 boolean continueLoop = true;  
    do {  
        try {  
18 10 3 System.out.print("Enter an integer numerator: ");  
19 11 4 int numerator = scanner.nextInt();  
12 5 System.out.print("Enter an integer denominator: ");  
13 6 int denominator = scanner.nextInt();  
14 int result = quotient(numerator, denominator);  
        System.out.printf("Result: %d / %d = %d\n", numerator, denominator, result);  
        scanner.close();  
        continueLoop = false;  
    } catch (InputMismatchException inputMismatchException) {  
7 System.err.printf("Exception: %s\n", inputMismatchException);  
8 scanner.nextLine(); // discard input so user can try again  
    } catch (ArithmeticException arithmeticException) {  
16 System.err.printf("Exception: %s\n", arithmeticException);  
    }  
    } while(continueLoop); 9 17  
}  
  
public static int quotient(int numerator, int denominator) {  
15 return numerator / denominator;  
}
```

Enter an integer numerator: 3



```
public static void main(String[] args) {  
1 Scanner scanner = new Scanner(System.in);  
2 boolean continueLoop = true;  
    do {  
        try {  
18 10 3 System.out.print("Enter an integer numerator: ");  
19 11 4 int numerator = scanner.nextInt();  
20 12 5 System.out.print("Enter an integer denominator: ");  
13 6 int denominator = scanner.nextInt();  
14 int result = quotient(numerator, denominator);  
        System.out.printf("Result: %d / %d = %d\n", numerator, denominator, result);  
        scanner.close();  
        continueLoop = false;  
    } catch (InputMismatchException inputMismatchException) {  
7 System.err.printf("Exception: %s\n", inputMismatchException);  
8 scanner.nextLine(); // discard input so user can try again  
    } catch (ArithmeticException arithmeticException) {  
16 System.err.printf("Exception: %s\n", arithmeticException);  
    }  
    } while(continueLoop); 9 17  
}  
  
public static int quotient(int numerator, int denominator) {  
15 return numerator / denominator;  
}
```

Enter an integer denominator:



```
public static void main(String[] args) {  
1 Scanner scanner = new Scanner(System.in);  
2 boolean continueLoop = true;  
    do {  
        try {  
18 10 3 System.out.print("Enter an integer numerator: ");  
19 11 4 int numerator = scanner.nextInt();  
20 12 5 System.out.print("Enter an integer denominator: ");  
21 13 6 int denominator = scanner.nextInt();  
14 int result = quotient(numerator, denominator);  
        System.out.printf("Result: %d / %d = %d\n", numerator, denominator, result);  
        scanner.close();  
        continueLoop = false;  
    } catch (InputMismatchException inputMismatchException) {  
7 System.err.printf("Exception: %s\n", inputMismatchException);  
8 scanner.nextLine(); // discard input so user can try again  
    } catch (ArithmeticException arithmeticException) {  
16 System.err.printf("Exception: %s\n", arithmeticException);  
    }  
    } while(continueLoop); 9 17  
}  
  
public static int quotient(int numerator, int denominator) {  
15 return numerator / denominator;  
}
```

Enter an integer denominator: 2



```
public static void main(String[] args) {  
1 Scanner scanner = new Scanner(System.in);  
2 boolean continueLoop = true;  
    do {  
        try {  
18 10 3 System.out.print("Enter an integer numerator: ");  
19 11 4 int numerator = scanner.nextInt();  
20 12 5 System.out.print("Enter an integer denominator: ");  
21 13 6 int denominator = scanner.nextInt();  
22 14 int result = quotient(numerator, denominator);  
        System.out.printf("Result: %d / %d = %d\n", numerator, denominator, result);  
        scanner.close();  
        continueLoop = false;  
    } catch (InputMismatchException inputMismatchException) {  
7 System.err.printf("Exception: %s\n", inputMismatchException);  
8 scanner.nextLine(); // discard input so user can try again  
    } catch (ArithmeticException arithmeticException) {  
16 System.err.printf("Exception: %s\n", arithmeticException);  
    }  
    } while(continueLoop); 9 17  
}  
  
public static int quotient(int numerator, int denominator) {  
15 return numerator / denominator;  
}
```



```
public static void main(String[] args) {  
1 Scanner scanner = new Scanner(System.in);  
2 boolean continueLoop = true;  
    do {  
        try {  
18 10 3 System.out.print("Enter an integer numerator: ");  
19 11 4 int numerator = scanner.nextInt();  
20 12 5 System.out.print("Enter an integer denominator: ");  
21 13 6 int denominator = scanner.nextInt();  
22 14 int result = quotient(numerator, denominator);  
        System.out.printf("Result: %d / %d = %d\n", numerator, denominator, result);  
        scanner.close();  
        continueLoop = false;  
    } catch(InputMismatchException inputMismatchException) {  
7 System.err.printf("Exception: %s\n", inputMismatchException);  
8 scanner.nextLine(); // discard input so user can try again  
    } catch(ArithmeticException arithmeticException) {  
16 System.err.printf("Exception: %s\n", arithmeticException);  
    }  
    } while(continueLoop); 9 17  
}  
  
public static int quotient(int numerator, int denominator) {  
23 15 return numerator / denominator;  
}
```



```
public static void main(String[] args) {
1  Scanner scanner = new Scanner(System.in);
2  boolean continueLoop = true;
    do {
        try {
18 10 3  System.out.print("Enter an integer numerator: ");
19 11 4  int numerator = scanner.nextInt();
20 12 5  System.out.print("Enter an integer denominator: ");
21 13 6  int denominator = scanner.nextInt();
22 14    int result = quotient(numerator, denominator);
24    System.out.printf("Result: %d / %d = %d\n", numerator, denominator, result);
        scanner.close();
        continueLoop = false;
    } catch (InputMismatchException inputMismatchException) {
7      System.err.printf("Exception: %s\n", inputMismatchException);
8      scanner.nextLine(); // discard input so user can try again
    } catch (ArithmeticException arithmeticException) {
16     System.err.printf("Exception: %s\n", arithmeticException);
    }
    } while(continueLoop); 9 17
}

public static int quotient(int numerator, int denominator) {
23 15 return numerator / denominator;
}
```

Result: 3 / 2 = 1



```
public static void main(String[] args) {  
1 Scanner scanner = new Scanner(System.in);  
2 boolean continueLoop = true;  
    do {  
        try {  
18 10 3 System.out.print("Enter an integer numerator: ");  
19 11 4 int numerator = scanner.nextInt();  
20 12 5 System.out.print("Enter an integer denominator: ");  
21 13 6 int denominator = scanner.nextInt();  
22 14 int result = quotient(numerator, denominator);  
24 System.out.printf("Result: %d / %d = %d\n", numerator, denominator, result);  
25 scanner.close();  
        continueLoop = false;  
    } catch(InputMismatchException inputMismatchException) {  
7 System.err.printf("Exception: %s\n", inputMismatchException);  
8 scanner.nextLine(); // discard input so user can try again  
    } catch(ArithmeticException arithmeticException) {  
16 System.err.printf("Exception: %s\n", arithmeticException);  
    }  
    } while(continueLoop); 9 17  
}  
  
public static int quotient(int numerator, int denominator) {  
23 15 return numerator / denominator;  
}
```




```
public static void main(String[] args) {  
1 Scanner scanner = new Scanner(System.in);  
2 boolean continueLoop = true;  
    do {  
        try {  
18 10 3 System.out.print("Enter an integer numerator: ");  
19 11 4 int numerator = scanner.nextInt();  
20 12 5 System.out.print("Enter an integer denominator: ");  
21 13 6 int denominator = scanner.nextInt();  
22 14 int result = quotient(numerator, denominator);  
24 System.out.printf("Result: %d / %d = %d\n", numerator, denominator, result);  
25 scanner.close();  
26 continueLoop = false;  
        } catch (InputMismatchException inputMismatchException) {  
7 System.err.printf("Exception: %s\n", inputMismatchException);  
8 scanner.nextLine(); // discard input so user can try again  
        } catch (ArithmeticException arithmeticException) {  
16 System.err.printf("Exception: %s\n", arithmeticException);  
        }  
    } while (continueLoop); 9 17  
}  
  
public static int quotient(int numerator, int denominator) {  
23 15 return numerator / denominator;  
}
```



```
public static void main(String[] args) {  
1  Scanner scanner = new Scanner(System.in);  
2  boolean continueLoop = true;  
    do {  
        try {  
18 10 3  System.out.print("Enter an integer numerator: ");  
19 11 4  int numerator = scanner.nextInt();  
20 12 5  System.out.print("Enter an integer denominator: ");  
21 13 6  int denominator = scanner.nextInt();  
22 14    int result = quotient(numerator, denominator);  
24      System.out.printf("Result: %d / %d = %d\n", numerator, denominator, result);  
25      scanner.close();  
26      continueLoop = false;  
        } catch (InputMismatchException inputMismatchException) {  
7      System.err.printf("Exception: %s\n", inputMismatchException);  
8      scanner.nextLine(); // discard input so user can try again  
        } catch (ArithmeticException arithmeticException) {  
16     System.err.printf("Exception: %s\n", arithmeticException);  
        }  
    } while (continueLoop); 9 17 27  
}  
  
public static int quotient(int numerator, int denominator) {  
23 15  return numerator / denominator;  
}
```

**No exception occurs,
continue with the loop
condition test**



```
public static void main(String[] args) {  
1  Scanner scanner = new Scanner(System.in);  
2  boolean continueLoop = true;  
    do {  
        try {  
18 10 3  System.out.print("Enter an integer numerator: ");  
19 11 4  int numerator = scanner.nextInt();  
20 12 5  System.out.print("Enter an integer denominator: ");  
21 13 6  int denominator = scanner.nextInt();  
22 14    int result = quotient(numerator, denominator);  
24      System.out.printf("Result: %d / %d = %d\n", numerator, denominator, result);  
25      scanner.close();  
26      continueLoop = false;  
        } catch (InputMismatchException inputMismatchException) {  
7      System.err.printf("Exception: %s\n", inputMismatchException);  
8      scanner.nextLine(); // discard input so user can try again  
        } catch (ArithmeticException arithmeticException) {  
16     System.err.printf("Exception: %s\n", arithmeticException);  
        }  
    } while(continueLoop); 9 17 27  
}  
  
public static int quotient(int numerator, int denominator) {  
23 15  return numerator / denominator;  
}
```

**Exit loop and main
method terminates
normally**



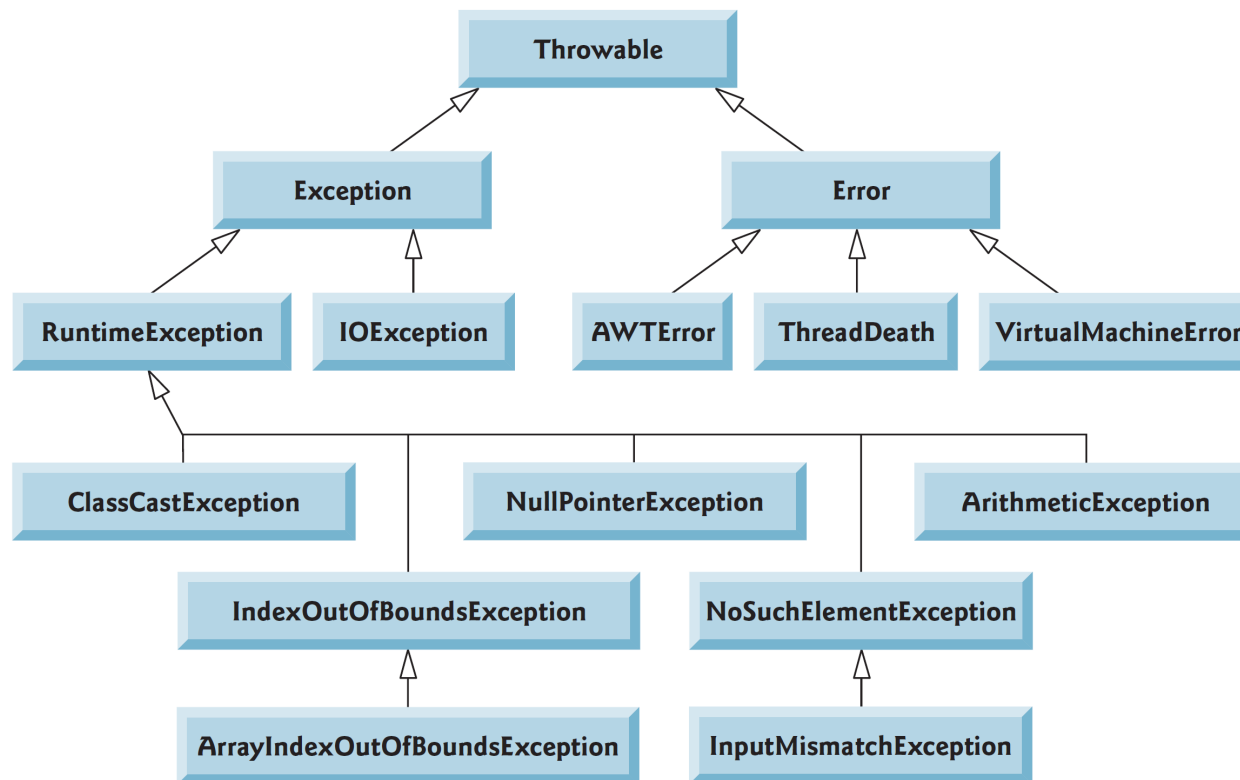
Objectives

- ▶ What exceptions are
- ▶ How exception handling works
- ▶ Exception class hierarchy
- ▶ Checked/unchecked exceptions
- ▶ Stack traces and chained exceptions



Java Exception Hierarchy

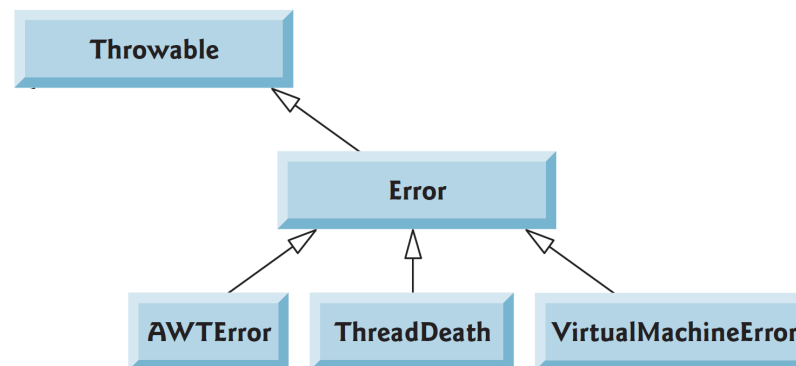
- ▶ In Java, only `Throwable` objects can be used with the exception-handling mechanism.





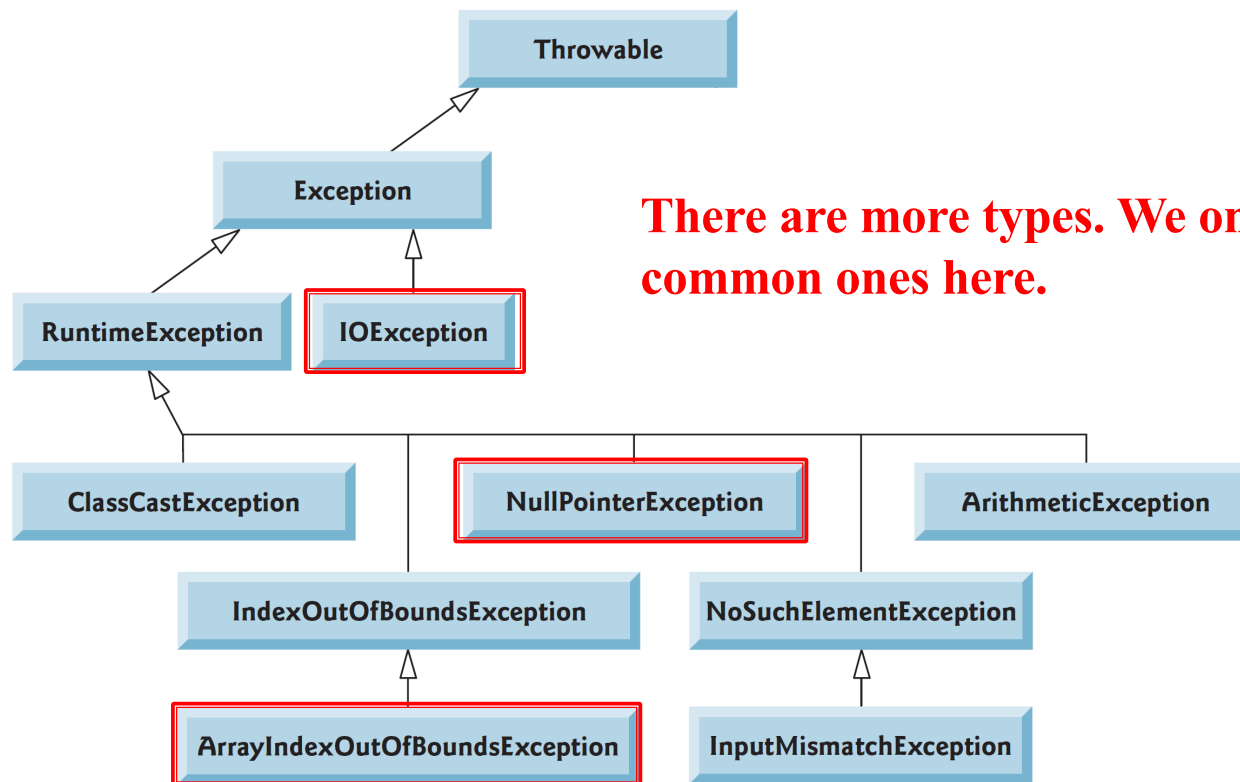
Java Exception Hierarchy

- ▶ Error and its subclasses represent abnormal situations that happen in the JVM (e.g., JVM out of memory) and **should not** be caught by applications. It's usually impossible for applications to recover from Errors.



Java Exception Hierarchy

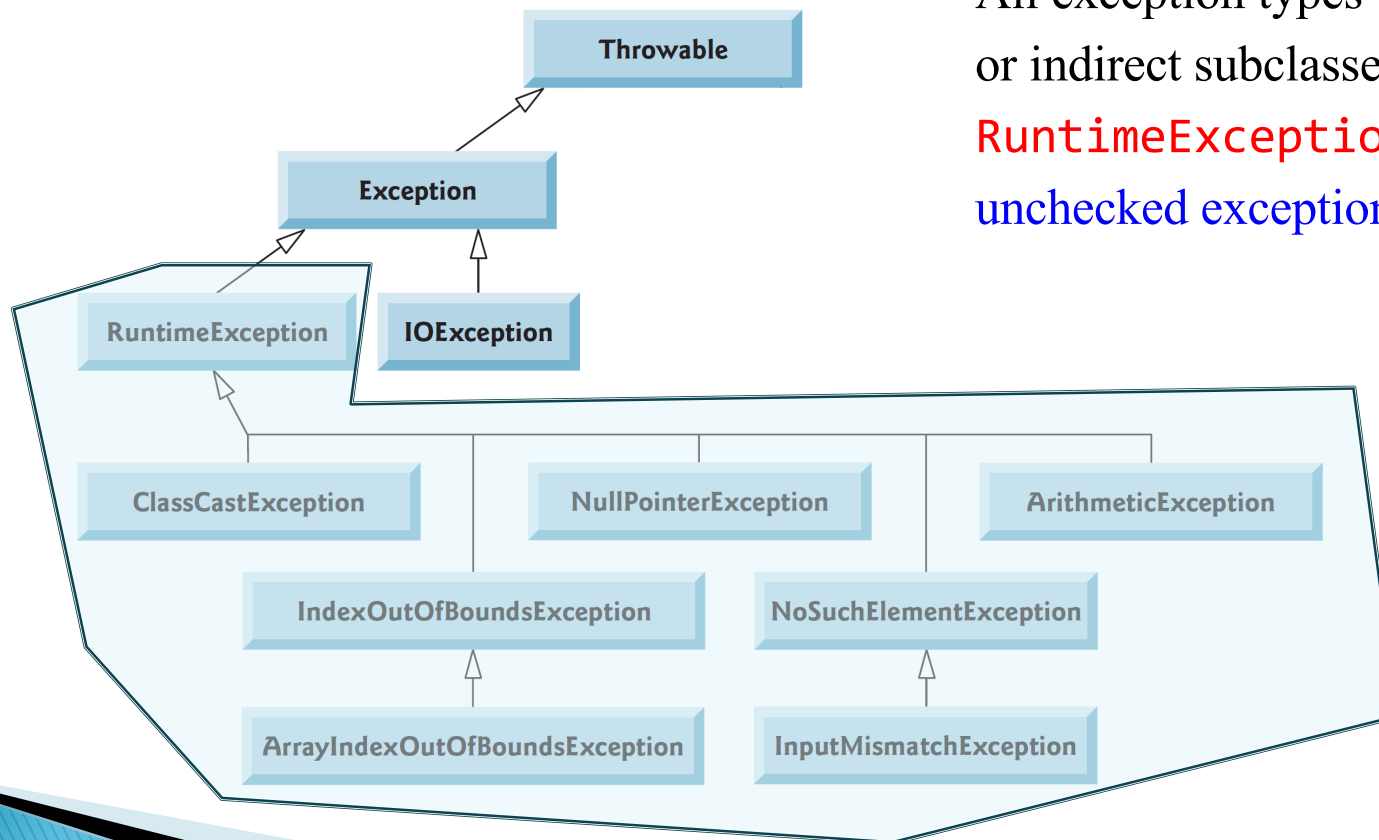
- ▶ Exception (in `java.lang`) and its subclasses represent exceptional situations that can occur in a Java program and **should be** caught by applications





Java Exception Hierarchy

- ▶ Java distinguishes between **checked exceptions** and **unchecked exceptions**.



All exception types that are direct or indirect subclasses of the class **RuntimeException** are **unchecked exceptions**.

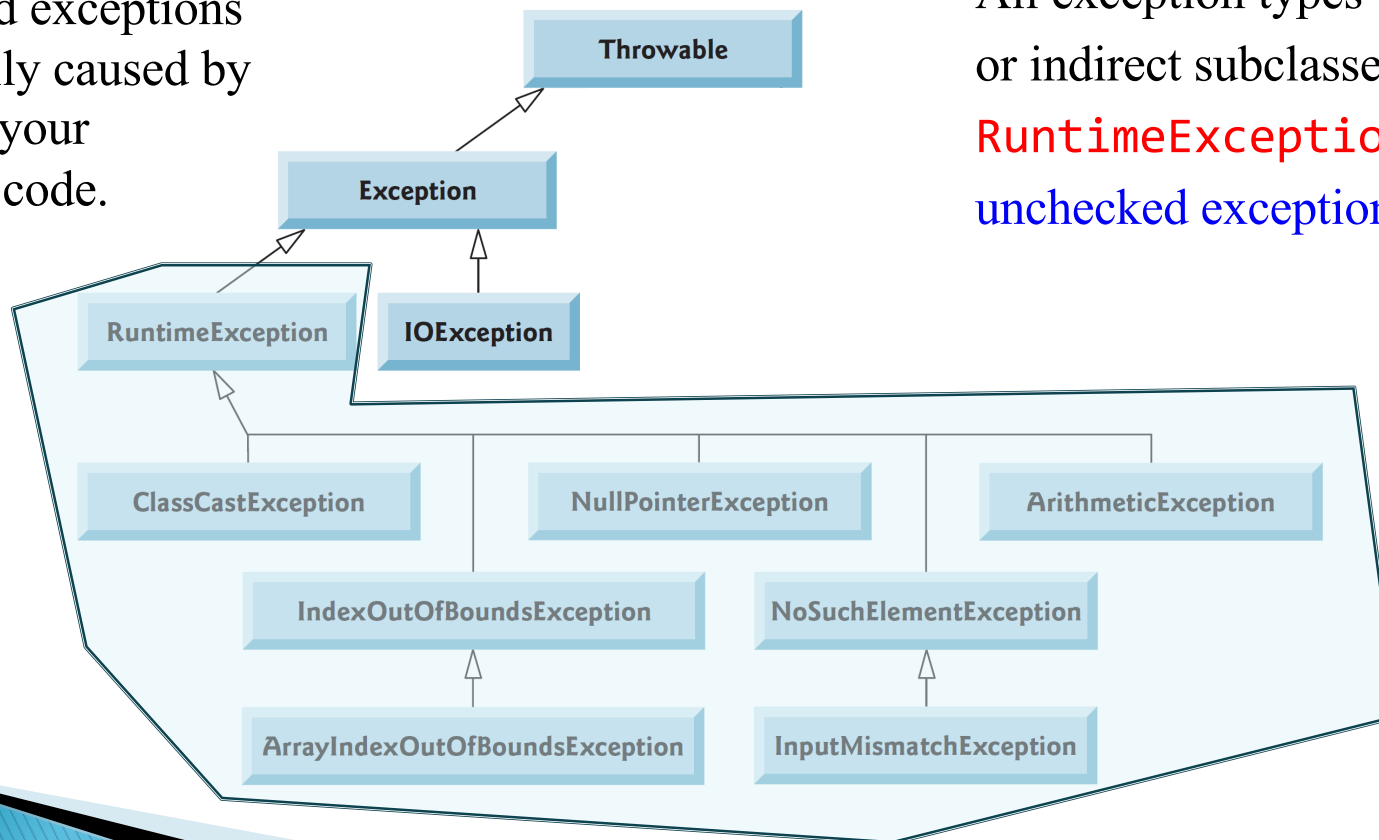


Java Exception Hierarchy

- ▶ Java distinguishes between **checked exceptions** and **unchecked exceptions**.

Unchecked exceptions are typically caused by defects in your program's code.

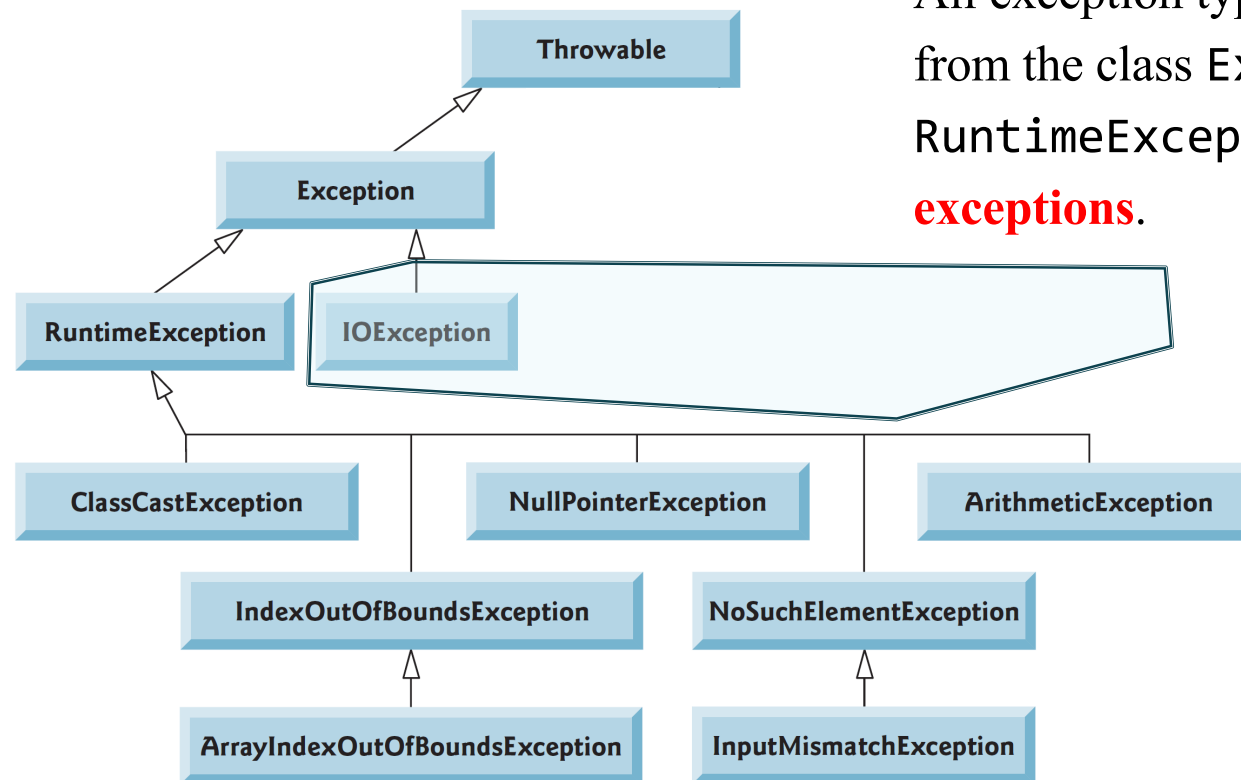
All exception types that are direct or indirect subclasses of the class **RuntimeException** are **unchecked exceptions**.





Java Exception Hierarchy

- ▶ Java distinguishes between **checked exceptions** and **unchecked exceptions**.



All exception types that inherit from the class **Exception** but not **RuntimeException** are **checked exceptions**.

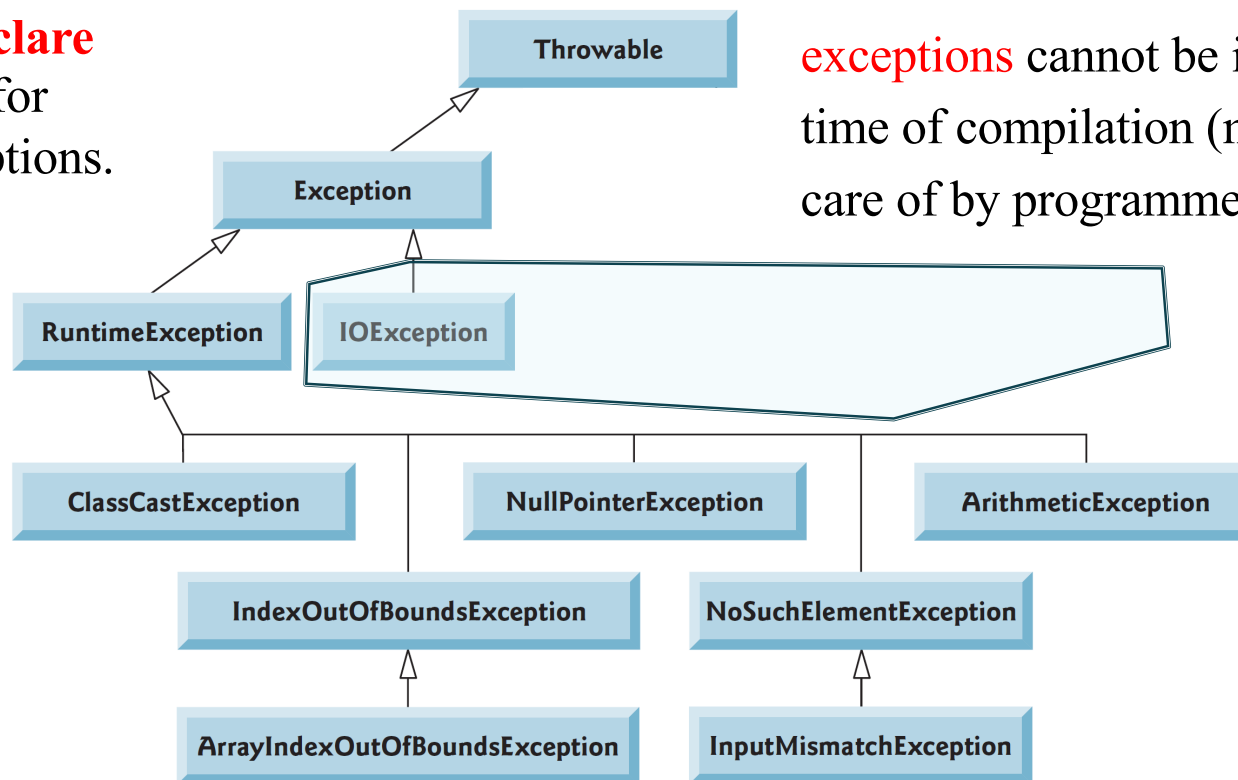


Java Exception Hierarchy

- ▶ Java distinguishes between **checked exceptions** and **unchecked exceptions**.

Java compiler enforces a **catch-or-declare requirement** for checked exceptions.

Unlike unchecked exceptions, **checked exceptions** cannot be ignored at the time of compilation (must be taken care of by programmers).





Example: Unchecked Exceptions

```
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    System.out.print("Enter an integer numerator: ");
    int numerator = scanner.nextInt(); // potential runtime exception
    System.out.print("Enter an integer denominator: ");
    int denominator = scanner.nextInt(); // potential runtime exception
    int result = quotient(numerator, denominator);
    System.out.printf("Result: %d / %d = %d\n", numerator, denominator, result);
    scanner.close();
}

public static int quotient(int numerator, int denominator) {
    return numerator / denominator; // potential runtime exception
}
```

It's fine if programmers do not take care of unchecked exceptions in the code, which can still compile.

Example: Checked Exceptions

```
public static void main(String[] args) {
```

```
    File f = new File("test.txt");
```

```
    FileReader reader = new FileReader(f);
```

```
    reader.close();
```

```
}
```

⚠ Unhandled exception type IOException

2 quick fixes available:

- ⚡ [Add throws declaration](#)
- ⚡ [Surround with try/catch](#)

⚠ Unhandled exception type FileNotFoundException

2 quick fixes available:

- ⚡ [Add throws declaration](#)
- ⚡ [Surround with try/catch](#)

Fix: catch or declare



The catch solution

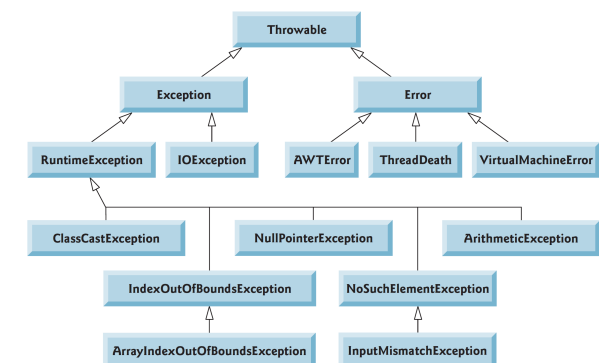
```
public static void main(String[] args) {  
    try {  
        File f = new File("test.txt");  
        FileReader reader = new FileReader(f);  
        reader.close();  
    } catch(FileNotFoundException e) {  
        // handle file not found exception  
    } catch(IOException e) {  
        // handle IO exception  
    }  
}
```



Catching subclass exceptions

- ▶ If a catch handler is written to catch superclass-type exceptions, it can also catch subclass-type exceptions.
 - Good: This makes the exception handling code concise (when the handling behavior is the same for all types) and allows for **polymorphic processing of exception objects**.
 - Bad: Exception info is too general to be useful

```
public static void main(String[] args) {  
    try {  
        File f = new File("test.txt");  
        FileReader reader = new FileReader(f);  
        reader.close();  
    } catch(Exception e) {  
        // catch and handle multiple types of exceptions  
    }  
}
```





The declare/throw solution

```
public static void main(String[] args)

    throws FileNotFoundException, IOException {

    File f = new File("test.txt");

    FileReader reader = new FileReader(f);

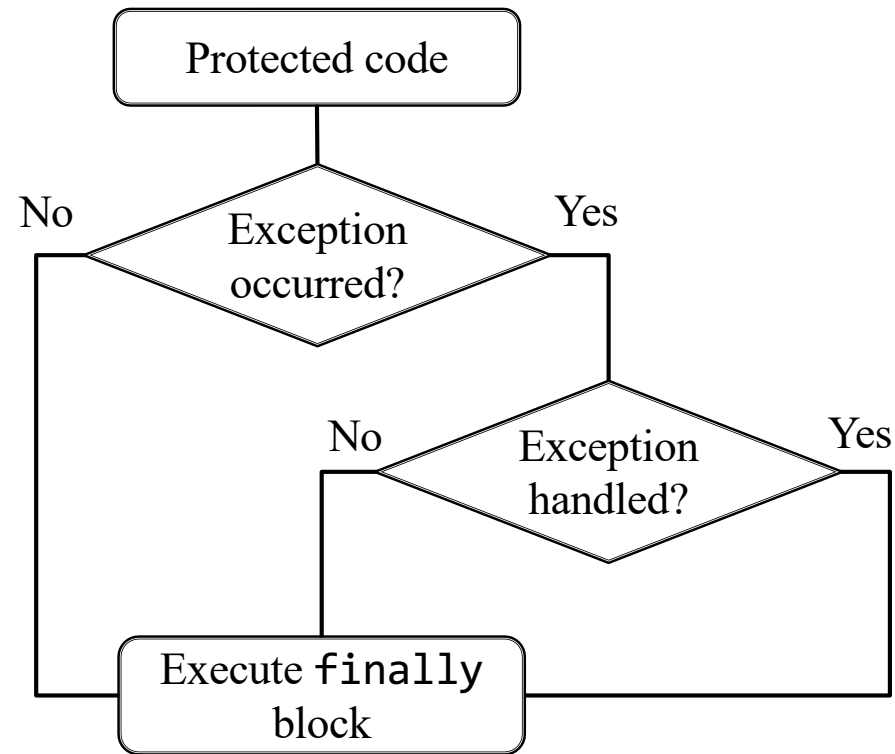
    reader.close();

}
```

The throws clause declares the exceptions that might be thrown when the method is executed and let the callers handle the exceptions.

finally block

```
try {  
    // protected code  
}  
// catch blocks (optional)  
finally {  
    // always execute  
    //when try block exits  
}
```



`finally` is useful for more than just exception handling. It prevents cleanup code from being accidentally bypassed by statements like `return`. **Putting cleanup code in a `finally` block is a good practice**, even when no exceptions are anticipated.



finally block

In the example below, the `finally` block ensures that the used resource is closed regardless of whether the `try` statement completes normally or abruptly

```
public String readFirstLineFromFile(String path) throws IOException {  
    BufferedReader br;  
    try {  
        br = new BufferedReader(new FileReader(path));  
        return br.readLine();  
    } finally {  
        if (br != null)  
            br.close();  
    }  
}
```



Common methods of exceptions

- ▶ `printStackTrace`: output the stack trace to the standard error stream.
- ▶ Standard output stream (`System.out`) and standard error stream (`System.err`) are sequences of bytes. The former displays a program's output in the command prompt and the latter displays errors.
- ▶ Using two streams helps separate error messages from other output.



```
public class CheckedExceptionExample {  
    public static void main(String[] args) {  
        try {  
            File f = new File("not-exist.txt");  
            FileReader reader = new FileReader(f);  
            reader.close();  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
    }  
}
```

```
java.io.FileNotFoundException: not-exist.txt (系统找不到指定的文件。)  
    at java.io.FileInputStream.open0(Native Method)  
    at java.io.FileInputStream.open(Unknown Source)  
    at java.io.FileInputStream.<init>(Unknown Source)  
    at java.io.FileReader.<init>(Unknown Source)  
    at CheckedExceptionExample.main(CheckedExceptionExample.java:11)
```

Common methods of exceptions



```
public class CheckedExceptionExample {  
    public static void main(String[] args) {  
        try {  
            method1();  
        } catch (Exception e) {  
            StackTraceElement[] traceElements = e.getStackTrace();  
            for(StackTraceElement element : traceElements) {  
                System.out.printf("%s\t", element.getFileName());  
                System.out.printf("%s\t", element.getLineNumber());  
                System.out.printf("%s\n", element.getMethodName());  
            }  
        }  
    }  
    public static void method1() throws Exception {  
        throw new Exception("Exception thrown in method1");  
    }  
}
```

getStackTrace: retrieves the stack trace for customized processing

The **throw** keyword can be used to throw an exception object

CheckedExceptionExample.java	15	method1
CheckedExceptionExample.java	4	main



Common methods of exceptions

```
public class CheckedExceptionExample {  
    public static void main(String[] args) {  
        try {  
            method1();  
        } catch (Exception e) {  
            System.err.println(e.getMessage());  
        }  
    }  
  
    public static void method1() throws Exception {  
        throw new Exception("Exception thrown in method1");  
    }  
}
```

getMessage: returns the detailed message of the exception

Exception thrown in method1

Chained exceptions

```
public class CheckedExceptionExample {  
    public static void main(String[] args) throws Exception {  
        try {  
            method1();  
        } catch (Exception e) {  
            throw new Exception("Exception thrown in main");  
        }  
    }  
    public static void method1() throws Exception {  
        throw new Exception("Exception thrown in method1");  
    }  
}
```

Sometimes we need to throw an exception in the catch block

The information of the original exception in method1 will be lost in such cases

Exception in thread "main" java.lang.Exception: Exception thrown in main
at CheckedExceptionExample.main(CheckedExceptionExample.java:6)



Chained exceptions

```
public class CheckedExceptionExample {  
    public static void main(String[] args) throws Exception {  
        try {  
            method1();  
        } catch (Exception e) {  
            throw new Exception("Exception thrown in main", e);  
        }  
    }  
    public static void method1() throws Exception {  
        throw new Exception("Exception thrown in method1");  
    }  
}
```

Chained exceptions enable an exception object to maintain complete stack-trace information from the original exception

```
Exception in thread "main" java.lang.Exception: Exception thrown in main  
    at CheckedExceptionExample.main(CheckedExceptionExample.java:6)  
Caused by: java.lang.Exception: Exception thrown in method1  
    at CheckedExceptionExample.method1(CheckedExceptionExample.java:10)  
    at CheckedExceptionExample.main(CheckedExceptionExample.java:4)
```


User-defined exceptions (Checked)

```
public class MyException extends Exception {  
    public MyException(String s) {  
        super(s);  
    }  
}
```

New exception types can be defined by extending an existing exception class

```
public class UserDefinedExceptionDemo {  
    public static void main(String args[]) {  
        try {  
            throw new MyException("User-defined exception");  
        } catch (MyException e) {  
            System.err.println(e.getMessage());  
        }  
    }  
}
```

Running result: **User-defined exception**



User-defined exceptions (Unchecked)

```
public class MyException2 extends RuntimeException {  
    public MyException2(String s) {  
        super(s);  
    }  
}
```

Extending RuntimeException makes this new exception unchecked (compiler will ignore unchecked exceptions)

```
public class UserDefinedExceptionDemo2 {  
    public static void main(String args[]) {  
        throw new MyException2("User-defined exception");  
    }  
}
```

Running result:

```
Exception in thread "main" MyException2: User-defined exception  
    at UserDefinedExceptionDemo2.main(UserDefinedExceptionDemo2.java:9)
```



Assertions (断言)

- ▶ When implementing and debugging a class, it's sometimes useful to state conditions that should be true at a particular point in a method.
- ▶ These conditions, called **assertions**, help ensure a program's correctness by catching potential bugs (such as logic errors) during development.
- ▶ Java has two versions of `assert` statements
 - `assert expression;` // throws an `AssertionError` if expression is false
 - `assert expression1 : expression2;` // throws an `AssertionError` with `expression2` as the error message if `expression1` is false



Assertions

```
public class AssertionExample {  
    public static void main(String[] args) {  
        Scanner input = new Scanner(System.in);  
        System.out.print("Enter a number between 0 and 10: ");  
        int number = input.nextInt();  
        assert (number >= 0 && number <= 10) : "bad number: " + number;  
    }  
}
```

You must explicitly enable assertions: `java -ea AssertionExample`

```
Enter a number between 0 and 10: 12  
Exception in thread "main" java.lang.AssertionError: bad number: 12  
at AssertionExample.main(AssertionExample.java:8)
```