Ch 14 – Bitwise, Equals, toString, and Exceptions

Bitwise Operators

Java Bitwise Operators

- Java has six bitwise operators:

Symbol	Operator
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
~	Bitwise NOT
<<	LEFT SHIFT
>>	RIGHT SHIFT

Java AND and OR

AND operator (&)

A	B	A & B
0	0	0
0	1	0
1	0	0
1	1	1

OR operator (|)

A	B	A | B
0	0	0
0	1	1
1	0	1
1	1	1

Java XOR and NOT

XOR operator (^)

A	B	A ^ B
0	0	0
0	1	1
1	0	1
1	1	0

NOT operator (~)

A	~A
0	1
1	0

Binary to Decimal

Decimal	Binary	Decimal	Binary
0	0000b	8	1000b
1	0001b	9	1001b
2	0010b	10	1010b
3	0011b	11	1011b
4	0100b	12	1100b
5	0101b	13	1101b
6	0110b	14	1110b
7	0111b	15	1111b

Binary to Decimal

- 0-9 are used for decimal numbers (base-10):
  - 149 = $1*10^2 + 4*10^1 + 9*10^0$
- 0-1 are used for binary numbers (base-2):
  - 1010b = $1*2^3 + 0*2^2 + 1*2^1 + *2^0 = 8 + 2 = 10$
- Example:
  - 10111b in decimal?
  - $1*2^4 + 0*2^3 + 1*2^2 + 1*2^1 + 1*2^1 = 16 + 4 + 2 + 1 = 23$
  - What is 14 in binary?
  - $8 + 4 + 2 = 1*2^3 + 1*2^2 + 1*2^1 + 0*2^0 =$ 1110b

Bitwise Operator Examples

- 4-bit numbers:
  - 6 & 5 = 0110b & 0101b = 0100b = 4
  - 6 | 5 = 0110b | 0101b = 0111b = 7
  - 6 ^ 5 = 0110b ^ 0101b = 0011b = 3
  - ~6 = ~0110b = 1001b = 9
- 8-bit numbers:
  - 6 << 3 = 00000110b << 3 = 00110000b = 48 (6 * 8)
  - 48 >> 4 = 00110000b >> 4 = 00000011b = 3 (48 / 16)

Masking Operations

- Clearing bits:
  - x = 00101001b = 41
  - want to clear top 4-bits
  - x = x & 00001111b = x & 15 = 00001001b = 9
- Setting bits:
  - x = 00101001b = 41
  - want to set bottom 4-bits
  - x = x | 00001111b = x | 15 = 00101111b = 47

Methods (toString, equals)

CS163 Fall 2018

The `toString()` method

- tells Java how to convert an object into a `String`
- called when an object is printed or concatenated to a `String`:

```
Point p = new Point(7, 2);
System.out.println(”p: " + p);
```

  – Same as:

```
System.out.println("p: " + p.toString());
```

- Every class has a `toString()`, even if it isn't in your code.
  – The default is the class's name and a hex (base-16) hash-code:

```
Point@9e8c34
```

toString() implementation

```
public String toString() {
        code that returns a suitable String;
}
```

– Example: toString() method for our Student class:

```
public String toString(){
    return ”name: " + name+ "\n"
        + ”id: " + id + "\n"
        + ”average: " + average;
}
```

- // SHOW Eclipse example of Student class

toString in ArrayLists and other collections call toString automatically

- ArrayList<Student> students = new ArrayList<>();
- ...
- System.out.println(students);

- println(students) calls students.toString(), which automatically calls s.toString() for every point s

// SHOW Eclipse example of Student class

Primitive Equality

- Suppose we have two integers `i` and `j`
- How does the statement `i==j` behave?
- `i==j` if `i` and `j` contain the same value

Object Equality

- Suppose we have two pet instances `pet1` and `pet2`
- How does the statement `pet1==pet2` behave?

Object Equality

- Suppose we have two pet instances `pet1` and `pet2`
- How does the statement `pet1==pet2` behave?
- `pet1==pet2` is true if ***both*** refer to the ***same*** object
- The `==` operator checks if the ***addresses*** of the two objects are equal
- May not be what we want!

Object Equality - extended

- If you want a different notion of equality define your own `.equals()` method.
- Use `pet1.equals(pet2)` instead of `pet1==pet2`
- The default definition of `.equals()` is the value of `==`

```
but for Strings the contents are compared
```

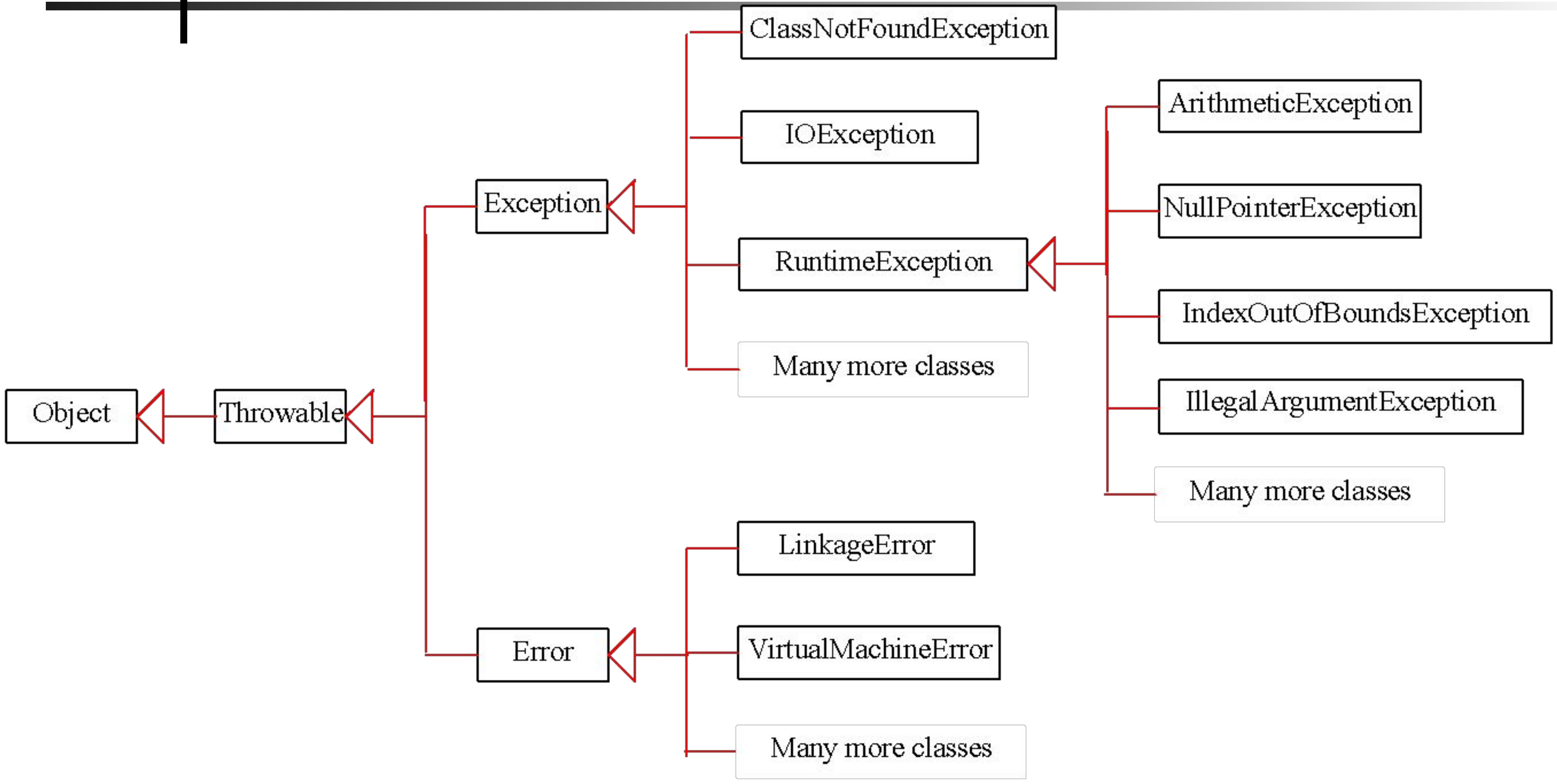
.equals for the Pet class

```
public boolean equals (Object other) {
    if (!other instanceof Pet) {
       return false;
   }
   Pet otherPet = (Pet) other;
    return ((this.age == otherPet.age)
     &&(Math.abs(this.weight - otherPet.weight) < 1e-8)
     &&(this.name.equals(otherPet.name)));
}
```

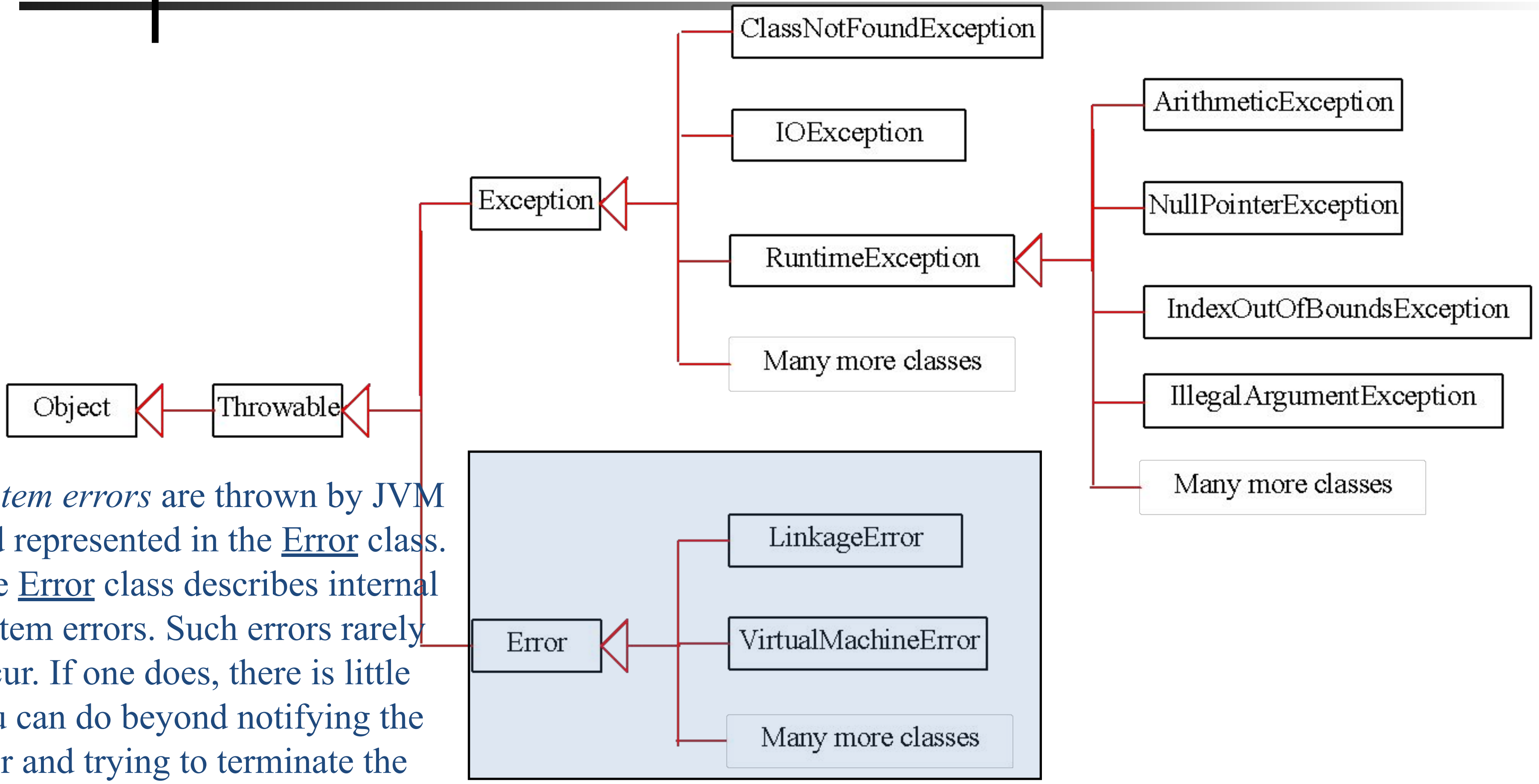
// SHOW ECLIPSE EXAMPLE OF Equals code.

Exceptions

Exception Types



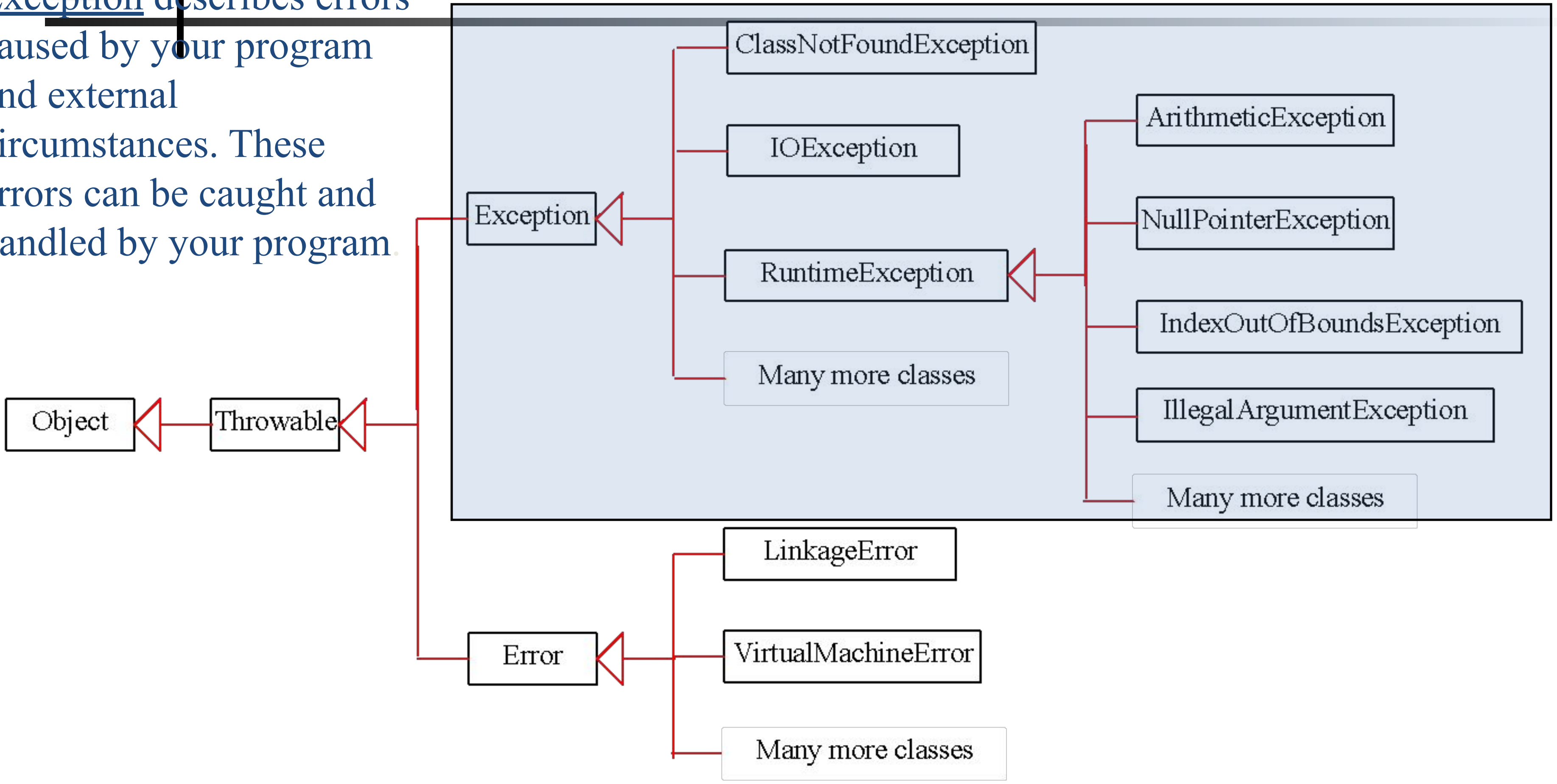
System Errors



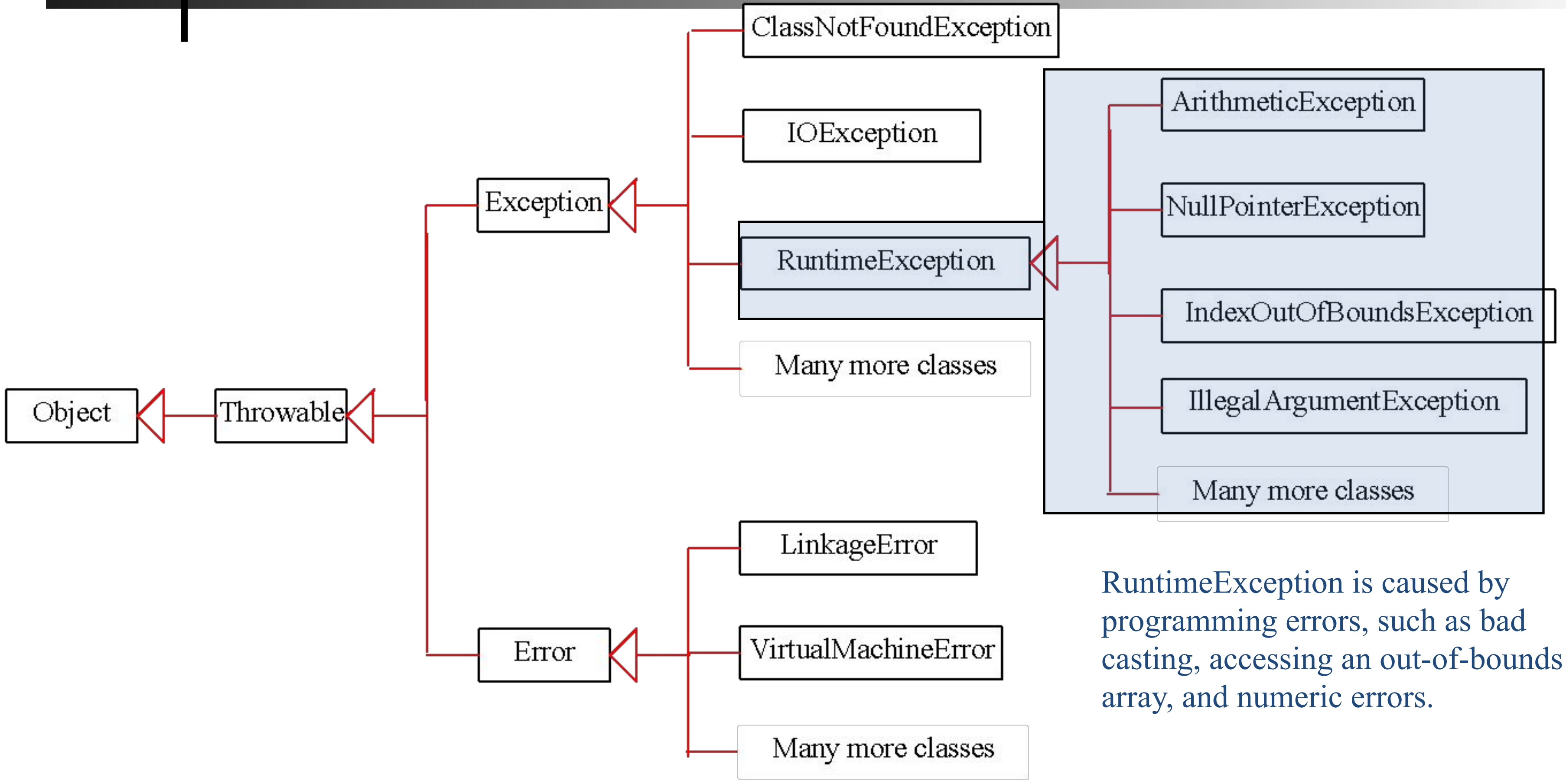
System errors are thrown by JVM and represented in the Error class. The Error class describes internal system errors. Such errors rarely occur. If one does, there is little you can do beyond notifying the user and trying to terminate the program gracefully.

Exceptions

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



Runtime Exceptions



RuntimeException is caused by programming errors, such as bad casting, accessing an out-of-bounds array, and numeric errors.

The `finally` Clause

```
try {
   statements;
}
catch(TheException ex) {
   handling ex;
}
finally {
   finalStatements;
}
```

animation

Trace a Program Execution

Suppose no exceptions in the statements

```
try {
  statements;
}
catch(TheException ex) {
  handling ex;
}
finally {
  finalStatements;
}

Next statement;
```

animation

Trace a Program Execution

```
try {
   statements;
}
catch(TheException ex) {
   handling ex;
}
finally {
   finalStatements;
}

Next statement;
```

The final block is always executed

animation

Trace a Program Execution

```
try {
   statements;
}
catch(TheException ex) {
   handling ex;
}
finally {
   finalStatements;
}

Next statement;
```

Next statement in the method is executed

Trace a Program Execution

```
try {
   statement1;
   statement2;
   statement3;
}
catch(Exception1 ex) {
   handling ex;
}
finally {
   finalStatements;
}

Next statement;
```

Suppose an exception of type Exception1 is thrown in statement2

animation

Trace a Program Execution

```
try {
   statement1;
   statement2;
   statement3;
}
catch(Exception1 ex) {
   handling ex;
}
finally {
   finalStatements;
}

Next statement;
```

The exception is handled.

Trace a Program Execution

```
try {
   statement1;
   statement2;
   statement3;
}
catch(Exception1 ex) {
   handling ex;
}
finally {
   finalStatements;
}

Next statement;
```

The final block is always executed.

animation

Trace a Program Execution

```
try {
   statement1;
   statement2;
   statement3;
}
catch(Exception1 ex) {
   handling ex;
}
finally {
   finalStatements;
}

Next statement;
```

The next statement in the method is now executed.

animation

Trace a Program Execution

```
try {
  statement1;
  statement2;
  statement3;
}
catch(Exception1 ex) {
  handling ex;
}
catch(Exception2 ex) {
  handling ex;
  throw ex;
}
finally {
  finalStatements;
}

Next statement;
```

statement2 throws an exception of type Exception2.

animation

Trace a Program Execution

```
try {
  statement1;
  statement2;
  statement3;
}
catch(Exception1 ex) {
  handling ex;
}
catch(Exception2 ex) {
  handling ex;
  throw ex;
}
finally {
  finalStatements;
}

Next statement;
```

Handling exception

Trace a Program Execution

```
try {
  statement1;
  statement2;
  statement3;
}
catch(Exception1 ex) {
  handling ex;
}
catch(Exception2 ex) {
  handling ex;
  throw ex;
}
finally {
  finalStatements;
}

Next statement;
```

Execute the final block

animation

Trace a Program Execution

```
try {
  statement1;
  statement2;
  statement3;
}
catch(Exception1 ex) {
  handling ex;
}
catch(Exception2 ex) {
  handling ex;
  throw ex;
}
finally {
  finalStatements;
}

Next statement;
```

Rethrow the exception and control is transferred to the caller

Writing Data Using PrintWriter

java.io.PrintWriter	
+PrintWriter(filename: String)	Creates a PrintWriter for the specified file.
+print(s: String): void	Writes a string.
+print(c: char): void	Writes a character.
+print(cArray: char[]): void	Writes an array of character.
+print(i: int): void	Writes an int value.
+print(l: long): void	Writes a long value.
+print(f: float): void	Writes a float value.
+print(d: double): void	Writes a double value.
+print(b: boolean): void	Writes a boolean value.
Also contains the overloaded println methods.	A println method acts like a print method; additionally it prints a line separator. The line separator string is defined by the system. It is \r\n on Windows and \n on Unix.
Also contains the overloaded printf methods.	The printf method was introduced in §4.6, "Formatting Console Output and Strings."

WriteData Run