CISC 3115 TY3

C13a: Exceptions

Hui Chen

Department of Computer & Information Science

CUNY Brooklyn College

Outline

- Error and error handling
  - Two approaches
- Exception
- The throwable class hierarchy
  - System errors and semantics
  - Runtime errors and semantics
  - Checked errors and semantics

Runtime Error

- When the JVM detects that an operation cannot be carried out
- Example: the divide-by-zero error

The Divide-by-Zero Error

```
import java.util.Scanner;

public class Quotient {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);

    System.out.println("Enter two integers: ");
    int n1 = input.nextInt();
    int n2 = input.nextInt();
    System.out.println(n1 + " / " + n2 + " is " + (n1 / n2));
  }
}
```

Handling Error: the What-IF Approach

- Use an if statement to check whether the input is valid
- Example:

```
if (number2 != 0) {
    System.out.println(number1 + " / " + number2 + " is " +
     (number1 / number2));
} else {
    System.out.println("Divisor cannot be zero ");
}
```

The What-IF Approach: Disadvantage?

- Use an if statement to check whether the input is valid
- Is there any disadvantage?
- Example:

```
if (number2 != 0) {
    System.out.println(number1 + " / " + number2 + " is " +
      (number1 / number2));
} else {
    System.out.println("Divisor cannot be zero ");
}
```

You must handle the error at the point where you detects it.

Handling Error: the Exception Approach

- Java supports Exception, representing an error or a condition that prevents execution from proceeding normally
- Example:

```
try {
    int result = quotient(n1, n2);
      ……
} catch (ArithmeticException e) {
    System.out.println("Divisor cannot be zero");
}
```

The Exception Approach: Advantage

- Separate notifying error from handling error

```
public static int quotient(int n1, int n2)
{
   if (n2 == 0) {
     throw new ArithmeticException(
           "Divisor cannot be zero.");
   }
   return n1 / n2;
 }
}
```

Notifying the caller an error occurred

```
public static void main(String[] args) {
                ……
  try {
    int result = quotient(n1, n2);
    System.out.println(n1 + " / " + n2 +
          " is "  + result);
  } catch (ArithmeticException e) {
    System.out.println("Exception: " +
          e.getMessage());
  }
```

Handling the error upon receiving the notification

Notifying Error

- Using throws
- Example

```
throw new ArithmeticException("Divisor cannot be zero.");
```

Handling Error

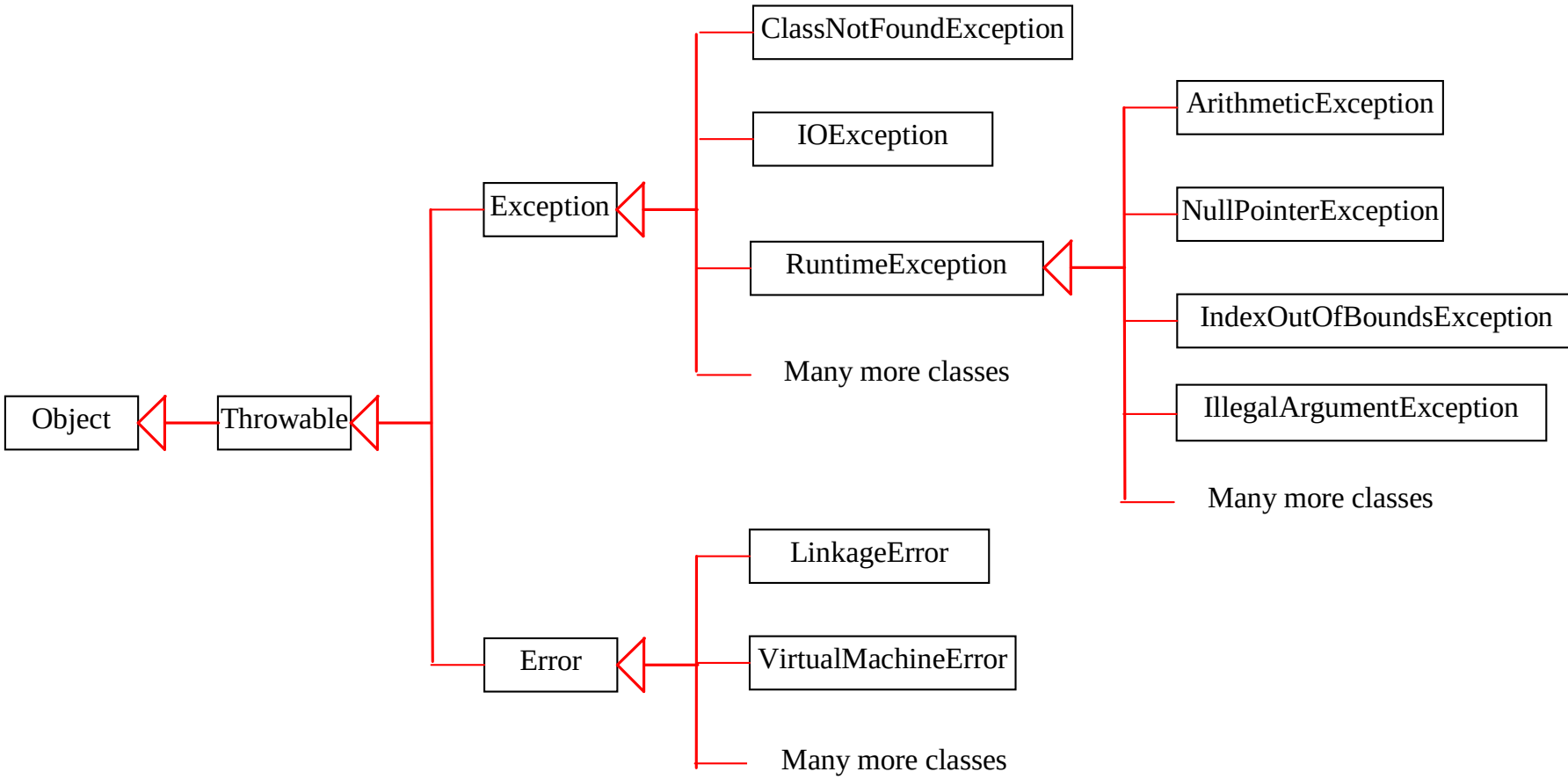
- Use try … catch …
- Example

```
try {
    int result = quotient(n1, n2);
    System.out.println(n1 + " / " + n2 +
     " is "  + result);
  } catch (ArithmeticException e) {
    System.out.println("Exception: " +
      e.getMessage());
  }
```

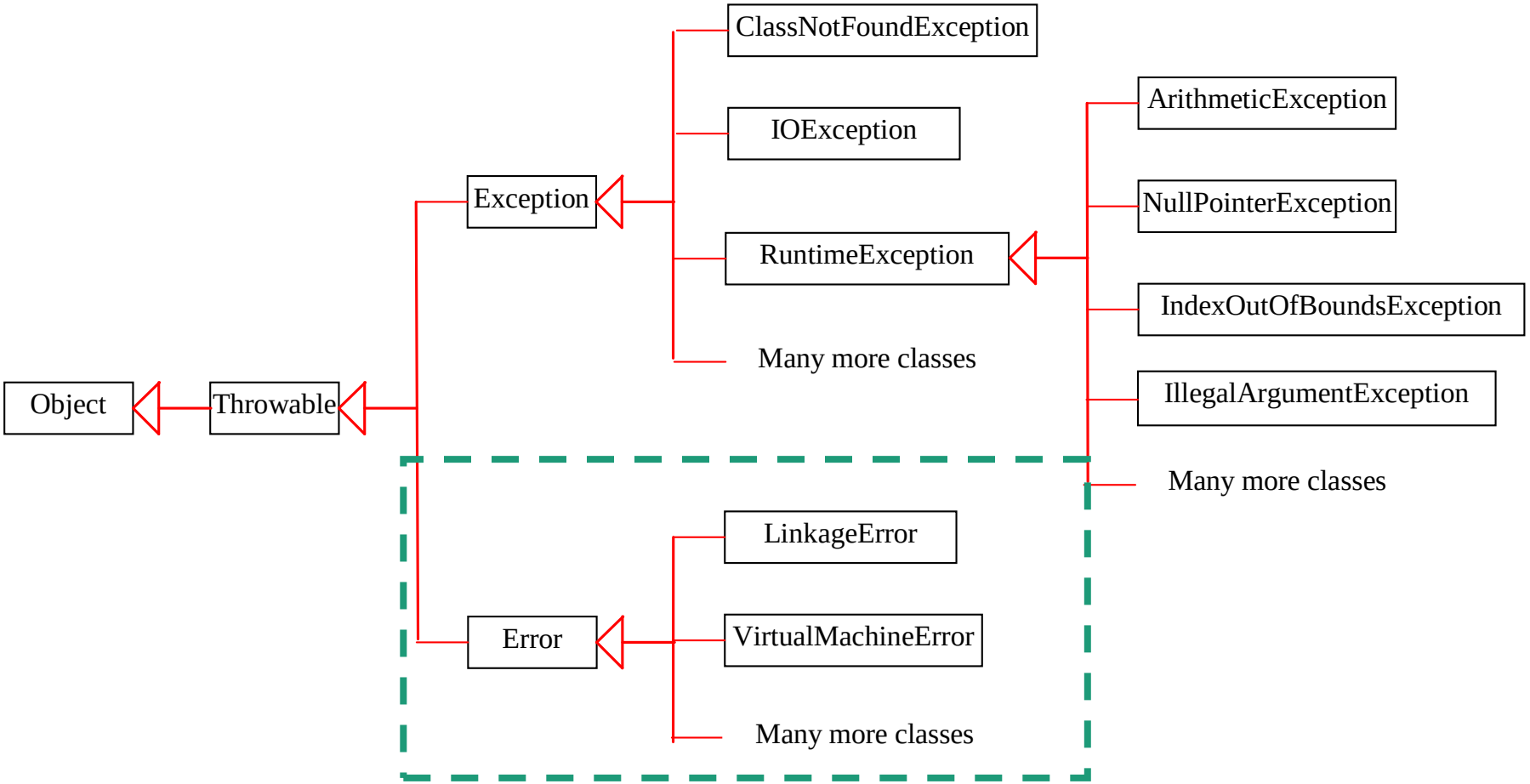
Types of Exceptions

- Java defines a list of exceptions and errors called Throwables that forms a class hierarchy
  - System Error
  - Runtime Exception

Exception Hierarchy



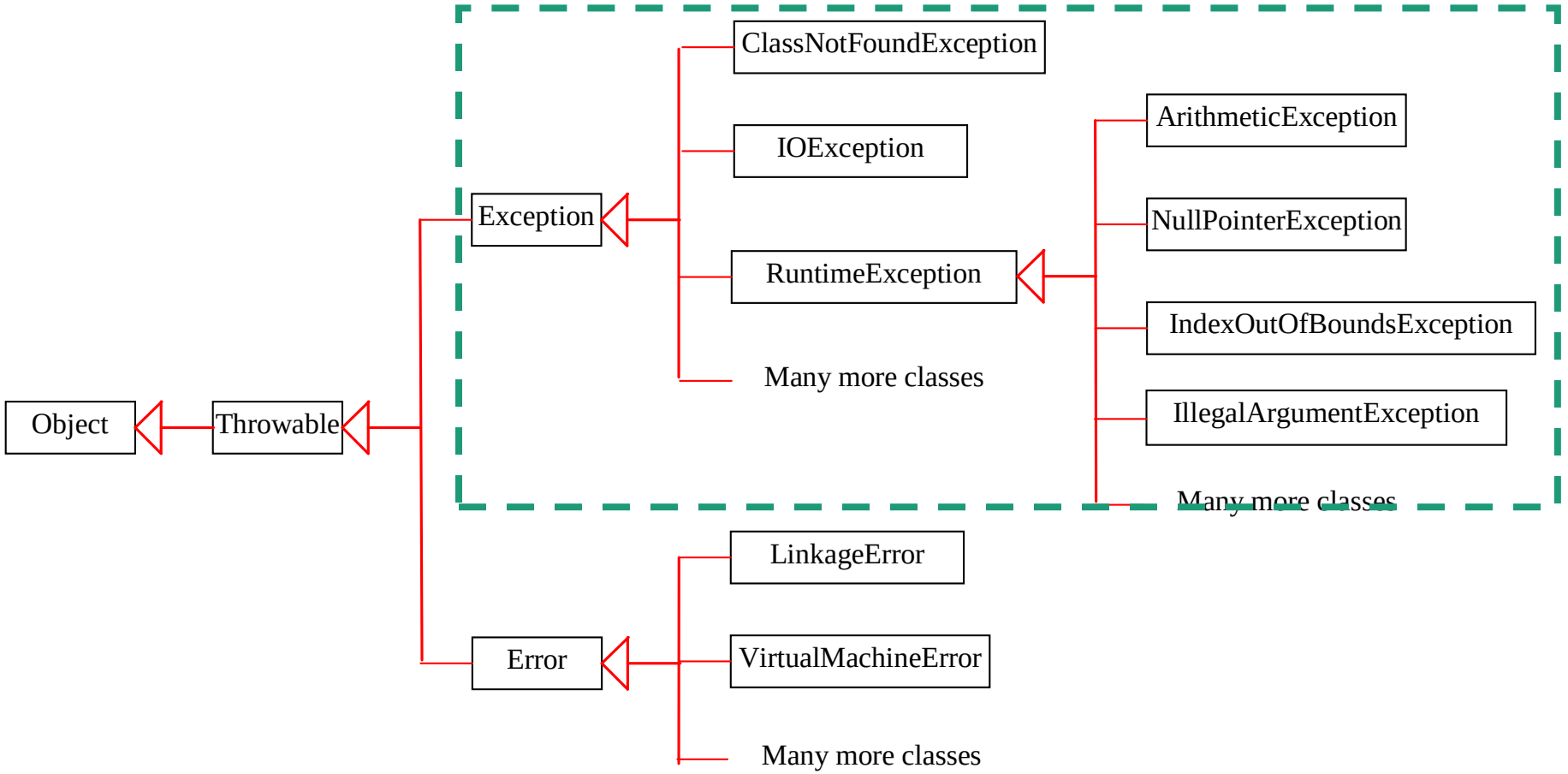
System Errors



Semantics of System Errors

- System errors are thrown by JVM
- System errors extend the Error class.
- Semantics: internal system error
  - Such errors rarely occur. If one does, there is little one can do beyond notifying the occurrence of the error and handling the termination of the program in a graceful manner.

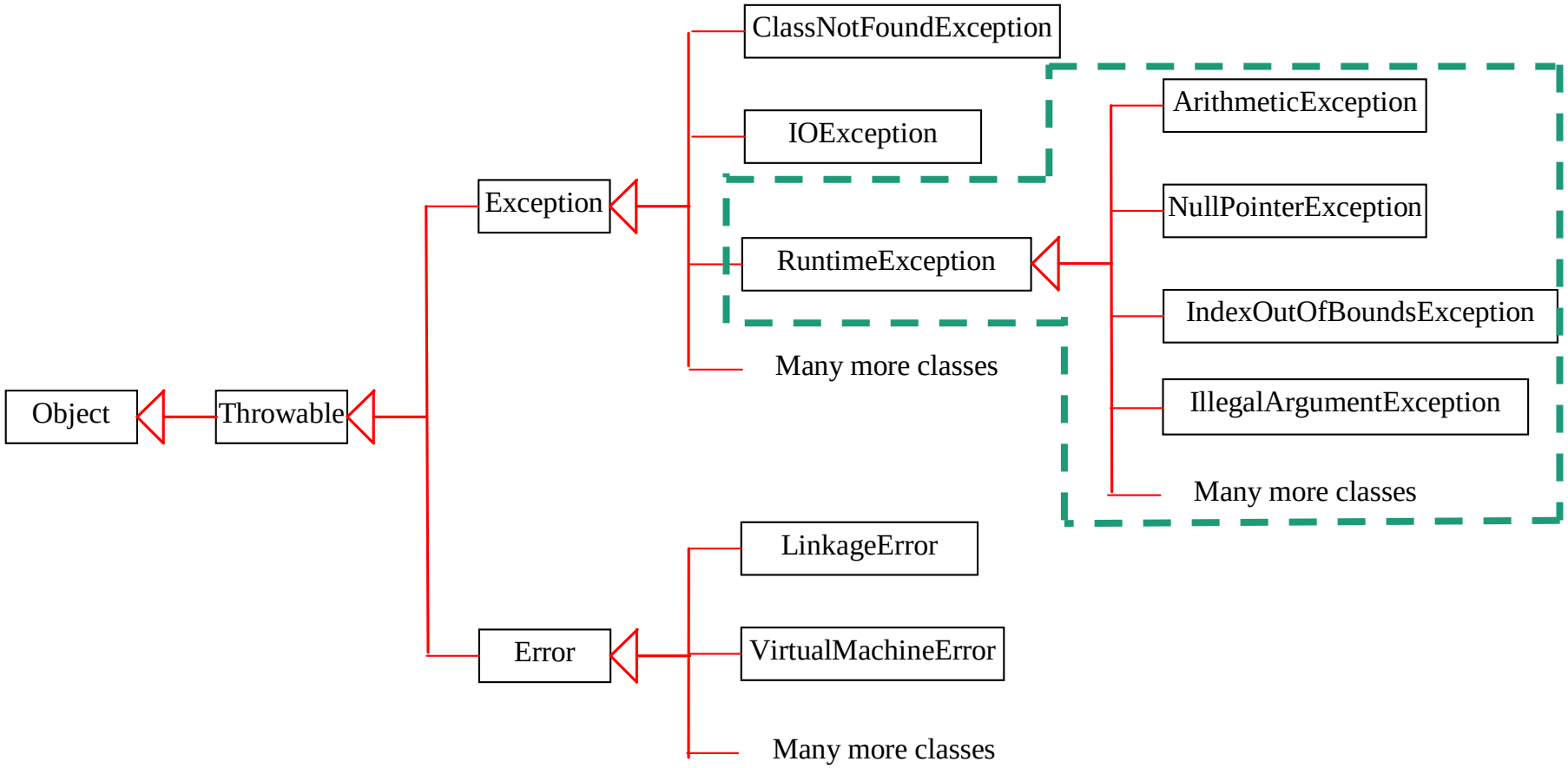
Exceptions


ClassNotFoundException
IOException
Exception
RuntimeException
ArithmeticException
NullPointerException
IndexOutOfBoundsException
IllegalArgumentException
Many more classes
Many more classes
Object
Throwable
LinkageError
Error
VirtualMachineError
Many more classes

Semantics of Exceptions

- Exception describes errors caused by your program and external circumstances.
- It is expected that one may recover from these errors or provide a meaningful intervention from careful handling of the errors.

Runtime Exceptions



Semantics of Runtime Exceptions

- RuntimeException is caused by programming errors, such as bad casting, accessing an out-of-bounds array, and numeric errors.
- That is to day, if we, as programmers did not make any mistakes, they should not have occurred.
  - How is it my mistake when a user of my program enters 0 for n2 in n1/n2?
  - As an awesome programmer as you are, you should have anticipated that a user may enter whatever she or he wishes to enter.
- Ideally, RuntimeExceptions should never occur in your program

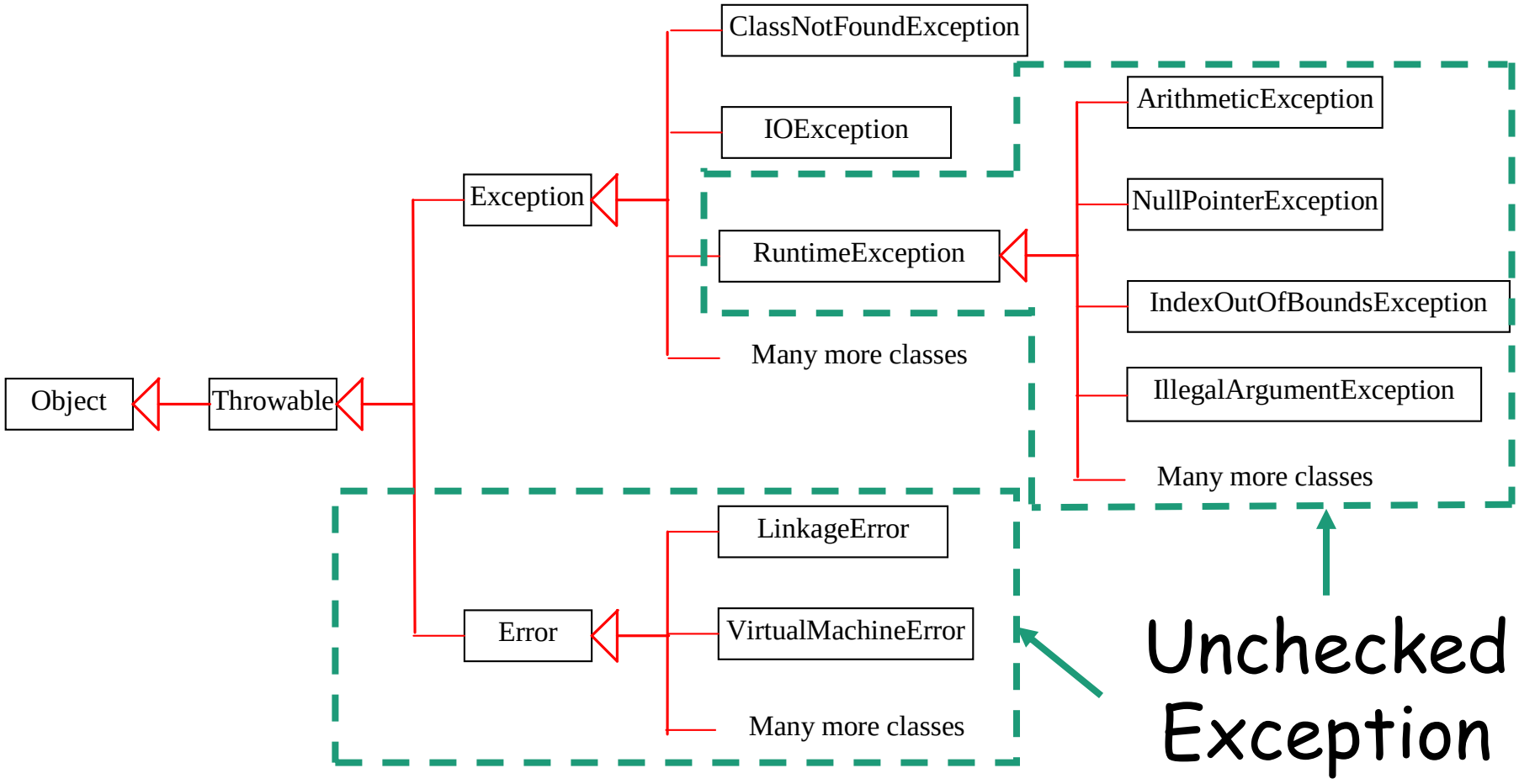
Questions?

- Class hierarchy of Throwable and subclasses
- Semantics of SystemError,
- Semantics of RuntimeException
- throws and try…catch…

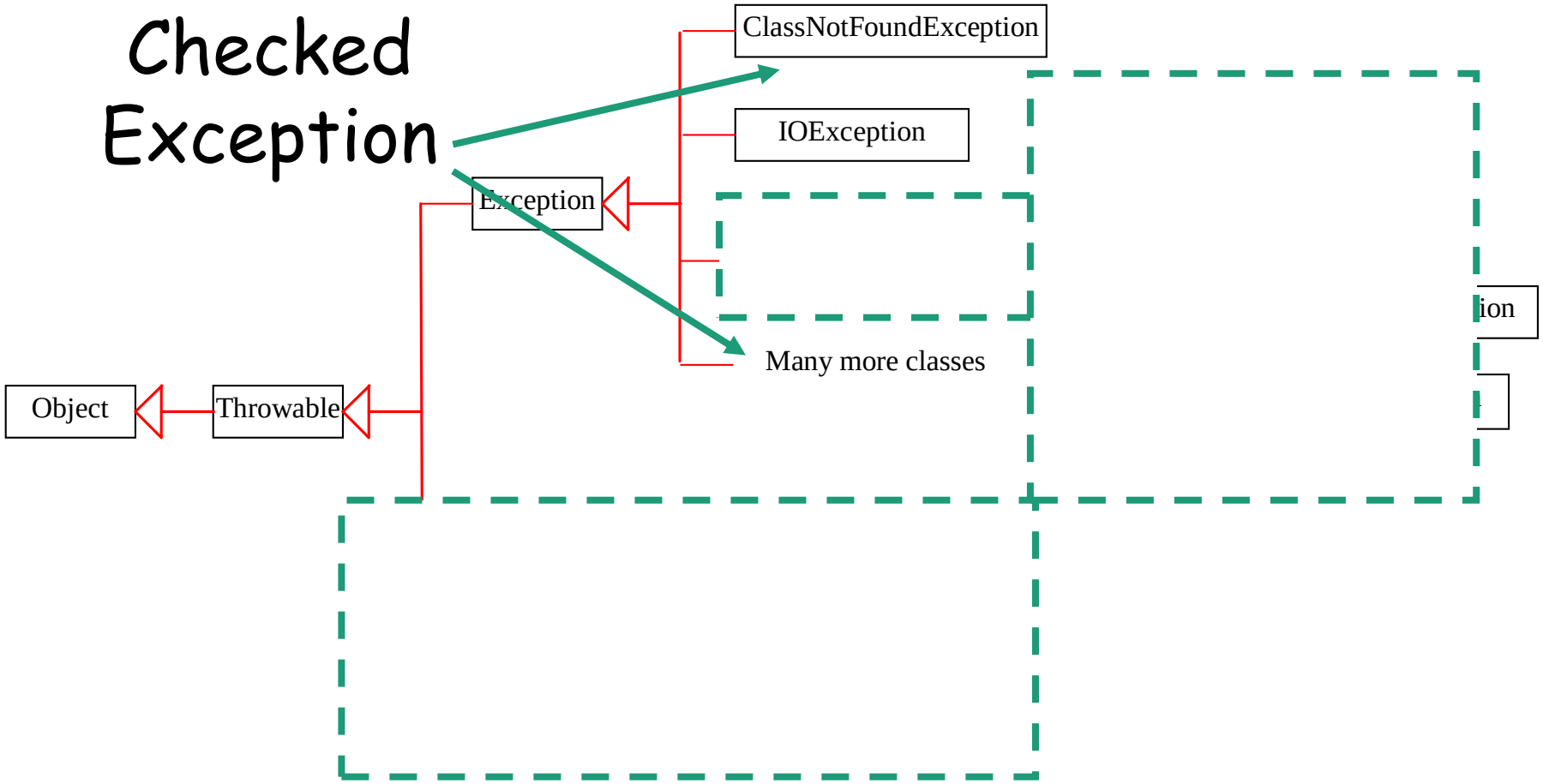
Checked and Unchecked Exceptions

- Unchecked Exceptions
  - RuntimeException, Error and their subclasses
- Checked Exceptions
  - Any others in the Throwable class hierarchy

Unchecked Exceptions



Checked Exceptions



Checked vs. Unchecked Exception

- The Java compiler forces the programmer to check and deal with the checked exceptions.
- The Java compiler does not forces the programmer to check and deal with the unchecked exceptions

Unchecked Exceptions

- Subclasses of Error and RuntimeException
- Programming logic errors that are not recoverable during runtime
- These are the logic errors that should be corrected in the program.
- They may occur anywhere in your program.

Unchecked Exceptions

- Two examples of commonly seen RuntimeExceptions
  - NullPointException
  - IndexOutOfBoundException

Questions

- The Throwable class hierarchy
- SystemError, RuntimeException
- Checked and unchecked exceptions
- NullPointException and IndexOutOfBoundException