NYU

Courant Institute of Mathematical Sciences
Department of Computer Science
CS101 Introduction to Computer Science

Anasse Bari, Ph.D.

Chapter#14: Java Exceptions

Objectives

- Introducing Exceptions in Java
- Catching Exceptions
- Retrieving Useful Information from Exception Methods

Introducing Exceptions in Java

- **An Exception is an error condition that disrupts the flow of a program.**
- There are a variety of exceptions in Java. Some can be handled to allow the program to continue while others prevent execution altogether.
- Runtime errors in Java are indicated by **exception objects.**
- Exceptions are **thrown** by methods to indicate that an error has occurred during execution.
- The calling method can either handle the exception or pass it to its caller. Exceptions should **ONLY** be thrown if the method cannot handle the error itself or if there is no better alternative.

Example One

Without Exceptions

```
public int divide ( int x, int y) {
  return x/y;
}
public static void main (String [] args ) {
  System.out.println(divide(78, 3));
  System.out.println(divide(78, 0));
  System.out.println("you should not see this");
}
```

Example One cont.

With Exceptions

```
public int divide ( int x, int y) throws ArithmeticException {
  if ( y ==0 ) throw new ArithmeticException ;
  return x/y;
}
public static void main (String [] args ) {
  try {
    System.out.println(divide(78, 3));
    System.out.println(divide(78, 0));
  }
  catch ( ArithmeticException e )
  {
    System.out.println("you should know better than trying to divide by zero");
  }
  System.out.println("this should be printed");
}
```

Example One cont.

- From the previous slide the method `divide()` does not know what to do if the divisor is zero.
- In this situation, the `divide()` method can tell `main` that something went wrong. The `main` can do the following:
  - Print an error message
  - Ignore the attempted division
  - Terminate the program
  - Ignore the exception
    - This passes the exception down the stack to the `main's` calling method and the JVM terminates the program with an error.

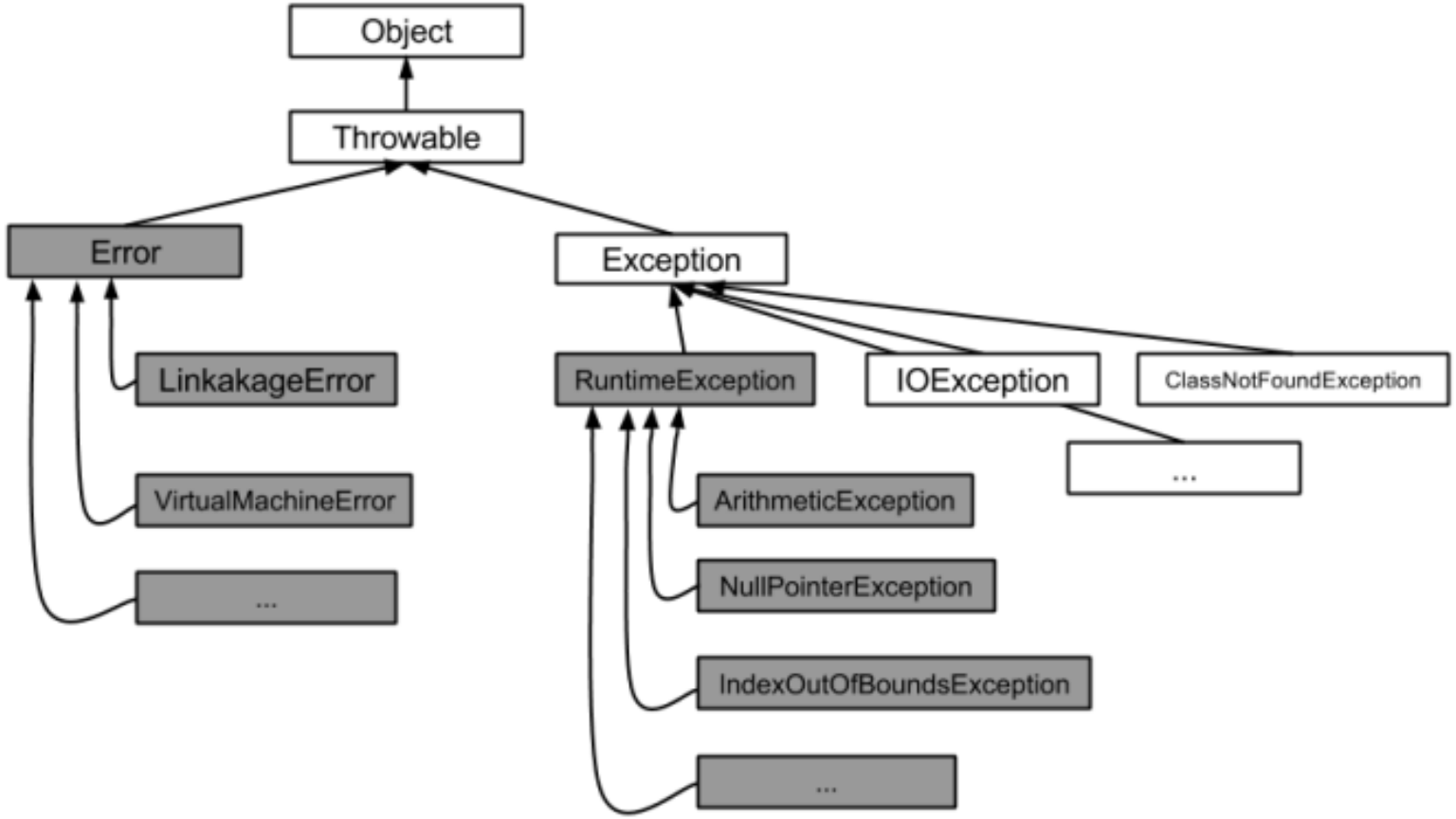
Syntax for Exceptions

- General Syntax for catching exceptions:

```
try {
  //code that may throw exception
}
catch ( ExceptionType1 e ) {
  // handle exception of type 1
}
catch ( ExceptionType2 e ) {
  // handle exception of type 2
}
...
catch ( ExceptionTypeN e ) {
  // handle exception of type N
}
```

Catching Exceptions

- Java will require some exceptions to be caught and handles, while others do not need to be. The figure below shows a sample inheritance tree for exception classes:



Syntax for Exceptions

- In previous diagram, the **Exception** class represents errors you are allowed to handle. And below are examples of exceptions. Generally, exceptions fall within one of two types:
  - **Unchecked Exceptions**
  - **Checked Exceptions**
- **Checked exceptions** require the program to catch or pass them to the calling methods. These are represented by the unshaded boxes in the diagram.
- **Unchecked exceptions** do not require the program to catch or handle them. These are represented by the shaded boxes in the diagram.

Retrieving Useful Information from Exceptions Methods

- The **Throwable** class provides methods to be used for when exceptions are caught:
  - `getMessage()`
  - `printStackTrace()`
  - `getStackTrace()`
- `getMessage()` returns a message, in the form of a `string` object, describing the exception object.
- `printStackTrace()` prints a copy of the stack trace information
- returns a copy of the stack trace information in the form of a StringgetStackTrace() `string` object.

More details on Exceptions to Read

- https://docs.oracle.com/javase/tutorial/essential/exceptions/
- https://www.tutorialspoint.com/java/java_exceptions.htm
- http://beginnersbook.com/2013/04/java-exception-handling/

NYU

Courant Institute of Mathematical Sciences
Department of Computer Science
CS101 Introduction to Computer Science

Anasse Bari, Ph.D.

Chapter#14: End of Lecture Exceptions