COSC 121: Computer Programming II

Today's Key Concepts

- An exception is an object that represents an error or a condition that prevents execution from proceeding normally
  - When dealt with properly, program execution continues
- Unchecked vs. Checked exceptions
  - Declaring exceptions via throws
  - Handling exceptions via try-catch or try-catch-finally

What are Java Exceptions?

- When a program runs into a runtime error, the program terminates abnormally
- How can you handle the runtime error so that the program can continue to run or terminate gracefully?

What are Java Exceptions?

- When a program runs into a runtime error, the program terminates abnormally
- How can you handle the runtime error so that the program can continue to run or terminate gracefully?
- Exception handling in Java is a mechanism to handle runtime errors, allowing the normal flow of a program to continue
- An exception is an object that represents an error or a condition that prevents execution from proceeding normally

What are Java Exceptions?

- When a program runs into a runtime error, the program terminates abnormally
- How can you handle the runtime error so that the program can continue to run or terminate gracefully?
- Exception handling in Java is a mechanism to handle runtime errors, allowing the normal flow of a program to continue
- An exception is an object that represents an error or a condition that prevents execution from proceeding normally
- Ex: A program tries to access an out-of-bounds array
  A program runs out of memory
  A program tries to read a file of integers but finds a string value in the file
  A program tries to connect to a website using an invalid address

Example Reading Array Content

```
import java.util.Scanner;
public class ArrayReading {
    public static void main(String[] args) {
        int[] numbers = {5, 7, 3};
        Scanner input = new Scanner(System.in);
        System.out.print("Enter the index: ");
        int i = input.nextInt();
        printElement(numbers, i);
        input.close();
    }
    private static void printElement(int[] numbers, int i) {
        System.out.println(numbers[i]);
    }
}
```

Output?

Example Reading Array Content

```
import java.util.Scanner;
public class ArrayReading {
    public static void main(String[] args) {
        int[] numbers = {5, 7, 3};
        Scanner input = new Scanner(System.in);
        System.out.print("Enter the index: ");
        int i = input.nextInt();
        printElement(numbers, i);
        input.close();
    }
    private static void printElement(int[] numbers, int i) {
        System.out.println(numbers[i]);
    }
}
```

Output?

```
Enter the index: 1
7
```

Example Reading Array Content

```
import java.util.Scanner;
public class ArrayReading {
    public static void main(String[] args) {
        int[] numbers = {5, 7, 3};
        Scanner input = new Scanner(System.in);
        System.out.print("Enter the index: ");
        int i = input.nextInt();
        printElement(numbers, i);
        input.close();
    }
    private static void printElement(int[] numbers, int i) {
        System.out.println(numbers[i]);
    }
}
```

Output?

```
Enter the index: 5
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5
        at ch12.ArrayReading.printElement(ArrayReading.java:14)
        at ch12.ArrayReading.main(ArrayReading.java:10)
```

Possible Fix

- Defensive programming approach – anticipate the error:

```
import java.util.Scanner;
public class ArrayReading {
    public static void main(String[] args) {
        int[] numbers = {5, 7, 3};
        Scanner input = new Scanner(System.in);
        System.out.print("Enter the index: ");
        int i = input.nextInt();
        printElement(numbers, i);
        input.close();
    }
    private static void printElement(int[] numbers, int i) {
        if(i >= numbers.length || i < 0)
            System.out.println("Wrong index!");
        else
            System.out.println(numbers[i]);
    }
}
```

verify index first

Another Possible Fix

- Using Java's exception handling:

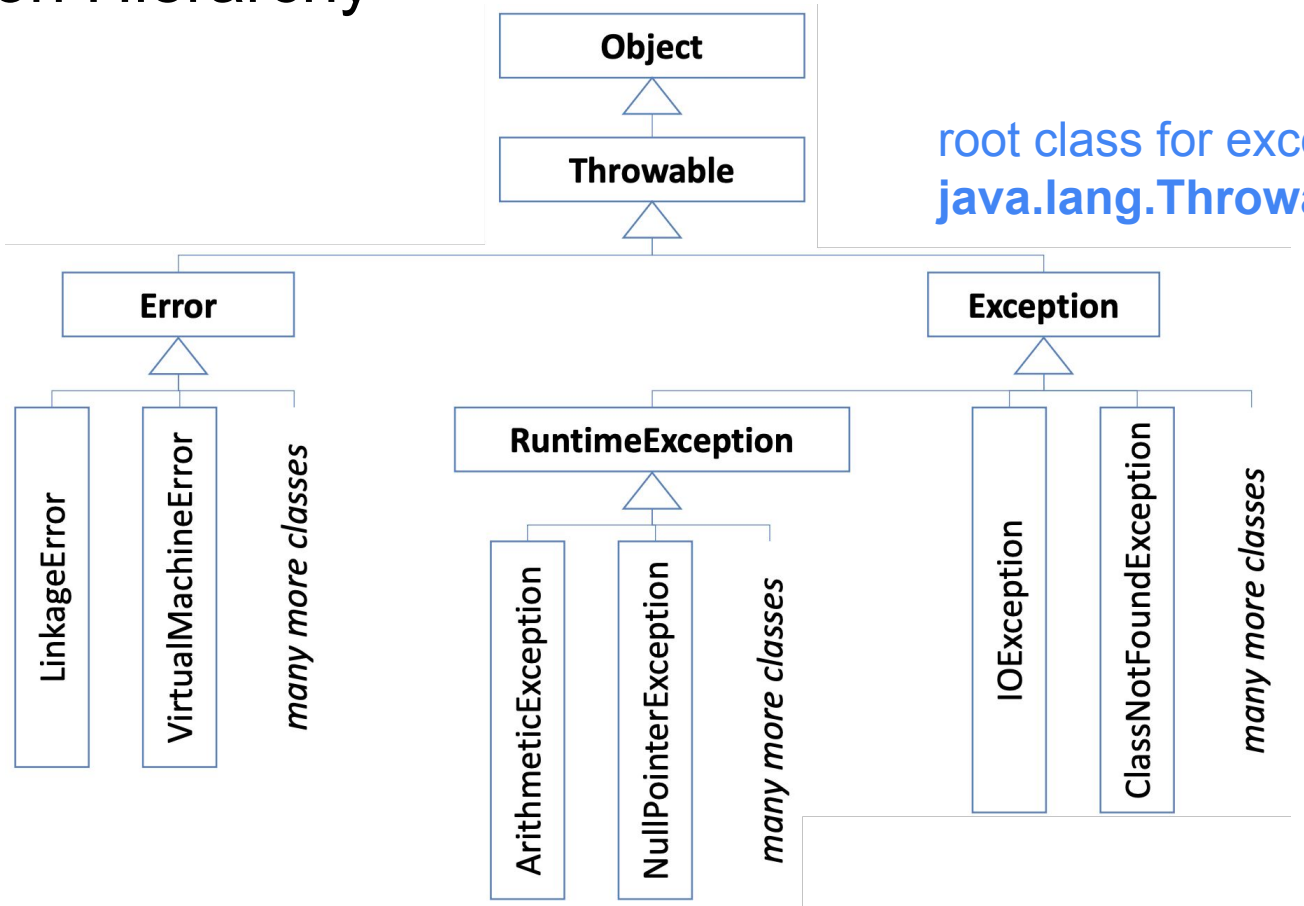
```
import java.util.Scanner;
public class ArrayReading2 {
    public static void main(String[] args) {
        int[] numbers = {5, 7, 3};
        Scanner input = new Scanner(System.in);
        System.out.print("Enter the index: ");
        int i = input.nextInt();
        printElement(numbers, i);
        input.close();
    }
    private static void printElement(int[] numbers, int i) {
        try{
            System.out.println(numbers[i]);
        } catch (IndexOutOfBoundsException e){
            System.out.println("Wrong index!");
        }
    }
}
```

same idea (details below)

How Exceptions are Handled

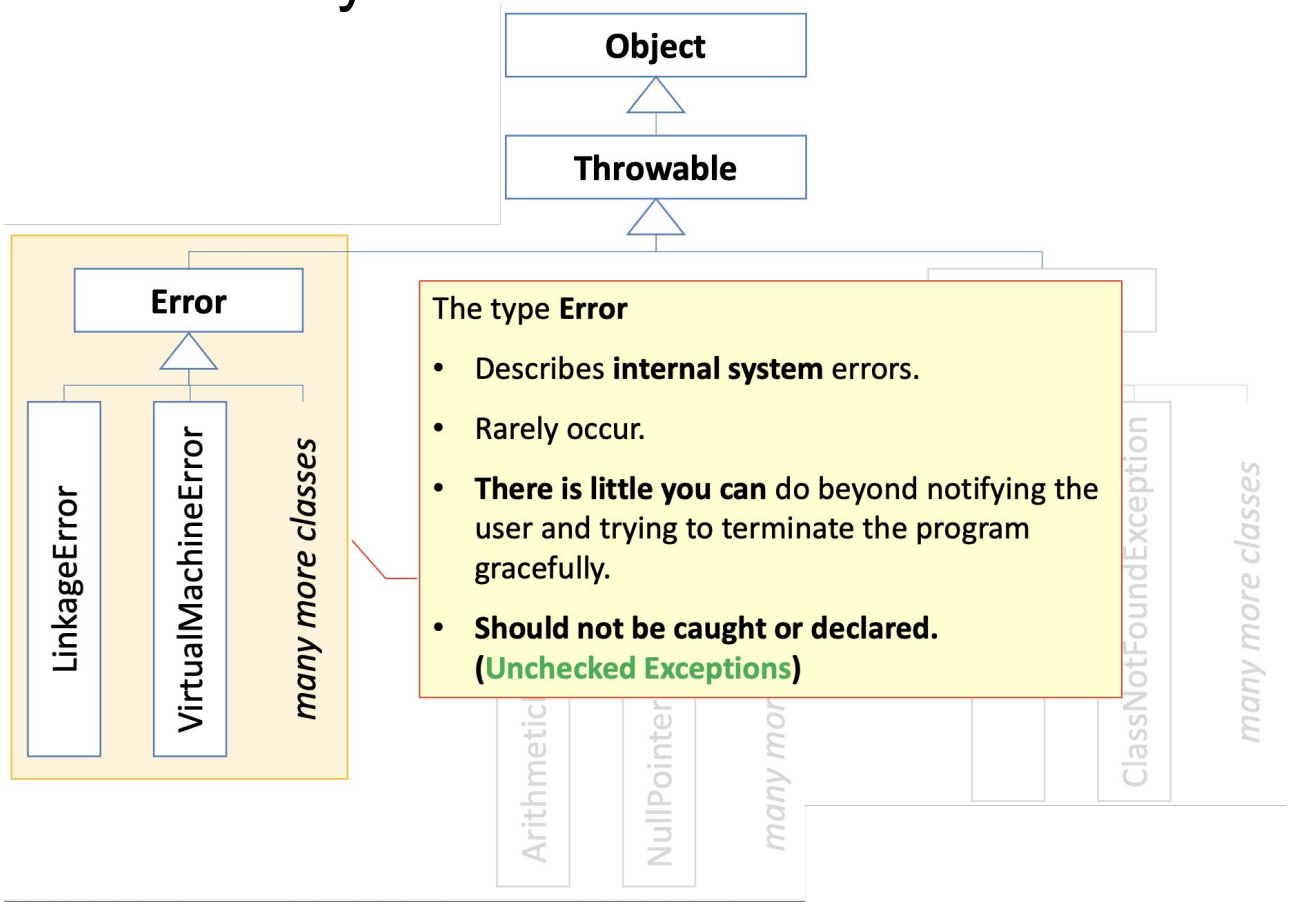
1. When an error is detected, Java stops the normal flow of execution
   - The code throws an exception
2. An Exception object is created
   - This object stores information about the error
   - There are different kinds of exception classes
3. Java transfers control from the erroneous part of the program to an exception handler, which deals with the situation
4. The Exception object is passed to the exception handler code where you decide on the appropriate action

Exception Hierarchy



root class for exceptions is **java.lang.Throwable**

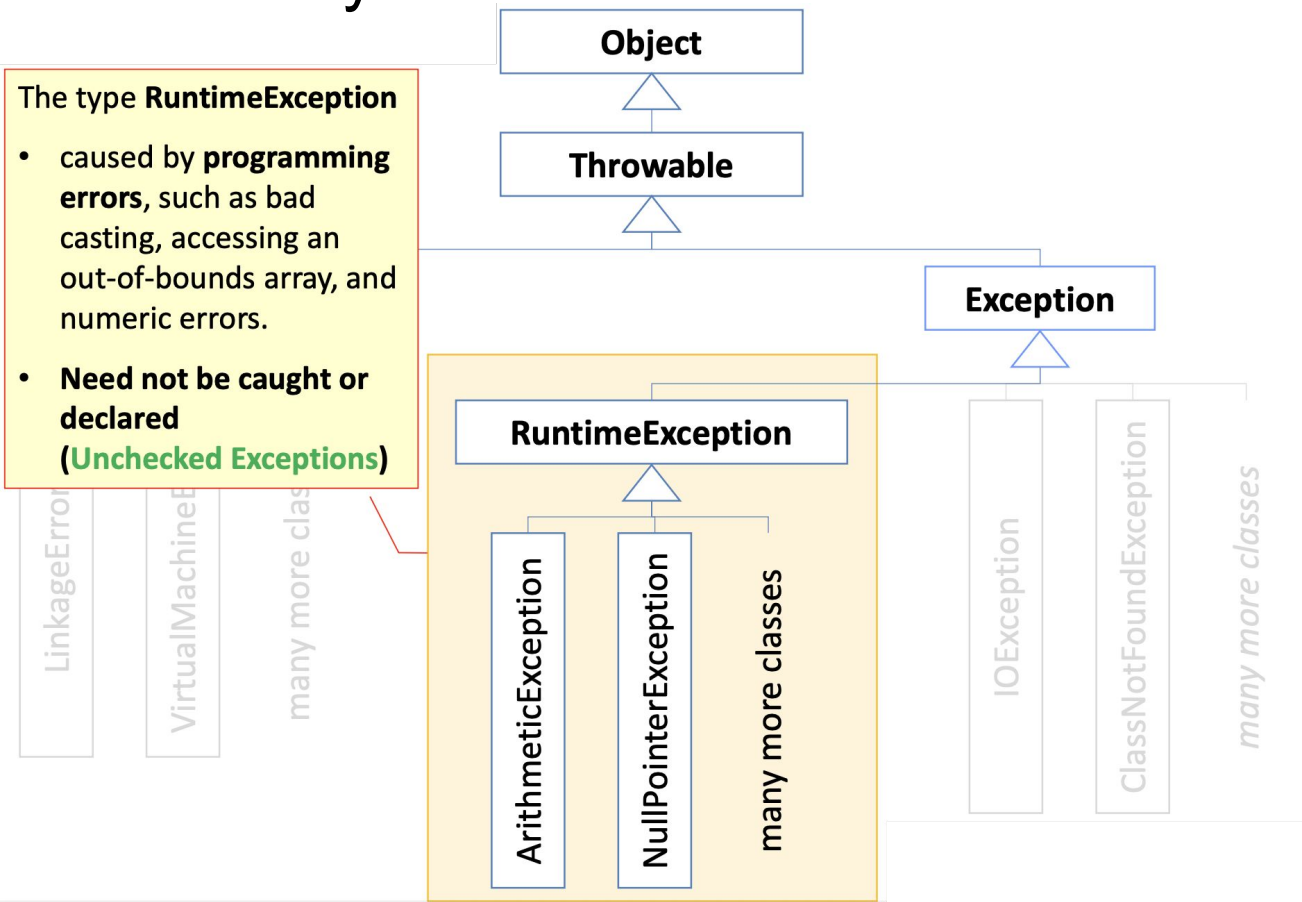
Exception Hierarchy



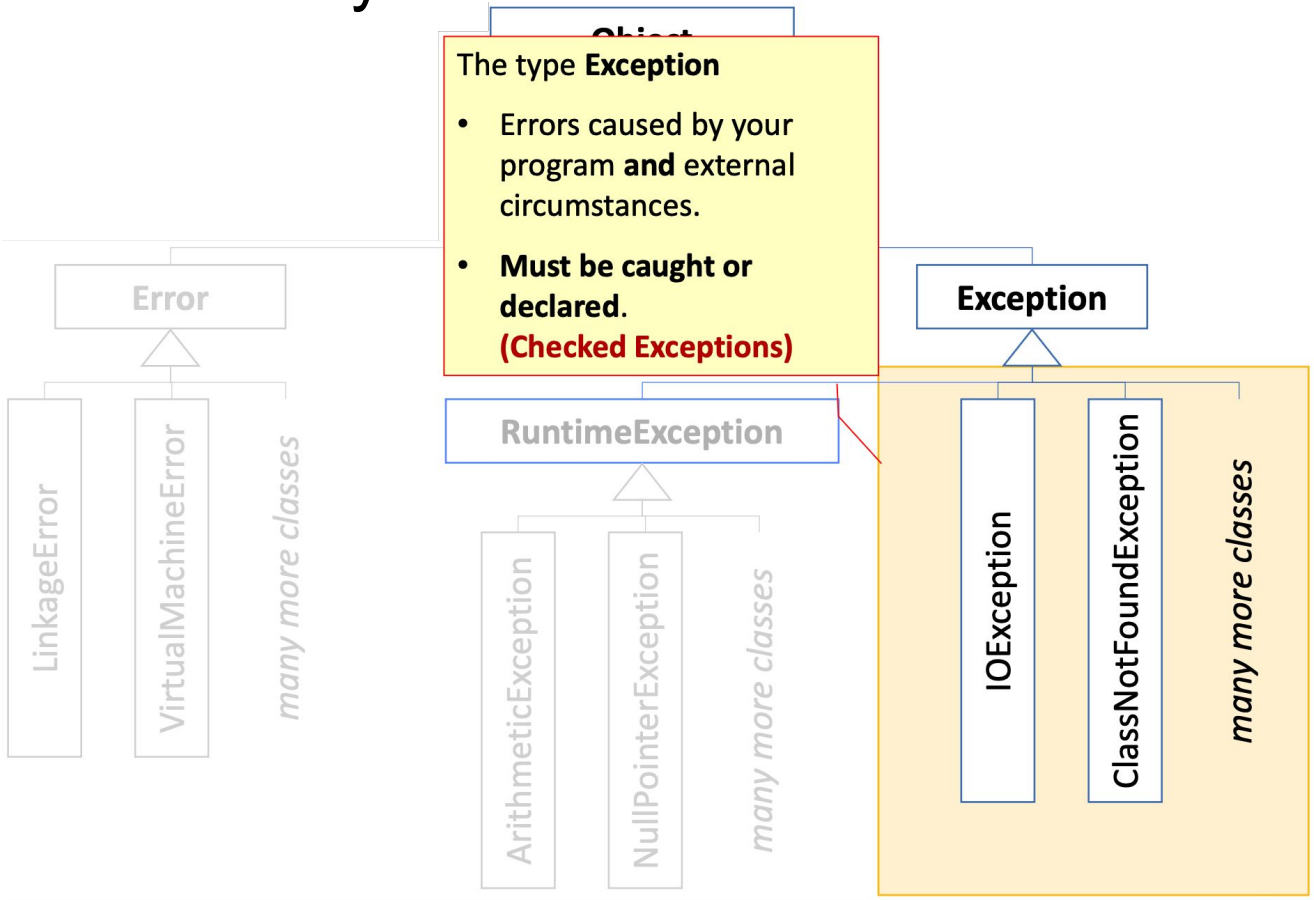
Exception Hierarchy

The type **RuntimeException**

- caused by **programming errors**, such as bad casting, accessing an out-of-bounds array, and numeric errors.
- **Need not be caught or declared (Unchecked Exceptions)**



Exception Hierarchy



Unchecked Exceptions

- These refer to Error, RuntimeException, and their subclasses
  - Can occur anywhere in a program, usually due to programming logic errors

Unchecked Exceptions

- These refer to Error, RuntimeException, and their subclasses
  - Can occur anywhere in a program, usually due to programming logic errors
- Example **Error** Subclass:
  - VirtualMachineError: JVM is broken or has run out of resources needed to continue operation

Unchecked Exceptions

- These refer to Error, RuntimeException, and their subclasses
  - Can occur anywhere in a program, usually due to programming logic errors
- Example **Error** Subclass:
  - VirtualMachineError: JVM is broken or has run out of resources needed to continue operation
- Example **RunTimeException** Subclasses:
  - ArithmeticException: Dividing an integer by zero
  - IndexOutOfBoundsException: Accessing an index outside the valid limits
  - NumberFormatException: When converting a String to a number, but the string does not have the appropriate format, e.g., int x = Integer.parseInt(s); //but s = "abc" instead of s = "1"
  - NullPointerException: Access object through reference var before an object is assigned to it

Checked Exceptions

- These refer to Exception subclasses except **RuntimeException**
  - Compiler forces you to check and deal with these
  - No default action when encountered (i.e. Java doesn’t throw nor catch the exception, needs you to tell it what to do)

Checked Exceptions

- These refer to Exception subclasses except **RuntimeException**
  - Compiler forces you to check and deal with these
  - No default action when encountered (i.e. Java doesn't throw nor catch the exception, needs you to tell it what to do)
- Examples of IOException Subclasses:
  - EOFException: Occurs when program attempts to read past the end of a file
  - FileNotFoundException: Occurs when program attempts to open or write to a file that cannot be found
  - MalformedURLException: Indicates an invalid form of URL has occurred

iClicker Question

All Java exceptions are instances of which class?

A. RuntimeException
B. Exception
C. Error
D. Throwable
E. NumberFormatException

iClicker Question

What type of exception will the following code throw?

```
public class Test {
   public static void main( String[] args ) {
      int[] list = new int[5];
      System.out.println( list[5] );
   }
}
```

A. ArithmeticException
B. ArrayIndexOutOfBoundsException
C. StringIndexOutOfBoundsException
D. ClassCastException
E. No exception

iClicker Question

What type of exception will the following code throw?

```
public class Q {
   public static void main( String[] args ) {
      Object obj = new Object();
      String str = ( String )obj;
   }
}
```

A. ArithmeticException
B. ArrayIndexOutOfBoundsException
C. StringIndexOutOfBoundsException
D. ClassCastException
E. No exception

Remember: How Exceptions are Handled

1. When an error is detected, Java **stops the normal flow** of execution
2. An Exception object is created
3. Java transfers control from the erroneous part of the program to an exception handler, which deals with the situation
4. The Exception object is passed to the exception handler code where **you decide on the appropriate action**

How You Need to Handle Exceptions

- For checked exceptions, you must do one of two things:
  - (1) Declaring exceptions: Declare in the method header that it may throws an exception
    In this case you do not handle the exception within the method, but pass it on to the calling method
  - (2) Handling exceptions: Catch the exception and deal with it within the method using a try-catch statement
- For unchecked exceptions, Java uses technique (1) by default (unless you write code to change that)

Declaring Exceptions

- Declaring an exception causes the methods where the exception occurs to throw the exception to the calling method
  - To declare an exception in a method, use the keyword throws followed by the type of exception in the method header
    Ex:
    ```
    public void m() throws IOException
    ```
    or
    ```
    public void m() throws Exception1, … , ExceptionN
    ```
    If the method might throw multiple exceptions

Where did we see this before?

Recall the Cloneable Interface

```
class Robot implements Cloneable {
    public int x, y;
    public Robot( int xpos, int ypos ) {
        x = xpos;
        y = ypos;
    }
    public String toString() {
        return "my coordinates: x = " + x + ", y = " + y;
    }
    public Object clone() throws CloneNotSupportedException {
        return super.clone();
    }
}
```

the clone() method

throws same exception

```
public class TestRobot {
    public static void main( String[] args )
            throws CloneNotSupportedException {
        Robot r1 = new Robot( 1, 2 );
        Robot r2 = ( Robot )r1.clone();
        // same coordinates
        System.out.println( r1.toString() );
        System.out.println( r2.toString() );
        // still point to different objects
        System.out.println( r1 == r2 );
    }
}
```

What Declaring Exceptions Means

- When you have: public Type method() throws Exception { … }
- You are telling the compiler:
  - This method may exit by throwing an Exception (or any subclass)
  - If it does, the caller must handle it or also declare it
- In the clone() example: body did not have code to handle the exception and the caller main() also had to declare the exception

Handling Exceptions

- To deal with an exception within a method, "catch it" by using try-catch
- Example: When this handler code is executed, the variable within the catch clause will contain a reference to the exception object that has been thrown

```
import java.util.Scanner;
public class ArrayReading2 {
    public static void main(String[] args) {
        int[] numbers = {5, 7, 3};
        Scanner input = new Scanner(System.in);
        System.out.print("Enter the index: ");
        int i = input.nextInt();
        printElement(numbers, i);
        input.close();
    }
    private static void printElement(int[] numbers, int i) {
        try{
            System.out.println(numbers[i]);
        } catch (IndexOutOfBoundsException e){
            System.out.println("Wrong index!");
        }
    }
}
```

try-catch statement

Handling Exceptions (cont.)

- Convenient trick: You can also catch an exception using a parent class of the
- thrown exception *Why?*

```
import java.util.Scanner;
public class ArrayReading2 {
    public static void main(String[] args) {
        int[] numbers = {5, 7, 3};
        Scanner input = new Scanner(System.in);
        System.out.print("Enter the index: ");
        int i = input.nextInt();
        printElement(numbers, i);

        input.close();
    }
    private static void printElement(int[] numbers, int i) {
        try{
            System.out.println(numbers[i]);
        } catch (Exception e){
            System.out.println("Wrong index!");
        }
    }
}
```

Handling Exceptions (cont.)

- Convenient trick: You can also catch an exception using a parent class of the
- thrown exception *Why?* Polymorphism!

```
import java.util.Scanner;
public class ArrayReading2 {
    public static void main(String[] args) {
        int[] numbers = {5, 7, 3};
        Scanner input = new Scanner(System.in);
        System.out.print("Enter the index: ");
        int i = input.nextInt();
        printElement(numbers, i);

        input.close();
    }
    private static void printElement(int[] numbers, int i) {
        try{
            System.out.println(numbers[i]);
        } catch (Exception e){
            System.out.println("Wrong index!");
        }
    }
}
```

Simple Exception Example

```
public class Ex1 {

    public static void main(String[] args) {
        System.out.println("Start of main method");
        divide(5,0);
        System.out.println("End of main method");
    }

    public static void divide(int n1, int n2) {
        System.out.println("Start of divide method");
        try {
            System.out.println("Begin of try");
            System.out.println(n1/n2);
            System.out.println("End of try");
        }catch(ArithmeticException e) {
            System.out.println("Arith Error.");
        }
        System.out.println("End of divide method");
    }
}
```

Output?

Simple Exception Example

```
public class Ex1 {

    public static void main(String[] args) {
        System.out.println("Start of main method");
        divide(5,0);
        System.out.println("End of main method");
    }

    public static void divide(int n1, int n2) {
        System.out.println("Start of divide method");
        try {
            System.out.println("Begin of try");
            System.out.println(n1/n2);
            System.out.println("End of try");
        }catch(ArithmeticException e) {
            System.out.println("Arith Error.");
        }
        System.out.println("End of divide method");
    }
}
```

Output:
Start of main method
Start of divide method
Begin of try

Simple Exception Example

```
public class Ex1 {

    public static void main(String[] args) {
        System.out.println("Start of main method");
        divide(5,0);
        System.out.println("End of main method");
    }

    public static void divide(int n1, int n2) {
        System.out.println("Start of divide method");
        try {
            System.out.println("Begin of try");
            System.out.println(n1/n2);
            System.out.println("End of try");
        }catch(ArithmeticException e) {
            System.out.println("Arith Error.");
        }
        System.out.println("End of divide method");
    }
}
```

Output:
Start of main method
Start of divide method
Begin of try
Arith Error.
End of divide method
End of main method

In Contrast …

When there is no try-catch handling:

Output?

```
public class Ex1 {

    public static void main(String[] args) {
        System.out.println("Start of main method");
        divide(5,0);
        System.out.println("End of main method");
    }

    public static void divide(int n1, int n2) {
        System.out.println("Start of divide method");
        System.out.println(n1/n2);
        System.out.println("End of divide method");
    }
}
```

In Contrast …

When there is no try-catch handling:

```
public class Ex1 {

    public static void main(String[] args) {
        System.out.println("Start of main method");
        divide(5,0);
        System.out.println("End of main method");
    }

    public static void divide(int n1, int n2) {
        System.out.println("Start of divide method");
        System.out.println(n1/n2);
        System.out.println("End of divide method");
    }
}
```

Output:
Start of main method
Start of divide method
Exception in thread …

Incorrect Exception Type

```
public class Ex1 {

    public static void main(String[] args) {
        System.out.println("Start of main method");
        divide(5,0);
        System.out.println("End of main method");
    }

    public static void divide(int n1, int n2) {
        System.out.println("Start of divide method");
        try {
            System.out.println("Begin of try");
            System.out.println(n1/n2);
            System.out.println("End of try");
        }catch(NullPointerException e) {
            System.out.println("Null Pointer Error");
        }
        System.out.println("End of divide method");
    }
}
```

Output?

Incorrect Exception Type

```
public class Ex1 {

    public static void main(String[] args) {
        System.out.println("Start of main method");
        divide(5,0);
        System.out.println("End of main method");
    }

    public static void divide(int n1, int n2) {
        System.out.println("Start of divide method");
        try {
            System.out.println("Begin of try");
            System.out.println(n1/n2);
            System.out.println("End of try");
        }catch(NullPointerException e) {
            System.out.println("Null Pointer Error");
        }
        System.out.println("End of divide method");
    }
}
```

Output:
Start of main method
Start of divide method
Begin of try
… will exception be caught?

Incorrect Exception Type

```
public class Ex1 {

    public static void main(String[] args) {
        System.out.println("Start of main method");
        divide(5,0);
        System.out.println("End of main method");
    }

    public static void divide(int n1, int n2) {
        System.out.println("Start of divide method");
        try {
            System.out.println("Begin of try");
            System.out.println(n1/n2);
            System.out.println("End of try");
        }catch(NullPointerException e) {
            System.out.println("Null Pointer Error");
        }
        System.out.println("End of divide method");
    }
}
```

Output:
Start of main method
Start of divide method
Begin of try
Exception in thread …

Using a Parent Exception Class

```
public class Ex1 {

    public static void main(String[] args) {
        System.out.println("Start of main method");
        divide(5,0);
        System.out.println("End of main method");
    }

    public static void divide(int n1, int n2) {
        System.out.println("Start of divide method");
        try {
            System.out.println("Begin of try");
            System.out.println(n1/n2);
            System.out.println("End of try");
        }catch(Exception e) {
            System.out.println("Error");
        }
        System.out.println("End of divide method");
    }
}
```

Output?

Using a Parent Exception Class

```
public class Ex1 {

    public static void main(String[] args) {
        System.out.println("Start of main method");
        divide(5,0);
        System.out.println("End of main method");
    }

    public static void divide(int n1, int n2) {
        System.out.println("Start of divide method");
        try {
            System.out.println("Begin of try");
            System.out.println(n1/n2);
            System.out.println("End of try");
        }catch(Exception e) {
            System.out.println("Error");
        }
        System.out.println("End of divide method");
    }
}
```

Output:
Start of main method
Start of divide method
Begin of try
… will exception be caught?

Using a Parent Exception Class

```
public class Ex1 {

    public static void main(String[] args) {
        System.out.println("Start of main method");
        divide(5,0);
        System.out.println("End of main method");
    }

    public static void divide(int n1, int n2) {
        System.out.println("Start of divide method");
        try {
            System.out.println("Begin of try");
            System.out.println(n1/n2);
            System.out.println("End of try");
        }catch(Exception e) {
            System.out.println("Error");
        }
        System.out.println("End of divide method");
    }
}
```

Output:
Start of main method
Start of divide method
Begin of try
Error
End of divide method
End of main method

Handling Exceptions (cont.)

- If more than one type of exception is possible, we can add additional catch clauses:

```
try {
  statements; // Statements that may throw exceptions
}
catch (Exception1 exVar1) {
  handler for exception1;
}
catch (Exception2 exVar2) {
  handler for exception2;
}
...
catch (ExceptionN exVarN) {
  handler for exceptionN;
}
```

- Each catch block is **examined in turn**, from first to last, to see whether the type of the Exception object is an instance of the Exception class in the catch block

Handling Exceptions (cont.)

- It is possible to have more than one try statement within a method
- This clearly defines the area of code where each exception is expected to arise:

```
try
{
    // code that may throw a FileNotFoundException
}
catch (FileNotFoundException ex)
{
    // code that handles a FileNotFoundException
}
//
// other code
//
try
{
    // code that may throw an EOFException
}
catch (EOFException ex)
{
    // code that handles an EOFException
}
```

one try-catch block

another try-catch block

Multiple Catch Statements

```
public class Ex1 {
    public static void main(String[] args) {
        System.out.println("Start of main method");
        try {
            divide(5,0);
        } catch(ArithmeticException e) {
            System.out.println("Arithmetic Error");
        }
        System.out.println("End of main method");
    }
    public static void divide(int n1, int n2) {
        System.out.println("Start of divide method");
        try {
            System.out.println("Begin of try");
            System.out.println(n1/n2);
            System.out.println("End of try");
        }catch(NullPointerException e) {
            System.out.println("Null Pointer Error");
        } catch(ClassCastException e) {
            System.out.println("Cast Error");
        } catch(Exception e) {  //ORDER MATTERS!!
            System.out.println("Undefined Error");
        }
        System.out.println("End of divide method");
}}
```

Output?

Multiple Catch Statements

```
public class Ex1 {
    public static void main(String[] args) {
        System.out.println("Start of main method");
        try {
            divide(5,0);
        } catch(ArithmeticException e) {
            System.out.println("Arithmetic Error");
        }
        System.out.println("End of main method");
    }
    public static void divide(int n1, int n2) {
        System.out.println("Start of divide method");
        try {
            System.out.println("Begin of try");
            System.out.println(n1/n2);
            System.out.println("End of try");
        }catch(NullPointerException e) {
            System.out.println("Null Pointer Error");
        } catch(ClassCastException e) {
            System.out.println("Cast Error");
        } catch(Exception e) {  //ORDER MATTERS!!
            System.out.println("Undefined Error");
        }
        System.out.println("End of divide method");
}}
```

Output:
Start of main method
Start of divide method
Begin of try

Multiple Catch Statements

```
public class Ex1 {
    public static void main(String[] args) {
        System.out.println("Start of main method");
        try {
            divide(5,0);
        } catch(ArithmeticException e) {
            System.out.println("Arithmetic Error");
        }
        System.out.println("End of main method");
    }
    public static void divide(int n1, int n2) {
        System.out.println("Start of divide method");
        try {
            System.out.println("Begin of try");
            System.out.println(n1/n2);
            System.out.println("End of try");
        }catch(NullPointerException e) {
            System.out.println("Null Pointer Error");
        } catch(ClassCastException e) {
            System.out.println("Cast Error");
        } catch(Exception e) {  //ORDER MATTERS!!
            System.out.println("Undefined Error");
        }
        System.out.println("End of divide method");
}}
```

Output:
Start of main method
Start of divide method
Begin of try
Undefined Error
End of divide method
… what about arithmetic error?

Multiple Catch Statements

```
public class Ex1 {
    public static void main(String[] args) {
        System.out.println("Start of main method");
        try {
            divide(5,0);
        } catch(ArithmeticException e) {
            System.out.println("Arithmetic Error");
        }
        System.out.println("End of main method");
    }
    public static void divide(int n1, int n2) {
        System.out.println("Start of divide method");
        try {
            System.out.println("Begin of try");
            System.out.println(n1/n2);
            System.out.println("End of try");
        }catch(NullPointerException e) {
            System.out.println("Null Pointer Error");
        } catch(ClassCastException e) {
            System.out.println("Cast Error");
        } catch(Exception e) {  //ORDER MATTERS!!
            System.out.println("Undefined Error");
        }
        System.out.println("End of divide method");
}}
```

Output:
Start of main method
Start of divide method
Begin of try
Undefined Error
End of divide method
End of main method

iClicker Question

What is displayed when the following program is run?

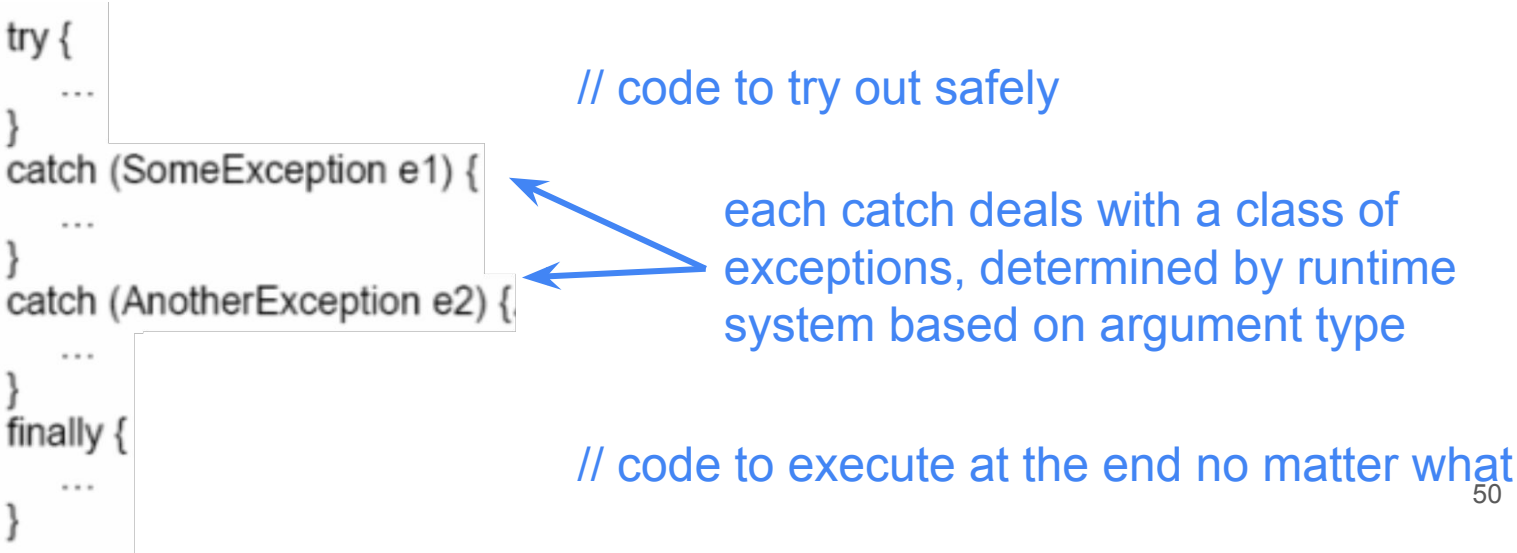
A. Arithmetic
B. Runtime
C. Exception
D. Error

E. Nothing, the program runs fine

```
public static void main(String[] args) {
    try {
            int[] list = new int[10];
            System.out.println(list[10]);
    } catch (ArithmeticException ex) {
            System.out.println("Arithmetic");
    } catch (RuntimeException ex) {
            System.out.println("Runtime");
    } catch (Exception ex) {
            System.out.println("Exception");
    }
}
```

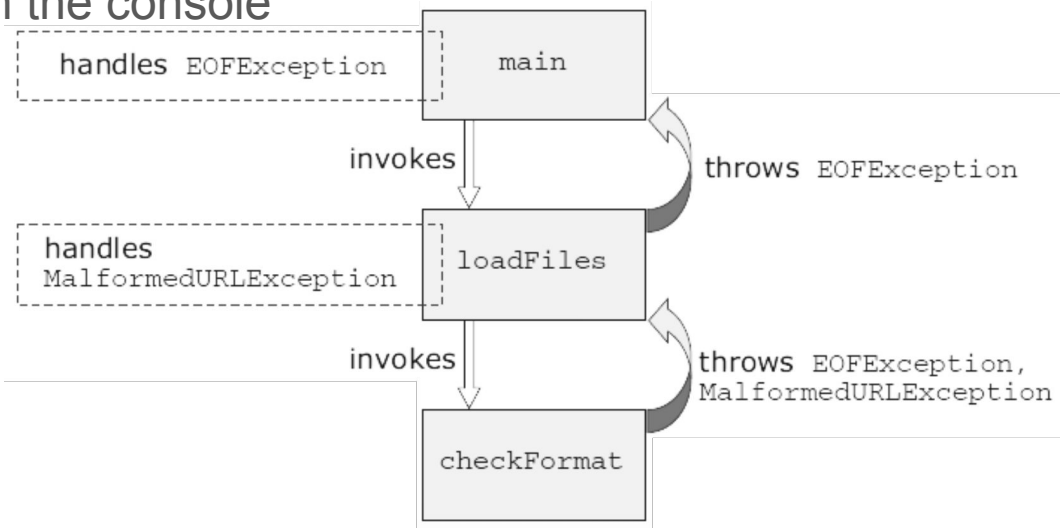
The finally Clause

- The finally clause is always executed regardless whether an exception occurred or not
- It may be used to clean up by releasing any resources (e.g., memory or files) that a method has been using before the exception was thrown



Exception Propagation

- If no handler is found in a method:
  - Java exits this method, passes the exception to the method that invoked it, continues the same process to find a handler up the call chain
- If no handler is found by the end, program terminates and prints an error message on the console



Getting Information from the Exception Object

- Instead of just printing the type of error caught, you might wish to let the programmer know something more informative
- You can use the following methods with an exception object to get valuable information about the exception:
  - String getMessage(): Returns a concise, human-readable string that explains why the exception was thrown
  - String toString(): Returns the "name of the exception class: getMessage()", which is the exception class name concatenated with the result of getMessage().
  - String printStackTrace(): Returns a detailed record of the sequence of method calls that led to an exception being thrown
- Ex: System.out.println("Error Message: " + e.getMessage());

Reviewing Flow of Execution

- Suppose statement2 causes an exception in the following code structure:

```
try {
  statement1;
  statement2;
  statement3;
} catch (Exception1 ex1) {
} catch (Exception2 ex2) {
}
statement4;
```

- Will statement3 run?
- If the exception is ***not*** caught, will statement4 run?
- If the exception is caught in one of the catch blocks, will statement4 run?

Reviewing Flow of Execution

- Suppose statement2 causes an exception in the following code structure:

```
try {
  statement1;
  statement2;
  statement3;
} catch (Exception1 ex1) {
} catch (Exception2 ex2) {
}
statement4;
```

- Will statement3 run? No
- If the exception is ***not*** caught, will statement4 run?
- If the exception is caught in one of the catch blocks, will statement4 run?

Reviewing Flow of Execution

- Suppose statement2 causes an exception in the following code structure:

```
try {
  statement1;
  statement2;
  statement3;
} catch (Exception1 ex1) {
} catch (Exception2 ex2) {
}
statement4;
```

- Will statement3 run? No
- If the exception is ***not*** caught, will statement4 run? No
- If the exception is caught in one of the catch blocks, will statement4 run?

Reviewing Flow of Execution

- Suppose statement2 causes an exception in the following code structure:

```
try {
  statement1;
  statement2;
  statement3;
} catch (Exception1 ex1) {
} catch (Exception2 ex2) {
}
statement4;
```

- Will statement3 run? No
- If the exception is ***not*** caught, will statement4 run? No
- If the exception is caught in one of the catch blocks, will statement4 run? Yes

Reviewing Flow of Execution with finally

- Suppose statement2 causes an exception in the following code structure:

```
try {
  statement1;
  statement2;
  statement3;
} catch (Exception1 ex1) {
} catch (Exception2 ex2) {
} finally {
 statement4;
}
statement5;
```

- If the exception is caught in one of the catch blocks, will statement4 and statement5 run?
- If the exception is ***not*** caught, will statement4 and statement5 run?

Reviewing Flow of Execution with finally

- Suppose statement2 causes an exception in the following code structure:

```
try {
  statement1;
  statement2;
  statement3;
} catch (Exception1 ex1) {
} catch (Exception2 ex2) {
} finally {
 statement4;
}
statement5;
```

- If the exception is caught in one of the catch blocks, will statement4 and statement5 run? Yes to both
- If the exception is ***not*** caught, will statement4 and statement5 run?

Reviewing Flow of Execution with finally

- Suppose statement2 causes an exception in the following code structure:

```
try {
  statement1;
  statement2;
  statement3;
} catch (Exception1 ex1) {
} catch (Exception2 ex2) {
} finally {
 statement4;
}
statement5;
```

- If the exception is caught in one of the catch blocks, will statement4 and statement5 run? Yes to both
- If the exception is ***not*** caught, will statement4 and statement5 run? Yes to statement4 but not statement5

iClicker Question

What is displayed when running this program with method() causing an exception?

A. After
   Exception in …
B. After Arithmetic Bye
C. Arithmetic Bye
D. Arithmetic Runtime Exception Bye
E. Arithmetic Runtime Exception

```
public static void main(String[] args) {
    try {
        method();
        System.out.print("After ");
    }catch (ArithmeticException ex) {
        System.out.print("Arithmetic ");
    }catch (RuntimeException ex) {
        System.out.print("Runtime ");
    }catch (Exception e) {
        System.out.print("Exception ");
    }
    System.out.print("Bye");
}
static void method() throws Exception {
    System.out.print(1 / 0);
}
```

iClicker Question

What is displayed when running this program with method() causing an exception?

A. Runtime2
B. Exception
C. Runtime1 After
D. Runtime2 After

```
public static void main(String[] args) {
   try {
         method();
         System.out.print("After ");
   } catch (ArithmeticException ex) {
         System.out.print("Runtime1 ");
   } catch (Exception e) {
         System.out.print("Exception ");
   }
}
static void method() throws Exception {
   try {
         System.out.println("abc".charAt(8));
   } catch (RuntimeException ex) {
         System.out.print("Runtime2 ");
   } catch (Exception ex) {
         System.out.print("IOException "); }
}
```

iClicker Question

What is displayed when running this (error-free) program?

A. 2Try Finally Bye
B. 2Try Catch Finally Bye
C. Catch Finally Bye
D. Finally Error in …
E. Error in …

```
public static void main(String[] args) {
        try {
          System.out.print(4/2);
          System.out.print("Try ");
        } catch (ArithmeticException ex) {
          System.out.print("Catch ");
        } finally {
          System.out.print("Finally ");
        }
        System.out.print("Bye ");
}
```

iClicker Question

What is displayed when running this program?

A. 2Try Finally Bye
B. 2Try Catch Finally Bye
C. Catch Finally Bye
D. Finally Error in …
E. Error in …

```
public static void main(String[] args) {
    try {
      System.out.print(1/0);
      System.out.print("Try ");
    } catch (ArithmeticException ex) {
      System.out.print("Catch ");
    } finally {
      System.out.print("Finally ");
    }
    System.out.print("Bye ");
}
```

iClicker Question

What is displayed when running this program?

A. 2Try Finally Bye
B. 2Try Catch Finally Bye
C. Catch Finally Bye
D. Finally Error in …
E. Error in …

```
public static void main(String[] args) {
    try {
      System.out.print(1/0);
      System.out.print("Try ");
    } catch (NullPointerException ex) {
      System.out.print("Catch ");
    } finally {
      System.out.print("Finally ");
    }
    System.out.print("Bye ");
}
```