

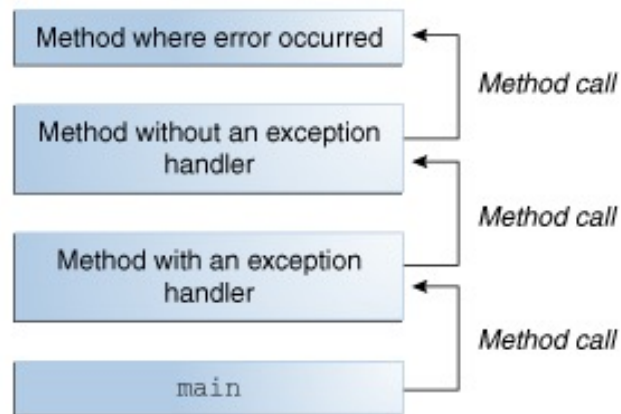
COMP2511

Exceptions in Java

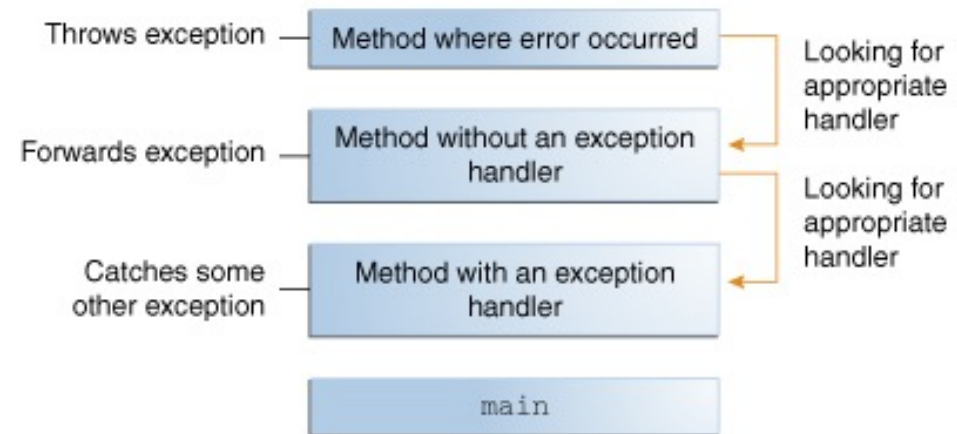
Prepared by
Dr. Ashesh Mahidadia

Exceptions in Java

- ❖ An **exception** is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions.
- ❖ When error occurs, an *exception* object is created and given to the runtime system, this is called **throwing** an exception.
- ❖ The runtime system searches the call stack for a method that contains a block of code that can **handle** the exception.
- ❖ The exception handler chosen is said to **catch** the exception.



The call stack.



Searching the call stack for the exception handler.

Exceptions in Java

The **Three Kinds** of Exceptions

- ❖ Checked exception (IOException, SQLException, etc.)
- ❖ Error (VirtualMachineError, OutOfMemoryError, etc.)
- ❖ *Runtime exception* (ArrayIndexOutOfBoundsExceptions, ArithmeticException, etc.)

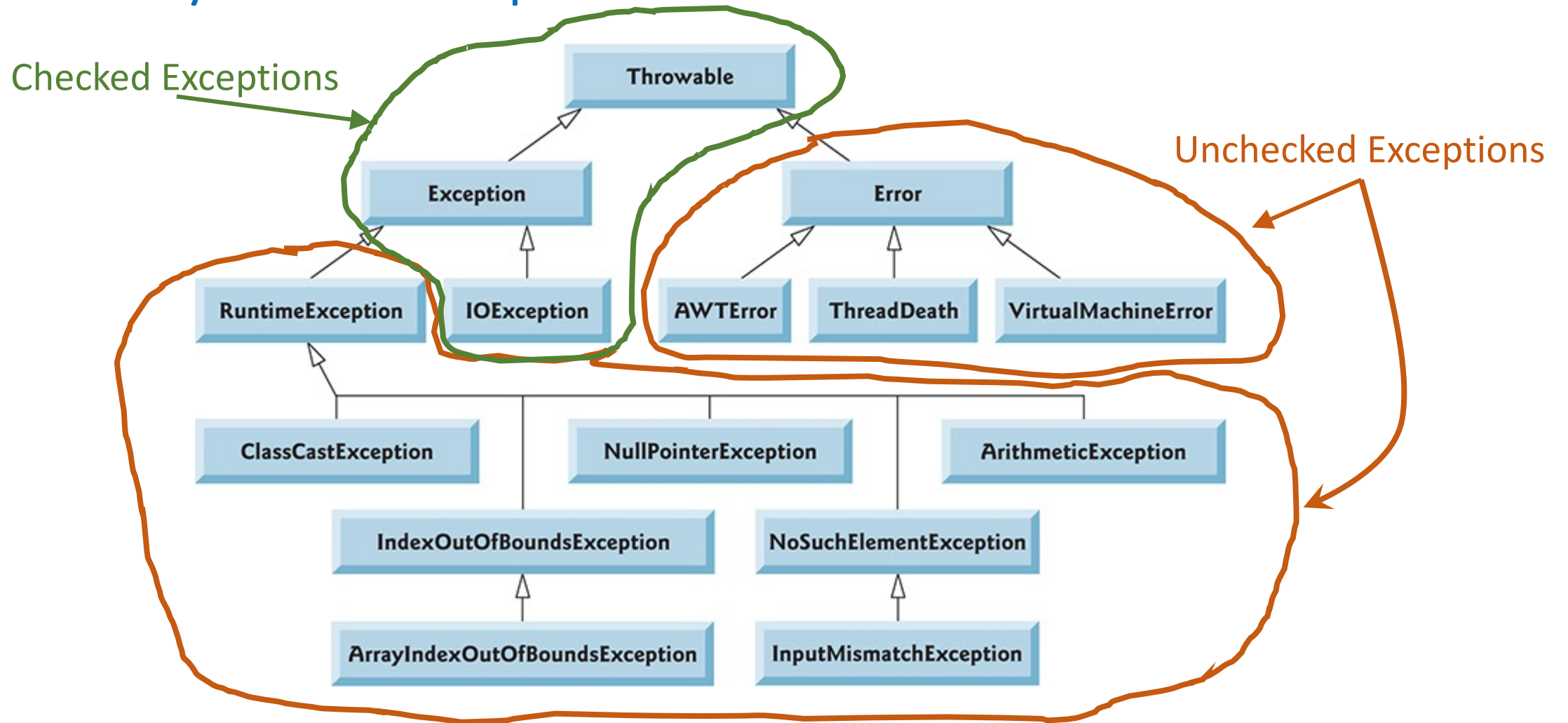
Checked vs. Unchecked Exceptions

- ❖ An exception's type determines whether it's checked or unchecked.
- ❖ All classes that are subclasses of *RuntimeException* (typically caused by defects in your program's code) or *Error* (typically 'system' issues) are **unchecked** exceptions.
- ❖ All classes that inherit from class *Exception* but not directly or indirectly from class *RuntimeException* are considered to be **checked** exceptions.

Exceptions in Java

- ❖ Good **introduction on Exceptions** at <https://docs.oracle.com/javase/tutorial/essential/exceptions/index.html>
- ❖ **Unchecked Exceptions — The Controversy** <https://docs.oracle.com/javase/tutorial/essential/exceptions/runtime.html>

Hierarchy of Java Exceptions



From the book "Java How to Program, Early Objects", 11th Edition, by Paul J. Deitel; Harvey Deitel

Example

```
public void writeList() {  
    PrintWriter out = null;  
  
    try {  
        System.out.println("Entering" + " try statement");  
  
        out = new PrintWriter(new FileWriter("OutFile.txt"));  
        for (int i = 0; i < SIZE; i++) {  
            out.println("Value at: " + i + " = " + list.get(i));  
        }  
    } catch (IndexOutOfBoundsException e) {  
        System.err.println("Caught IndexOutOfBoundsException: " + e.getMessage());  
    } catch (IOException e) {  
        System.err.println("Caught IOException: " + e.getMessage());  
    } finally {  
        if (out != null) {  
            System.out.println("Closing PrintWriter");  
            out.close();  
        } else {  
            System.out.println("PrintWriter not open");  
        }  
    }  
}
```

try

catch

finally

User Defined Exceptions in Java

- ❖ We can also create `user defined` exceptions.
- ❖ All exceptions must be a child of `Throwable`.
- ❖ A `checked` exception need to extend the `Exception` class,
but `not` directly or indirectly from class `RuntimeException`.
- ❖ An `unchecked` exception (like a runtime exception) need to extend the `RuntimeException` class.

User Defined / Custom Checked Exception

- Normally we define a *checked* exception, by extending the `Exception` class.

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


User Defined / Custom Exceptions: A Simple Example

```
try {
    out = new PrintWriter(new FileWriter("myData.txt"));
    for(int i=0; i<SIZE; i++){

        int idx = i + 5;

        if(idx >= SIZE){
            throw new MyException("idx is out of index range!");
        }
        out.println(list.get(idx));
    }
}
catch(IOException e){
    System.out.println(" In writeln ....");
}
catch(MyException e){
    System.out.println(e.getMessage());
}
catch(Exception e){
    System.out.println(" In writeln, Exception ....");
}
}
```

Exceptions in Inheritance

- ❖ If a subclass method **overrides** a superclass method, a subclass's **throws** clause can contain a subset of a superclass's **throws** clause.

It must **not** throw more exceptions!

- ❖ Exceptions are **part of** an **API** documentation and **contract**.

Demo: Exceptions in Java

Demo ...

Assertions in Java

- An **assertion** is a statement in the Java that enables you to test your assumptions about your program. Assertions are **useful** for checking:
 - Preconditions, Post-conditions, and Class Invariants (DbC!)
 - Internal Invariants and Control-Flow Invariants
- You should **not** use assertions:
 - for argument checking in **public methods**.
 - to do any work that your application requires for correct operation.
- Evaluating assertions should **not** result in side effects.
- The following document shows how to use **assertions in Java** :


<https://docs.oracle.com/javase/8/docs/technotes/guides/language/assert.html>

Important: for backward compatibility, by **default**, Java **disables** assertion validation feature. It needs to be explicitly **enabled** using the following command line argument:

- **-enableassertions** command line argument, or
- **-ea** command line argument

Assert : Example

```
/**
 * Sets the refresh interval (which must correspond to a legal frame rate).
 *
 * @param interval refresh interval in milliseconds.
 */
private void setRefreshInterval(int interval) {
    // Confirm adherence to precondition in nonpublic method
    assert interval > 0 && interval <= 1000/MAX_REFRESH_RATE : interval;
    ~~~~~ // Set the refresh interval
}
```



Exceptions: Summary Points

- ❖ Consider your exception-handling and error-recovery strategy in the **design process**.
- ❖ Sometimes you can **prevent an exception** by validating data first.
- ❖ If an exception can be handled meaningfully in a method, the method should **catch** the exception **rather than declare** it.
- ❖ If a subclass method overrides a superclass method, a subclass's **throws** clause can contain a subset of a superclass's **throws** clause. It must not throw more exceptions!
- ❖ Programmers should **handle checked** exceptions.
- ❖ If **unchecked** exceptions are **expected**, you must handle them **gracefully**.
- ❖ Only the **first** matching **catch** is executed, so select your catching class(es) carefully.
- ❖ Exceptions are part of an API documentation and contract.
- ❖ Assertions can be used to check preconditions, post-conditions and invariants.