

Exceptions

Throwing an Exception

```
public class SSN {  
    private String SSNumber;  
  
    public SSN (String s) {  
        if (isValidSSN(s) )  
            SSNumber = s;  
        else  
            throw new IllegalArgumentException("Invalid SSN: "+s);  
    }  
    ...  
}
```

Defining a new Exception

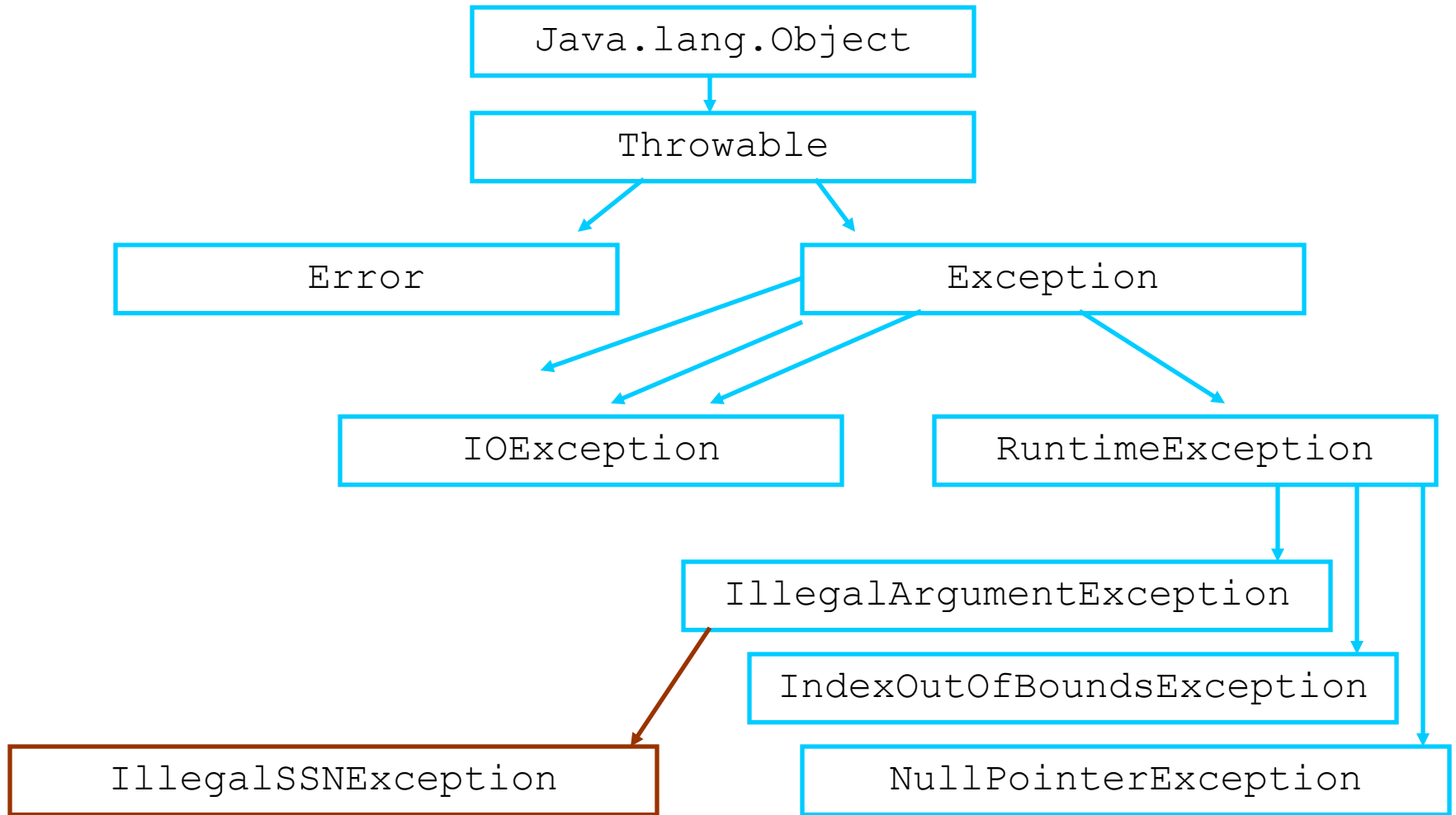
```
public class SSN {  
    private String SSNumber;  
  
    public SSN (String s) {  
        if (isValidSSN(s) )  
            SSNumber = s;  
        else  
            throw new IllegalArgumentException("Invalid SSN: "+s);  
    }  
    ...  
}
```

This exception is not in the Java class library.

Extend an Existing Exception

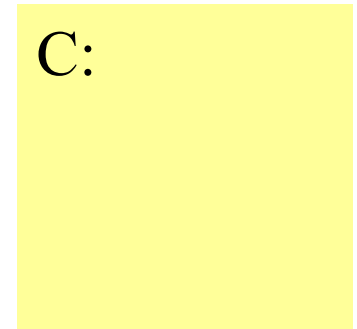
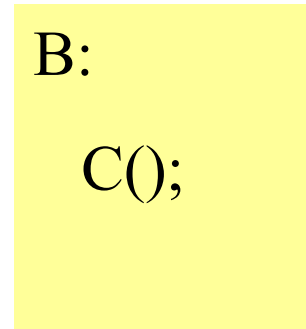
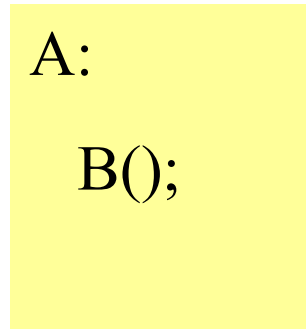
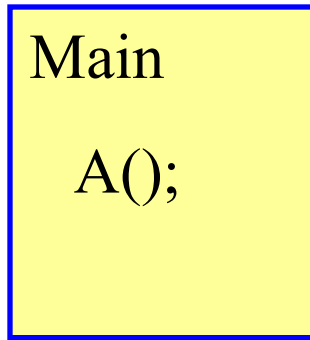
```
public class IllegalSSNException
    extends IllegalArgumentException {
    public IllegalSSNException(String message) {
        super (message);
    }
}
```

Exception Hierarchy



What Happens When an Exception is Thrown?

- The Runtime System looks for a method that can handle the exception
- If no such method is found, the Runtime System handles the exception and terminates the program
- The Runtime System looks at the most recently called method, and backs up all the way to the main program



Main program starts



Runtime Stack

Main

A();

A:

B();

B:

C();

C:

Main program calls A

A
Main

Runtime Stack

Main

A();

A:

B();

B:

C();

C:

A calls B

B

A

Main

Runtime Stack

Main

A();

A:

B();

B:

C();

C:

B calls C

C

B

A

Main

Runtime Stack

Main

A();

A:

B();

B:

C();

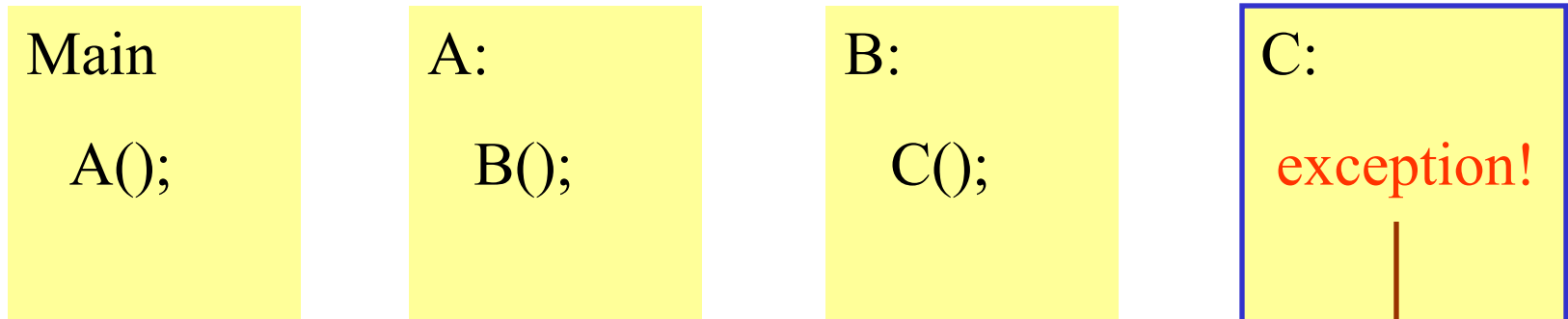
C:

exception!

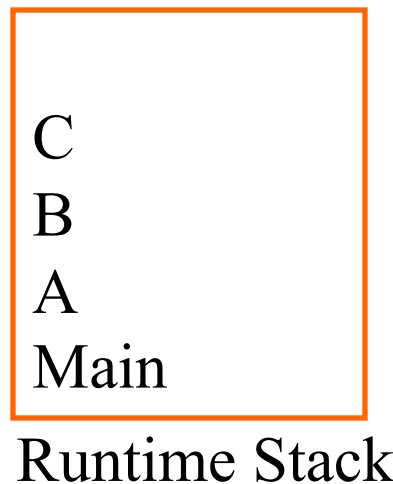
C causes an exception

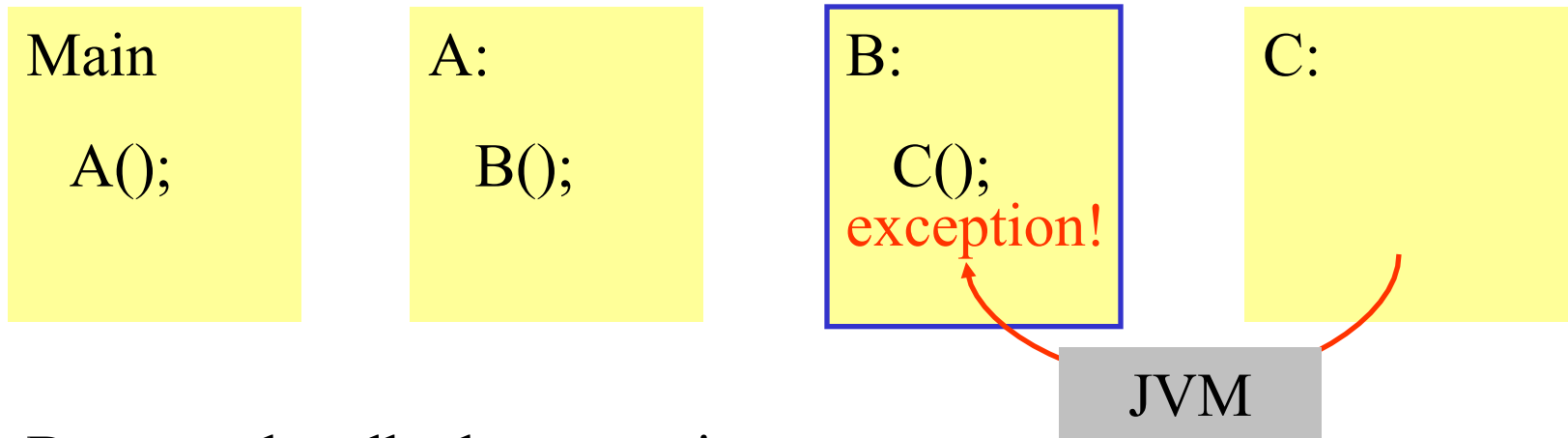
C
B
A
Main

Runtime Stack



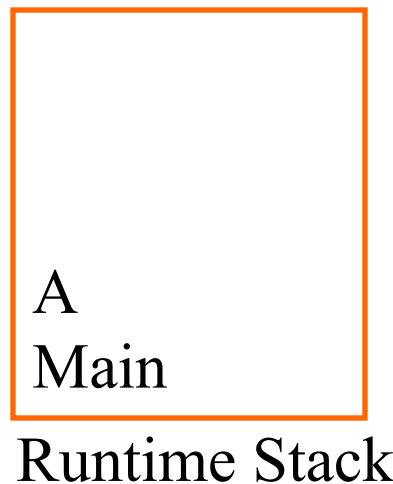
C cannot handle the exception
The JVM terminates C, removes it from the
stack, and throws the exception to B

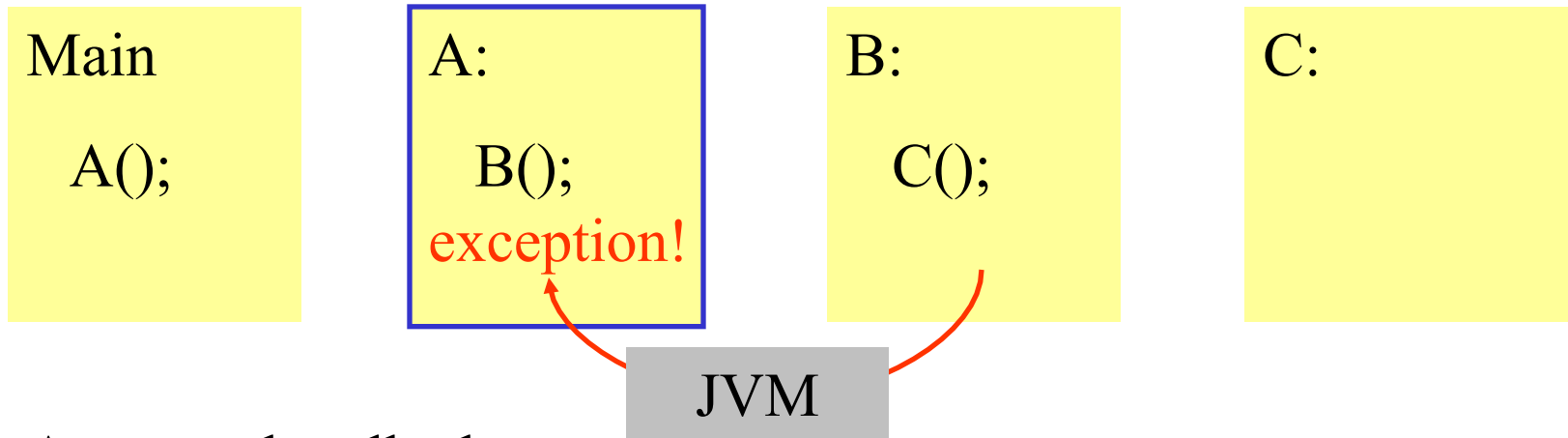




B cannot handle the exception

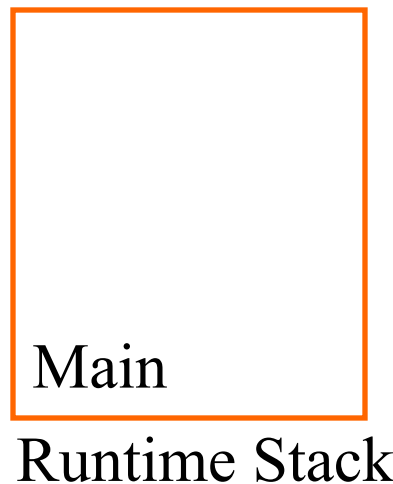
The JVM terminates B, removes it from the stack, and throws the exception to A

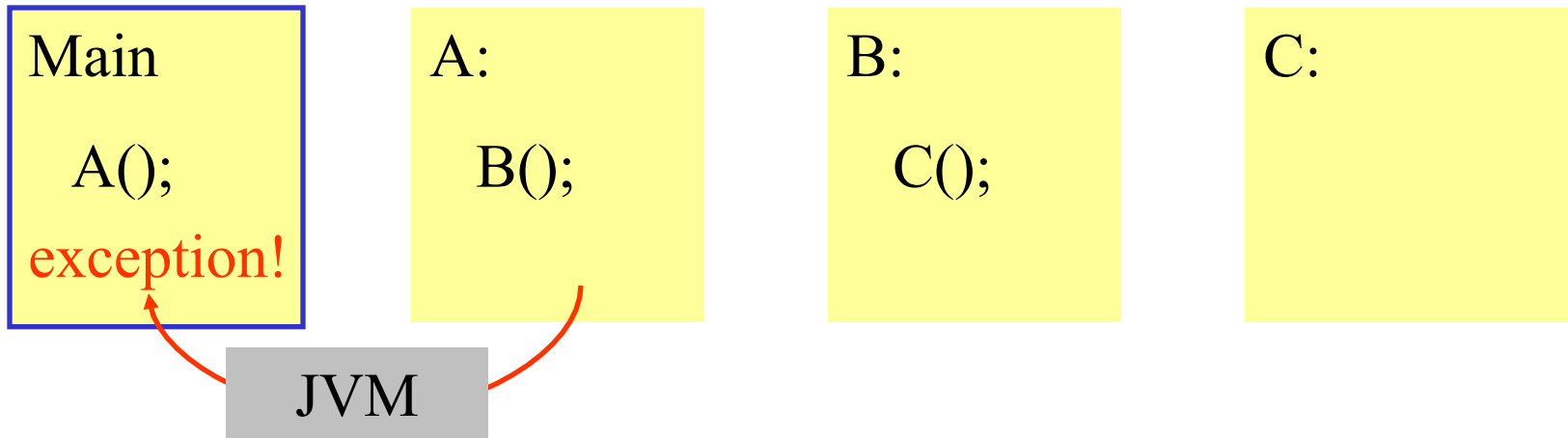




A cannot handle the exception

The JVM terminates A, removes it from the stack, and throws the exception to Main



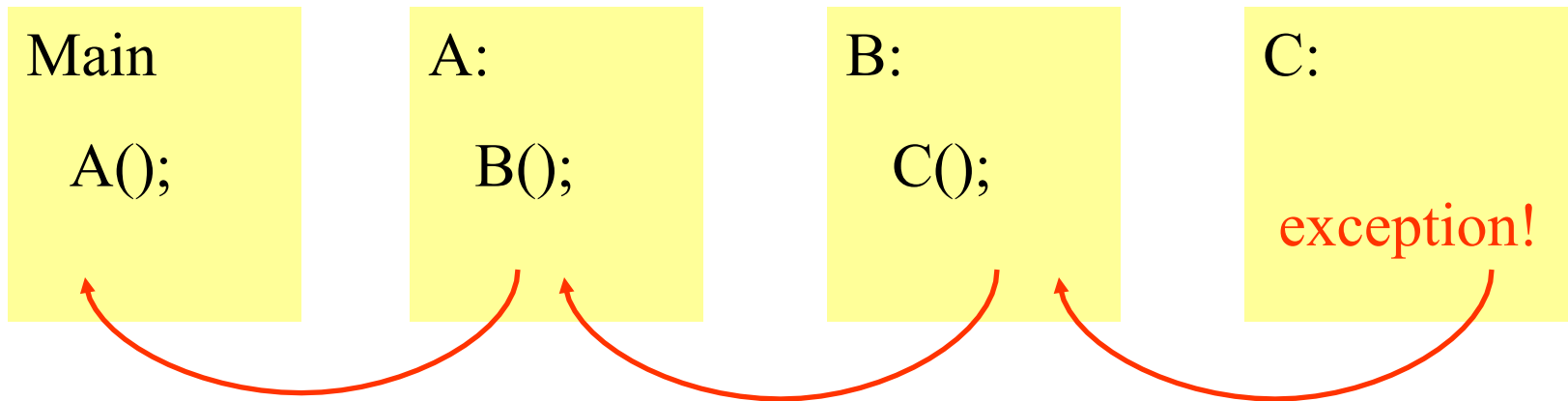


Main cannot handle the exception

The JVM terminates Main, removes it from the stack, and prints a stack trace to the console



Runtime Stack



A, B and C are all *exception propagators* because by not handling the exception, they pass it back to the calling program.

If one of these methods knew how to handle the exception, it would be an *exception catcher*

The try/catch block

```
try {
```

some statements here which may throw an exception

```
catch (Exception e) {
```

some statements to be executed if an Exception happens

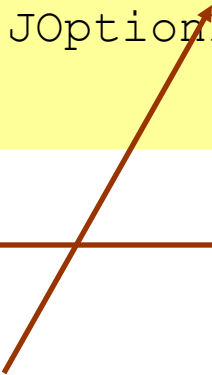
```
}
```

the program then continues...

Example

```
while (true) {  
    try {  
        num = JOptionPane.showInputDialog  
                           (null, "Enter a number");  
        n = Integer.parseInt(num);  
        JOptionPane.showMessageDialog  
               (null, "Thank you for entering "+n);  
        if (n==0) System.exit(0);  
    }  
    catch (Exception e) {  
        JOptionPane.showMessageDialog  
               (null, "That is not a number! Try again");  
    }  
}
```

```
while (true) {  
    try {  
        num = JOptionPane.showInputDialog  
            (null, "Enter a number");  
        n = Integer.parseInt(num);  
        JOptionPane.showMessageDialog  
            (null, "Thank you for entering "+n);  
        if (n==0) System.exit(0);  
    }  
    catch (Exception e) {  
        JOptionPane.showMessageDialog  
            (null, "That is not a number! Try again");  
    }  
}
```



This is very general... any exception that happens will be caught here.

What kind of exception does *parseInt* throw?

parseInt

public static int parseInt(String s) throws NumberFormatException

```
while (true) {
    try {
        num = JOptionPane.showInputDialog
                        (null, "Enter a number");
        n = Integer.parseInt(num);
        JOptionPane.showMessageDialog
                        (null, "Thank you for entering "+n);
        if (n==0) System.exit(0);
    }
    catch (NumberFormatException e) {
        JOptionPane.showMessageDialog
                        (null, "That is not a number! Try again");
    }
}
```

What if multiple exceptions can be thrown in the *try* block?

```
try {  
    n = Integer.parseInt(num);  
    a[i] = n;  
}  
catch (NumberFormatException nfe) {  
    JOptionPane.showMessageDialog(null, "Not a number!" 或者可以写[nfe.getMessage()]);  
}  
catch (IndexOutOfBoundsException ioob) {  
    System.out.println("Bad array index: "+i);  
}  
catch (Exception e) {  
    System.out.println("An exception occurred.");  
}
```

The JVM will go through the catch blocks **top to bottom** until a matching exception is found

What if multiple exceptions can be thrown in the *try* block?

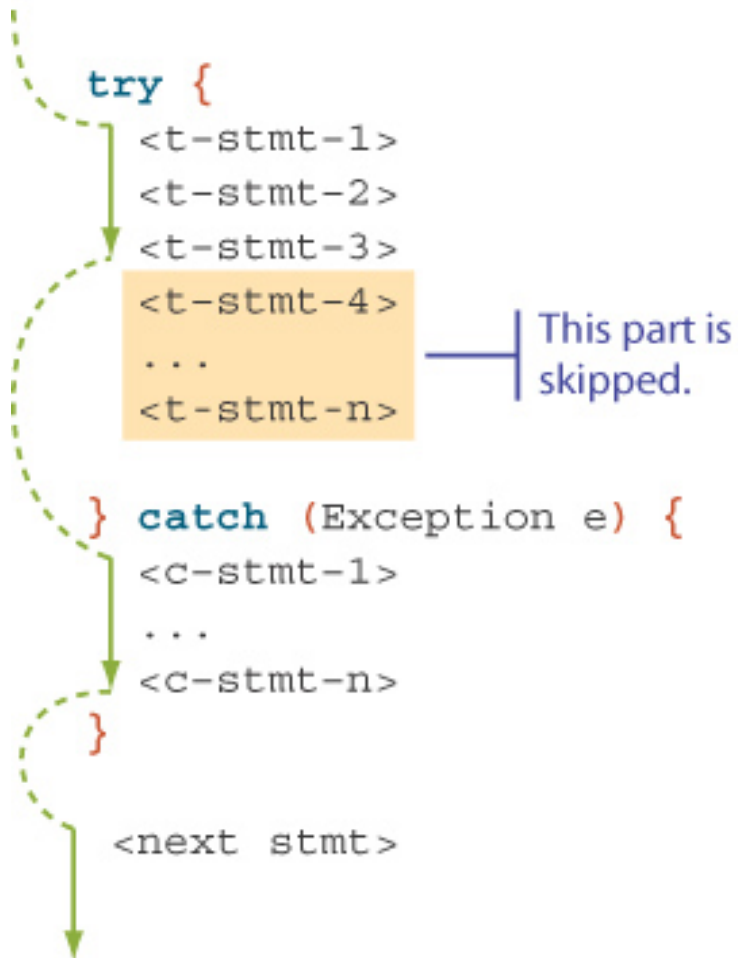
```
try {  
    n = Integer.parseInt(num);  
    a[i] = n;  
}  
catch (Exception e) {  
    System.out.println("An exception occurred.");  
}  
catch (NumberFormatException nfe) {  
    JOptionPane.showMessageDialog(null, "Not a number!");  
}  
catch (IndexOutOfBoundsException ioob) {  
    System.out.println("Bad array index: "+i);  
}
```

This is why the order of the exceptions listed is important.. because of the class hierarchy and inheritance, a *NumberFormatException* **is** an *Exception*, and the first *catch* block will handle it.

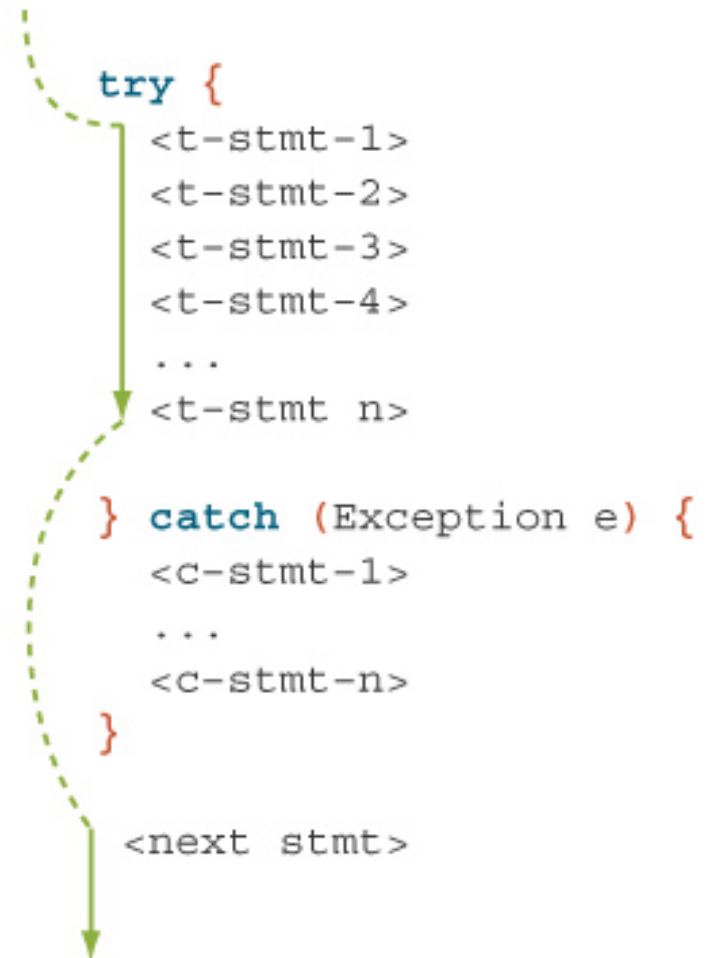
Try/Catch Flow

Exception

Assume **<t-stmt-3>** throws an exception.



No Exception



Example from the SSN class

```
ssnRead = inFile.readLine();
while (ssnRead != null) {
    try {
        mySSN = new SSN(ssnRead);
        mySubscripts.append(Integer.toString(subscript++)+"\n");
        myTextArea.append(mySSN+"\n");
        ssnRead = inFile.readLine();
    }
    catch (IllegalSSNException issne) {
        System.out.println(issne.getMessage());
    }
}
```

There is an error here!

If an exception occurs, the next line from the file is not read.
This needs to be whether or not there is an exception.

The *finally* block is executed whether or not an exception occurs.

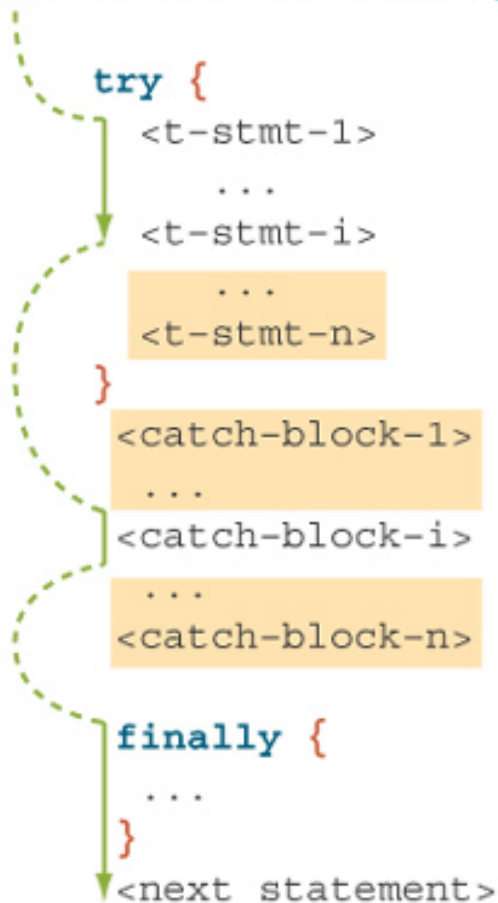
```
ssnRead = inFile.readLine();
while (ssnRead != null) {
    try {
        mySSN = new SSN(ssnRead);
        mySubscripts.append(Integer.toString(subscript++)+"\n");
        myTextArea.append(mySSN+"\n");
    }
    catch (IllegalSSNException issne) {
        System.out.println(issne.getMessage());
    }
    finally {
        ssnRead = inFile.readLine();
    } (finally code will always be done)
}
```

finally executes even if there is a *return* statement prior to the final code.

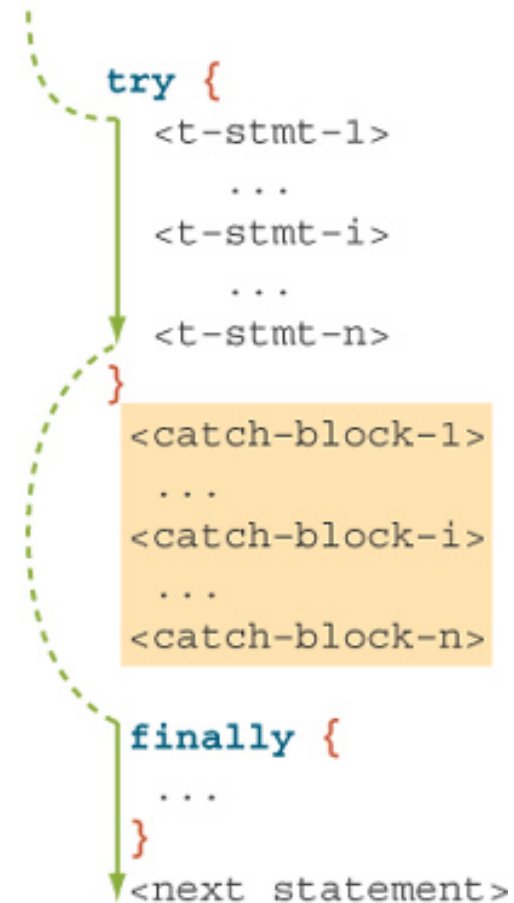
Try/catch/finally flow

Exception

Assume `<t-stmt-i>` throws an exception and `<catch-block-i>` is the matching catch block.



No Exception




Skipped portion

Checked and Unchecked Exceptions

- Excluding exceptions in the class *RuntimeException*, the compiler must find a catcher or a propagator for every exception.
- *RuntimeException* is an *unchecked* exception.
- All other exceptions are *checked* exceptions.


```
class A {  
    public A () throws NumberFormatException {}  
}
```

```
class B {  
    public void C () {  
        A xyz = new A();  
    }  
}
```



This is OK; the
exception will
be propagated.

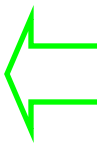
```
class B {  
    try {  
        A xyz = new A();  
    }  
    catch {NumberFormatException nfe {  
    }  
}
```



This is OK; the
exception will
be caught.

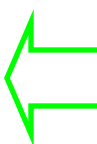
```
class A {  
    public A () throws IOException {}  
}
```

```
class B {  
    public void C () throws IOException  
    {  
        A xyz = new A();  
    }  
}
```



This is OK; the
exception will
be propagated.

```
class B {  
    try {  
        A xyz = new A();  
    }  
    catch {IOException nfe {  
    }  
}
```



This is OK; the
exception will
be caught.