Introduction to Java

CSC212 Lecture 6
D. Thiebaut, Fall 2014

What is it?

- It is always surprising…
- It is common in most programming languages…
- It is a natural way to save computer resources
- Most compilers adopt it…
- It makes for very strange "bugs"…

The Difference between **Shallow** & **Deep** Copies

The Difference between **Shallow** & **Deep** Copies

What the compiler does

What we think it does

Example

```java
class Dummy {
   private int f;
   private int s;
   private int t;

   Dummy( int ff, int ss, int tt ) {
      f = ff; s = ss; t = tt;
   }

   public void setF( int ff ) {
      f = ff;
   }

   public String toString() {
      return String.format( "[%d,%d,%d]", f, s, t );
   }

}
```

```java
public class DeepShallowCopy {

  static public void main( String[] args ) {
    Dummy d1, d2;
    d1 = new Dummy( 1, 2, 3 );
    d2 = d1;

    System.out.println( "d1 = " + d1 );
    System.out.println( "d2 = " + d2 );

    d1.setF( 10 );

    System.out.println( "d1 = " + d1 );
    System.out.println( "d2 = " + d2 );
  }
}
```

```
d1 = [1,2,3]
d2 = [1,2,3]
d1 = [10,2,3]
d2 = [10,2,3]
```

That was an illustration of d2 being a **shallow copy** of d1!

This was also an example of a **side-effect**: The action of changing something somewhere.

Implementing Deep-Copy

```
class Dummy {
   private int f;
   private int s;
   private int t;

   Dummy( int ff, int ss, int tt ) {
      f = ff; s = ss; t = tt;
   }

   public void setF( int ff ) {
      f = ff;
   }

   public String toString() {
      return String.format( "[%d,%d,%d]", f, s, t );
   }

   public Dummy deepCopy( ) {
      return new Dummy( f, s, t );
   }
}
```

```java
public class DeepShallowCopy2 {

  static public void main( String[] args ) {
    Dummy d1, d2;
    d1 = new Dummy( 1, 2, 3 );
    // d2 = d1;
    d2 = d1.deepCopy();

    System.out.println( "d1 = " + d1 );
    System.out.println( "d2 = " + d2 );

    d1.setF( 10 );

    System.out.println( "d1 = " + d1 );
    System.out.println( "d2 = " + d2 );
  }
}
```

```
d1 = [1,2,3]
d2 = [1,2,3]
d1 = [10,2,3]
d2 = [1,2,3]
```

Generic Classes

Example: Creating a Tuple as a **Pair of Ints**

```java
class PairInts {
    private int first;
    private int second;

    public PairInts( int f, int s ) {
        first = f;
        second = s;
    }

    public int getFirst() { return first; }
    public int getSecond() { return second; }
    public void setFirst( int f ) { first = f; }
    public void setSecond( int s ) { second = s; }
    public String toString( ) {
     return String.format( "(%d,%d)", first, second ); }
}
```

Exercise 1

- Create an simple program that

  1. uses the PairInts class

  2. creates an array of 10 pairs, where the first number of the pair in a positive integer (random), and the second number is the fibonacci term equivalent to the first number. For example: (0, 1), (1, 1), (2, 2), (3, 3), (4, 5), (5, 8), (6, 13), etc.

  3. displays the array on the screen.

Many Possible Ways of Pairing Variables

- 2 ints (coordinates on screen)
- 2 doubles (coordinates of points in plane)
- 2 strings (first name, last name)
- 1 float, 1 string (grades: {3.7, "A"})
- etc…

Do we need **One Pair for Each Possible Combination** Of Types?

NEW
NOTATION!

Generic Classes

```java
public class Pair<T, P> {
    private T first;
    private P second;

    public Pair( T f, P s ) {
        first = f;
        second = s;
    }

    public T getFirst() { return first; }
    public P getSecond() { return second; }
    public void setFirst( T f ) { first = f; }
    public void setSecond( P s ) { second = s; }
    public String toString( ) {
        return "(" + first + ", " + second + ")"; }
}
```

```
class PairInts {
    private int first;
    private int second;

    public PairInts( int f, int s ) {
        first = f;
        second = s;
    }

    public int getFirst() { return first; }
    public int getSecond() { return second; }
    public void setFirst( int f ) { first = f; }
    public void setSecond( int s ) { second = s; }
    public String toString( ) {
        return String.format( "(%d,%d)", first, second ); }
}
```

```
public class Pair<T, P> {
    private T first;
    private P second;

    public Pair( T f, P s ) {
        first = f;
        second = s;
    }

    public T getFirst() { return first; }
    public P getSecond() { return second; }
    public void setFirst( T f ) { first = f; }
    public void setSecond( P s ) { second = s; }
    public String toString( ) {
        return "(" + first + ", " + second + ")"; }
}
```

Testing 1

```java
class TestPair {
    static public void main( String[] args ) {
        Pair<Integer, Integer> p1 = null, p2 = null;

        p1 = new Pair<Integer, Integer>( 1, 3 );
        p2 = new Pair<Integer, Integer>( 3000, -9 );

        System.out.println( "p1 = " + p1 );
        System.out.println( "p2 = " + p2 );
        p1.setFirst( p2.getFirst() );
        System.out.println( "p1 = " + p1 );
    }
}
```

```
p1 = (1, 3)
p2 = (3000, -9)
p1 = (3000, 3)
```

Testing 2

```java
class TestPair {
    static public void main( String[] args ) {
        Pair<String, Integer> p1 = null, p2 = null;

        p1 = new Pair<String, Integer>( "Sophia Smith", 3 );
        p2 = new Pair<String, Integer>( "Mickey Mouse", -9 );

        System.out.println( "p1 = " + p1 );
        System.out.println( "p2 = " + p2 );
        p1.setFirst( p2.getFirst() );
        System.out.println( "p1 = " + p1 );
    }
}
```

```
p1 = (Sophia Smith, 3)
p2 = (Mickey Mouse, -9)
p1 = (Mickey Mouse, 3)
```

Exercise 2

- Same as Exercise 1, but this time the program

  1. uses the generic Pair Class,

  2. creates an array of 10 pairs, where the first number of the pair in a positive integer (random), and the second number is the fibonacci term equivalent to the first number. For example: (0, 1), (1, 1), (2, 2), (3, 3), (4, 5), (5, 8), (6, 13), etc.

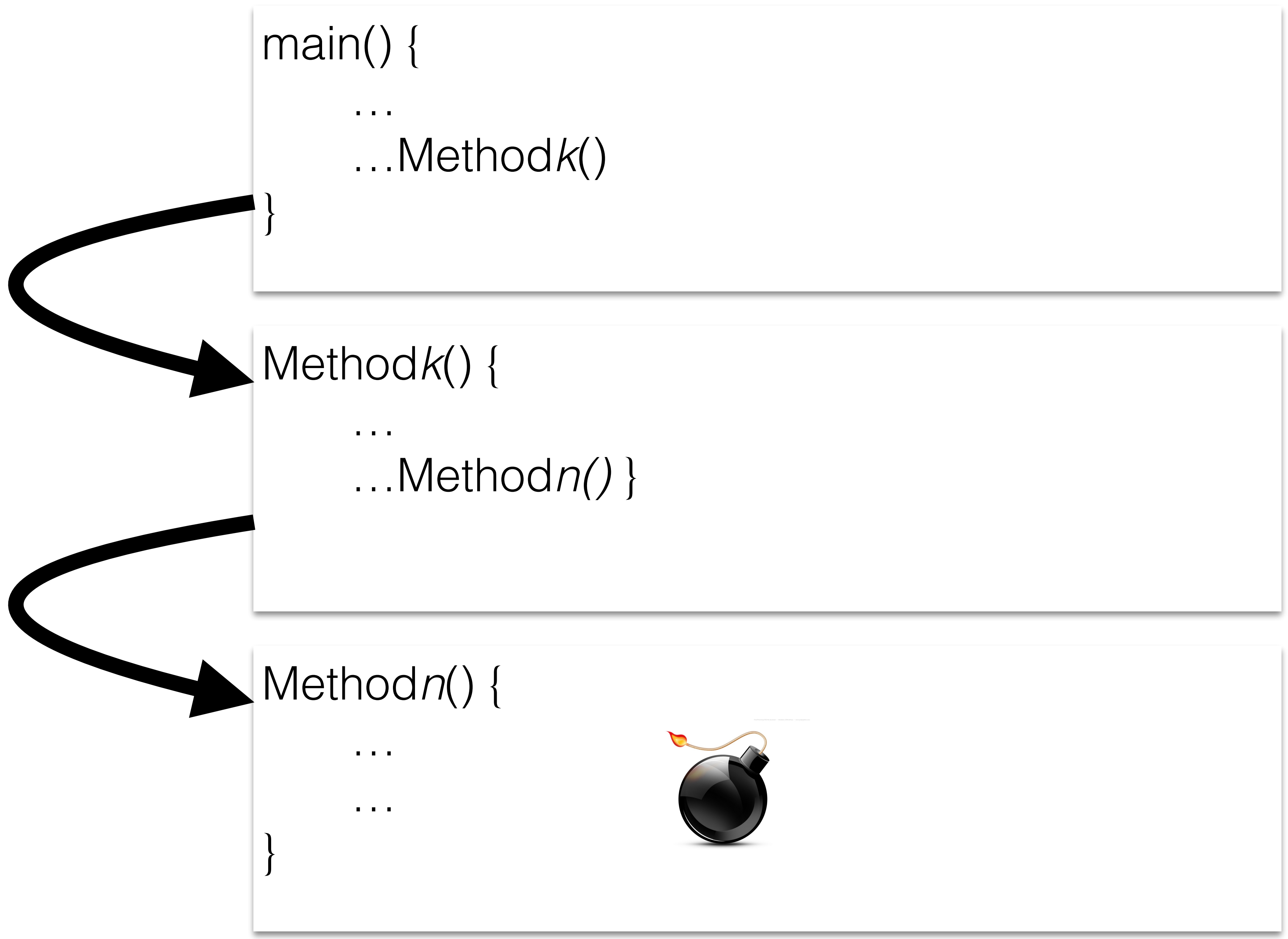
  3. displays the array on the screen.

Exceptions

http://docs.oracle.com/javase/tutorial/essential/exceptions/definition.html

Definition

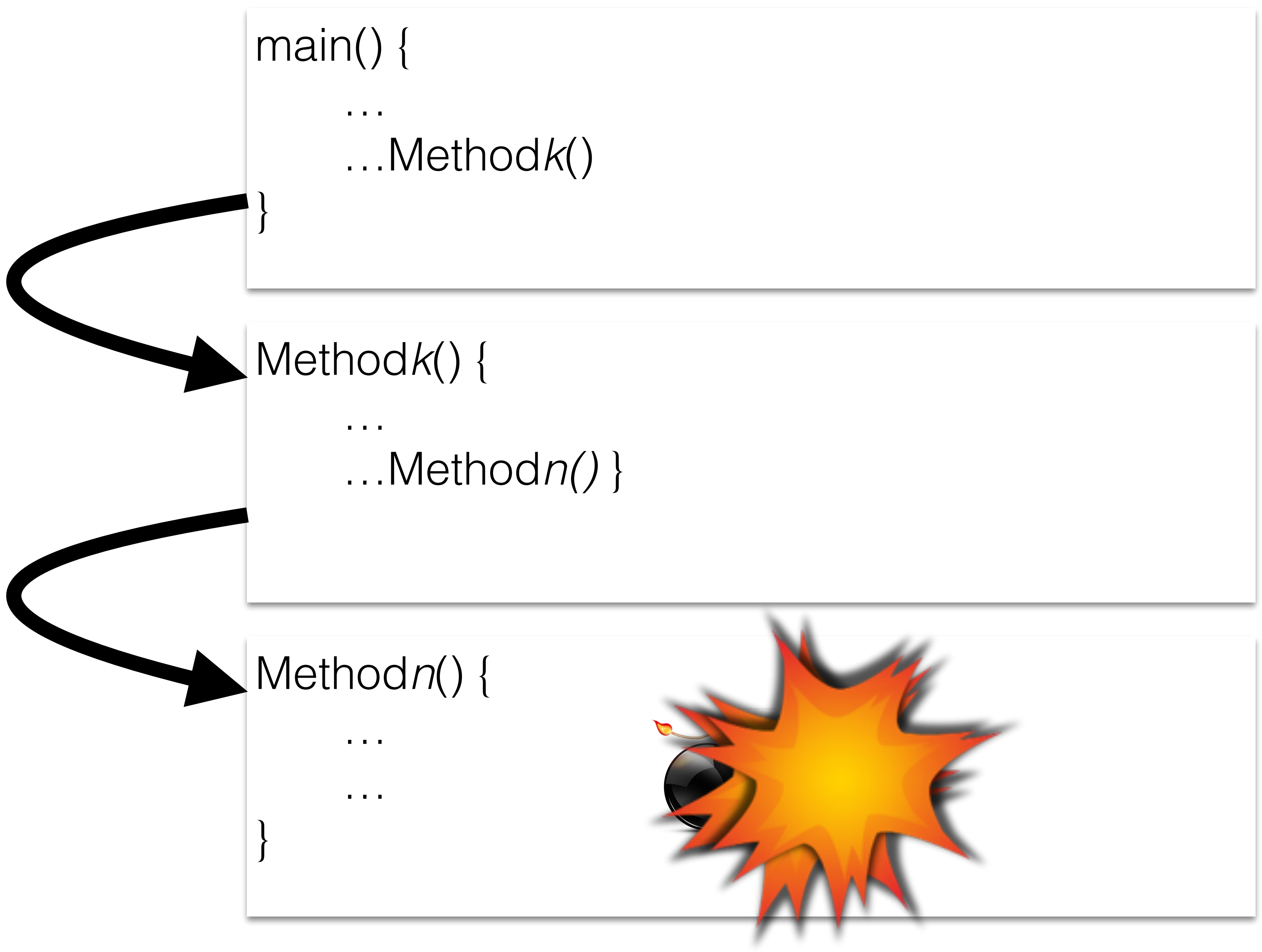
- An exception is an **event**, which occurs **during the execution** of a program, that **disrupts** the normal flow of the program's instructions.



```
main() {
        …
        …Methodk()
}
```

```
Methodk() {
        …
        …Methodn() }
```

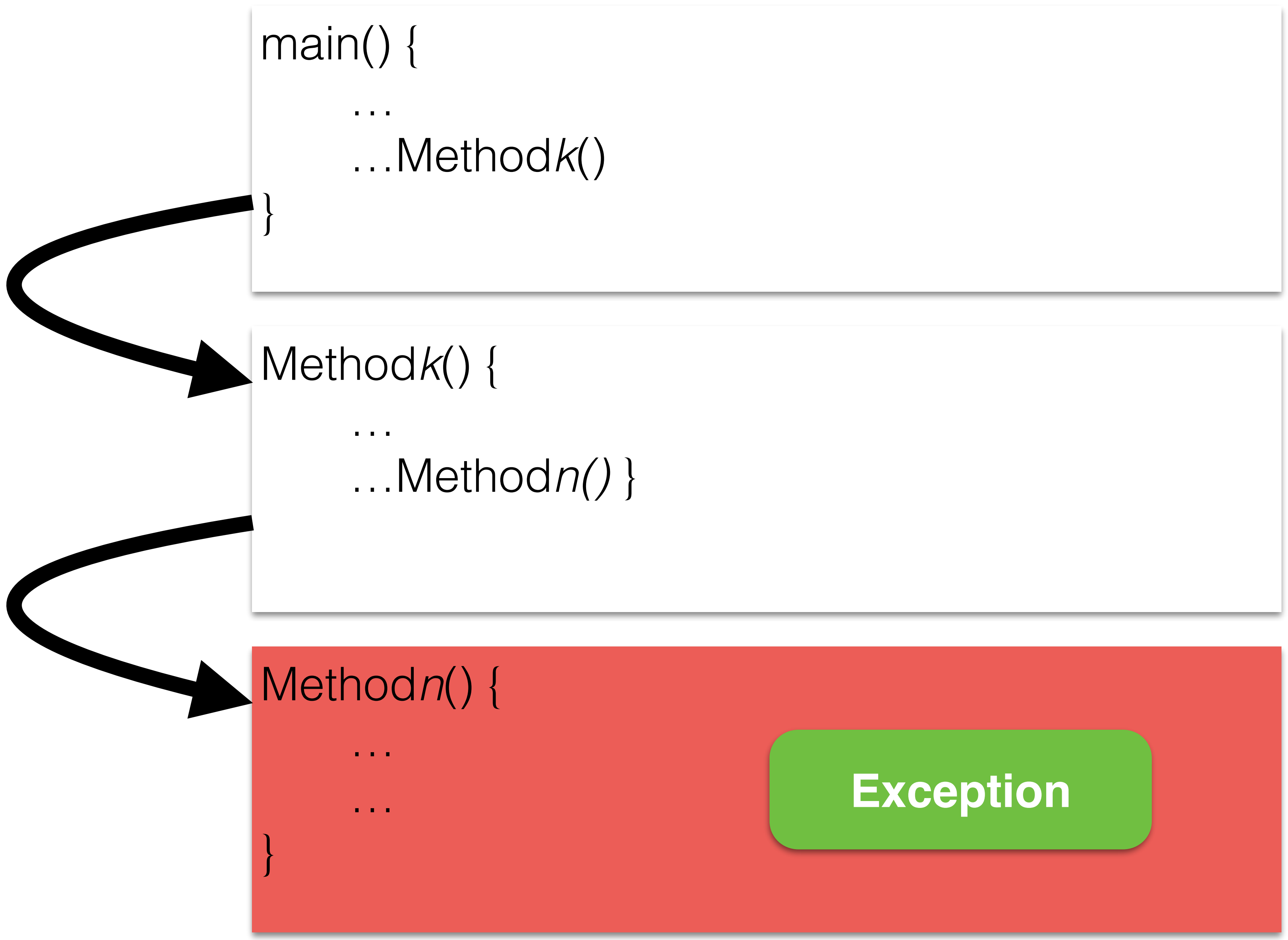
```
Methodn() {
        …
        …
}
```



```
main() {
        …
        …Methodk()
}
```

```
Methodk() {
        …
        …Methodn() }
```

```
Methodn() {
        …
        …
}
```

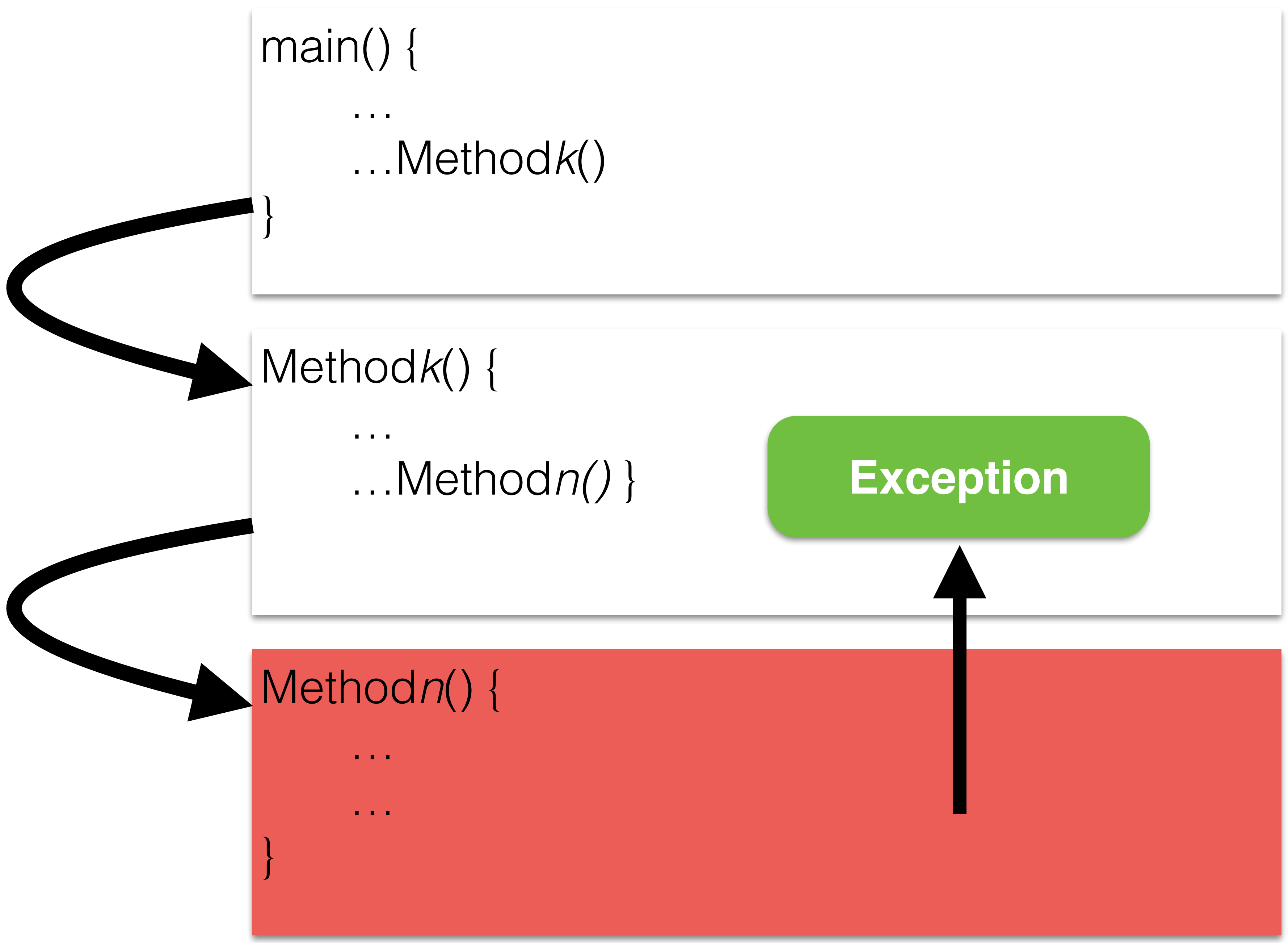


```
main() {
        …
        …Methodk()
}
```

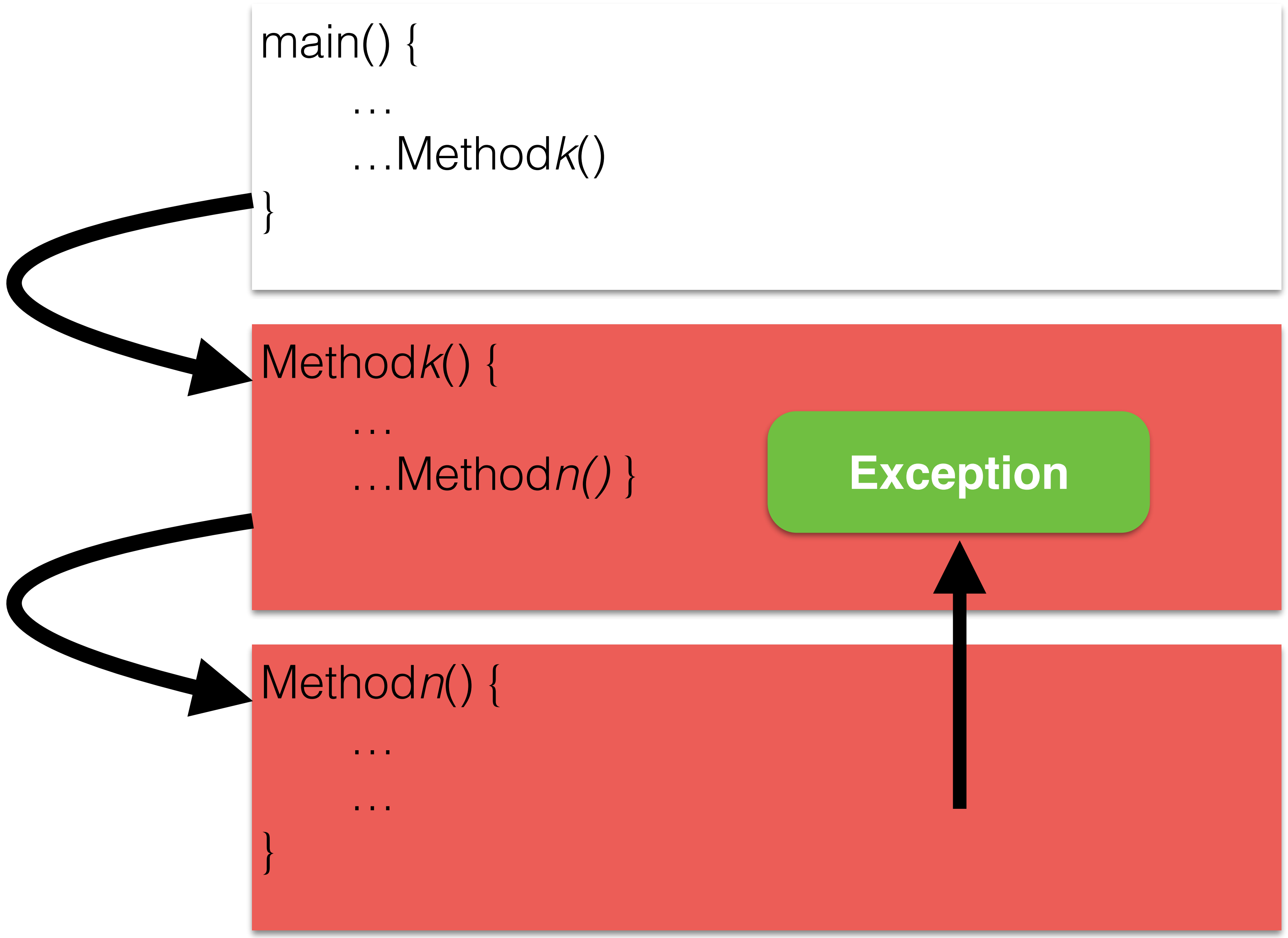
```
Methodk() {
        …
        …Methodn() }
```

```
Methodn() {
        …
        …
}
```

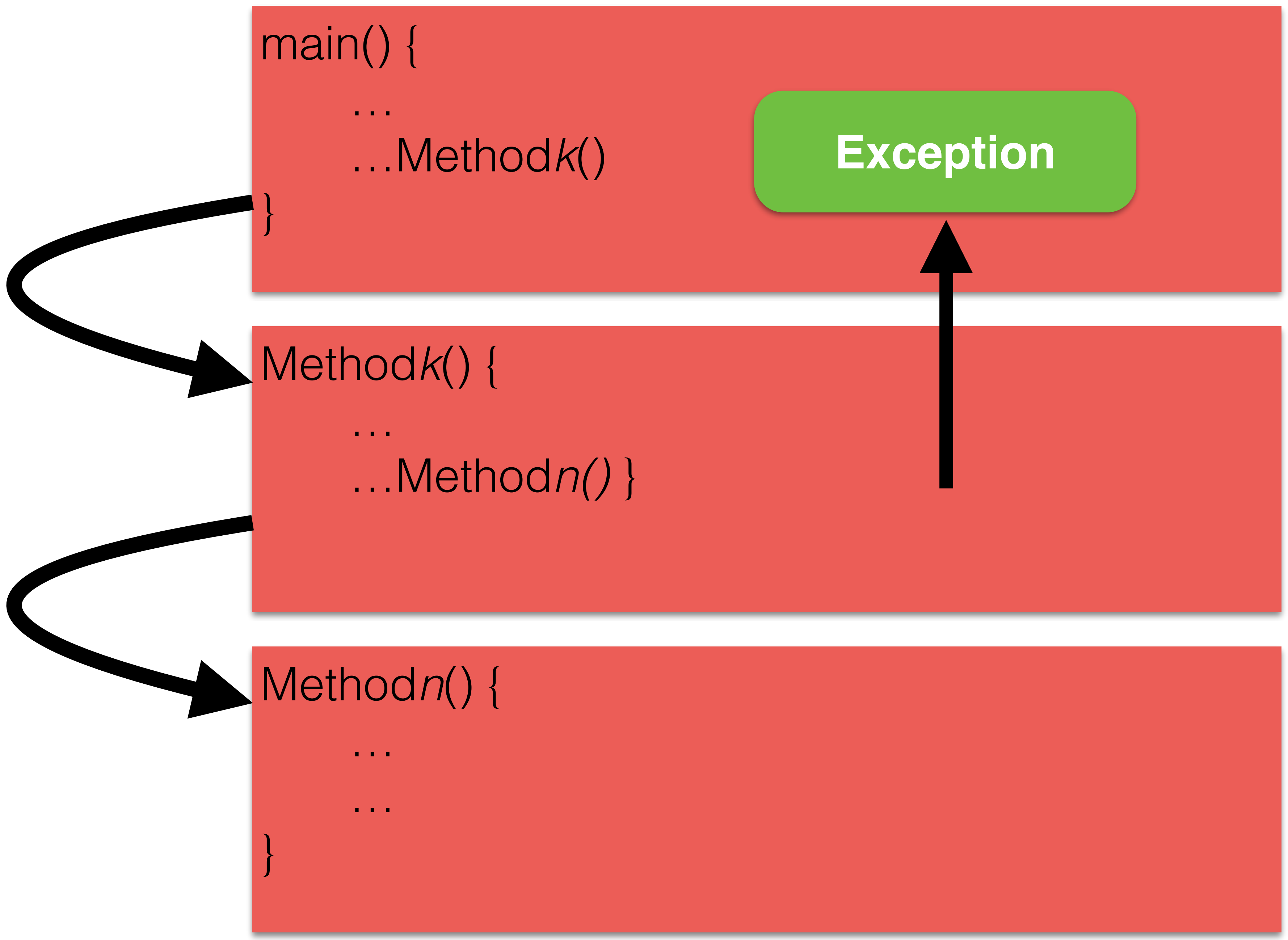
Method*n*() creates an Exception object…



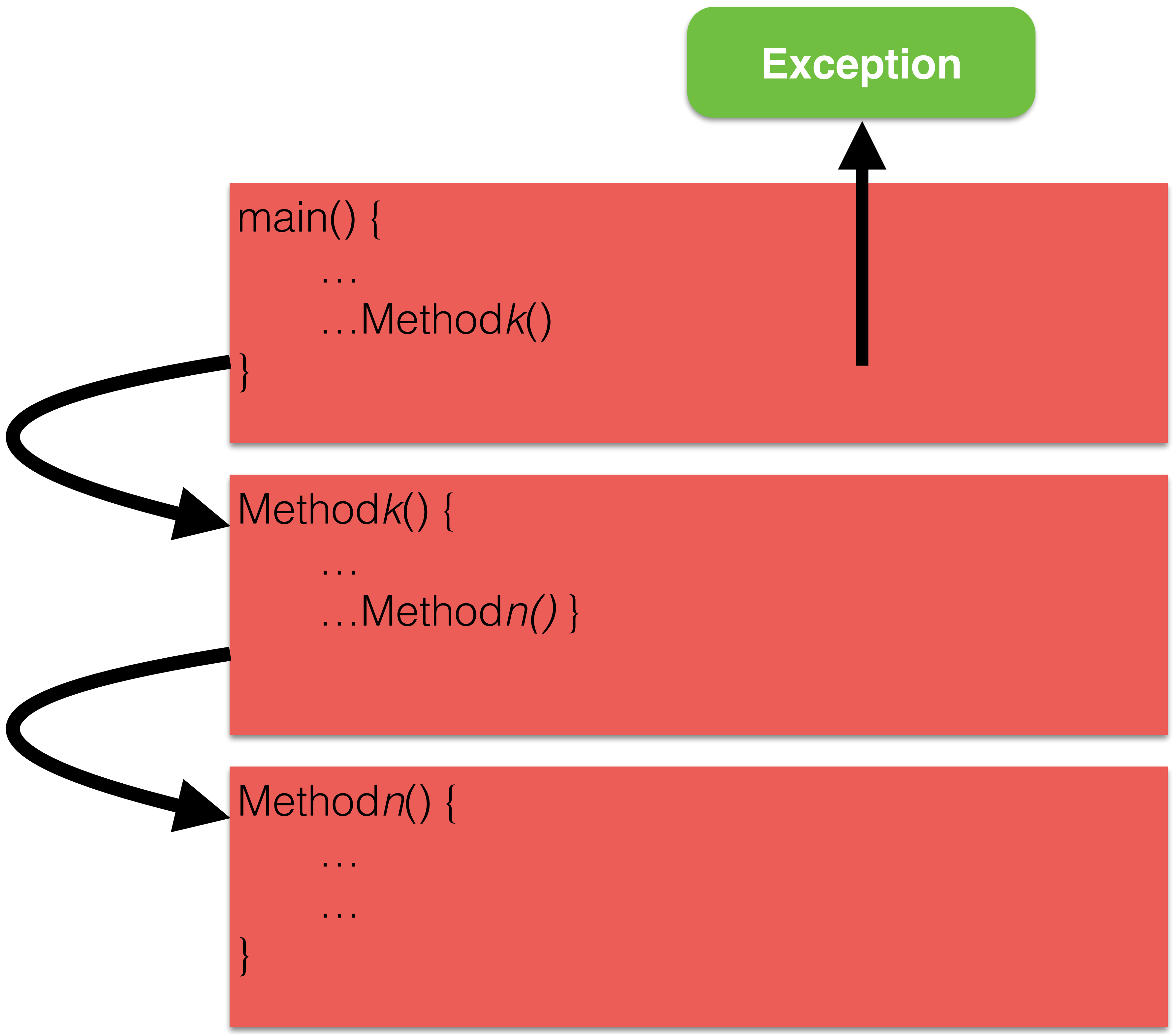
If Method*n*() doesn't have code to handle the Exception object, passes it on to its caller…



if Method*k*() doesn't have code to handle the object, passes it on to its caller…



main() gets the Exception object…



if main() doesn't have code to handle the Exception, it passes it on to its caller…

```
[beowulf2]
[11:40:02] ~/public_html/classes/212$: java Bomb
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 10
    at Bomb.methodn(Bomb.java:4)
    at Bomb.methodk(Bomb.java:8)
    at Bomb.main(Bomb.java:14)
```

Exception

```
class Bomb {

   public static void methodn( int[] A ) {
            A[10] = 0;
   }

   public static void methodk(int[] A) {
      methodn(A);
   }

   public static void main(String[] args) {

      int[] A = new int[10];

      methodk(A);
   }
}
```

```
class Bomb2 {

   public static void methodn( int[] A ) {
       try {
           A[10] = 0;
       }
       catch (ArrayIndexOutOfBoundsException e ) {
         System.err.println("ArrayIndexOutOfBoundsException: "
                                    + e.getMessage());
     }
   }

   public static void methodk(int[] A) {
      methodn(A);
   }

   public static void main(String[] args) {

      int[] A = new int[10];

      methodk(A);
   }
}
```

Syntax

```
try {
     …  // code that might create exception
}
catch ( SomeExection e ) {
    …  // what we should do for this type of exception…
}
catch ( SomeOtherException e ) {
    …  // what we should do for this other type of exception…
}
finally {
    … what to do regardless of exceptions or lack thereof…
}
```

Exercise 3

- Remember the Python list implemented with an array of strings?

  Revisit the *append(String s)* method, and implement it using a **try/catch** statement.