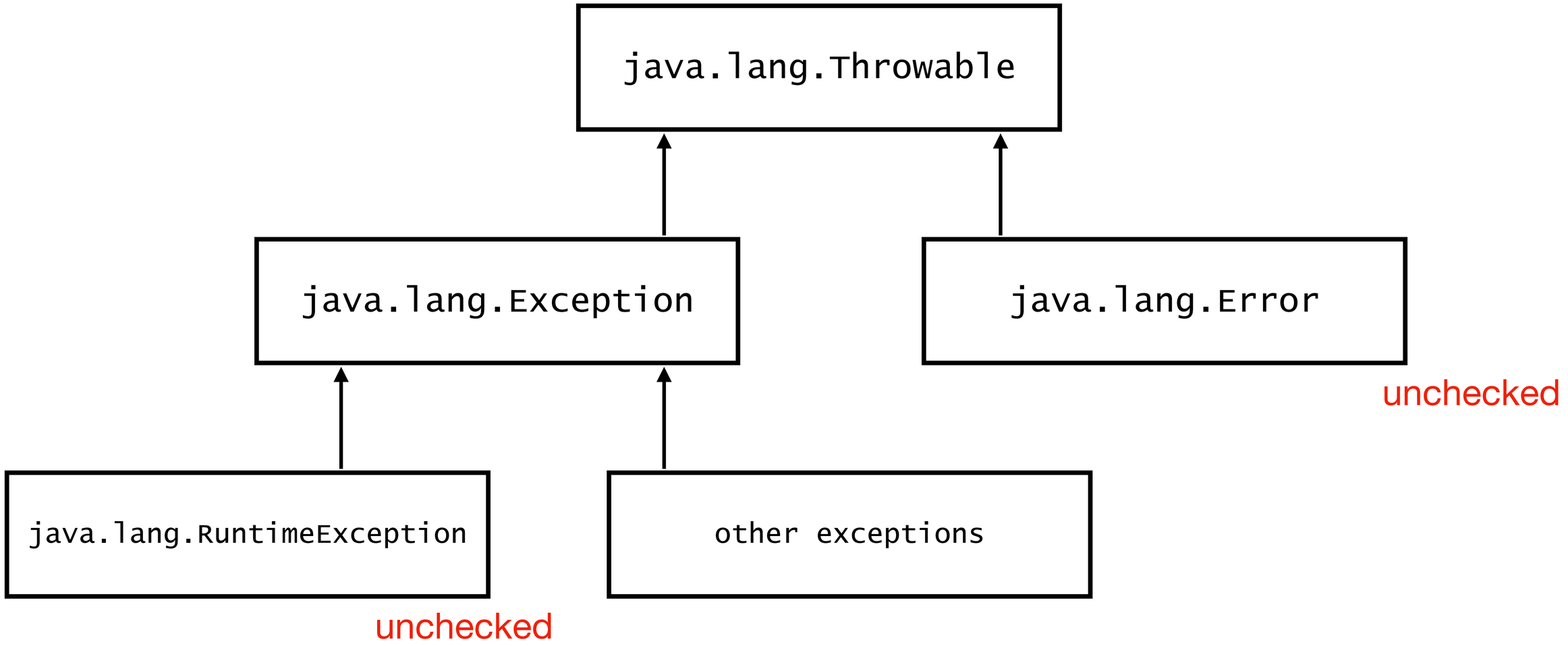
Exceptions

Handling Exceptions

What Are Exceptions?

- exception occurs when something goes wrong during the runtime, e.g.
  - you are trying to divide by zero
  - you are trying to fetch the index of the array which does not exist
  - you are trying to use `null` as a reference
  - you are passing a value to a method which is not internally allowed
  - you are trying to reach the file which does not exist
  - you are trying to connect to a database which is not accessible
  - etc. etc. etc.

Exception Types



Checked Exception

- must be declared or handled by the application code where it is thrown
- all checked exceptions inherit `Exception`, but not `RuntimeException`
- exception is declared when defining a method
  - using keyword `throws`
- exception is handled using `try-catch` block

```
// example
void readFirstByteFromFile (File fileName) {
    FileInputStream file = new FileInputStream(fileName);
    byte x = (byte) file.read();
    System.out.println(x);
}
```

this **does not compile** because there is a possibility that checked exception `IOException` to be thrown by this method

```
// fix #1 : declare the exception
void readFirstByteFromFile (File fileName) throws IOException {
    FileInputStream file = new FileInputStream(fileName);
    byte x = (byte) file.read();
    System.out.println(x);
}
```

this compiles, but still we didn't handle the exception

```
// fix #2 : handle the exception
void readFirstByteFromFile (File fileName) {
  try {
     FileInputStream file = new FileInputStream(fileName);
     byte x = (byte) file.read();
     System.out.println(x);
  } catch (IOException e) {
     e.printStackTrace();
  }
}
```

this compiles, and this is a good practice

```
// unchecked exceptions may or may not be handled
public class MyClass {
  private static void printFourthElement(int[] a) {
    System.out.println(a[3]);
  }
 public static void main(String args[]) {
    int[] a = {-7, 11, 3};
    printFourthElement(a);
   }
}
```

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException:
Index 4 out of bounds for length 3
        at MyClass.printFourthElement(MyClass.java:11)
        at MyClass.main(MyClass.java:6)
```

```
// same example, but with exception handled
public class MyClass {
  private static void printFourthElement(int[] a) {
    try {
      System.out.println(a[3]);
    } catch (ArrayIndexOutOfBoundsException e) {
      System.out.println("There is no fourth element in the array. Sorry.");
    }
  }
 public static void main(String args[]) {
    int[] a = {-7, 11, 3};
    printFourthElement(a);
   }
}
```

```
There is no fourth element in the array. Sorry.
```

```
// you can manually throw an exception
public class Student {
  private int id;
  private String fullName;
  public Student (int id, String fullName) {
    if (id < 10 || fullName.length() > 100)
      throw new IllegalArgumentException();
    this.id = id;
    this.fullName = fullName;
  }
}
// in main method..
Student student = new Student(1, "John Wayne");
```

the IllegalArgumentException will be thrown

```
// main method must also declare or handle the checked exception
class OutOfMilkException extends Exception {}   custom checked exception
public class MyClass {
   private static void getMilk() throws OutOfMilkException {
      System.out.println("Getting the milk...");
   }
   public static void main(String args[]) {   // DOES NOT COMPILE
     getMilk();
   }
}

// fix: declare or handle OutOfMilkException
public static void main(String args[]) throws OutOfMilkException { ... }    OK
```

Unchecked Exceptions

`ArithmeticException`	Thrown when code tries to divide with zero
`ArrayIndexOutOfBoundsException`	Thrown when code uses illegal index to access array element
`ClassCastException`	Thrown when code tries to cast object to a class of which it is not an instance
`NullPointerException`	Thrown when there is a `null` reference where object is required
`IllegalArgumentException`	Thrown by a programmer to indicate that the illegal or inappropriate argument has been passed to a method
`NumberFormatException`	Subclass of `IllegalArgumentException`, thrown then the code tries to convert String to a numeric type, but String doesn't have an appropriate format

```
// catching multiple exceptions
// example #1: one exception is a subclass of another
void readFirstByteFromFile (File fileName) {
  try {
     FileInputStream file = new FileInputStream(fileName);
     byte x = (byte) file.read();
     System.out.println(x);
  } catch (FileNotFoundException e) {
     e.printStackTrace();
  } catch (IOException e) {
     e.printStackTrace();
  }
}
```

#1 subclass

#2 superclass

if you change the order, the code will not compile!!

```
// catching multiple exceptions
// example #2: one exception is independent of another
void readFirstByteFromFile (File fileName) {
  try {
     FileInputStream file = new FileInputStream(fileName);
     byte x = (byte) file.read();
     System.out.println(x);
  } catch (IOException e) {
     e.printStackTrace();
  } catch (NumberFormatException e) {
     e.printStackTrace();
  }
}
```

these exceptions are independent of each other => OK

```
// catching multiple exceptions
// example #2A: one exception is independent of another
void readFirstByteFromFile (File fileName) {
  try {
     FileInputStream file = new FileInputStream(fileName);
     byte x = (byte) file.read();
     System.out.println(x);
  } catch (IOException | NumberFormatException e) {
     e.printStackTrace();
  }
}
```

this approach works only for independent classes !!

```
// examples of incorrect syntax:
catch (IOException e1 | NumberFormatException e2)
catch (IOException e | NumberFormatException e)
```

```
// finally block
void readFirstByteFromFile (File fileName) {
  try {
     FileInputStream file = new FileInputStream(fileName);
     byte x = (byte) file.read();
     System.out.println(x);
  } catch (IOException | NumberFormatException e) {
     e.printStackTrace();
  } finally {
     System.out.println("Running readFirstByteFromFile...");
  }
}
```

executed whether exception is caught or not