Null References

Primitives cannot be null and the compiler will reject such attempts with static errors.

null is not the same as an empty string or an empty array:
○ The length of an empty array or an empty string is 0
○ The length of a string variable that points to null isn't anything: calling length() throws a NullPointerException

Exceptions

A method's signature its name, parameter types, return type is a core part of its specification, and the signature may also include exceptions that the method may trigger.

In-Class Quiz 1

- We use `BirthdayBook` with the `lookup` method that `throws NotFoundException`
- Assume we have initialized the `birthdays` variable to point to a `BirthdayBook`, and that `"Makima"` is **not** in that birthday book
- What will happen with the following code:

```
try {
    LocalDate birthdate = birthdays.lookup("Makima");
}
System.out.println("done");
```

  - static error caused by incorrect syntax
  - static error caused by undeclared variable
  - dynamic error caused by `NotFoundException`
  - no errors and it prints `"done"`

In-Class Quiz 2

- We use `BirthdayBook` with the `lookup` method that `throws NotFoundException`
- Assume we have initialized the `birthdays` variable to point to a `BirthdayBook`, and that `"Makima"` is **not** in that birthday book
- What will happen with the following code:

```
try {
    LocalDate birthdate = birthdays.lookup("Makima");
} catch (NotFoundException nfe) {
    birthdate = LocalDate.now();
}
System.out.println("done");
```

  - static error caused by incorrect syntax
  - static error caused by undeclared variable
  - dynamic error caused by `NotFoundException`
  - no errors and it prints `"done"`

In-Class Quiz 3

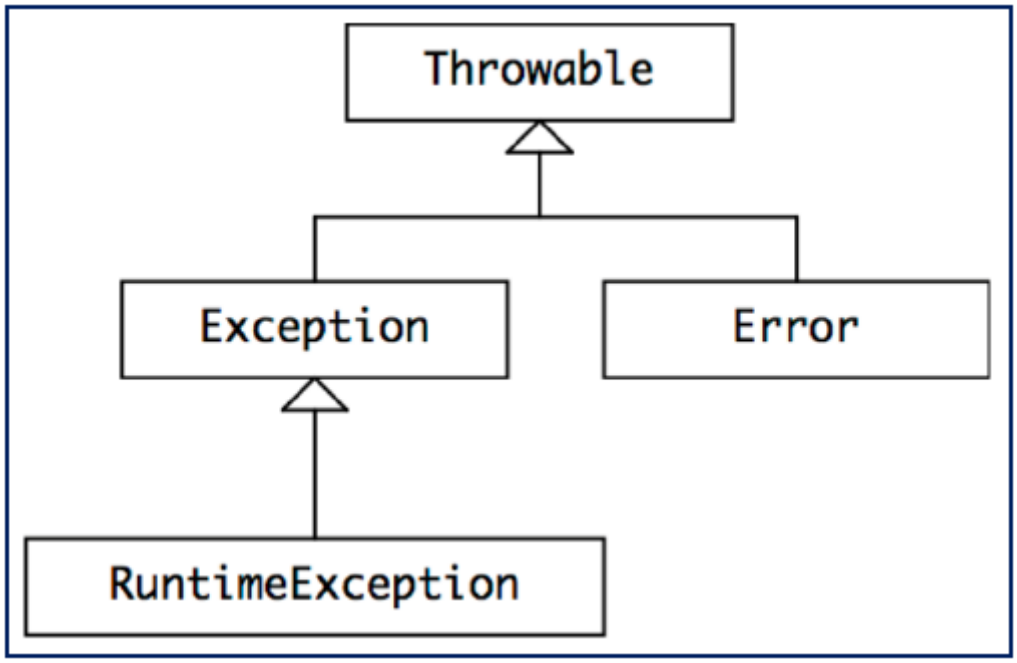
- We use `BirthdayBook` with the `lookup` method that `throws NotFoundException`
- Assume we have initialized the `birthdays` variable to point to a `BirthdayBook`, and that `"Makima"` is **not** in that birthday book
- What will happen with the following code:

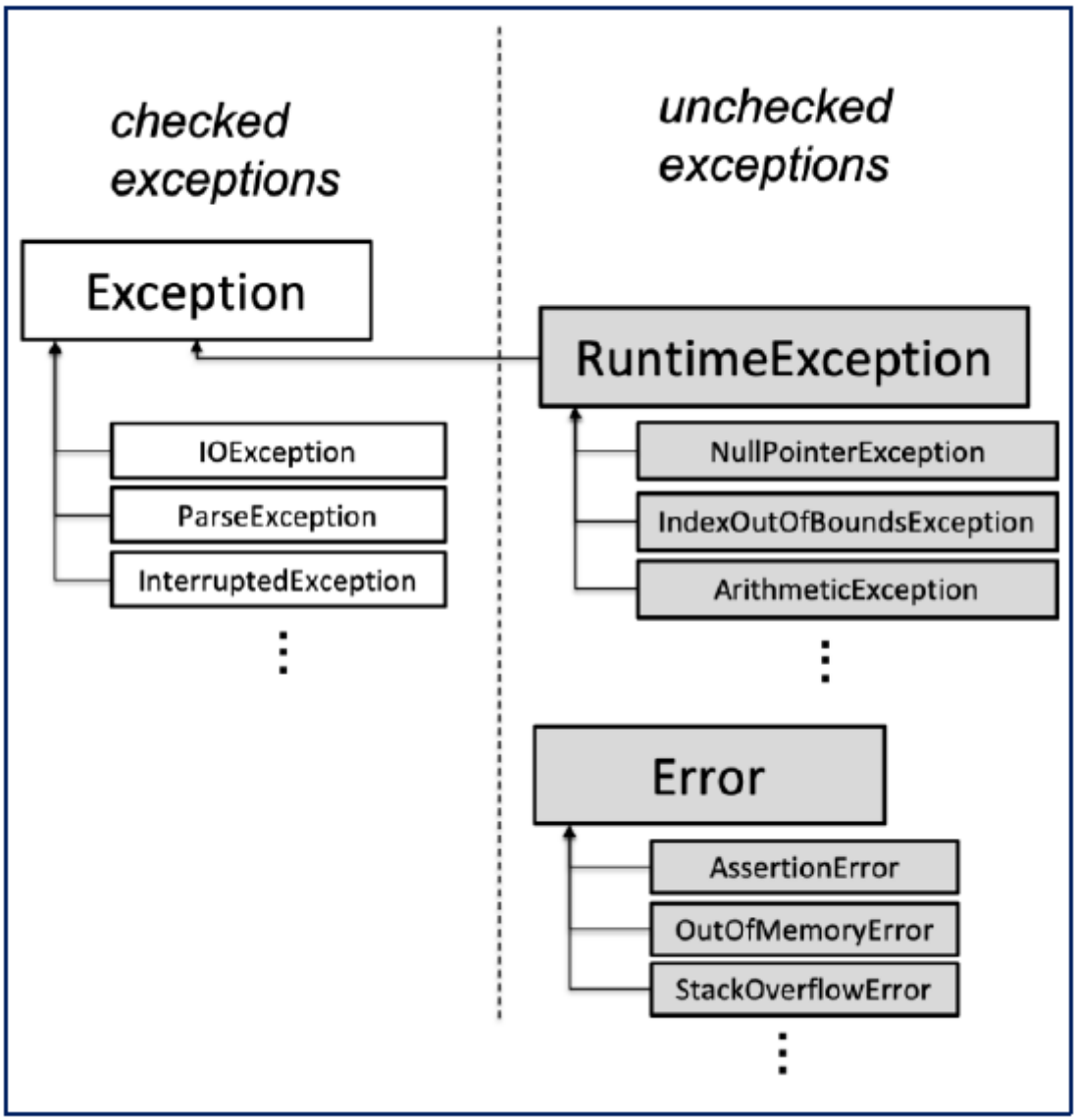
```
try {
    LocalDate birthdate = birthdays.lookup("Makima");
} catch (Exception NotFoundException) {
}
System.out.println("done");
```

  - static error caused by incorrect syntax
  - static error caused by undeclared variable
  - dynamic error caused by `NotFoundException`
  - no errors and it prints `"done"`

Checked and Unchecked Exceptions

use ***checked exceptions*** to signal ***special results*** and ***unchecked exceptions*** to signal ***bugs***.





Checked exceptions are called that because they are checked by the compiler.

Unchecked exception the compiler will not check for try catch or a throws declaration.

The compiler applies static checking to methods using these exceptions
○ A checked exception must either be caught or declared when it's possible for it to be thrown

如果出现 unchecked exception，都是程序员的问题。

```java
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;

public class Test9 {
    public static void main(String[] args) {
        try {
            readFile("joke.txt");
        } catch (FileNotFoundException e) {
            System.out.println("所需文件不存在！");
        } catch (IOException e) {
            System.out.println("文件读写错误！");
        }
    }
    public static void readFile(String fileName) throws FileNotFoundException,
    IOException {
        FileReader in = new FileReader(fileName);
        int tem = 0;
        try {
            tem = in.read();
            while (tem != -1) {
                System.out.print((char) tem);
                tem = in.read();
            }
        } finally {
            in.close();
        }
    }
}
```

【示例6-10】自定义异常类

```java
/**IllegalAgeException：非法年龄异常，继承Exception类*/
class IllegalAgeException extends Exception {
    //默认构造器
    public IllegalAgeException() {

    }
    //带有详细信息的构造器，信息存储在message中
    public IllegalAgeException(String message) {
        super(message);
    }
}
```

【示例6-11】自定义异常类的使用

```java
class Person {
    private String name;
    private int age;

    public void setName(String name) {
        this.name = name;
    }

    public void setAge(int age) throws IllegalAgeException {
        if (age < 0) {
            throw new IllegalAgeException("人的年龄不应该为负数");
        }
        this.age = age;
    }

    public String toString() {
        return "name is " + name + " and age is " + age;
    }
}

public class TestMyException {
    public static void main(String[] args) {
        Person p = new Person();
        try {
            p.setName("Lincoln");
            p.setAge(-1);
        } catch (IllegalAgeException e) {
            e.printStackTrace();
            System.exit(-1);
        }
        System.out.println(p);
    }
}
```

shallow copy

shallow copy (or aliasing):

```java
// Shallow Copy
// NOT what we want !
public SLList(SLList<T> other) {
    sentinel = other.sentinel;
    size = other.size;
}
```

you simply set the sentinel to point to other's sentinel and copy the size

deep copy

○ The input and the copy output should be different objects
○ If you change other, the new SLList you created should not change as well