M4 (b) – Design for Robustness

Jin L.C. Guo

Logistics

Lab Test 2 (Feb 13-24)

- Sign up now
- Focus: Types and Polymorphism and Object State, anything before is also in scope

-Bring your laptop with IDE set up locally (online IDE not allowed).

-If you do not have your own system, please go in advance to TR3120 and identify a system in which IntelliJ works for your user.

-We might ask you to import base code into your IDE and draw diagram during the lab test. Practice in advance and plan your time during the lab test.

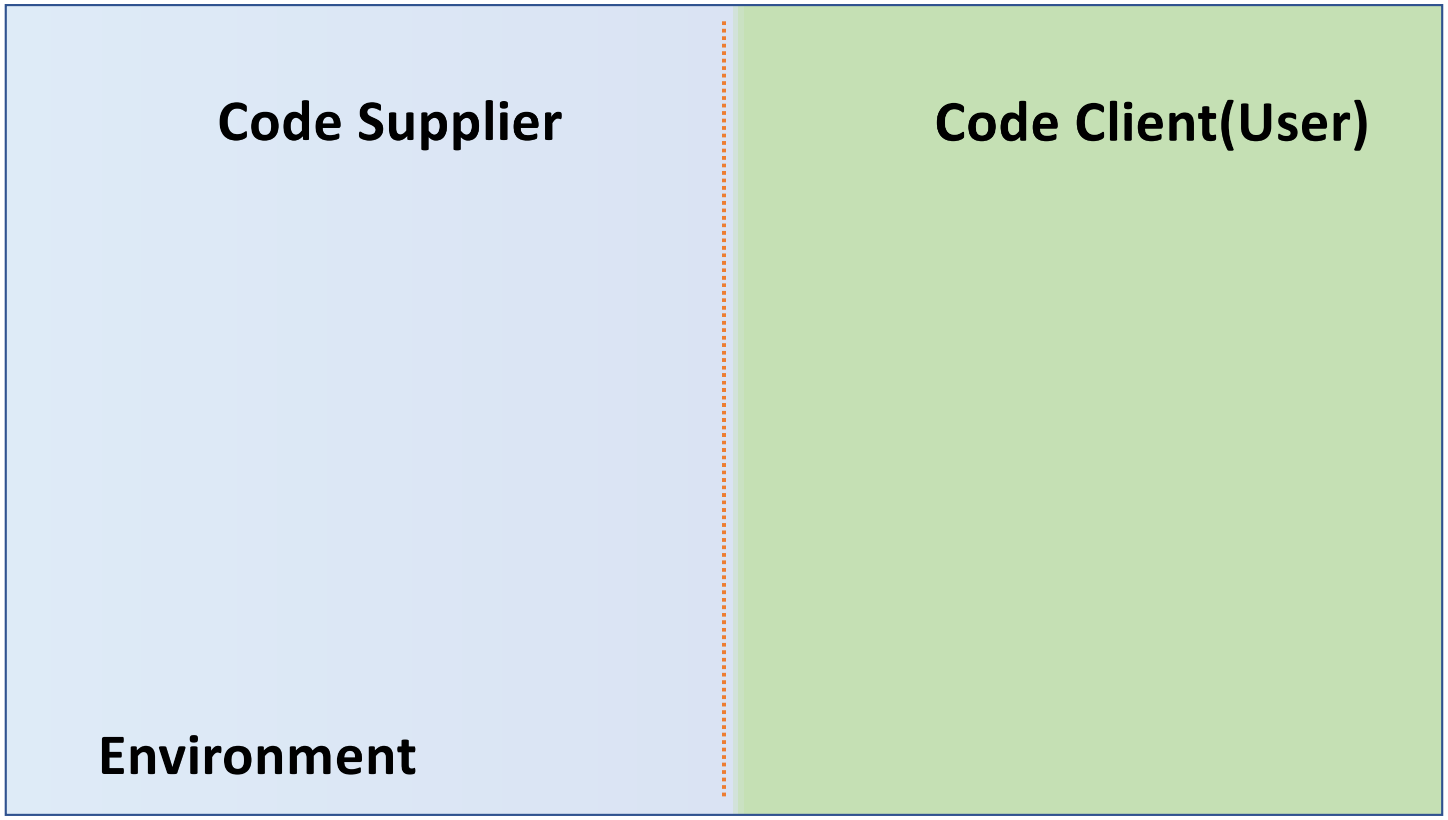
Logistics

Midterm

Date: Mar 9th, 6-8pm
Location: Leacock 132.
Coverage: All Content until Unit Testing (inclusive)

One-page handwritten crib sheet (not typed nor screenshots) is allowed.

Where can things go wrong?



Java Convention for Checking Preconditions

Explicit checks that throw particular, specified exceptions

Use assertion to test a *nonpublic* method's precondition that you believe will be true no matter what a (external) client does with the class.

Java Convention for Private Method

```
 * … …
 * @pre pStudent != null && !isFull()
 * @post aEnrollment.get(aEnrollment.size()-1) == pStudent()
 */
```

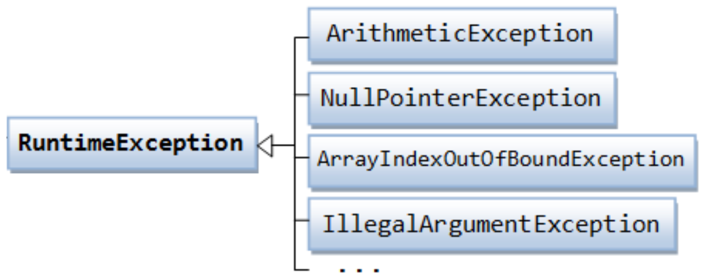
When this is private or protected

```
public void enroll(Student pStudent) {
    assert pStudent != null && !isFull() : this;
    aEnrollment.add(pStudent);
}

public boolean isFull() {
    return aEnrollment.size() == aCap;
}
```

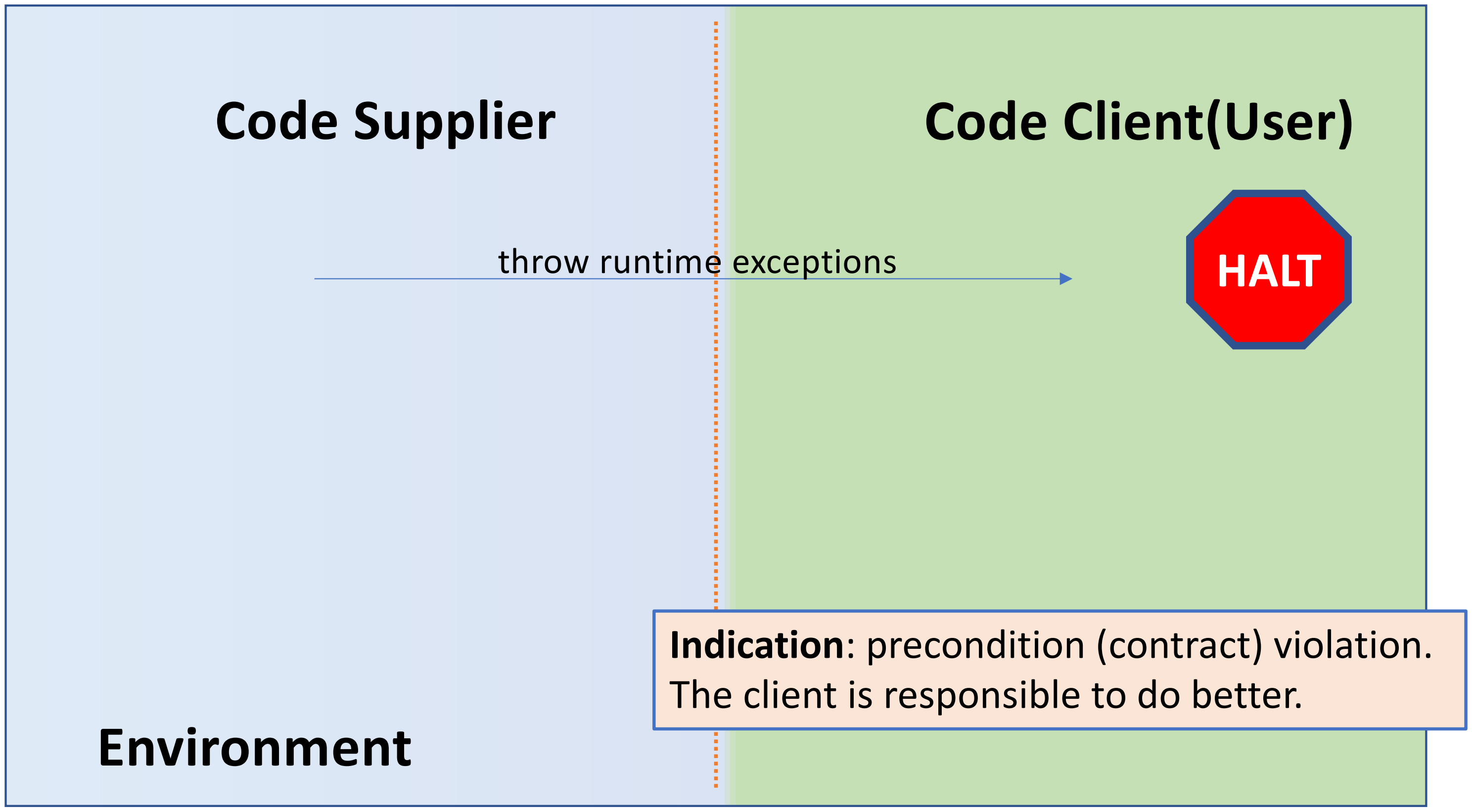
Java Convention for Public Method

```
/**
 * … …
 * @param Student to be enrolled to the Course
 * @throws IllegalStateException        if isFull()
 */

public void enroll(Student pStudent) {

    if (pStudent == null)
        throw new NullPointerException();
    if (isFull())
        throw new IllegalStateException();
    aEnrollment.add(pStudent);

}
```

Runtime Exceptions

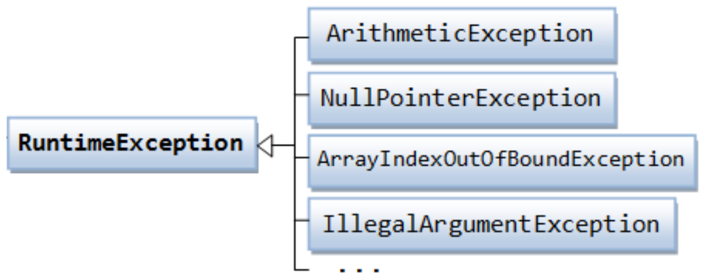


Code Interaction



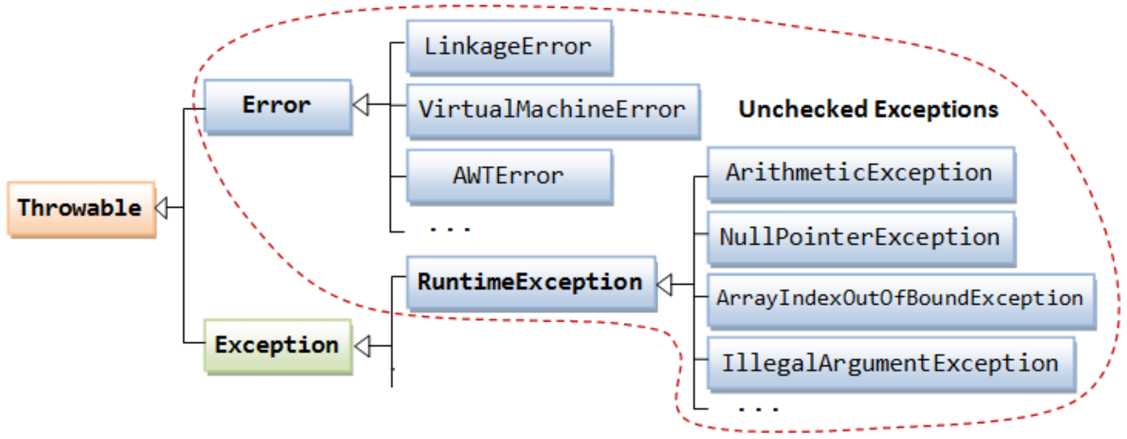
Indication: precondition (contract) violation.
The client is responsible to do better.

Runtime Exceptions



Unchecked Excaptions

They all cause the program to halt.



The whole hierarchy

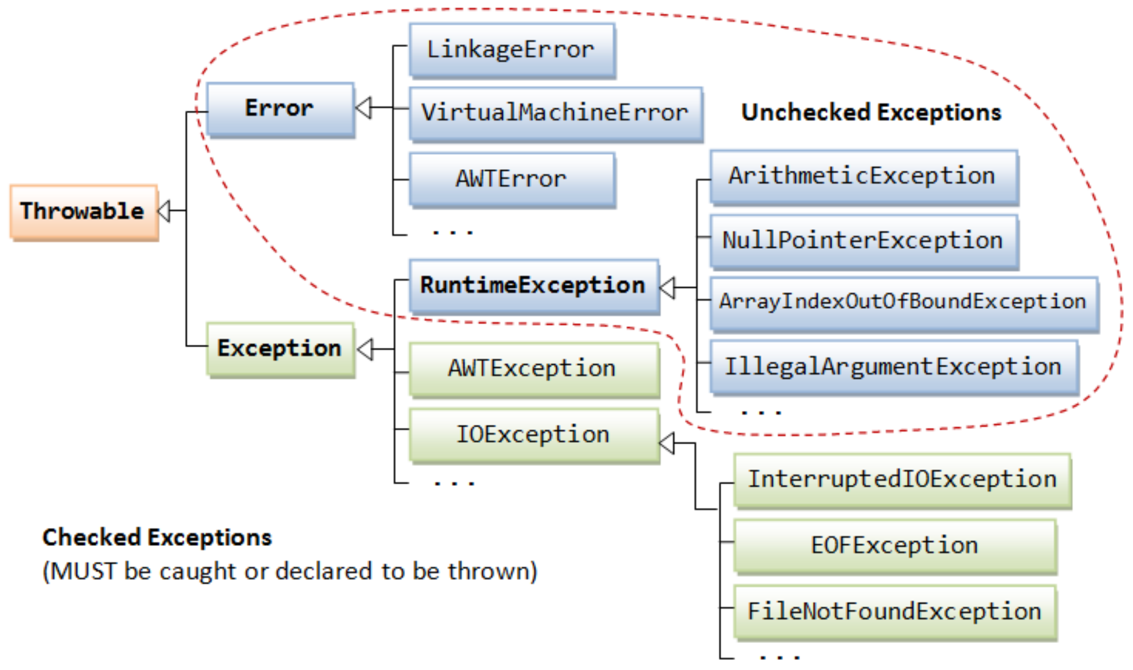
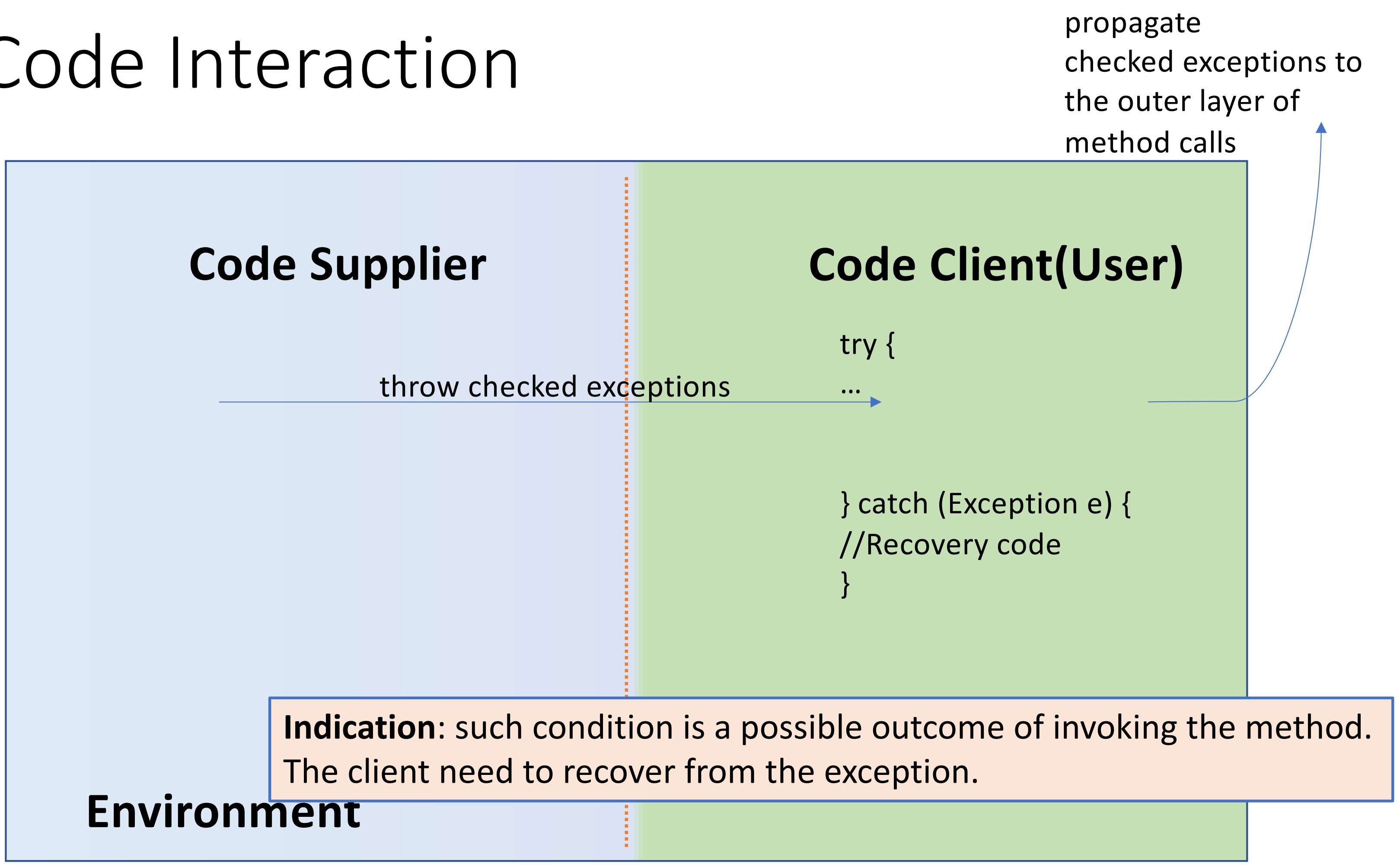


image source:http://www.ntu.edu.sg/home/ehchua/programming/java/images/Exception_Classes.png

Code Interaction



```
try {
...


} catch (Exception e) {
//Recovery code
}
```

Indication: such condition is a possible outcome of invoking the method. The client need to recover from the exception.

Another design of the `enroll` method

Assume `CourseFullException` is a Checked Exception

```
/**
 * Enroll the student to the course if the course currently is not full
 * @param pStudent to be enrolled to the Course
 * @throws    CourseFullException if isFull()
 */

public void enroll(Student pStudent) throws CourseFullException {
    if (pStudent == null)
        throw new NullPointerException();
    if (isFull())
        throw new CourseFullException();
    aEnrollment.add(pStudent);
}
```

`CourseFullException extends Exception`

Impact to the Client

The client is not obliged to check `isFull()` anymore. However...

```java
Course comp303 = new Course("COMP 303", 1);
Undergrad s1 = new Undergrad("00009", "James", "Harris");
Undergrad s2 = new Undergrad("00002", "Benny", "Will");

comp303.enroll(s1);
comp303.enroll(s2);

System.out.println("Done with enrolling students.");
comp303.printEnrolledStudent();
```

Impact to the Client

They have to catch the potential exception or propagate it

```
Course comp303 = new Course("COMP 303", 1);
Undergrad s1 = new Undergrad("00009", "James", "Harris");
Undergrad s2 = new Undergrad("00002", "Benny", "Will");
try {
    comp303.enroll(s1);
    comp303.enroll(s2);
    System.out.println("Done with enrolling students.");
} catch (CourseFullException e){
    … … // Handle the exception
    e.printStackTrace();
}
comp303.printEnrolledStudent();
```

Summary: Checked vs Unchecked Exception

- Checked Exceptions

Code supplier needs to declare in the method signature.
Code client needs to catch or declare.

Used for abnormal cases but can be recovered at runtime

- Unchecked Exceptions

Code supplier does **not** have to declare it
Code client does **not** have to catch nor declare it.

Used for programming errors or things should not happen at runtime.

Any problem with this method?

```
public void writeToFile(Course pCourse, String pFilePath) {
    File file = new File(pFilePath);

    try {
         FileWriter fileWriter = new FileWriter(file);
        for (Student s : pCourse) {
            fileWriter.write(s.toString());
            fileWriter.write("\n");
        }
      fileWriter.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

If exceptions happen here

The `final` block

```java
public void writeToFile(Course pCourse, String pFilePath) {
    File file = new File(pFilePath);
    FileWriter fileWriter = null;
    try {
        fileWriter = new FileWriter(file);
        for (Student s : pCourse) {
            fileWriter.write(s.toString());
            fileWriter.write("\n");
        }
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
         try {
             fileWriter.close();
         } catch (IOException e) {
             e.printStackTrace();
         }
    }
}
```

Alternative: try-with-Resources statement

```
public void writeToFile2(Course pCourse, String pFilePath) {
    File file = new File(pFilePath);
    try (FileWriter fileWriter = new FileWriter(file)){
        for (Student s : pCourse) {
            fileWriter.write(s.toString());
            fileWriter.write("\n");
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

`close()` will be called when the try block exits.

Case study:

```
if(!comp303.isFull())
    comp303.enroll(s2);
```

vs

```
try {
    comp303.enroll(s2);
} catch (CourseFullException e){
    … … // Handle the exception
}
```

When Not to use Exceptions

- For ordinary control flow

Acknowledgement

- Some examples are from the following resources:
  - *COMP 303 Lecture note* by Martin Robillard.
  - *The Pragmatic Programmer* by Andrew Hunt and David Thomas, 2000.
  - *Effective Java* by Joshua Bloch, 3rd ed., 2018.
  - *Clean Code* by Robert C. Martin, 2009
  - *A Philosophy of software design* by John Ousterhout, 2018