M4 (b) – Design for Robustness

Jin L.C. Guo

Logistics

Lab Test 2

- (If not done so) sign up before the end of Feb 9th
  - Follow the same rule (see the previous announcement on MyCourses).
- Focus: Types and Polymorphism and Object State, anything before is also in scope

Objective

- Programming mechanism:

Java Assertions, Exception Handling

- Concepts and Principles:

Code style

- Design techniques:

Design by contract, $\boxed{\text{Documentation}}$

Documentation

- Interface
  - a comment block precedes the declaration of a class, data structure, or method.
- Data fields
  - a comment next to the declaration of a static or non-static variable.
- Implementation comments
  - a comment inside a method

Interface Documentation

- Define abstractions
- Information for ***using*** a class or method

Interface Documentation

- Define abstractions
- Information for ***using*** a class or method

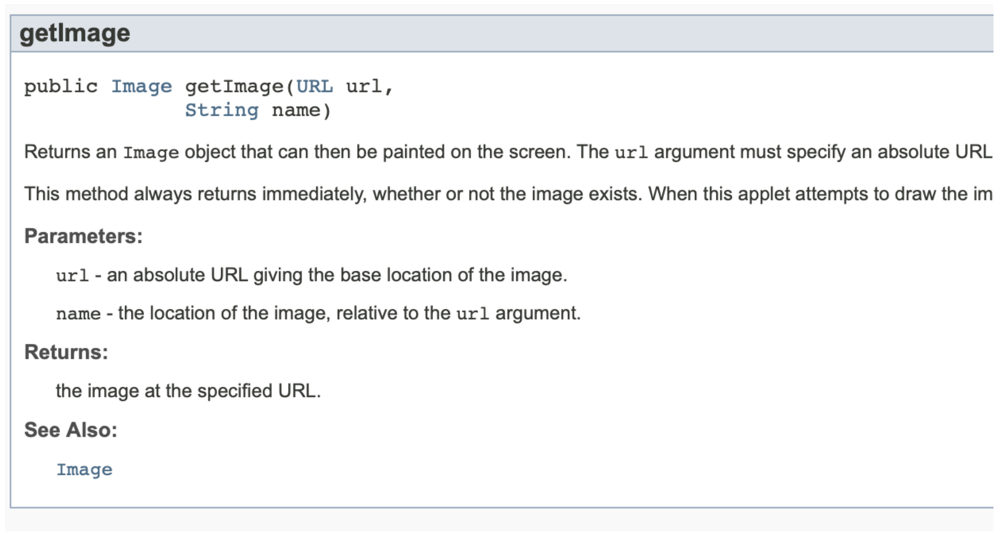
The comment doesn't do any of those!

```
/**
 * Returns an Image object by their url
 *
 * @param url image url
 * @param  name image name
 * @return      image object
 */
public Image getImage(URL url, String name) {
   try {
      return getImage(new URL(url, name));
   } catch (MalformedURLException e) {
      return null;
   }
}
```

```java
/**
 * Returns an Image object that can then be painted on the screen.
 * The url argument must specify an absolute {@link URL}. The name
 * argument is a specifier that is relative to the url argument.
 *
 * This method always returns immediately, whether or not the
 * image exists. When this applet attempts to draw the image on
 * the screen, the data will be loaded. The graphics primitives
 * that draw the image will incrementally paint on the screen.
 *
 * @param url   an absolute URL giving the base location of the image
 * @param  name the location of the image, relative to the url argument
 * @return      the image at the specified URL
 * @see         Image
 */
public Image getImage(URL url, String name) {
   try {
      return getImage(new URL(url, name));
   } catch (MalformedURLException e) {
      return null;
   }
}
```

Use Javadoc for Public APIs

- Documentation -> HTML pages describing the classes, interfaces, constructors, methods, and fields.



getImage

```
public Image getImage(URL url,
                      String name)
```

Returns an `Image` object that can then be painted on the screen. The `url` argument must specify an absolute URL

This method always returns immediately, whether or not the image exists. When this applet attempts to draw the im

Parameters:

`url` - an absolute URL giving the base location of the image.

`name` - the location of the image, relative to the `url` argument.

Returns:

the image at the specified URL.

See Also:

`Image`

Use Javadoc for Public APIs

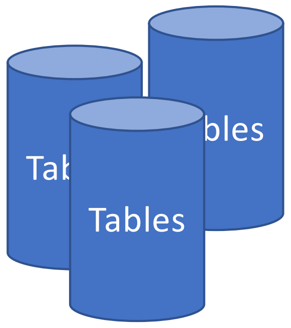
- @param
- @return
- @throws
- @see
- @author
- {@code}

… …

Adding customized tag is also possible
@custom.mytag

Activity 2

- `IndexLookup` class for distributed storage system.



Object	Name	Age	…
A-1	John	20	…
A-2	Elizabeth	21	…
…	…	…	…

```
IndexLookup query = new IndexLookup(table, index, key1, key2);
Iterator iterator = query.iterator();
while(iterator.hasNext())
{
    object = iterator.next()
    … …
}
```

Activity 2

- Does the user of the `IndexLookup` class need to know the following:

  1. The format of message that **IndexLookup** class sends to the servers holding indexes and objects.

  2. The comparison function used to determine whether a particular object falls in the designed range (comparison using integers, floating points, or strings).

  3. The data structure used to store indexes on servers.

  4. Whether **IndexLookup** issues multiple requests to different servers concurrently.

  5. The mechanisms for handling server crashes.

Data field

- Explain, not repeat

```
/**
 * the horizontal padding of each line in the text
 */
private static final int textHorizontalPadding = 4;
```

vs

```
/**
 * The amount of blank space to leave on the left and
 * right sides of each line of text, in pixels.
 */
private static final int textHorizontalPadding = 4;
```

Data field

- Fill in missing details (that you cannot get from name and type)

```
//Contains all term within the document and their number
of appearances
private TreeMap<String, Integer> termAppearances;
```

VS

```
//Hold the statistics about the term appearances within a
//document in the form of <term, count> where the term is the
//word in its dictionary form, and the count is how many times
//it matches the tokens in the document after preprocessing.
//If a term doesn't match any token in the document, then
//there's no entry for that term.
private TreeMap<String, Integer> termAppearances;
```

Implementation comments

- For understand what the code is doing
  - Add a comment before each major block for abstract description

```
// Compute the standard deviation of list elements that are
// less than the cutoff value.
 for (int i = 0; i < n; i++) {
    …
 }
```

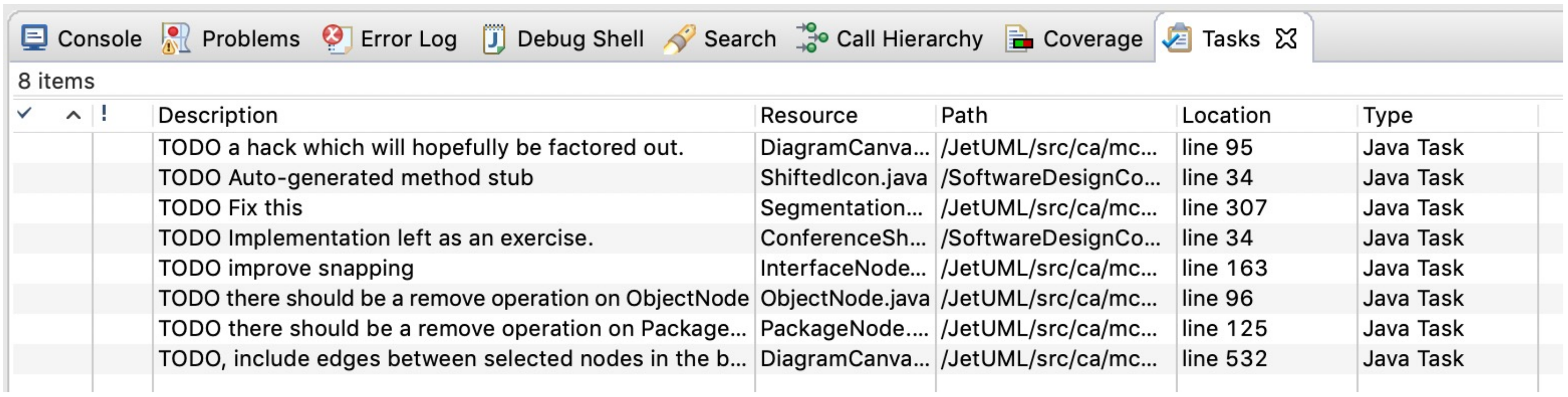
- For understand why the code is written this way.

```
// Arbitrary default value, used to simplify the testing code
 private static final int DEFAULT_DIMENSION = 1000;
```

More Informative Comments

- *Record Assumptions*
- *Record Limitations*
- *TODO comments*

……



8 items

✓	^	!	Description	Resource	Path	Location	Type
			TODO a hack which will hopefully be factored out.	DiagramCanva...	/JetUML/src/ca/mc...	line 95	Java Task
			TODO Auto-generated method stub	ShiftedIcon.java	/SoftwareDesignCo...	line 34	Java Task
			TODO Fix this	Segmentation...	/JetUML/src/ca/mc...	line 307	Java Task
			TODO Implementation left as an exercise.	ConferenceSh...	/SoftwareDesignCo...	line 34	Java Task
			TODO improve snapping	InterfaceNode...	/JetUML/src/ca/mc...	line 163	Java Task
			TODO there should be a remove operation on ObjectNode	ObjectNode.java	/JetUML/src/ca/mc...	line 96	Java Task
			TODO there should be a remove operation on Package...	PackageNode....	/JetUML/src/ca/mc...	line 125	Java Task
			TODO, include edges between selected nodes in the b...	DiagramCanva...	/JetUML/src/ca/mc...	line 532	Java Task

Smells in Comments

Repeat the code

About the implementation details

Journal comments

Misleading comments

Outdated comments

… …

Comments As a Design Tool

Write comments first:

- Capture the abstraction before implementation
- Reveal potential problem of design (complexity)
- Improve quality of documentation

Objective

- Programming mechanism:

Java Assertions, Exception Handling

- Concepts and Principles:

$\boxed{\text{Code style}}$

- Design techniques:

Design by contract, Documentation

Code Style

- Goal: reduce complexity
  - to understand the code
  - to make future changes

Naming Entities

- Packages
- Classes/Enums
- Interfaces/Annotations
- Members of Reference types
- Parameters
- Local variables

Naming Entities

- Principle
  - Be clear and descriptive
  - Reveal your intention
  - Follow conventions
    - [Java Naming Conventions]
    - EJ3: 68

```
int d; // elapsed time in days
```

↓

```
int elapsedTimeInDays;
```

Formatting

- Braces
- Indentation
- Spacing

…

```
public class CommentWidget extends TextWidget
{
  public static final String REGEXP = "^#[^\r\n]*(?:(?:\r\n)|\n|\r)?";
  public CommentWidget(ParentWidget parent, String text){super(parent, text);}
  public String render() throws Exception {return “”; }
}
```

Not Easy to read…

Formatting

- Braces
- Indentation
- Spacing

…

Easy to read

Consistent

```
return new MyClass() {
  @Override public void method() {
    if (condition()) {
      try {
        something();
      } catch (ProblemException e) {
        recover();
      }
    } else if (otherCondition()) {
      somethingElse();
    } else {
      lastThing();
    }
  }
};
```

Objective

- Programming mechanism:

Java Assertions, $\boxed{\text{Exception Handling}}$

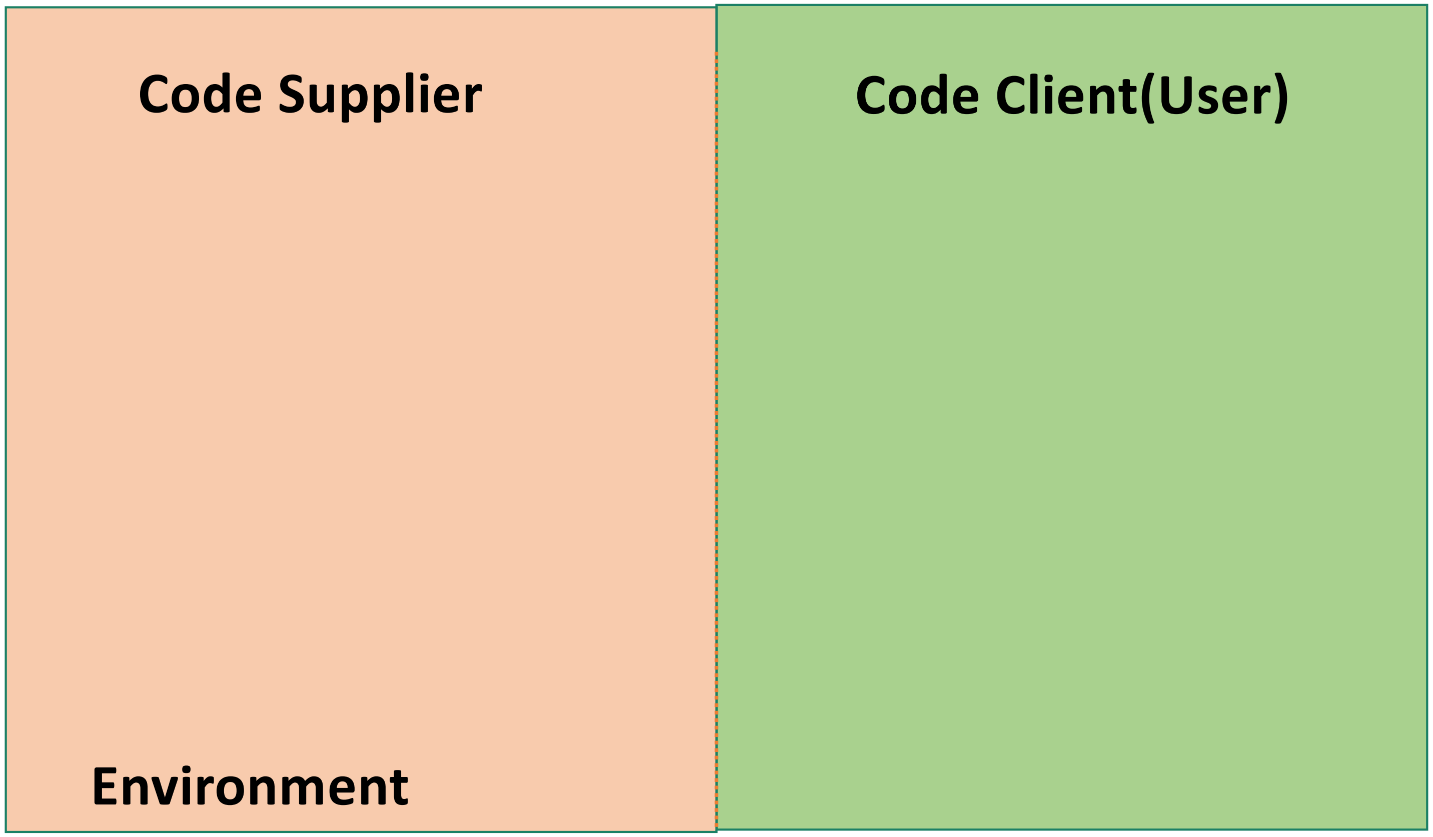
- Concepts and Principles:

Code style

- Design techniques:

Design by contract, Documentation

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()
 */
```

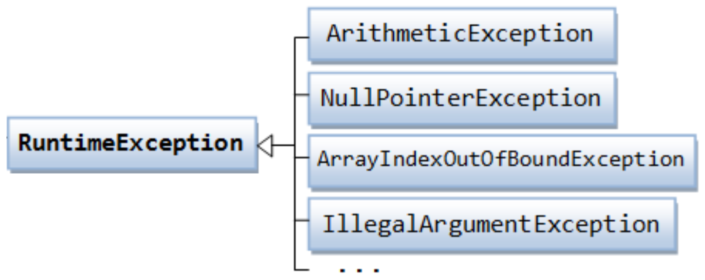
When this is private or protected

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

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

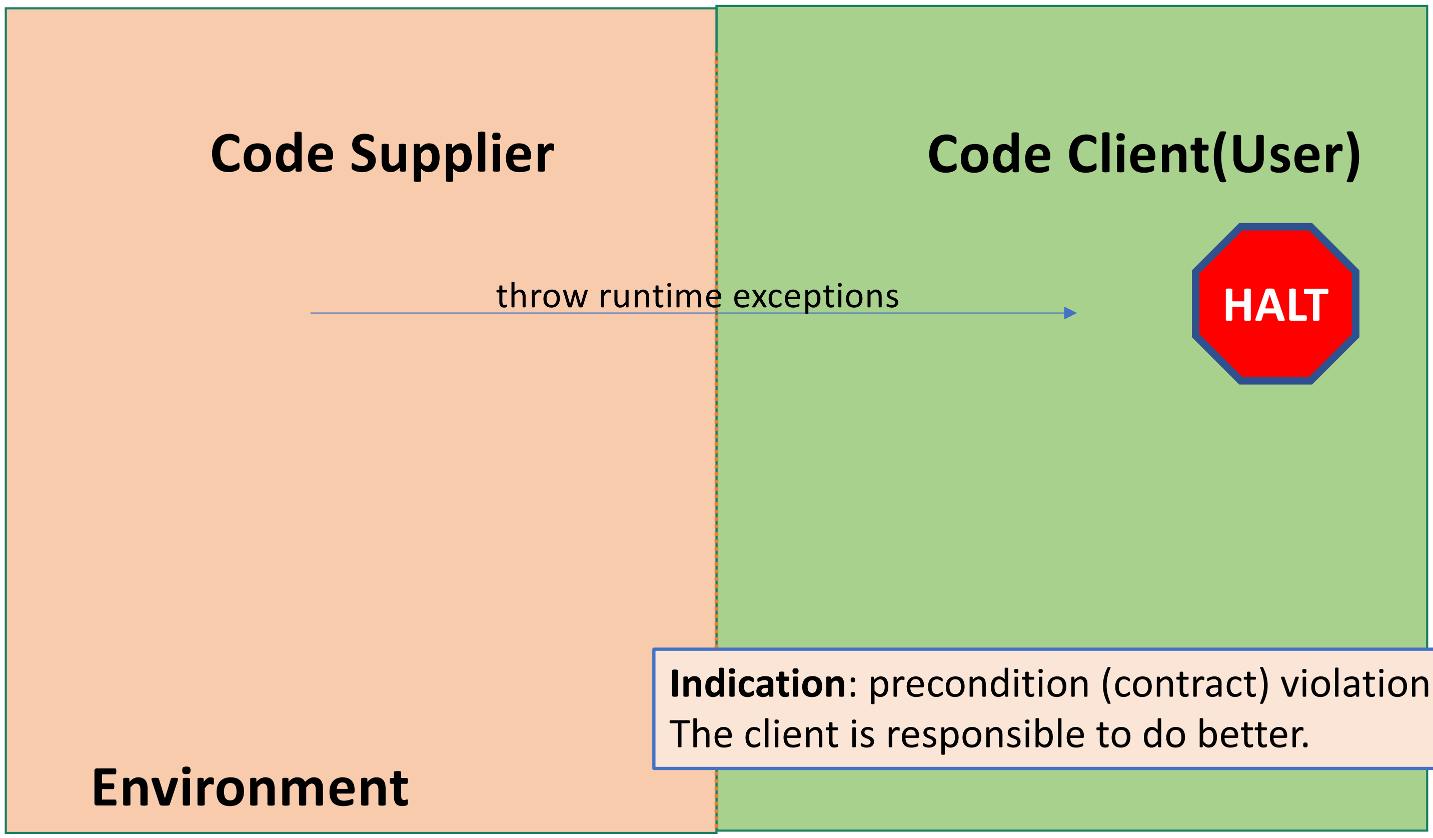
Java Convention for Public Method (used by external client)

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

public void enroll(Student pStudent) {
      if (pStudent == null)
          throw new NullPointerException();
      if (isFull())
          throw new IllegalStateException();

    aEnrollment.add(pStudent);

}
```

Runtime Exceptions

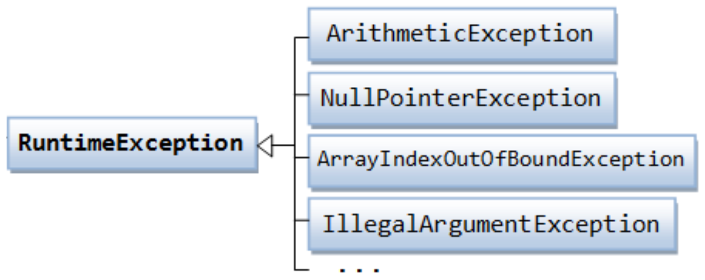


Code Interaction



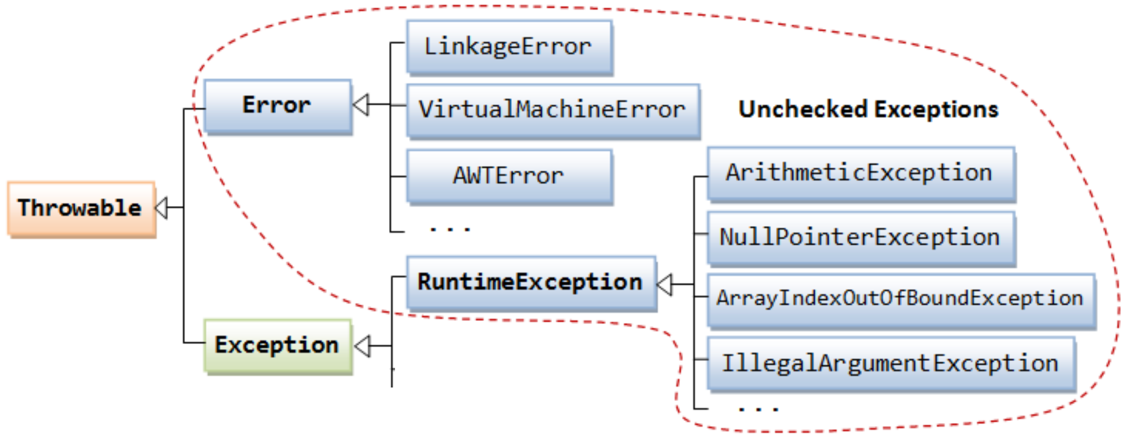
Indication: precondition (contract) violation. The client is responsible to do better.

Runtime Exceptions



Unchecked Excaptions

They all cause the program to halt.



The whole hierarchy

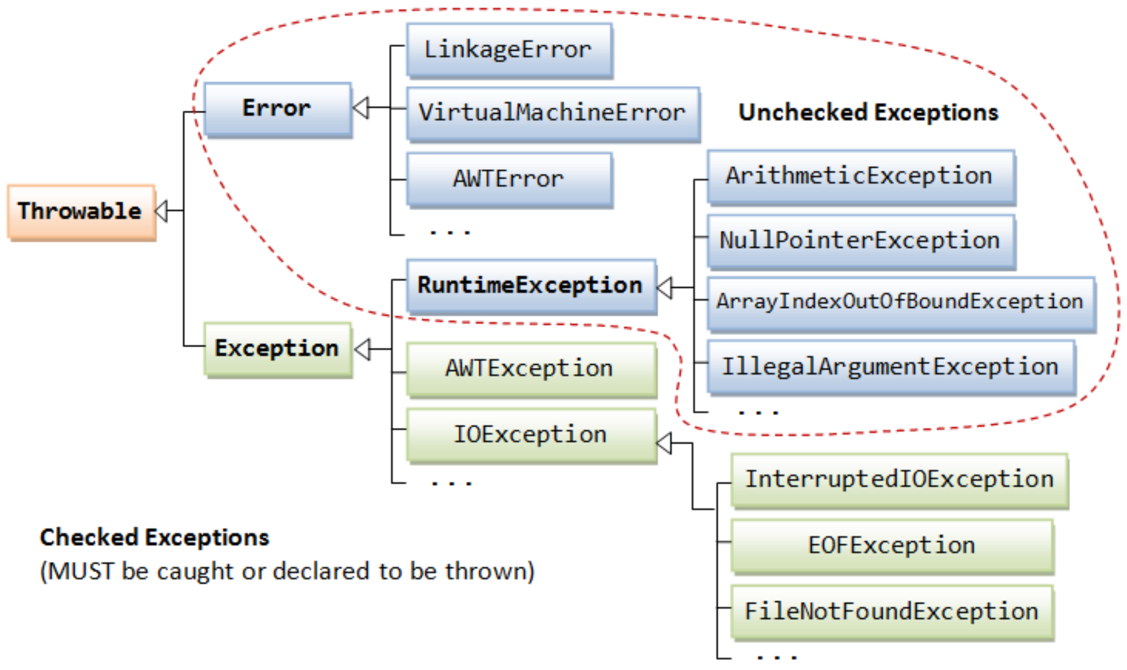
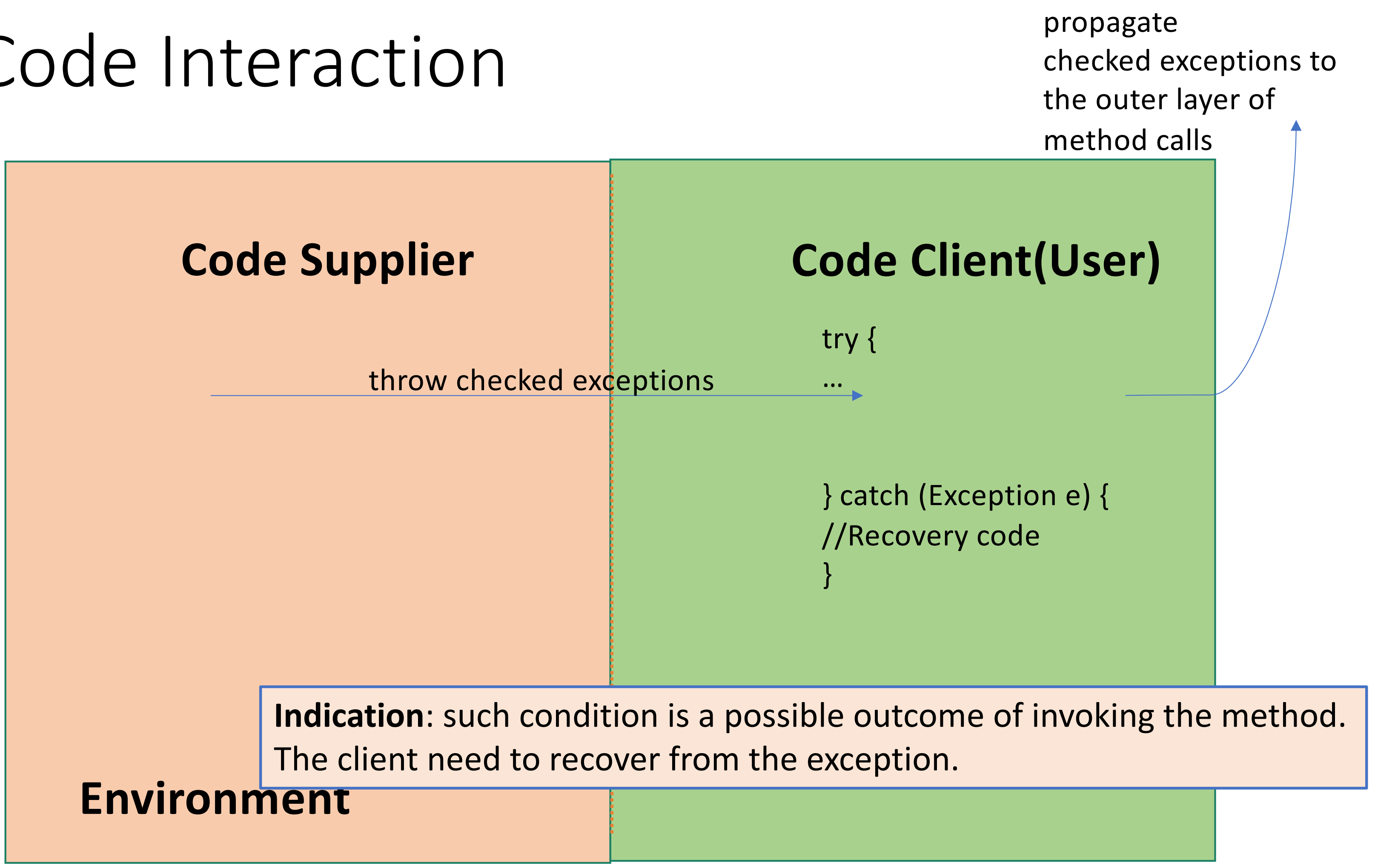


image source:http://www.ntu.edu.sg/home/ehchua/programming/java/images/Exception_Classes.png

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…

```
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