M4 (a) – Design for Robustness

Jin L.C. Guo

Objective

- Programming mechanism:

Java Assertions, Exception Handling

- Concepts and Principles:

Code style

- Design techniques:

Design by contract, Documentation

Consider the `enroll` method for `Course`

```
public class Course {

    private String aID;
    private int aCap;
    private List<Student> aEnrollment;



    public boolean enroll(Student pStudent) {
        if(aEnrollment.size()<aCap) {
            aEnrollment.add(pStudent);
            return true;
        }
        return false;
    }
    … …
}
```

Can things go wrong?
(assume that Student is immutable)

Consider the `enroll` method for `Course`

```
public class Course {

    private String aID;
    private int aCap;
    private List<Student> aEnrollment;



    public boolean enroll(Student pStudent) {
        if(aEnrollment.size()<aCap) {
            aEnrollment.add(pStudent);
            return true;
        }
        return false;
    }
    … …
}
```

Things can still go wrong!

```
                                  student == null

Course comp303 =
        new Course("COMP 303", 200);
comp303.enroll(student);
```

Consider the `enroll` method for `Course`

Things can still go wrong!

```
                                    student == null

Course comp303 =
        new Course("COMP 303", 200);
comp303.enroll(student);
```

```
Exception in thread "main" java.lang.NullPointerException Create breakpoint : Cannot invoke "ca.mcgill.cs.swdesign.m3.Student.getFirstName()" because "o1" is null
    at ca.mcgill.cs.swdesign.m3.CourseSystem$1.compare(CourseSystem.java:33)
    at ca.mcgill.cs.swdesign.m3.CourseSystem$1.compare(CourseSystem.java:30)
    at java.base/java.util.TimSort.binarySort(TimSort.java:296)
    at java.base/java.util.TimSort.sort(TimSort.java:221)
    at java.base/java.util.Arrays.sort(Arrays.java:1306)
    at java.base/java.util.ArrayList.sort(ArrayList.java:1721)
    at java.base/java.util.Collections.sort(Collections.java:179)
    at ca.mcgill.cs.swdesign.m3.Course.sortStudent(Course.java:69)
    at ca.mcgill.cs.swdesign.m3.CourseSystem.enrollAndRankStudent(CourseSystem.java:36)
    at ca.mcgill.cs.swdesign.m3.CourseSystem.main(CourseSystem.java:11)
```

Fix ideas?

```
public class Course {

    private String aID;
    private int aCap;
    private List<Student> aEnrollment;



    public boolean enroll(Student pStudent) {
        if(aEnrollment.size()<aCap&& pStudent!=null) {
            aEnrollment.add(pStudent);
            return true;
        }
        return false;
    }
     ……
}
```

Any drawbacks with this fix?

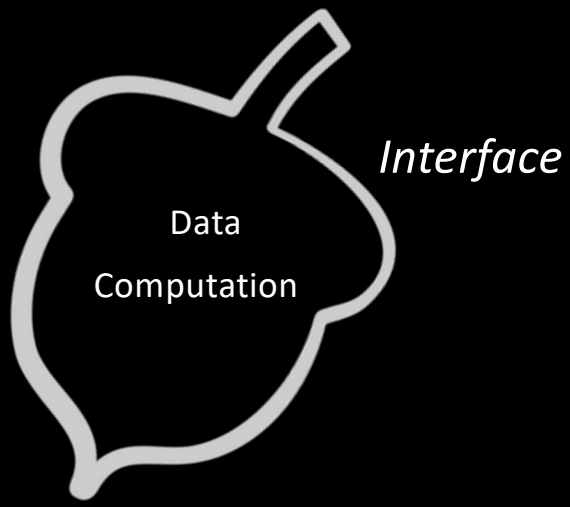
Fix ideas?

```
public class Course {

    private String aID;
    private int aCap;
    private List<Student> aEnrollment;

    public boolean enroll(Student pStudent) {
        if(pStudent == null)
            throw new IllegalArgumentException("The argument cannot be null");
        if(aEnrollment.size()<aCap) {
            aEnrollment.add(pStudent);
            return true;
        }
        return false;
    }

}
```

Defensive programming, more next class

From Module 2

Object Interaction



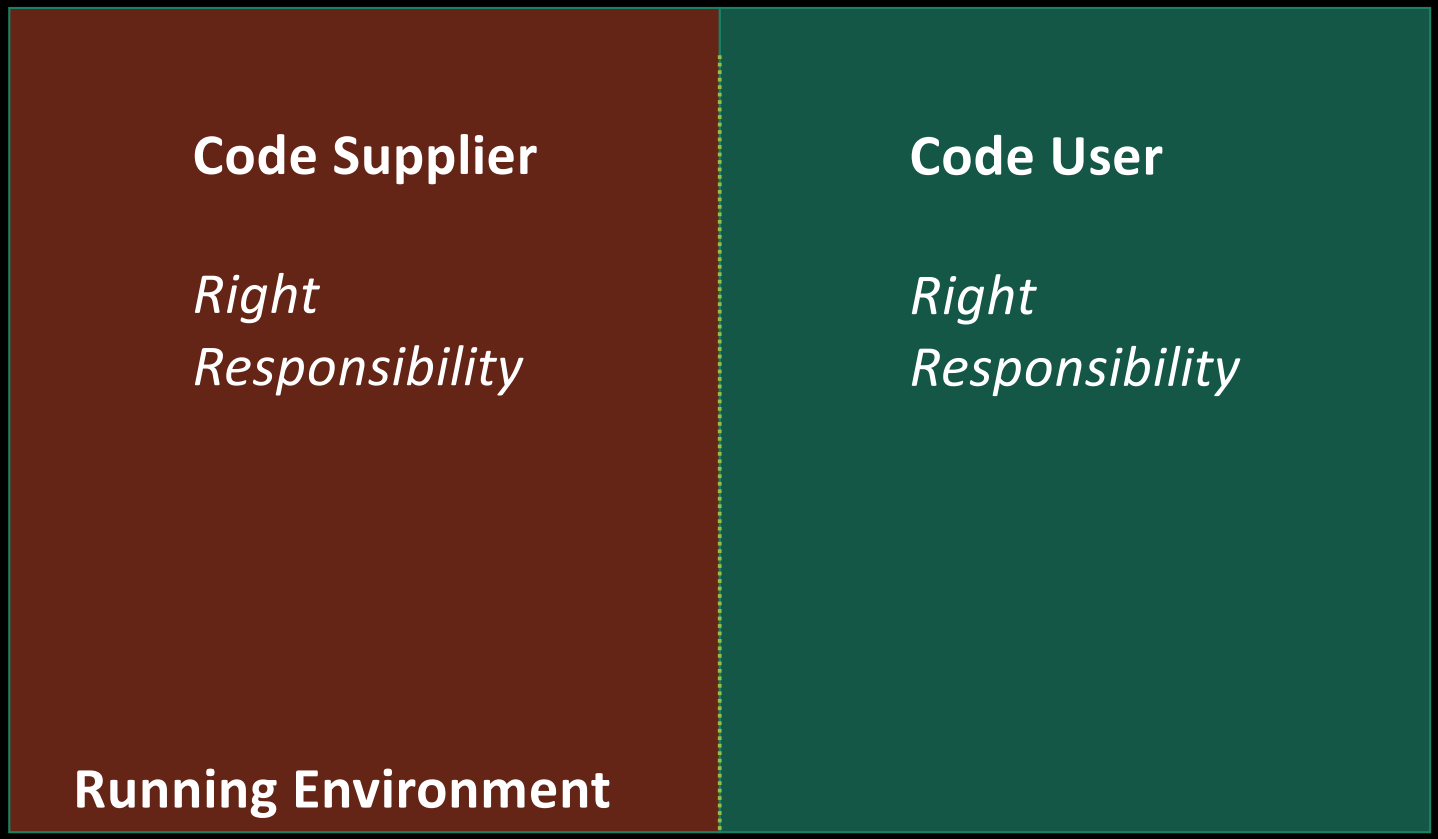
Supply the service through public interface

From Module 2

What should the public interface specify?

- Requires — What needs be true in order to call this the method?
- Modifies — When this method is called, is the state of any object going to be changed?
- Effects — What will happen if this method is called?

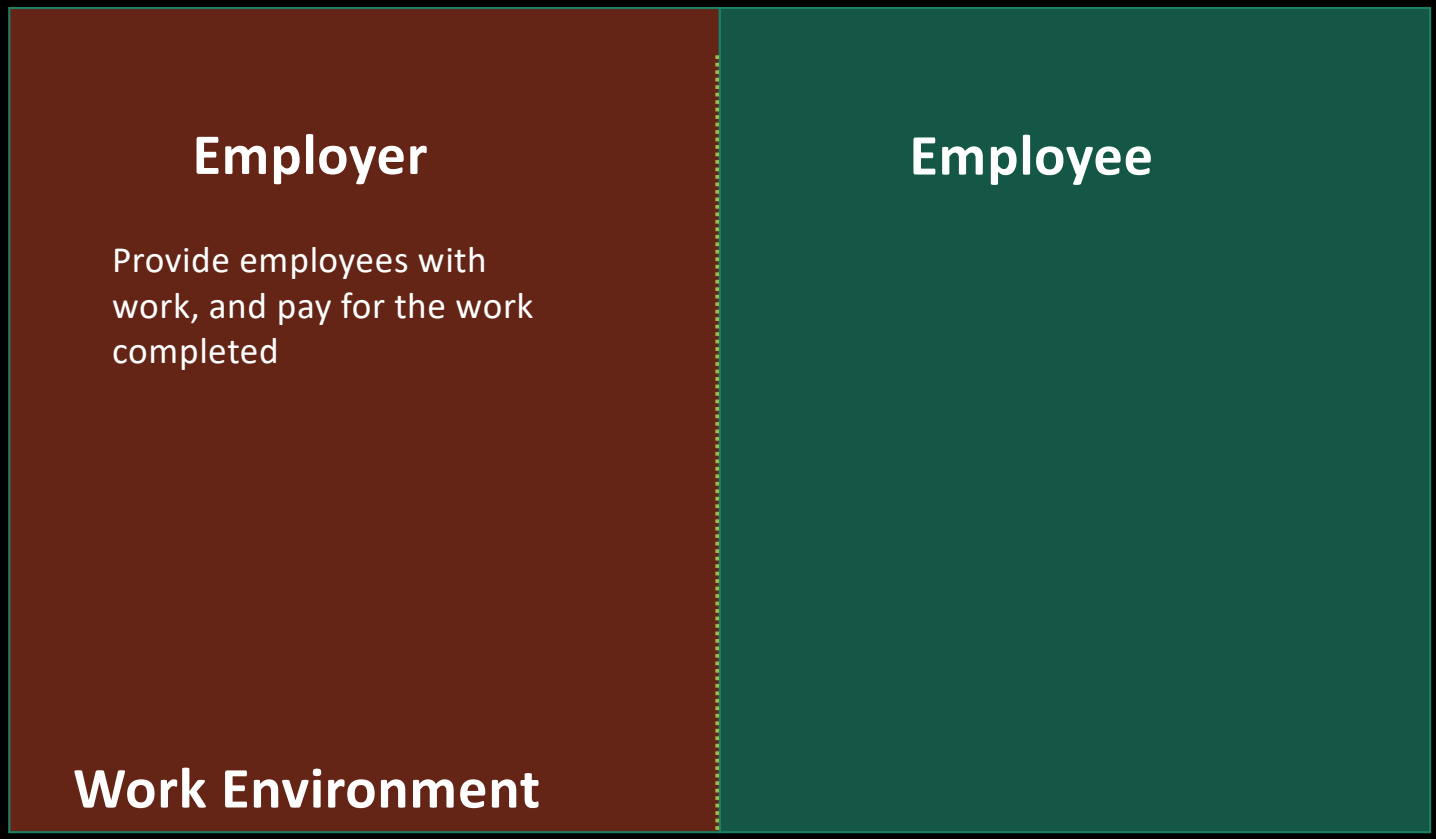
Contract (Human Interaction)



Example Contract (Human Interaction)

Example Contract (Human Interaction)

Example Contract (Human Interaction)



Example Contract (Human Interaction)

Design by Contract

- Documenting rights and responsibilities of software modules to ensure program correctness

does no more and no less than it claims to do

Specify the interface

- Precondition – What must be true in order for the routine to be called.

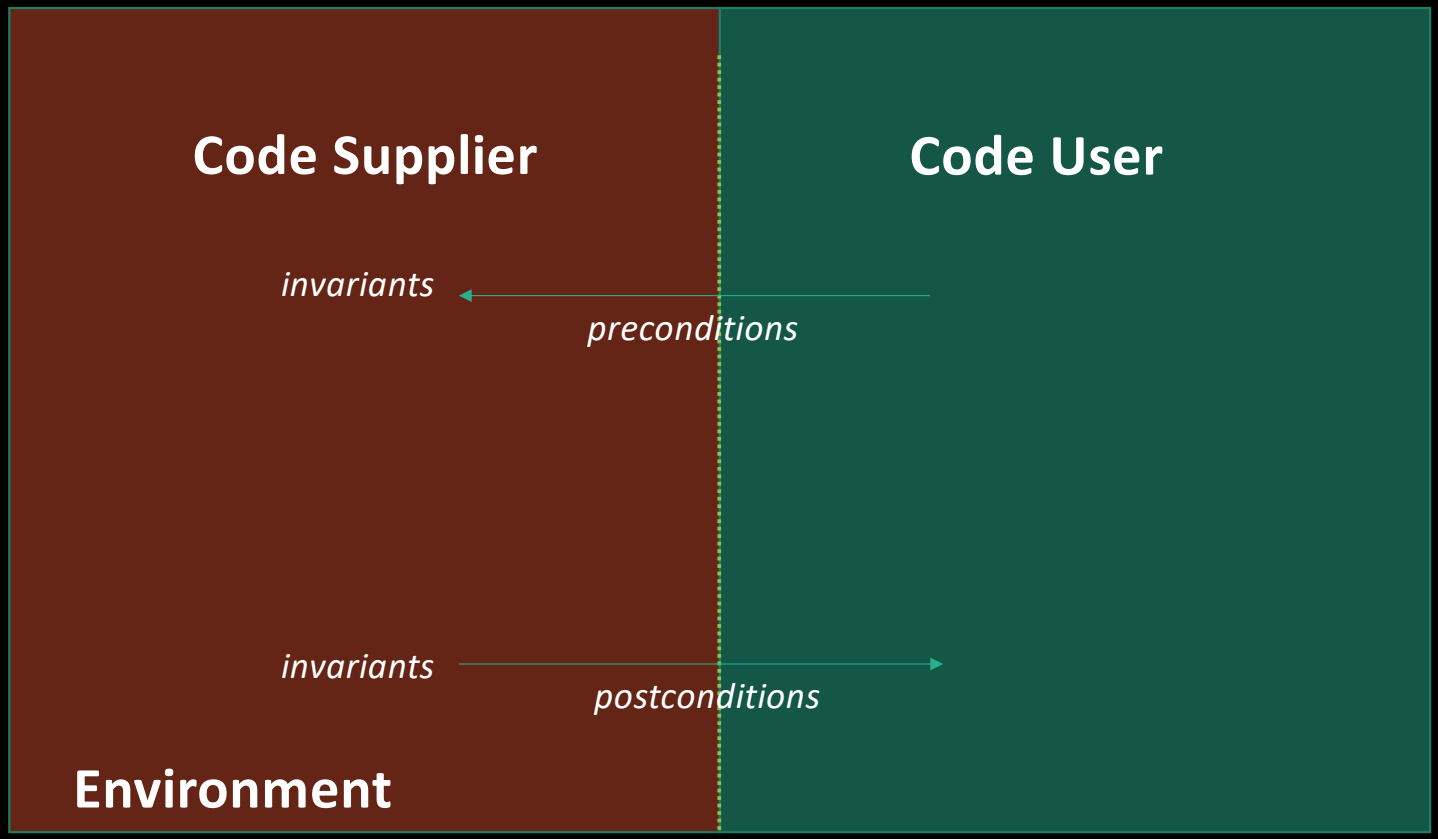
Code User's responsibility

- Postcondition – What the routine is guaranteed to do; the state of the world when the routine is done.

Code Supplier's responsibility

- Class invariants – Conditions that's always true (from the perspective of caller).

Design by Contract



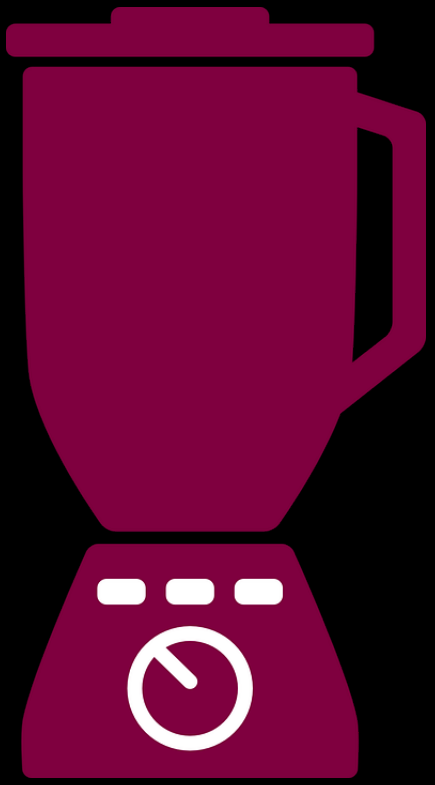
Specify Contract

```
/**
 * @invariant aEnrollment != null && aEnrollment.size() <= aCap
 *
 */


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

public void enroll(Student pStudent) {
    aEnrollment.add(pStudent);
}

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

Activity 1

- Consider an interface to a kitchen blender. It has ten speed settings (0-9, 0 means off). You can only operate when it's full. You can change the speed only one unit at a time (that is, from 0 to 1, and from 1 to 2, not from 0 to 2).
- Add appropriate pre- and postconditions and class invariant.

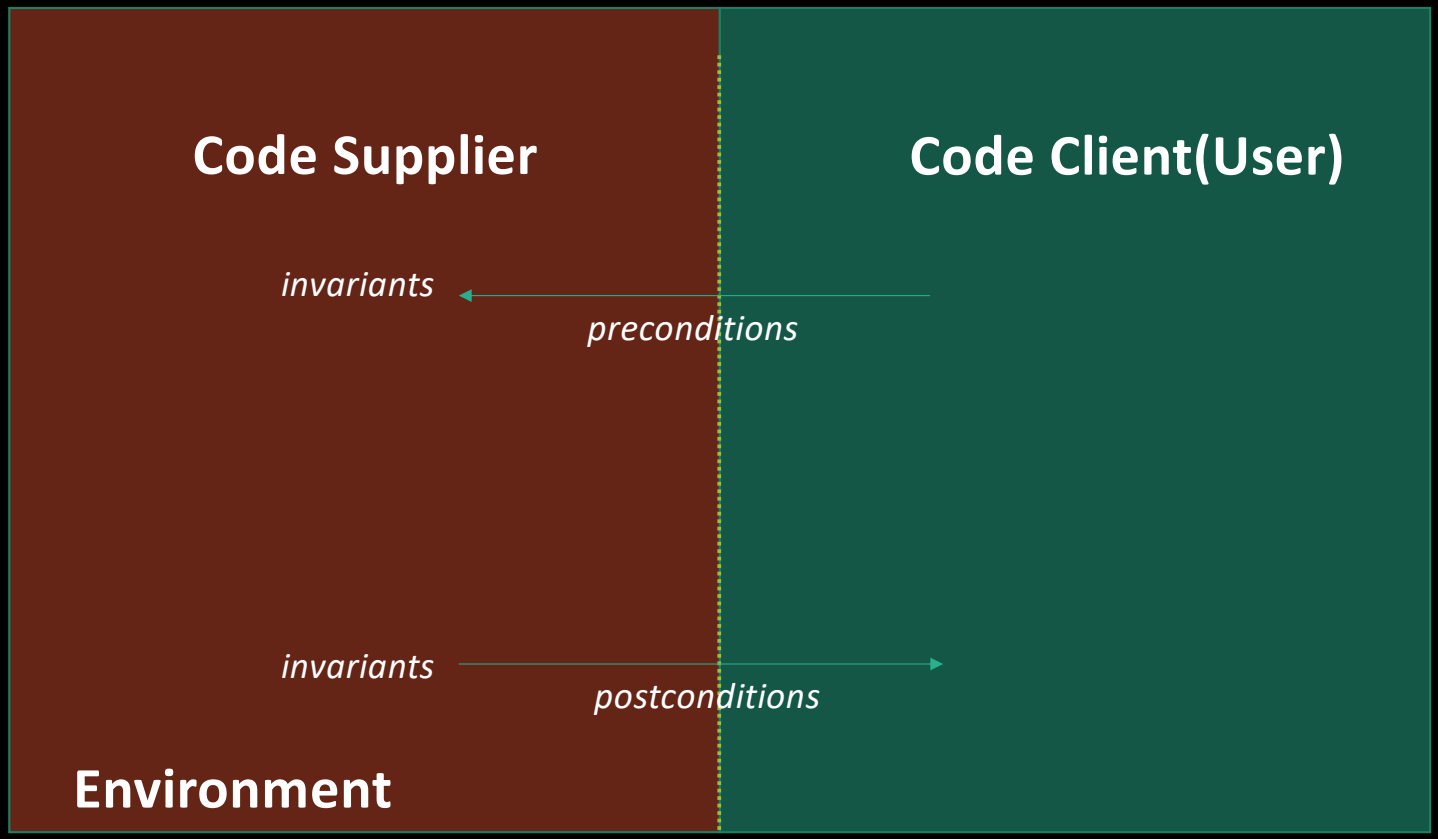
```
int getSpeed()
void setSpeed(int pSpeed)
boolean isFull()
void fill()
void empty()
```



```
/*
* @invariant if(getSpeed() >0) isFull()
* @invariant getSpeed()>=0 && getSpeed()<10
*/


/*
* @pre Math.abs(getSpeed() - pSpeed) == 1
* @pre pSpeed>=0 && pSpeed<10
* @post getSpeed() == pSpeed
*/
void setSpeed(int pSpeed)


/*
* @pre !isFull()
* @post isFull()
*/
void fill()
```

similar with empty()

Verifying Contract



Verifying Contract

- No build-in support in Java
- Partially achieved by assertion

Java Assertions

assert *Expression1* ;

assert *Expression1* : *Expression2* ;

Expression1 false `AssertionError`

Safety-net, not enforcement!

Ensure things shouldn't happened won't happen (correctness)

java –ea runs Java with **a**ssertions **e**nabled (disabled by default)

(Partially) Verifying Contract in Java

```
/**
 * … …
 * @pre pRank != null && pSuit != null
 * @post getRank() == pRank && getSuit() == pSuit
 */
public Card(Rank pRank, Suit pSuit) {
   assert pRank != null && pSuit != null;
   aRank = pRank;
   aSuit = pSuit;
   assert getRank() == pRank && getSuit() == pSuit;
}
```

(Partially) Verifying Contract in Java

- Evaluate the following contract for a stack class

```
/**
 * … …
 * @pre pCard != null
 * @post pop() == pCard
 */
public void push(Card pCard)
{… …}
```

pop() -> peek()

Heisenbug

a software bug that seems to disappear or alter its behavior when one attempts to study it.

Heisenberg

Design by Contract - Summary

- Purpose: ensure program correctness
- Correct -> does no more and no less than it claims to do
- Being “lazy”: be strict in what you will accept before you begin, and promise as little as possible in return

- Benefit: forces the issue of requirements and guarantees at design time – what your code (doesn’t) promise to deliver
- Means: documenting and verifying

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

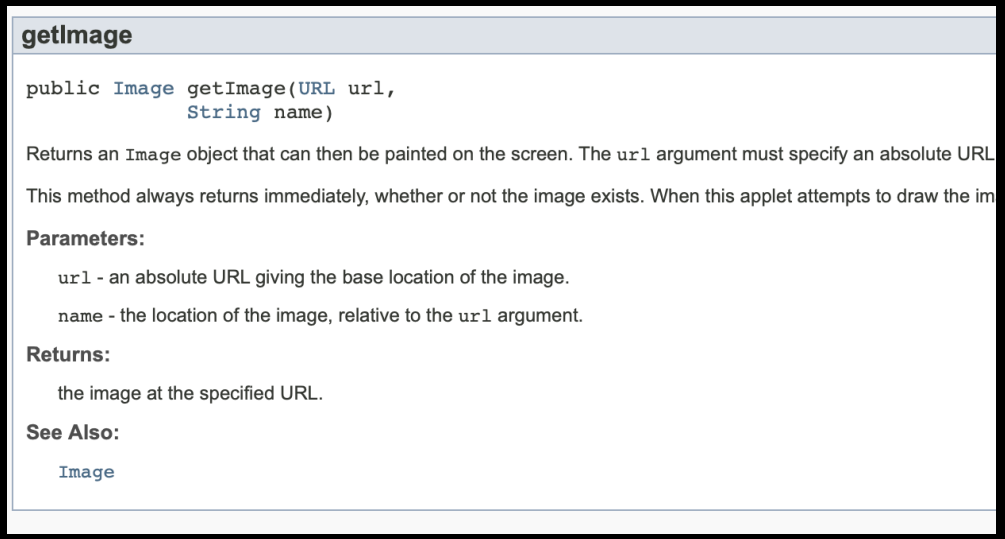
- Define abstractions
- Information for ***using*** a class or method

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



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

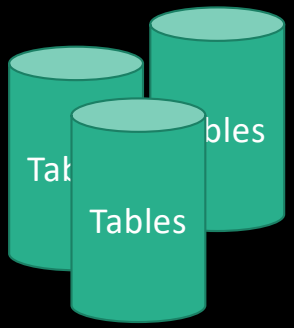
- @param
- @return
- @throws
- @see
- @author
- {@code}

… …

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 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++) {
    …
 }
```

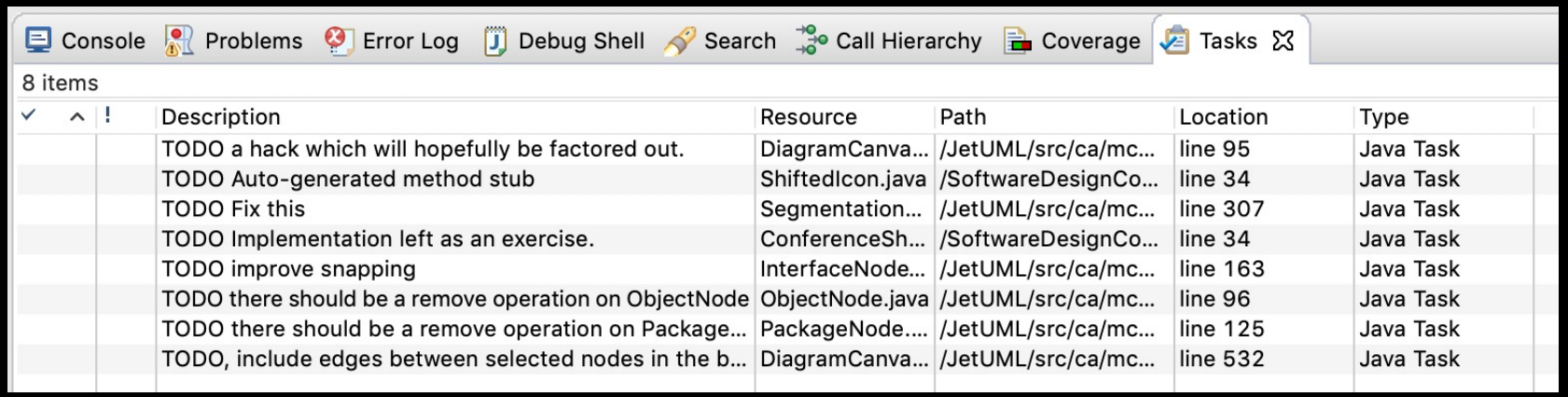
- For understand why the code is written this way.

```
// 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*

……



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

Recap Objective

- Programming mechanism:

Java Assertions, Exception Handling

- Concepts and Principles:

Code style

- Design techniques:

Design by contract, Documentation