7A. Object-Oriented Design I

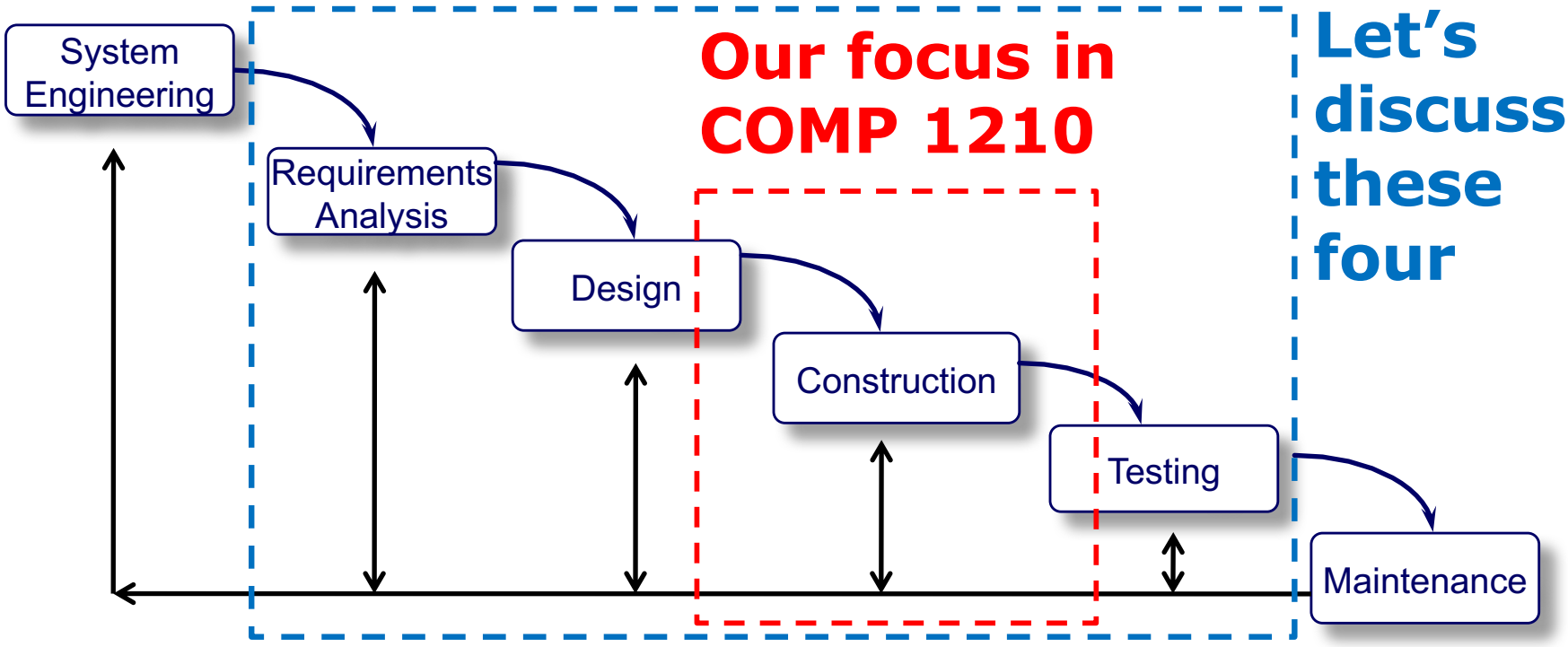
- Objectives - when we have completed this set of lecture notes, you should be familiar with:
  - Software development activities
  - determining the classes and objects that are needed for a program
  - the relationships that can exist among classes
  - the static modifier

Program Development

- The creation of software involves five basic activities plus maintenance:
  - Establishing **system** hardware/software requirements, boundaries, interfaces, etc. (esp. for embedded systems)
  - Establishing/analyzing the software **requirements**
  - creating a software **design**
  - **Constructing** the **code (a.k.a. Implementation)**
  - **testing** the implementation
  - **Maintaining** the software after delivery

Program Development

- The activities below can be linear, can overlap/interact, or can be naturally cyclical



- Many variants of the process model

Requirements

- *Software requirements* specify the tasks that a program must accomplish
  - what to do (not how to do it)
- Translating what the customer wants into what the software should accomplish.
- Sample customer requirement: "I need software that will provide interoperability between disparate C4ISR systems." (concept of Command, Control, Communications, Computers, Intelligence, Surveillance and Reconnaissance)
  - It's not time to start coding yet.

Requirements

- A simple requirement: "I want something to check products against their barcodes."
- A partial requirements document:

> **The user must be allowed to specify each product by its primary characteristics, including its name and product number. If the bar code does not match the product, then an error should be generated to the message window and entered into the error log. The summary report of all transactions must be structured as specified in section 7.A.**

Design

- S*oftware design*: how a program will accomplish its requirements.
- An object-oriented design:
  - Classes, methods, and data needed
  - Relationships between/among classes
  - How information will be stored (databases)
  - Etc.
- Your project descriptions in lab contain both requirements and design elements.

Construction

- *Construction*: turning design into source code
- Novice programmers often think that writing code is the hard part of software development; actually it may be the most straightforward step
  - The important/hard decisions were made during requirements and design stages
- Construction (a.k.a. Implementation) focuses on coding (and many times fine-grained design) as well as coding style (e.g., Checkstyle) and documentation (Javadoc comments); usually includes unit testing (formal or informal)

Testing

- *Testing* attempts to ensure that the program will solve the intended problem
  - A program should be thoroughly tested with the goal of finding errors/defects
- *Testing* discovers errors or defects; *debugging* locates and corrects/removes them
- We'll discuss the details of the testing process later in the class notes
- We'll use the JUnit Framework for unit testing
  - Web-CAT will run the usual reference tests as well as determine how well your own testing was carried out

Object-Oriented Design and Programming

- Objects and attributes (fields) are generally nouns, and methods are generally verbs

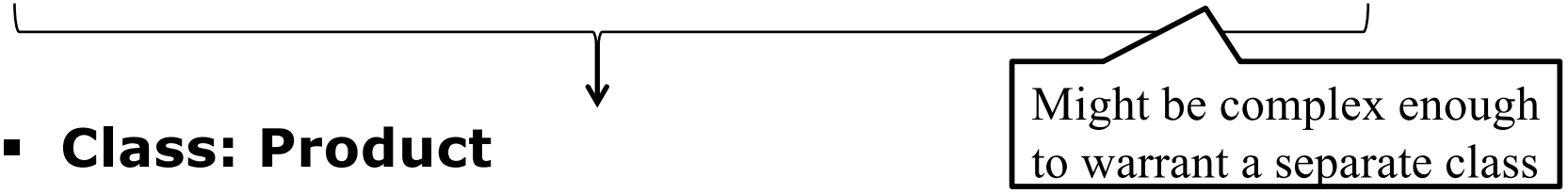
The user must be allowed to specify each product by its primary characteristics, including its name and product number. If the bar code does not match the product, then an error should be generated to the message window and entered into the error log. The summary report of all transactions must be structured as specified in section 7.A.

Identifying Classes (Objects) and Attributes

- Generally, classes that represent objects should be given names that are singular nouns
  - Class: Product
    - Attributes: name, product number, barcode
- We are free to instantiate as many of each objects as needed

Identifying Classes (Objects) and Attributes

- When a class becomes too complex, it often should be decomposed into multiple smaller classes to distribute the responsibilities
  - **Class: Product**
    - Attributes: name, product number, barcode



  - **Class: Product**
    - Attributes: name, product number, barcode
  - **Class: Barcode**
    - Attributes: version, format, data, visual representation, etc

Now back to the details of Java . . .

Static Class Members

- Static methods are invoked using the class name

  Example: `result = Math.sqrt(25)`

- Suppose that you have a static method called getName() in a Person class; what name would Person.getName() retrieve?

- **A static method <u>cannot</u> access the instance variables of its class.**

  - This is because static methods can be invoked before an object of the class is instantiated

Static Methods

```
public class Helper
{
    public static int cube (int num)
    {
        return num * num * num;
    }
}
```

Because it is declared as static, the method can be invoked as

```
value = Helper.cube(5);
```

Static Variables

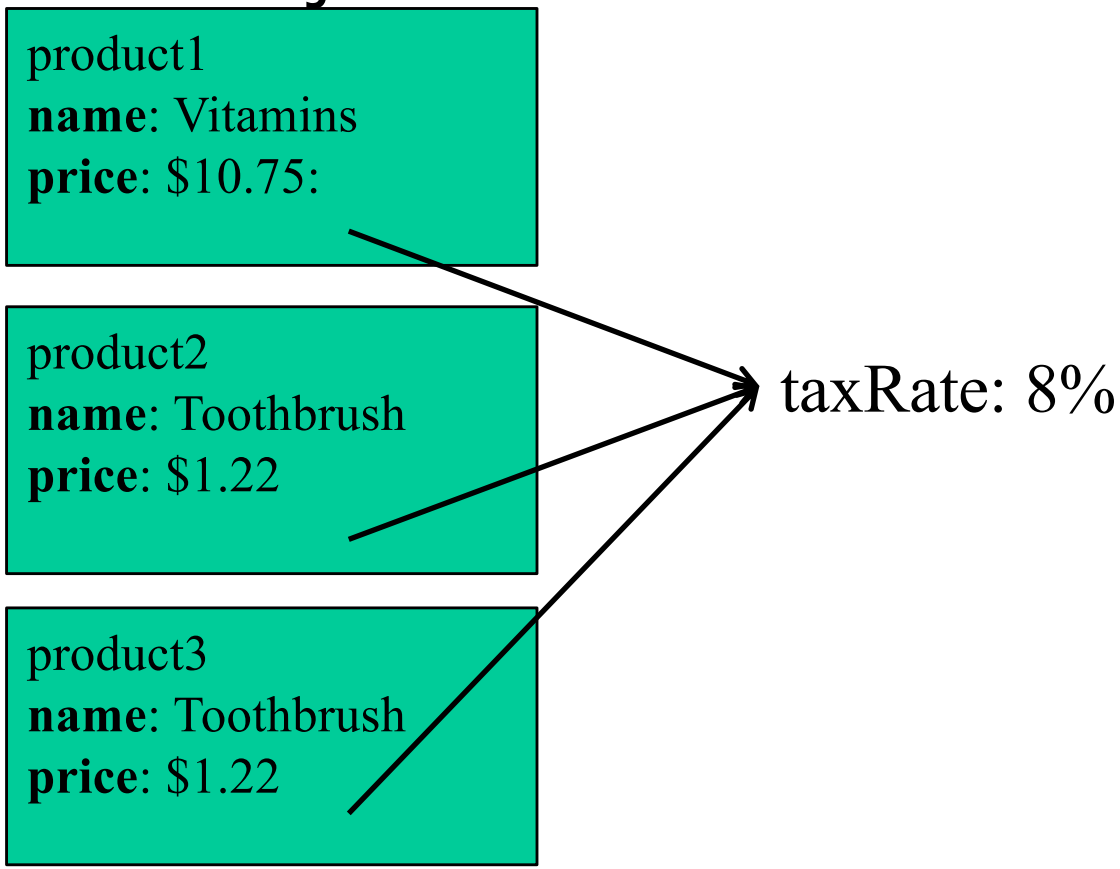
- If a variable is declared as static, it exists at the class level rather than for each instance

```
private static float taxRate;
```

- Space is reserved in memory when the class is first referenced
- A static variable is the same for all instances of the class (i.e., it is shared among the instances) and it is accessible from all instance methods in the class
- If the value of a variable needs to be unique for each object, then it should **not** be static.

Static Variables

- In this case, the taxRate is the same for all objects; changing the static variable changes it for all objects of the class



The *static* Modifier

- Static methods are called *class methods* and static variables are called *class variables*
- Instance methods can reference instance variables and static variables
- Static methods can only access static variables of the class, not instance variables
- Before you declare a field to be a static variable, consider the following:
  - Should the value of the variable be the same for all objects? That is, should it be shared by all objects?
  - If the value of the variable is changed, should it change for all objects?

Q1

Static Class Members

- Static methods and static variables often work together
- The following example keeps track of how many **`Magazine`** objects have been created using a static variable, and makes that information available using a static method
- See Magazine.java and MagazineExample.java
- See StaticExample.java

Class Relationships

- Three of the most common relationships between two classes:
  - Dependency (general): A *uses* B
  - Aggregation: A *has-a* B
  - Inheritance: A *is-a* B (introduced later in the course)
- General dependency: The ProductList class uses the DecimalFormat class for formatting
- Aggregation: The ProductList class **has a** set of Products (aggregation)

Dependency

- General dependency: one class relies on another in some way (e.g., by invoking its methods)
- We don't want a lot of classes with complex dependencies, but we also don't want very large, complex classes either
- Some dependencies occur between objects of the same class. Example, the `concat` method of the `String` class takes a String parameter

```
str3 = str1.concat(str2);
```

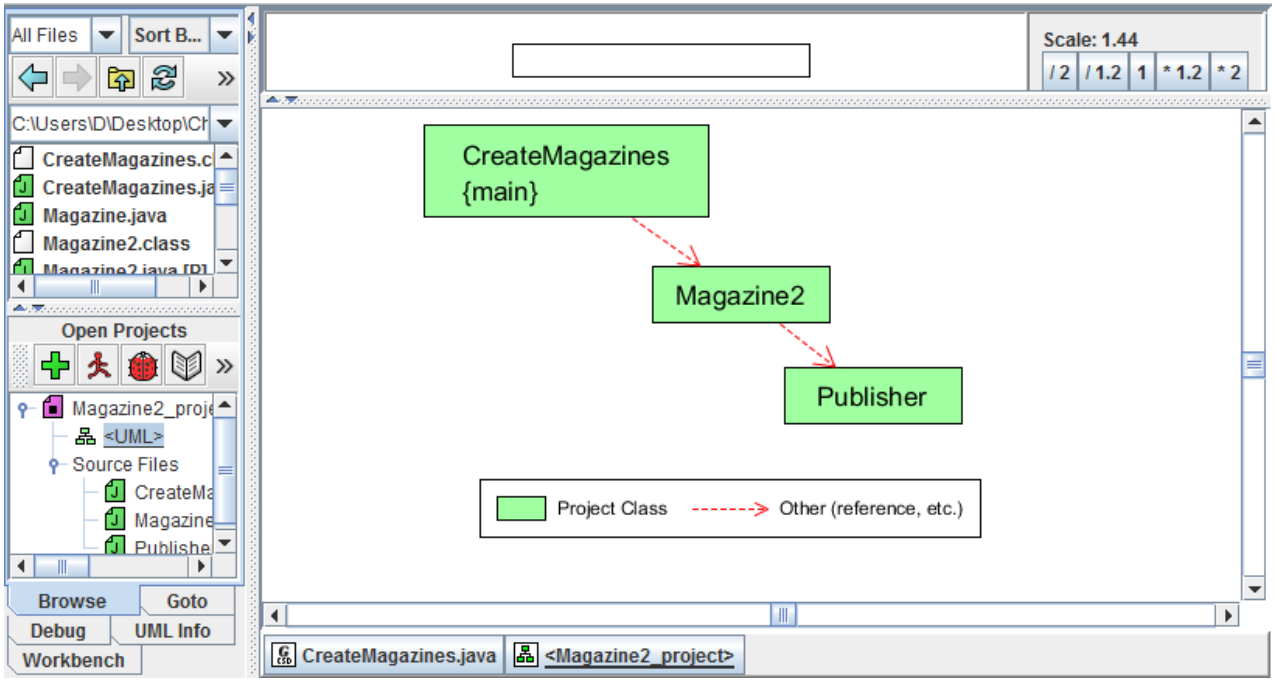
Aggregation

- An *aggregate* is an object that is made up of other objects
- Therefore aggregation is a *has-a* relationship
  - A car *has a* chassis
  - A product *has a* barcode
- An aggregate object contains references to other objects as instance data

Q2 **Q3**

Aggregation Dependencies in UML

- Open Magazine2_project.gpj and compile the project. What type of relationships exist among the three classes?
  - In jGRASP, the general dependency and the aggregate relationship are both shown as red, dashed arrows.



The this reference

- The `this` reference allows an object to refer to itself
- The constructor of the `Magazine2` class could have been written as follows:

```
public Magazine2(String title, int pages,
        String publisher, String city)
{
   this.title = title;
   this.pages = pages;
   this.publisher = new Publisher(publisher, city);
   count++;
}
```

Our coding standard (i.e., our Checkstyle rules) requires unique names for parameters, so the *this* reference will generally be optional.