TDD Boot Camp – How to write maintainable code? – Part 01

Write better object-oriented programming

Workshop Objective:

Identify classes from the requirements and apply the object-oriented principles.

Teams:

All participants will group into three teams and a business requirement will be given to each team.

Workshop:

Each team should do the following activities one by one. All activities are timeboxed so teams should complete each activity on time, the timekeeper will help teams to maintain the time.

1. Understand the business requirements and do the brainstorm - 10 mins
2. Prepare CRC cards - 15 mins
    a. Identify Nouns and verbs from the requirements. Define classes for each noun and create methods for each verb.
    b. Find the responsibility of each class.
        i. What information wish to maintain
        ii. What it does
    c. Identify supporting or hidden classes.
    d. Define the collaborators.
    e. Move the cards around – Group the cards depend on the relationship
3. Create the classes with the right responsibility – 10 mins
    a. Define the class
        i. Define attributes
        ii. Define methods
    b. Apply the abstraction principles
        i. Check whether you have provided a crisp conceptual boundary and unique identity
        ii. Check whether you have mapped the requirement to a domain entity
        iii. Check whether you have ensured cohesion and completeness
        iv. Check whether you have assigned single and meaningful responsibility
        v. Check whether you have removed duplicates
    c. Apply the encapsulation principles
        i. Check whether you have hidden the implementation details
        ii. Check whether you have hidden variation details
4. Define the relationship between collaborators – 10 mins
    a. Define the types of relationship
        i. Association – uses a
        ii. Aggregation – part of
        iii. Composition – has a
        iv. Inheritance – is a

v. Realization – is subsuitable/can do?

5. Identify the re-usability – 10 mins
    a. Identify the context-dependent behavior
        i. Identify the contracts (must have items/ can do items)
        ii. Identify the reusable classes/components
6. Explain your learning and experience of this workshop – 9 mins

Appendix

CRC Card

A Class Responsibility Collaborator (CRC) model (Beck & Cunningham 1989; Wilkinson 1995; Ambler 1995) is a collection of standard index cards that have been divided into three sections, as depicted in the following picture. A class represents a collection of similar objects, responsibility is something that a class knows or does, and a collaborator is another class that a class interacts with to fulfill its responsibilities.

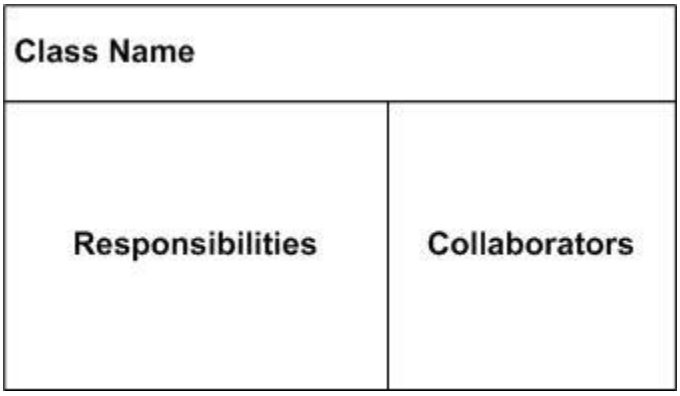


Figure 1 CRC Template

Although CRC cards were originally introduced as a technique for teaching object-oriented concepts, they have also been successfully used as a full-fledged modeling technique. The CRC models are an incredibly effective tool for conceptual modeling as well as for detailed design. CRC cards feature prominently in eXtreme Programming (XP) (Beck 2000) as a design technique.

A class represents a collection of similar objects. An object is a person, place, thing, event, or concept that is relevant to the system at hand. For example, in a university system, classes would represent students, tenured professors, and seminars. The name of the class appears across the top of a CRC card and is typically a singular noun or singular noun phrases, such as *Student, Professor*, and *Seminar*. You use singular names because each class represents a generalized version of a singular object. Although there may be student John O'Brien, you would model the class *Student*. The information about a student describes a single person, not a group of people. Therefore, it makes sense to use the name *Student* and not *Students*. Class

names should also be simple. For example, which name is better: *Student* or *Person* who takes seminars?

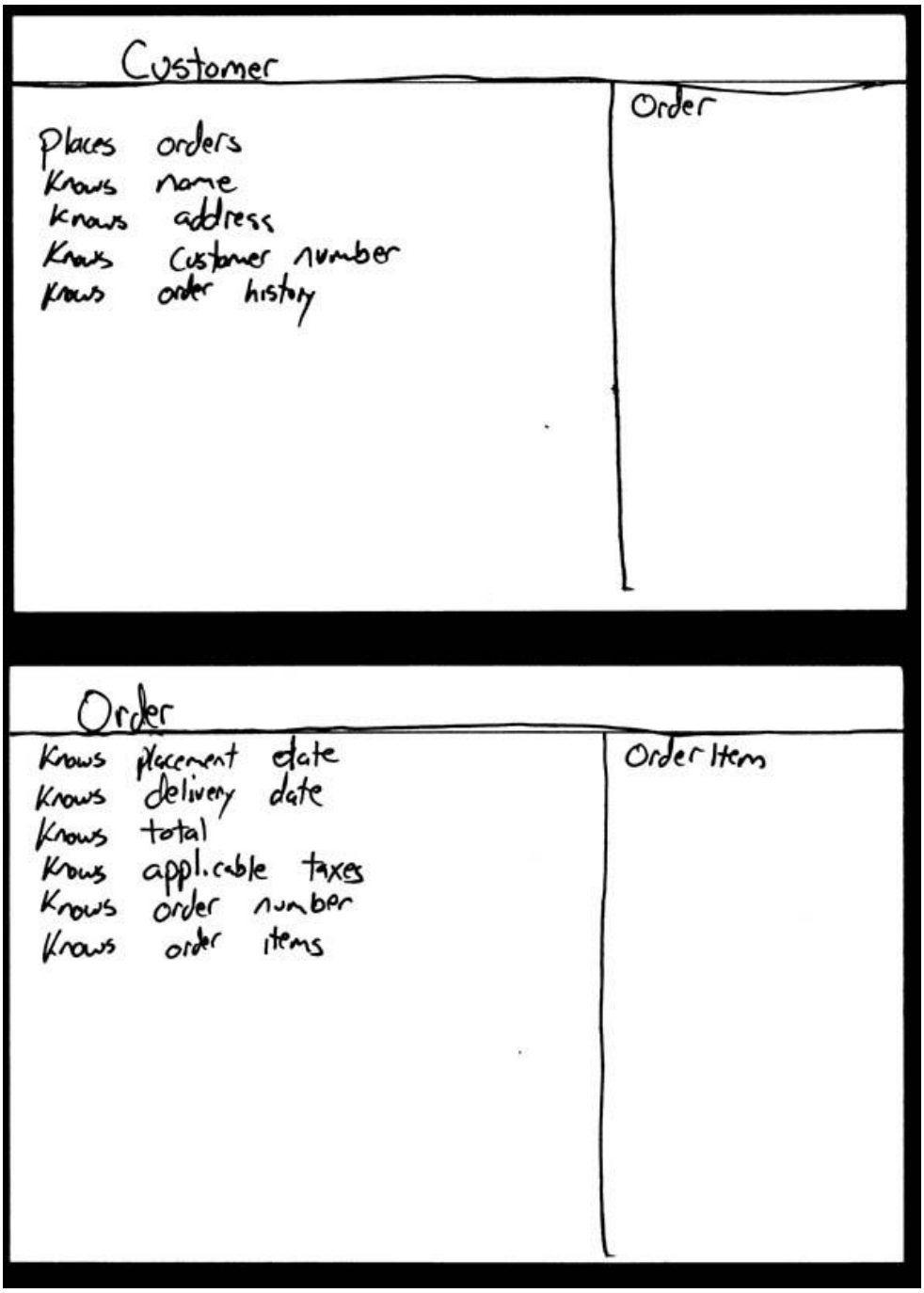
A responsibility is anything that a class knows or does. For example, students have names, addresses, and phone numbers. These are the things a student knows. Students also enroll in seminars, drop seminars, and request transcripts. These are the things a student does. The things a class knows and does constitute its responsibilities. Important: A class is able to change the values of the things it knows, but it is unable to change the values of what other classes know.

Sometimes a class has a responsibility to fulfill, but not have enough information to do it. For example, as you see in Figure 3 students enroll in seminars. To do this, a student needs to know if a spot is available in the seminar and if so, he then needs to be added to the seminar. However, students only have information about themselves (their names and so forth), and not about seminars. What the student needs to do is collaborate/interact with the card labeled Seminar to sign up for a seminar. Therefore, *Seminar* is included in the list of collaborators of *Student*.

Student	
Student number **Name** **Address** **Phone number** **Enroll in a seminar** **Drop a seminar** **Request transcripts**	**Seminar**

Figure 2 Student CRC Card

Another example:



How do you create CRC models?

Do the following steps iteratively,

- **Find classes**. Finding classes is fundamentally an analysis task because it deals with identifying the building blocks for your application. A good rule of thumb is that you should look for the three-to-five main classes right away, such as *Student*, *Seminar*, and *Professor* in the following image. You can include UI classes such

as *Transcript* and *Student Schedule*, both are reports, although others will stick to just entity classes.

- **Find responsibilities**. You should ask yourself what a class does as well as what information you wish to maintain about it. You will often identify a responsibility for a class to fulfill a collaboration with another class.
- **Define collaborators**. A class often does not have sufficient information to fulfill its responsibilities. Therefore, it must collaborate (work) with other classes to get the job done. Collaboration will be in one of two forms: a request for information or a request to perform a task. To identify the collaborators of a class for each responsibility ask yourself "does the class have the ability to fulfill this responsibility?". If not then look for a class that either has the ability to fulfill the missing functionality or the class which should fulfill it. In doing so you'll often discover the need for new responsibilities in other classes and maybe even the need for a new class or two.
- **Move the cards around**. To improve everyone's understanding of the system, the cards should be placed on the table in an intelligent manner. Two cards that collaborate with one another should be placed close together on the table, whereas two cards that don't collaborate should be placed far apart. Furthermore, the more two cards collaborate, the closer they should be on the desk. By having cards that collaborate with one another close together, it's easier to understand the relationships between classes.

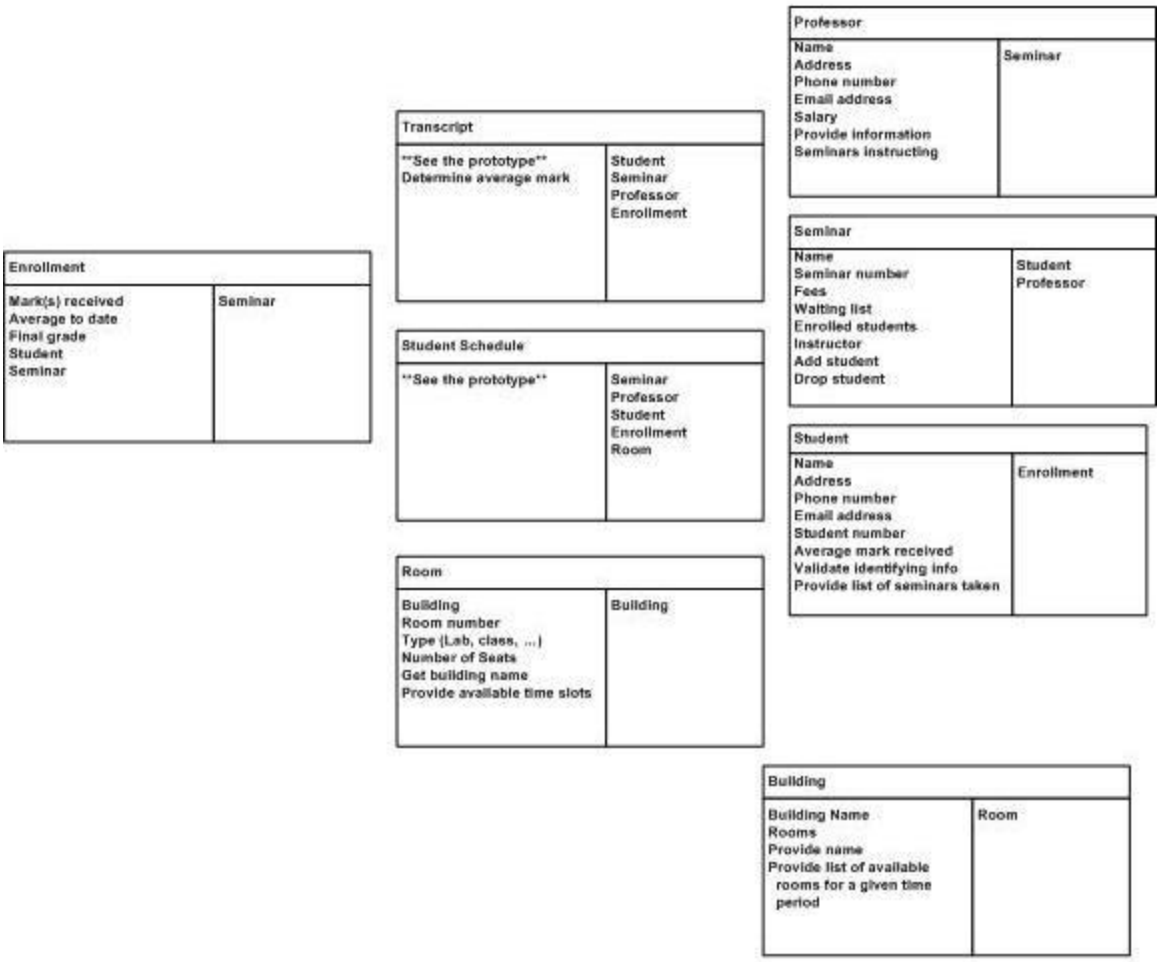
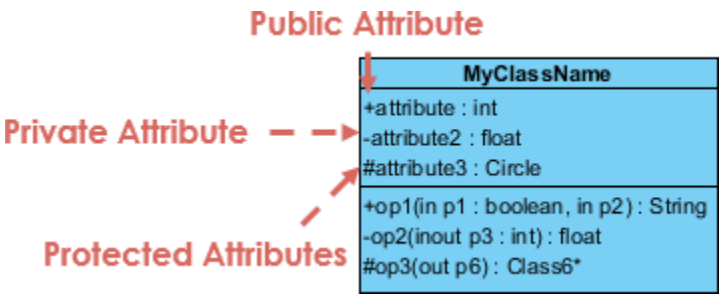


Figure 3 CRC Model

Reference link: http://www.agilemodeling.com/artifacts/crcModel.htm

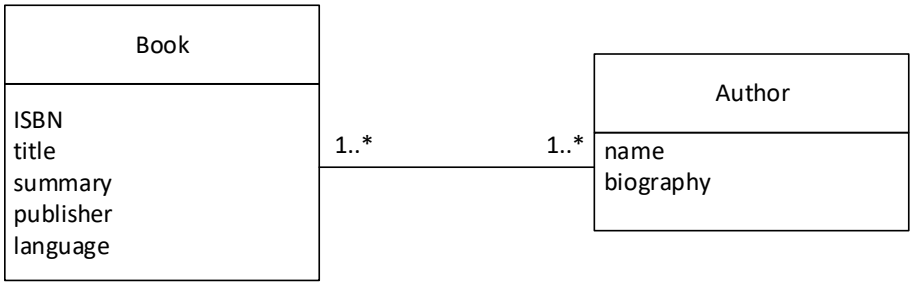
UML(Unified Modeling Language)

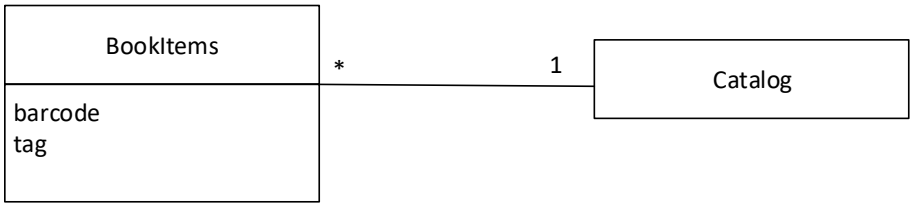
Class



Association

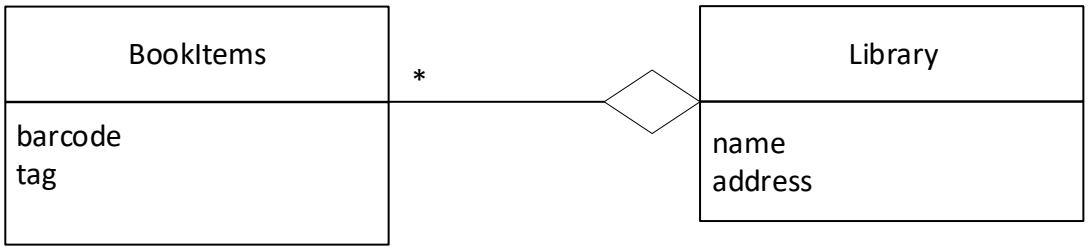
Associations represent static relationships between classes.





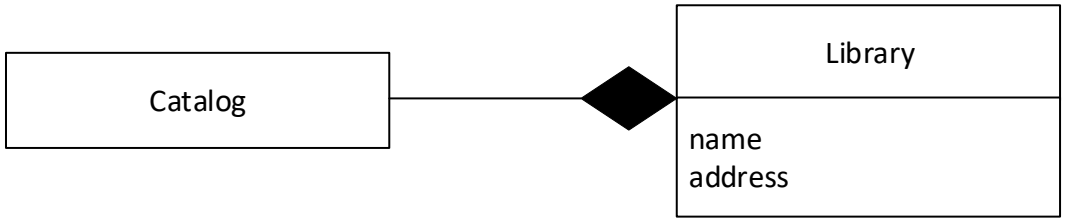
Aggregation

A special type of association. It represents a "part of" relationship.



Composition

A special type of aggregation where parts are destroyed when the whole is destroyed.

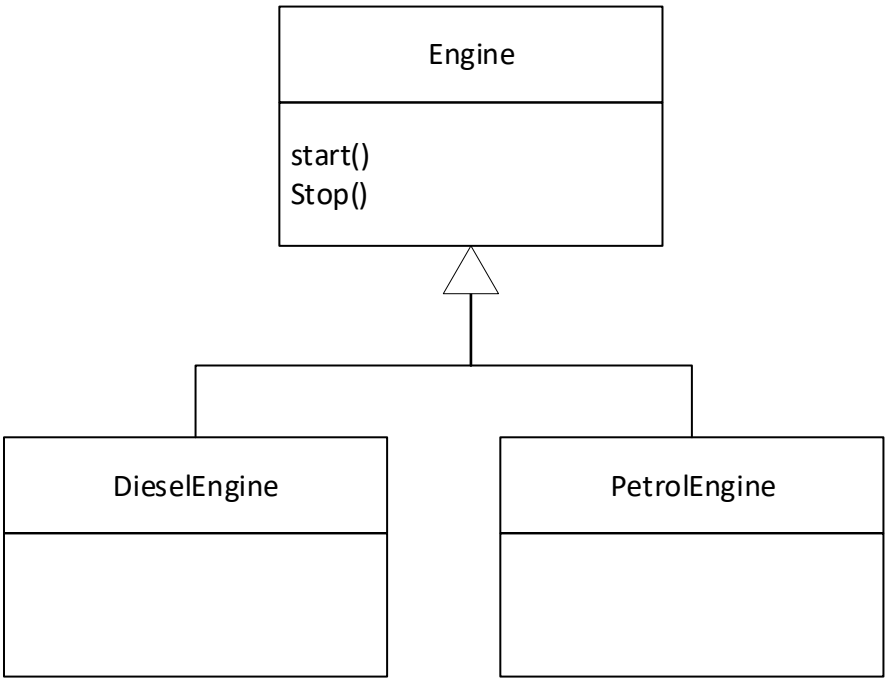


Cardinality

Indicator	Meaning
0..1	Zero or one
1	One only
0..*	0 or more
1..* *	1 or more
n	Only n (where n > 1)
0..n	Zero to n (where n >1)
1..n	One to n (where n > 1)

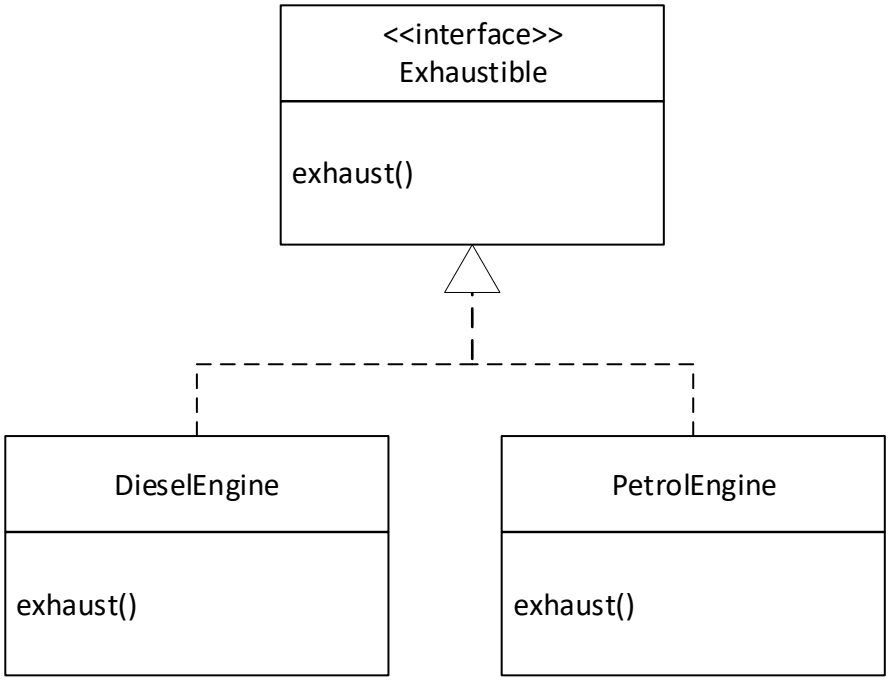
Inheritance

Generalization is another name for inheritance or an "is a" relationship. It refers to a relationship between two classes where one class is a specialized version of another



Realization

Realization is a relationship between the blueprint class and the object containing its respective implementation level details.



Reference

https://www.smartdraw.com/class-diagram/

https://www.lucidchart.com/pages/uml-class-diagram