Designing Classes

Goals

© Petrea Alexandra/Stockphoto.

- To learn how to discover new classes and methods
- To use CRC cards for class discovery
- To identify inheritance, aggregation, and dependency relationships between classes
- To describe class relationships using UML class diagrams

Discovering Classes

- When designing a program, you work from a requirements specification
  - The designer's task is to discover structures that make it possible to implement the requirements
- To discover classes, look for nouns in the problem description.
- Find methods by looking for verbs in the task description.

Example: Invoice

INVOICE

Sam's Small Appliances
100 Main Street
Anytown, CA 98765

Item	Qty	Price	Total
Toaster	3	$29.95	$89.85
Hair Dryer	1	$24.95	$24.95
Car Vacuum	2	$19.99	$39.98

AMOUNT DUE: **$154.78**

Figure 1 An Invoice

Example: Invoice

- Classes that come to mind:
  - `Invoice`
  - `LineItem`
  - `Customer`
- Good idea to keep a list of candidate classes.
- Brainstorm: put all ideas for classes onto the list.
- Cross not useful ones later.
- Concepts from the problem domain are good candidates for classes.
- Not all classes can be discovered from the program requirements:
  - Most programs need tactical classes

The CRC Card Method

© Oleg Prikhodko/iStockphoto.

In a class scheduling system, potential classes from the problem domain include `Class`, `LectureHall`, `Instructor`, and `Student`.

The CRC Card Method

- After you have a set of classes
  - Define the behavior (methods) of each class
- Look for verbs in the task description
  - Match the verbs to the appropriate objects
- The invoice program needs to compute the amount due
  - Which class is responsible for this method?
    - `Invoice` class

The CRC Card Method

- To find the class responsibilities, use the CRC card method.
- A CRC card describes a class, its responsibilities, and its collaborating classes.
  - CRC - stands for "classes", "responsibilities", "collaborators"
- Use an index card for each class.
- Pick the class that should be responsible for each method (verb).
- Write the responsibility onto the class card.
- Indicate what other classes are needed to fulfill responsibility (collaborators).

The CRC Card Method

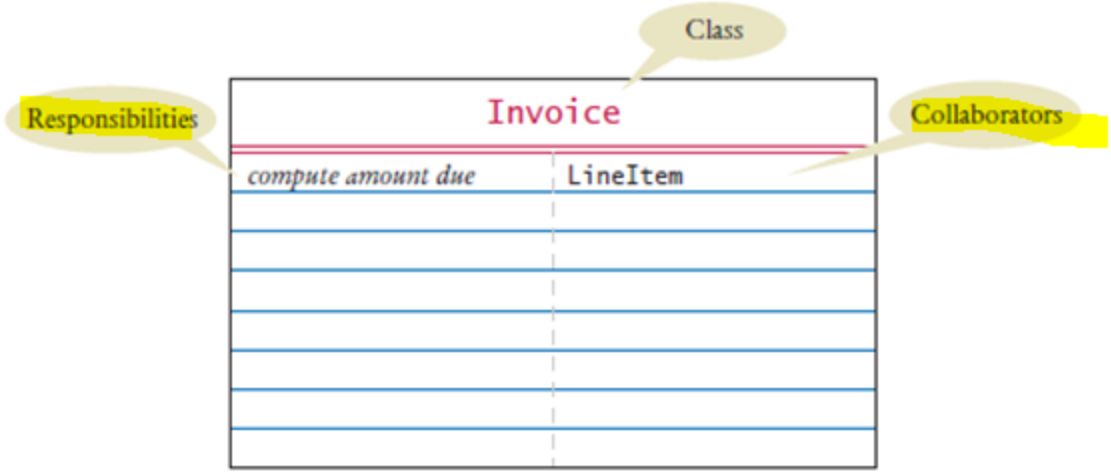


Figure 2 A CRC Card

Relationships Between Classes

The most common types of relationships:

- Dependency
- Aggregation
- Inheritance

Dependency

- A class depends on another class if it uses objects of that class. *The "knows about" relationship.*
- Example: `CashRegister` depends on `Coin`

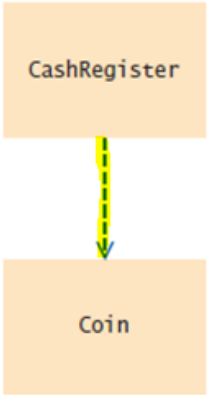


Figure 3 Dependency Relationship Between the `CashRegister` and `Coin` Classes

Dependency

- It is a good practice to minimize the coupling (i.e., dependency) between classes.

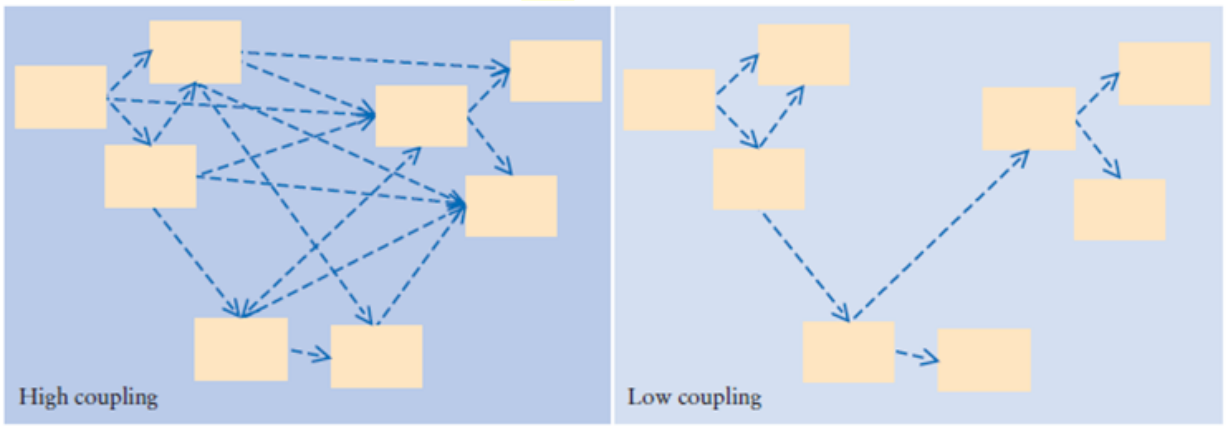


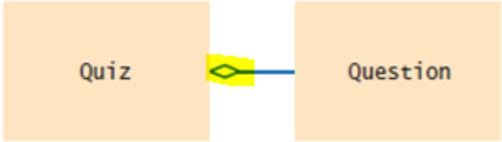
Figure 4 High and Low Coupling Between Classes

- When a class changes, coupled classes may also need updating.

Aggregation

- A class aggregates another if its objects contain objects of the other class.
  - *Has-a* relationship
- Example: a `Quiz` class aggregates a `Question` class.
- The UML for aggregation:

Figure 5
Class Diagram
Showing Aggregation



- Aggregation is a stronger form of dependency.
- Use aggregation to remember another object between method calls.

Aggregation

© bojan fatur/iStockphoto.

A car has a motor and tires. In object-oriented design, this "has-a" relationship is called aggregation.

Inheritance

- Inheritance is a relationship between a more general class (the superclass) and a more specialized class (the subclass).
  - The "is-a" relationship.
  - Example: Every truck is a vehicle.
- Inheritance is sometimes inappropriately used when the has-a relationship would be more appropriate.

Inheritance

- Every car is a vehicle. (Inheritance)
- Every car has a tire (or four). (Aggregation)

```
class Car extends Vehicle
{
   private Tire[] tires;
   . . .
}
```

- Aggregation denotes that objects of one class contain references to objects of another class.

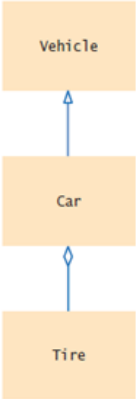


Figure 6 UML Notation for Inheritance and Aggregation

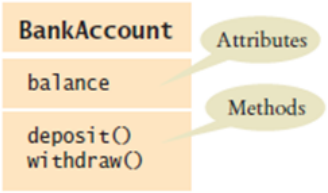
UML Relationship Symbols

Relationship	Symbol	Line Style	Arrow Tip
Inheritance	——▷	Solid	Triangle
Interface Implementation	- - - - ▷	Dotted	Triangle
Aggregation	◇——	Solid	Diamond
Dependency	- - - - >	Dotted	Open

Why should coupling be minimized between classes?

Answer: If a class doesn't depend on another, it is not affected by interface changes in the other class.

Attributes and Methods in UML Diagrams



Multiplicities

- any number (zero or more): *
- one or more: 1..*
- zero or one: 0..1
- exactly one: 1

An Aggregation Relationship with Multiplicities

Aggregation and Association, and Composition

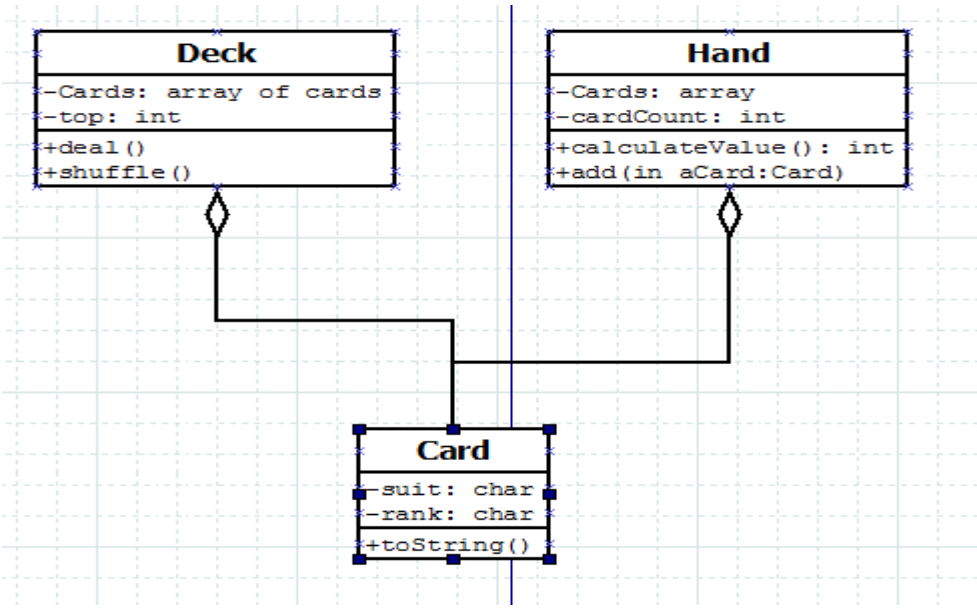
- Association: More general relationship between classes.
- Use early in the design phase.
- A class is associated with another if you can navigate from objects of one class to objects of the other.
- Given a `Bank` object, you can navigate to `Customer` objects.

An Association Relationship

Aggregation and Association, and Composition

- Composition: one of the classes can not exist without the other.

A Composition Relationship



Both Composition and Aggregation are Associations.
Composition IS-A Association.
Aggregation IS-A Association.

Composition is a strong association.
Aggregation is a weak association.