Object Oriented Analysis & Design

Lecture # 3

Department of Computer Science and Technology
University of Bedfordshire

Written by David Goodwin,
based on the book *Applying UML and Patterns (3rd ed.)*
by *C. Larman* (2005).

Modelling and Simulation, 2012

Outline

Introduction to Object Orientation

Objects

Class

Attributes

Operations

Associations

Aggregation

Composition

Inheritance

Dependency

Multiplicity

Polymorphism

Encapsulation

Interface

Package

Introduction to Object Orientation

Object Orientation

- Knowing and object-oriented language (such as Java) is a necessary but insufficient step to create object systems.
- UML is just a diagramming tool; it's full use isn't realised without object oriented design.

OOA/D

- **Object-oriented analysis (OOA):**
  - process of analysing a task (also known as a problem domain)
  - finding and describing objects
  - typical:
    - set of use cases
    - one or more class diagrams
    - a number of interaction diagrams
- **Object-oriented design (OOD):**
  - defining software objects and how they collaborate to fulfill the requirements.
  - constraints to conceptual model produced in object-oriented analysis
  - Concepts in the analysis model are mapped onto implementation classes and interfaces resulting in a model of the solution domain.

Objects

Definition of an object

- **Object:**
  - Objects directly relate to real-world 'entities'.
  - An object has identity, state & behaviour.
    - Identity: the property of an object that distinguishes it from other objects
    - State: describes the data stored in the object
    - Behaviour: describes the methods in the object's interface by which the object can be used
  - The state of an object is one of the possible conditions in which an object may exist.
  - The state of an object is represented by the values of its properties (attributes).

Definition of an object

- Real-world objects share three characteristics:
  - **Identity:**
    - Dog
    - Bicycle
  - **State:**
    - Dogs have state (name, color, breed, hungry)
    - Bicycles also have state (current gear, current pedal cadence, current speed)
  - **Behaviour:**
    - Dogs have behaviour (barking, fetching, wagging tail)
    - Bicycles also have behaviour (changing gear, changing pedal cadence, applying brakes)

Identifying & using objects' states

- "What possible states can this object be in?"
- "What possible behaviour can this object perform?"
- Some objects will contain other objects.
- By attributing state (current speed, current pedal cadence, and current gear) and providing methods for changing that state, the object remains in control of how the outside world is allowed to use it.
  - if the bicycle only has 6 gears, a method to change gears could reject any value that is less than 1 or greater than 6.

Advantages of using objects

- Bundling code into individual software objects provides a number of benefits, including:
  - Modularity:
    - The source code for an object can be written and maintained independently of the source code for other objects.
  - Information-hiding:
    - By interacting only with an object's methods, the details of its internal implementation remain hidden from the outside world.
  - Code re-use:
    - If an object already exists, you can use that object in your program.
  - Pluggability and debugging ease:
    - If a particular object turns out to be problematic, you can simply remove it from your application and plug in a different object as its replacement.

Class

What is a Class?

- Many individual objects can be of the same kind:
  - There may be thousands of other bicycles in existence, all of the same make and model.
  - Each bicycle was built from the same set of blueprints and therefore contains the same components.
- In object-oriented terms, we say that your bicycle is an instance of the class of objects known as bicycles.
- A class is the blueprint from which individual objects are created.

Class example in Java

- The fields cadence, speed, and gear represent the object's state, and the methods (changeCadence, changeGear, speedUp etc.) define its interaction.

```
class Bicycle {

    int cadence = 0;
    int speed = 0;
    int gear = 1;

    void changeCadence(int newValue) {
         cadence = newValue;
    }

    void changeGear(int newValue) {
         gear = newValue;
    }

    void speedUp(int increment) {
         speed = speed + increment;
    }

    void applyBrakes(int decrement) {
         speed = speed - decrement;
    }

    void printStates() {
         System.out.println("cadence:" +
             cadence + " speed:" +
             speed + " gear:" + gear);
    }
}
```

Class example in Java

- Not a complete application; it's just the blueprint for bicycles that might be used in an application. The responsibility of creating and using new Bicycle objects belongs to some other class in your application.

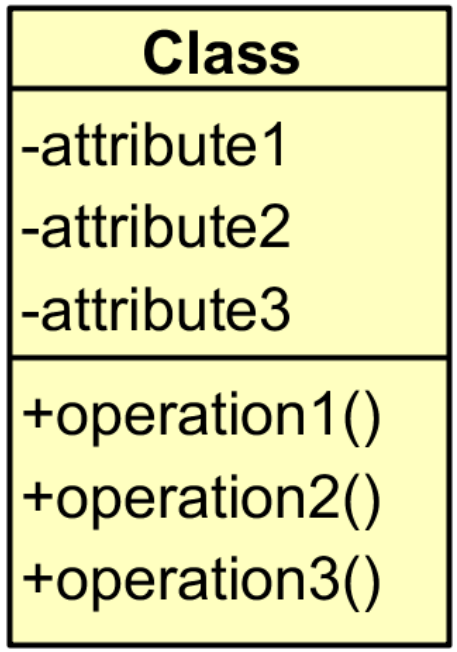
```
class BicycleDemo {
    public static void main(String[] args) {

        // Create two different
        // Bicycle objects
        Bicycle bike1 = new Bicycle();
        Bicycle bike2 = new Bicycle();

        // Invoke methods on
        // those objects
        bike1.changeCadence(50);
        bike1.speedUp(10);
        bike1.changeGear(2);
        bike1.printStates();

        bike2.changeCadence(50);
        bike2.speedUp(10);
        bike2.changeGear(2);
        bike2.changeCadence(40);
        bike2.speedUp(10);
        bike2.changeGear(3);
        bike2.printStates();
    }
}
```

Class notation

Name appears in the top division of the class box

Attribute appears in the middle division of the class box

Operation appears in the bottom division of the class box



What is an attribute?

- An element of data that helps to describe an object.
- Attribute types may be restricted by a programming language.
- A class that includes many attributes may be decomposed into other classes.
- Complex attribute lists in a class may be defined elsewhere.

What is an operation?

- It helps to specify the behaviour of a class (object).
- Can use the syntax of a programming language.
- Specify the visibility of each operation:
  - public (+);
  - private (-);
  - protected (#).
- Consider the responsibilities of each class:
  - Responsibilities often imply more than one operation.
  - A method is the implementation of an operation.

Class Associations

- Classes (objects) must interact with each other so that a piece of software can do something.
- How do classes/objects therefore interact?
- How are associations modelled?

Class Associations

- An association represents a family of links.
- Binary associations (with two ends) are normally represented as a line.
- An association can be named, and the ends of an association can be adorned with role names, ownership indicators, multiplicity, visibility, and other properties.
- There are four different types of association:
  - bi-directional, uni-directional, Aggregation (includes Composition aggregation) and Reflexive.
- Bi-directional and uni-directional associations are the most common ones.
  - For instance, a flight class is associated with a plane class bi-directionally. Association represents the static relationship shared among the objects of two classes. Example: "department offers courses", is an association relation

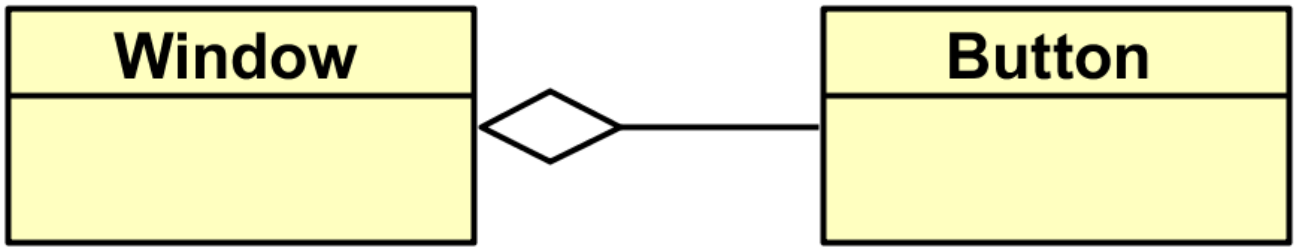
Aggregation

- Whole/Part relationship.
- 'Smaller' classes are parts of 'larger' classes.
- Objects belonging to an aggregate can belong to more than one class.
- Each class that makes up an aggregation relationship is a class in its own right.

Aggregation

- Aggregation is a variant of the "has a" or association relationship; aggregation is more specific than association.
- It is an association that represents a part-whole or part-of relationship.
- As a type of association, an aggregation can be named and have the same adornments that an association can.
- However, an aggregation may not involve more than two classes.
- Aggregation can occur when a class is a collection or container of other classes, but where the contained classes do not have a strong life cycle dependency on the container-essentially,
- if the container is destroyed, its contents are not.

Aggregation Notation



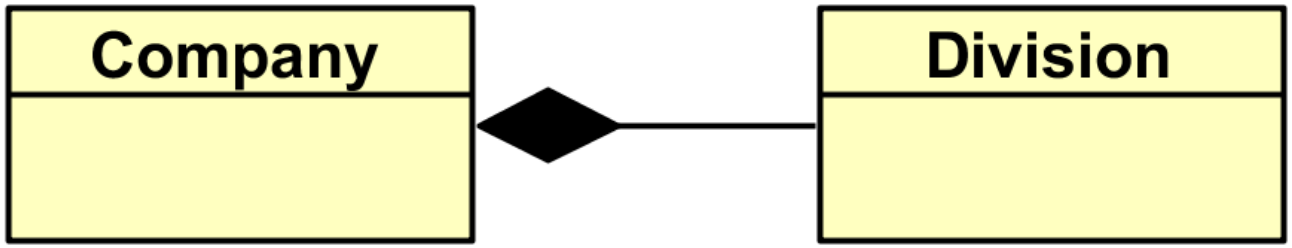
Composition

- A stronger form of aggregation.
- Objects may only be part of one composite at a time.
- The composite object has sole responsibility for the creation and destruction of all of its parts.

Composition

- Composition is a stronger variant of the "owns a" or association relationship; composition is more specific than aggregation.
- Composition usually has a strong life cycle dependency between instances of the container class and instances of the contained class(es):
  - If the container is destroyed, normally every instance that it contains is destroyed as well.

Composition Notation



Aggregation vs. Composition

- When attempting to represent real-world whole-part relationships,
  - e.g., an engine is a part of a car, the composition relationship is most appropriate.
- However, when representing a software or database relationship,
  - e.g., car model engine ENG01 is part of a car model CM01, an aggregation relationship is best, as the engine, ENG01 may be also part of a different car model.
- The whole of a composition must have a multiplicity of 0..1 or 1, indicating that a part must belong to only one whole; the part may have any multiplicity.
  - For example, consider University and Department classes. A department belongs to only one university, so University has multiplicity 1 in the relationship. A university can (and will likely) have multiple departments, so Department has multiplicity 1..*.

Inheritance

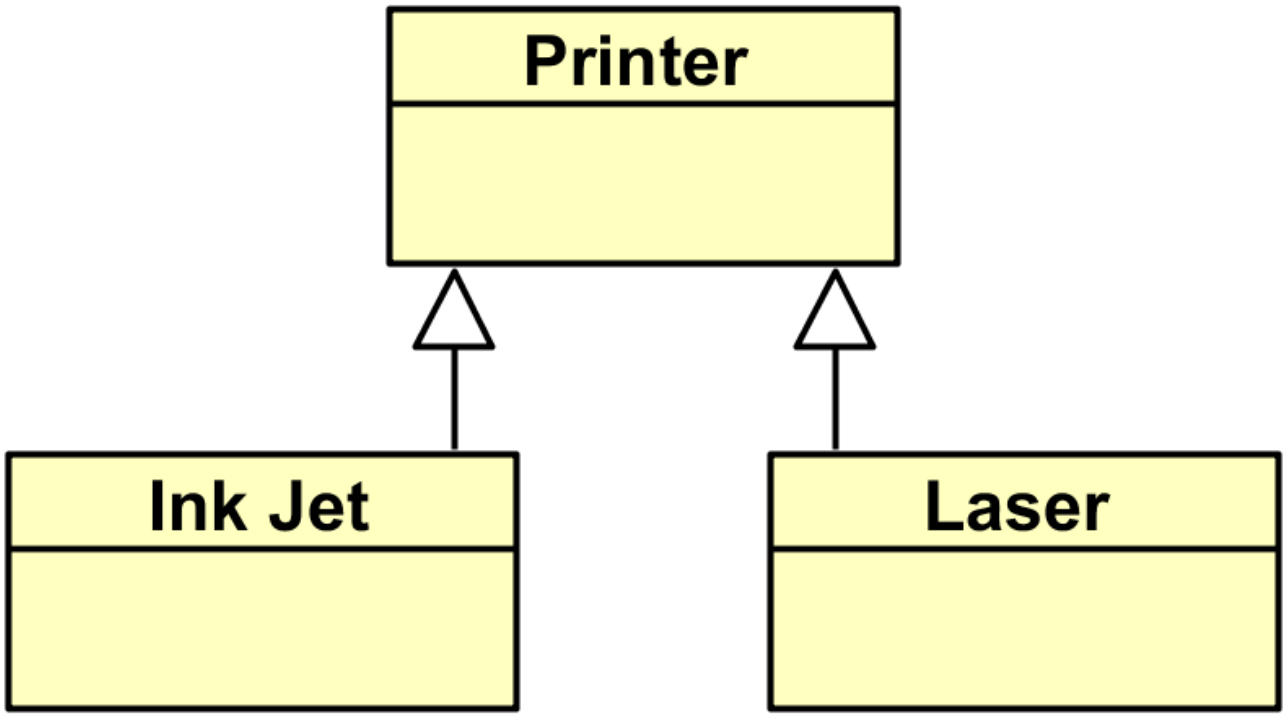
Inheritance (Generalisation)

- Inheritance is a code reuse mechanism to build new objects out of old ones.
- Inheritance defines a relationship among classes where one or more classes share the behaviour and/or attributes of another class.
- For example, a class called ink-jet printer inherits all the behaviour and attributes of the class computer printer.
- Inheritance also enables 'polymorphism'.
- The relationship superclass and subclass is used.
- Each subclass is a specialised version of its superclass.
- Within UML, generalisation is the preferred term.

Inheritance (Generalisation)

- Different kinds of objects often have a certain amount in common with each other.
  - Mountain bikes, road bikes, and tandem bikes, for example, all share the characteristics of bicycles (current speed, current pedal cadence, current gear).
  - Additional features that make them different: tandem bicycles have two seats and two sets of handlebars; road bikes have drop handlebars; some mountain bikes have an additional chain ring, giving them a lower gear ratio.
- Object-oriented programming allows classes to inherit commonly used state and behavior from other classes.
  - In this example, Bicycle now becomes the superclass of MountainBike, RoadBike, and TandemBike.

Generalisation Notation



Dependency

Dependency

- Weaker form of relationship which indicates that one class depends on another.
- One class depends on another if the latter is a parameter variable or local variable of a method of the former.
- This is different from an association, where an attribute of the former is an instance of the latter.

Multiplicity

Multiplicity

- The association relationship indicates that (at least) one of the two related classes makes reference to the other.
- The UML representation of an association is a line with an optional arrowhead indicating the role of the object(s) in the relationship, and an optional notation at each end indicating the multiplicity of instances of that entity (the number of objects that participate in the association).
  - 0..1 No instances, or one instance (optional)
  - 1 Exactly one instance
  - 0..* or * Zero or more instances
  - 1..* One or more instances (at least one)

Polymorphism

Polymorphism

- Polymorphism means having many forms.
- Invokes many different kinds of behaviour.
- It requires an inheritance hierarchy to be present.
- Each class in an inheritance hierarchy is dependent upon other classes, and are therefore not classes in their own right.
- objects of various types define a common interface of operations for users
- the "+" operator which allows similar or polymorphic treatment of numbers (addition), strings (concatenation), and lists (attachment).

Encapsulation

Encapsulation

- Encapsulation is the process of hiding the implementation details of an object.
- The only access to manipulate the object data is through its interface.
- It protects an object's internal state from being corrupted by other objects.
- Also, other objects are protected from changes in the object implementation.
- Encapsulation allows objects to be viewed as 'black boxes'.
- Communication is achieved through an 'interface'.

Interface

Interface

- Object's interface with the outside world;
  - the buttons on the front of your television set, for example, are the interface between you and the electrical wiring on the other side of its plastic casing. You press the "power" button to turn the television on and off.
- In its most common form, an interface is a group of related methods with empty bodies.
- Implementing an interface allows a class to become more formal about the behaviour.
- Interfaces form a contract between the class and the outside world.

Package

Package

- A package is a namespace that organises a set of related classes and interfaces.
  - Conceptually you can think of packages as being similar to different folders on your computer.
  - You might keep HTML pages in one folder, images in another, and scripts or applications in yet another.
- Because software can be composed of hundreds or thousands of individual classes, it makes sense to keep things organised by placing related classes and interfaces into packages.

Packages in Java

- The Java platform provides an enormous class library (a set of packages) suitable for use in your own applications. This library is known as the "Application Programming Interface", or "API".
  - a String object contains state and behavior for character strings;
  - a File object allows a programmer to easily create, delete, inspect, compare, or modify a file on the filesystem;
  - a Socket object allows for the creation and use of network sockets;
  - various GUI objects control buttons and checkboxes and anything else related to graphical user interfaces.
- There are literally thousands of classes to choose from. This allows you, the programmer, to focus on the design of your particular application, rather than the infrastructure required to make it work.