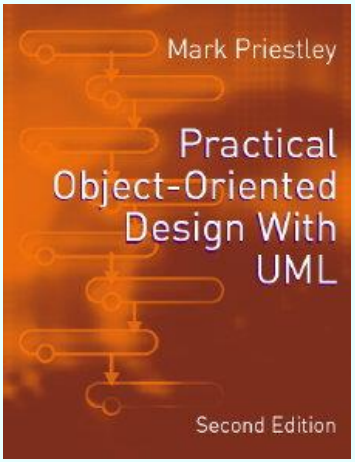
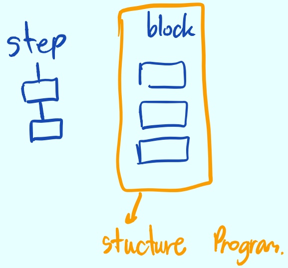
PRACTICAL OBJECT-ORIENTED DESIGN WITH UML 2e



Chapter 2:
Modelling with Objects

เข้าเรื่อง 10.22

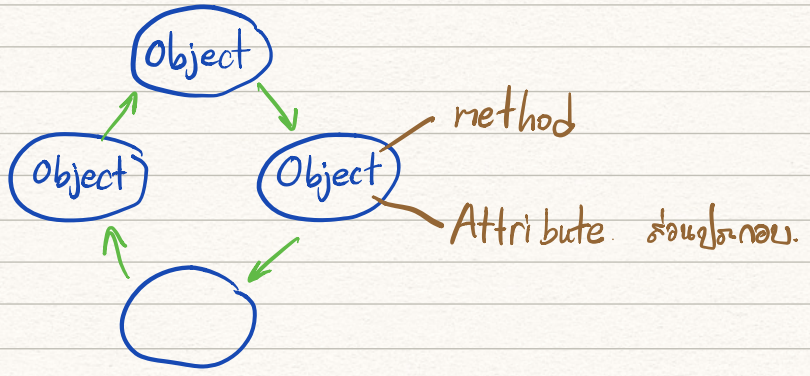
0.00 - 9.25 พูดโดยรวมทั้งหมด



The Object Model

- A general view of program structure shared by UML and object-oriented programming languages like Java and C++
- Computation takes place in *objects* that: (องค์ประกอบสำคัญ)
  - store data (Attribute) and implement behaviour (method)
  - are *linked* together in a network
  - communicate by sending *messages* (ทำให้เกิดการทำงาน)
  - are described by *classes* (เหมือน factory ที่ผลิต Object)

สิ่งรอบ Object ก่อน.
ให้นับทุกๆแต่ละ Object
เกิดการเรียกใช้งานกัน, ทำงานร่วมกัน.



UML and Code

- Because of the shared object model:
  - UML diagrams can easily be implemented
  - object-oriented programs can easily be documented in UML
- Reverse engineering:
  - creating a UML model for existing code
  - often useful in dealing with undocumented legacy systems

17.50

Stock Control Example

ยกตัวอย่าง Warehouse. เก็บชิ้นส่วนไปทำ. พวกนี้ควร,ต้อง เป็น Object.

- Keep track of *parts* in a warehouse and how they are used
  - parts have a part number, name and cost
  - the system must track individual parts
  - parts can be used to form *assemblies*
  - assemblies have a hierarchical structure
- Could be used for a variety of applications
  - e.g. find the cost of the materials in an assembly

18.50

A Simple Part Class

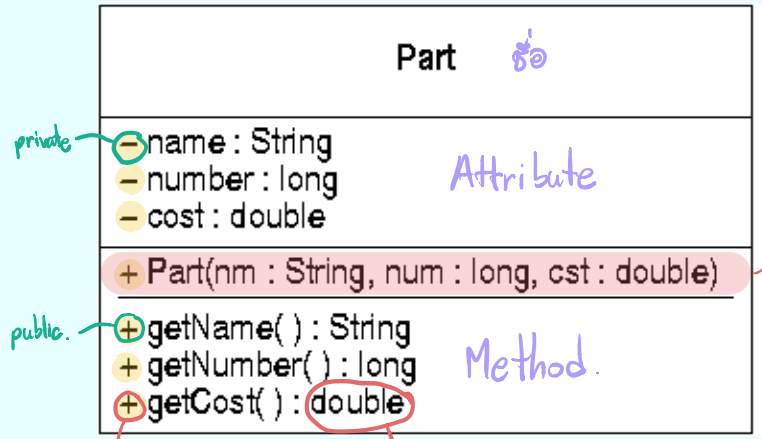
ตัวใหญ่ ถ้าตัวเล็กหมดเป็น Object

```
public class Part {
    private String name ;
    private long number ;
    private double cost ;

    public Part(String nm, long num, double cost) {
      name = nm ;
      // etc
    }
    public String getName( ) { return name; }
    // etc
}
```

Constucture ใช้สร้าง Object

The Part Class in UML

- A class is represented by a single UML icon



UML Class Notation

- Class icons have three compartments
  - the first shows the class *name*
  - the second optionally shows *attributes*
    - derived from its fields (data members)
  - the third optionally shows *operations*
    - derived from its methods (member functions)
    - constructors (and static methods) underlined
- Class *features* = attributes + operations
  - access levels: public (+) and private (-)

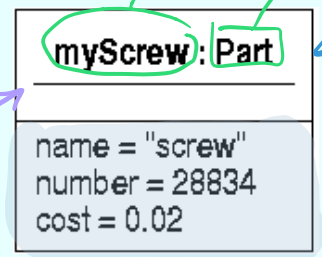
29.42

Objects

- Objects are created at run-time as *instances* of classes:

```
Part myScrew = new Part("screw", 28834, 0.02) ;
```

- Individual objects can be shown in UML:



UML Object Notation

- Object icons have two compartments
  - the first shows the object and class names
    - variable names can be used as object names
    - the object name is optional
    - the class name is preceded by a colon
    - the names are always underlined
  - the second shows the actual data stored in the object, as attribute/value pairs
    - this compartment is usually omitted

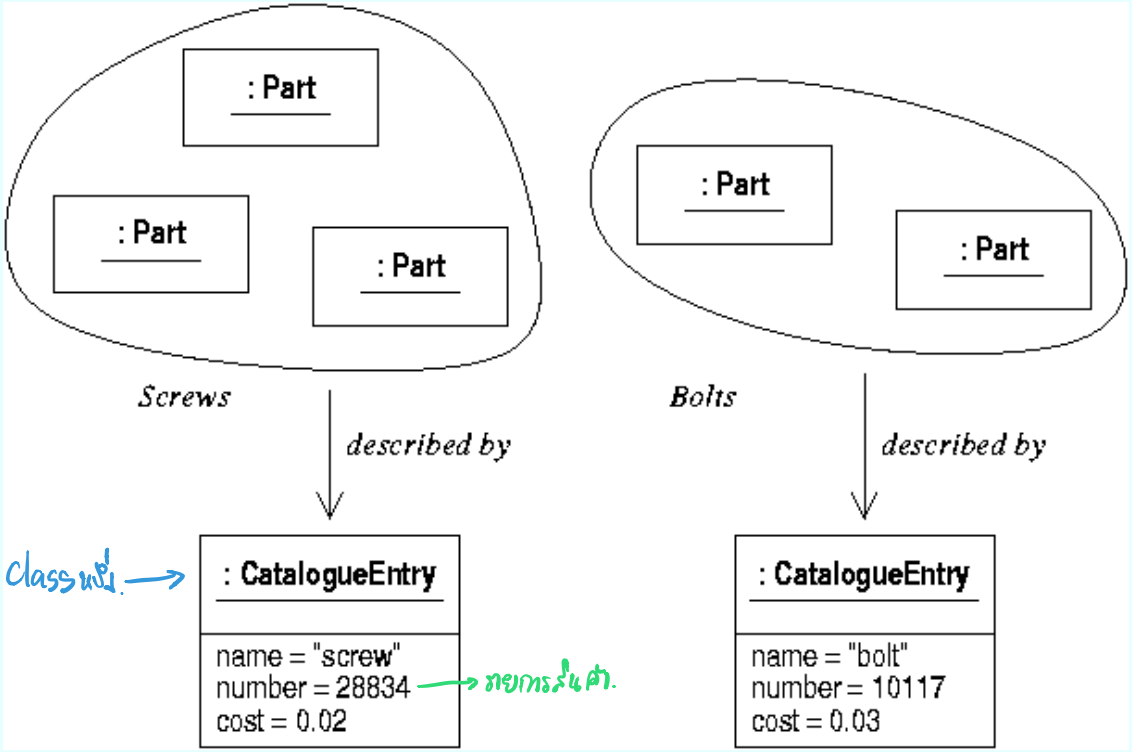
Object Properties

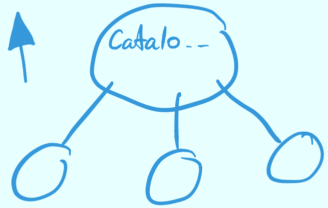
- Objects have:
  - *state*: the data values held in the object
  - *behaviour*: the functionality it contains
  - *identity*: a distinguishing property of an object
    - not an attribute value
    - could be the object's address in memory
  - *names*: useful in modelling to refer to object
- Objects normally *encapsulate* their data

Avoid Data Replication

- All parts of the same kind have the same attribute values (name, number, cost)
  - wasteful of resources
  - hard to maintain, eg if the cost changes
- Better to create a new class for shared data
  - call this a ‘catalogue entry’
  - describes a type of part
  - each individual part described by one entry

Parts and Catalogue Entries





Implementation

- Catalogue entry class defines shared data
- Each part holds a reference to an entry

```
public class Part {
   private CatalogueEntry entry ;
   public Part(CatalogueEntry e) { entry = e; }
}
```

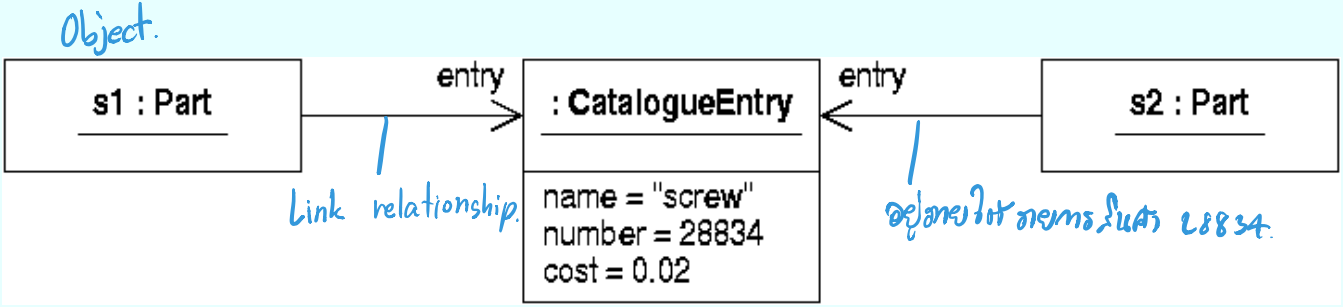
- When parts are created, an entry is given

```
CatalogueEntry screw = new
  CatalogueEntry("screw",28834,0.02);
Part s1 = new Part(screw) ;
Part s2 = new Part(screw) ;
```

40.34

Links

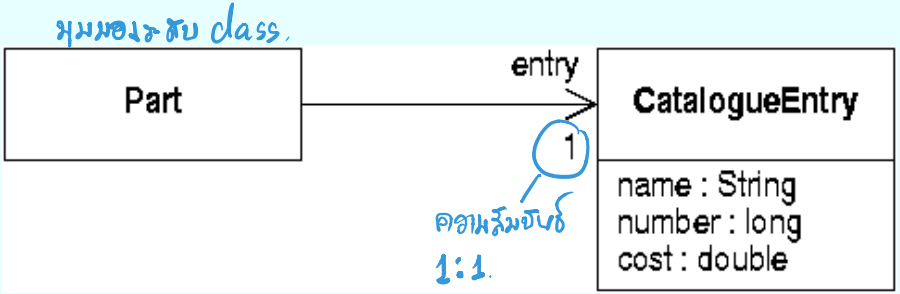
- References can be shown in UML as *links*
  - object name is name of variable
  - field name 'entry' used to label end of link



42.10

Associations

- Data relationships between classes
- Links are *instances* of associations



44.14 ไปหน้า 13

48.50

Properties of Associations

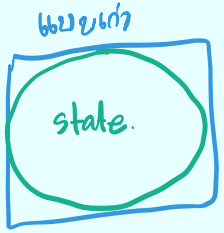
- Associations can have:
  - a *name* (not shown)
  - a *rolename* (ใช้บทบาท) at each end
    - derived from field name in code
  - a *multiplicity* (ระบุจำนวน) at either end
    - this specifies how many links there can be
  - an arrowhead at either end showing *navigability*
    - derived from direction of reference in code

52.02

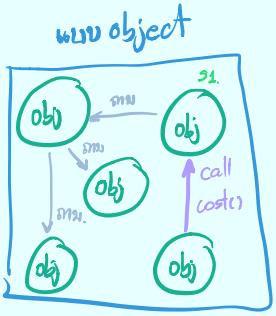
UML Diagram Types

- *Object diagrams* show objects and links
  - this describes the *run-time* properties of a program
- *Class diagrams* show classes and associations
  - this shows the *compile-time* properties of a program
  - i.e. the logical structure of its source code

Object Interaction



- Objects interact by method calls




```
public class Part {
   private CatalogueEntry entry ;
   public double cost( ) { return
     entry.getCost(); }
}
...
Part s1 = new Part(screw) ;
double s1cost = s1.cost() ;
```

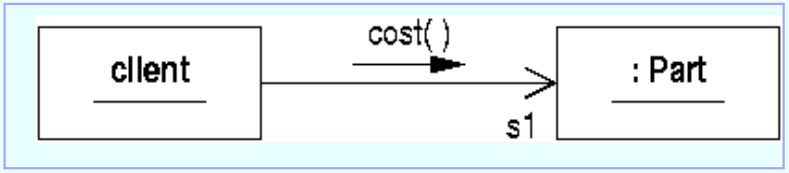
- Functionality is distributed among objects which must communicate

58.49.

Messages

- Object communication is shown in UML as *message passing*
  - a message corresponds to a method call

Collaboration diagram.



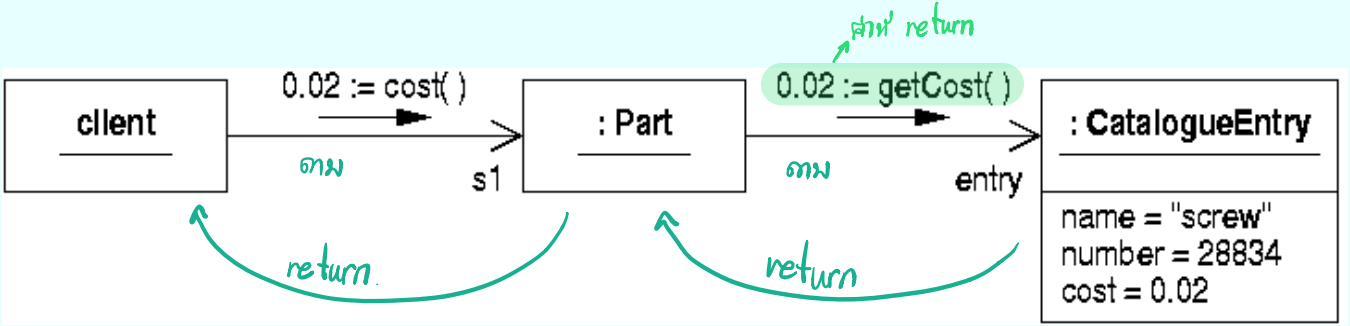
  - the class of the client object is not specified in this diagram

1.0.10

แสดงความเป็น Team work ของ Object

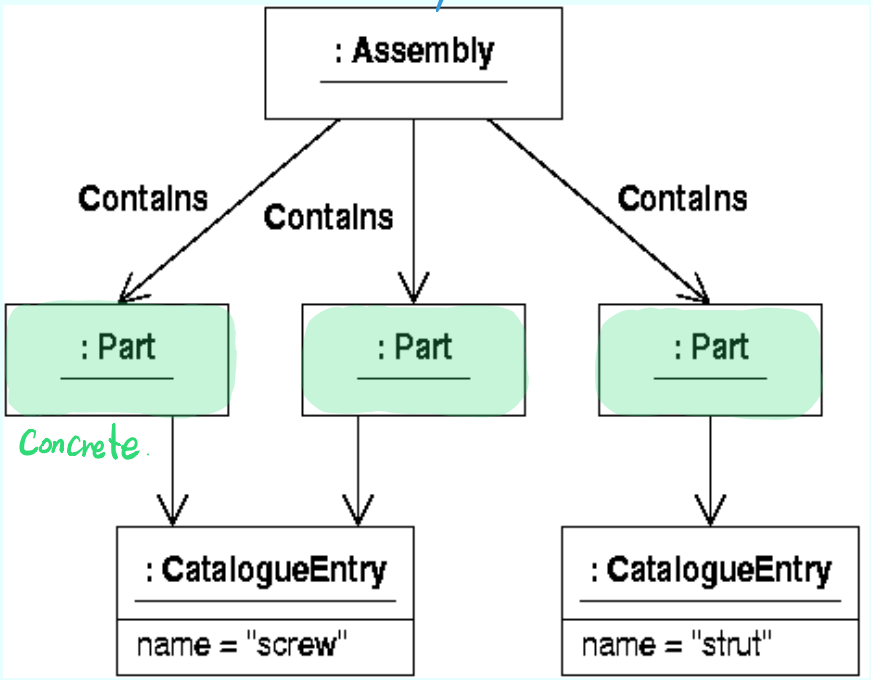
Collaboration Diagrams

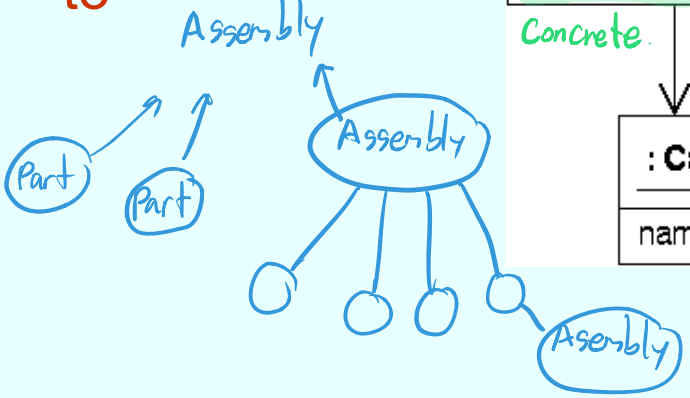
- Object diagrams with messages show *collaborations*
  - multiple messages can be shown on one diagram
  - return values are shown using the ':=' notation



Assemblies

- Assemblies contain parts
  - Links show which assembly a part belongs to





Implementing Assemblies

- Assemblies could be implemented using a data structure to hold references to parts

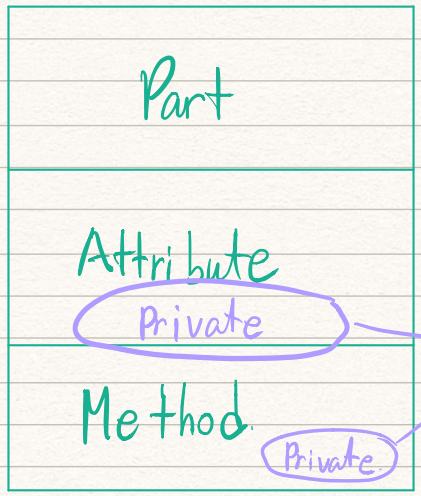
```
public class Assembly {
   private Vector parts = new Vector() ;

   public void add(Part p) {
   parts.addElement(p) ;
   }
}
```

- These links are instances of a new association

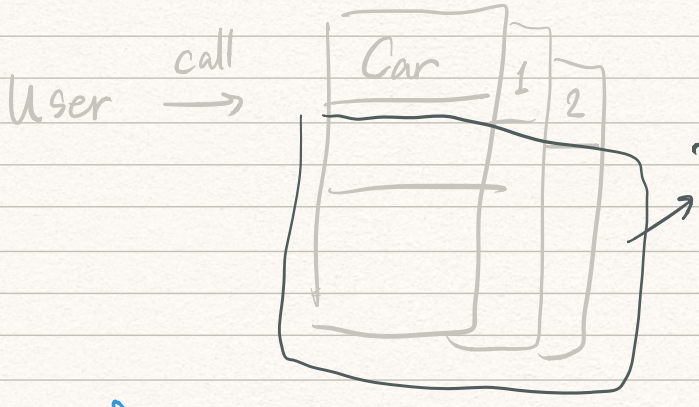
1.3.21

Encapsulation → ของในฐานะของ User

ใช้อะไรใส่ข้าง → Interface.



ป้องใช้ user ไม่ต้องใช้.



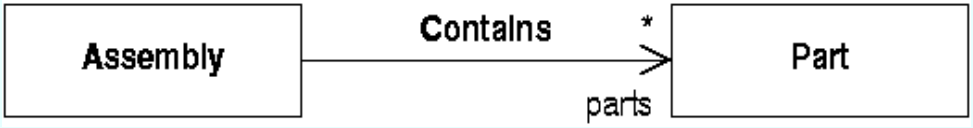
ไม่เปลี่ยนเหมือนเดิม.

การใช้ Attribute โดยตรงอันตรายมาก ต้องใช้ Method get, set แทนกัน.

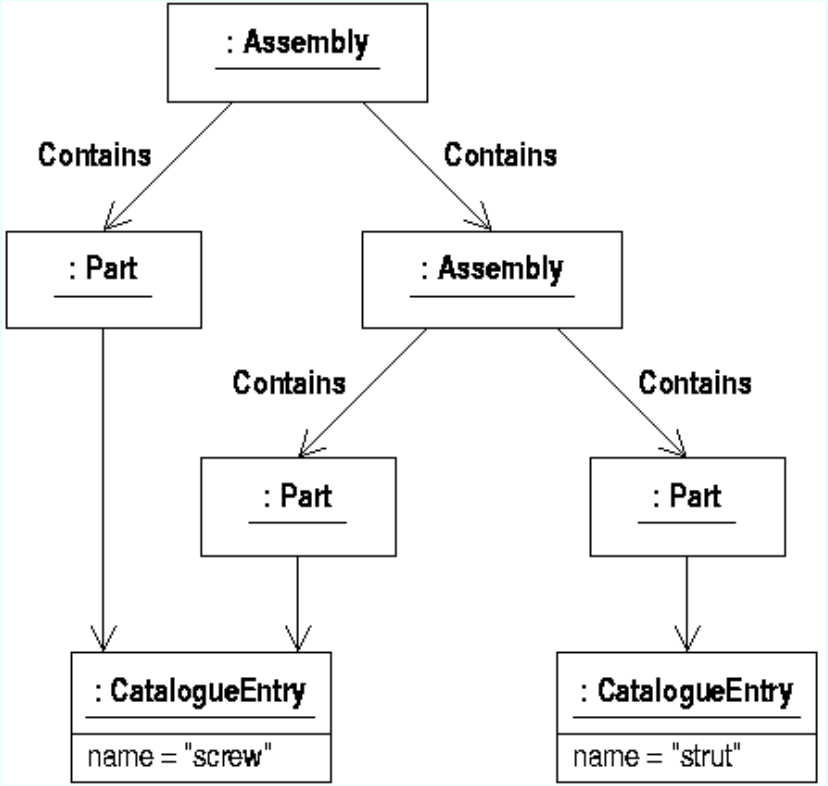
1.13.11 goto p.25

The 'Contains' Association

- 'Contains' is the *association name*
  - "an assembly *contains* parts"
- 'parts' is the *rolename*
- Assemblies can contain zero or more parts
  - this is documented by the 'parts' multiplicity



Subassemblies



Polymorphism

- This requires assemblies to hold a mix of pointers to parts and (sub)assemblies
- Inheritance is often used to implement this:

```
public abstract class Component { ... }
public class Part extends Component { ... }
public class Assembly extends Component {
  private Vector components = new Vector() ;
  public void addComponent(Component c) {
   components.addElement(c) ;
  }
}
```

Ex. เครื่องแบ่งนับ.

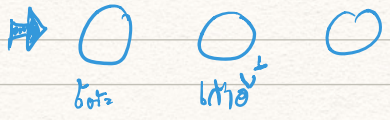
Abtract Class

Extend.

Concrete Class

- Class ที่มีอยู่จริง เหมือนของจริง.

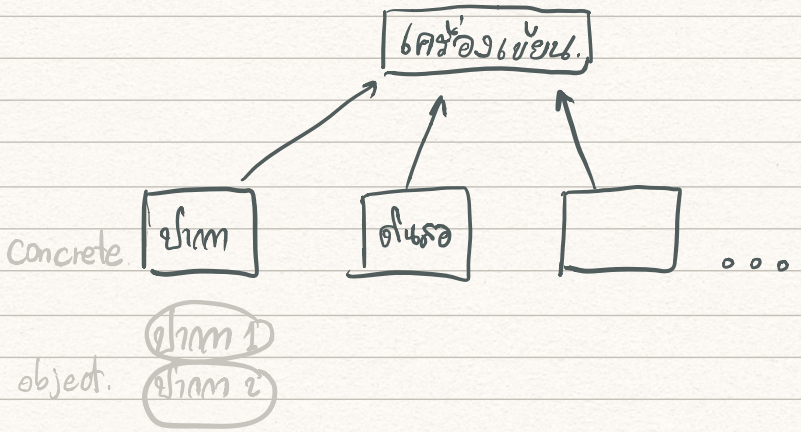
Physical Class.

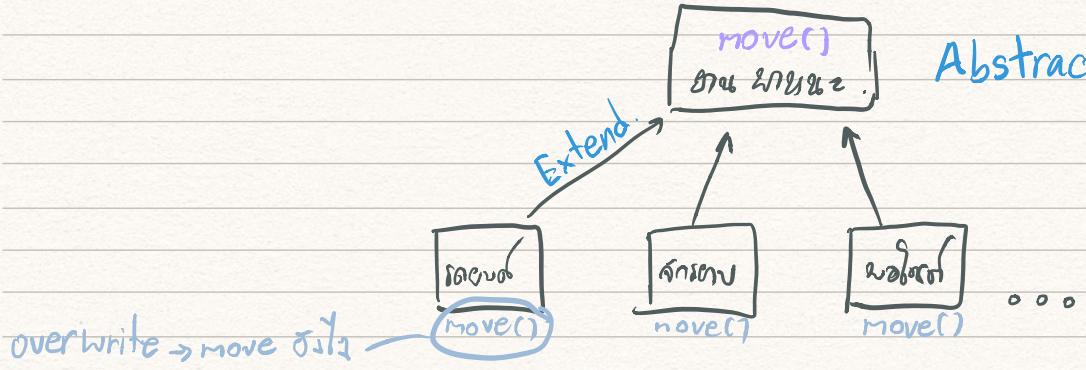


ตัวแบบ.

reality

Ex. ปากกา ดินสอ.





1.32.30 goto p.21.

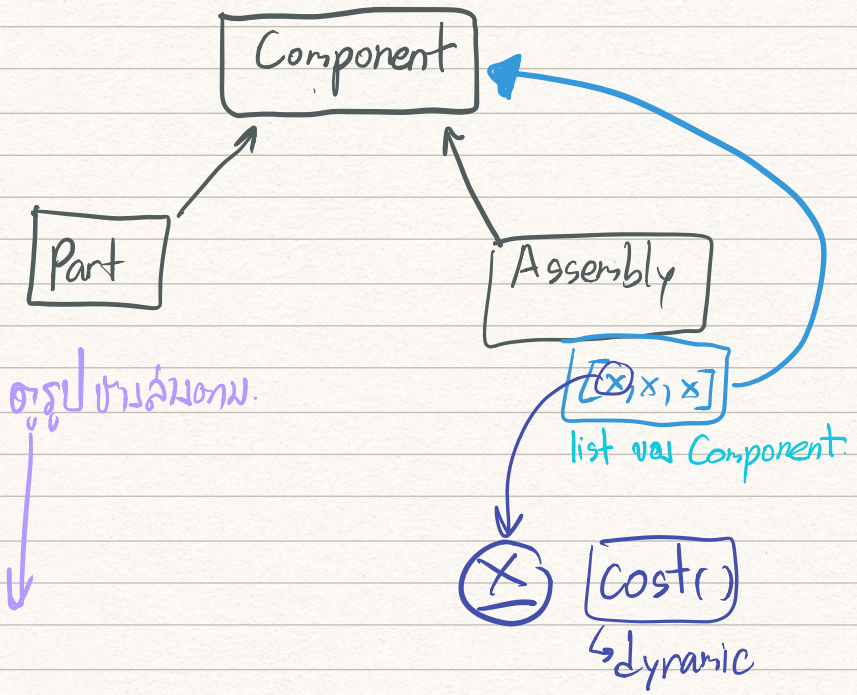
1.37.33.

Practical Object-Oriented Design with UML 2e

Polymorphism

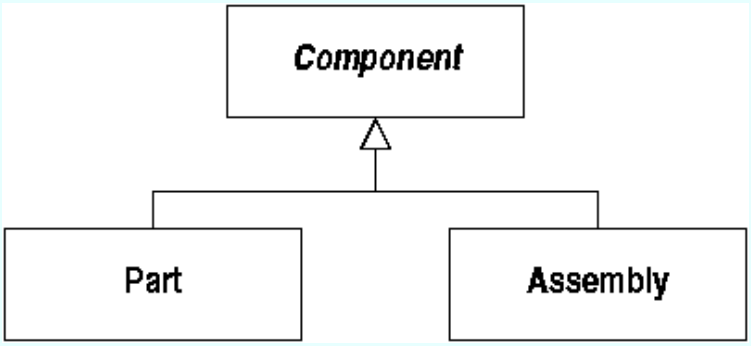
- This requires assemblies to hold a mix of pointers to parts and (sub)assemblies
- Inheritance is often used to implement this:

```
public abstract class Component { ... }
public class Part extends Component { ... }
public class Assembly extends Component {
 private Vector components = new Vector() ;
 public void addComponent(Component c) {
  components.addElement(c) ;
 }
}
```



Generalization

- Java's 'extends' relationship is represented by *generalization* in UML
  - Also called *specialization* if viewed 'downwards'



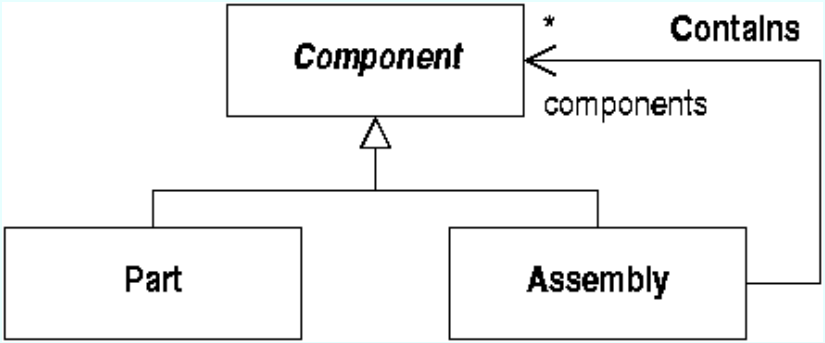
1.41.25 goto 28.

Properties of Generalization

- Generalization is not an association:
  - does not give rise to links between objects
  - usually unlabelled
  - never has multiplicity annotations
- Explained by ‘substitutability’:
  - an instance of a subclass can be used wherever an instance of a superclass is expected
  - similar to ‘reference conversions’ in Java

A Polymorphic Association

- Assemblies contain Components:
  - instances of ‘Contains’ can link assemblies to instances of both Component subclasses
  - the Component class is *abstract* (no instances)



C++, for example, the mechanism of virtual functions must be used.

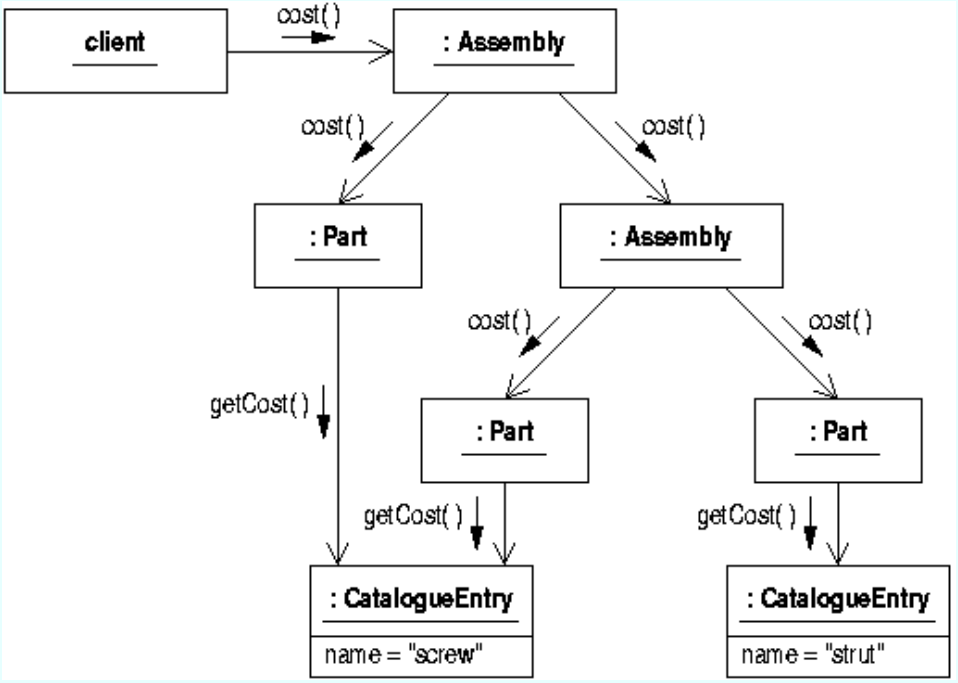
```
public abstract class Component
{
  public abstract double cost () ;
}

public class Part extends Component
{
  private CatalogueEntry entry ;

  public double cost() {
    return entry.getCost() ;
  }
}

public class Assembly extends Component
{
  private Vector components = new Vector() ;

  public double cost() {
    double total = 0.0 ;
    Enumeration enum = components.elements() ;
    while (enum.hasMoreElements()) {
      total += ((Component) enum.nextElement()).
    }
    return total ;
  }
```

1.49.46

อ.อธิบายบนกระดาน 1.49.50 – 1.53.07

The Cost of an Assembly

- Add up the cost of all the components



Dynamic Binding

- Both ‘Part’ and ‘Assembly’ classes must implement a ‘cost’ method
- An assembly sends the ‘cost’ message to all its components
- The run-time system checks the type of the message receiver and invokes the correct method

Applicability of Object Model

- Some classes inspired by real-world objects
- Real-world doesn't necessarily provide a good software design
  - e.g. data replication in the original 'Part' class
- Object-orientation better understood as an effective approach to software architecture