Basic Object Design

C++ Object Oriented Programming

Pei-yih Ting
NTOUCS

Contents

- Object Oriented Analysis/Design

Contents

- Object Oriented Analysis/Design
- Elements of a well-designed class

Contents

- Object Oriented Analysis/Design
- Elements of a well-designed class
  - Strong Cohesion

Contents

- Object Oriented Analysis/Design
- Elements of a well-designed class
  - Strong Cohesion
  - Completeness and Convenience

Contents

- Object Oriented Analysis/Design
- Elements of a well-designed class
  - Strong Cohesion
  - Completeness and Convenience
  - Consistency

Contents

- Object Oriented Analysis/Design
- Elements of a well-designed class
  - Strong Cohesion
  - Completeness and Convenience
  - Consistency
  - Loose Coupling

Contents

- Object Oriented Analysis/Design
- Elements of a well-designed class
  - Strong Cohesion
  - Completeness and Convenience
  - Consistency
  - Loose Coupling
- Design classes before you code it

Contents

- Object Oriented Analysis/Design
- Elements of a well-designed class
  - Strong Cohesion
  - Completeness and Convenience
  - Consistency
  - Loose Coupling
- Design classes before you code it
  - CRC cards

Contents

- Object Oriented Analysis/Design
- Elements of a well-designed class
  - Strong Cohesion
  - Completeness and Convenience
  - Consistency
  - Loose Coupling
- Design classes before you code it
  - CRC cards
  - Class description

Contents

- Object Oriented Analysis/Design
- Elements of a well-designed class
  - Strong Cohesion
  - Completeness and Convenience
  - Consistency
  - Loose Coupling
- Design classes before you code it
  - CRC cards
  - Class description
  - Function description

Contents

- Object Oriented Analysis/Design
- Elements of a well-designed class
  - Strong Cohesion
  - Completeness and Convenience
  - Consistency
  - Loose Coupling
- Design classes before you code it
  - CRC cards
  - Class description
  - Function description
- Discover your classes

Contents

- Object Oriented Analysis/Design
- Elements of a well-designed class
  - Strong Cohesion
  - Completeness and Convenience
  - Consistency
  - Loose Coupling
- Design classes before you code it
  - CRC cards
  - Class description
  - Function description
- Discover your classes
  - Object discovery techniques

Contents

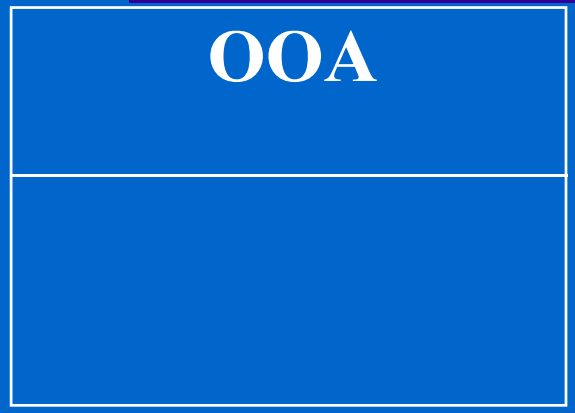
- Object Oriented Analysis/Design
- Elements of a well-designed class
  - Strong Cohesion
  - Completeness and Convenience
  - Consistency
  - Loose Coupling
- Design classes before you code it
  - CRC cards
  - Class description
  - Function description
- Discover your classes
  - Object discovery techniques
  - Noun-verb analysis example

Contents

- Object Oriented Analysis/Design
- Elements of a well-designed class
  - Strong Cohesion
  - Completeness and Convenience
  - Consistency
  - Loose Coupling
- Design classes before you code it
  - CRC cards
  - Class description
  - Function description
- Discover your classes
  - Object discovery techniques
  - Noun-verb analysis example
  - Tentative classes

Object Oriented Analysis/Design

Object Oriented Analysis/Design



- Object-Oriented Analysis (OOA)
  - What are the classes in the system?
  - What are the operations and attributes?
  - What are the inheritance relationships?

Object Oriented Analysis/Design

OOA
Identification

- Object-Oriented Analysis (OOA)
  - What are the classes in the system?
  - What are the operations and attributes?
  - What are the inheritance relationships?

Object Oriented Analysis/Design

OOA
Identification

What objects do I need to implement the system?

- Object-Oriented Analysis (OOA)
  - What are the classes in the system?
  - What are the operations and attributes?
  - What are the inheritance relationships?

Object Oriented Analysis/Design

OOA **Identification**	OOD
What objects do I need to implement the system?	

- Object-Oriented Analysis (OOA)
  - What are the classes in the system?
  - What are the operations and attributes?
  - What are the inheritance relationships?
- Object-Oriented Design (OOD)
  - How do objects relate to other objects?
  - How is the system constructed with the objects?

Object Oriented Analysis/Design

OOA Identification	OOD Integration
What objects do I need to implement the system?	

- Object-Oriented Analysis (OOA)
  - What are the classes in the system?
  - What are the operations and attributes?
  - What are the inheritance relationships?
- Object-Oriented Design (OOD)
  - How do objects relate to other objects?
  - How is the system constructed with the objects?

Object Oriented Analysis/Design

OOA **Identification**	OOD **Integration**
What objects do I need to implement the system?	How do I integrate the objects to make the system work?

- Object-Oriented Analysis (OOA)
  - What are the classes in the system?
  - What are the operations and attributes?
  - What are the inheritance relationships?
- Object-Oriented Design (OOD)
  - How do objects relate to other objects?
  - How is the system constructed with the objects?

Object Oriented Analysis/Design

OOA **Identification**	OOD **Integration**	OOP
What objects do I need to implement the system?	How do I integrate the objects to make the system work?	

- Object-Oriented Analysis (OOA)
  - What are the classes in the system?
  - What are the operations and attributes?
  - What are the inheritance relationships?
- Object-Oriented Design (OOD)
  - How do objects relate to other objects?
  - How is the system constructed with the objects?
- Object-Oriented Programming (OOP)

  How do you create the system using your particular object-oriented programming language?

Object Oriented Analysis/Design

OOA Identification	OOD Integration	OOP Implementation
What objects do I need to implement the system?	How do I integrate the objects to make the system work?	

- Object-Oriented Analysis (OOA)
  - What are the classes in the system?
  - What are the operations and attributes?
  - What are the inheritance relationships?
- Object-Oriented Design (OOD)
  - How do objects relate to other objects?
  - How is the system constructed with the objects?
- Object-Oriented Programming (OOP)

  How do you create the system using your particular object-oriented programming language?

Object Oriented Analysis/Design

OOA Identification	OOD Integration	OOP Implementation
What objects do I need to implement the system?	How do I integrate the objects to make the system work?	How do I use the programming lang to create each object?

- Object-Oriented Analysis (OOA)
  - What are the classes in the system?
  - What are the operations and attributes?
  - What are the inheritance relationships?
- Object-Oriented Design (OOD)
  - How do objects relate to other objects?
  - How is the system constructed with the objects?
- Object-Oriented Programming (OOP)

  How do you create the system using your particular object-oriented programming language?

Object Oriented Analysis/Design

- There are generally four phases to the object-oriented analysis/design process:

Object Oriented Analysis/Design

- There are generally four phases to the object-oriented analysis/design process:
- In the **problem domain**,

Object Oriented Analysis/Design

- There are generally four phases to the object-oriented analysis/design process:
- In the **problem domain**,
  - identification of objects from the program specification.

Object Oriented Analysis/Design

- There are generally four phases to the object-oriented analysis/design process:
- In the **problem domain**,
  - identification of objects from the program specification.
  - identification of the attributes and behaviors of these objects.

Object Oriented Analysis/Design

- There are generally four phases to the object-oriented analysis/design process:
- In the **problem domain**,
  - identification of objects from the program specification.
  - identification of the attributes and behaviors of these objects.
  - identification of any super-classes.

Object Oriented Analysis/Design

- There are generally four phases to the object-oriented analysis/design process:
- In the **problem domain**,
  - identification of objects from the program specification.
  - identification of the attributes and behaviors of these objects.
  - identification of any super-classes.

  specification of the behaviors of the identified classes.

Basic Object Design

Objects in general have two important properties:

Basic Object Design

Objects in general have two important properties:

1. State

Basic Object Design

Objects in general have two important properties:

1. State
2. Behaviour

- Object **States**:

Basic Object Design

Objects in general have two important properties:

1. State
2. Behaviour

- Object **States**:

  An object contains certain information about itself e.g.

Basic Object Design

Objects in general have two important properties:

1. State
2. Behaviour

- Object **States**:

  An object contains certain information about itself e.g.

  - a lecturer "knows" his name, address, age, courses he teach ...

Basic Object Design

Objects in general have two important properties:

1. State
2. Behaviour

- Object **States**:

  An object contains certain information about itself e.g.

  - a lecturer "knows" his name, address, age, courses he teach ...
  - a student "knows" his name, address, age, ID, courses studied ...

Basic Object Design

Objects in general have two important properties:

1. State
2. Behaviour

- Object **States**:

  An object contains certain information about itself e.g.

  - a lecturer "knows" his name, address, age, courses he teach ...
  - a student "knows" his name, address, age, ID, courses studied ...
  - a lecture theatre "knows" its location, capacity etc.

Basic Object Design

Objects in general have two important properties:

1. State
2. Behaviour

- Object **States**:

  An object contains certain information about itself e.g.

  - a lecturer "knows" his name, address, age, courses he teach ...
  - a student "knows" his name, address, age, ID, courses studied ...
  - a lecture theatre "knows" its location, capacity etc.

The information that an object maintains determines its state.

The individual components of information are known as the objects ***attributes***.

Basic Object Design (cont'd)

- Object **Behaviours**:

Basic Object Design (cont'd)

- Object **Behaviours**:

  Apart from maintaining information about itself, an object is also capable of performing certain actions. e.g.

Basic Object Design (cont'd)

- Object **Behaviours**:

  Apart from maintaining information about itself, an object is also capable of performing certain actions. e.g.

  - a lecturer teach a class, grade assignments, set an examination paper, etc.

Basic Object Design (cont'd)

- Object **Behaviours**:

  Apart from maintaining information about itself, an object is also capable of performing certain actions. e.g.

  - a lecturer teach a class, grade assignments, set an examination paper, etc.
  - a student attend a lecture, complete an assignment, sit in an exam, etc.

Basic Object Design (cont'd)

- Object **Behaviours**:

  Apart from maintaining information about itself, an object is also capable of performing certain actions. e.g.

  - a lecturer teach a class, grade assignments, set an examination paper, etc.
  - a student attend a lecture, complete an assignment, sit in an exam, etc.

The actions that an object can perform are known as its behaviours.

Basic Object Design (cont'd)

- Object **Behaviours**:

  Apart from maintaining information about itself, an object is also capable of performing certain actions. e.g.

  - a lecturer teach a class, grade assignments, set an examination paper, etc.
  - a student attend a lecture, complete an assignment, sit in an exam, etc.

The actions that an object can perform are known as its behaviours.

When applying an ***object-orientated analysis & design*** to a problem specification we ***identify objects***, ***record their states***, and ***specify their behaviours***.

Specifying Good Objects

- Strong Cohesion

Specifying Good Objects

- Strong Cohesion
- Completeness and Convenience

Specifying Good Objects

- Strong Cohesion
- Completeness and Convenience
- Consistency

Specifying Good Objects

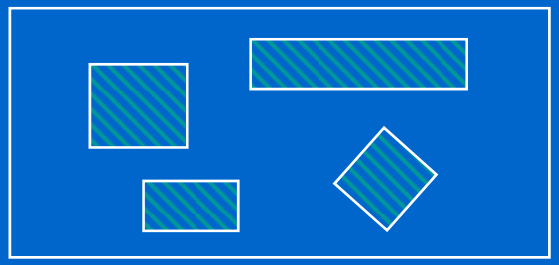
- Strong Cohesion
- Completeness and Convenience
- Consistency
- Loose Coupling

Cohesion

- A good class describes a single abstraction

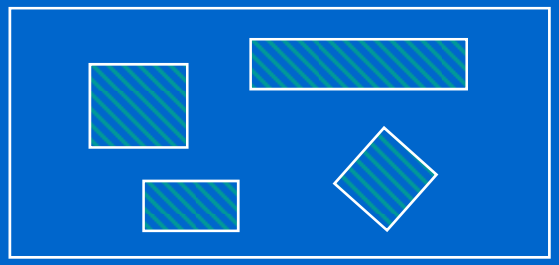
Cohesion

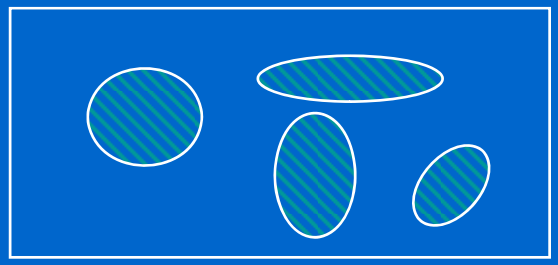
- A good class describes a single abstraction



Cohesion

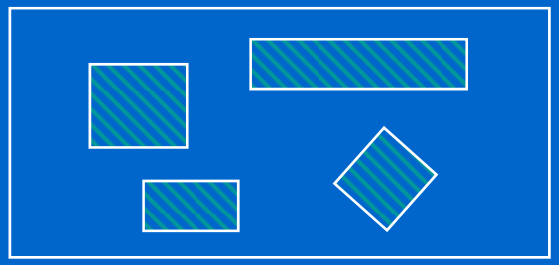
- A good class describes a single abstraction

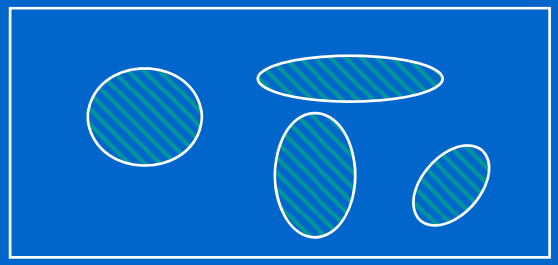




Cohesion

- A good class describes a single abstraction

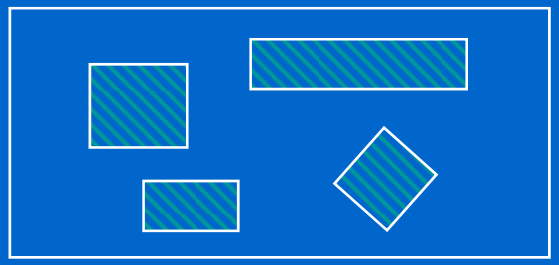


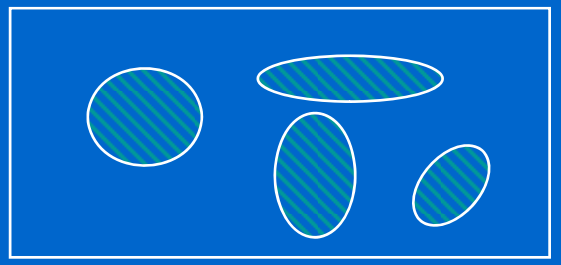


- Assume we are writing a networking email program

Cohesion

- A good class describes a single abstraction



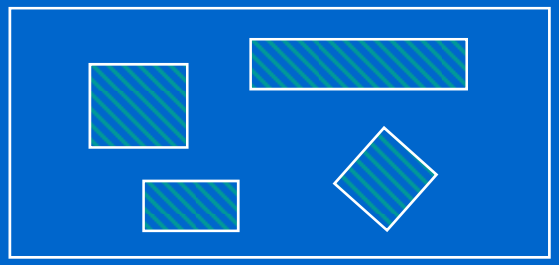


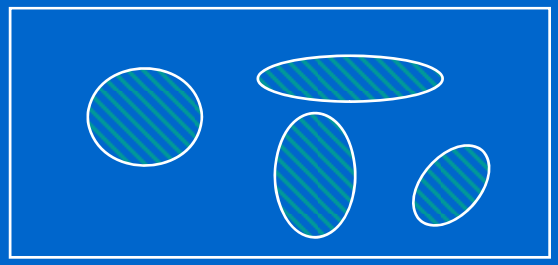
- Assume we are writing a networking email program

```
class Mail {
public:
    void sendMessage() const;
    void receiveMessage();
    void displayMessage() const;
    void processCommand();
    void getCommand();
private:
    char *m_message;
    char *m_command;
};
```

Cohesion

- A good class describes a single abstraction





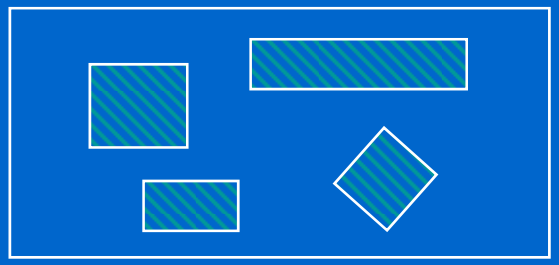
- Assume we are writing a networking email program

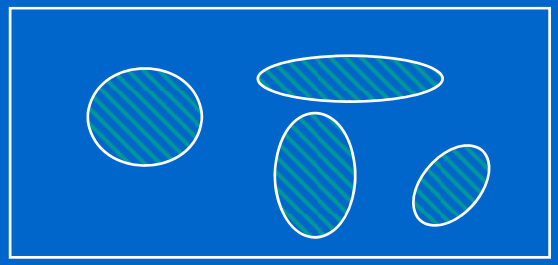
Why does this class lack cohesion?

```
class Mail {
public:
   void sendMessage() const;
   void receiveMessage();
   void displayMessage() const;
   void processCommand();
   void getCommand();
private:
   char *m_message;
   char *m_command;
};
```

Cohesion

- A good class describes a single abstraction





- Assume we are writing a networking email program

```
class Mail {
public:
   void sendMessage() const;
   void receiveMessage();
   void displayMessage() const;
   void processCommand();
   void getCommand();
private:
   char *m_message;
   char *m_command;
};
```

Why does this class lack cohesion?

- To achieve good cohesion, you must classify objects into groups with **close functionalities**.

Completeness and Convenience

- Every class must contain all necessary features.

Completeness and Convenience

- Every class must contain all necessary features.

```
class String {
public:
   String(char *inputData);
   void displayString() const;
   char getLetter(int slot) const;
   char getLength() const;
private:
   char *m_string;
};
```

Completeness and Convenience

- Every class must contain all necessary features.
  - Why is this class **not complete**?
  - What would be **desirable** but not **essential** features?

```
class String {
public:
   String(char *inputData);
   void displayString() const;
   char getLetter(int slot) const;
   char getLength() const;
private:
   char *m_string;
};
```

Completeness and Convenience

- Every class must contain all necessary features.
  - Why is this class **not complete**?
  - What would be **desirable** but not **essential** features?

```
class String {
public:
   String(char *inputData);
   void displayString() const;
   char getLetter(int slot) const;
   char getLength() const;
private:
   char *m_string;
};
```

- The opposite problem is a class that is over-complete in the name of convenience.

Completeness and Convenience

- Every class must contain all necessary features.
  - Why is this class **not complete**?
  - What would be **desirable** but not **essential** features?

```
class String {
public:
   String(char *inputData);
   void displayString() const;
   char getLetter(int slot) const;
   char getLength() const;
private:
   char *m_string;
};
```

- The opposite problem is a class that is over-complete in the name of convenience.

```
char getLetter(int slot) const;
char getFirstLetter() const;
char getLastLetter() const;
char getPreviousLetter() const;
char getNextLetter() const;
char findLetter(char letter) const; // find first occurrence of letter
char findLetterEnd(char letter) const; // finds last occurrence
```

Completeness and Convenience

- Every class must contain all necessary features.
  - Why is this class **not complete**?
  - What would be **desirable** but not **essential** features?

```
class String {
public:
   String(char *inputData);
   void displayString() const;
   char getLetter(int slot) const;
   char getLength() const;
private:
   char *m_string;
};
```

- The opposite problem is a class that is over-complete in the name of convenience.

```
char getLetter(int slot) const;
char getFirstLetter() const;
char getLastLetter() const;
char getPreviousLetter() const;
char getNextLetter() const;
char findLetter(char letter) const; // find first occurrence of letter
char findLetterEnd(char letter) const; // finds last occurrence
```

  - A class stuffed with **unnecessary** features is not convenient.

Consistency

- Here is a very inconsistent class.

Consistency

- Here is a very inconsistent class.

```
class Data {
public:
   Data(); ~Data();
   Data(char *name, int weight, int height);
   void setWeight(int weight);
   void setHeight(int height);
   int returnWeight();
   int getSize();
private:
   char *m_name;
   int m_weight;
   int m_length;
};
```

Consistency

- Here is a very inconsistent class.

```
class Data {
public:
   Data(); ~Data();
   Data(char *name, int weight, int height);
   void setWeight(int weight);
   void setHeight(int height);
   int returnWeight();
   int getSize();
private:
   char *m_name;
   int m_weight;
   int m_length;
};
```

```
class Graphics {
   void drawLine(int x, int y); // absolute coordinate
   void movePen(int deltaX, int deltaY); // relative offsets
};
```

Consistency

- Here is a very inconsistent class.

```
class Data {
public:
   Data(); ~Data();
   Data(char *name, int weight, int height);
   void setWeight(int weight);
   void setHeight(int height);
   int returnWeight();
   int getSize();
private:
   char *m_name;
   int m_weight;
   int m_length;
};
```

```
class Graphics {
   void drawLine(int x, int y); // absolute coordinate
   void movePen(int deltaX, int deltaY); // relative offsets
};
```

- This Graphics class is both **inconsistent** and **unclear**

Consistency

- Here is a very inconsistent class.

```
class Data {
public:
   Data(); ~Data();
   Data(char *name, int weight, int height);
   void setWeight(int weight);
   void setHeight(int height);
   int returnWeight();
   int getSize();
private:
   char *m_name;
   int m_weight;
   int m_length;
};
```

```
class Graphics {
   void drawLine(int x, int y); // absolute coordinate
   void movePen(int deltaX, int deltaY); // relative offsets
};
```

- This Graphics class is both **inconsistent** and **unclear**
  - drawLine() draws a line from the current pen position to the new coordinate (x, y) which is specified in *absolute* coordinates

Consistency

- Here is a very inconsistent class.

```
class Data {
public:
   Data(); ~Data();
   Data(char *name, int weight, int height);
   void setWeight(int weight);
   void setHeight(int height);
   int returnWeight();
   int getSize();
private:
   char *m_name;
   int m_weight;
   int m_length;
};
```

```
class Graphics {
   void drawLine(int x, int y); // absolute coordinate
   void movePen(int deltaX, int deltaY); // relative offsets
};
```

- This Graphics class is both **inconsistent** and **unclear**
  - drawLine() draws a line from the current pen position to the new coordinate (x, y) which is specified in *absolute* coordinates
  - movePen() moves the pen from the current position by the amounts (x, y) which is specified in *relative* coordinates

Consistency

- Here is a very inconsistent class.

```
class Data {
public:
   Data(); ~Data();
   Data(char *name, int weight, int height);
   void setWeight(int weight);
   void setHeight(int height);
   int returnWeight();
   int getSize();
private:
   char *m_name;
   int m_weight;
   int m_length;
};
```

```
class Graphics {
   void drawLine(int x, int y); // absolute coordinate
   void movePen(int deltaX, int deltaY); // relative offsets
};
```

Without these descriptions, it is hard to guess what functions of this class are about.

- This Graphics class is both **inconsistent** and **unclear**
  - drawLine() draws a line from the current pen position to the new coordinate (x, y) which is specified in *absolute* coordinates
  - movePen() moves the pen from the current position by the amounts (x, y) which is specified in *relative* coordinates

Coupling

- Classes with many interconnections are *highly coupled*.

Coupling

- Classes with many interconnections are *highly coupled*.

```
class Input { // returns data from file at location
   fileReferenceNum
public:
   double readFromFile(long &fileReferenceNum);
};
```

Coupling

- Classes with many interconnections are *highly coupled*.

```
class Input { // returns data from file at location
   fileReferenceNum
public:
   double readFromFile(long &fileReferenceNum);
};
```

```
class Math { // returns sine or cosine of current data in file
public:
   double sine(Input source, long &fileReferenceNum);
   double cosine(Input source, long &fileReferenceNum);
};
```

Coupling

- Classes with many interconnections are *highly coupled*.

```
void main() {
   Math mathObject;
   Input inputObject;
   long fileReferenceNum = 0; // do not forget initialization
   cout << mathObject.sine(inputObject, fileReferenceNum);
}
```

```
class Input { // returns data from file at location
   fileReferenceNum
public:
   double readFromFile(long &fileReferenceNum);
};
```

```
class Math { // returns sine or cosine of current data in file
public:
   double sine(Input source, long &fileReferenceNum);
   double cosine(Input source, long &fileReferenceNum);
};
```

Reduce Coupling

- Encapsulation reduces coupling

Reduce Coupling

- Encapsulation reduces coupling

```
class Input {
public:
   Input();  // will set m_refNum to zero
   double readFromFile();
private:     // will take care of m_refNum
   int m_refNum;
};
```

Reduce Coupling

- Encapsulation reduces coupling

```
class Input {
public:
   Input();  // will set m_refNum to zero
   double readFromFile();
private:     // will take care of m_refNum
   int m_refNum;
};
```

```
class Math {
public:
   Math(Input &);
   double sine();       // will handle m_data
   double cosine();     // automatically
private:
   Input m_data;
};
```

Reduce Coupling

- Encapsulation reduces coupling

```
class Input {
public:
   Input();  // will set m_refNum to zero
   double readFromFile();
private:     // will take care of m_refNum
   int m_refNum;
};
```

```
class Math {
public:
   Math(Input &);
   double sine();       // will handle m_data
   double cosine();     // automatically
private:
   Input m_data;
};
```

```
void main() {
   Input inputObject;
   Math mathObject(inputObject);
   cout << mathObject.sine();
}
```

Reduce Coupling

- Encapsulation reduces coupling

```
class Input {
public:
   Input();  // will set m_refNum to zero
   double readFromFile();
private:     // will take care of m_refNum
   int m_refNum;
};
```

```
void main() {
   Input inputObject;
   Math mathObject(inputObject);
   cout << mathObject.sine();
}
```

```
class Math {
public:
   Math(Input &);
   double sine();       // will handle m_data
   double cosine();     // automatically
private:
   Input m_data;
};
```

- Avoid passing a great amount of data across object boundaries. Object should provide abstract and simple services.

Reduce Coupling

- Encapsulation reduces coupling

```
class Input {
public:
   Input();  // will set m_refNum to zero
   double readFromFile();
private:     // will take care of m_refNum
   int m_refNum;
};
```

```
class Math {
public:
   Math(Input &);
   double sine();       // will handle m_data
   double cosine();     // automatically
private:
   Input m_data;
};
```

```
void main() {
   Input inputObject;
   Math mathObject(inputObject);
   cout << mathObject.sine();
}
```

- Avoid passing a great amount of data across object boundaries. Object should provide abstract and simple services.

- As opposed to the *data flow* design methodology, in which data flows along processing units, object oriented/based programming design objects to keep and handle data intelligently. Put all responsible objects together with close links for accomplishing a specific work without looking into their detailed processed data.

Design Classes Before You Code It

- Before writing a large program, decide on your classes, what they do, and how they relate to other classes.

Design Classes Before You Code It

- Before writing a large program, decide on your classes, what they do, and how they relate to other classes.
- CRC cards – Classes – Responsibilities, Collaborators

Design Classes Before You Code It

- Before writing a large program, decide on your classes, what they do, and how they relate to other classes.
- CRC cards – Classes – Responsibilities, Collaborators
- Example

Class Math	
Responsibilities	Collaborators
Return sine of file data	Input
Return cosine of file data	Input

Design Classes Before You Code It

- Before writing a large program, decide on your classes, what they do, and how they relate to other classes.
- CRC cards – Classes – Responsibilities, Collaborators
- Example

Class Math	
Responsibilities	Collaborators
Return sine of file data	Input
Return cosine of file data	Input

Class Input	
Responsibilities	Collaborators
Read next data from file	-

Design Classes Before You Code It

- Before writing a large program, decide on your classes, what they do, and how they relate to other classes.
- CRC cards – Classes – Responsibilities, Collaborators
- Example

Class Math	
Responsibilities	Collaborators
Return sine of file data	Input
Return cosine of file data	Input

Class Input	
Responsibilities	Collaborators
Read next data from file	-

- What about the data members?
  These are hashed out after all the CRC cards have been prepared.

Class Description

- An alternative approach to the CRC method

Class Description

- An alternative approach to the CRC method

Name	**Array**
Purpose	**Create a fixed-size array which protects against out of bounds and off by one errors.**
Constructors	**Default set the array to size 0** **Non default sets the array to a size specified by the client**
Destructors	**Deletes the memory associated with the array**
Operations **Mutators** **Accessors**	 **Insert data into a specified slot** **Retrieve data from a specified slot**
Fields	**m_dataSize** **m_data**

Class Description

- An alternative approach to the CRC method

Name	Array
Purpose	Create a fixed-size array which protects against out of bounds and off by one errors.
Constructors	Default set the array to size 0 Non default sets the array to a size specified by the client
Destructors	Deletes the memory associated with the array
Operations Mutators Accessors	 Insert data into a specified slot Retrieve data from a specified slot
Fields	m_dataSize m_data

- Codes

```
class Array {
public:
   Array();
   Array(int arraySize);
   ~Array();
   void insertElement(int element, int slot);
   int getElement(int slot) const;
private:
   int m_dataSize;
   int *m_data;
};
```

Function Descriptions

- Each function should be completely specified before coding.

Function Descriptions

- Each function should be completely specified before coding.

Prototype	**int getElement(int slot) const;**
Purpose	**To return the integer in the array at position slot**
Receives	**The slot which the client would like to access. The first element in the array is slot 0.**
Returns	**The integer if the function succeeds, otherwise returns an error value specified as kError**
Remarks	**kError is currently set to 0.**

Function Descriptions

- Each function should be completely specified before coding.

Prototype	**int getElement(int slot) const;**
Purpose	**To return the integer in the array at position slot**
Receives	**The slot which the client would like to access. The first element in the array is slot 0.**
Returns	**The integer if the function succeeds, otherwise returns an error value specified as kError**
Remarks	**kError is currently set to 0.**

- Alternatively, write the complete function documentation and prepare a skeleton function declaration

Function Descriptions

- Each function should be completely specified before coding.

Prototype	**int getElement(int slot) const;**
Purpose	**To return the integer in the array at position slot**
Receives	**The slot which the client would like to access. The first element in the array is slot 0.**
Returns	**The integer if the function succeeds, otherwise returns an error value specified as kError**
Remarks	**kError is currently set to 0.**

- Alternatively, write the complete function documentation and prepare a skeleton function declaration

```
/* function: getElement
 * Usage: value = getElement(slot);
 * ----------------------------------
 * Returns the integer in the array corresponding to slot.
 * The first element is slot zero.  If the slot is out of range
 * kError is returned, which is currently zero.
 */
int Array::getElement(int slot) {
}
```

Discover Your Classes

- Bertrand Meyer in "Object-oriented Software Construction"

Discover Your Classes

- Bertrand Meyer in "Object-oriented Software Construction"

  "When software design is understood as operational modeling, object-oriented design is a natural approach

Discover Your Classes

- Bertrand Meyer in "Object-oriented Software Construction"

  "When software design is understood as operational modeling, object-oriented design is a natural approach: the world being modeled is made of objects – sensors, devices, airplanes, employees, paychecks, tax returns

Discover Your Classes

- Bertrand Meyer in "Object-oriented Software Construction"

  "When software design is understood as operational modeling, object-oriented design is a natural approach: the world being modeled is made of objects – sensors, devices, airplanes, employees, paychecks, tax returns – and it is appropriate to organize the model around computer representations of theses objects.

Discover Your Classes

- Bertrand Meyer in "Object-oriented Software Construction"

"When software design is understood as operational modeling, object-oriented design is a natural approach: the world being modeled is made of objects – sensors, devices, airplanes, employees, paychecks, tax returns – and it is appropriate to organize the model around computer representations of theses objects. This is why object-oriented designers usually do not spend their time in academic discussions of methods to find objects: in the physical or abstract reality being modeled, the objects are just there for the picking!

Discover Your Classes

- Bertrand Meyer in "Object-oriented Software Construction"

  "When software design is understood as operational modeling, object-oriented design is a natural approach: the world being modeled is made of objects – sensors, devices, airplanes, employees, paychecks, tax returns – and it is appropriate to organize the model around computer representations of theses objects. This is why object-oriented designers usually do not spend their time in academic discussions of methods to find objects: in the physical or abstract reality being modeled, the objects are just there for the picking! The software objects will simply reflect these external objects."

Discover Your Classes

- Bertrand Meyer in "Object-oriented Software Construction"

  "When software design is understood as operational modeling, object-oriented design is a natural approach: the world being modeled is made of objects – sensors, devices, airplanes, employees, paychecks, tax returns – and it is appropriate to organize the model around computer representations of theses objects. This is why object-oriented designers usually do not spend their time in academic discussions of methods to find objects: in the physical or abstract reality being modeled, the objects are just there for the picking! The software objects will simply reflect these external objects."

- How do the experts identify objects?

Discover Your Classes

- Bertrand Meyer in "Object-oriented Software Construction"

  "When software design is understood as operational modeling, object-oriented design is a natural approach: the world being modeled is made of objects – sensors, devices, airplanes, employees, paychecks, tax returns – and it is appropriate to organize the model around computer representations of theses objects. This is why object-oriented designers usually do not spend their time in academic discussions of methods to find objects: in the physical or abstract reality being modeled, the objects are just there for the picking! The software objects will simply reflect these external objects."

- How do the experts identify objects?

  "It's a Holy Grail. There is no panacea." -- Bjarne Stroustrup

Discover Your Classes

- Bertrand Meyer in "Object-oriented Software Construction"

  "When software design is understood as operational modeling, object-oriented design is a natural approach: the world being modeled is made of objects – sensors, devices, airplanes, employees, paychecks, tax returns – and it is appropriate to organize the model around computer representations of theses objects. This is why object-oriented designers usually do not spend their time in academic discussions of methods to find objects: in the physical or abstract reality being modeled, the objects are just there for the picking! The software objects will simply reflect these external objects."

- How do the experts identify objects?

  "It's a Holy Grail. There is no panacea." -- Bjarne Stroustrup

  "That's a fundamental question for which there is no easy answer." -- R. Gabriel, designer of Common Lisp Object System (CLOS)

Object Discovery Techniques

- **Real-world modeling**:

Object Discovery Techniques

- **Real-world modeling**:
  - Use objects in the application domain as the basis for objects in the system.

Object Discovery Techniques

- **Real-world modeling**:
  - Use objects in the application domain as the basis for objects in the system.
- **Behavior modeling**:

Object Discovery Techniques

- **Real-world modeling**:
  - Use objects in the application domain as the basis for objects in the system.
- **Behavior modeling**:
  - Determine the overall behaviors of the system (what it does).

Object Discovery Techniques

- **Real-world modeling**:
  - Use objects in the application domain as the basis for objects in the system.
- **Behavior modeling**:
  - Determine the overall behaviors of the system (what it does).
  - Components which play significant roles in each behavior are objects.

Object Discovery Techniques

- **Real-world modeling**:
  - Use objects in the application domain as the basis for objects in the system.
- **Behavior modeling**:
  - Determine the overall behaviors of the system (what it does).
  - Components which play significant roles in each behavior are objects.
- **Scenario-based analysis**:

Object Discovery Techniques

- **Real-world modeling**:
  - Use objects in the application domain as the basis for objects in the system.
- **Behavior modeling**:
  - Determine the overall behaviors of the system (what it does).
  - Components which play significant roles in each behavior are objects.
- **Scenario-based analysis**:
  - Create scenarios of the system.

Object Discovery Techniques

- **Real-world modeling**:
  - Use objects in the application domain as the basis for objects in the system.
- **Behavior modeling**:
  - Determine the overall behaviors of the system (what it does).
  - Components which play significant roles in each behavior are objects.
- **Scenario-based analysis**:
  - Create scenarios of the system.
  - What are the required entities in each scenario?

Object Discovery Techniques

- **Real-world modeling**:
  - Use objects in the application domain as the basis for objects in the system.
- **Behavior modeling**:
  - Determine the overall behaviors of the system (what it does).
  - Components which play significant roles in each behavior are objects.
- **Scenario-based analysis**:
  - Create scenarios of the system.
  - What are the required entities in each scenario?
- **Grammatical analysis**:

Object Discovery Techniques

- **Real-world modeling**:
  - Use objects in the application domain as the basis for objects in the system.
- **Behavior modeling**:
  - Determine the overall behaviors of the system (what it does).
  - Components which play significant roles in each behavior are objects.
- **Scenario-based analysis**:
  - Create scenarios of the system.
  - What are the required entities in each scenario?
- **Grammatical analysis**:
  - Write a natural language description of the system.

Object Discovery Techniques

- **Real-world modeling**:
  - Use objects in the application domain as the basis for objects in the system.
- **Behavior modeling**:
  - Determine the overall behaviors of the system (what it does).
  - Components which play significant roles in each behavior are objects.
- **Scenario-based analysis**:
  - Create scenarios of the system.
  - What are the required entities in each scenario?
- **Grammatical analysis**:
  - Write a natural language description of the system.
  - The nouns are the classes; the verbs are the methods.

Noun-Verb Analysis Example

- **Program description** (specification, highly abbreviated)

Noun-Verb Analysis Example

- **Program description** (specification, highly abbreviated)

  "The program allows the user to assign students to sections based on the available times.

Noun-Verb Analysis Example

- **Program description** (specification, highly abbreviated)

  "The program allows the user to assign students to sections based on the available times. Times are input by the teacher.

Noun-Verb Analysis Example

- **Program description** (specification, highly abbreviated)

  "The program allows the user to assign students to sections based on the available times. Times are input by the teacher. Students rank times by preference (up to three allowed) using a form.

Noun-Verb Analysis Example

- **Program description** (specification, highly abbreviated)

  "The program allows the user to assign students to sections based on the available times. Times are input by the teacher. Students rank times by preference (up to three allowed) using a form. All of the student inputs are collected into a central database.

Noun-Verb Analysis Example

- **Program description** (specification, highly abbreviated)

  "The program allows the user to assign students to sections based on the available times. Times are input by the teacher. Students rank times by preference (up to three allowed) using a form. All of the student inputs are collected into a central database. When the teacher indicates the database is complete

Noun-Verb Analysis Example

- **Program description** (specification, highly abbreviated)

  "The program allows the user to assign students to sections based on the available times. Times are input by the teacher. Students rank times by preference (up to three allowed) using a form. All of the student inputs are collected into a central database. When the teacher indicates the database is complete, the final result is optimized so that no section has more than 12 students

Noun-Verb Analysis Example

- **Program description** (specification, highly abbreviated)

  "The program allows the user to assign students to sections based on the available times. Times are input by the teacher. Students rank times by preference (up to three allowed) using a form. All of the student inputs are collected into a central database. When the teacher indicates the database is complete, the final result is optimized so that no section has more than 12 students and each student has received the highest possible preference.

Noun-Verb Analysis Example

- **Program description** (specification, highly abbreviated)

  "The program allows the user to assign students to sections based on the available times. Times are input by the teacher. Students rank times by preference (up to three allowed) using a form. All of the student inputs are collected into a central database. When the teacher indicates the database is complete, the final result is optimized so that no section has more than 12 students and each student has received the highest possible preference. The results are stored in a file showing which students have been assigned to which sections."

Noun-Verb Analysis Example

- **Program description** (specification, highly abbreviated)

  "The program allows the user to assign students to sections based on the available times. Times are input by the teacher. Students rank times by preference (up to three allowed) using a form. All of the student inputs are collected into a central database. When the teacher indicates the database is complete, the final result is optimized so that no section has more than 12 students and each student has received the highest possible preference. The results are stored in a file showing which students have been assigned to which sections."

- **Noun analysis**: **students, sections, times, teacher, preferences, form, student inputs, database, results, output file**

Noun-Verb Analysis Example

- **Program description** (specification, highly abbreviated)

  "The program allows the user to assign students to sections based on the available times. Times are input by the teacher. Students rank times by preference (up to three allowed) using a form. All of the student inputs are collected into a central database. When the teacher indicates the database is complete, the final result is optimized so that no section has more than 12 students and each student has received the highest possible preference. The results are stored in a file showing which students have been assigned to which sections."

- **Noun analysis**: **students, sections, times, teacher, preferences, form, student inputs, database, results, output file**

  This can be simplified further to just these categories

  form, (section) times, database, results (optimization process), output file

Noun-Verb Analysis Example

- **Program description** (specification, highly abbreviated)

  "The program allows the user to assign students to sections based on the available times. Times are input by the teacher. Students rank times by preference (up to three allowed) using a form. All of the student inputs are collected into a central database. When the teacher indicates the database is complete, the final result is optimized so that no section has more than 12 students and each student has received the highest possible preference. The results are stored in a file showing which students have been assigned to which sections."

- **Noun analysis**: **students, sections, times, teacher, preferences, form, student inputs, database, results, output file**

  This can be simplified further to just these categories

  form, (section) times, database, results (optimization process), output file

- **Possible classes**: **optimization process, student, teacher, form, sections, database, output**

Noun-Verb Analysis Example

- **Program description** (specification, highly abbreviated)

  "The program allows the user to assign students to sections based on the available times. Times are input by the teacher. Students rank times by preference (up to three allowed) using a form. All of the student inputs are collected into a central database. When the teacher indicates the database is complete, the final result is optimized so that no section has more than 12 students and each student has received the highest possible preference. The results are stored in a file showing which students have been assigned to which sections."

- **Noun analysis**: **students, sections, times, teacher, preferences, form, student inputs, database, results, output file**

  This can be simplified further to just these categories

  form, (section) times, database, results (optimization process), output file

- **Possible classes**: **optimization process, student, teacher, form, sections, database, output**

- **Verb analysis**: **assign students, input sections, rank by preference, collect into database, indicate database is complete, optimize results, store results in file**

Tentative Classes

- **Assign verbs to nouns**, that is, assign methods to classes. This is the usual classification problem.

Tentative Classes

- **Assign verbs to nouns**, that is, assign methods to classes. This is the usual classification problem.
  - Ex: class **Optimization**

Tentative Classes

- **Assign verbs to nouns**, that is, assign methods to classes. This is the usual classification problem.
  - Ex: class **Optimization**

    { optimize data

Tentative Classes

- **Assign verbs to nouns**, that is, assign methods to classes. This is the usual classification problem.
  - Ex: class **Optimization**

    { optimize data
    { store in file

Tentative Classes

- **Assign verbs to nouns**, that is, assign methods to classes. This is the usual classification problem.
  - Ex: class **Optimization** Possible collaborators: Database, File

    { optimize data
    store in file

Tentative Classes

- **Assign verbs to nouns**, that is, assign methods to classes. This is the usual classification problem.
  - Ex: class **Optimization** Possible collaborators: Database, File

    optimize data
    store in file
- **Expect changes**:

Tentative Classes

- **Assign verbs to nouns**, that is, assign methods to classes. This is the usual classification problem.
  - Ex: class **Optimization** Possible collaborators: Database, File

    { optimize data
    { store in file
- **Expect changes**:

  Designs always turn out to be wrong or incomplete

Tentative Classes

- **Assign verbs to nouns**, that is, assign methods to classes. This is the usual classification problem.
  - Ex: class **Optimization** Possible collaborators: Database, File

    { optimize data
    { store in file
- **Expect changes**:

  Designs always turn out to be wrong or incomplete, but having no design is worse.

Tentative Classes

- **Assign verbs to nouns**, that is, assign methods to classes. This is the usual classification problem.
  - Ex: class **Optimization** Possible collaborators: Database, File

    { optimize data
    { store in file
- **Expect changes**:

  Designs always turn out to be wrong or incomplete, but having no design is worse. In a suitably encapsulated object system, it is easy to **refactor**.

Tentative Classes

- **Assign verbs to nouns**, that is, assign methods to classes. This is the usual classification problem.
  - Ex: class **Optimization** Possible collaborators: Database, File
    { optimize data
    { store in file
- **Expect changes**:

  Designs always turn out to be wrong or incomplete, but having no design is worse. In a suitably encapsulated object system, it is easy to **refactor**. It is easy to create new objects

Tentative Classes

- **Assign verbs to nouns**, that is, assign methods to classes. This is the usual classification problem.
  - Ex: class **Optimization** Possible collaborators: Database, File

    { optimize data
    store in file
- **Expect changes**:

  Designs always turn out to be wrong or incomplete, but having no design is worse. In a suitably encapsulated object system, it is easy to **refactor**. It is easy to create new objects and to reassign methods or data from one class to another class.

Tentative Classes

- **Assign verbs to nouns**, that is, assign methods to classes. This is the usual classification problem.
  - Ex: class **Optimization** Possible collaborators: Database, File

    optimize data
    store in file
- **Expect changes**:

  Designs always turn out to be wrong or incomplete, but having no design is worse. In a suitably encapsulated object system, it is easy to **refactor**. It is easy to create new objects and to reassign methods or data from one class to another class.
- **Checking your design**:

Tentative Classes

- **Assign verbs to nouns**, that is, assign methods to classes. This is the usual classification problem.
  - Ex: class **Optimization** Possible collaborators: Database, File
    { optimize data
    { store in file
- **Expect changes**:

  Designs always turn out to be wrong or incomplete, but having no design is worse. In a suitably encapsulated object system, it is easy to **refactor**. It is easy to create new objects and to reassign methods or data from one class to another class.

- **Checking your design**:

  Once you have the classes,

Tentative Classes

- **Assign verbs to nouns**, that is, assign methods to classes. This is the usual classification problem.
  - Ex: class **Optimization** Possible collaborators: Database, File
    { optimize data
    { store in file
- **Expect changes**:

  Designs always turn out to be wrong or incomplete, but having no design is worse. In a suitably encapsulated object system, it is easy to **refactor**. It is easy to create new objects and to reassign methods or data from one class to another class.

- **Checking your design**:

  Once you have the classes, rewrite the program description using the new terms and actions.

Tentative Classes

- **Assign verbs to nouns**, that is, assign methods to classes. This is the usual classification problem.
  - Ex: class **Optimization** Possible collaborators: Database, File
    { optimize data
    { store in file
- **Expect changes**:

  Designs always turn out to be wrong or incomplete, but having no design is worse. In a suitably encapsulated object system, it is easy to **refactor**. It is easy to create new objects and to reassign methods or data from one class to another class.

- **Checking your design**:

  Once you have the classes, rewrite the program description using the new terms and actions. If the description does not make sense, you have a **bad design**.

Tentative Classes

- **Assign verbs to nouns**, that is, assign methods to classes. This is the usual classification problem.
  - Ex: class **Optimization** Possible collaborators: Database, File
    { optimize data
    { store in file
- **Expect changes**:

  Designs always turn out to be wrong or incomplete, but having no design is worse. In a suitably encapsulated object system, it is easy to **refactor**. It is easy to create new objects and to reassign methods or data from one class to another class.

- **Checking your design**:

  Once you have the classes, rewrite the program description using the new terms and actions. If the description does not make sense, you have a **bad design**. If it does, you have a better and cleaner description.

Tentative Classes

- **Assign verbs to nouns**, that is, assign methods to classes. This is the usual classification problem.
  - Ex: class **Optimization** Possible collaborators: Database, File
    { optimize data
    { store in file
- **Expect changes**:

  Designs always turn out to be wrong or incomplete, but having no design is worse. In a suitably encapsulated object system, it is easy to **refactor**. It is easy to create new objects and to reassign methods or data from one class to another class.

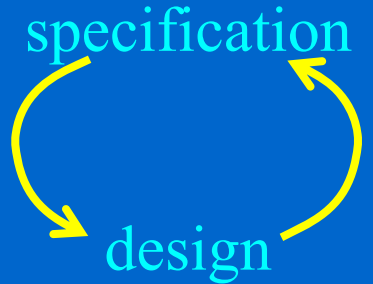
- **Checking your design**:

  Once you have the classes, rewrite the program description using the new terms and actions. If the description does not make sense, you have a **bad design**. If it does, you have a better and cleaner description. The model extracted will become **gradually simpler**.

Tentative Classes

- **Assign verbs to nouns**, that is, assign methods to classes. This is the usual classification problem.
  - Ex: class **Optimization** Possible collaborators: Database, File
    { optimize data
    { store in file
- **Expect changes**:

  Designs always turn out to be wrong or incomplete, but having no design is worse. In a suitably encapsulated object system, it is easy to **refactor**. It is easy to create new objects and to reassign methods or data from one class to another class.



- **Checking your design**:

  Once you have the classes, rewrite the program description using the new terms and actions. If the description does not make sense, you have a **bad design**. If it does, you have a better and cleaner description. The model extracted will become **gradually simpler**.

範例一

範例一

- 昨天我去剪頭髮，看到店裡的客人蠻多的，就問店員：現在可以馬上剪嗎？店員回答我：可以啊。在我坐下來後，店員走到我旁邊問我：你有指定的設計師嗎？我想了想回答他：沒有耶，都可以。

範例一

- 昨天我去剪頭髮，看到店裡的客人蠻多的，就問店員：現在可以馬上剪嗎？店員回答我：可以啊。在我坐下來後，店員走到我旁邊問我：你有指定的設計師嗎？我想了想回答他：沒有耶，都可以。隨後有一位帥哥來幫我洗頭髮，洗幾下之後，他就問我：這樣的力道可以嗎？本來想跟他說用力一點，但又怕太用力會抓破頭皮，所以就跟他說：很好。

範例一

- 昨天我去剪頭髮，看到店裡的客人蠻多的，就問店員：現在可以馬上剪嗎？店員回答我：可以啊。在我坐下來後，店員走到我旁邊問我：你有指定的設計師嗎？我想了想回答他：沒有耶，都可以。隨後有一位帥哥來幫我洗頭髮，洗幾下之後，他就問我：這樣的力道可以嗎？本來想跟他說用力一點，但又怕太用力會抓破頭皮，所以就跟他說：很好。洗完頭後，另一位設計師來幫我剪頭髮，首先他問我說：你要剪什麼樣的髮型？我跟他說：剪短一點就好。其實目的是剪短一點可以再撐三四個月不用剪頭髮。他又問：短一點就好嗎？我當下覺得，他可能不清楚我說的短是多短，所以我就說：可以很短沒關係。

範例一

- 昨天我去剪頭髮，看到店裡的客人蠻多的，就問店員：現在可以馬上剪嗎？店員回答我：可以啊。在我坐下來後，店員走到我旁邊問我：你有指定的設計師嗎？我想了想回答他：沒有耶，都可以。隨後有一位帥哥來幫我洗頭髮，洗幾下之後，他就問我：這樣的力道可以嗎？本來想跟他說用力一點，但又怕太用力會抓破頭皮，所以就跟他說：很好。洗完頭後，另一位設計師來幫我剪頭髮，首先他問我說：你要剪什麼樣的髮型？我跟他說：剪短一點就好。其實目的是剪短一點可以再撐三四個月不用剪頭髮。他又問：短一點就好嗎？我當下覺得，他可能不清楚我說的短是多短，所以我就說：可以很短沒關係。剪完我滿意的長度的頭髮後，我拿著帳單去櫃檯買單，問店員多少錢？他回答我說：**350**元，於是我拿一千元給他找，他找我 **650**元，收下錢，我便踏著輕快的腳步回家去了。

範例一 (cont'd)

- 物件:

範例一 (cont'd)

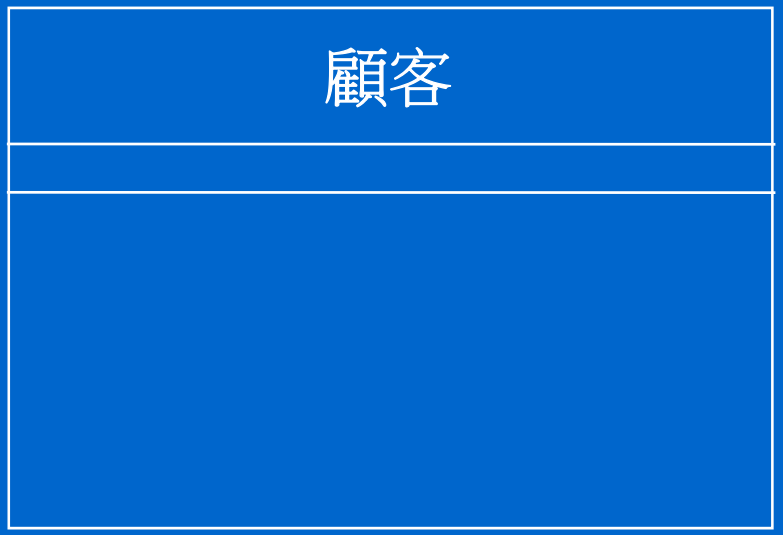
- 物件: 店員, 設計師, 顧客 (帳單, 發票)

範例一 (cont'd)

- 物件: 店員, 設計師, 顧客 (帳單, 發票)
- 類別圖:

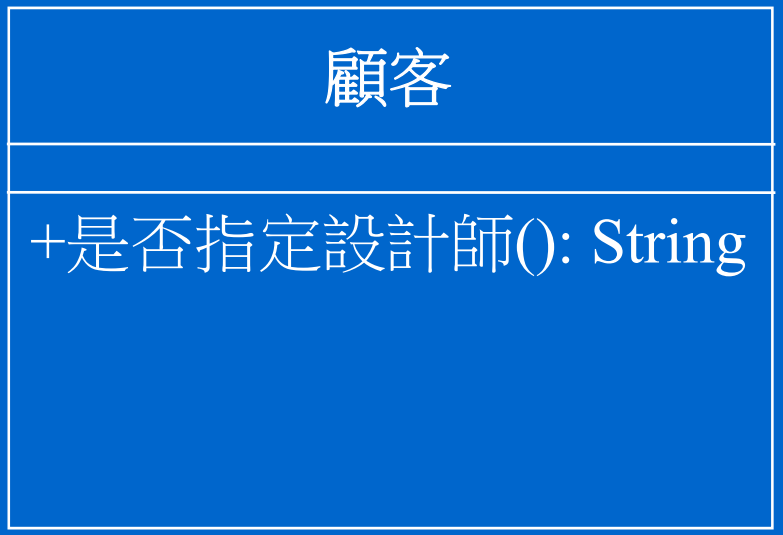
範例一 (cont'd)

- 物件: 店員, 設計師, 顧客 (帳單, 發票)
- 類別圖:



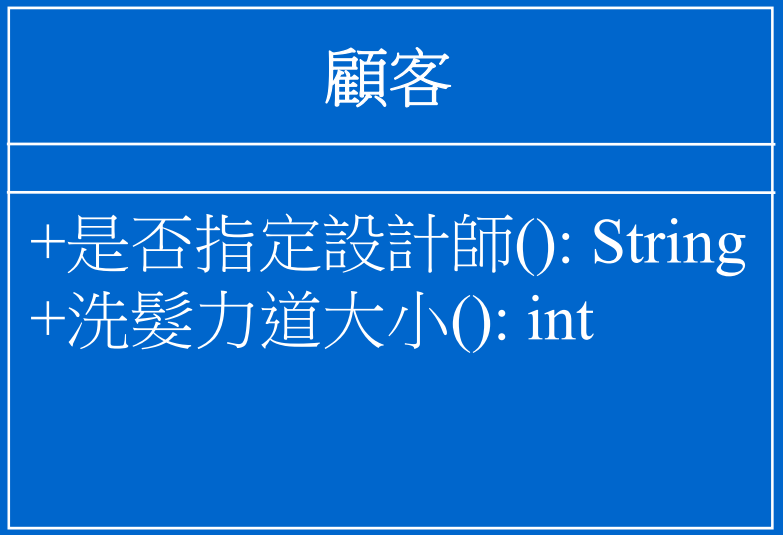
範例一 (cont'd)

- 物件: 店員, 設計師, 顧客 (帳單, 發票)
- 類別圖:



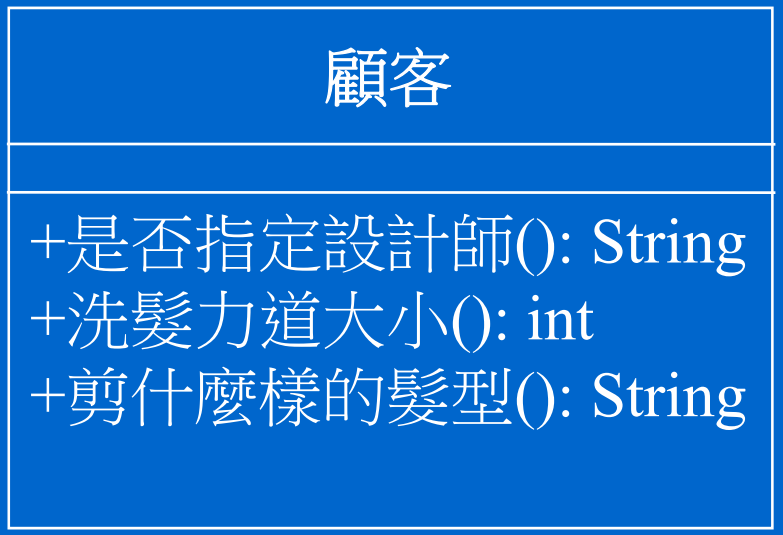
範例一 (cont'd)

- 物件: 店員, 設計師, 顧客 (帳單, 發票)
- 類別圖:



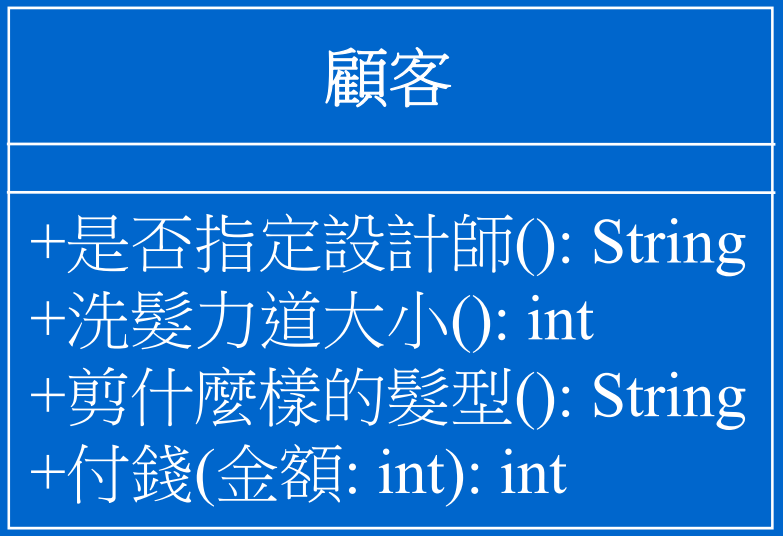
範例一 (cont'd)

- 物件: 店員, 設計師, 顧客 (帳單, 發票)
- 類別圖:



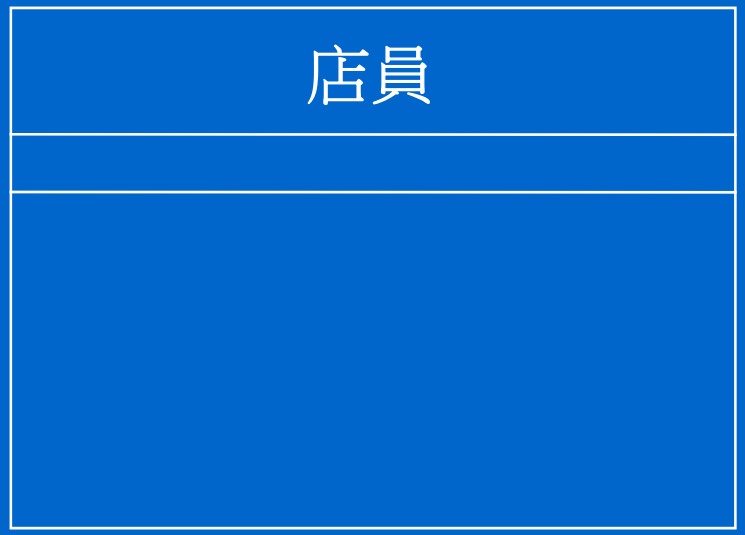
範例一 (cont'd)

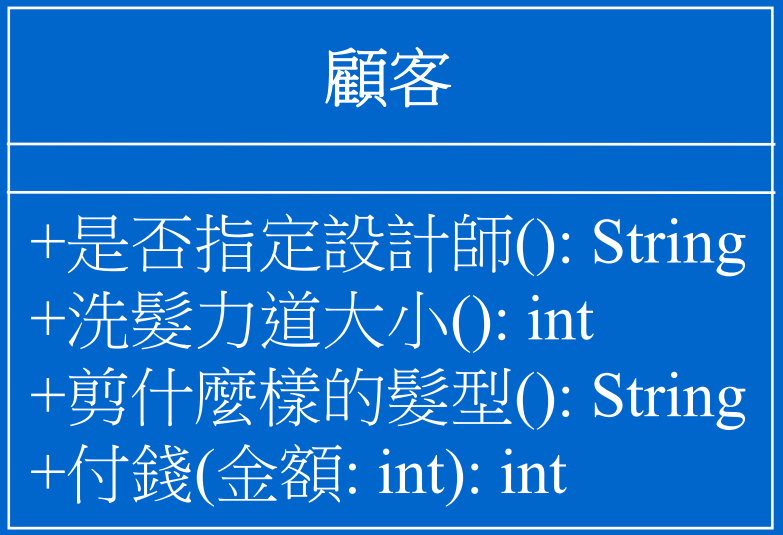
- 物件: 店員, 設計師, 顧客 (帳單, 發票)
- 類別圖:



範例一 (cont'd)

- 物件: 店員, 設計師, 顧客 (帳單, 發票)
- 類別圖:





範例一 (cont'd)

- 物件: 店員, 設計師, 顧客 (帳單, 發票)
- 類別圖:

範例一 (cont'd)

- 物件: 店員, 設計師, 顧客 (帳單, 發票)
- 類別圖:

範例一 (cont'd)

- 物件: 店員, 設計師, 顧客 (帳單, 發票)
- 類別圖:

範例一 (cont'd)

- 物件: 店員, 設計師, 顧客 (帳單, 發票)
- 類別圖:

店員
+可以馬上剪嗎(): boolean +洗頭髮(): void +多少錢(帳單: int): int +找錢(金額: int): int

顧客
+是否指定設計師(): String +洗髮力道大小(): int +剪什麼樣的髮型(): String +付錢(金額: int): int

範例一 (cont'd)

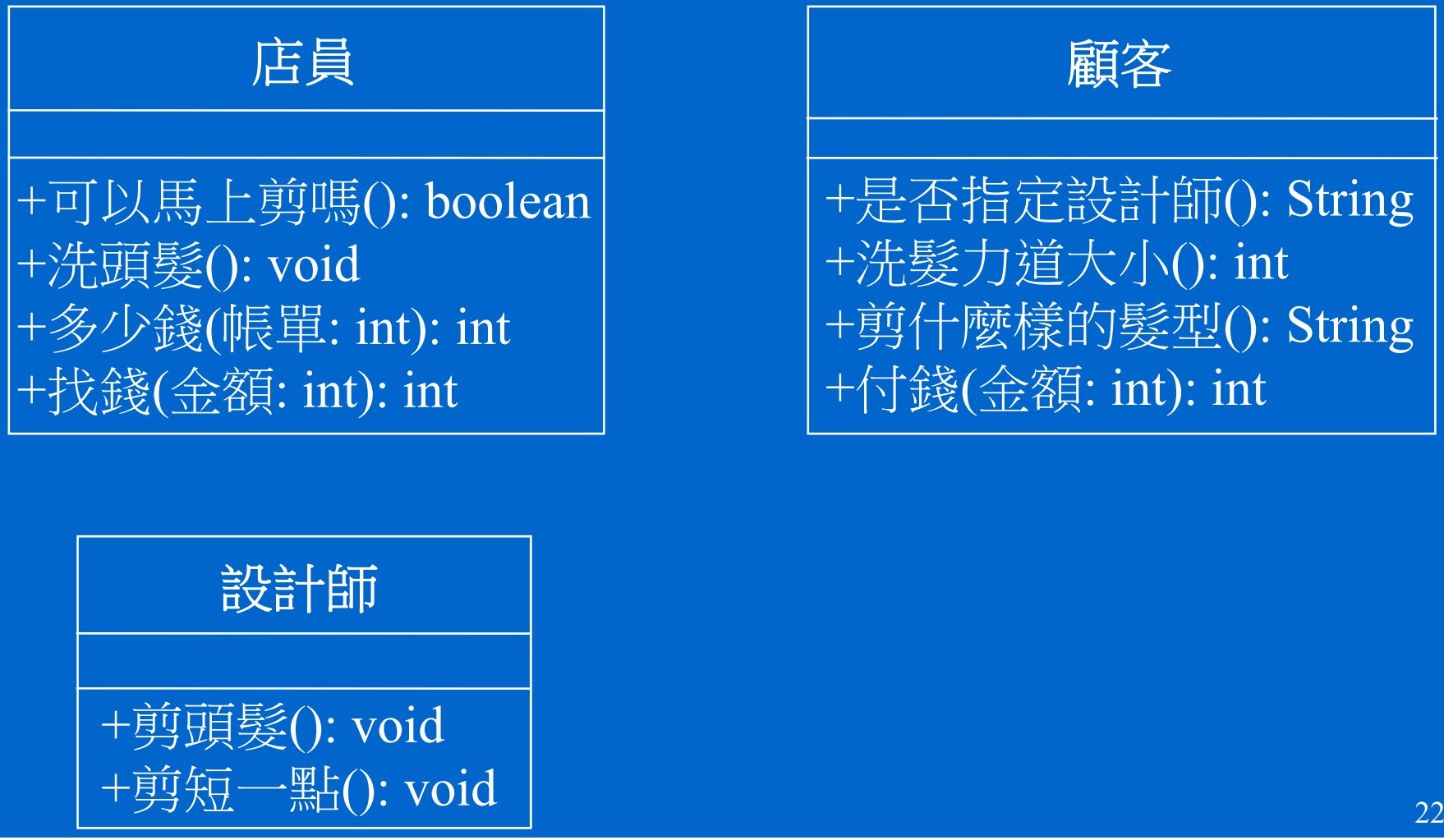
- 物件: 店員, 設計師, 顧客 (帳單, 發票)
- 類別圖:

範例一 (cont'd)

- 物件: 店員, 設計師, 顧客 (帳單, 發票)
- 類別圖:

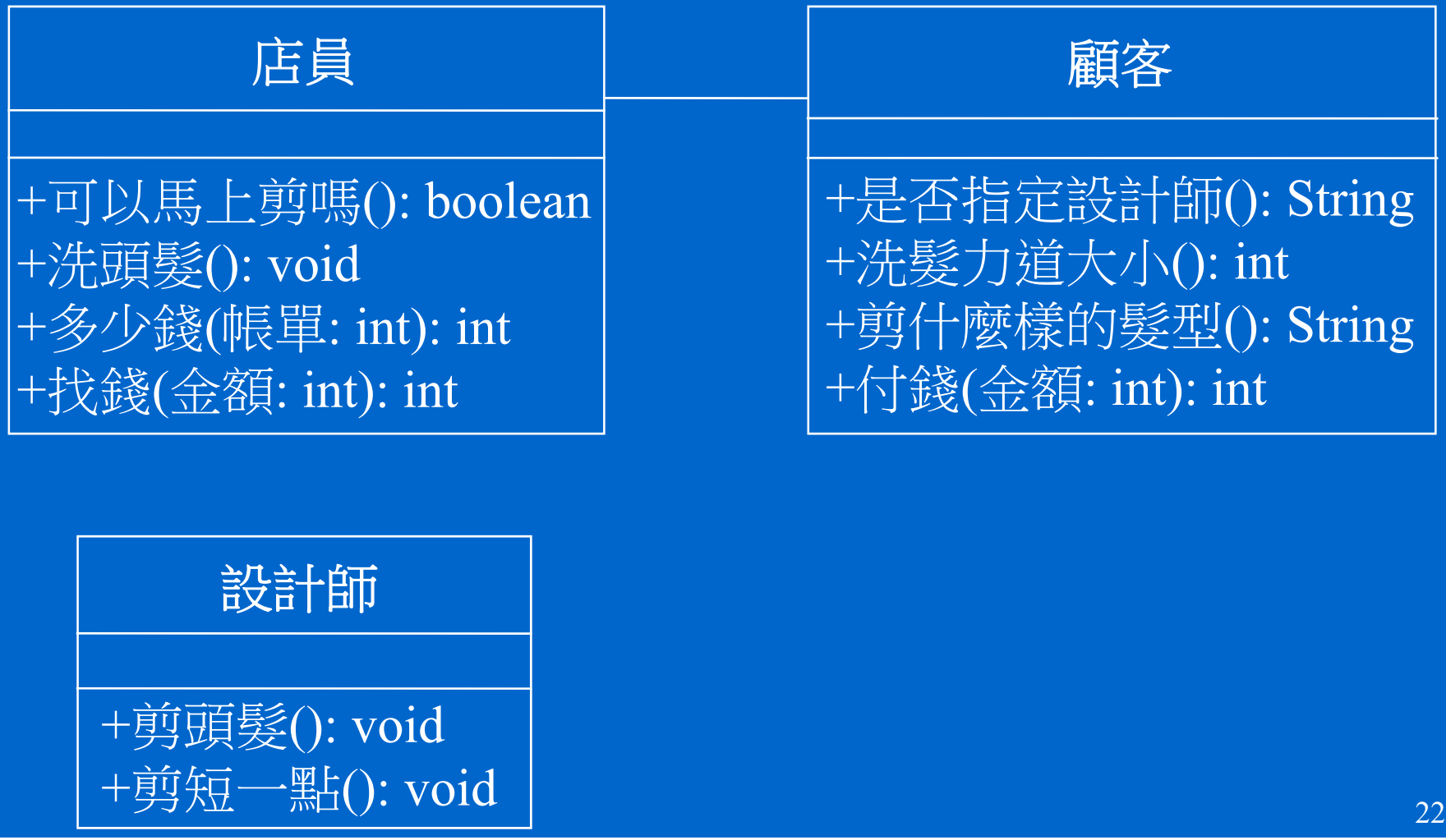
範例一 (cont'd)

- 物件: 店員, 設計師, 顧客 (帳單, 發票)
- 類別圖:



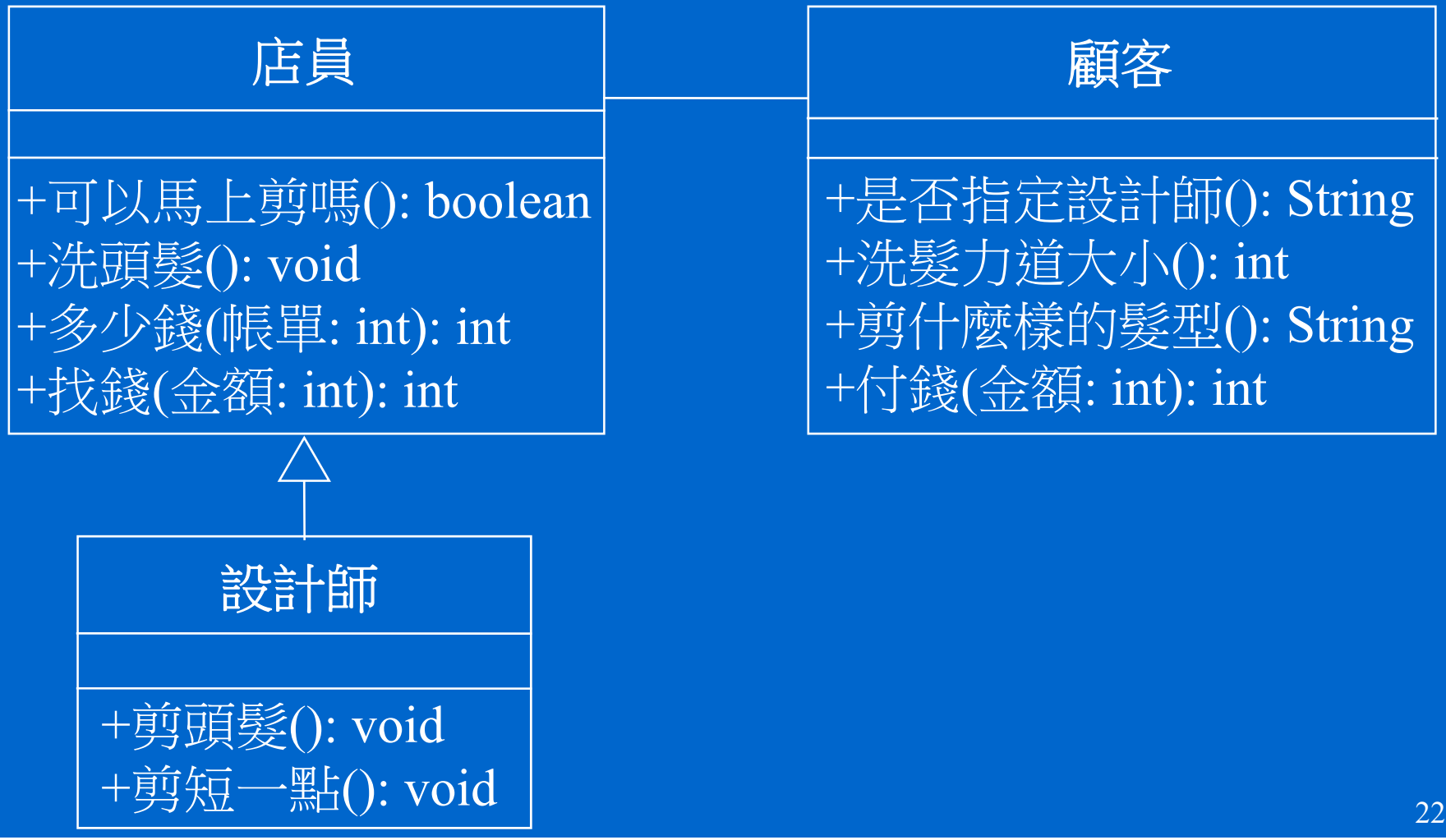
範例一 (cont'd)

- 物件: 店員, 設計師, 顧客 (帳單, 發票)
- 類別圖:



範例一 (cont'd)

- 物件: 店員, 設計師, 顧客 (帳單, 發票)
- 類別圖:



範例二

- 賣場裡販賣各種電腦零件：主機板、記憶體、螢幕、**CPU** 等等

範例二

- 賣場裡販賣各種電腦零件：主機板、記憶體、螢幕、**CPU** 等等
- 顧客購買電腦零件，如果是會員可以打八折，但特價品不打。每週會選部分產品為特價品，特價方式有兩種：打八五折或買二送一。

範例二

- 賣場裡販賣各種電腦零件：主機板、記憶體、螢幕、**CPU** 等等
- 顧客購買電腦零件，如果是會員可以打八折，但特價品不打。每週會選部分產品為特價品，特價方式有兩種：打八五折或買二送一。
- 店員薪水有兩種：時薪制與銷售額抽成計酬制。

範例二

- 賣場裡販賣各種電腦零件：主機板、記憶體、螢幕、**CPU** 等等
- 顧客購買電腦零件，如果是會員可以打八折，但特價品不打。每週會選部分產品為特價品，特價方式有兩種：打八五折或買二送一。
- 店員薪水有兩種：時薪制與銷售額抽成計酬制。
  - 時薪制 - 依照工作時數給錢，

範例二

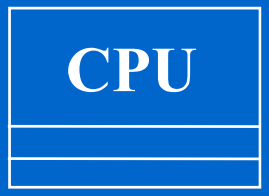
- 賣場裡販賣各種電腦零件：主機板、記憶體、螢幕、CPU 等等
- 顧客購買電腦零件，如果是會員可以打八折，但特價品不打。每週會選部分產品為特價品，特價方式有兩種：打八五折或買二送一。
- 店員薪水有兩種：時薪制與銷售額抽成計酬制。
  - 時薪制 - 依照工作時數給錢，
  - 銷售額抽成計酬制 - 根據賣出零件的價錢乘上一定的百分比為酬勞。

範例二

- 賣場裡販賣各種電腦零件：主機板、記憶體、螢幕、CPU 等等
- 顧客購買電腦零件，如果是會員可以打八折，但特價品不打。每週會選部分產品為特價品，特價方式有兩種：打八五折或買二送一。
- 店員薪水有兩種：時薪制與銷售額抽成計酬制。
  - 時薪制 - 依照工作時數給錢，
  - 銷售額抽成計酬制 - 根據賣出零件的價錢乘上一定的百分比為酬勞。
- 設計每次交易的金額、一天營業的總金額，以及兩個員工– 一為銷售額抽成計酬制一為時薪制 – 一天的薪水。

範例二 (cont'd)

範例二 (cont'd)

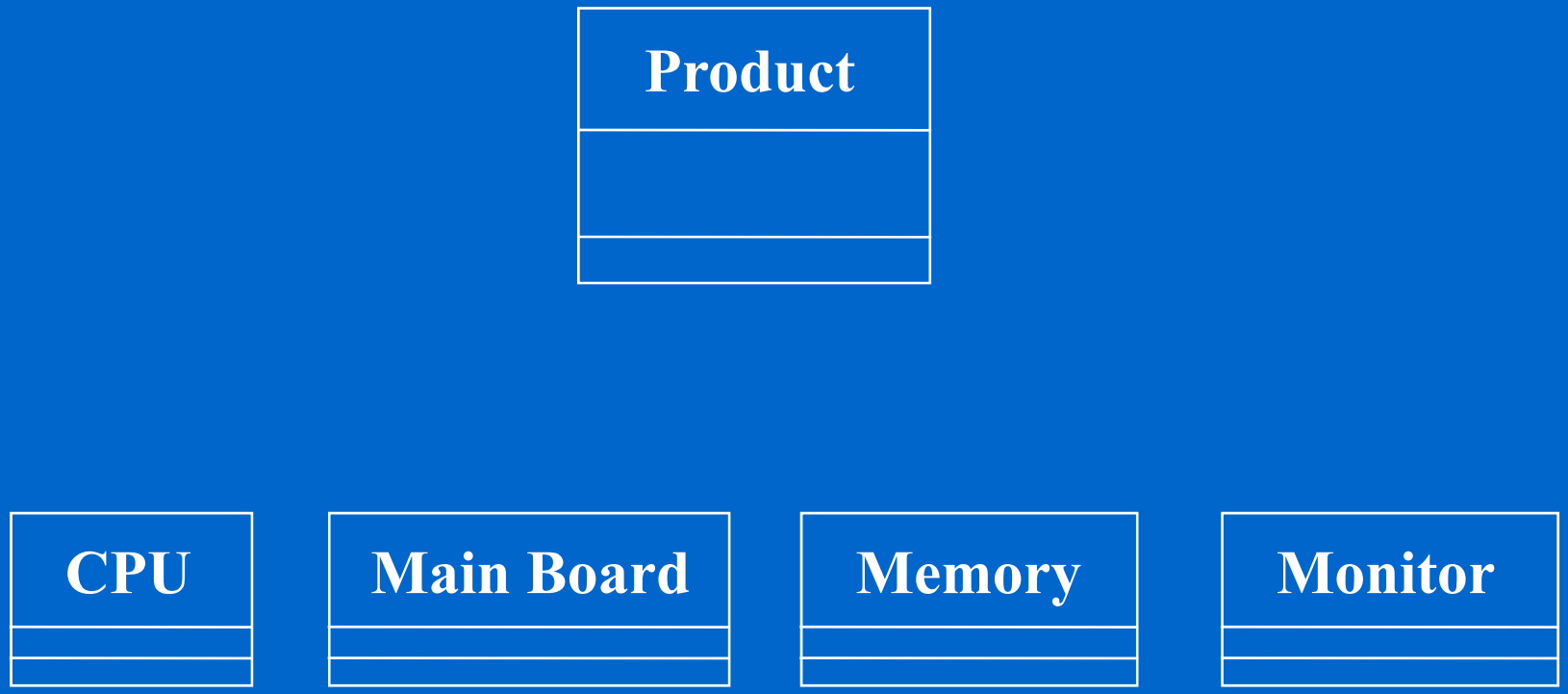


範例二 (cont'd)

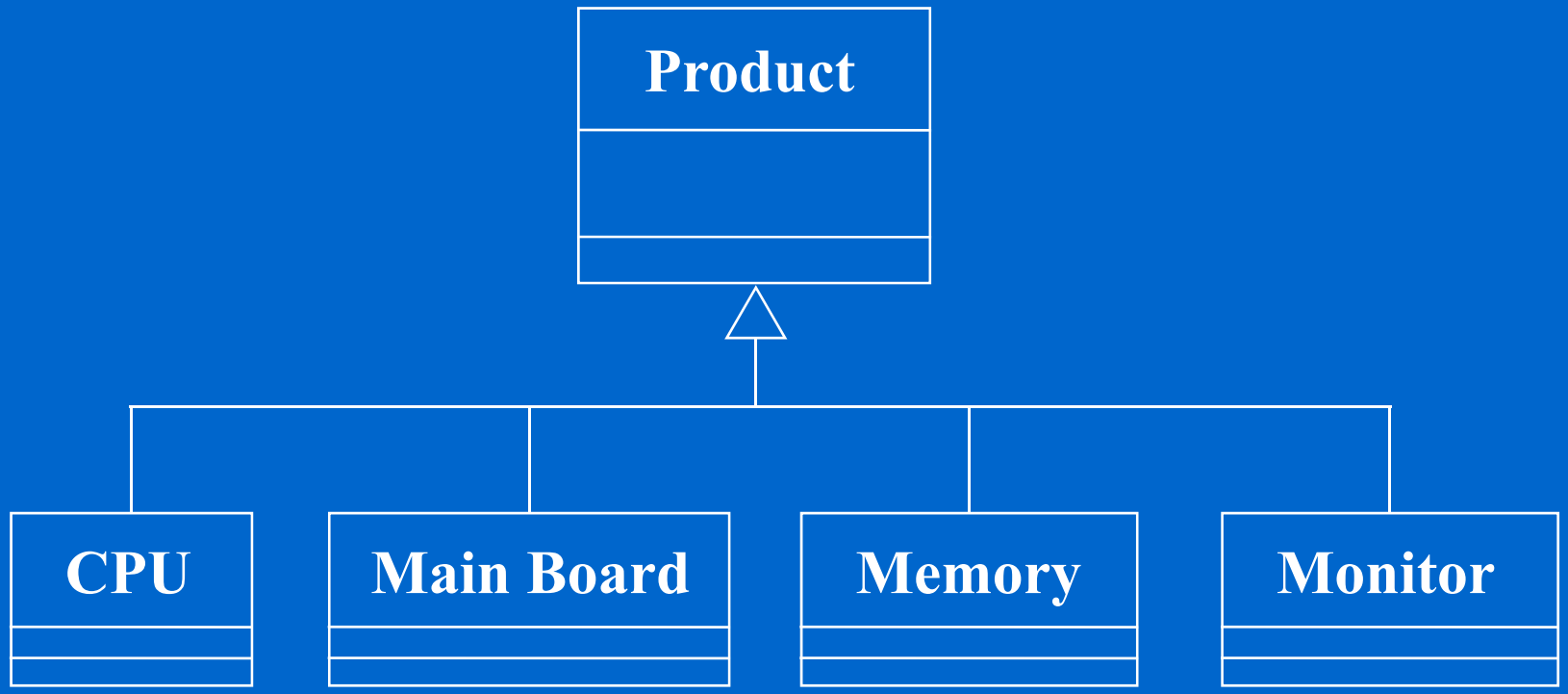
範例二 (cont'd)

範例二 (cont'd)

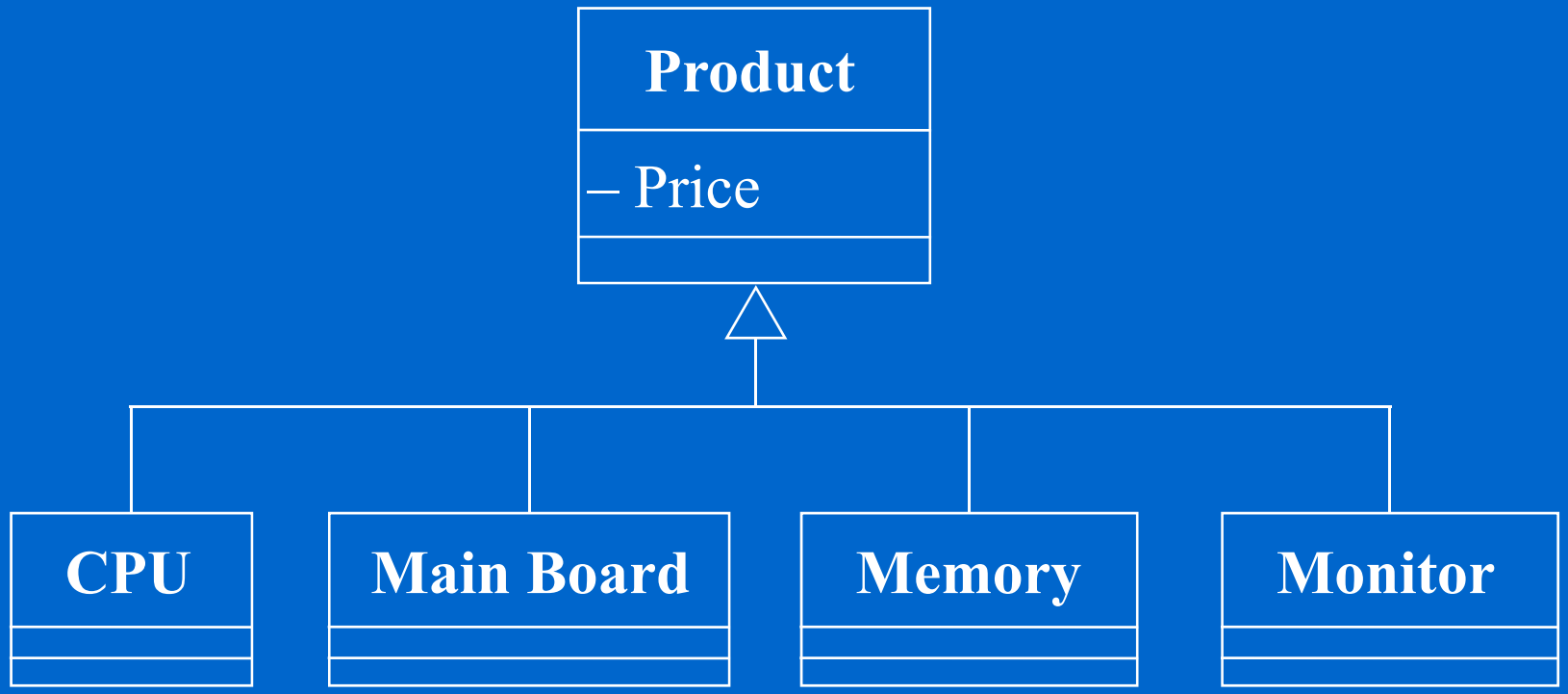
範例二 (cont'd)



範例二 (cont'd)



範例二 (cont'd)



範例二 (cont'd)

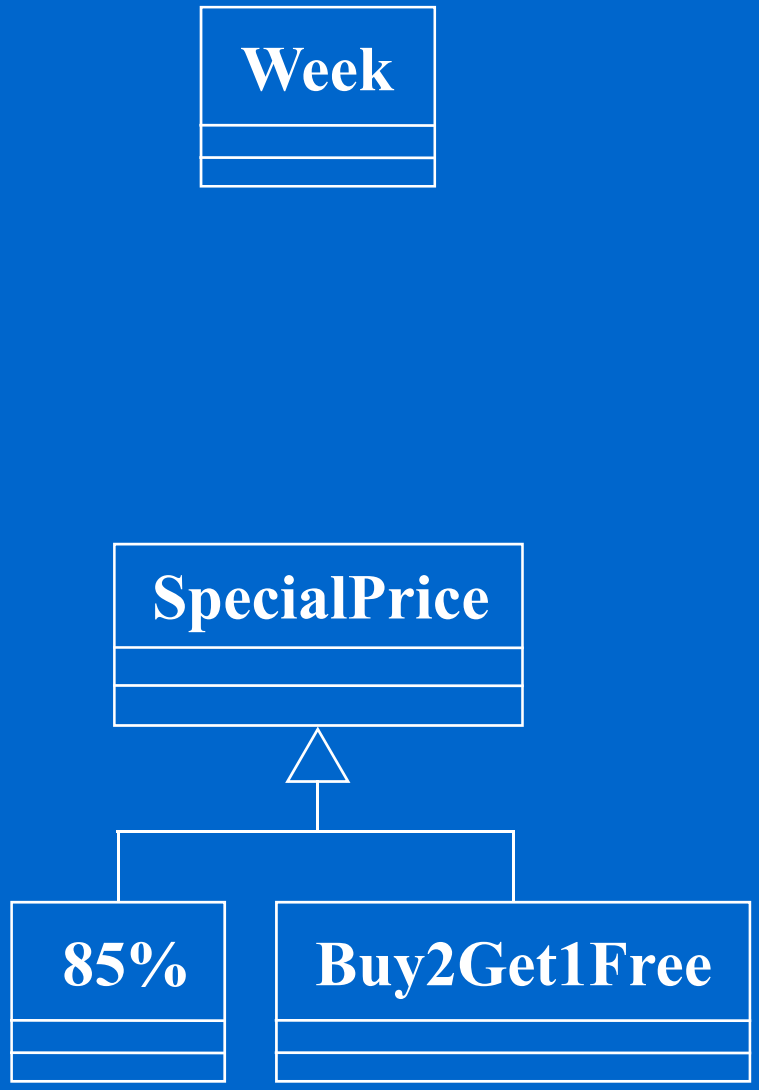
範例二 (cont'd)

範例二 (cont'd)

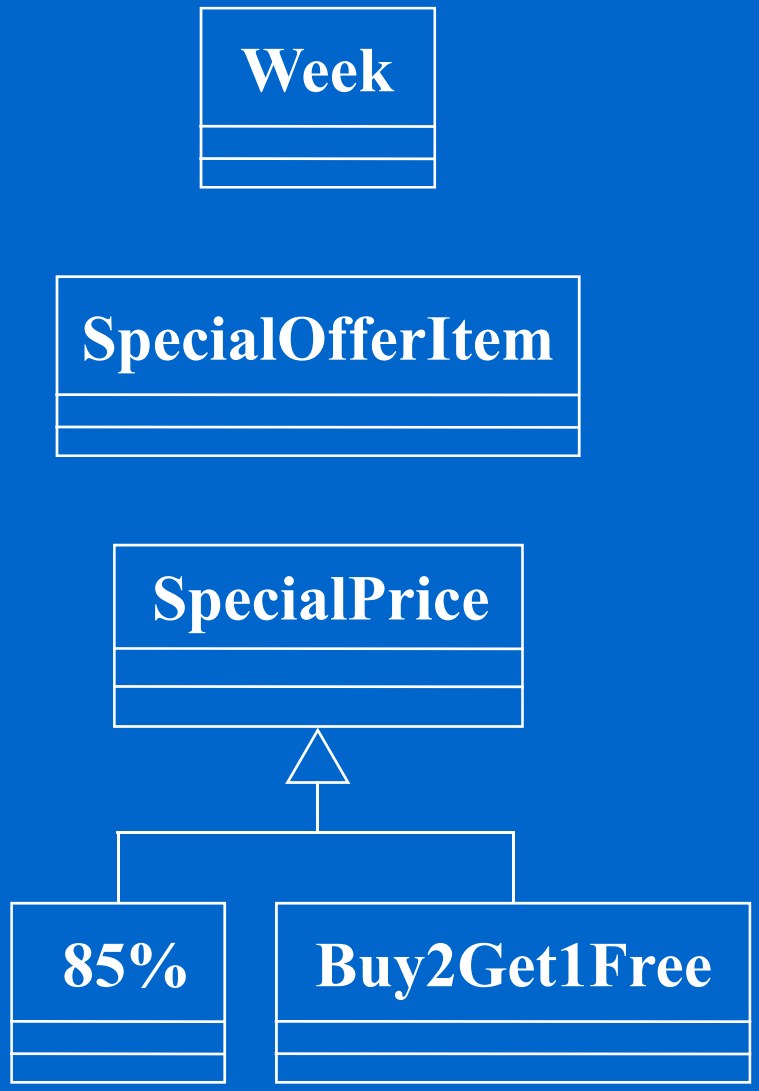
範例二 (cont'd)

範例二 (cont'd)

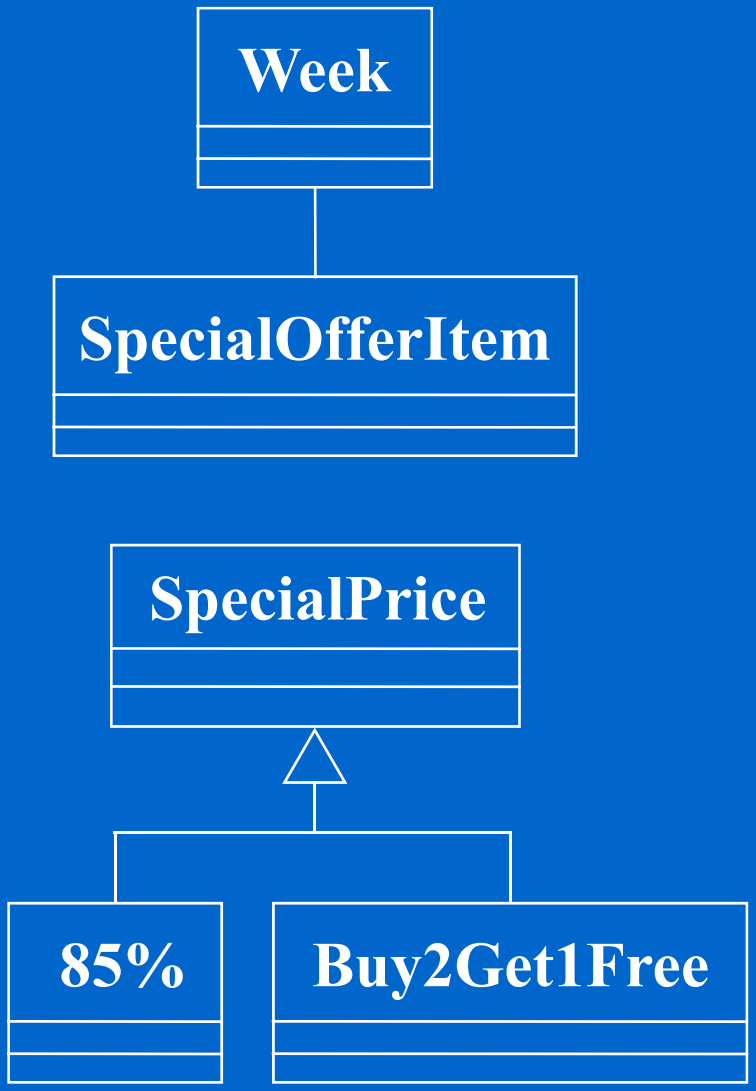
範例二 (cont'd)



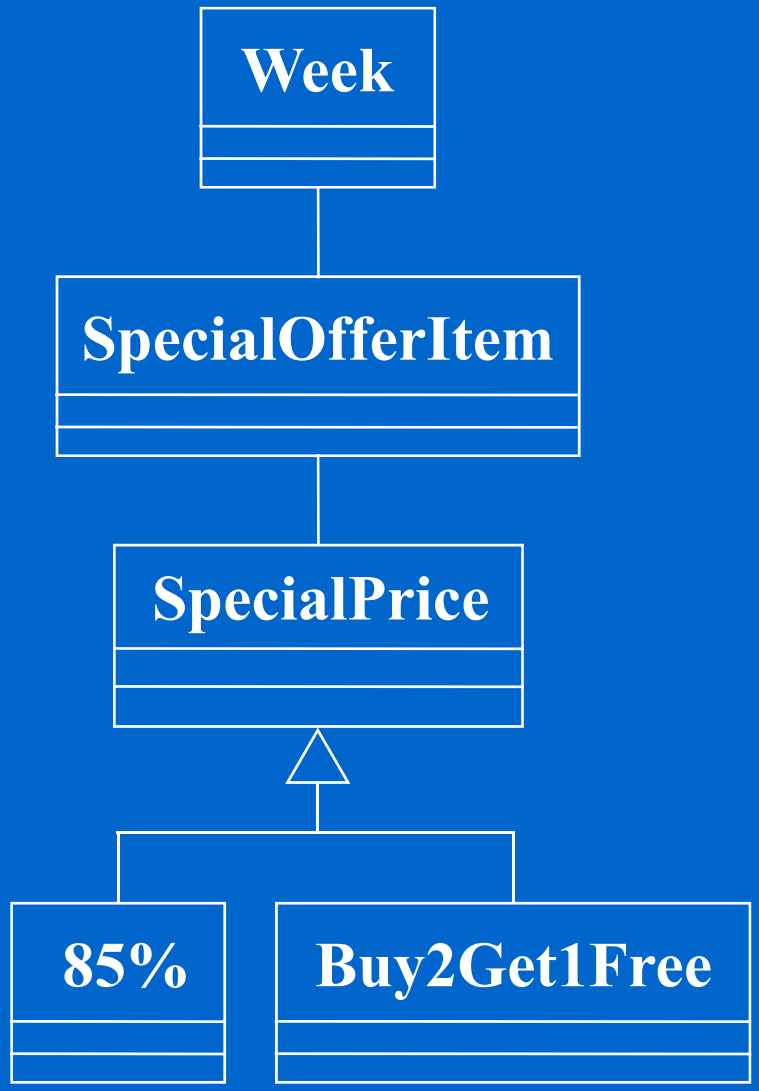
範例二 (cont'd)



範例二 (cont'd)



範例二 (cont'd)



範例二 (cont'd)

範例二 (cont'd)

範例二 (cont'd)

PerHour
– rate

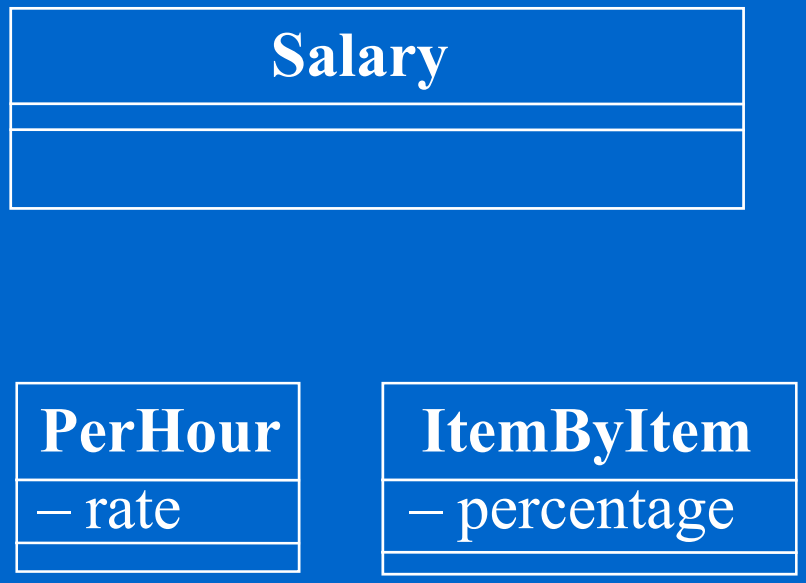
範例二 (cont'd)

範例二 (cont'd)

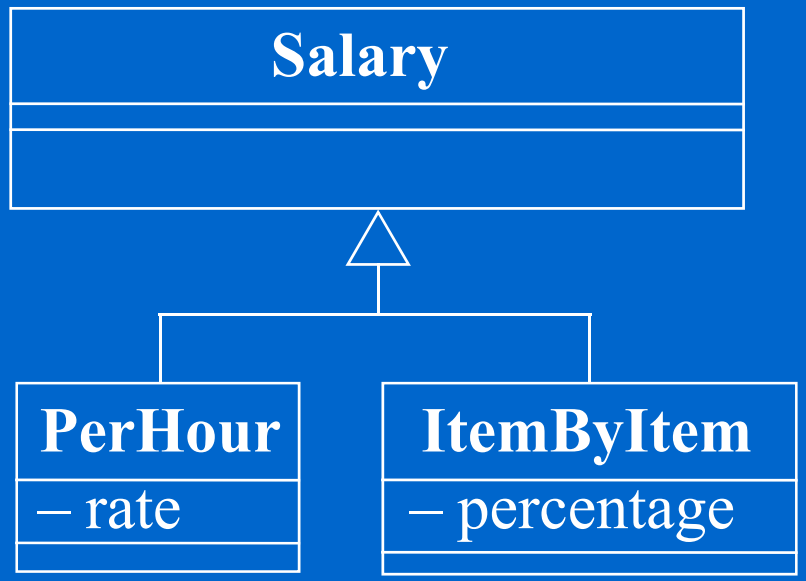
PerHour
– rate

ItemByItem
– percentage

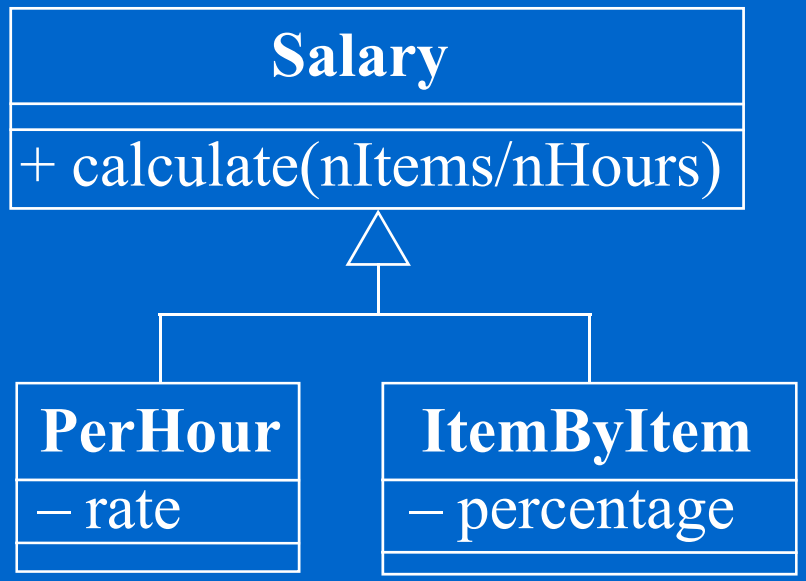
範例二 (cont'd)



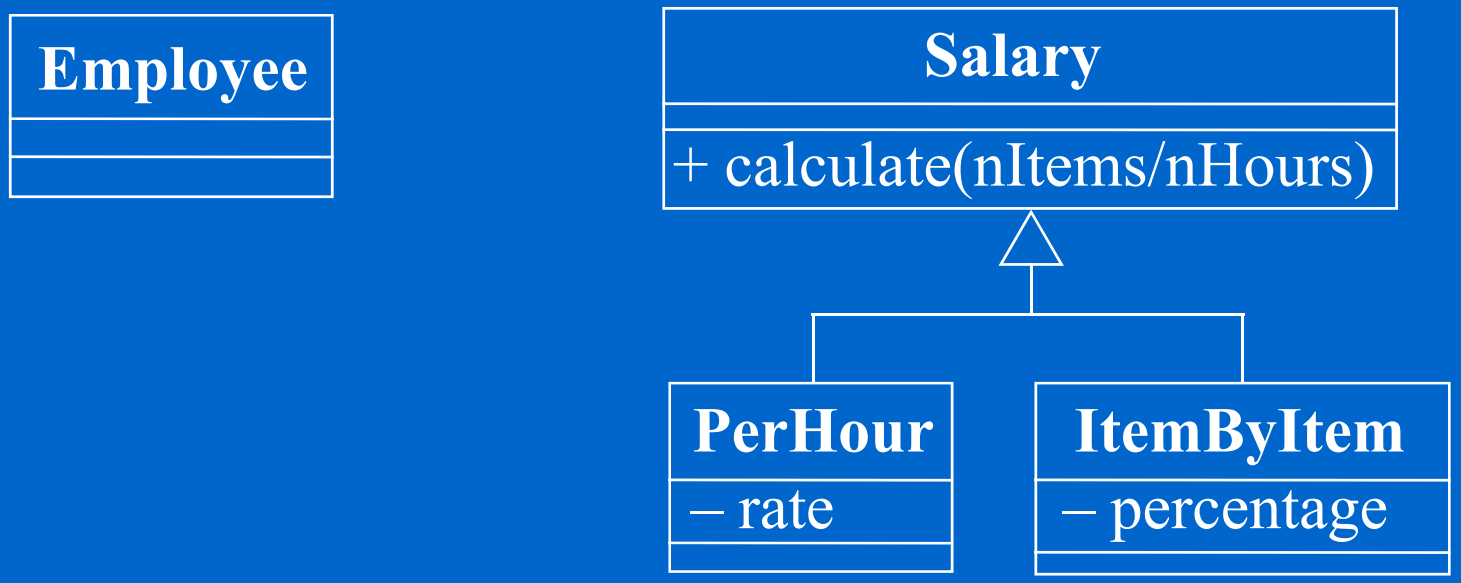
範例二 (cont'd)



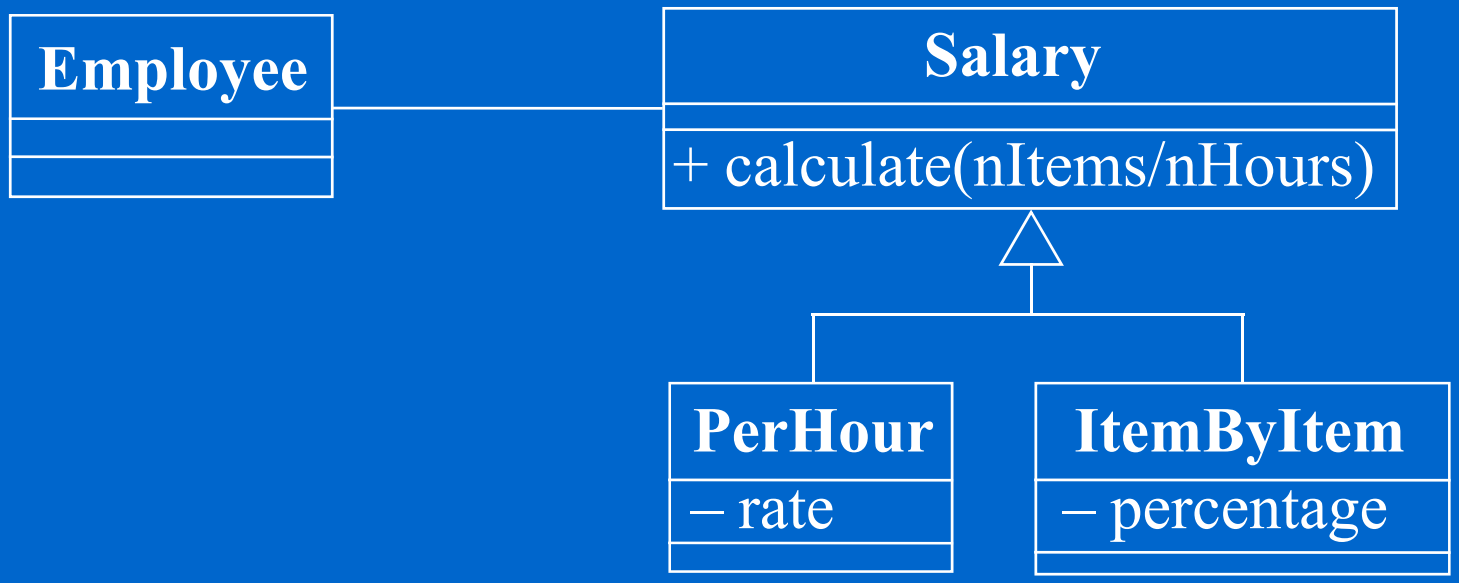
範例二 (cont'd)


Salary
+ calculate(nItems/nHours)
PerHour
– rate
ItemByItem
– percentage

範例二 (cont'd)


Employee
Salary
+ calculate(nItems/nHours)
PerHour
– rate
ItemByItem
– percentage

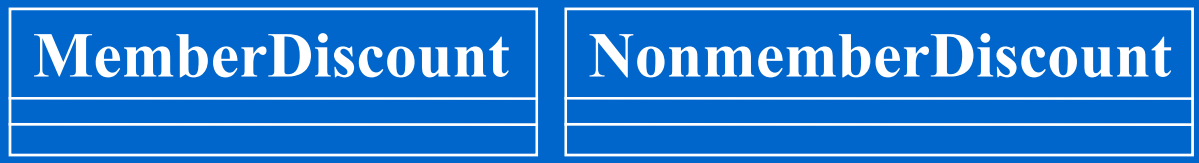
範例二 (cont'd)


Employee
Salary
+ calculate(nItems/nHours)
PerHour
– rate
ItemByItem
– percentage

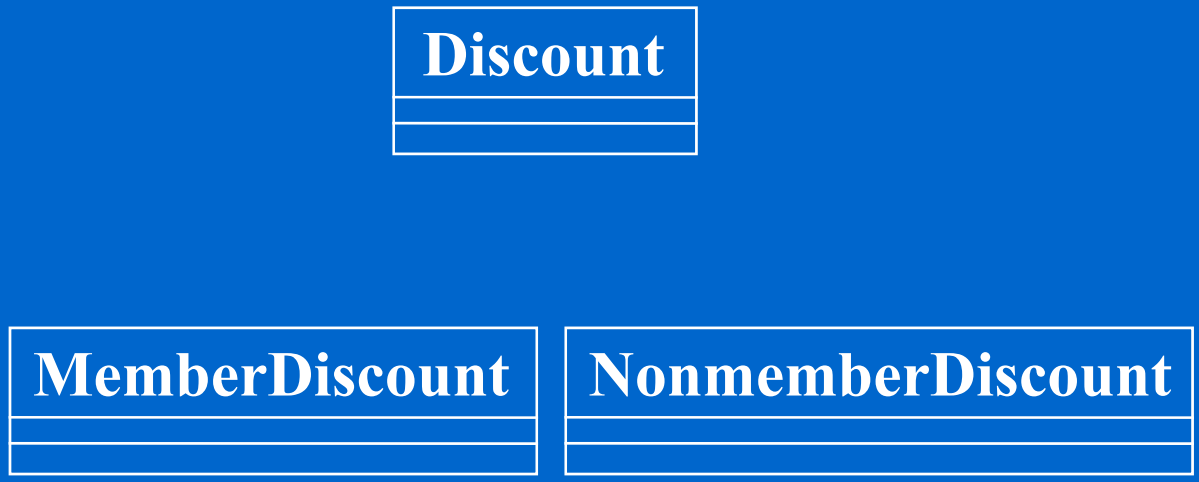
範例二 (cont'd)

範例二 (cont'd)

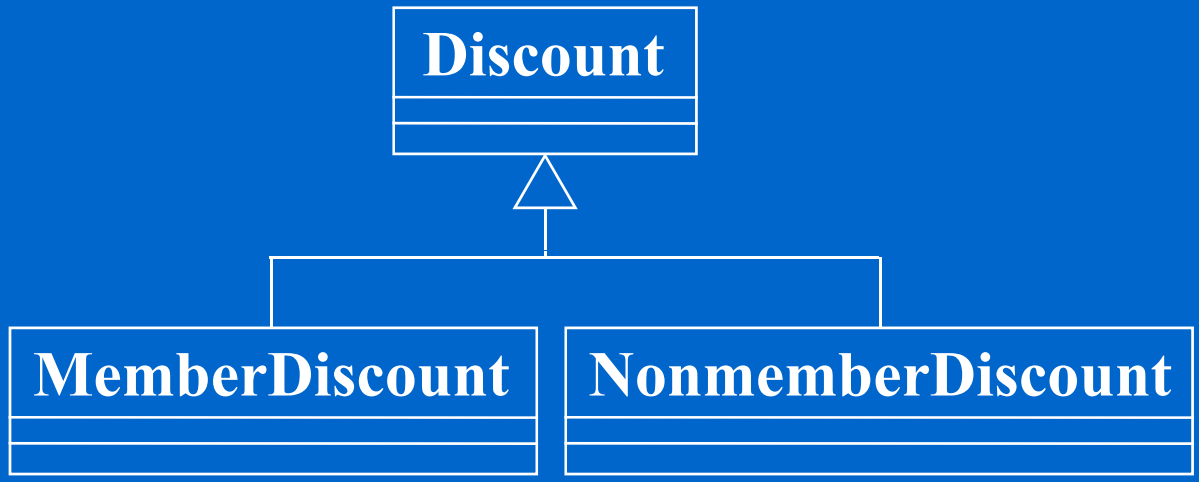
範例二 (cont'd)



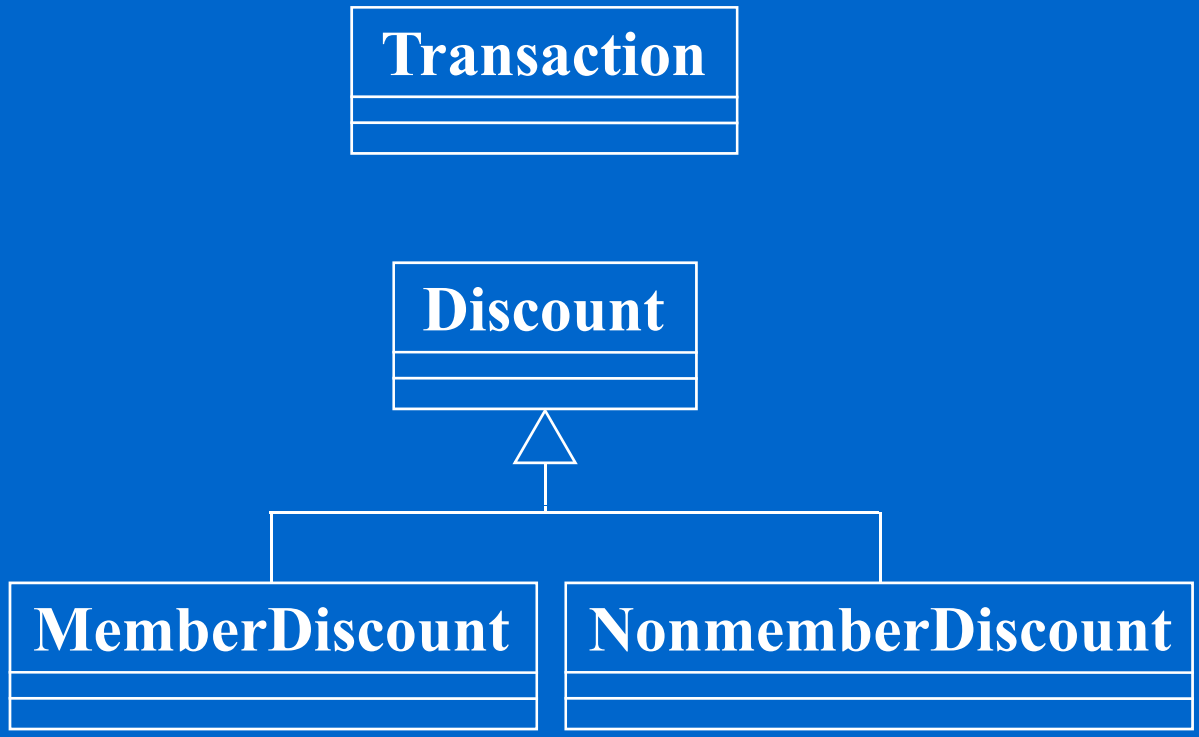
範例二 (cont'd)



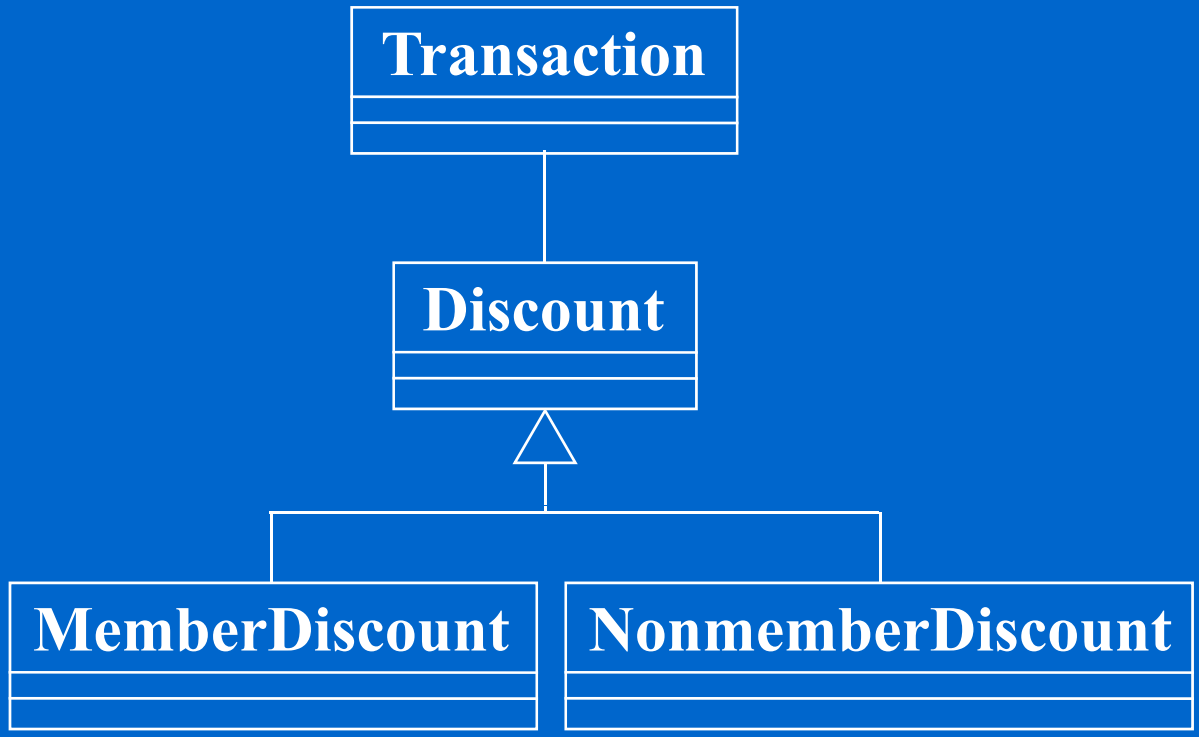
範例二 (cont'd)



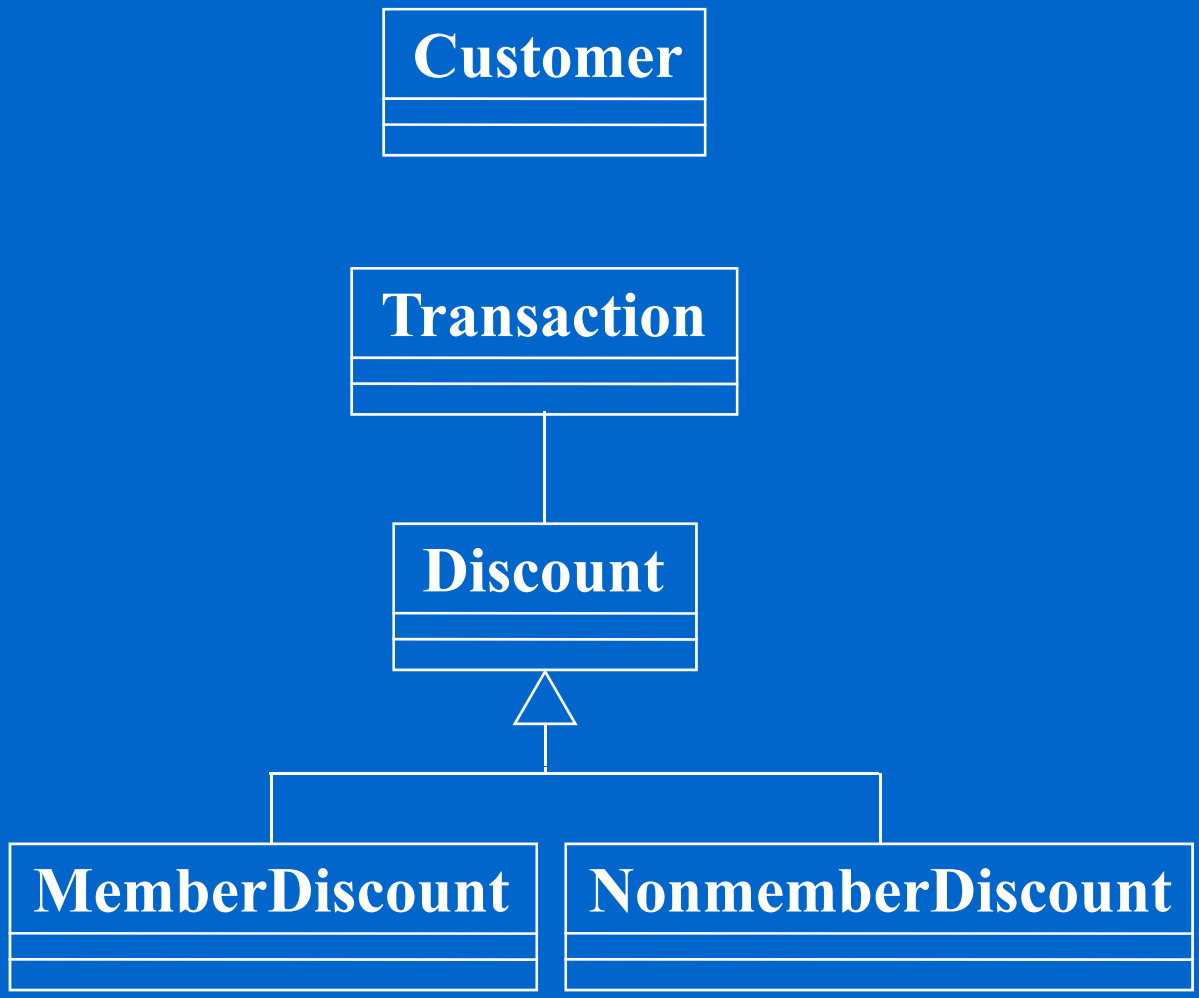
範例二 (cont'd)


Transaction
Discount
MemberDiscount
NonmemberDiscount

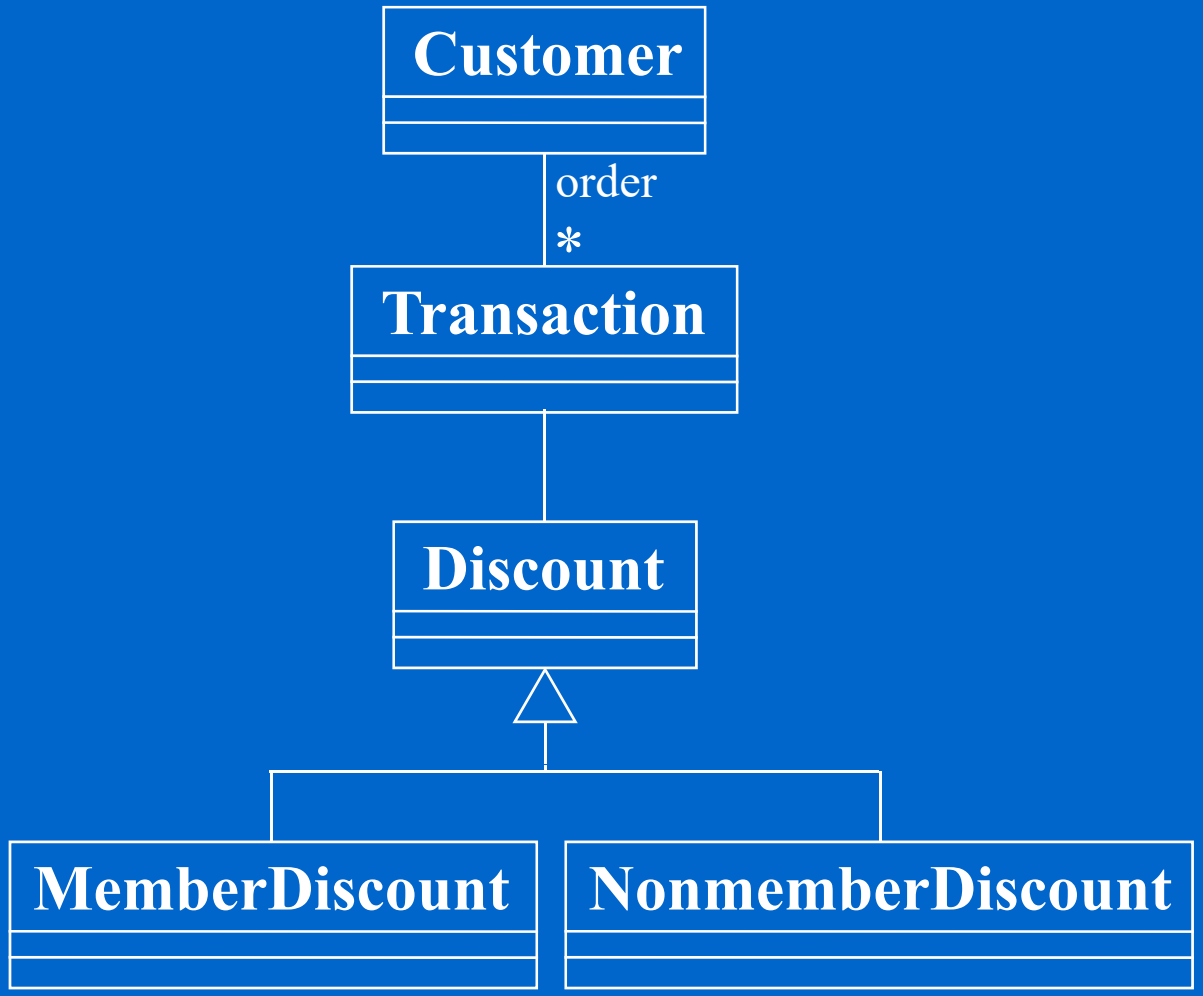
範例二 (cont'd)


Transaction
Discount
MemberDiscount
NonmemberDiscount

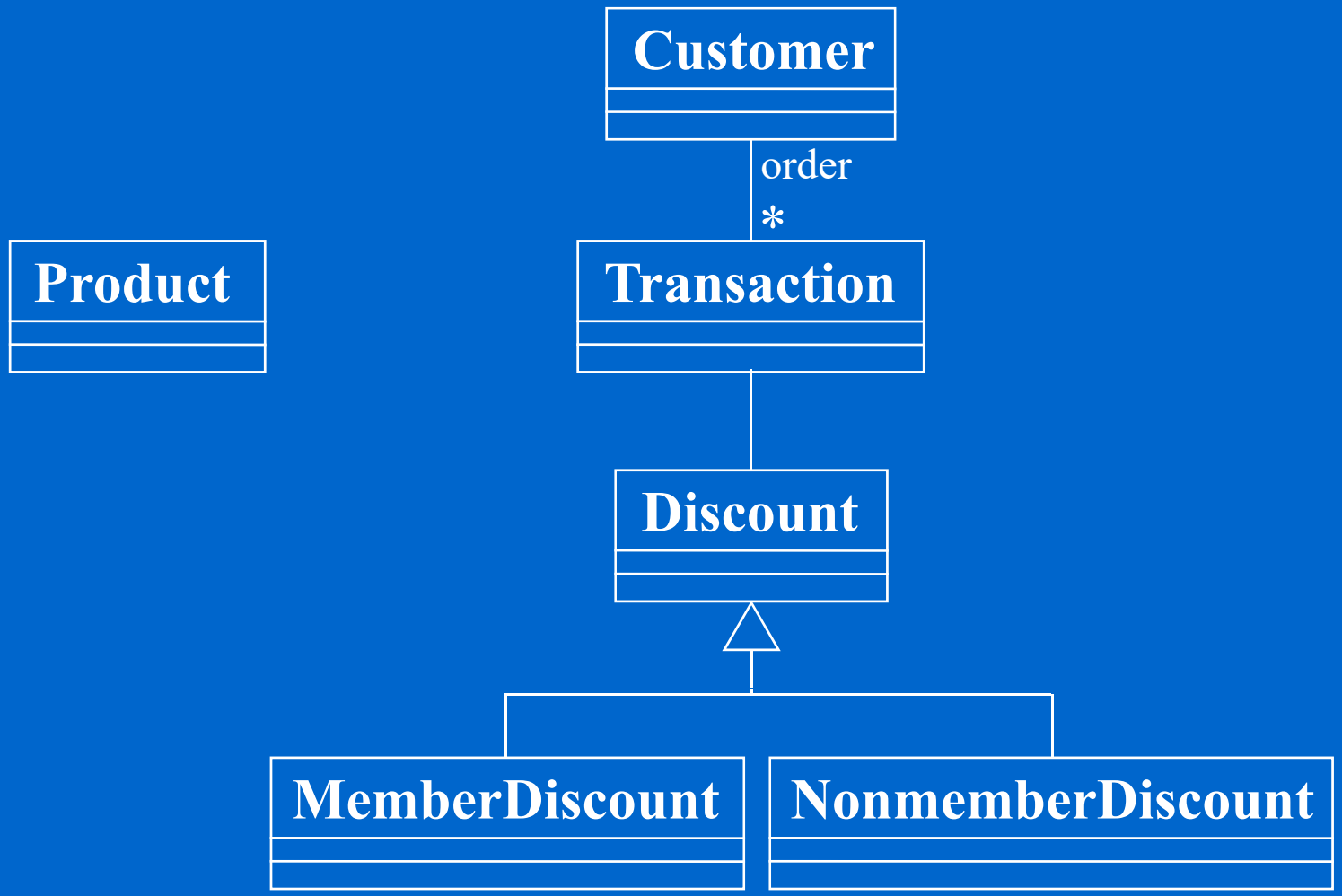
範例二 (cont'd)


Customer
Transaction
Discount
MemberDiscount
NonmemberDiscount

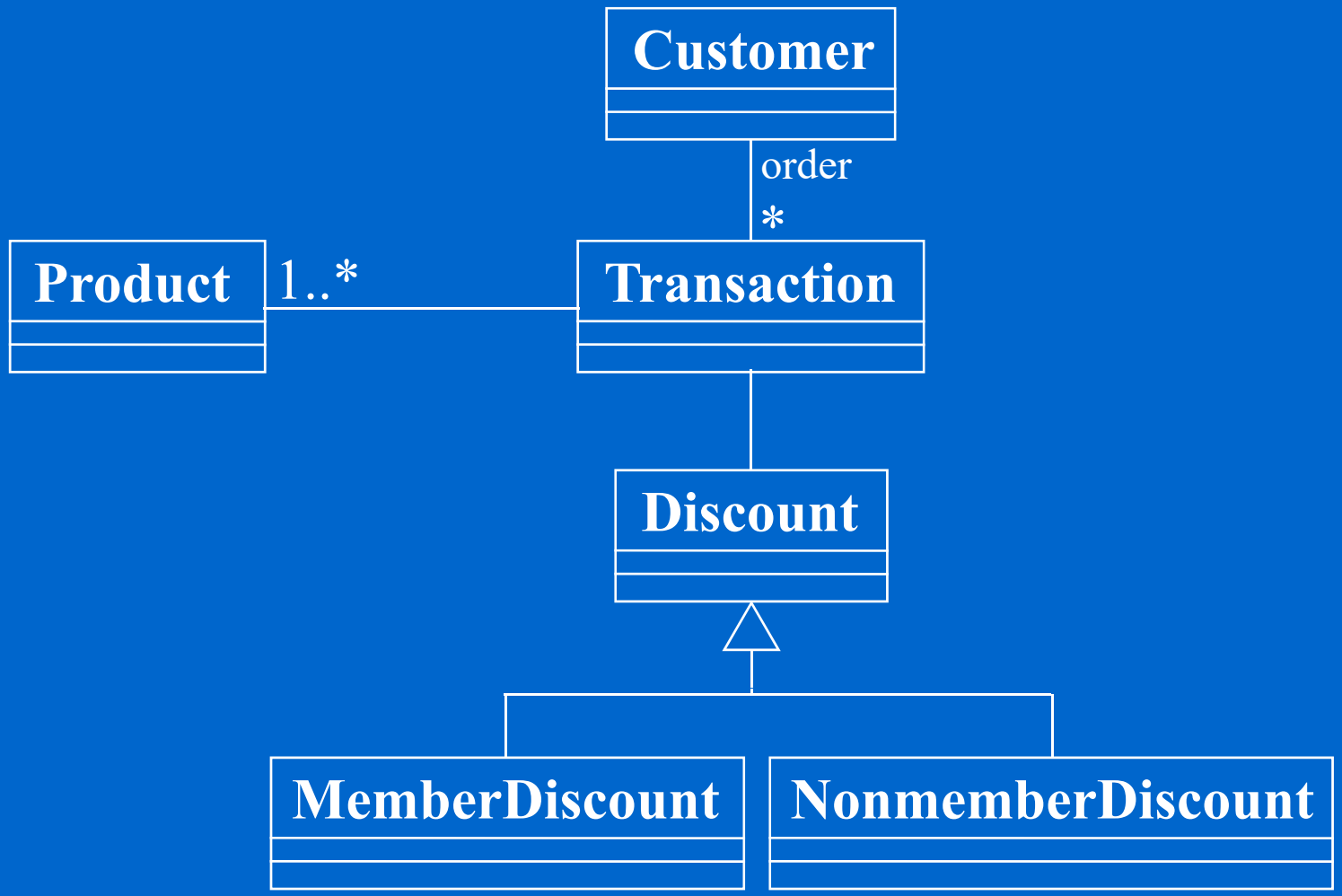
範例二 (cont'd)


Customer
order
*
Transaction
Discount
MemberDiscount
NonmemberDiscount

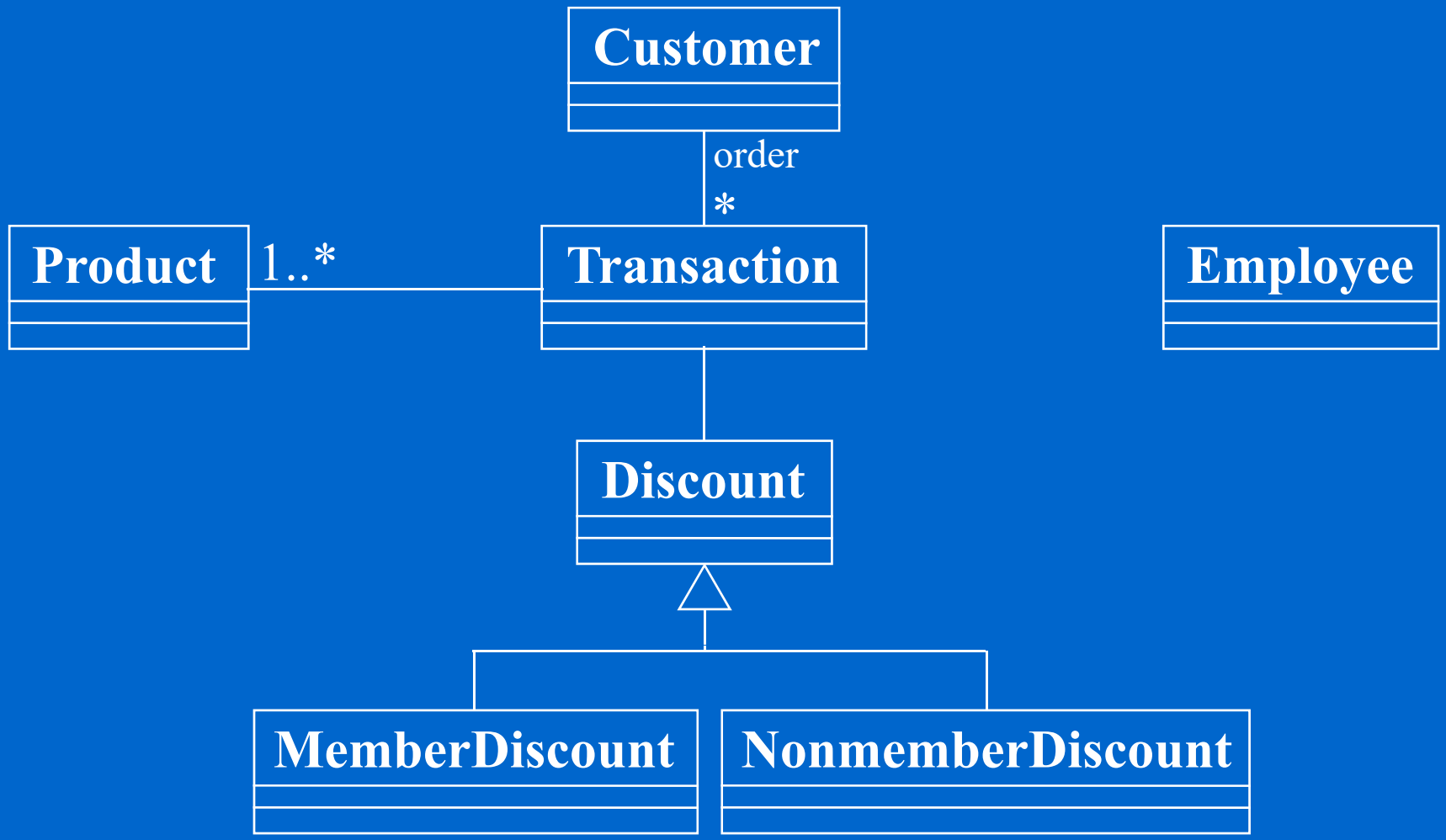
範例二 (cont'd)


Customer
order
*
Product
Transaction
Discount
MemberDiscount
NonmemberDiscount

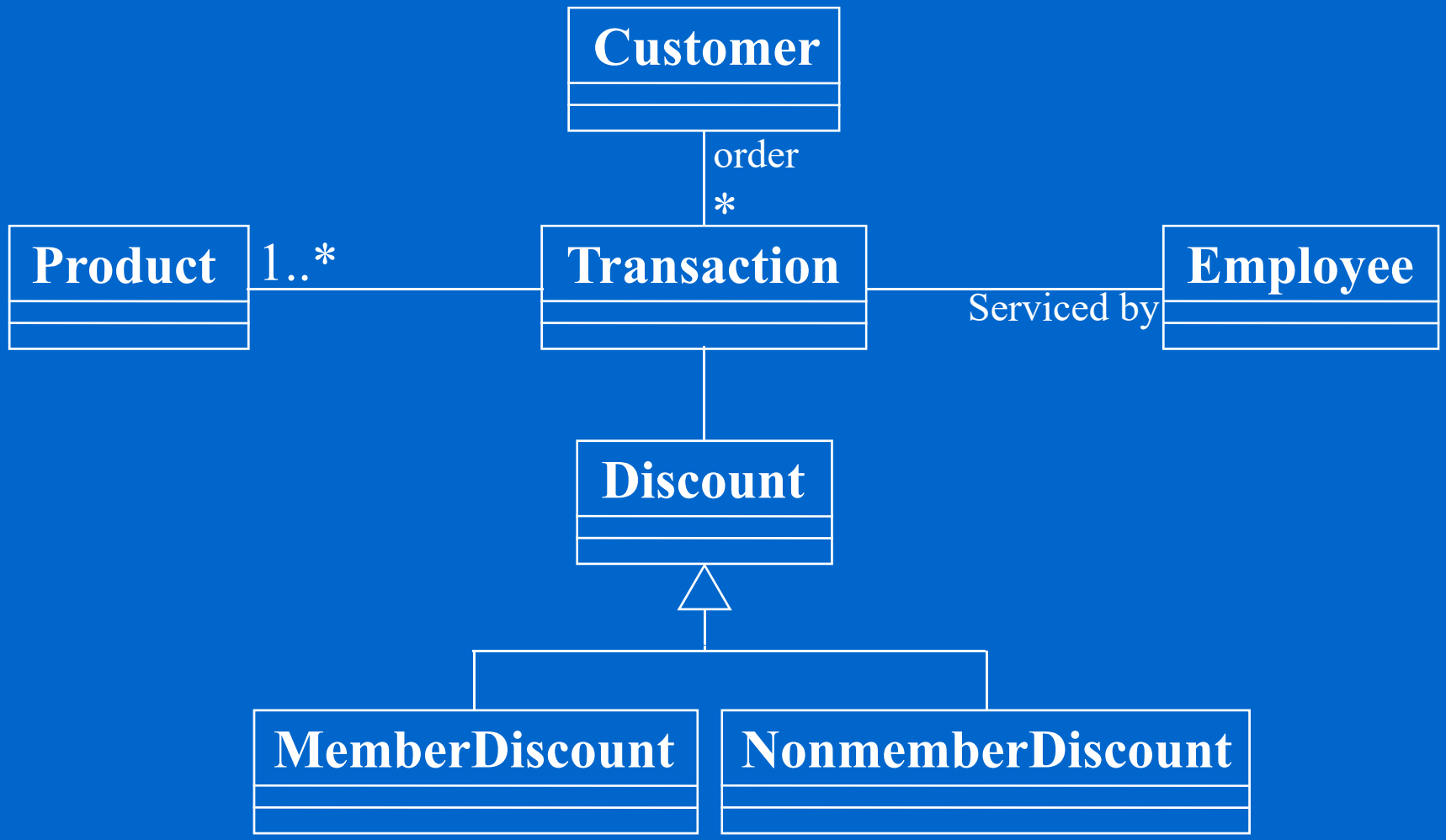
範例二 (cont'd)


Customer
order
*
Product
1..*
Transaction
Discount
MemberDiscount
NonmemberDiscount

範例二 (cont'd)

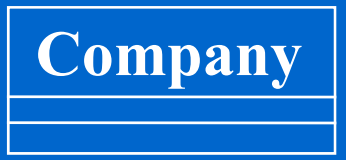

Customer
order
*
Product
1..*
Transaction
Employee
Discount
MemberDiscount
NonmemberDiscount

範例二 (cont'd)

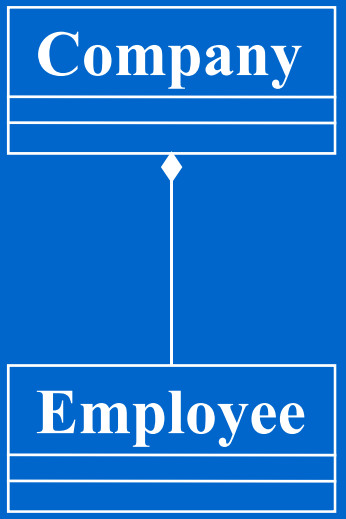

Customer
order
*
Product
1..*
Transaction
Serviced by
Employee
Discount
MemberDiscount
NonmemberDiscount

範例二 (cont'd)

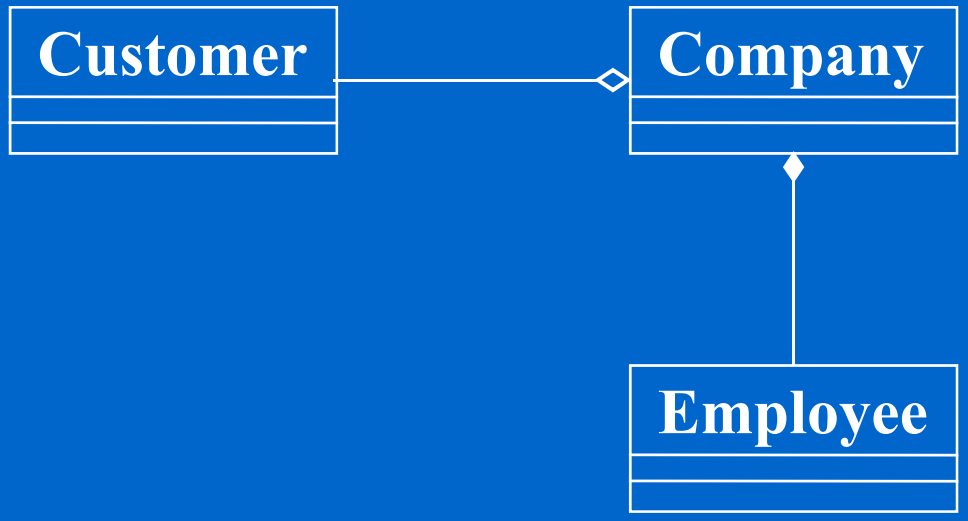
範例二 (cont'd)



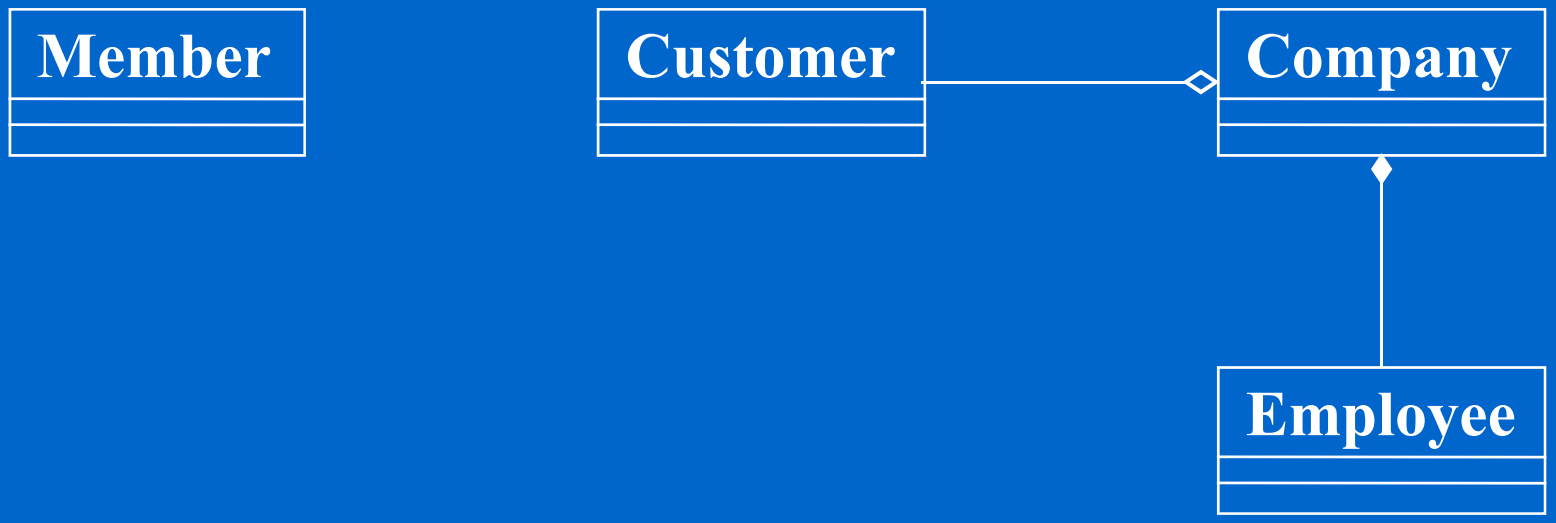
範例二 (cont'd)



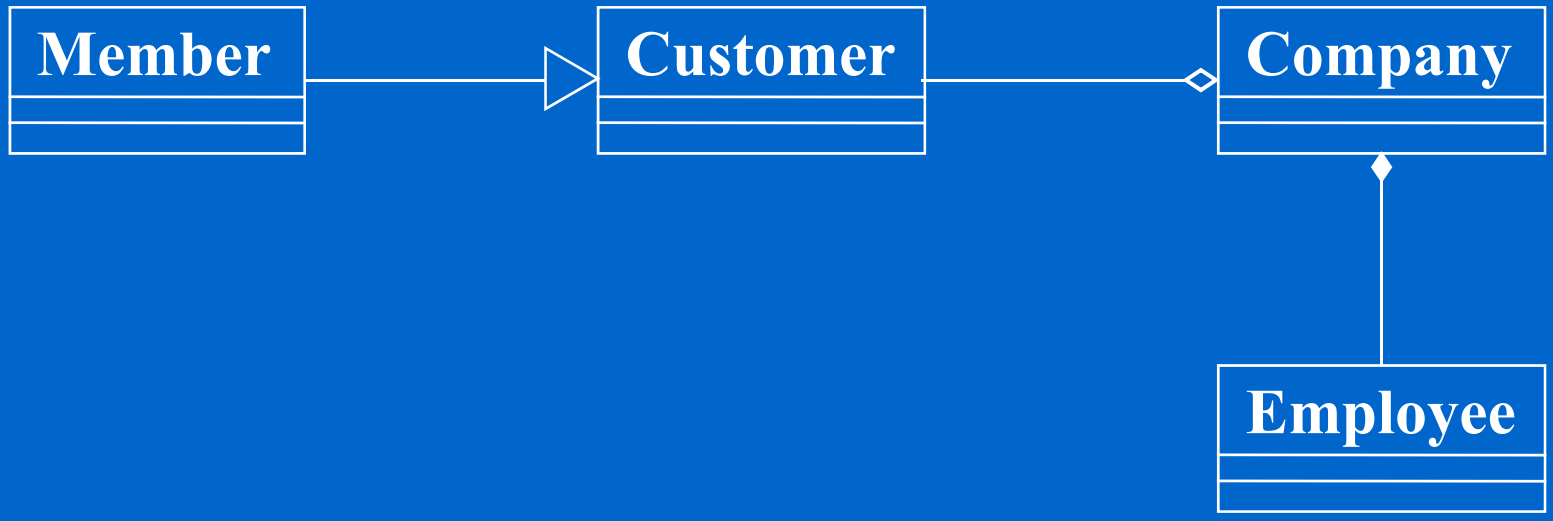
範例二 (cont'd)



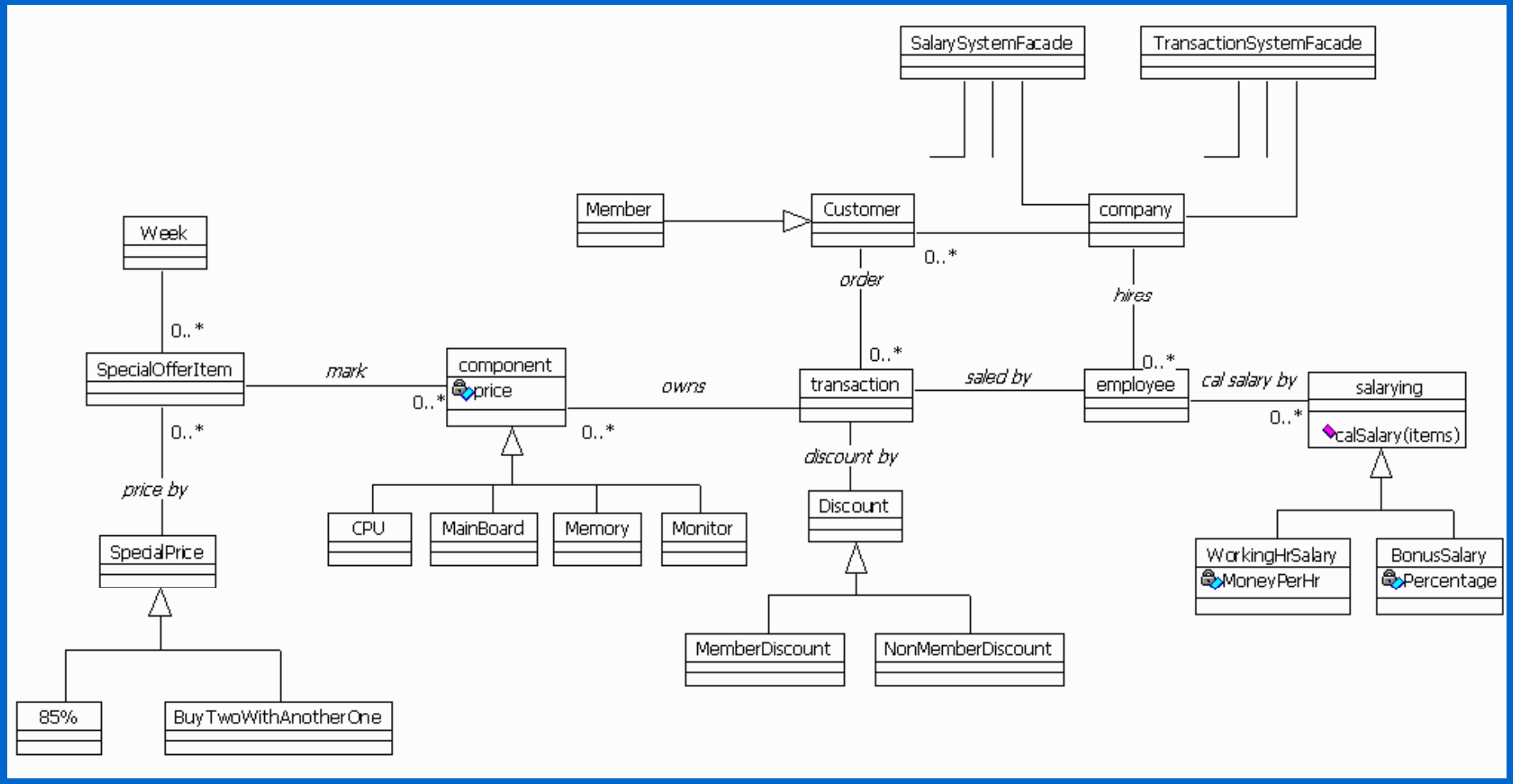
範例二 (cont'd)



範例二 (cont'd)



範例二 (cont'd)


SalarySystemFacade
TransactionSystemFacade
Member
Customer
company
Week
0..*
order
hires
SpecialOfferItem
mark
component
price
owns
transaction
saled by
employee
cal salary by
salarying
calSalary(items)
price by
discount by
Discount
CPU
MainBoard
Memory
Monitor
SpecialPrice
WorkingHrSalary
MoneyPerHr
BonusSalary
Percentage
MemberDiscount
NonMemberDiscount
85%
BuyTwoWithAnotherOne