Basic Object Design

C++ Object Oriented Programming
Pei-yih Ting
93/04 NTOU CS

Contents

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 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?

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 the system?

Object Oriented Analysis/Design

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

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" their name, address, age, courses they teach ...
- a student "knows" their 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

Object Behaviour

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

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

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

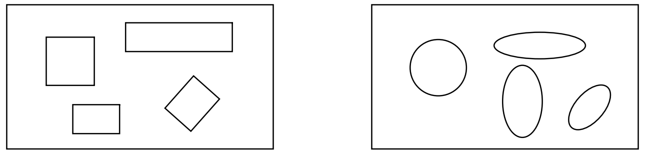
When applying an object-orientated design to a problem specification we identify objects, record their states and specify their behaviour.

Specifying Good Objects

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

Cohesion

- ✧ A good class describes a single abstraction



- ✧ Assume we are writing a networking mail 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.

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;
};
```

  - Why is this class not complete?
  - What would be desirable but not essential features?
- 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.

```
class Data {
public:
   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;
};
```

- This class is both inconsistent and unclear

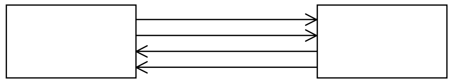
```
class Graphics {
   void drawLine(int x, int y);
   void movePen(int x, int y);
};
```

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

  - 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*.



```
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);
};

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

Reducing Coupling

- Encapsulation reduces coupling

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

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

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

- Avoid passing a great amount of data across object boundaries. Object should provide abstract and simple services.
- As opposed to the data flow design of application programs, in which data flow between processing units, object oriented/based programming tries to design objects that keeps and handles data intelligently. Put all responsible objects together for accomplishing a specific work without looking into their detailed processed data.

Design a Class 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

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.

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"

  "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." by Bjarne Stroustrup

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

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)

  "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.
- Ex: class Optimization Possible collaborators: Database, File

  optimize data

  store in file
- Expect change:

  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.

$$\begin{tikzcd}
\text{specification} \arrow[d, bend right] \\
\text{design} \arrow[u, bend right]
\end{tikzcd}$$