模块化与信息隐藏

刘钦

知识点

- 模块化与信息隐藏（P35-P48）
- 结构化的耦合（P70,71，82,85）
- 耦合的种类（P91）
- 结构化的内聚（P93）
- 内聚的种类（P95）
- 模块化思想的应用（P96-P99）
- Module Guide（P101-103）
- 信息隐藏思想的应用（P108,109）
- 原则1 全局变量有害
- 原则2 优先显示表达
- 原则3 不要重复
- 原则4 按接口编程

```
Public class Rous{

  public static int findPattern( String text, String pattern)
    { … }
  public static int average( Vector numbers )
    { … }
  public static OutputStream openFile( String fileName )
    { … }
}
```

? 内聚

```
public class Employee{
        public String name, emailID.
  ...
}


public class Emailer{
        public void sendEmail(Employee e, String text)
        {...}
  ...
}
```

？耦合

Outline

- 模块化与信息隐藏思想
- 结构化的模块化
- 结构化的信息隐藏

Outline

- 模块化与信息隐藏思想
  - 动机
  - 发展
  - 模块化与信息隐藏
  - KWIC案例
- 结构化的模块化
- 结构化的信息隐藏

动机

Design “Good” Software

- What does “good” stand for?

Parnas1972

- Managerial
- Product flexibility
- Comprehensibility

- 特征
  - allow one module to be written with little knowledge of the code in another module
  - allow modules to be reassembled and replaced without reassembly of the whole system.

Stevens1974

- 简洁性（Simplicity）
  - 易于调试
  - 易于分解
- 可观察性（Observability）
  - 易于修改

Beohm1976

- Maintainability
- Extendibility
- Understandability
- Reusability

发展

背景

- 1960s
  - Software Is Not Like Hardware
  - Software Crafting
- 1970s
  - Software = Data + Algorithm
  - Water Fall Process Model
  - Formal Method
- 1980s
  - Reuse
  - Object
  - Peopleware

历史发展

- 萌芽
  - Wirth1971; Parnas1972;
- 形成
  - Stevens1974; Parnas1978; Parnas1985;
- 发展
  - Eder1992; Hitz1995;
- 反思
  - McConnell1996; Demarco2002

Wirth1971

- “Program Development by Stepwise Refinement”
- 核心思想
  - 逐步求精
- The program is gradually developed in a sequence of refinement steps
  - To decompose decisions as much as possible
  - To untangle aspects which are only seemingly interdependent
  - To defer those decision which concern details of representation as long as possible

Wirth1971

- Refinement of the description of program and data structure should proceed in parallel.
- The degree of modularity obtained in this way will determine the ease or difficulty with which a program can be adapted to changes or extensions of the purpose or changes in the environment.
- Each refinement implies a number of design decisions based on a set of design criteria.
- Careful programming is not a trivial subject.

Parnas1972

- “On the Criteria To Be Used in Decomposing Systems into Modules”
- What is Modularization?
  - Module: to be a responsibility assignment rather than a subprogram
  - Modularizations: include the design decisions which must be made before the work on independent modules can begin(“System level” decisions)
- The Criterion of Decomposition
  - Information Hiding
  - Every module in the decomposition is characterized by its knowledge of design decision which it hides from all others.

Parnas1972

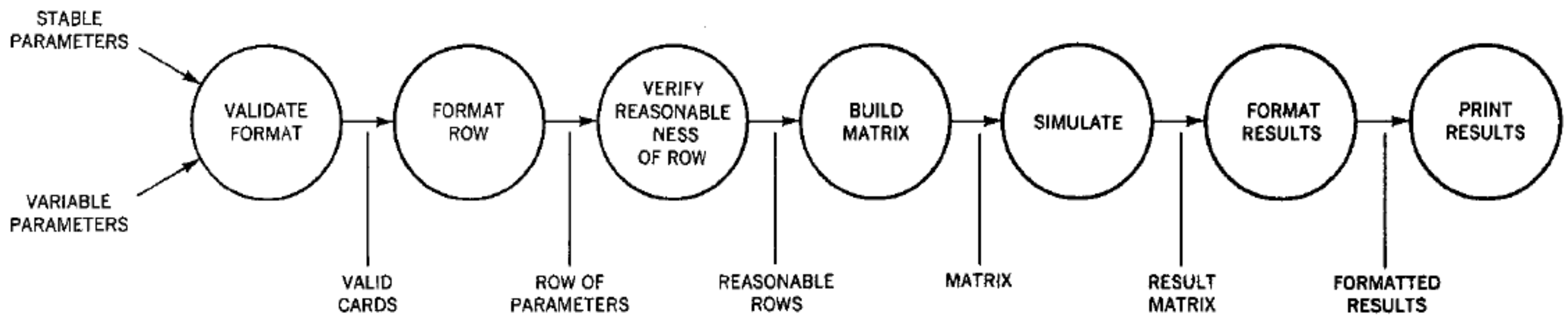
- Hierarchical Structure
    - If a certain relation may be defined between the modules or programs
    - That relation is a partial ordering
    - The relation we are concerned with is “use” or “depends upon”
- A data structure , its internal linking , accessing procedures and modifying procedures are part of a single module (the idea of encapsulation)
- The formats of control blocks used in queues in operating systems and similar programs must be hidden with in a “control block module”

Stevens1974

- “Structured design”
- Module
    - A set of one or more contiguous program statements having a name by which other parts of the system can invoke it and preferably having its own distinct set of variable names
- Coupling
    - The measure of the strength of association established by a connection from one module to another
- Cohesiveness
    - Coincidental; Logical; Temporal; Communicational; Sequential; Functional

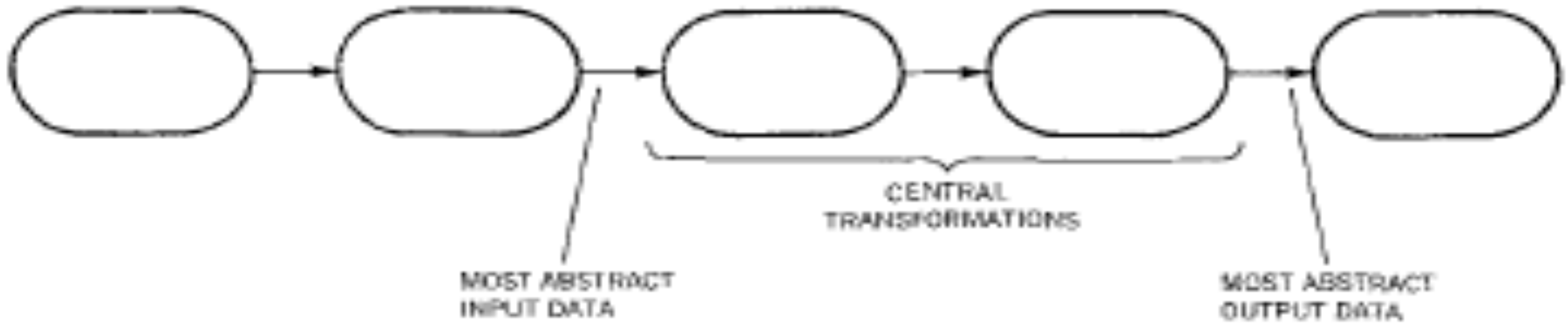
Stevens1974

- Designing the structure
    - Step 1: Sketch a functional picture of the problem



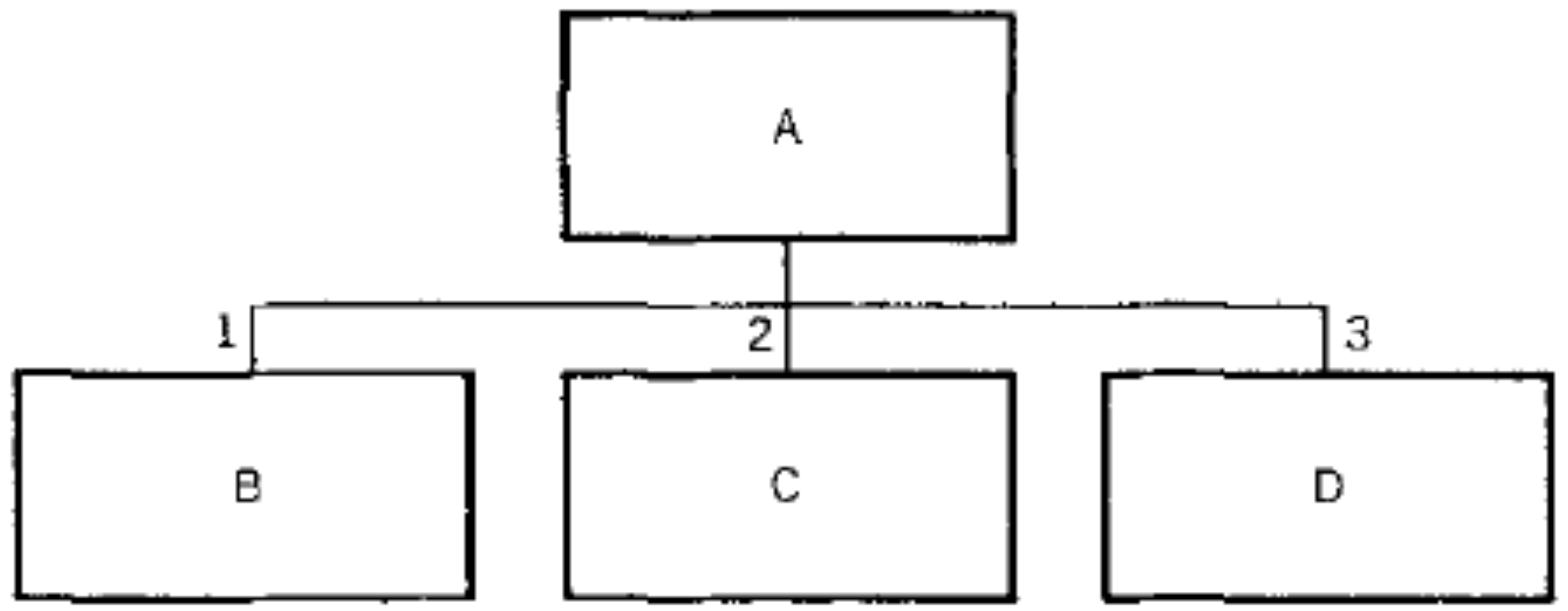
Stevens1974

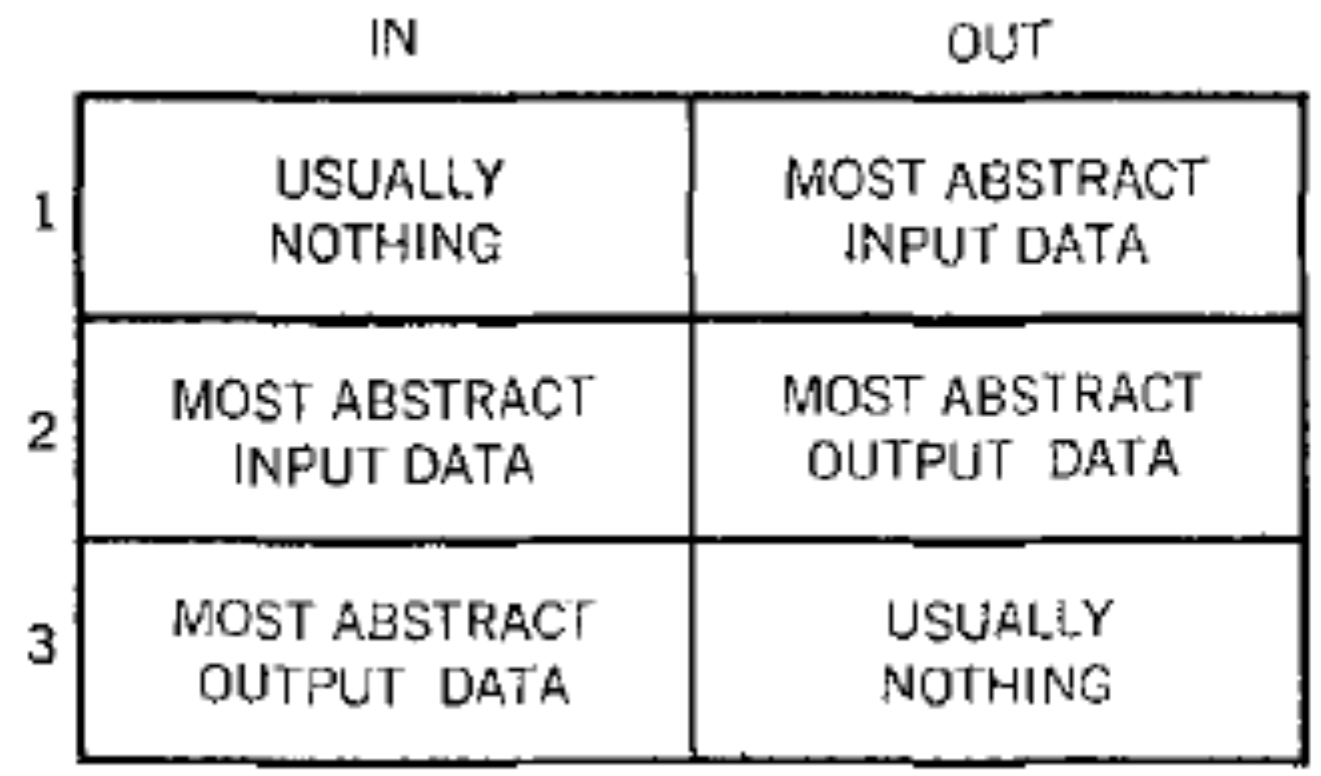
- Step 2: Identify the external conceptual streams of data.
- Step 3: Identify the major external conceptual stream of data in the problem



Stevens1974

- Step 4: Design the structure from the previous information with a source module for each conceptual input steam which exists at the point of most abstract input data; do sink modules similarly

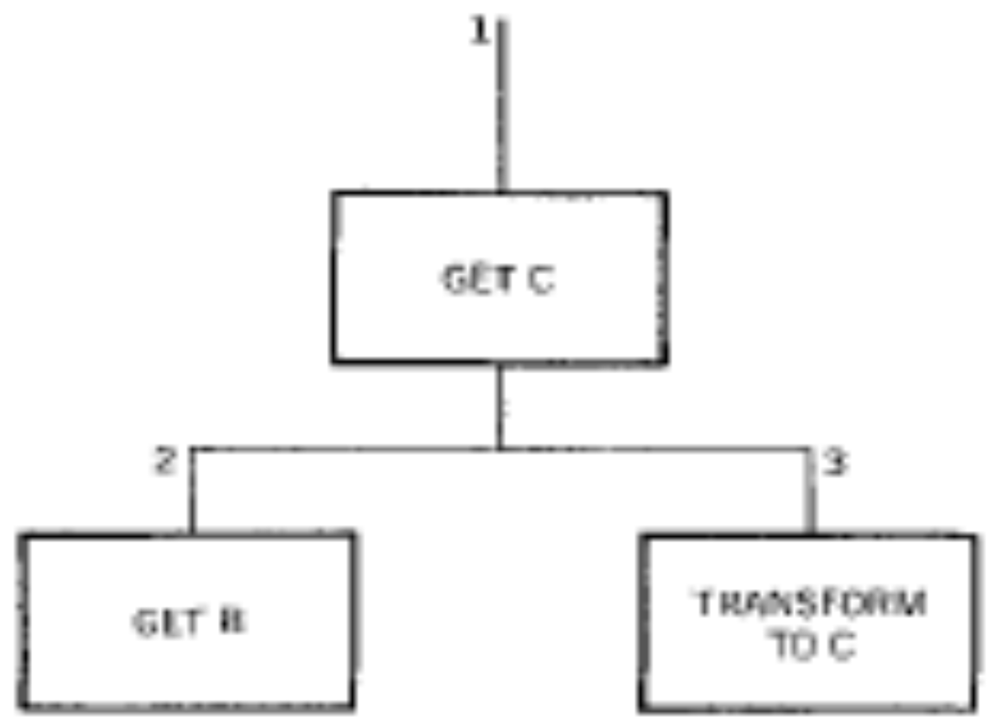




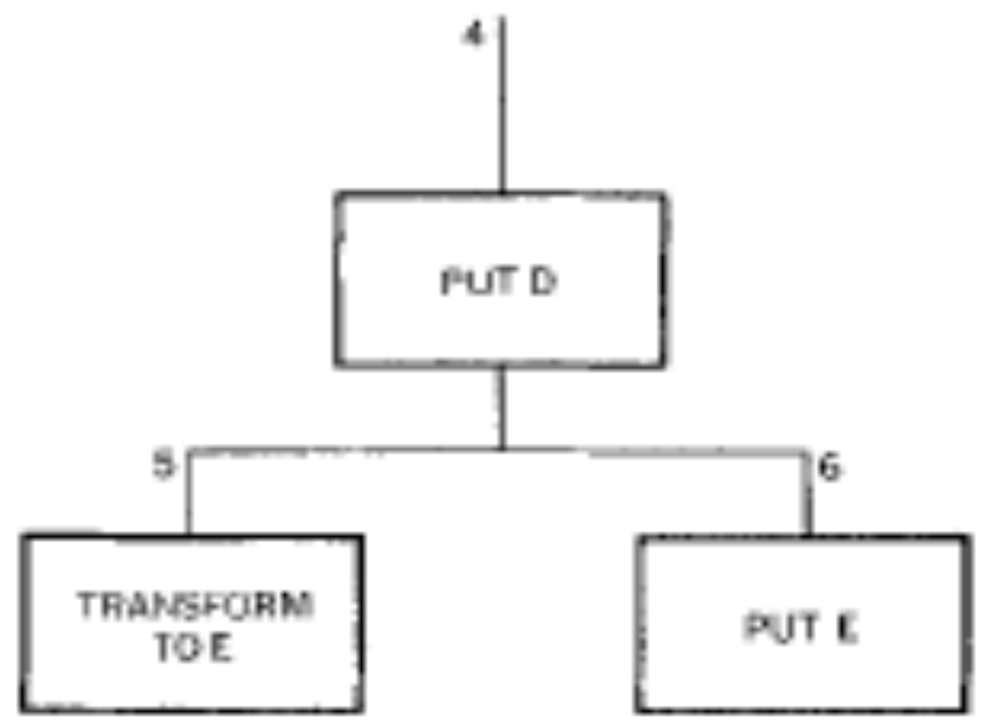
	IN	OUT
1	USUALLY NOTHING	MOST ABSTRACT INPUT DATA
2	MOST ABSTRACT INPUT DATA	MOST ABSTRACT OUTPUT DATA
3	MOST ABSTRACT OUTPUT DATA	USUALLY NOTHING

Stevens1974

- Step 5: For each source module, identify the last transformation necessary to produce the form being returned by that module



	IN	OUT
1		C
2		B
3	B	C



	IN	OUT
4	D	
5	D	E
6	E	

Stevens1974

- The Scope of control of a module
- The Scope of effect of a decision

- The System is simpler when the scope of effect of a decision is in the scope of control of the module containing the decision

Parnas1978

- “Designing Software for Ease of Extension and Contraction”

- Steps Towards a Better Structure
  - Requirements definition: identifying the subsets first
  - Information hiding: interface and module definition
  - The virtual machine concept
  - Designing the “uses” structure

Parnas1978

- “Uses” relation
  - A uses B if correct execution of B may be necessary for A to complete the task described in its specification
  - Allow A “uses” B when all of the following conditions hold:
    - 1) A is essentially simpler because it uses B
    - 2) B is not substantially more complex because it is not allowed to use A
    - 3) There is a useful subset containing B and not needing A
    - 4) There is no conceivably useful subset containing A but not B

Parnas1985

- "The Modular Structure of Complex Systems"

- Module Guide
    - Primary Secret
    - Role
    - Criteria behind assigning the module its particular responsibility

- Module Hierarchy
    - Top Level Decomposition
    - Second Level Decomposition
    - Third Level

Parnas1985

- Secret:
  - Primary Secret: Hidden information contained within the module.
  - Secondary Secret: Implementation decisions used to implement the module.

- Top Level Decomposition
  - Hardware-Hiding Module
  - Behavior-Hiding Module
  - Software Decision Module

Eder1992

- “Coupling and Cohesion in Object-Oriented Systems”

- Guidelines for “Good” design
- Coupling in OO
  - Interaction coupling; Component coupling; Inheritance coupling
- Cohesion in OO
  - Method cohesion; Class cohesion; Inheritance cohesion

Hitz1995

- “Measuring Coupling and Cohesion In Object-Oriented Systems”

- Class level coupling
- Object level coupling

- A framework for a comprehensive metric for coupling and cohesion

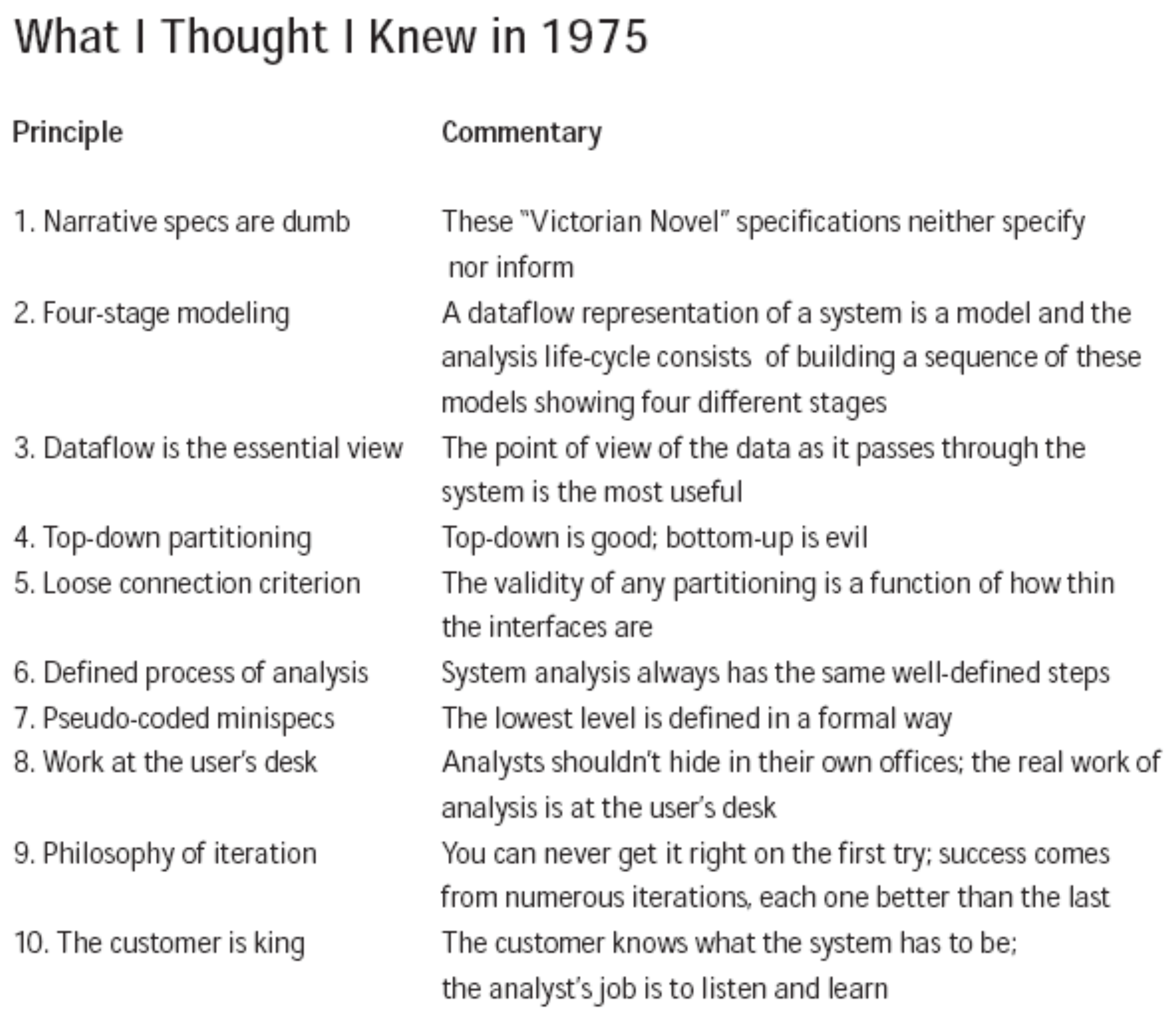
McConnell1996

- “Missing in Action: Information Hiding”

- Information hiding
  - is one of software engineering’s seminal design ideas
  - Doesn’t require or depend on any particular design methodology
  - Is characterized by the idea of “secrets”
  - The most common kind of secret is a design decision that you think might change
  - Asking what needs to be hidden supports good design decisions at all levels

Demarco2002

“Structured Analysis”



What I Thought I Knew in 1975

Principle	Commentary
1. Narrative specs are dumb	These "Victorian Novel" specifications neither specify nor inform
2. Four-stage modeling	A dataflow representation of a system is a model and the analysis life-cycle consists of building a sequence of these models showing four different stages
3. Dataflow is the essential view	The point of view of the data as it passes through the system is the most useful
4. Top-down partitioning	Top-down is good; bottom-up is evil
5. Loose connection criterion	The validity of any partitioning is a function of how thin the interfaces are
6. Defined process of analysis	System analysis always has the same well-defined steps
7. Pseudo-coded minispecs	The lowest level is defined in a formal way
8. Work at the user's desk	Analysts shouldn't hide in their own offices; the real work of analysis is at the user's desk
9. Philosophy of iteration	You can never get it right on the first try; success comes from numerous iterations, each one better than the last
10. The customer is king	The customer knows what the system has to be; the analyst's job is to listen and learn

What I Still Believe

Principle	Revised Commentary (as of 2001)
1. Narrative specs are dumb	Narrative specs are not the problem; a suitably partitioned spec with narrative text used at the bottom level makes a fine statement of work
2. Four-stage modeling	The four stages I proposed in 1975 were far too time consuming
3. **Dataflow** is the essential view	Dataflow is one of the essential views, not the only one
4. Top-down **partitioning**	Partitioning is essential in dealing with anything complex, but top-down partitioning is often far too difficult to achieve and not at all the great advantage it was touted to be
5. Loose connection criterion	This is an important truth: when you're attacking complexity by partitioning, the thinner the interface, the better the partitioning – if the interfaces are still thick, go back and partition again, searching for the natural seams of the domain
6. Defined process of analysis	Defined process is a holy grail that has never yet been found and probably never will be
7. Pseudo-coded **minispecs**	It's useful to partition the whole and then specify the pieces, but pseudo-code was an awful mistake (puts analysts into coding mode when they should be busy analyzing)
8. Work at the user's desk	Analysts have a tendency to hide at their own desks, but much of the action is in the business area and they need to venture out to find it
9. Philosophy of iteration	We never get it right the first time; the best we can do is improve from one iteration to the next; if we can continue to do this at each iteration, we can get arbitrarily close to a perfect product
10. The customer is king	See below . . .

模块化与信息隐藏

Modularity

- Computer systems are not monolithic:
  - they are usually composed of multiple, interacting modules.
- Modularity has long been seen as a key to cheap, high quality software.
- The goal of system design is to decide:
  - – what the modules are;
  - – what the modules should be;
  - – how the modules interact with one-another.

What is a module?

- Common view: a piece of code. Too limited.
- Compilation unit, including related declarations and interface
- David Parnas: a unit of work.
- Collection of programming units (procedures, classes, etc.)
    - with a well-defined interface and purpose within the entire system,
    - that can be independently assigned to a developer

Why modularize a system? I

- Management: Partition the overall development effort
    - – Divide and conquer
- Evolution: Decouple parts of a system so that changes to one part are isolated from changes to other parts
    - Principle of directness (clear allocation of requirements to modules, ideally one requirement (or more) maps to one module)
    - Principle of continuity/locality (small change in requirements triggers a change to one module only)

Why modularize a system? II

- Understanding: Permit system to be understood
  - as composition of mind-sized chunks, e.g., the 7±2 Rule
  - with one issue at a time, e.g., principles of locality, encapsulation, separation of concerns
- Key issue: what criteria to use for modularization?
- --> Information Hiding

Information

- Information ->secrets
- what’s a “secret”? -> Change
    - Representation of data
    - Properties of a device, other than required properties
    - Implementation of world models
    - Mechanisms that support policies

the design areas that are most likely to change

- hardware dependencies
  - External software system
- input and output formats
  - DB, Internet, UI, …
- nonstandard language features and library routines;
  - Platform: os, middleware, framework…

the design areas that are most likely to change

- difficult design and implementation areas
  - especially areas that might be developed poorly and require redesign or reimplementation;
  - Complex…, monitor, exception, log, …
- complex data structures, data structures that are used by more than one class, or data structures you haven’t designed to your satisfaction;
  - Separate model from logic
- complex logic, which is almost as likely to change as complex data structures;
  - Algorithm, schedule, time-critical, performance-critical, …

the design areas that are most likely to change

- global variables, which are probably never truly needed, but which always benefit from being hidden behind access routines;
  - Data Access Routines
- data-size constraints such as array declarations and loop limits;
- and business rules such as the laws, regulations, policies, and procedures that are embedded into a computer system.

Hiding

- Try to localize future change
  - Hide system details likely to change independently
  - Separate parts that are likely to have a different rate of change
  - Expose in interfaces assumptions unlikely to change

Information Hiding

- the most common kind of secret is a design decision that you think might change.
- You then separate each design secret by assigning it to its own class, subroutine, or other design unit.
- Next you isolate (encapsulate) each secret so that if it does change, the change doesn’t affect the rest of the program.

Interface vs. Implementation

- Users and implementers of a module have different views of it.

- Interface: user's view of a module.

- describes only what a user needs to know to use the module
- makes it easier to understand and use
- describes what services the module provides, but not how it's able to provide them

What Is an Interface?

- Interface as a contract - whatever is published by a module that
  - Provided interface: clients of the module can depend on and
  - Required interface: the module can depend on from other modules
- Syntactic interfaces
  - How to call operations
    - List of operation signatures
    - Sometimes also valid orders of calling operations
- Semantic interfaces
  - What the operations do, e.g.,
    - Pre- and post-conditions
    - Use cases

Further Principles

- Explicit interfaces
  - Make all dependencies between modules explicit (no hidden coupling)
- Low coupling - few interfaces
  - Minimize the amount of dependencies between modules
- Small interfaces
  - Keep the interfaces narrow
    - Combine many parameters into structs/objects
    - Divide large interfaces into several interfaces
- High cohesion
  - A module should encapsulate some well-defined, coherent piece of functionality (more on that later)

Coupling and Cohesion

- Cohesion is a measure of the coherence of a module amongst the pieces of that module.
- Coupling is the degree of interaction between modules.
- You want high cohesion and low coupling.

KWIC

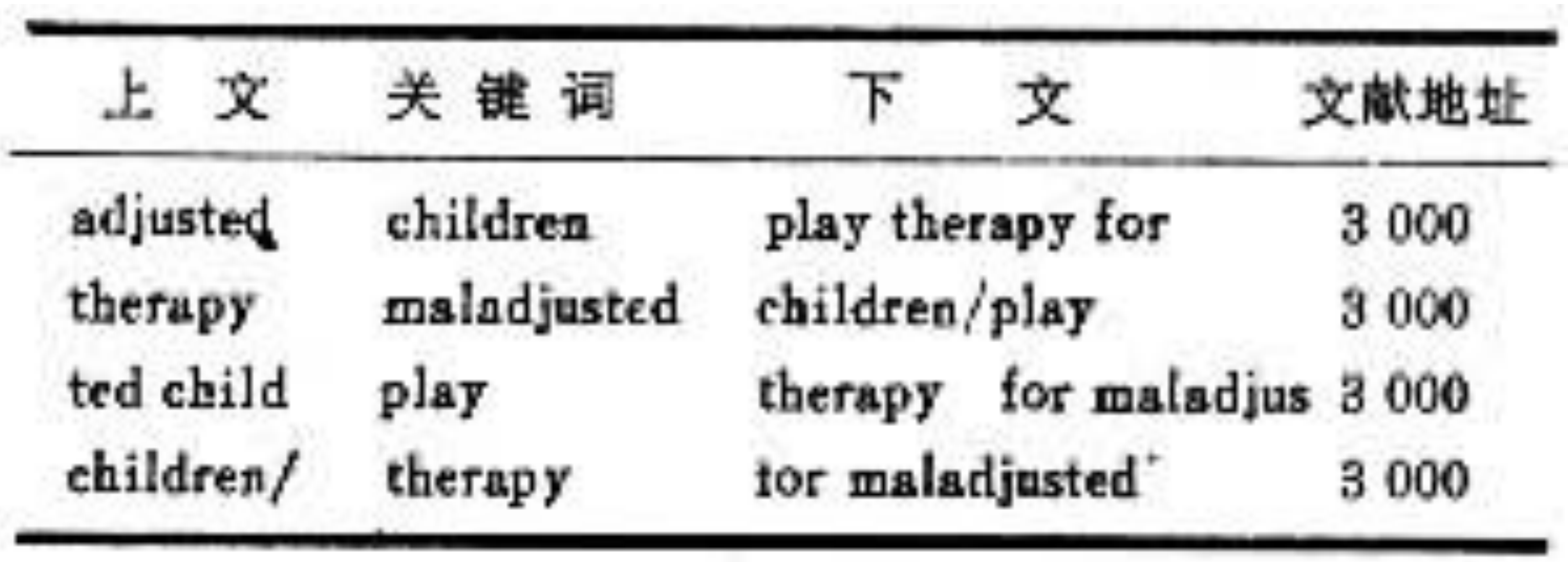
KWIC

- 简称KWIC，又称上下文关键词索引，由IBM的卢恩首创，是最早出现的机编索引，1960年首次用于美国化学文摘社出版的《化学题录>>(Chemical Titles)。

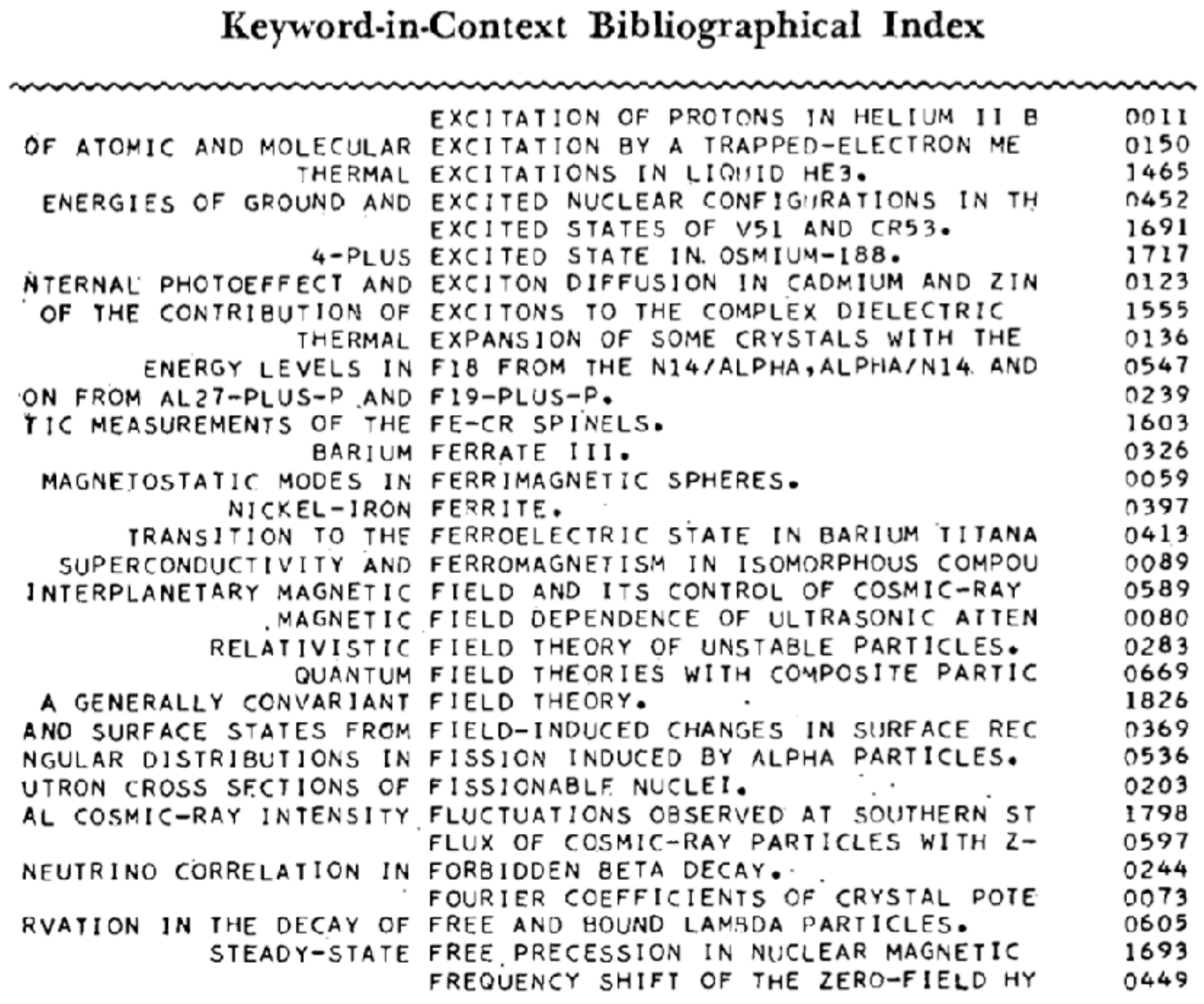
- KWIC索引的的编制特点是：使用禁用词表选择标题中具有检索意义的词为关键词，并将其作为确定索引条目的依据；关键词的排检点设于标题的中部，所有索引条目按关键词的字顺竖向排列；保留文献篇名中关键词前后的上下文，如文献名称过长，可以以轮排的形式移至条目的前部或后部；款目后跟随该信息资源的位置。

- 上述条目均按关键词的字顺排列在相应位置，检索时先在检索入口处查找与检索课题有关的关键词，再通过阅读上下文寻找符合检索要求的文献。可以按排检点为中心对同一关键词有关的资源集中检索查找，是这一索引的优点；不足是将索引的排检点设置在中部不符合用户使用习惯。

一篇题为Play therapy for maladjusted children(《孤僻儿童的游戏疗法》)的论文，输入计算机后可产生以下几条索引款目：



上 文	关键词	下 文	文献地址
adjusted	children	play therapy for	3 000
therapy	maladjusted	children/play	3 000
ted child	play	therapy for maladjus	3 000
children/	therapy	for maladjusted	3 000



Keyword-in-Context Bibliographical Index

```
                           EXCITATION OF PROTONS IN HELIUM II B       0011
 OF ATOMIC AND MOLECULAR   EXCITATION BY A TRAPPED-ELECTRON ME        0150
                 THERMAL   EXCITATIONS IN LIQUID HE3.                 1465
  ENERGIES OF GROUND AND   EXCITED NUCLEAR CONFIGURATIONS IN TH       0452
                           EXCITED STATES OF V51 AND CR53.            1691
                  4-PLUS   EXCITED STATE IN OSMIUM-188.               1717
NTERNAL PHOTOEFFECT AND    EXCITON DIFFUSION IN CADMIUM AND ZIN       0123
 OF THE CONTRIBUTION OF    EXCITONS TO THE COMPLEX DIELECTRIC         1555
                 THERMAL   EXPANSION OF SOME CRYSTALS WITH THE        0136
        ENERGY LEVELS IN   F18 FROM THE N14/ALPHA,ALPHA/N14 AND       0547
ON FROM AL27-PLUS-P AND    F19-PLUS-P.                                0239
TIC MEASUREMENTS OF THE    FE-CR SPINELS.                             1603
                  BARIUM   FERRATE III.                               0326
  MAGNETOSTATIC MODES IN   FERRIMAGNETIC SPHERES.                     0059
             NICKEL-IRON   FERRITE.                                   0397
       TRANSITION TO THE   FERROELECTRIC STATE IN BARIUM TITANA       0413
  SUPERCONDUCTIVITY AND    FERROMAGNETISM IN ISOMORPHOUS COMPOU       0089
INTERPLANETARY MAGNETIC    FIELD AND ITS CONTROL OF COSMIC-RAY        0589
                MAGNETIC   FIELD DEPENDENCE OF ULTRASONIC ATTEN       0080
            RELATIVISTIC   FIELD THEORY OF UNSTABLE PARTICLES.        0283
                 QUANTUM   FIELD THEORIES WITH COMPOSITE PARTIC       0669
 A GENERALLY CONVARIANT    FIELD THEORY.                              1826
AND SURFACE STATES FROM    FIELD-INDUCED CHANGES IN SURFACE REC       0369
NGULAR DISTRIBUTIONS IN    FISSION INDUCED BY ALPHA PARTICLES.        0536
UTRON CROSS SECTIONS OF    FISSIONABLE NUCLEI.                        0203
AL COSMIC-RAY INTENSITY    FLUCTUATIONS OBSERVED AT SOUTHERN ST       1798
                           FLUX OF COSMIC-RAY PARTICLES WITH Z-       0597
NEUTRINO CORRELATION IN    FORBIDDEN BETA DECAY.                      0244
                           FOURIER COEFFICIENTS OF CRYSTAL POTE       0073
RVATION IN THE DECAY OF    FREE AND BOUND LAMBDA PARTICLES.           0605
            STEADY-STATE   FREE PRECESSION IN NUCLEAR MAGNETIC        1693
                           FREQUENCY SHIFT OF THE ZERO-FIELD HY       0449
```

例子

Example System: KWIC

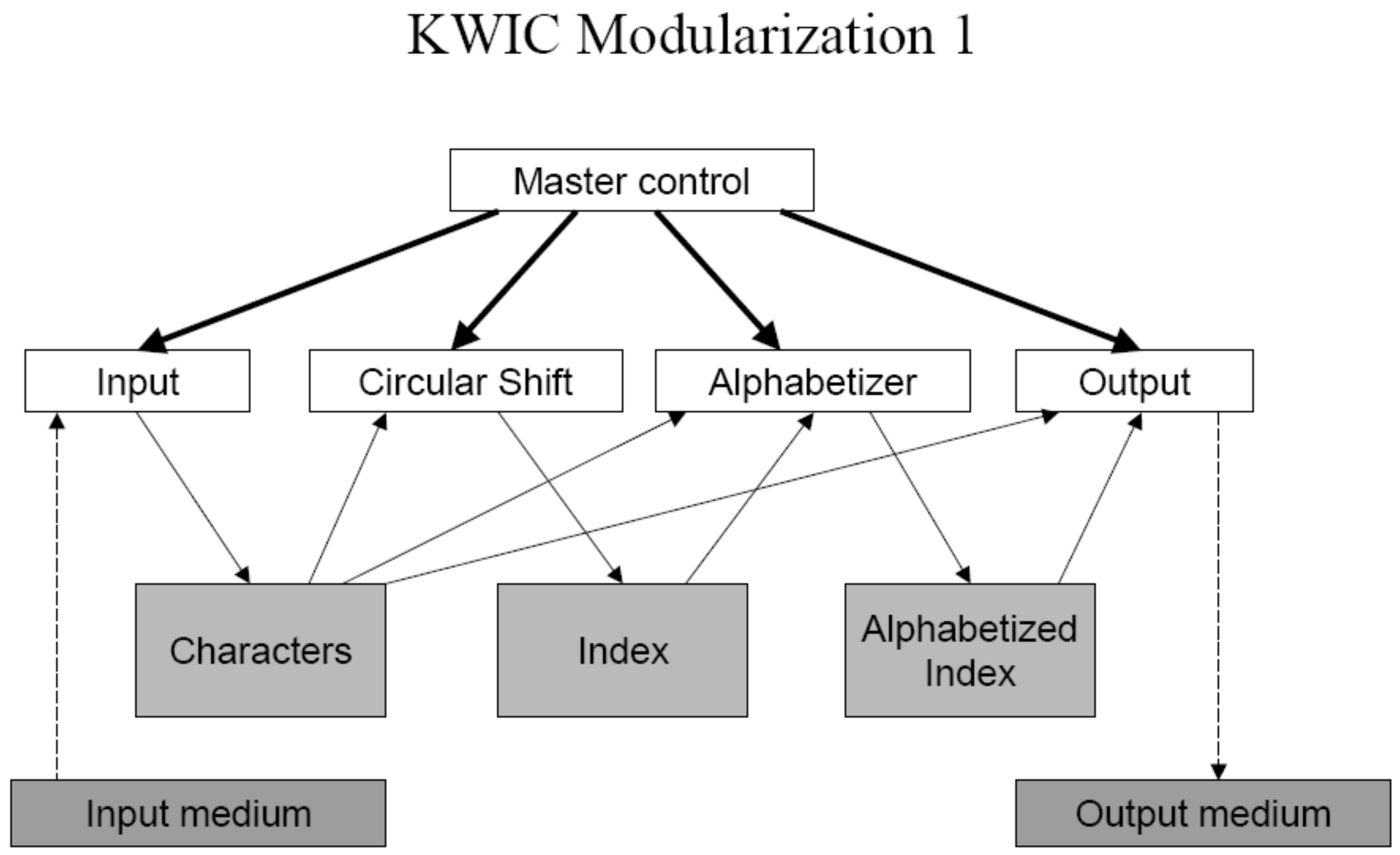
- The KWIC index system accepts:
  - an ordered set of lines
  - each line is an ordered set of words
  - each word is an ordered set of characters
- Every line is "circularly shifted" and copied by:
  - repeatedly removing the first word
  - appending it at the end of the line
- Outputs a listing of all circular shifts of all lines in alphabetical order

KWIC Example

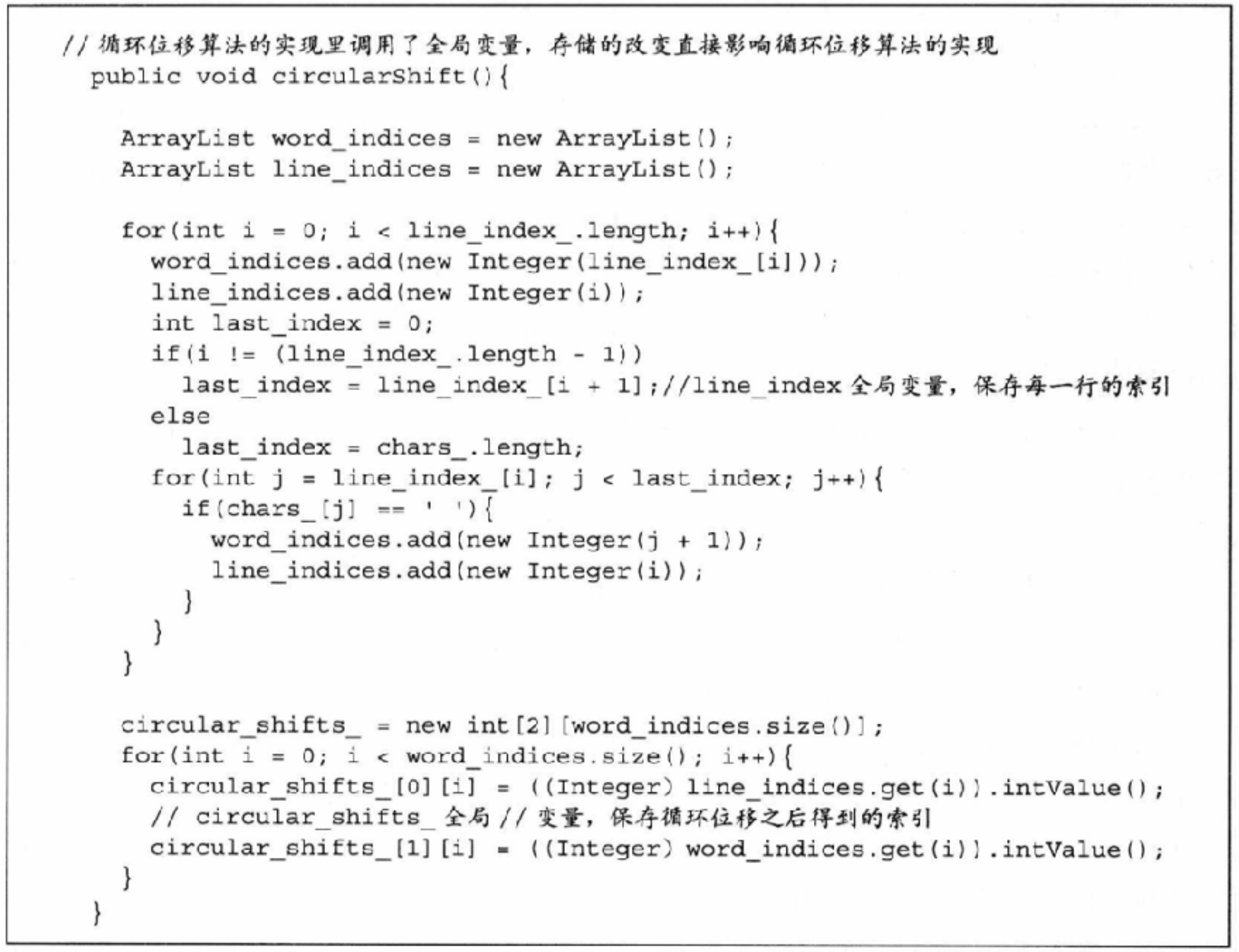
- Input:
- bar sock
- car dog
- town fog
- Output:
- bar sock
- car dog
- dog car
- fog town
- sock bar
- town fog

Four steps:
Input
Circular shift
Alphabetize
Output

Index:
Circular shift positions: 1 5 10 14 18 23
Alphabetize: 1 10 14 23 5 18



Design according to function

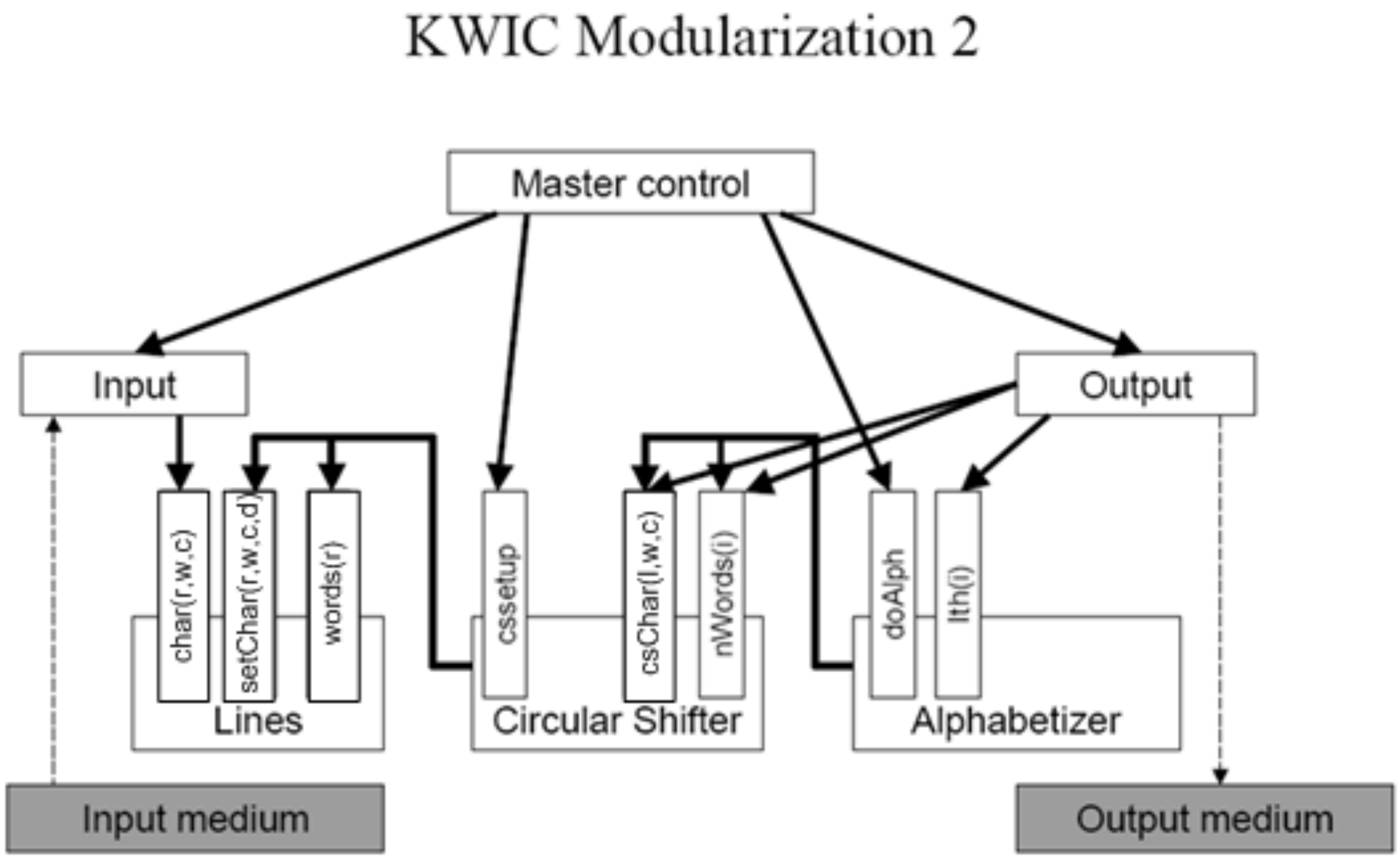


```
// 循环位移算法的实现里调用了全局变量，存储的改变直接影响循环位移算法的实现
  public void circularShift(){

    ArrayList word_indices = new ArrayList();
    ArrayList line_indices = new ArrayList();

    for(int i = 0; i < line_index_.length; i++){
      word_indices.add(new Integer(line_index_[i]));
      line_indices.add(new Integer(i));
      int last_index = 0;
      if(i != (line_index_.length - 1))
        last_index = line_index_[i + 1];//line_index 全局变量，保存每一行的索引
      else
        last_index = chars_.length;
      for(int j = line_index_[i]; j < last_index; j++){
        if(chars_[j] == ' '){
          word_indices.add(new Integer(j + 1));
          line_indices.add(new Integer(i));
        }
      }
    }

    circular_shifts_ = new int[2][word_indices.size()];
    for(int i = 0; i < word_indices.size(); i++){
      circular_shifts_[0][i] = ((Integer) line_indices.get(i)).intValue();
      // circular_shifts_ 全局 // 变量，保存循环位移之后得到的索引
      circular_shifts_[1][i] = ((Integer) word_indices.get(i)).intValue();
    }
  }
```

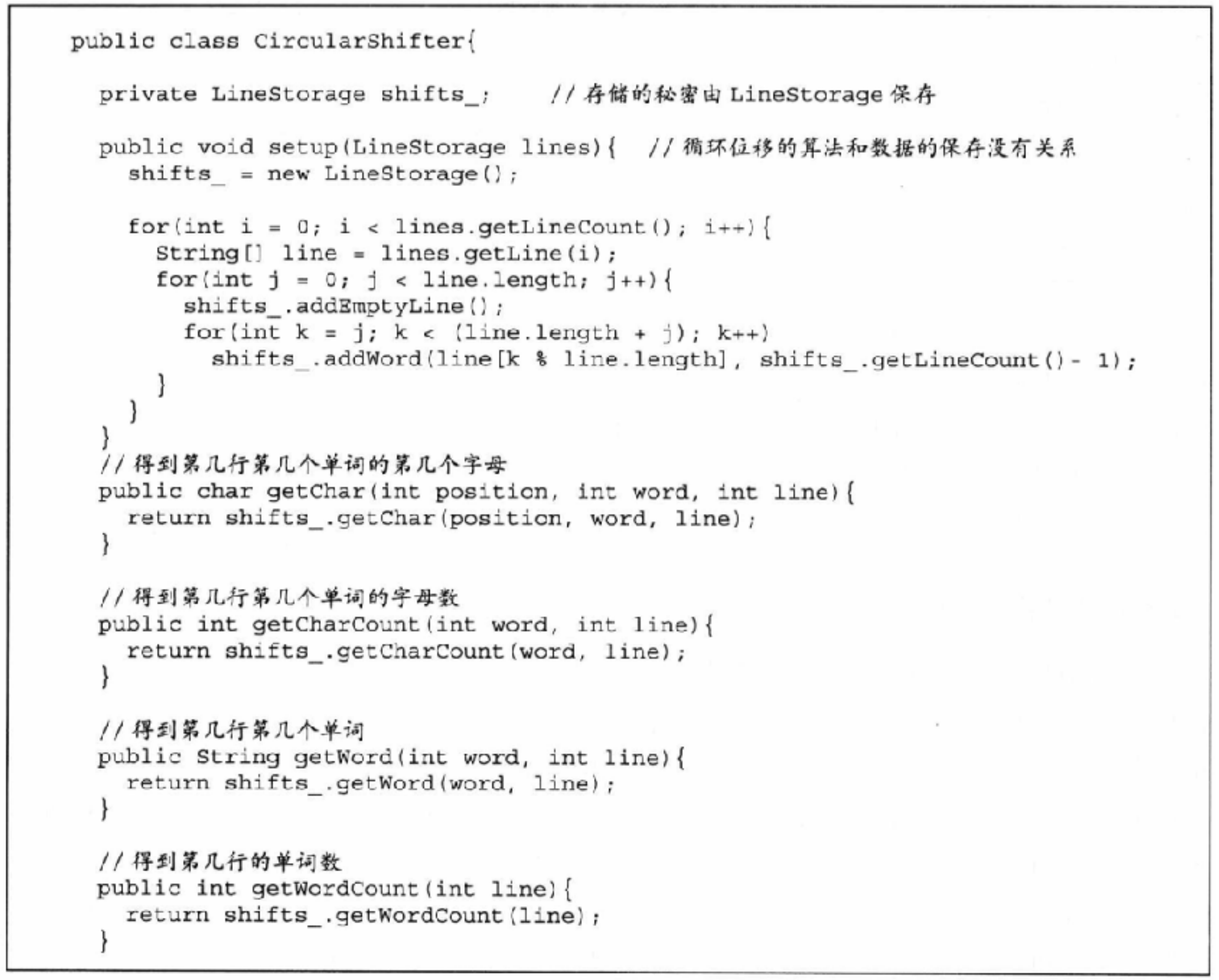
循环位移算法的实现

Information Hiding

- Each module hides the implementation of an important design decision so that only the constituents of that module know the details
    - Especially if there is a list of all possible design changes is made - Hiding Assumption List
- All design decisions are independent of each other

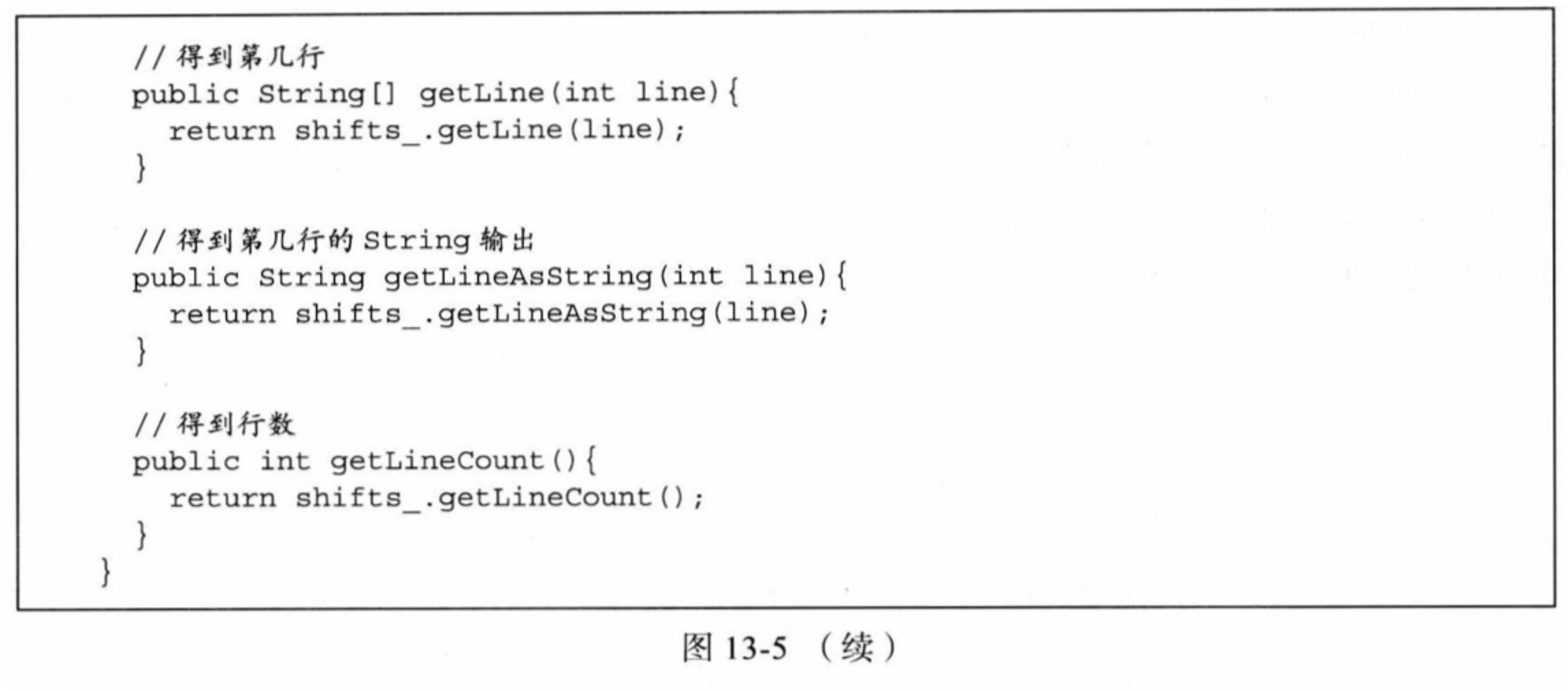


Design according to decisions



```
public class CircularShifter{

  private LineStorage shifts_;     //存储的秘密由 LineStorage 保存

  public void setup(LineStorage lines){   //循环位移的算法和数据的保存没有关系
    shifts_ = new LineStorage();

    for(int i = 0; i < lines.getLineCount(); i++){
      String[] line = lines.getLine(i);
      for(int j = 0; j < line.length; j++){
        shifts_.addEmptyLine();
        for(int k = j; k < (line.length + j); k++)
          shifts_.addWord(line[k % line.length], shifts_.getLineCount()- 1);
      }
    }
  }
  //得到第几行第几个单词的第几个字母
  public char getChar(int position, int word, int line){
    return shifts_.getChar(position, word, line);
  }

  //得到第几行第几个单词的字母数
  public int getCharCount(int word, int line){
    return shifts_.getCharCount(word, line);
  }

  //得到第几行第几个单词
  public String getWord(int word, int line){
    return shifts_.getWord(word, line);
  }

  //得到第几行的单词数
  public int getWordCount(int line){
    return shifts_.getWordCount(line);
  }
```

CircularShifter的定义



```
    //得到第几行
    public String[] getLine(int line){
      return shifts_.getLine(line);
    }

    //得到第几行的 String 输出
    public String getLineAsString(int line){
      return shifts_.getLineAsString(line);
    }

    //得到行数
    public int getLineCount(){
      return shifts_.getLineCount();
    }
}
```

图 13-5 （续）

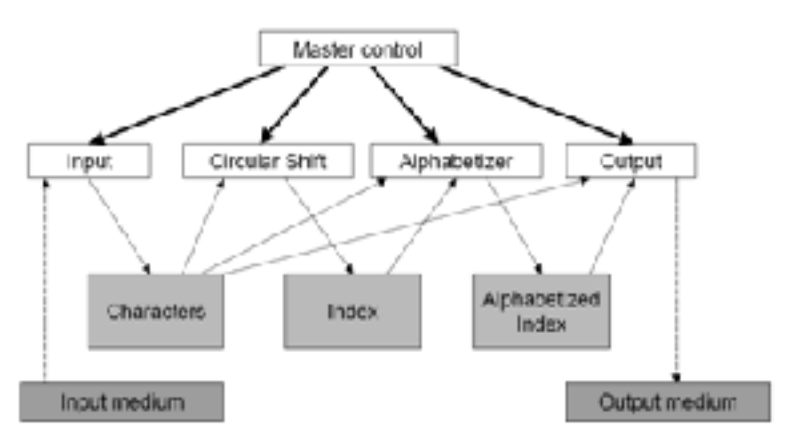
CircularShifter的定义

Criteria for decomposition

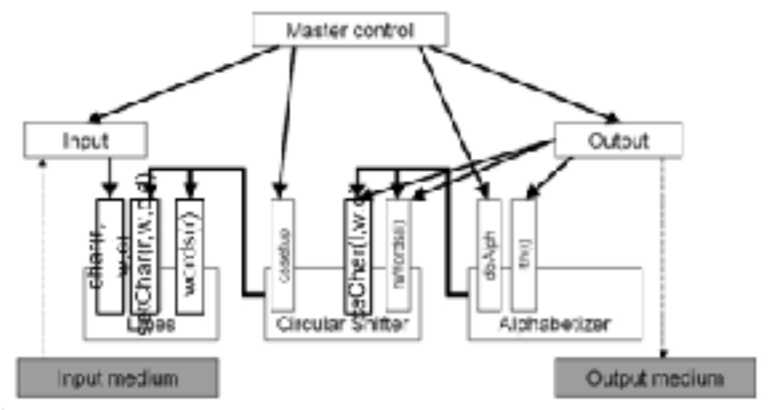
- Modularization 1
    - Each major step in the processing was a module
- Modularization 2
    - Information hiding
        - Each module has one or more "secrets”
            - Lines
                - how characters/lines are stored
            - Circular Shifter
                - algorithm for shifting, storage for shifts
            - Alphabetizer
                - algorithm for alpha, laziness of alpha
        - Each module is characterized by its knowledge of design decisions which it hides from all others.

Changeability Comparison

KWIC Modularization 1



KWIC Modularization 2



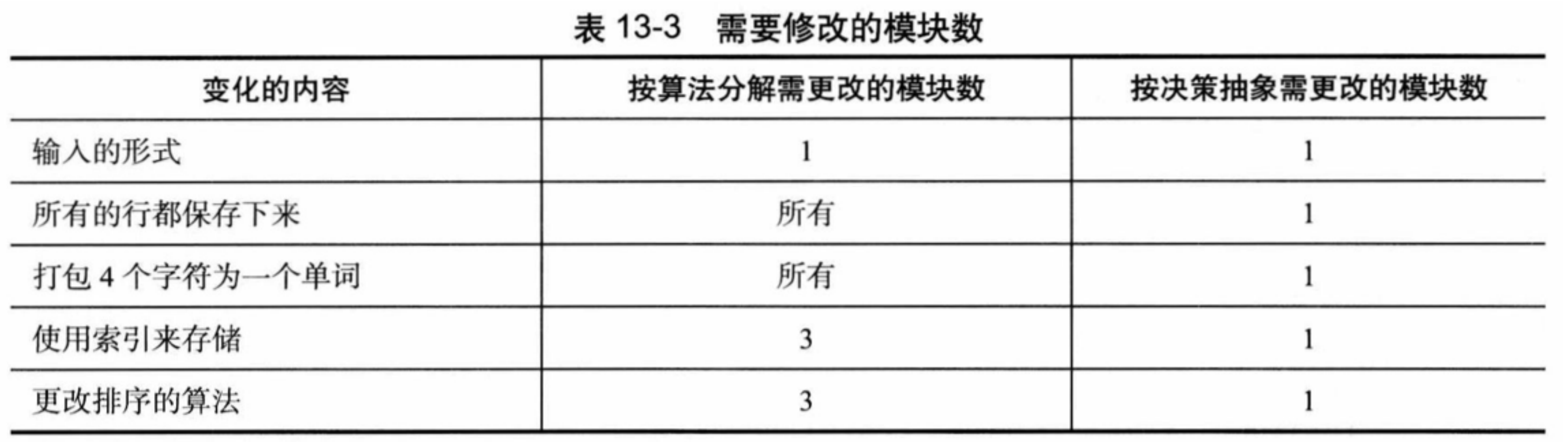


表 13-3　需要修改的模块数

变化的内容	按算法分解需更改的模块数	按决策抽象需更改的模块数
输入的形式	1	1
所有的行都保存下来	所有	1
打包 4 个字符为一个单词	所有	1
使用索引来存储	3	1
更改排序的算法	3	1

Independent Development

- Modularization 1
  - Must design all data structures before parallel work can proceed
  - Complex descriptions needed
- Modularization 2
  - Must design interfaces before parallel work can begin
  - Simple descriptions only

Comprehensibility

- Modularization 2 is better
  - Parnas subjective judgment

  - Less coupling among modules
    - Programming to interfaces

Modularization

- A Responsibility Assignment rather than sub-program.
- Represented by a design decision specific to itself and unknown to other modules
- Support flexibility in implementation
- Do Not represent steps in processing
- Low coupling, high cohesion.

Conclusion: Some Buzzwords

- Module
  - - parts that can be put together to make a complete system
  - - work assignment, subroutine, memory load , functional component
- Modularization
  - – Making “independent” modules
- Encapsulation
  - - language facility
- Information Hiding
  - - design principle

Outline

- 模块化与信息隐藏思想
- 结构化的模块化
    - 耦合
    - 内聚
    - 思想的应用
- 结构化的信息隐藏

耦合

模块化

- 两种思路

FIGURE 7.1 The design of a computer.

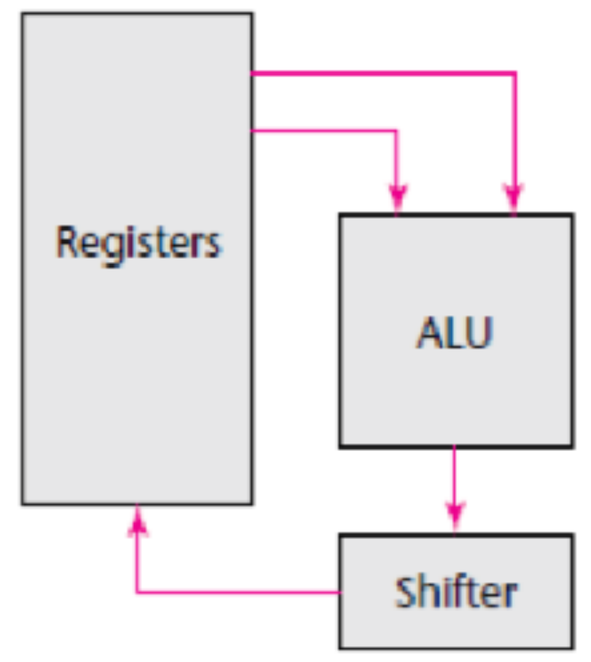


FIGURE 7.2 The computer of Figure 7.1 fabricated on three chips.

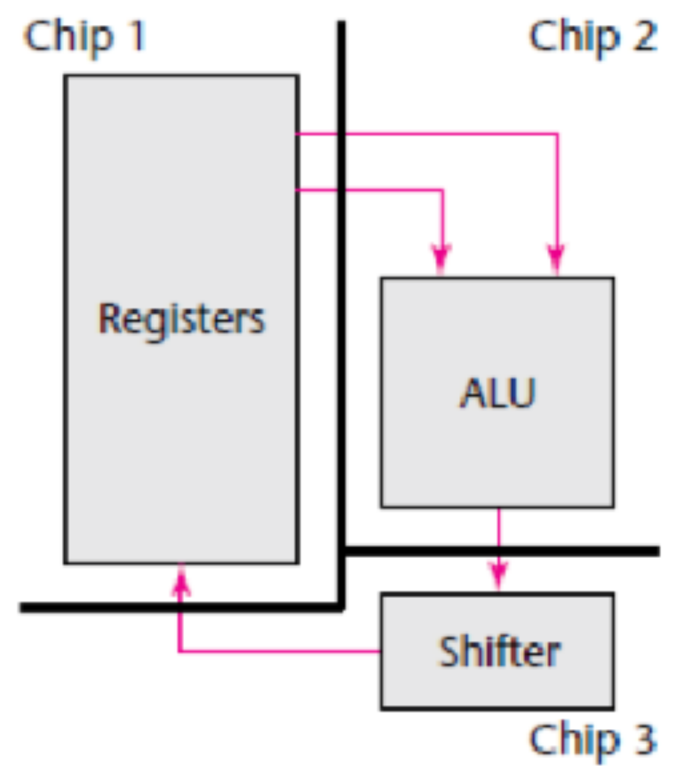
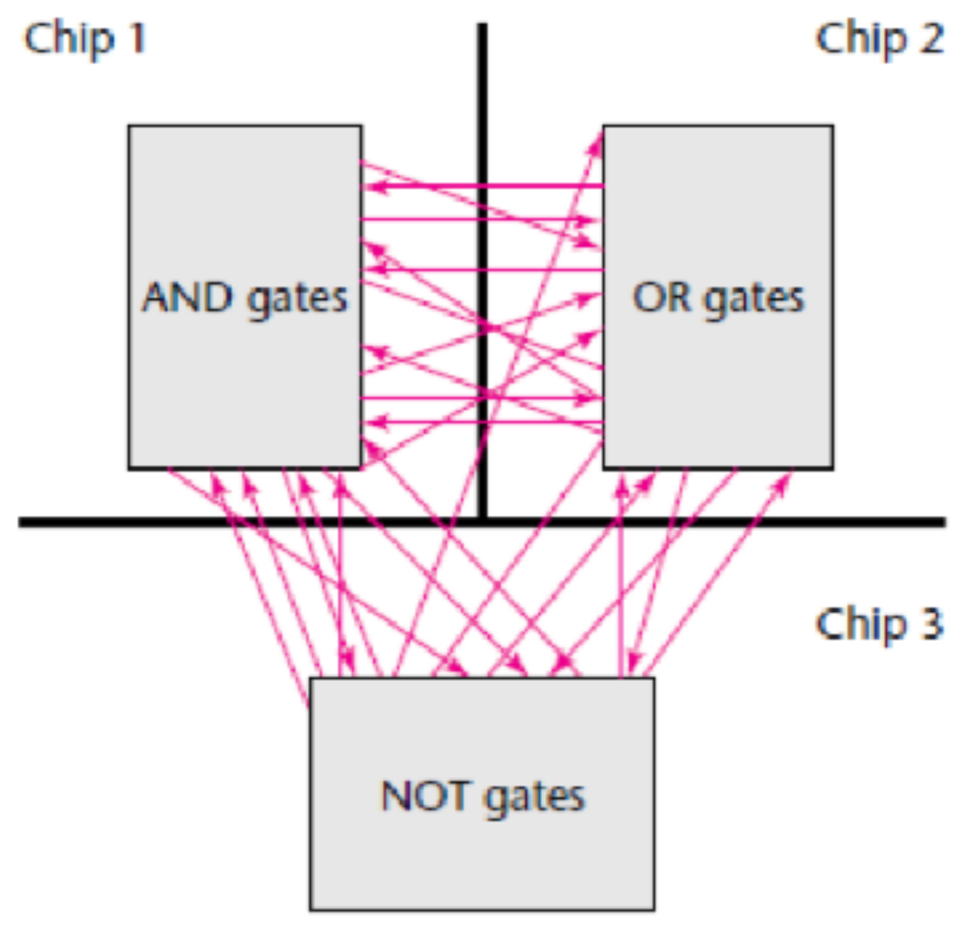


FIGURE 7.3 The computer of Figure 7.1 fabricated on three other chips.



模块之间

- Connection：
  - A connection is a reference to some label or address defined elsewhere
- 联系的复杂度
  - 数量
  - 程度

结构化的耦合

- Coupling is the measure of the strength of association established by a connection from one module to another
  - How complicated the connection is
  - Whether the connection refers to the module itself or something inside it
  - What is being sent or received

耦合的强度 1 - How complicated the connection is

- Connection of a module
    - To Common environments
        - General, Scoped
    - To other module

Principle 1:
Global Variables Consider Harmful!

Connection to Common Environment

- Assume N1+N2=N, M1+M2=M
  - If there are N shared elements by M modules
    - there are potential N*M connections
      - (N1+N2) *(M1+M2)= N1*M1+N2*M2+ N1*M2+N2*M1
  - If there are N1 shared elements by M1 modules, and N2 shared elements by M2 modules
    - There are potential (N1*M1+N2*M2) connections
- Encapsulation reduce coupling: Abstraction and Decomposition
  - Subdividing the potential shared elements into groups
  - Limiting each group access to the smallest possible subset of modules

Defects of Common environments

- A connection couple every module sharing it to every other such module
    - Error and change in one module can propagate to every others
    - Comprehension one module needs helps of every others
    - More different to reuse
- Each element in common environments adds to the complexity of the total system

Principle 2:
To be Explicit

```
class Person
        attr_accessor :data
        def initialize()
                @data = {}
        end
end

def frag2
        martin = Person.new
        martin.data["firstName"] = "Martin"
        martin.data["lastName"] = "Fowler"
        martin.data["numberOfDependents"] = 1

        print (martin.data["firstName"]," ",
                    martin.data["lastName"], " has ",
                    martin.data["numberOfDependents"],
                    " dependents")
end
```

```
class Person
       attr_accessor :lastName, :firstName, :numberOfDependents
end

def frag1
       martin = Person.new
       martin.firstName = “Martin”
       martin.lastName = “Fowler”
       martin.numberOfDependents = 1
       print (martin.firstName, ” ”, martin.lastName, ” has ”,
                     martin.numberOfDependents, ” dependents”)

end
```

Explicit VS Changeability

- Attributes and dictionaries
- Explicit call and Events
- Explicit subclasses and Data-driven code

Principle 3:
Don’t repeat!

```
class Invoice...

  String asciiStatement() {
    StringBuffer result = new StringBuffer();
    result.append("Bill for " + customer + "\n");
    Iterator it = items.iterator();
    while(it.hasNext()) {
        LineItem each = (LineItem) it.next();
        result.append("\t" + each.product() + "\t\t"
          + each.amount() + "\n");
    }
    result.append("total owed:" + total + "\n");
    return result.toString();
  }
```

```
  String htmlStatement() {
    StringBuffer result = new StringBuffer();
    result.append("<P>Bill for <I>" + customer + "</I></P>");
    result.append("<table>");
    Iterator it = items.iterator();
    while(it.hasNext()) {
        LineItem each = (LineItem) it.next();
        result.append("<tr><td>" + each.product()
          + "</td><td>" + each.amount() + "</td></tr>");
    }
    result.append("</table>");
    result.append("<P> total owed:<B>" + total + "</B></P>");
    return result.toString();
  }
```

```
interface Printer {
  String header(Invoice iv);
  String item(LineItem line);
  String footer(Invoice iv);
}
```

(a)

```
static class AsciiPrinter implements Printer {
  public String header(Invoice iv) {
    return "Bill for " + iv.customer + "\n";
  }
  public String item(LineItem line) {
    return "\t" + line.product() + "\t\t" + line.amount() + "\n";
  }
  public String footer(Invoice iv) {
    return "total owed:" + iv.total + "\n";
  }
 }
```

(b)

```
class Invoice...
  public String statement(Printer pr) {
    StringBuffer result = new StringBuffer();
    result.append(pr.header(this));
    Iterator it = items.iterator();
    while(it.hasNext()) {
         LineItem each = (LineItem) it.next();
         result.append(pr.item(each));
    }
    result.append(pr.footer(this));
    return result.toString();
  }
```

(a)

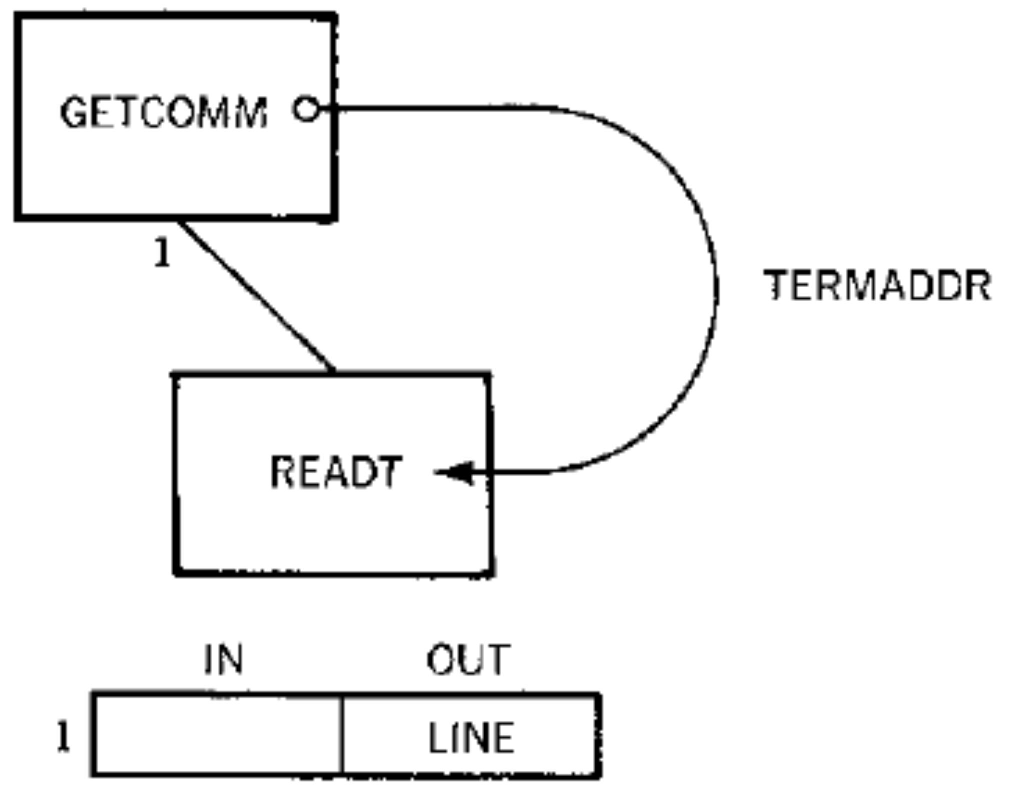
```
class Invoice...
  public String asciiStatement2() {
    return statement (new AsciiPrinter());
  }
```

(b)

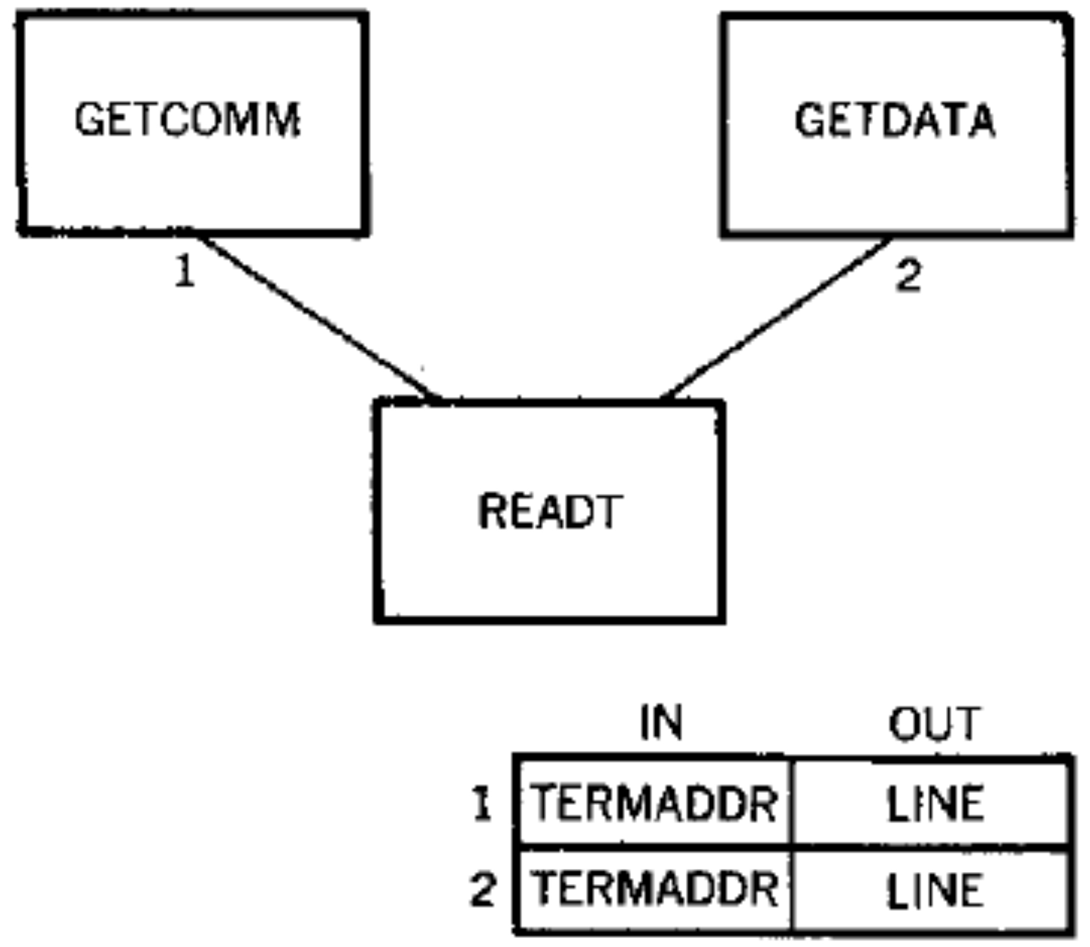
耦合的强度 2 - Whether the connection refers to the module itself or something inside it

- Connections that address or refer to a module as a whole by its name yield lower coupling than connections referring to the internal elements of another module

Figure 2 Module connections



	IN	OUT
1		LINE



	IN	OUT
1	TERMADDR	LINE
2	TERMADDR	LINE

Principle 4: Programming to Interface!

A well-structured system

- in which communication is via passed parameters through defined interfaces,

耦合的强度 3 - What is being sent and received

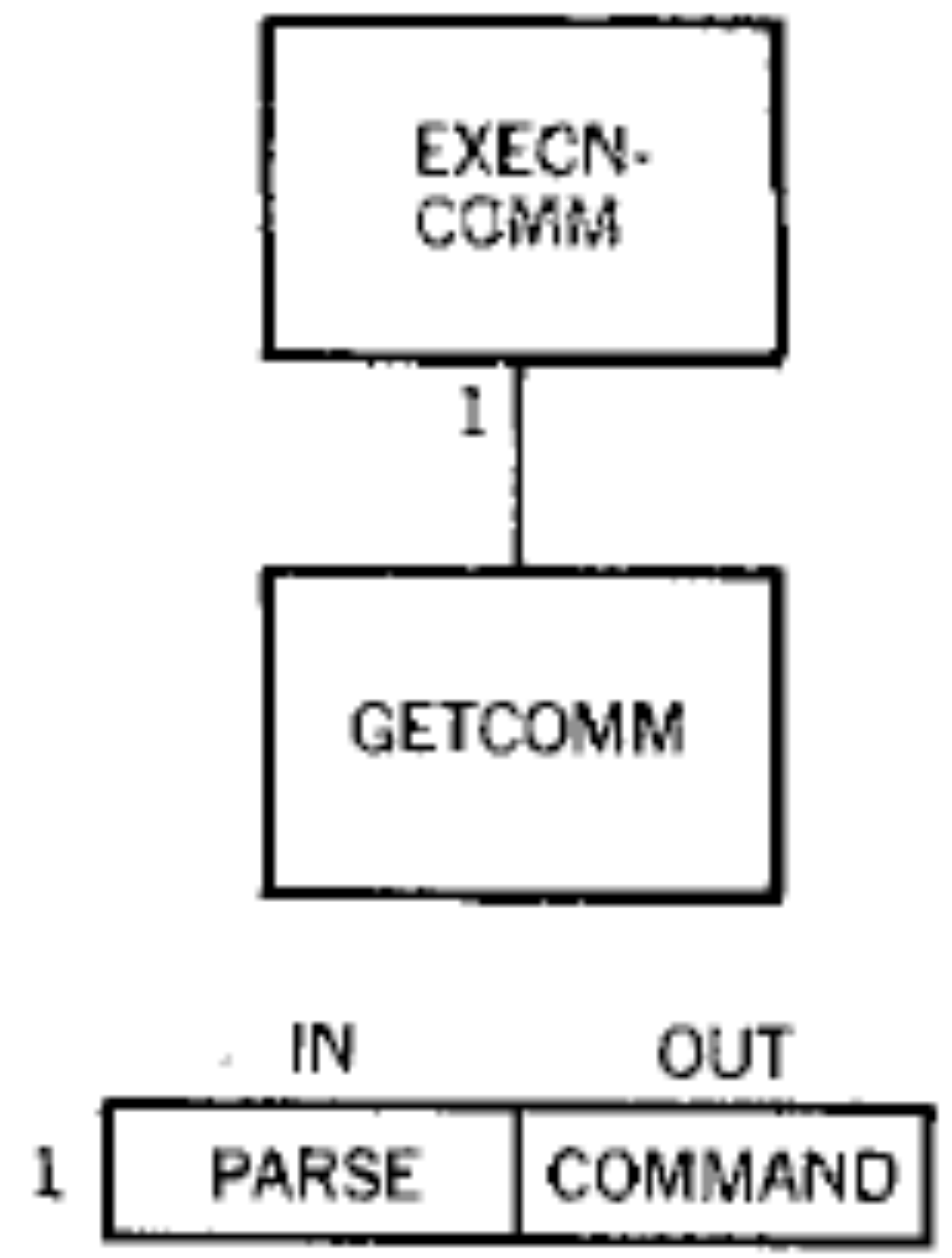
- Data Coupling
    - Connections that pass necessary data
- Stamp Coupling
    - Connections that pass data more than necessary
- Control Coupling
    - Connections that pass data and control elements

- Obviously Stamp Coupling couples more Data Coupling
- Control Coupling also couples more than data coupling
    - Information Hiding

Content Coupling

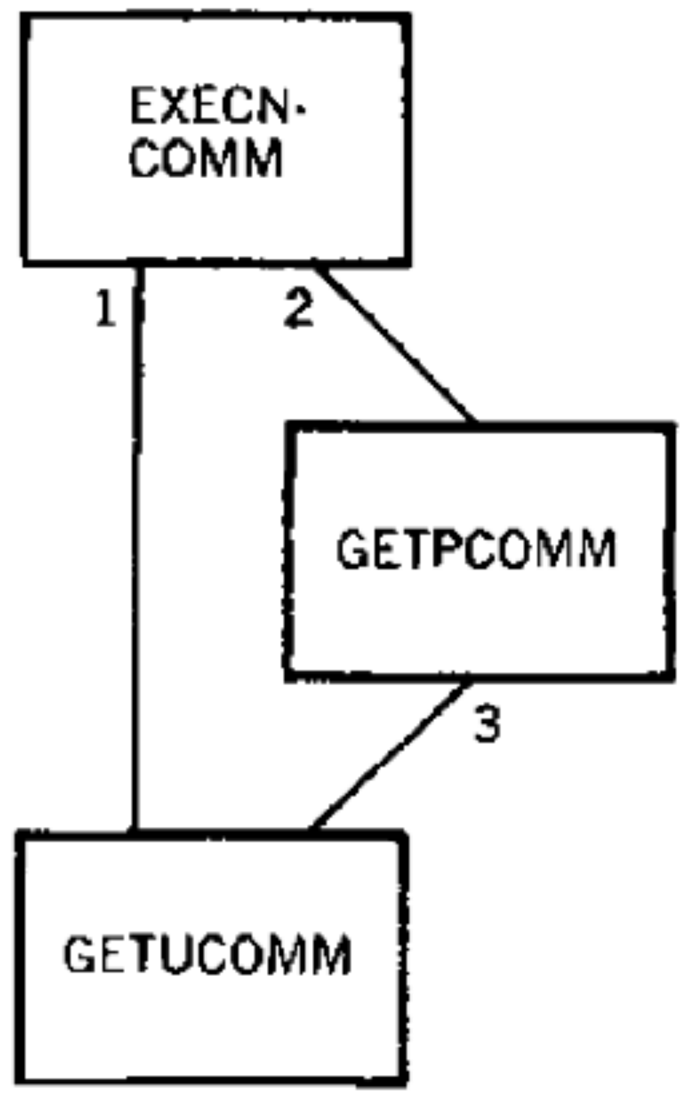
- Hybrid of data and control elements
    - Modification of one module’s code by another module
    - The target module is very dependent on the modifying module.

Figure 4 Control-coupled modules



Control - Coupled

Figure 5 Simplified coupling



	IN	OUT
1		COMMAND
2		COMMAND
3		COMMAND

Simplified coupling

	Interface Complexity	**Type of Connection**	**Type of Communication**
Low	Simple, Obvious	To module, by name	Data
COUPLING			Control
High	Complicated, Obscure	To Internal Elements	Hybrid

Strength of Coupling

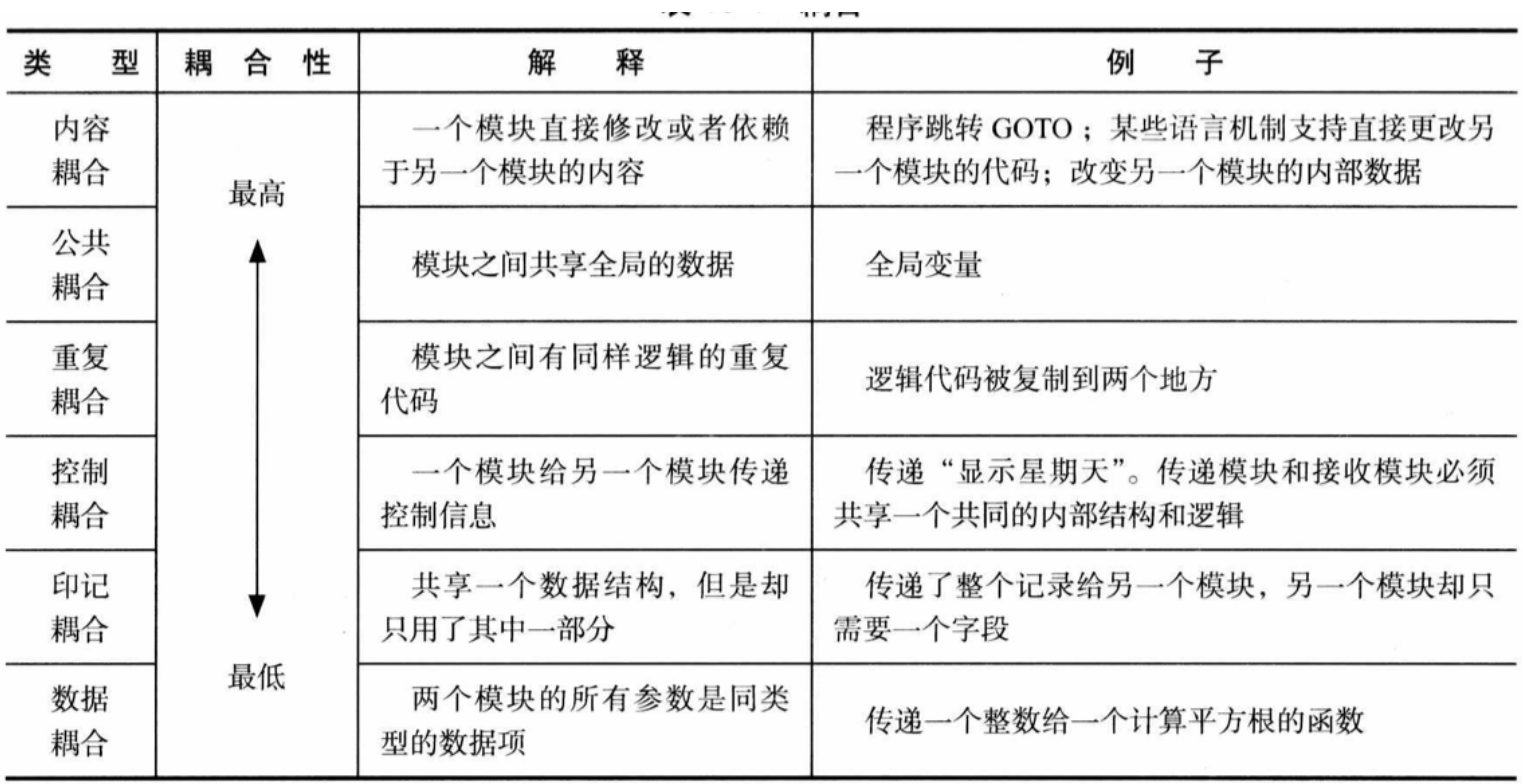
Degree of Coupling in Structured Method

- Content Coupling
- Common Coupling
- Control Coupling
- Stamp Coupling
- Data Coupling

(High Coupling – Bad)

(Low Coupling – Good)

- No coupling is best!



类　　型	耦　合　性	解　　释	例　　子
内容耦合	最高	一个模块直接修改或者依赖于另一个模块的内容	程序跳转 GOTO；某些语言机制支持直接更改另一个模块的代码；改变另一个模块的内部数据
公共耦合	↑	模块之间共享全局的数据	全局变量
重复耦合	│	模块之间有同样逻辑的重复代码	逻辑代码被复制到两个地方
控制耦合	│	一个模块给另一个模块传递控制信息	传递"显示星期天"。传递模块和接收模块必须共享一个共同的内部结构和逻辑
印记耦合	↓	共享一个数据结构，但是却只用了其中一部分	传递了整个记录给另一个模块，另一个模块却只需要一个字段
数据耦合	最低	两个模块的所有参数是同类型的数据项	传递一个整数给一个计算平方根的函数

耦合

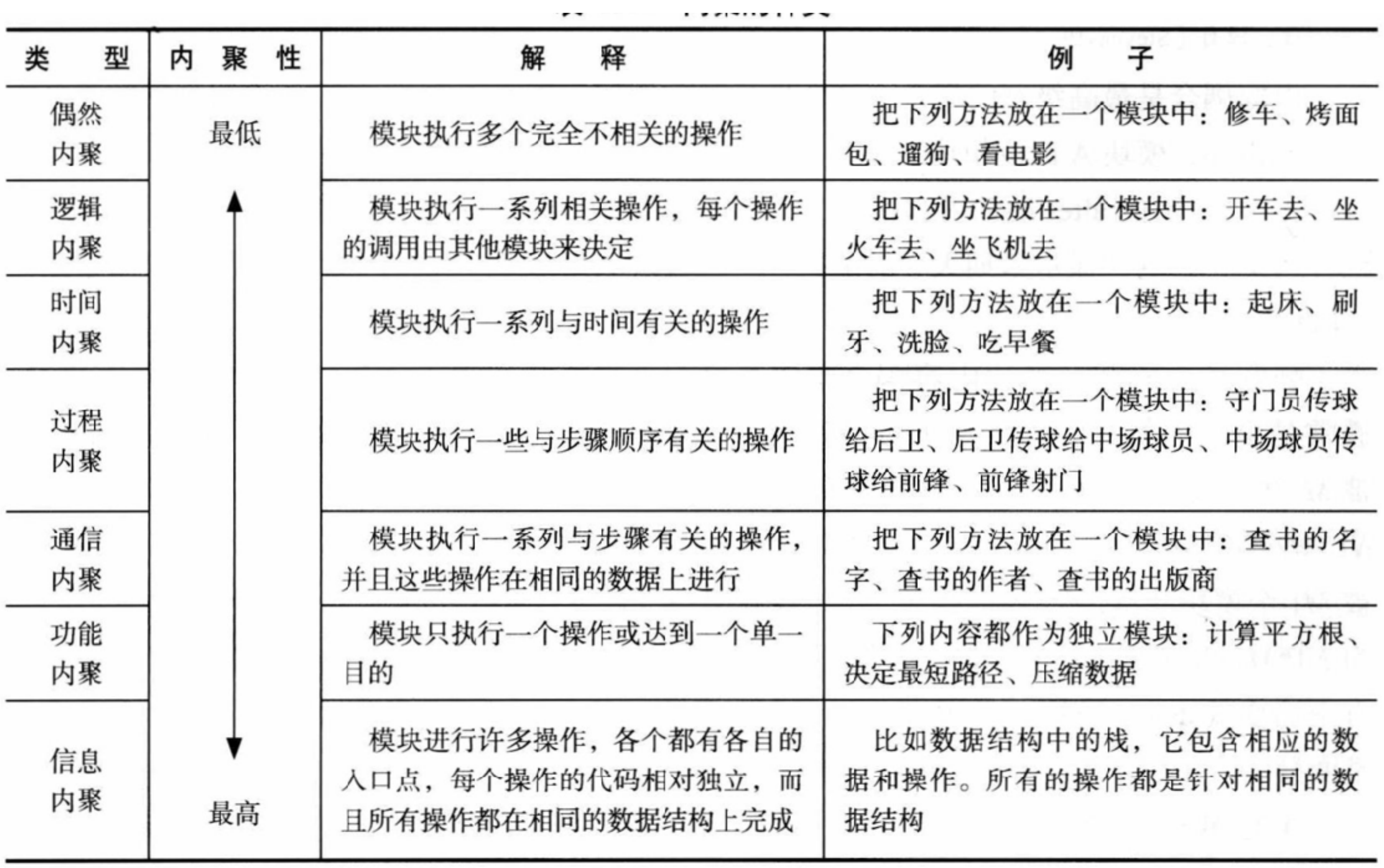
内聚

Cohesiveness

- Ways of achieving independent modules
  - REDUCE relationships between elements NOT IN the same module
  - INCREASE relationships between elements IN same module
- BINDING - a measure of cohesivness

Degree of Cohesiveness (Types of Binding)

• Coincidental	Low cohesion
Logical	
Temporal	
Communicational	
• Functional	
• 信息内聚	High cohesion



类　型	内 聚 性	解　释	例　子
偶然内聚	最低	模块执行多个完全不相关的操作	把下列方法放在一个模块中：修车、烤面包、遛狗、看电影
逻辑内聚		模块执行一系列相关操作，每个操作的调用由其他模块来决定	把下列方法放在一个模块中：开车去、坐火车去、坐飞机去
时间内聚		模块执行一系列与时间有关的操作	把下列方法放在一个模块中：起床、刷牙、洗脸、吃早餐
过程内聚		模块执行一些与步骤顺序有关的操作	把下列方法放在一个模块中：守门员传球给后卫、后卫传球给中场球员、中场球员传球给前锋、前锋射门
通信内聚		模块执行一系列与步骤有关的操作，并且这些操作在相同的数据上进行	把下列方法放在一个模块中：查书的名字、查书的作者、查书的出版商
功能内聚		模块只执行一个操作或达到一个单一目的	下列内容都作为独立模块：计算平方根、决定最短路径、压缩数据
信息内聚	最高	模块进行许多操作，各个都有各自的入口点，每个操作的代码相对独立，而且所有操作都在相同的数据结构上完成	比如数据结构中的栈，它包含相应的数据和操作。所有的操作都是针对相同的数据结构

内聚

思想的应用

模块化思想的应用:

- 低耦合处理
  - 软件体系结构的分层设计中:
    - 不同层的模块之间仅能通过程序调用与数据传递 实现交互,不能共享数据(例如 Model 层建立一个数据对象并将引用传递给 Logic 层使用)否则会导致公共耦合。
  - 软件体系结构的逻辑包设计中:
    - 依据功能的特点将三个层次进一步划分为更小 的包,而不是只使用 Presentation、Logic 和 Model 三个包,可以通过包分割实现接口最小化,这能去除不必要的耦合。

- 软件体系结构的物理包设计中:
    - 将不同包的重复内容独立为单独的包以消除重 复,避免产生隐式的重复耦合;
- 详细设计中对象创建者的选择:
    - 如果两个对象 A、B 间已有比较高的耦合度了, 那么使用 A 创建 B 或者反之就不会带来额外的耦合度。这就是表 12-4 内容的核 心思想——不增加新的耦合。
- 详细设计中选择控制风格:
    - 解除界面与逻辑对象的直接耦合。

模块化思想的应用:

- 高内聚处理
  - 软件体系结构的分层设计中:
    - 三个层次都是高内聚的,一个处理交互任务, 一个处理业务逻辑,一个处理数据持久化。
  - 软件体系结构的逻辑包设计中:
    - 将三个层次进一步划分为更小的包,可以实 现每个更小的包都是高内聚的。
  - 详细设计中抽象类的职责:
    - 要求状态与方法紧密联系就是为了达到高内聚 (信息内聚)。
  - 详细设计中使用控制风格:
    - 控制风格分离了控制逻辑,可以实现业务逻辑对 象的高内聚(功能内聚)。因为封装了控制逻辑,所以控制器对象承载了不 可避免的顺序内聚、通信内聚和逻辑内聚,这就要求控制器对 象必须是受控的,也是它们为什么倾向于对外委托而不是自己进行业务计算的原因。

Outline

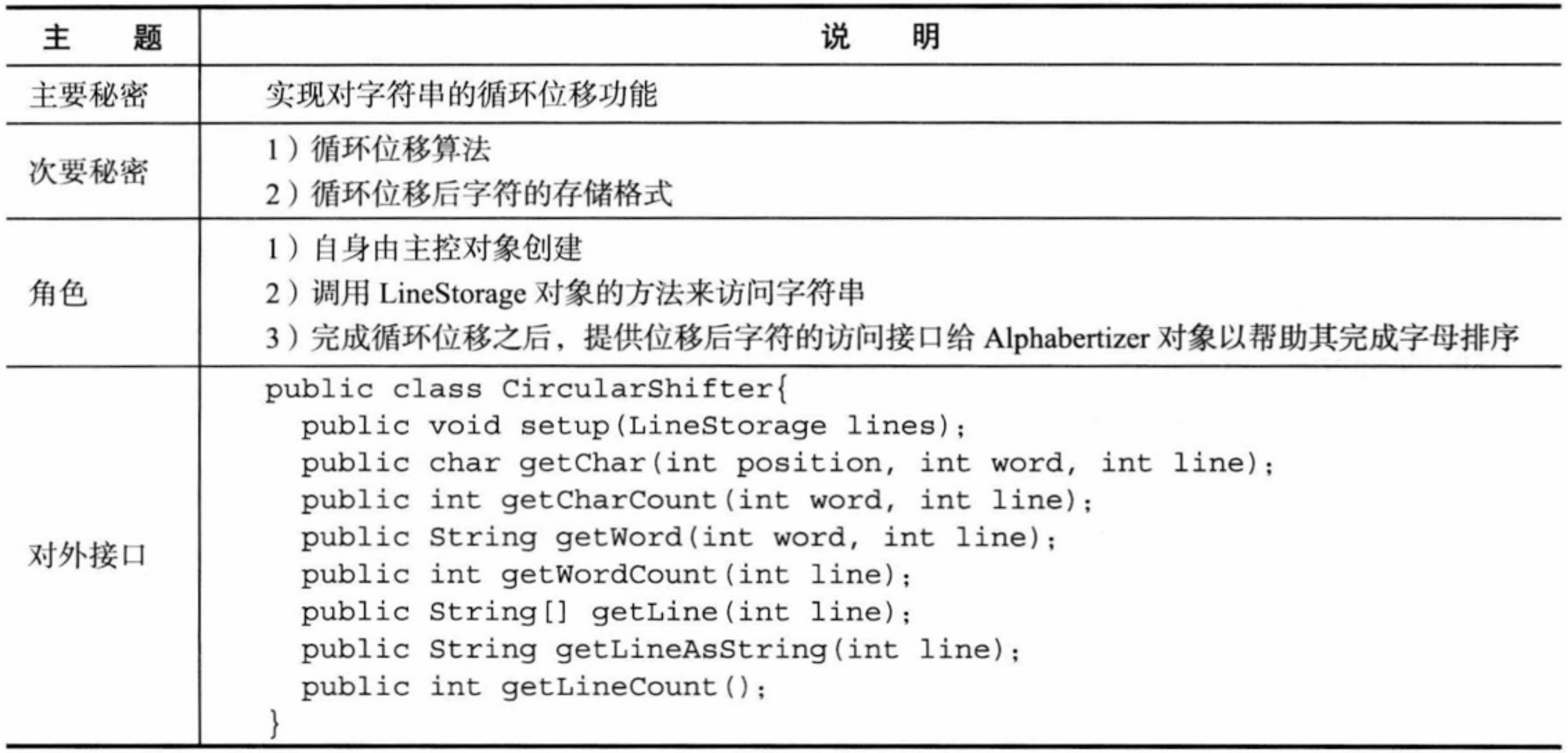
- 模块化与信息隐藏思想
- 结构化的模块化
- 结构化的信息隐藏
  - Module Guide
  - 思想的应用

Module Guide

- 模块的主要秘密
- 模块的次要秘密
- 模块的角色
- 模块的对外接口

- 模块的主要秘密：
  - 主要秘密描述的是这个模块所要实现的用户需求。是设计者对用户需求的实现的一次职责分配。有了这个描述以后，我们可以利用它检查我们是否完成所有的用户需求，还可以利用它和需求优先级来决定开发的次序。
- 模块的次要秘密：
  - 次要秘密描述的是这个模块在实现职责时候所涉及的具体的实现细节。包括数据结构，算法，硬件平台等信息。

- 模块的角色：
  - 　描述了独立的模块在整个系统中所承担的角色，所起的作用。以及与哪些模块有相关联的关系。
- 模块的对外接口：
  - 　模块提供给别的模块的接口。



主　　题	说　　明
主要秘密	实现对字符串的循环位移功能
次要秘密	1）循环位移算法 2）循环位移后字符的存储格式
角色	1）自身由主控对象创建 2）调用 LineStorage 对象的方法来访问字符串 3）完成循环位移之后，提供位移后字符的访问接口给 Alphabertizer 对象以帮助其完成字母排序

对外接口	`public class CircularShifter{`
	`  public void setup(LineStorage lines);`
	`  public char getChar(int position, int word, int line);`
	`  public int getCharCount(int word, int line);`
	`  public String getWord(int word, int line);`
	`  public int getWordCount(int line);`
	`  public String[] getLine(int line);`
	`  public String getLineAsString(int line);`
	`  public int getLineCount();`
	`}`

循环位移模块的模块说明

Quick example of a Complex System

- The A-7E Aircraft
- Extremely complex Onboard Flight Program
- Limited memory
- Real-time constraints

Parnas’s Experience

- “When we tried to work without the guide, ……responsibilities ended up either in two modules or in none. With the module guide, further progress on the design has revealed relatively few oversights”.

Parnas's Experience

- Integration testing took only a week
- Only nine bugs were discovered.
- Location of bugs isolated to single module.
- All bugs were quickly fixed.

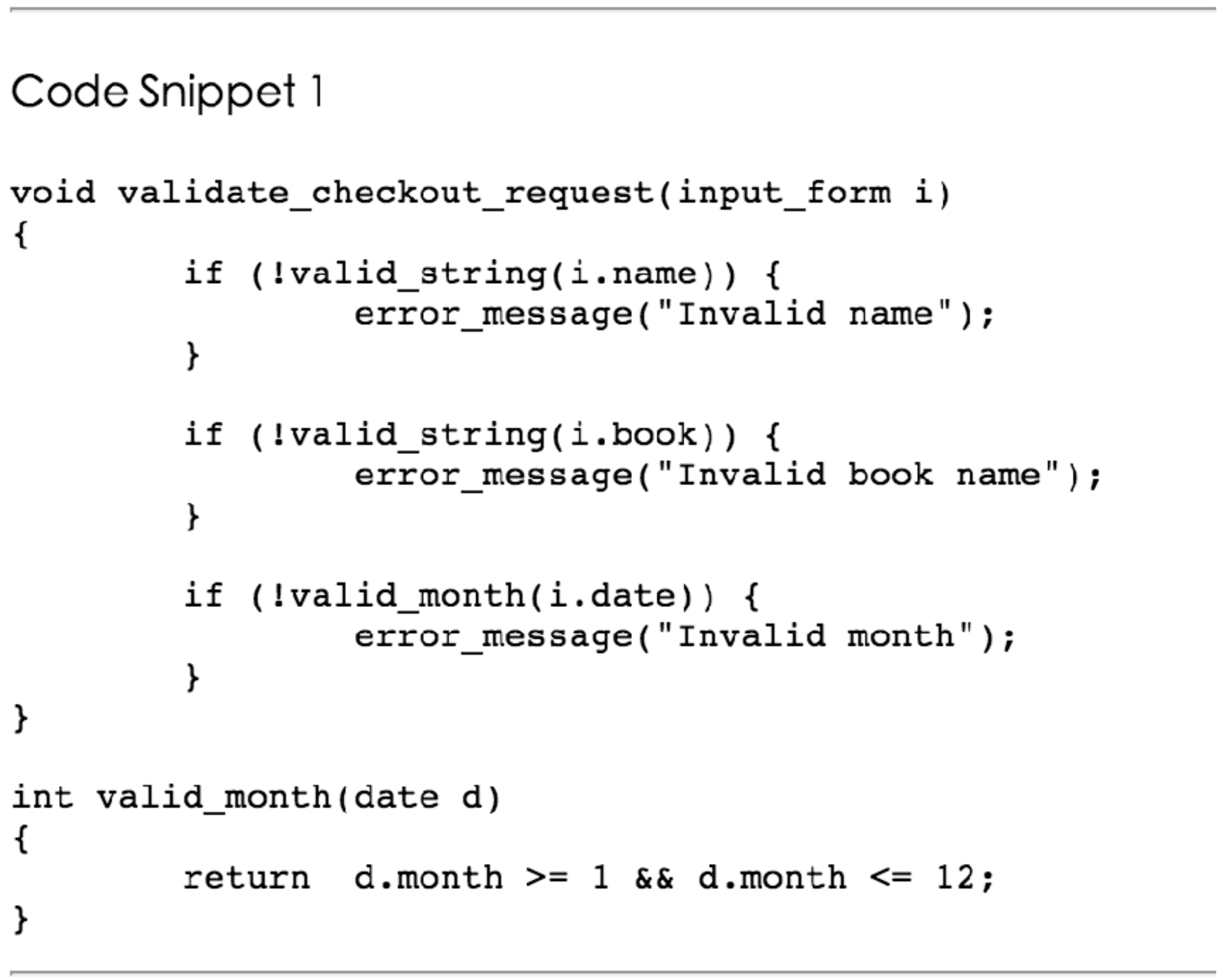
信息隐藏思想的应用

- 信息隐藏处理
  - 在软件体系结构设计的分层设计中:
    - 经验表明软件系统的界面是最经常变化的, 其次是业务逻辑,最稳定的是业务数据。这就是分层风格建立 Prensentation、 Logic 和 Model 三个层次的原因,它们体现了决策变化的划分类型,它们之间的 依赖关系符合各自的稳定性。
  - 在软件体系结构设计的物理包设计中:
    - 消除重复可以避免重复耦合,同时可以 避免同一个设计决策出现在多个地方——这意味着该决策没有被真正地隐藏 (这也是控制耦合比数据耦合差的原因)。

- 在软件体系结构设计的物理包设计中:
  - 建立独立的安全包、通信包和数据库连接包,是为了封装各自的设计决策——安全处理、网络通信与数据库处理。
- 在软件体系结构设计与详细设计中:
  - 严格要求定义模块与类的接口,可以便利开发,更是为了实现信息隐藏。
- 在详细设计中使用控制风格:
  - 专门用控制器对象封装关于业务逻辑的设计决策, 而不是将其拆散分布到整个对象网络中去。

Exercise

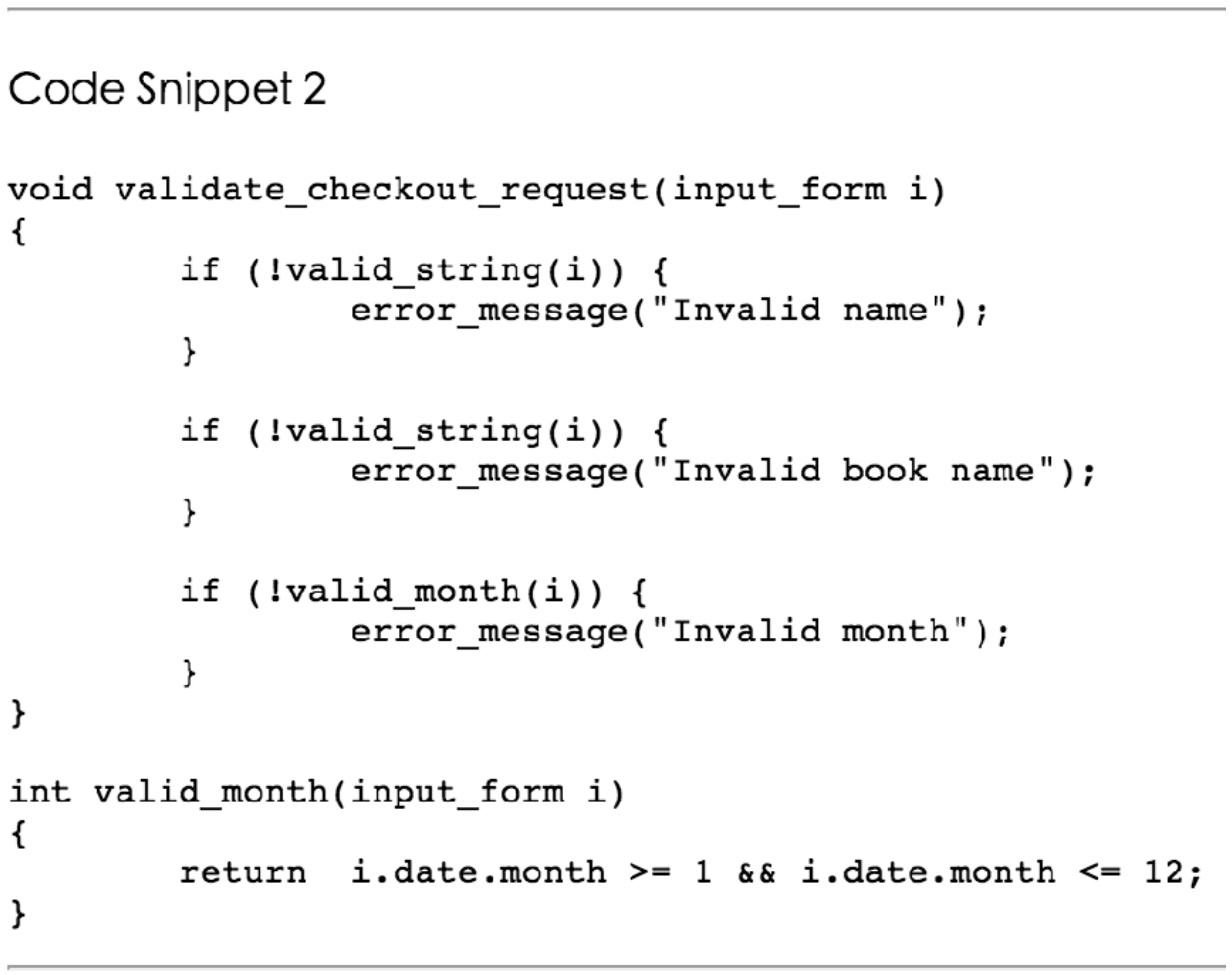
- For each of the following code snippets, identify the types of coupling or cohesion shown:



Code Snippet 1

```
void validate_checkout_request(input_form i)
{
        if (!valid_string(i.name)) {
                error_message("Invalid name");
        }

        if (!valid_string(i.book)) {
                error_message("Invalid book name");
        }

        if (!valid_month(i.date)) {
                error_message("Invalid month");
        }
}

int valid_month(date d)
{
        return  d.month >= 1 && d.month <= 12;
}
```

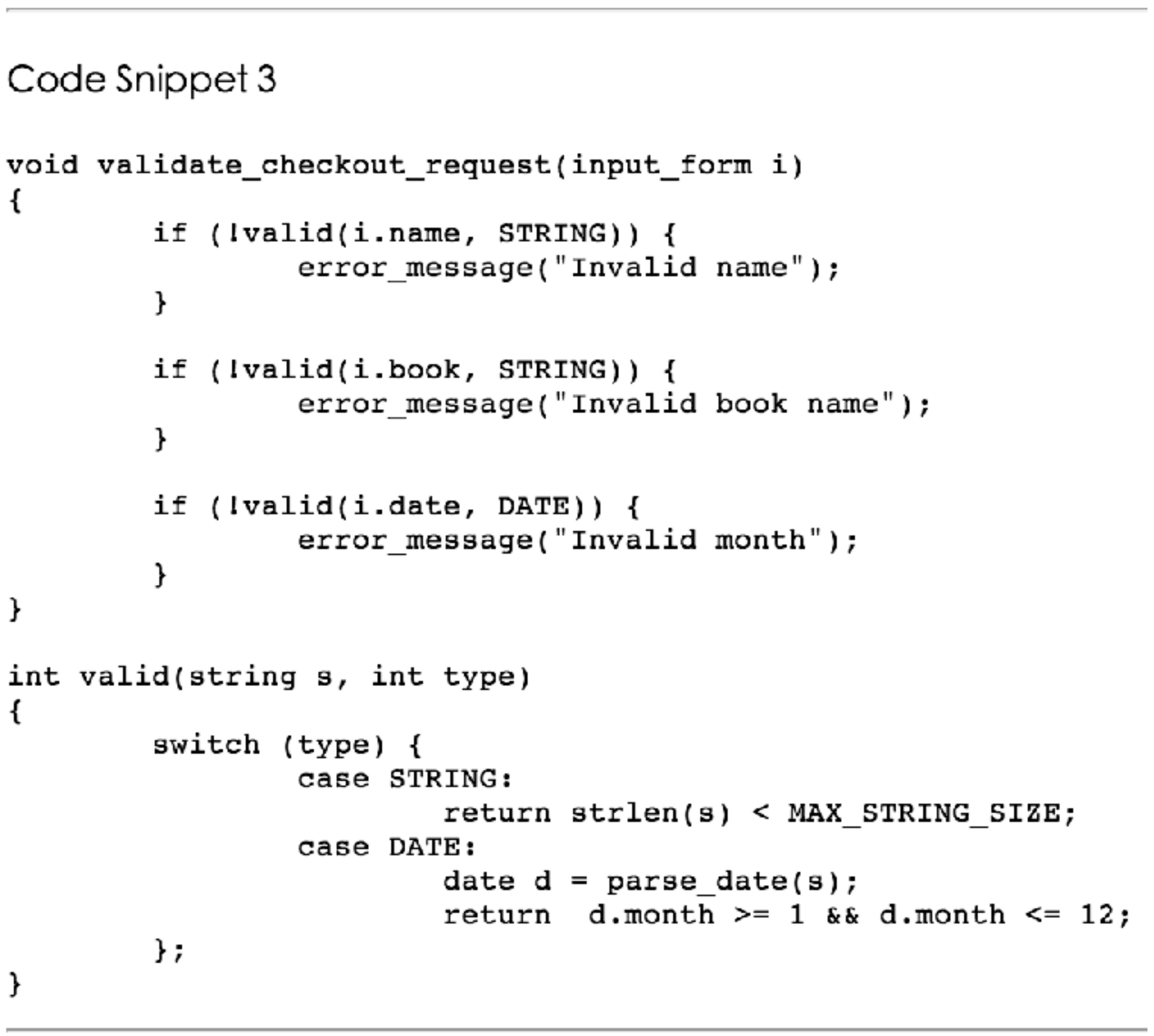
通信内聚、印记耦合



Code Snippet 2

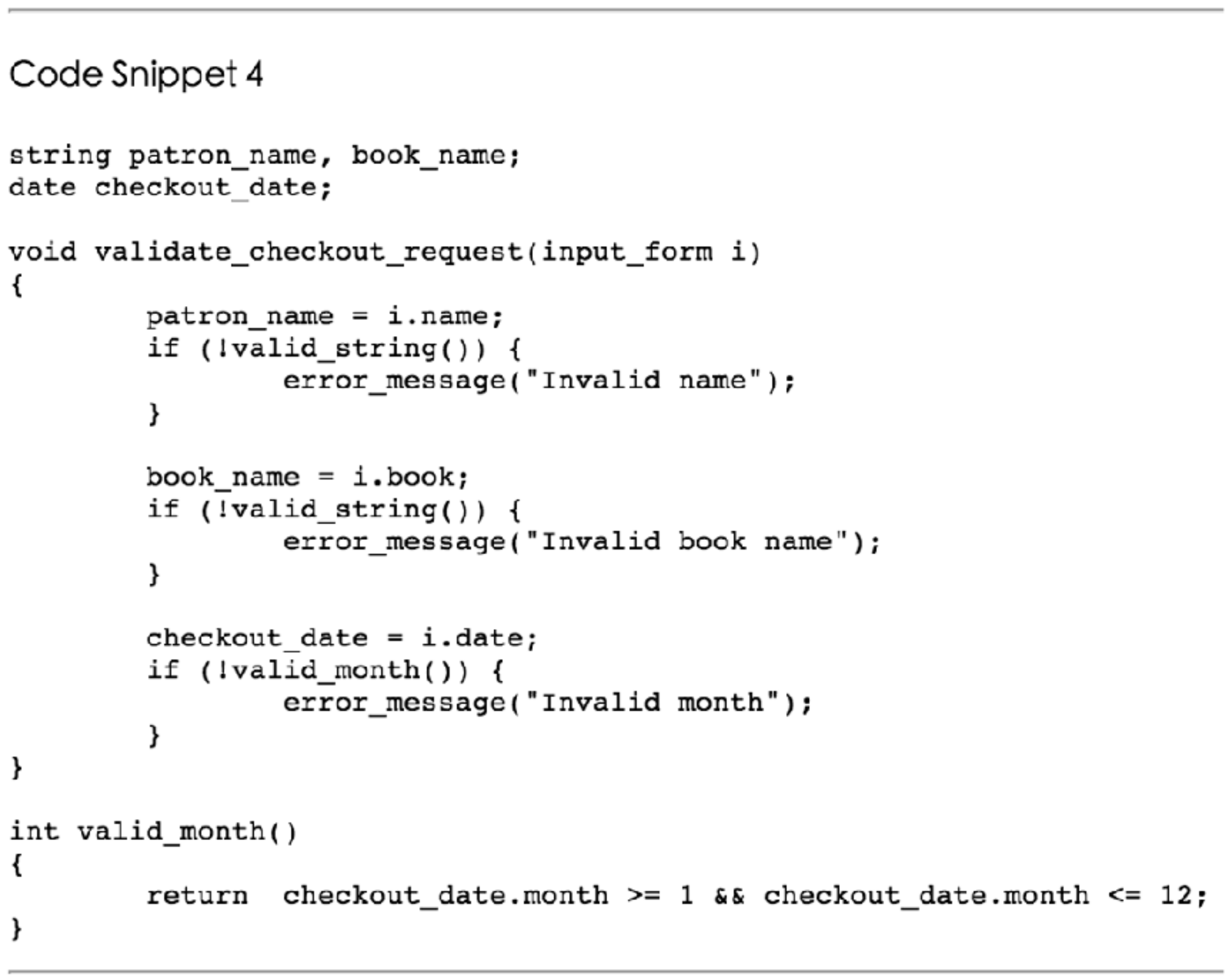
```
void validate_checkout_request(input_form i)
{
        if (!valid_string(i)) {
                error_message("Invalid name");
        }

        if (!valid_string(i)) {
                error_message("Invalid book name");
        }

        if (!valid_month(i)) {
                error_message("Invalid month");
        }
}

int valid_month(input_form i)
{
        return  i.date.month >= 1 && i.date.month <= 12;
}
```

印记耦合


Code Snippet 3

```
void validate_checkout_request(input_form i)
{
        if (!valid(i.name, STRING)) {
                error_message("Invalid name");
        }

        if (!valid(i.book, STRING)) {
                error_message("Invalid book name");
        }

        if (!valid(i.date, DATE)) {
                error_message("Invalid month");
        }
}

int valid(string s, int type)
{
        switch (type) {
                case STRING:
                        return strlen(s) < MAX_STRING_SIZE;
                case DATE:
                        date d = parse_date(s);
                        return  d.month >= 1 && d.month <= 12;
        };
}
```

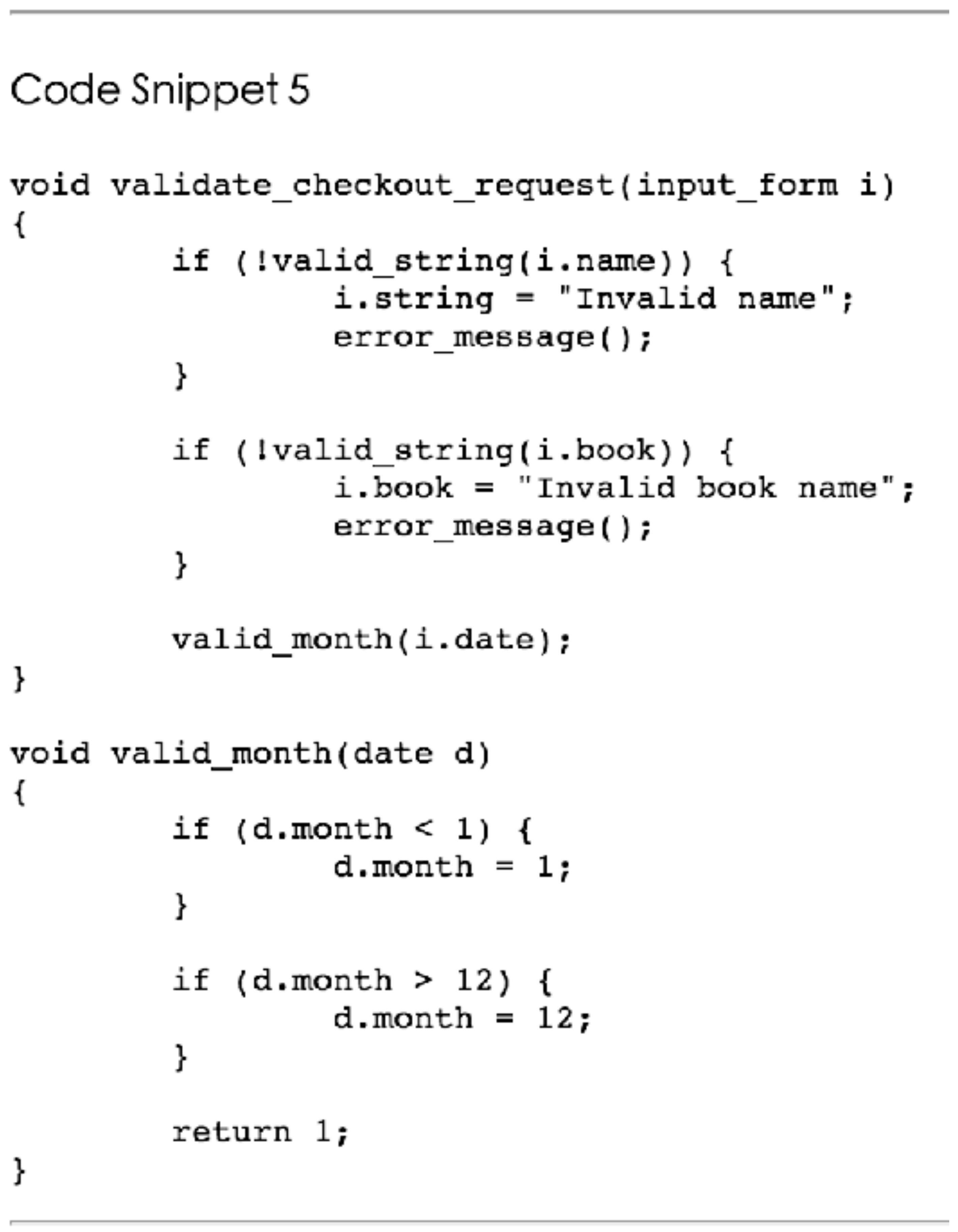
控制耦合



Code Snippet 4

```
string patron_name, book_name;
date checkout_date;

void validate_checkout_request(input_form i)
{
        patron_name = i.name;
        if (!valid_string()) {
                error_message("Invalid name");
        }

        book_name = i.book;
        if (!valid_string()) {
                error_message("Invalid book name");
        }

        checkout_date = i.date;
        if (!valid_month()) {
                error_message("Invalid month");
        }
}

int valid_month()
{
        return  checkout_date.month >= 1 && checkout_date.month <= 12;
}
```

公共耦合


Code Snippet 5

```
void validate_checkout_request(input_form i)
{
        if (!valid_string(i.name)) {
                i.string = "Invalid name";
                error_message();
        }

        if (!valid_string(i.book)) {
                i.book = "Invalid book name";
                error_message();
        }

        valid_month(i.date);
}

void valid_month(date d)
{
        if (d.month < 1) {
                d.month = 1;
        }

        if (d.month > 12) {
                d.month = 12;
        }

        return 1;
}
```

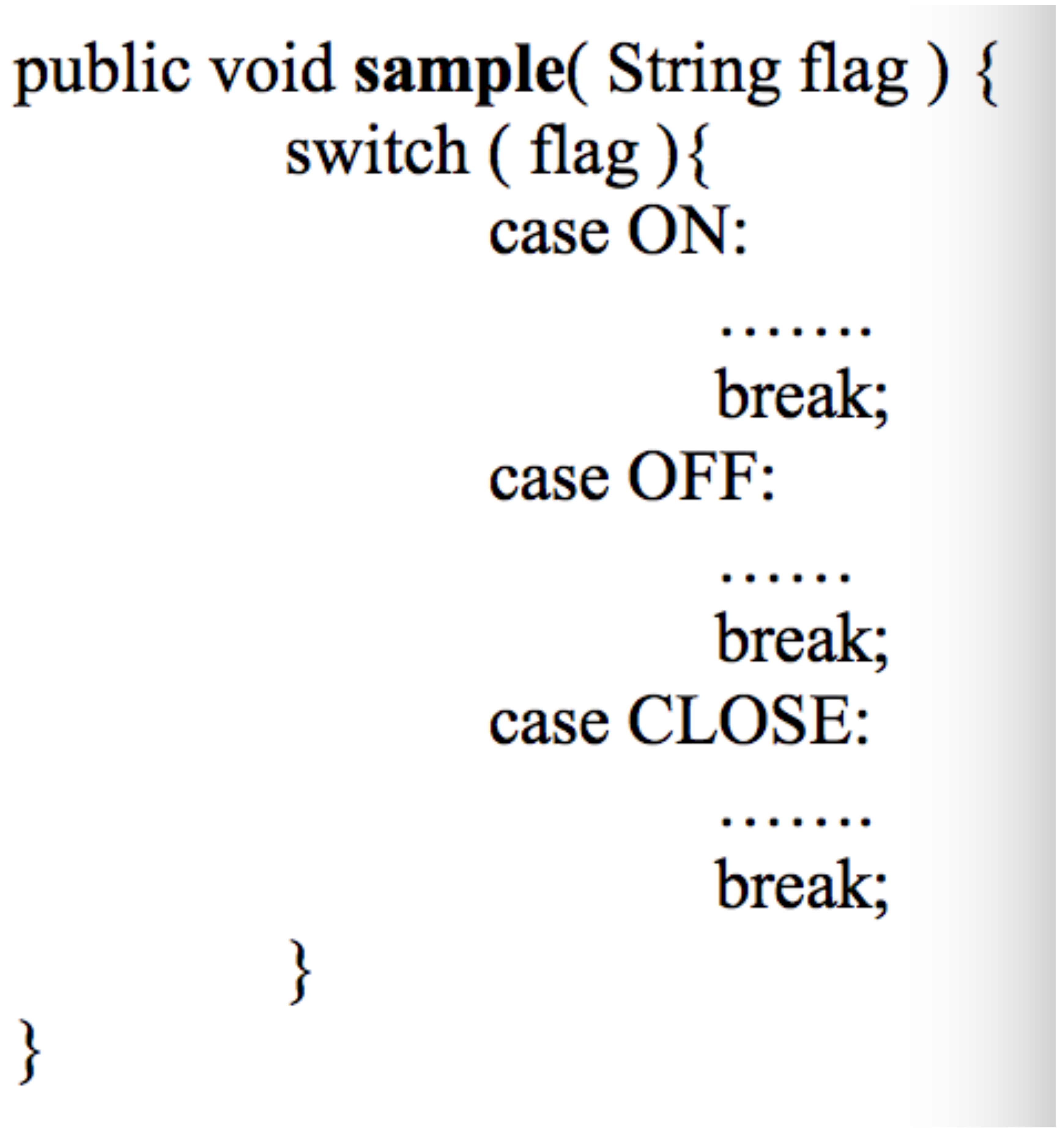
内容耦合

Code Snippet 6 -- 重复耦合

```
• void validate_checkout_request(input_form i)
• {
•         int len = 0;
•    boolean valid_string = false;
•          len = i.name.length();
•          char arr1[] = new char[len];
•         for(char c: arr1){
•              if c是小写字母 valid_string = true;
•         }
•         if (!valid_string) {
•          error_message("Invalid name");
•   }
•         len = i.book.length();
•          char arr2[] = new char[len];
•         for(char c: arr2){
•              if c是小写字母 valid_string = true;
•         }
•    if (! valid_string) {
•          error_message("Invalid book name");
•    }
•    if (!valid_month(i.date)) {
•          error_message("Invalid month");
•    }
• }
```

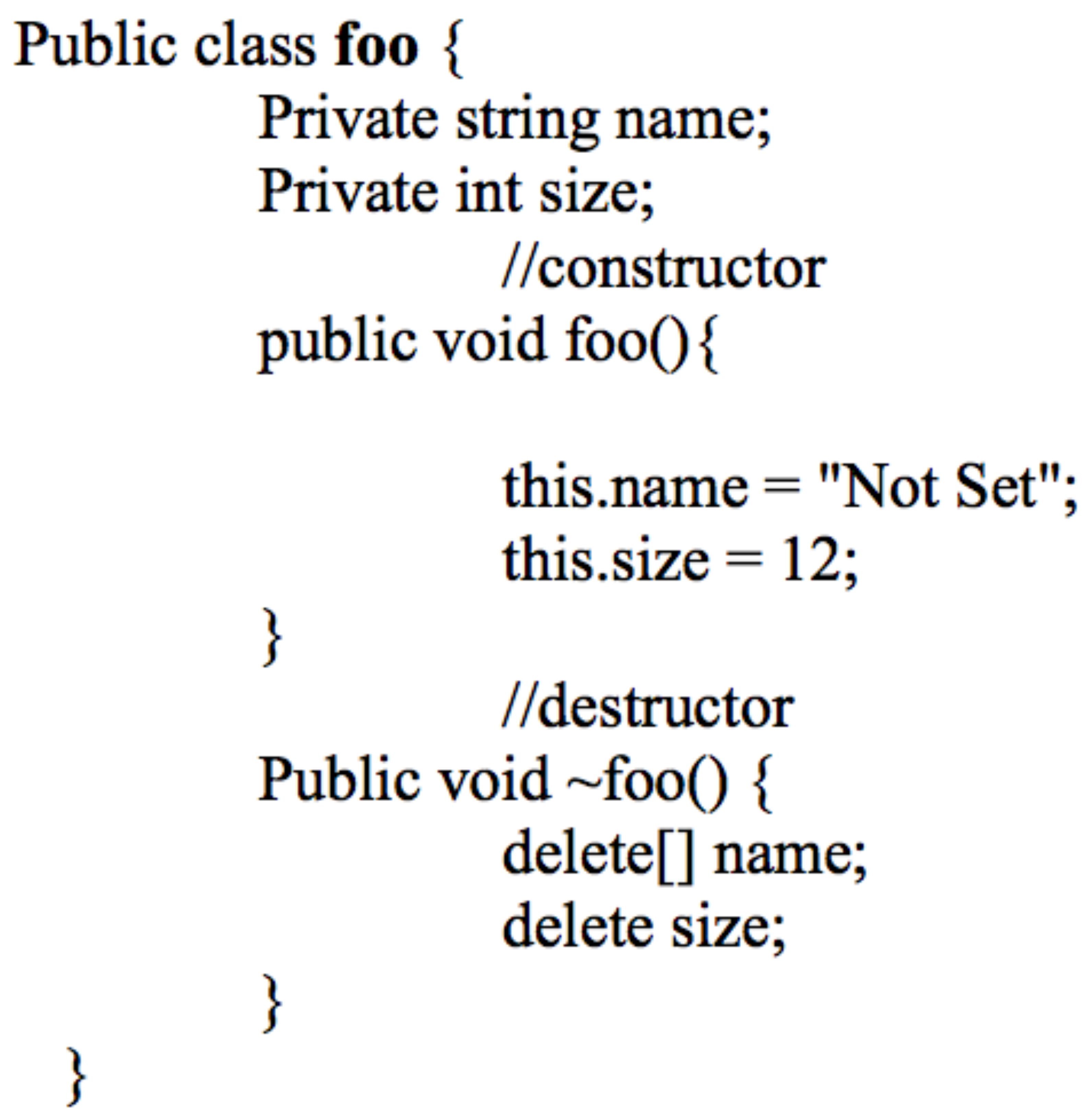
```
Public class Rous{

   public static int findPattern( String text, String pattern)
      { … }
   public static int average( Vector numbers )
      { … }
   public static OutputStream openFile( String fileName )
      { … }
}
```

偶然内聚



```
public void sample( String flag ) {
          switch ( flag ){
                    case ON:
                              .......
                              break;
                    case OFF:
                              ......
                              break;
                    case CLOSE:
                              .......
                              break;
          }
}
```

逻辑内聚

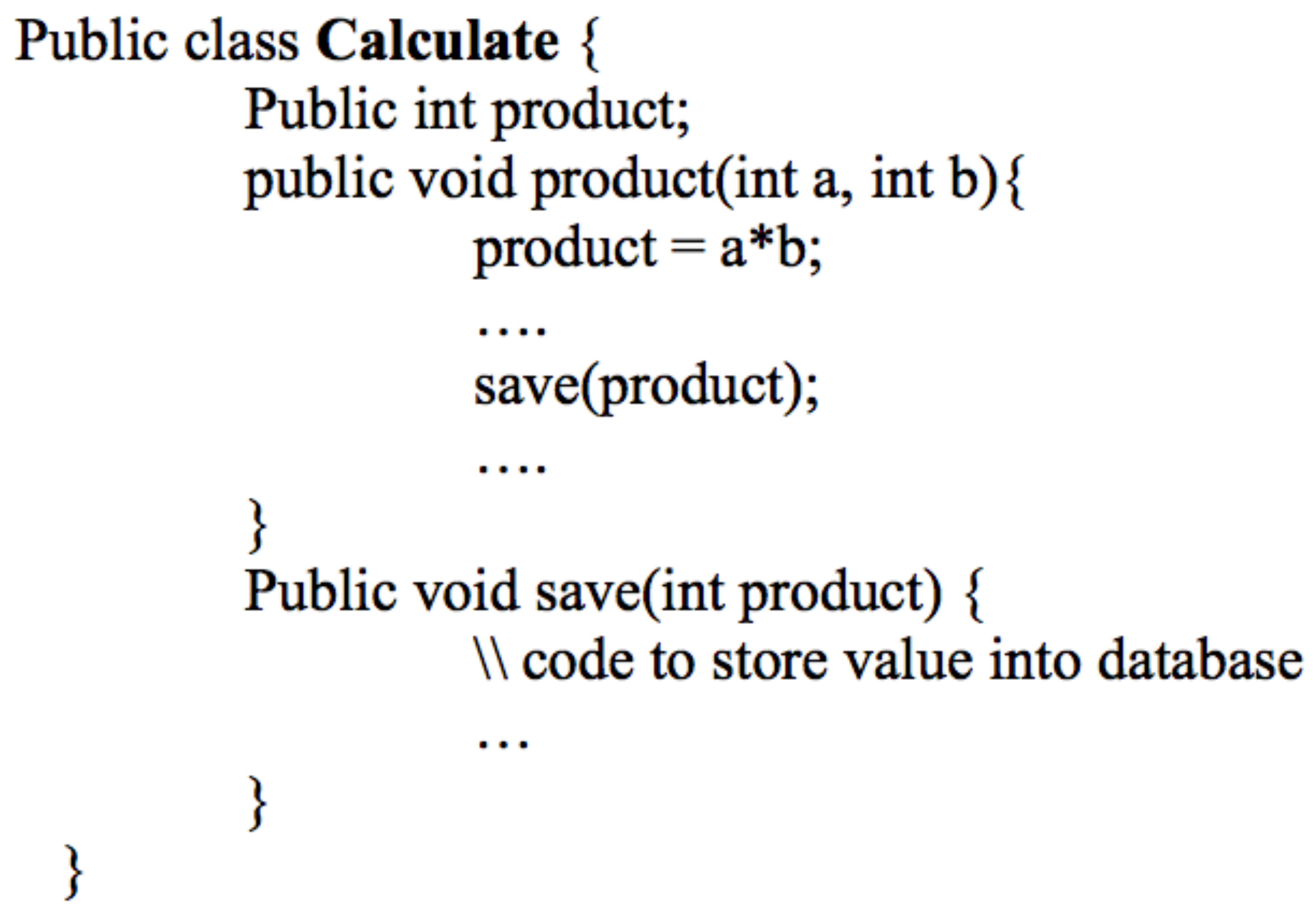


```
Public class foo {
        Private string name;
        Private int size;
                //constructor
        public void foo(){

                this.name = "Not Set";
                this.size = 12;
        }
                //destructor
        Public void ~foo() {
                delete[] name;
                delete size;
        }
  }
```

时间内聚

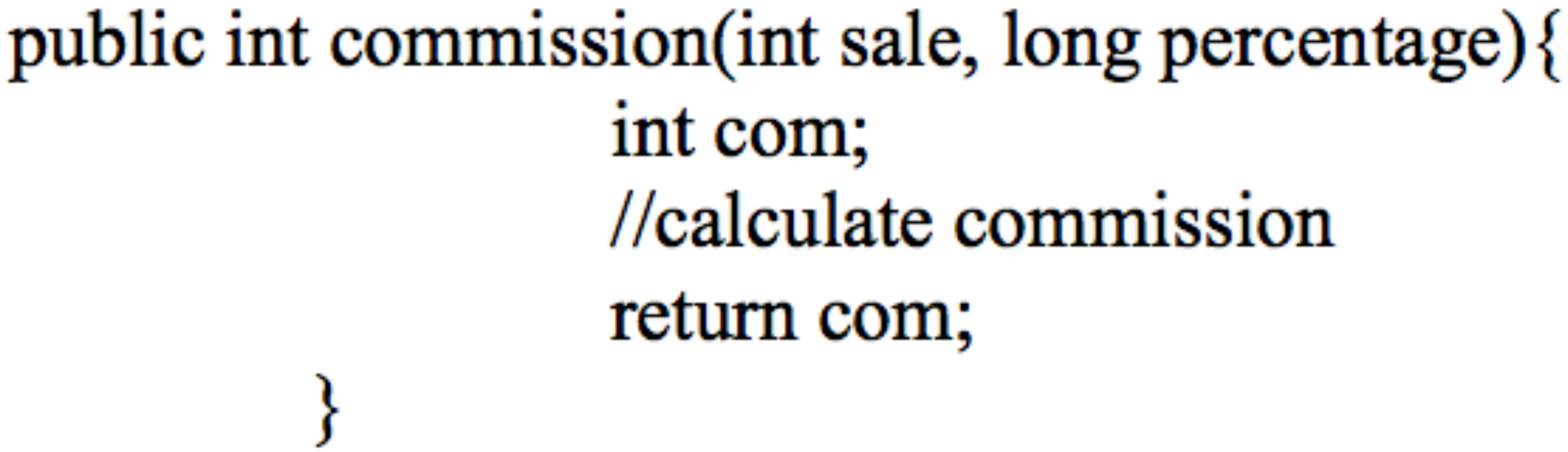
```
void MonthEnd()
{
  Report ExR = InitExpenseReport();
  Report rr = InitRevenueReport();
  Report EmpR = InitEmployeeReport();

  EmpR.Init();
  rr.Init();
  ExR.SetEmployees(true);

  if (ExR.GetReportParams())
    EmpR.GetReportParams();

  SendToPrinter(rr);
  SendToPrinter(ExR);
  SendToPrinter(EmpR);
}
```

过程内聚



```
Public class Calculate {
        Public int product;
        public void product(int a, int b){
                product = a*b;
                ….
                save(product);
                ….
        }
        Public void save(int product) {
                \\ code to store value into database
                …
        }
}
```

通信内聚



```
public int commission(int sale, long percentage){
                    int com;
                    //calculate commission
                    return com;
          }
```

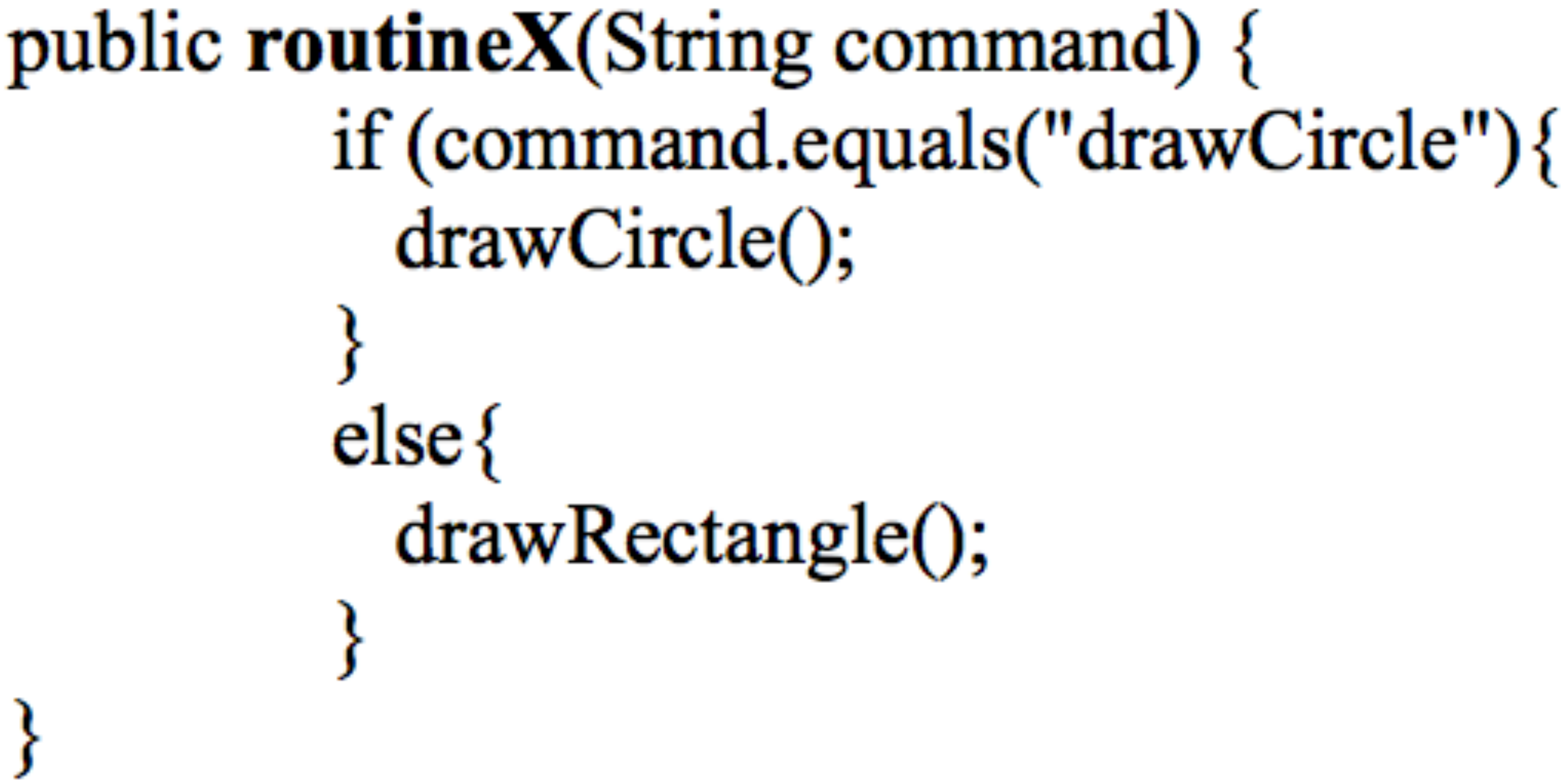
功能内聚

```
Public interface Addressee{
          ………….
          Public abstract String getName ();
          Public abstract String getAddress ();
          ….
}

Public class Employee implements Addressee {….}
```

信息内聚

```
public class Vector3D{
        public int x, y, z;
         ...
}
public class Arch {
         private Vector3D baseline;
         ...
       void slant(int newY){
                baseline.x = 10;
                baseline.y = 13;
       }
}
```

内容耦合

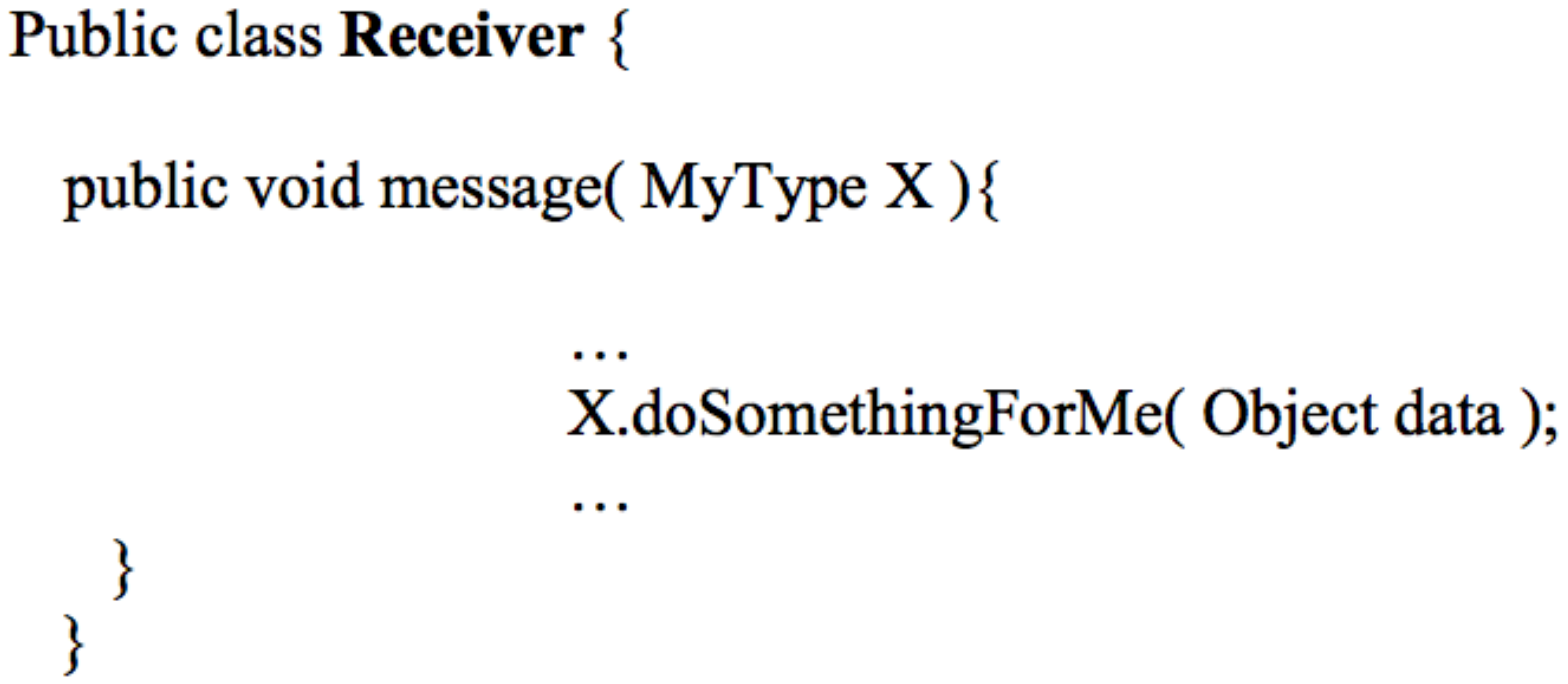


```
public routineX(String command) {
          if (command.equals("drawCircle"){
            drawCircle();
          }
          else{
            drawRectangle();
          }
}
```

控制耦合

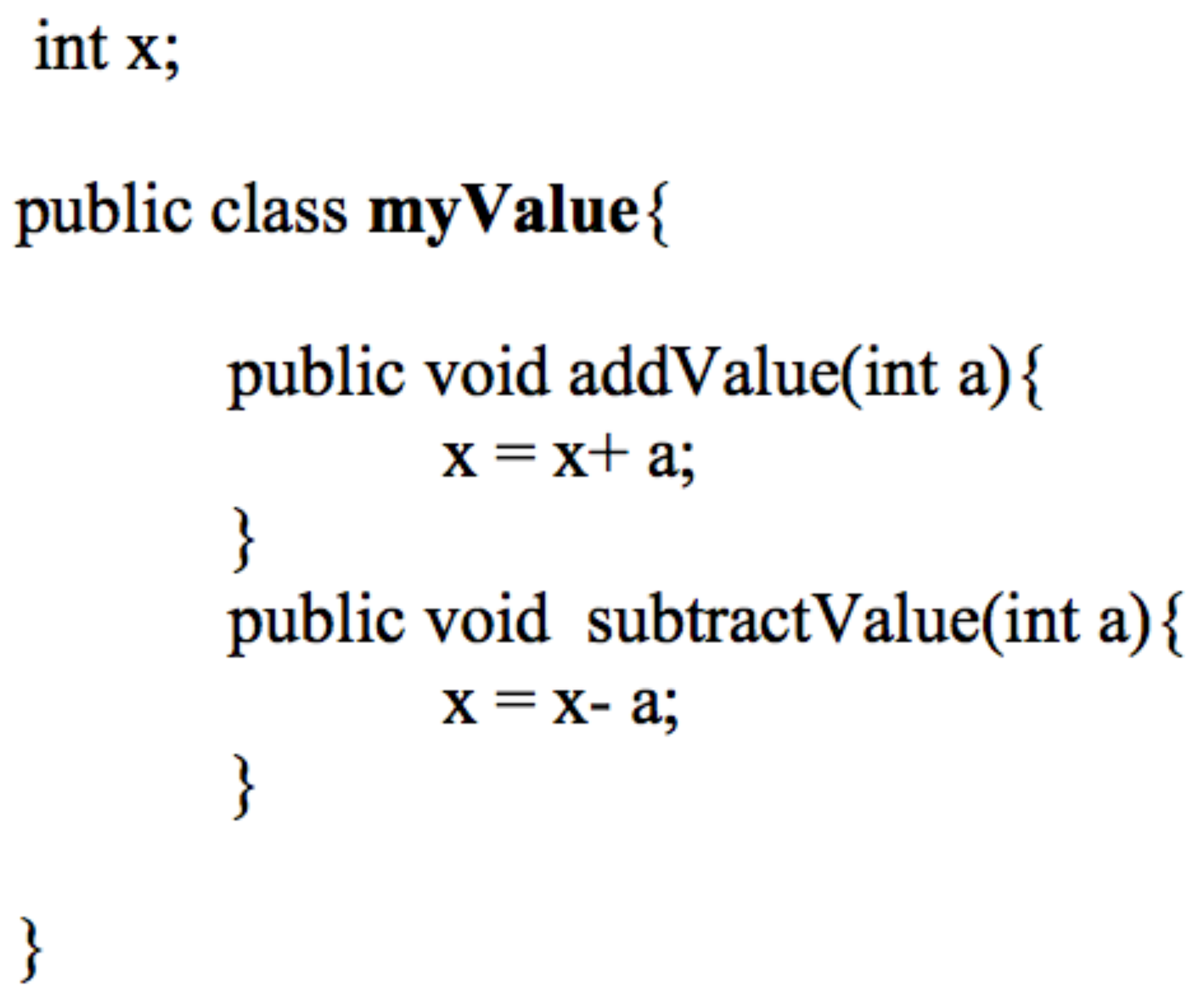
```
public class Employee{
        public String name, emailID.
  ...
}


public class Emailer{
          public void sendEmail(Employee e, String text)
          {...}
  ...
}
```

印记耦合



```
Public class Receiver {

  public void message( MyType X ){

                        …
                        X.doSomethingForMe( Object data );
                        …
    }
  }
```

数据耦合



```
 int x;

public class myValue{

        public void addValue(int a){
                x = x+ a;
        }
        public void  subtractValue(int a){
                x = x- a;
        }

}
```

公共耦合