Structured programming in software project

Covered topics

- Overview
- Structure of a program
- Separation of concerns

Objectives

- After this lesson, students will be able to:
  - Recall the term of structured programming
  - Summarize the relationships between software qualities, algorithms and the structure of program.
  - Formulate the separation of concerns.

I. OVERVIEWS

1. Introduction
2. Concepts of structured programming
3. Advantages
4. Disadvantages

1. Introduction

- Software engineering = problem solving activities
  - Understanding the problem
  - Designing an algorithm as a solution
  - Implementing the algorithm in a computer program
- Algorithm: sequence of steps that take from the input to the output for solving a problem
  - Correct: provide a correct solution according to the specifications
  - Finite : terminate
  - General: work for every instance of a problem
  - Efficient : use few resources (time, memory, bandwith, etc.)

→ **Need of a structured approach**

2.1. What is structured programming?

- Initially: programming without the use of the GOTO statement
- Then: a method of writing a computer program to minimize the problem complexity:
  - top-down analysis for problem solving
  - modularization for program structure and organization
  - structured code for the individual modules

2.1. What is structured programming?

- Now: a method of designing software components / program elements and their relationships to:
  - Minimize complexity
  - Adapt to change (identify modifications for additional functionalities or correcting errors)
  - Improve the reliability and clarity of programs

2.2. Structured programming: structured design of software

- Conceptualizing a problem into several well-organized elements of solution (mostly based on “divide and conquer” strategy)
- Maintaining the unified structure at different levels
  - Problem solving: top-down, bottom-up, middle-out
  - Program abstraction and organization: modules, services, functions, objects, …
  - Program elements: structured code

3. Advantages of structured programming

- The sequence of operations is simple to trace, thus facilitating debugging.
- There are a finite number of structures with standardized terminology.
  - Structures lend themselves easily to building subroutines.
  - The set of structures is complete; that is, a programs can be written using these structures.
- Structures are self-documenting and, therefore, easy to read.
- Structures are easy to describe in flowcharts, syntax diagrams, pseudo code, and so on.
- Structured programming results in increased programmer productivity-programs can be written faster.

4. Disadvantages of structured programming

- Some high-level languages (Pascal, C, Java, Lisp, …) accept the structures directly; while others require an extra stage of translation.
- In some cases, structured programs may execute slower and require more memory than the unstructured equivalent.
- Some problems (a minority) are more difficult to solve using only the three structures rather than a brute-force "spaghetti" approach.
- Nested structures can be difficult to follow.

II. STRUCTURE OF PROGRAMS

1. Structure of computer programs
2. Control structures
3. Data structures
4. Functions and procedures as program elements

1. Structure of computer programs

Program

- Library, package
  - File, class
    - Function, procedure, method
      - Block
        - » Statement
          - Expression
            - Word, token

Book

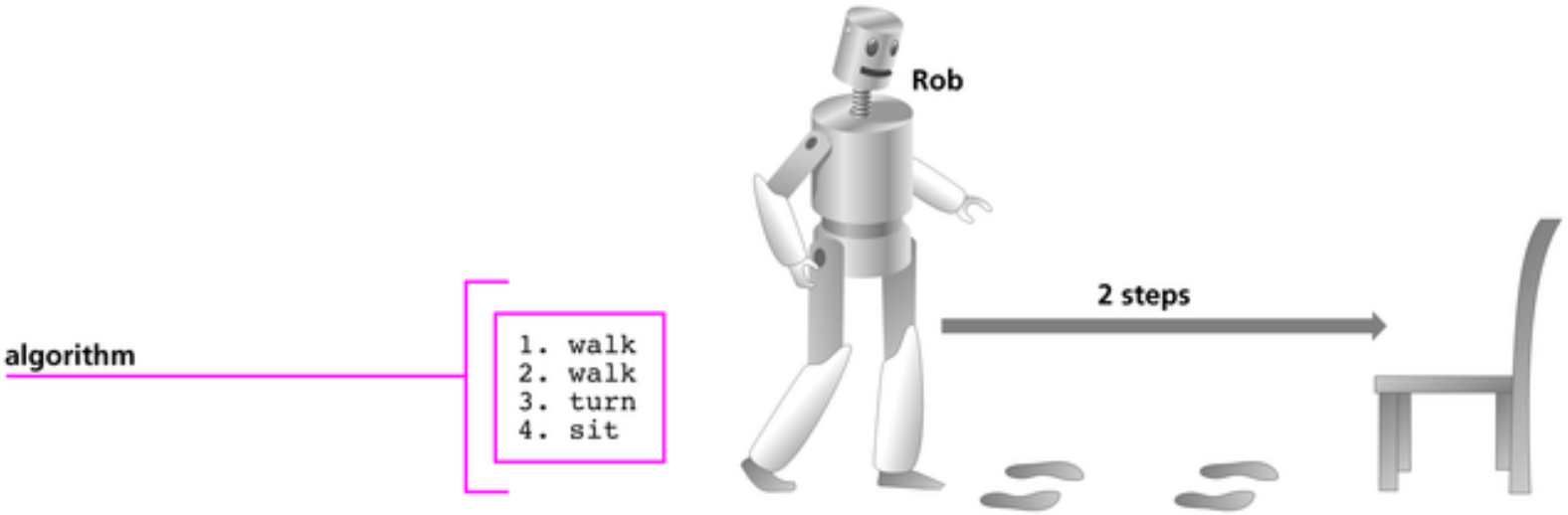
- Part
  - Chapter
    - Section
      - Paragraph
        - » Sentence
          - Phrase
            - Word

2. Control structures

- Computer program represents an algorithm resolving a given problem.
- All computer programs, no matter how simple or how complex, are written using one or more of three basic structures:
  - Sequence
  - Selection
  - Repetition
- These structures are called control structures or logic structures, because they control the program logic

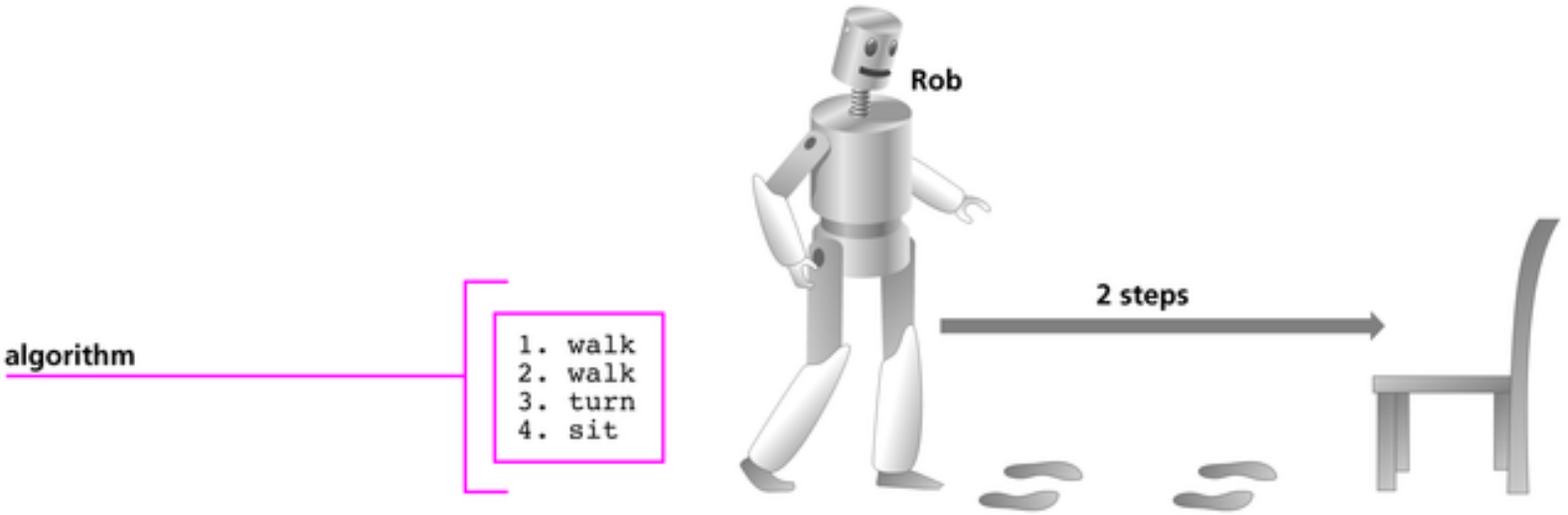
2.1. Sequence structure

- The sequence structure in a computer program directs the computer to process the statements one after another, in the order listed in the program



2.1. Sequence structure

- A statement may be:
  - Assignment statement
  - Input /output statement
  - Composite statement

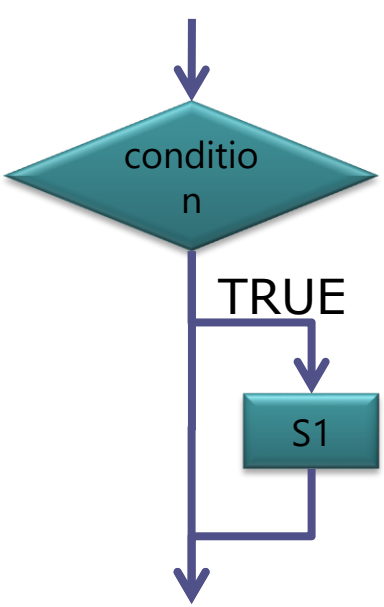


2.2. Selection structure

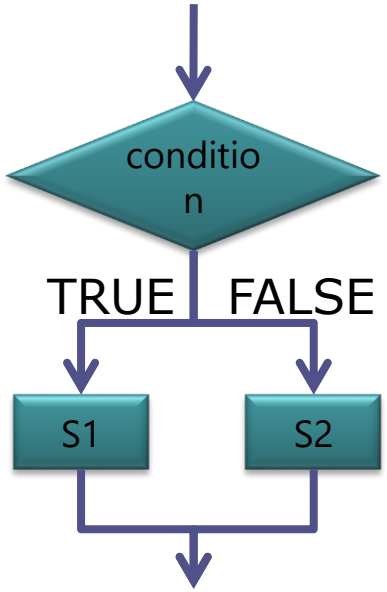
- Make a decision, and then take an appropriate action based on that decision
- Provide the appropriate action to take based on the result of that decision
- The decision depends on various condition values

2.2. Selection structure

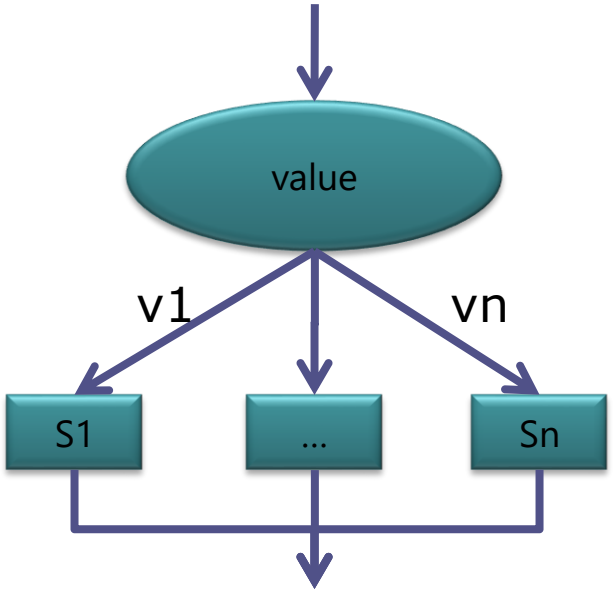
Single input – single output



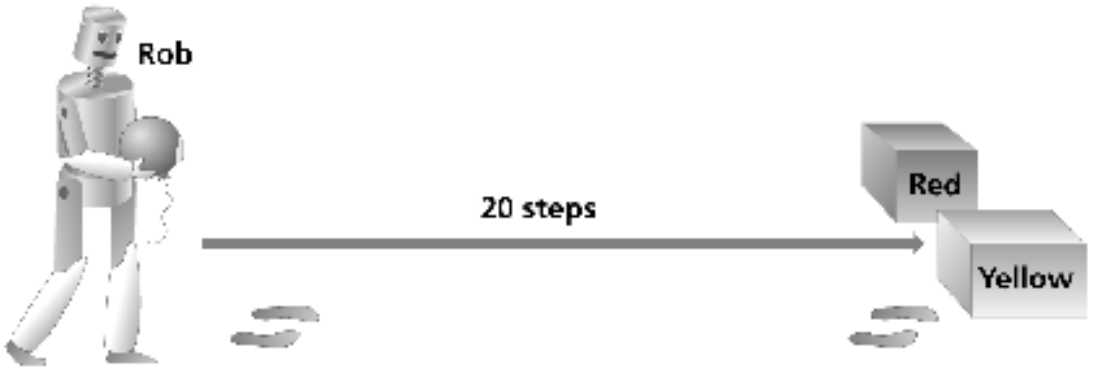
Single input – double output



Single input – multiple output



2.2. Selection structure



algorithm

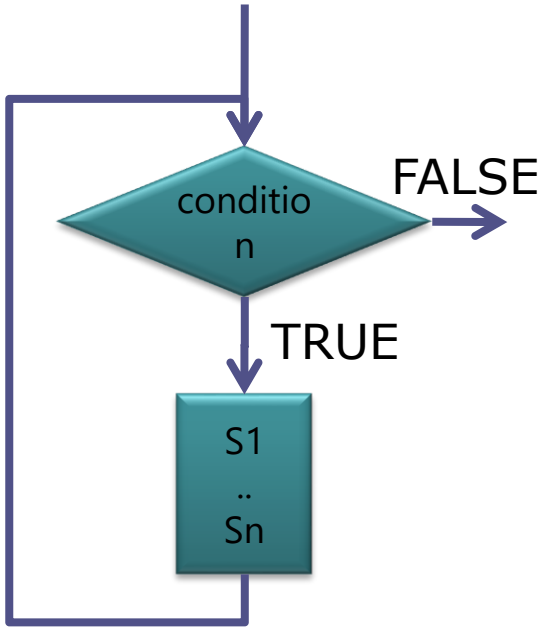
```
1.  repeat 20 times:
        walk
2.  if the balloon is red, do this:
        drop the balloon in the red box
    otherwise, do this:
        drop the balloon in the yellow box
3.  turn
4.  repeat 20 times:
        walk
5.  turn
```

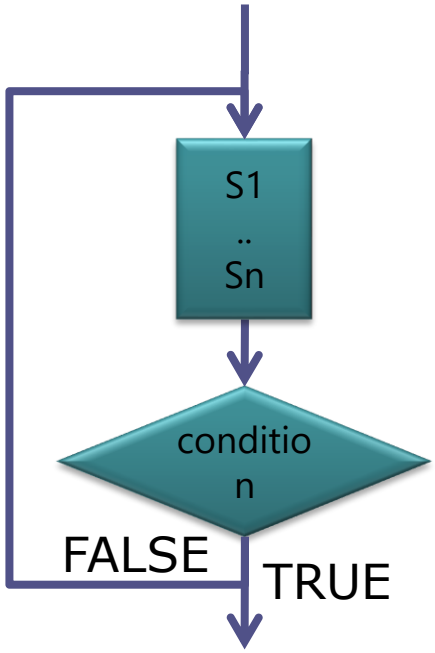
indent the instructions within the `if` and `otherwise` sections of a selection structure

2.3. Repetition structure

- Allow the programmer to specify that an action should be repeated, depending on the condition value
- When used in a program, the repetition structure, also referred to as a loop, directs the computer to repeat one or more statements until some condition is met, at which time the computer should stop repeating the statements

2.3. Repetition structure





2.3. Repetition structure

- Case 1: The repetition number is known in advance

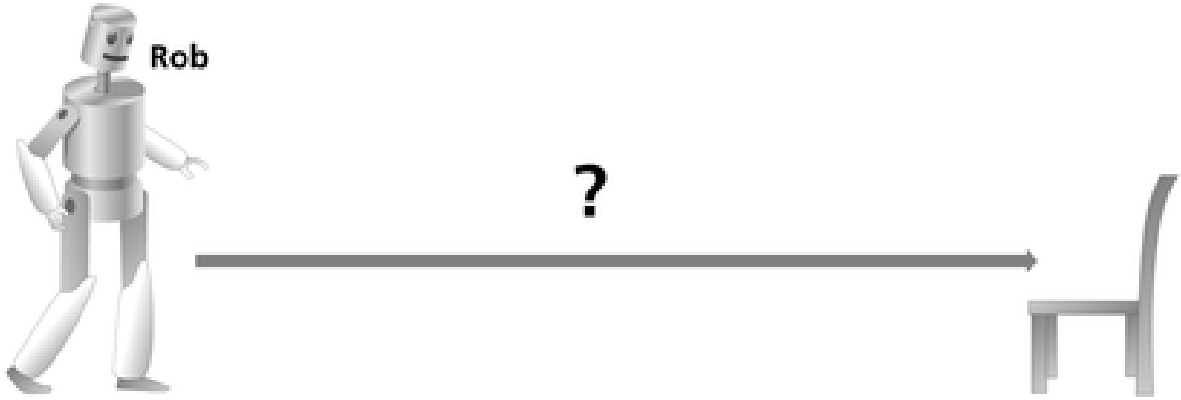
algorithm

```
1. repeat 50 times:
       walk
2. turn
3. sit
```

indent the instructions within a repetition structure

2.3. Repetition structure

- Case 2: The repetition number is not known in advance
- Statements in the body of this repetition structure are executed repeatedly as long as the loop-continuation test is evaluated to false
  - The condition is first evaluated: may be none of these statements is executed
  - Otherwise, these statements are executed at least one time



algorithm

```
1. repeat until you are directly in front of the chair:
       walk
2. turn
3. sit
```

3. Data structures

- How to choose or devise the appropriate data structures for a problem ?
  - Algorithms will have to manipulate data in some way → The way we choose to store and organize our data (i.e. data structure) directly affects the efficiency of our algorithm
  - Classic data structures : design, implementation and use

Type

- Primitive type:
  - Integer, Boolean, String, …
- Composite type
  - Tuple
- Abstract data type: data structure that is defined indirectly by the operations that may be performed on it, and the mathematical properties of those operations
  - Array
  - List
  - Tree
  - Hash
  - Graph

4. Functions and procedures as program elements

- Functions and procedures are part of computer program. They can be custom-defined.
- A function or a procedure is built out of control structures in order to manipulate on the determined data structures.
- Functions are really mathematical relations that map every input to exactly one output. Functions are designed to return their output value.
- Procedures are recipes for computation that perform side effects.
- Either function or procedure **is used to represent a concern.**

Example: Functions and procedures

- C, C++, Java : no distinct
- Pascal, .NET:
  - function returns value
  - procedure doesn’t return value.
- DBMS:
  - procedures (SPROCs) : stored compiled queries
  - functions (UDFs): built-in piece of expressions used to build queries

III. SEPARATION OF CONCERNS

1. Principles
2. Concerns
3. Types of separation
4. Stakeholders of concerns

1. Principles

- The principle of separation of concerns states that software should be organized so that each program element does one thing and one thing only.
- Each program element should therefore be understandable without reference to other elements.
- Program abstractions (procedures, objects, etc.) support the separation of concerns.
  - Procedural programming languages such as C and Pascal can separate concerns into procedures.
  - Object-oriented programming languages such as Java can separate concerns into objects.
  - Service-oriented architecture can separate concerns into services.

2. Concerns

- A concern is an area of interest or focus in a system.
- Concerns are the primary criteria for decomposing software into smaller, more manageable and comprehensible parts that have meaning to a software engineer.
  - Procedural programming, describing concerns as procedures
  - Object-oriented programming, describing concerns as objects

3. Types of separation

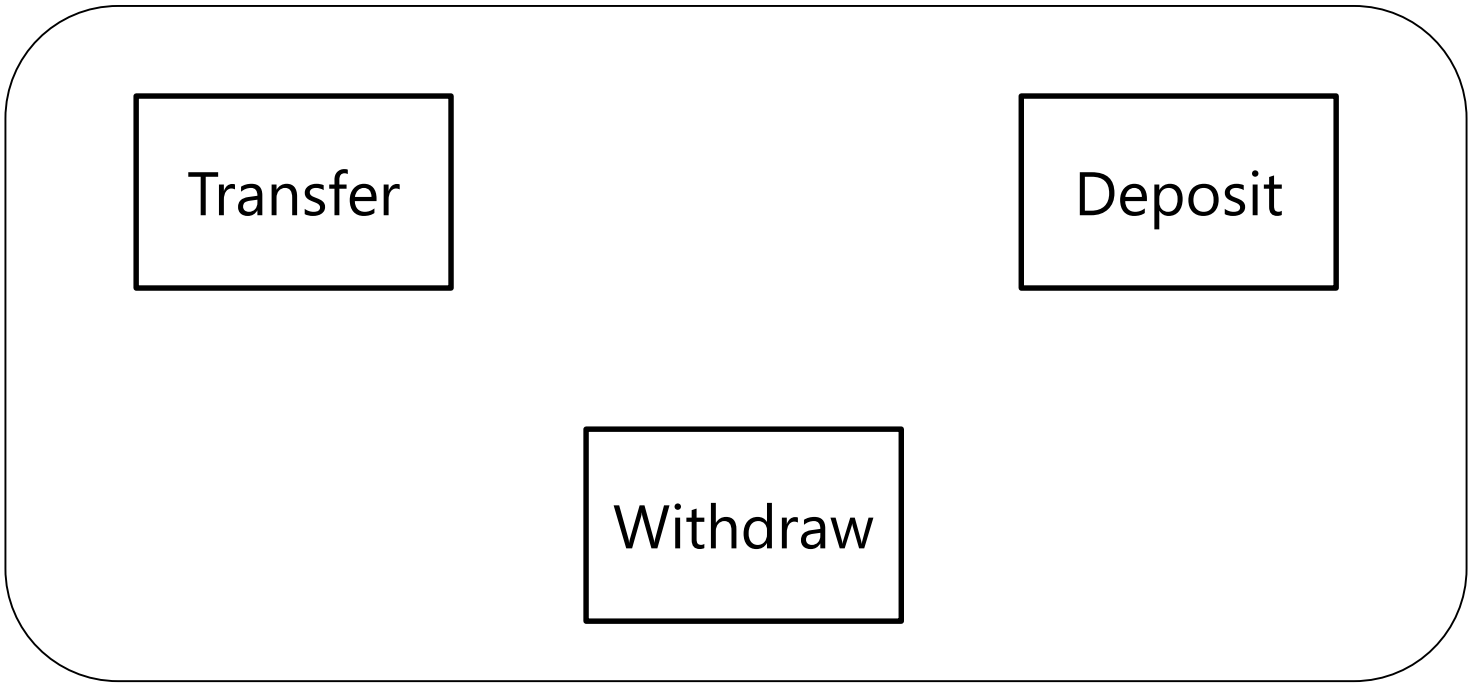
- Quality: deal separately different quality aspects of the system
  - E.g.: security
- Time: plan the activity of a system
  - E.g.: software life cycle
- View: consider & analyze separately the system
  - E.g.: control flow, data flow
- Size: dominate the system complexity
  - E.g.: component

4. Stakeholder concerns

- Functional concerns: related to specific functionalities to be included in a system
- Quality of service concerns: related to the non-functional behaviors of a system
- Policy concerns: related to the overall policies that govern the use of the system
- System concerns: related to attributes of the system as a whole, such as its maintainability or its configurability
- Organizational concerns: related to organizational goals and priorities such as:
  - producing a system within budget
  - making use of existing software assets
  - maintaining the reputation of an organization

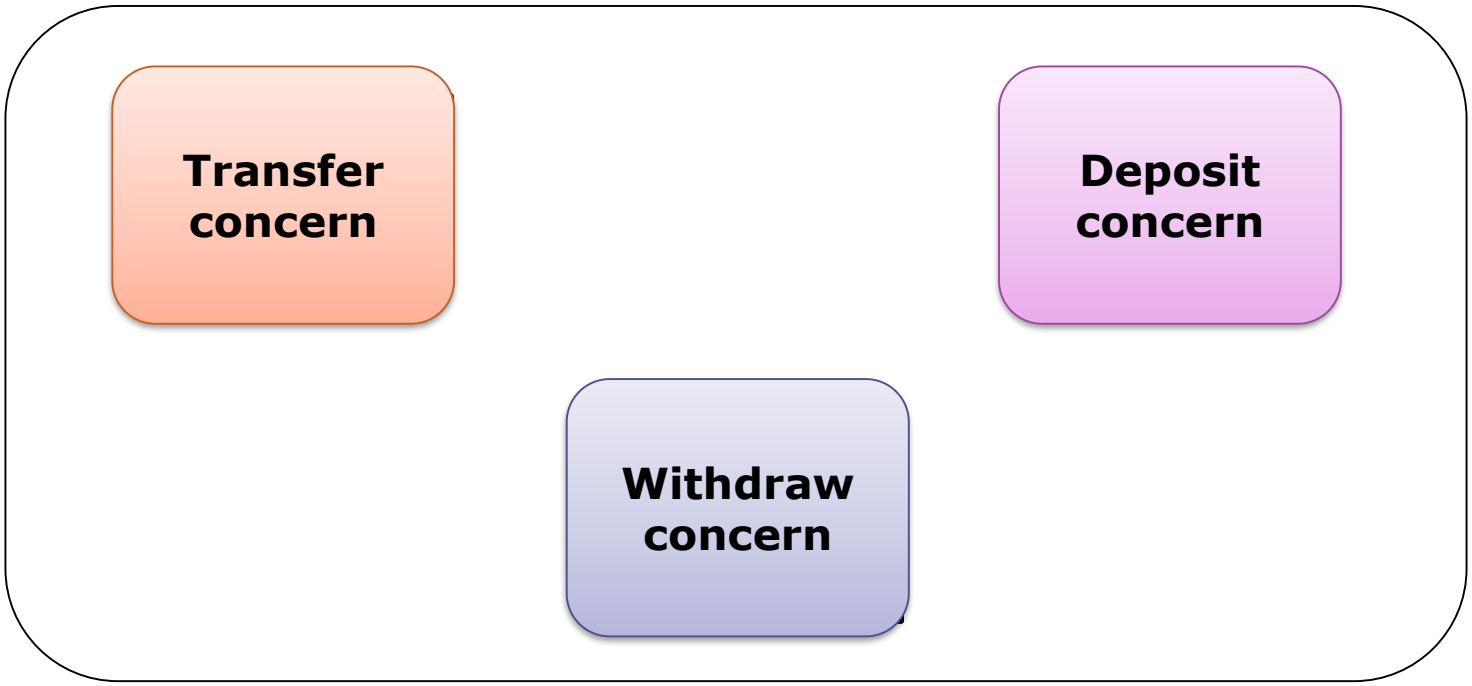
5. Example: Banking system

- Concerns are not program issues but reflect the system requirements and the priorities of the system stakeholders.
- By reflecting the separation of concerns in a program, there is clear mapping from requirements to implementation.



Example:
Banking system's core concerns

- Core concerns: functional concerns relating to the primary purpose of a system



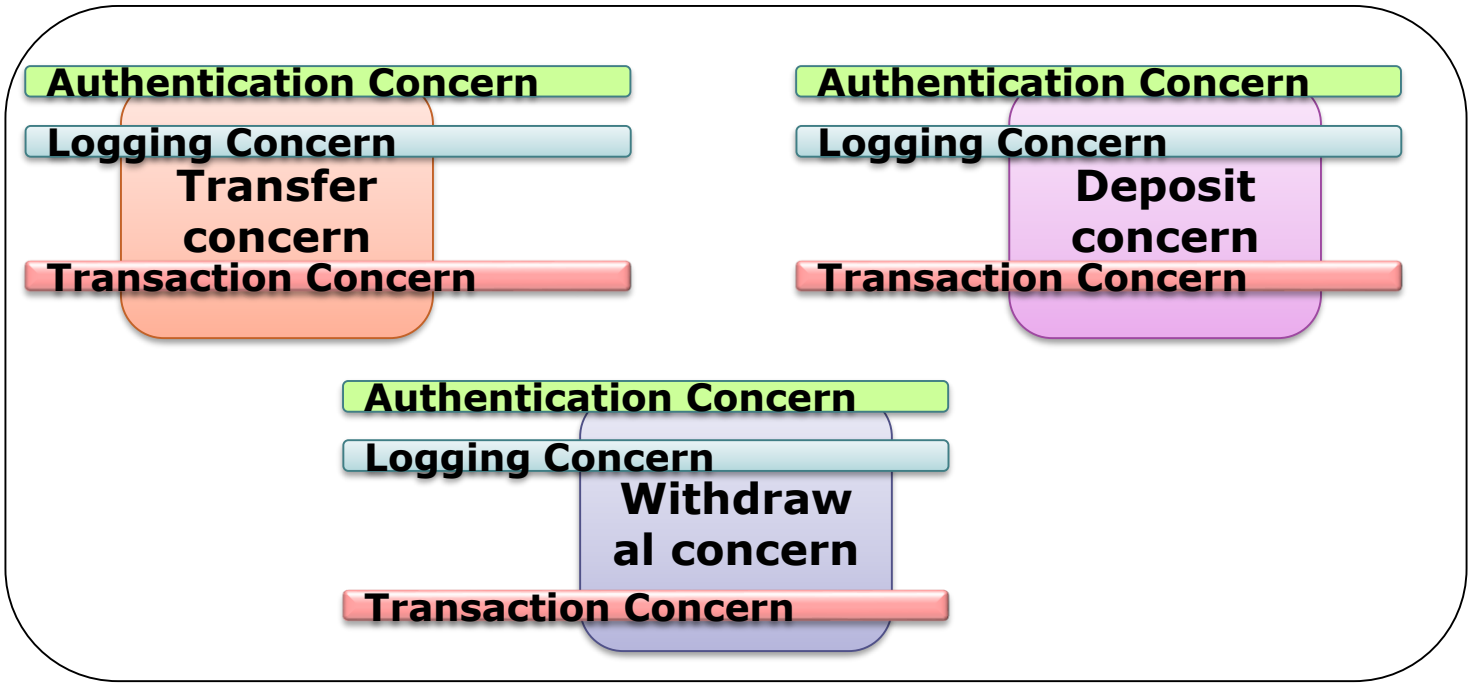
Example: Transfer concern

- Core concern allows bank customers to transfer an amount of money from a given account to another account

```
void transfer(Account fromAccount, Account toAccount, int amount) {
        if (fromAccount.getBalance() < amount) {
                throw new InsufficientFundsException();
        }
        fromAccount.withdraw(amount);
        toAccount.deposit(amount);
}
```

Example:
Banking system's secondary concerns

- Secondary concerns: functional concerns that reflect non-functional and QoS requirements



Example: authentication, transaction and logging concerns

```
void transfer(Account fromAccount, Account toAccount, int amount) throws Exception {
    if (!getCurrentUser().canPerform(OP_TRANSFER)) {
        throw new SecurityException();
    }
```
Authentication Concern

```
    Transaction tx = database.newTransaction();
```
Transaction Concern

```
    try {
        if (fromAccount.getBalance() < amount) {
            throw new InsufficientFundsException();
        }
        fromAccount.withdraw(amount);
        toAccount.deposit(amount);
```
Transfer Concern

```
        tx.commit();
```
Transaction Concern

```
        systemLog.logOperation(OP_TRANSFER, fromAccount, toAccount, amount);
```
Logging Concern

```
    }
    catch(Exception e){
        tx.rollback();
```
Transaction Concern

```
        throw e;
    }
}
```

Quiz and Exercises

- Now let’s go over what you have learned through this lesson by taking a quiz.
- When you’re ready, press Start button to take the quiz