CHAPTER 2: PROGRAM COMPONENTS AND ALGORITHSM

This lecture is based on the CSE OOJ-OJA lectures

Dr. Tran Hai Anh

Outline

1. Elements of a program
2. Execution starting point
3. How to solve problems
4. Algorithms using sequence, selection and repetition
5. Algorithm development using functions

1. Elements of a program

Program components

- Programs in all paradigms share common features
  - Input and output
  - Variables
  - Identifiers
  - Reserved words (keywords)
  - Statements
  - Comments

Input and output

- Input
  - A way of receiving information from the outside
  - Keyboard, files, devices
  - When starting the program or during the program
- Output
  - A way of sending information to the outside
  - Monitor, files, devices

Variables

- Store some data
- Are declared inside methods (or functions)
- The value stored may change as the program progresses
- In Java:
  - The data stored can be a value or a reference to an object

Identifiers

- Identifiers are names for classes, attributes, methods, variables
- In Java:
  - An identifier is made of letters, digits, $ and _
  - It must not begin with a digit
  - It must not be a reserved word (keyword)

Reserved words (keywords)

- Keywords have specific pre-defined meanings
- They cannot be used for other purposes
- In Java:
  - “import” “public” “class” are Java reserved words (keywords)
  - Java has 48 reserved words (listed in the next slide)

Reserved words (keywords)

abstract	boolean	break	byte	case
catch	char	class	const	continue
default	do	double	else	extends
final	finally	float	for	goto
if	implements	import	instanceof	int
interface	long	native	new	package
private	protected	public	return	short
static	strictftp	super	switch	synchronized
this	throw	throws	transient	try
void	volatile	while		

Statements

- A statement specifies an action
- In Java:
  - It must terminate with a semicolon
  - Examples

```
int sum;
int sum = 0;
int sum = n1 + n2;
int n1 = keyboard.nextInt( );
System.out.println("hello");
```

Comments

- Comments are ignored by the computer but written in a program to explain to the reader what the program is doing
- In Java:
  - They come in two forms: block comments and line comments
  - A block comment is enclosed between a /* and a */ and may extend over more than one line
  - A line comment starts from double slashes // and continues to the end of the line

White space

- Blanks, tabs, and new line characters are called white space characters
- Except when white space is used to separate keywords and identifiers, it is ignored by the compiler
- White space can be used to make programs easy to read
- Two main uses of white space
  - (1) indentation
  - (2) blank lines to separate parts of programs

2. Execution starting point

Execution starting point

- Every program needs a starting point at which to start executing

Execution staring point in Java

- Java programs are a collection of classes
- Each class must be stored in a file with the same name, but also with the .java extension
- A class may have the special class method main( ), which contains instructions to start a program
- The starting point of a program must be a main method specified to the interpreter:

  ```
  > java MyClass
  ```

  starts at the main method in the class MyClass

Execution staring point in Java

- Sometimes we create classes simply to give us a place to start in the program
- We call these classes **launcher classes** or **driver classes**

3. How to solve problems

(using algorithms)

Programs and algorithms

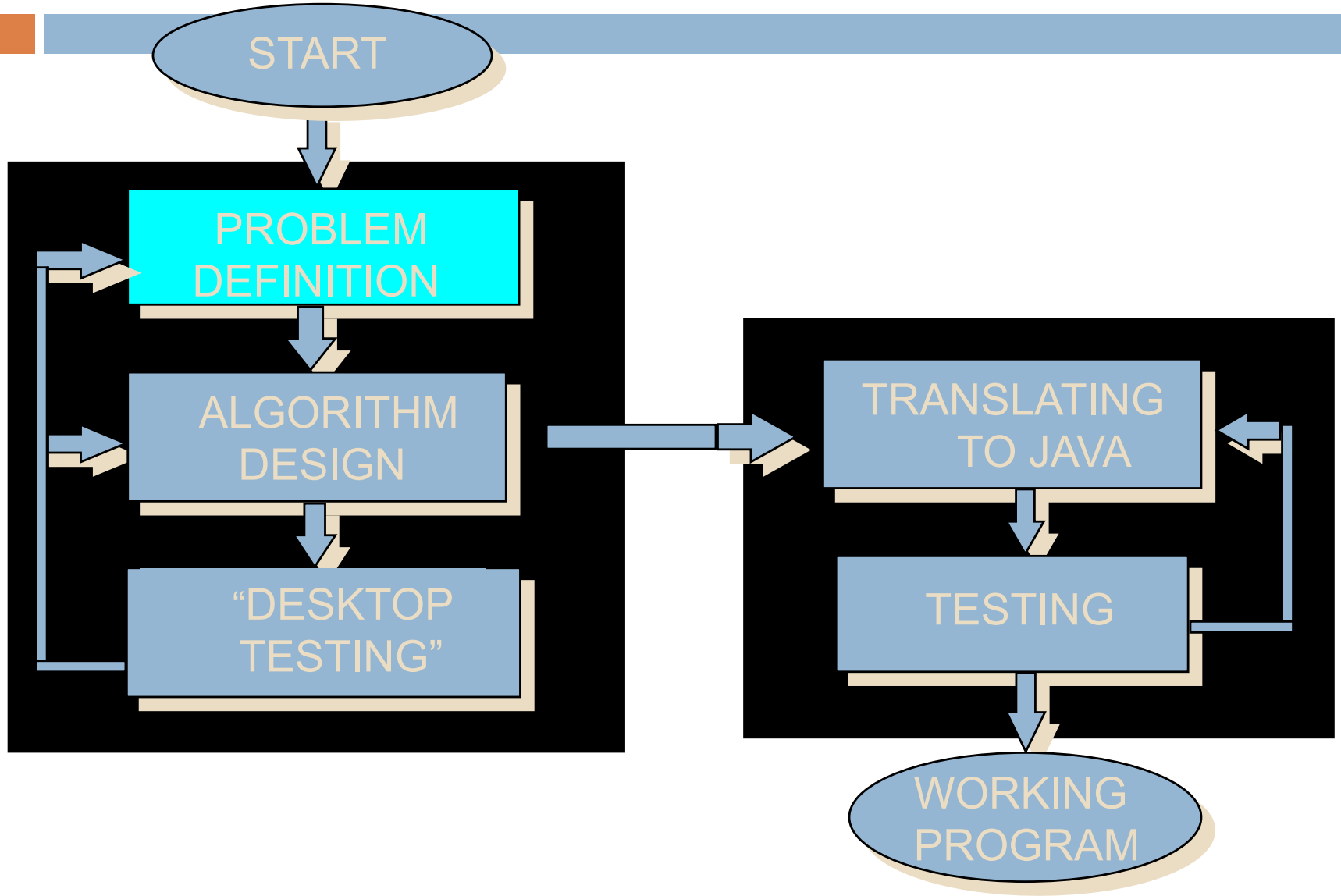
- A program is an algorithm written in some programming language
- An algorithm* is a set of instructions to solve a problem

* *To be more precise, an algorithm is a finite sequence of instructions which, when executed, solves the problem at hand*

Steps involved in solving problems on a computer

- Understand the problem
- Design the solution
- Implement (program) the solution
- Test the solution

In more detail …



Example

- Problem
  - We are planning a restaurant booking for a party. We need to know how many tables to book, given the number of guests attending the party and the number of seats at each table.

Step 1: Understand the problem

- One effective way is to think about the input and output, and perhaps solve the problem for various scenarios

Step 2: Design the algorithm

- Get the number of guests
- Get the number of seats per table
- Determine the number of tables needed
- Output the number of tables

Refine the algorithm

- Get number of guests (numberOfGuests)
- Get number of seats per table (tableSize)
- Calculate numberOfTables to be the least integer greater than or equal to the division

  numberOfGuests / tableSize
- Output numberOfTables

Step 3: Convert to a Java program

- Start with basic program

```
public class Party
{
    public static void main(String[ ] args)
    {
        // Get number of guests
        // Get number of seats per table
        // Calculate number of tables needed
        // Output number of tables
    }
}
```

Be careful, Java is case-sensitive

- Add instructions

```
import java.util.*;
public class Party
{
    public static void main(String[ ] args)
    {
        // Get number of  guests
        Scanner keyboard = new Scanner(System.in);
        System.out.print("Please enter the number of guests: ");
        int numberOfGuests = keyboard.nextInt();

        // Get number of seats per table
        System.out.print("Please enter the number of seats per table: ");
        int tableSize = keyboard.nextInt();

        // Calculate number of tables needed
        int numberOfTables =
          (int) Math.ceil( (double) numberOfGuests /  tableSize );

        // Output number of tables
        System.out.println("Then you will need " + numberOfTables + " tables.");
    }
}
```

Step 4: Create and test the program

- Use vi to edit the program

  > vi Party.java

- Compile the program

  > javac Party.java

- Run the program

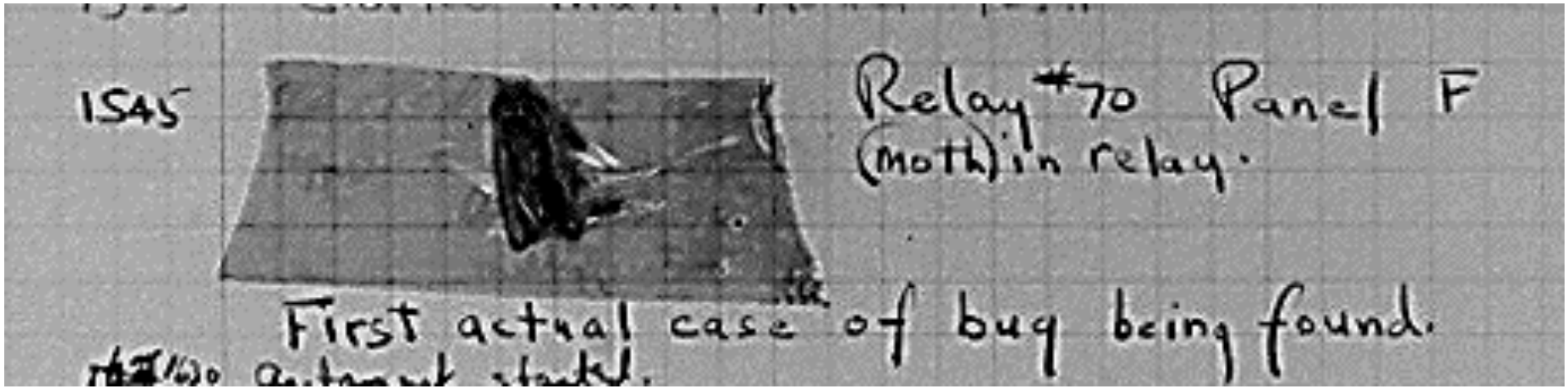
  > java Party

Results of tests

```
Please enter the number of guests: 23
Please enter the number of seats per table: 4
Then you will need 6 tables.

Please enter the number of guests: 12
Please enter the number of seats per table: 6
Then you will need 2 tables.

Please enter the number of guests: 0
Please enter the number of seats per table: 3
Then you will need 0 tables.

Please enter the number of guests: -1
Please enter the number of seats per table: 2
Then you will need 0 tables.
```

Testing and debugging

- Bug: an error in a program
- Debugging: the process of finding and removing bugs



picture from www.history.navy.mil/photos/pers-us/uspers-h/g-hoppr.htm

Testing and debugging

- Compiling: before the compiler converts a program into byte code or object code it must first check that the source code is correct
- Just because a program compiles does not mean it is bug free
- It must be thoroughly tested for less obvious error such as errors in logic

Errors in programs

- Compile-time error
  - Lexical error such as an invalid identifier name
  - Syntax error which is a mistake in the form of the program such as a missing semicolon
  - Semantic error which is a mistake in the meaning of a program such as not declaring a variable before it is used
  - All reported by the compiler

Errors in programs

- Run-time error
  - Occurs during execution and causes the program to stop
- Logical error
  - The program compiles and executes but produces the wrong answer

Types

- The concept of type applies to both simple data and objects
- The type of a simple value can be byte, short, int, long, float, double, boolean, or char
- The type of an object is the class to which it belongs
  - The terms object and instance are normally used interchangeably

4. Algorithms using sequence, selection and repetition

Steps involved in solving problems on a computer

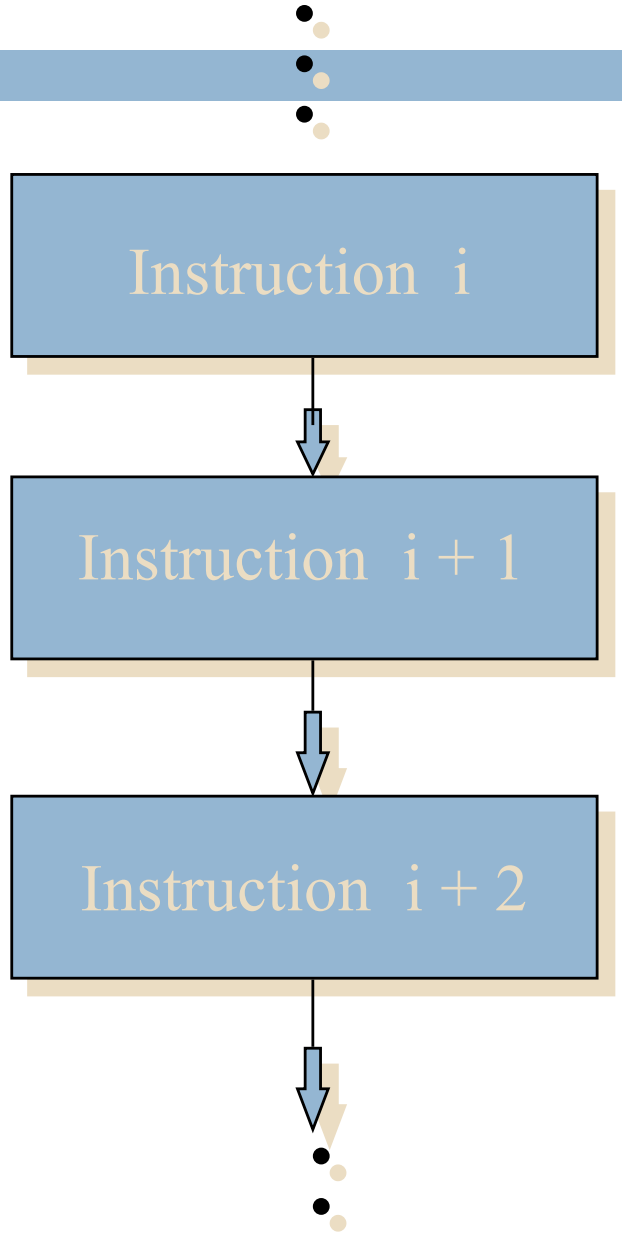
- Understand the problem
- Design a solution
- Implement (program) the solution
- Test the solution

4.1. The three control structures

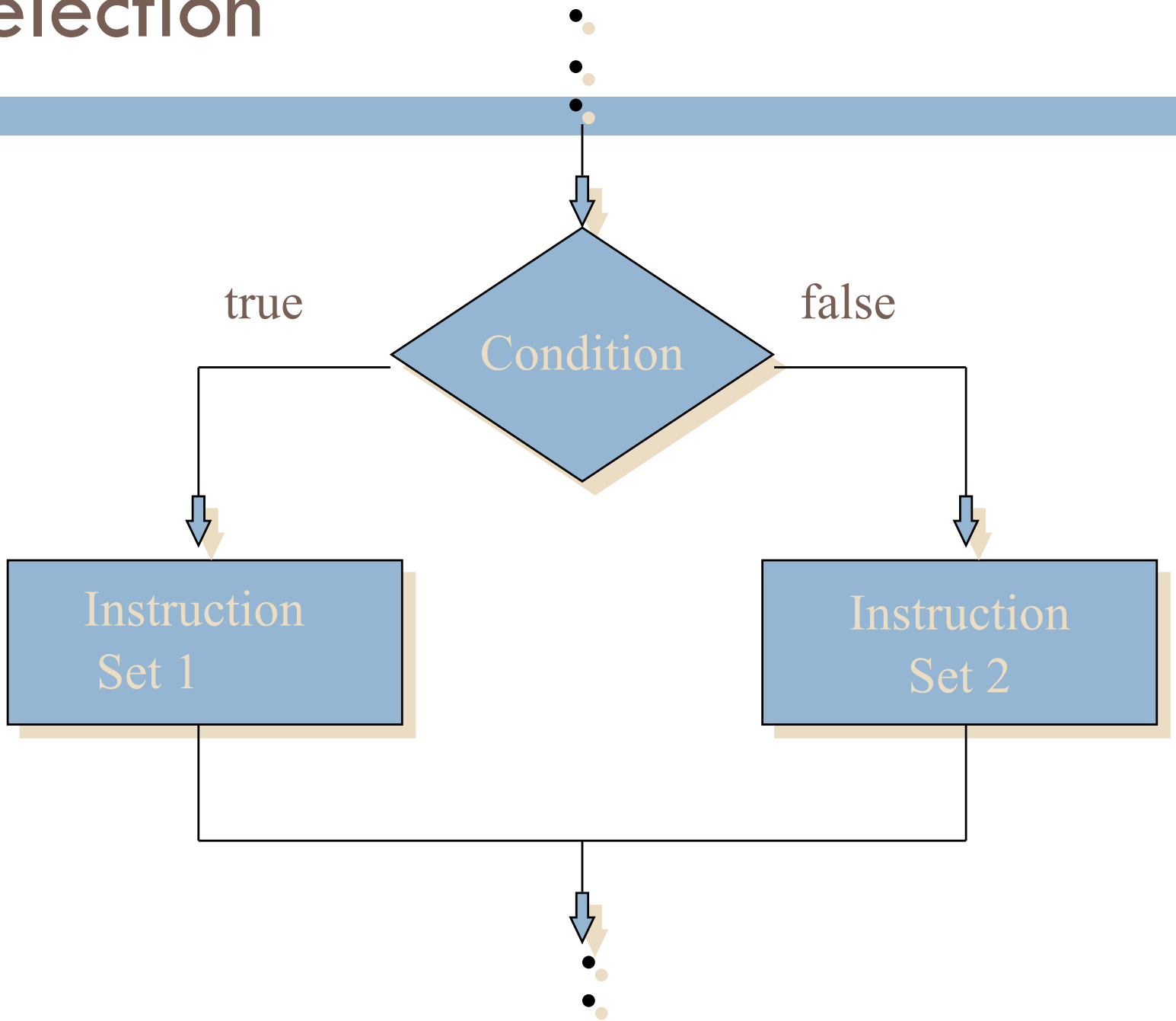
Control structures

- Sequence
  - Instructions executed in the order they are written
- Selection
  - Conditional execution of an instruction (or set of instructions)
- Repetition
  - Repeated execution of a set of instructions

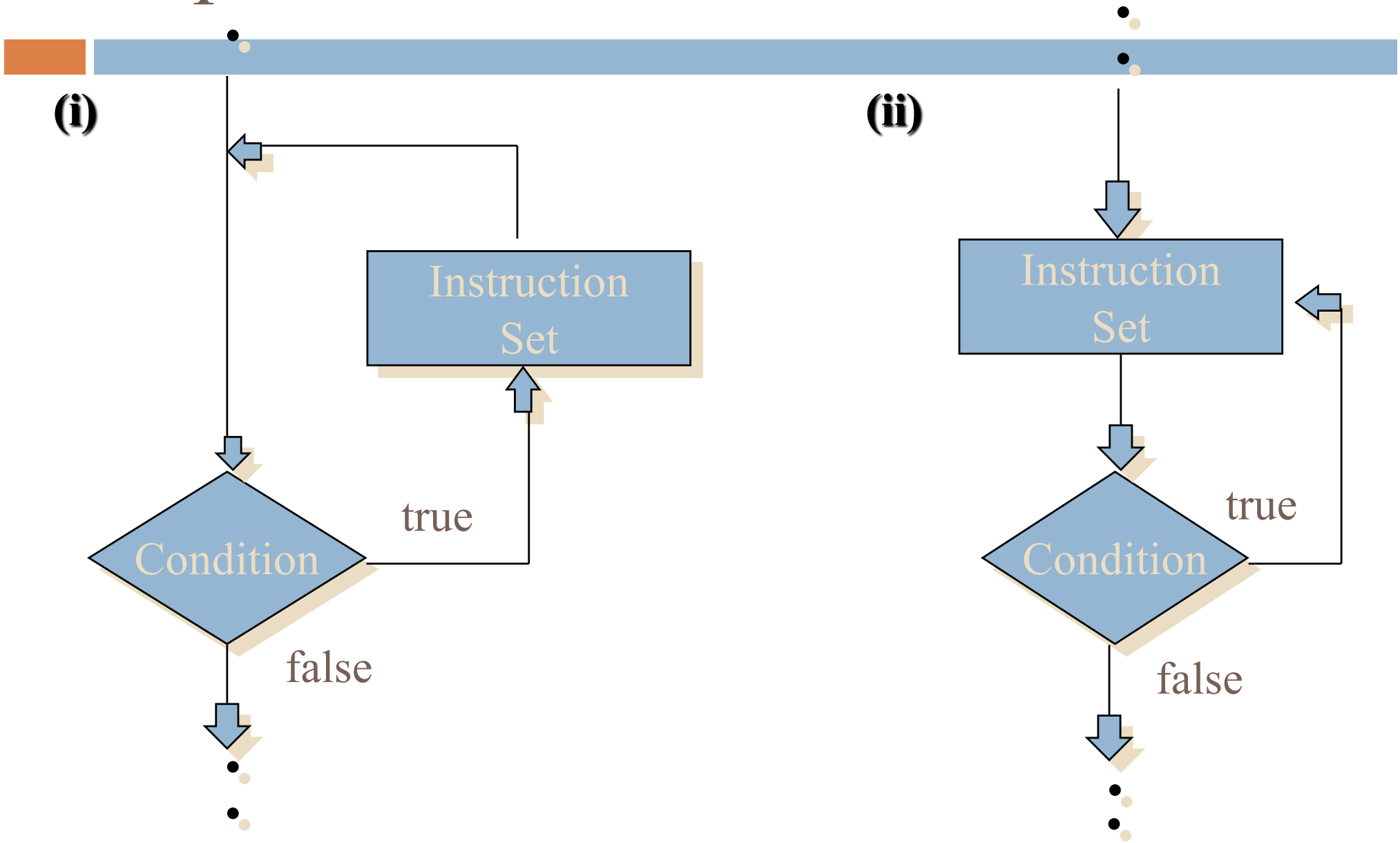
Sequence



Selection



Repetition


(i)
Instruction Set
Condition
true
false
(ii)
Instruction Set
Condition
true
false

4.2. Example 1

(using sequence)

Sequence: average of three numbers

- Problem
  - Display the average of three numbers entered by the user
- Algorithm:

Get the first number

Get the second number

Get the third number

Calculate the average

Display the average

Sequence: average of three numbers

- In Java

```
import java.util.*;
public class Average
{
    public static void main(String[ ] args)
    {
        Scanner keyboard = new Scanner(System.in);
        System.out.println("Enter the three numbers: ");
        int n1 = keyboard.nextInt( );
        int n2 = keyboard.nextInt( );
        int n3 = keyboard.nextInt( );
        double average = (n1+n2+n3) / 3.0;
        System.out.println("The average is " + average);
    }
}
```

4.3. Example 2

(using selection)

Selection: maximum of two numbers

- Problem
  - Display the maximum of two numbers entered by the user
- Algorithm

```
Get the first number n1
Get the second number n2
IF n1 > n2 THEN
   Output n1
ELSE
   Output n2
ENDIF
```

Selection: maximum of two numbers

- In Java

```
import java.util.*;
public class Maximum
{
    public static void main(String[ ] args)
    {
        Scanner keyboard = new Scanner(System.in);
        System.out.println("Input the two numbers: ");
        int n1 = keyboard.nextInt( );
        int n2 = keyboard.nextInt( );
        System.out.print("Maximum = ");
        if (n1 > n2)
        {
            System.out.println(n1);
        }
        else
        {
            System.out.println(n2);
        }
    }
}
```

4.4. Example 3

(using selection)

Determination of SubjectX pass

- Criteria for a pass
  - A student passes SubjectX if the student
    - Averages 50% or more on assignments and labs
    - Receives at least 40% in each exam
    - Gets 50% or more on the combined assignment/lab and exam marks where the assignments/labs contribute 30% and the exams contribute 70%

Determination of SubjectX pass

- Problem
  - Write a program to read in the assignment, lab and exam marks for a student and display “pass” or “fail” for each criterion, as well as the final mark
  - There will be 4 assignment marks, 2 lab marks and 2 exam marks

Determination of SubjectX pass

- Top level refinement
  - Express the problem in terms of major tasks and then solve each sub-task
- Solution

Are assignments and labs OK?

Are exams OK?

Is total mark OK?

Determination of SubjectX pass

- Refine sub-tasks
- Step 1: *Are assignments and labs OK?*
- Solution:

Determination of SubjectX pass

- Further refinement of step 1

```
Get assignment mark 1
Get assignment mark 2
Get assignment mark 3
Get assignment mark 4
Get lab mark 1
Get lab mark 2
average = (assign1 +assign2 +assign3 +assign4 +lab1 +lab2) /6
IF average >= 50 THEN
    Display “Passed assignment/lab hurdle!"
ELSE
    Display “Failed assignment/lab hurdle!"
ENDIF
```

Determination of SubjectX pass

- Refine subtasks
- Step 2: *Are exams OK?*
- Solution:

Determination of SubjectX pass

- Refine subtasks
- Step 3: *Is total mark OK?*
- Solution:

```
imprt java.util,*;
public class SubjectXPass
{
    public static void main(String[ ] args)
    {
        Scanner keyboard = new Scanner(System.in);
        boolean passedHurdle = true;
        System.out.println("Please enter 4 assignment marks and 2 lab marks: ");
        int assign1 = keyboard.nextInt( );
        int assign2 = keyboard.nextInt( );
        int assign3 = keyboard.nextInt( );
        int assign4 = keyboard.nextInt( );
        int lab1 = keyboard.nextInt( );
        int lab2 = keyboard.nextInt( );
        double pracAverage = (assign1 + assign2 + assign3 + assign4 + lab1 + lab2)
                                / 6.0;
        if (pracAverage >= 50)
        {
            System.out.println("Passed assignment/lab hurdle!");
        }
        else
        {
            passedHurdle = false;
            System.out.println("Failed assignment/lab hurdle!");
        }
```

Determination of SubjectX pass

```
System.out.println("Please enter 2 exam marks: ");

int exam1 = keyboard.nextInt( );
int exam2 = keyboard.nextInt( );

if ((exam1 >= 40) && (exam2 >= 40))
{
    System.out.println("Passed exam hurdle!");
}
else
{
    passedHurdle = false;
    System.out.println("Failed exam hurdle!");
}

double examAverage = (exam1 + exam2) / 2.0;
```

Determination of SubjectX pass

```
        double finalMark = 0.3 * pracAverage + 0.7 * examAverage;
        System.out.println("Final mark is " + finalMark + "%");

        if ((finalMark >= 50) && (passedHurdle == true))
        {
            System.out.println("Passed overall.");
        }
        else
        {
            System.out.println("Failed overall.");
        }
    }
}
```

4.5. Example 4

(using repetition)

SubjectX results

- Problem
  - Check the hurdle requirements and determine the final result for all students in the class
- Solution

SubjectX results

- Pseudocode solution

```
FUNCTION processStudentResult
    Get assignment/lab marks
    Check hurdle requirements
    Get exam marks
    Check hurdle requirements
    Compute final result
    Display final mark and pass or fail
ENDFUNCTION
```

SubjectX results

- To handle many students' results

```
WHILE (more students)
        processStudentResult
ENDWHILE
```

SubjectX results

- How do we know if there are any more students?

SubjectX results

- Pre-set number

```
int numberOfStudents = keyboard.nextInt( );
while (numberOfStudents > 0)
{
    // processStudentResult
    numberOfStudents = numberOfStudents - 1;
}
```

SubjectX results

- ‘Sentinel’ value
- Alter processing of a student’s result

```
int assign1 = keyboard.nextInt( );
while (assign1 >= 0)
{
    // processStudentResult
    assign1 = keyboard.nextInt( );
}
```

4.6. Example 5

(exercise)

Class exercise: control structures

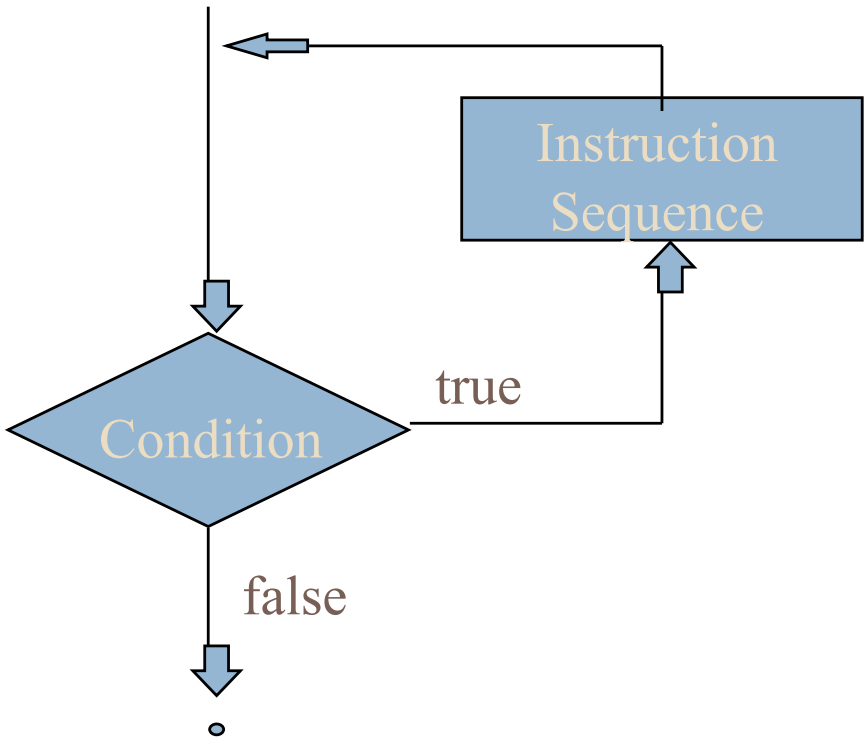
- Write pseudocode to solve the following problem
  - There is a (non-empty) line of people. Go to each person in the line and ask them their age. If they are older than 25, ask them to step forward.

A possible solution

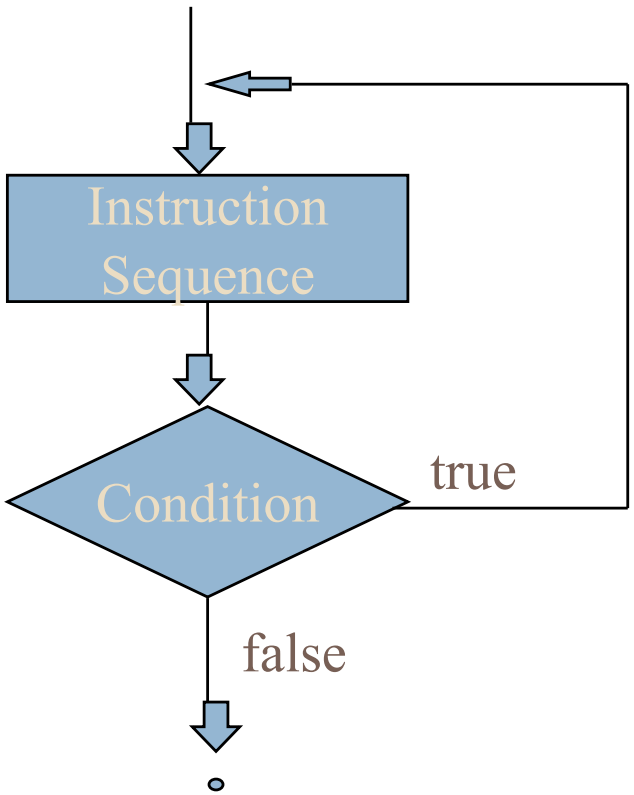
4.7. WHILE loops vs DO…WHILE loops

WHILE…ENDWHILE versus DO...WHILE loops

```
WHILE condition
    <instruction sequence>
ENDWHILE
```




```
DO
   <instruction sequence>
WHILE condition
```



DO...WHILE

- Problem
  - Write pseudocode to simulate crossing the road
- Basic actions
  - look left
  - look right
  - walk across
- Condition
  - road is busy

DO...WHILE

- Solution:

Class exercise

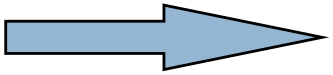
- Problem
  - Rewrite the solution to the “crossing the road” problem using the WHILE…ENDWHILE construct

Solution

WHILE…ENDWHILE versus DO...WHILE loops

- Using a WHILE…ENDWHILE loop to implement a DO…WHILE loop

```
DO
   <body>
WHILE condition
```

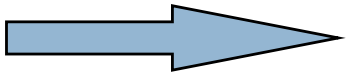


```
<body>
WHILE condition
    <body>
ENDWHILE
```

WHILE…ENDWHILE versus DO...WHILE loops

- Using a DO...WHILE loop and a selection control structure to implement a WHILE…ENDWHILE loop

```
WHILE condition
  <body>
ENDWHILE
```



```
IF condition THEN
  DO
    <body>
  WHILE condition
ENDIF
```

5. Algorithm development using functions

Algorithm development

- We cover algorithm development in 3 parts
  - Problem-solving procedure
  - Three control structures
  - Using functions (methods, operations)

What is a function?

- Algorithm for crossing the road

DO

Look left

Look right

Look left

WHILE road is busy

Walk across

Function Look left

```
FUNCTION Look left
    Turn head left
    Observe
    IF cars are coming THEN
        road is busy
    ENDIF
ENDFUNCTION
```

What is a function?

- A segment of code extracted separately
- Given a name
- May be used (called) more than once
- May be used from different parts of a program

Functions in Java

- In Java all pieces of code must be inside a class
- To write a function therefore it must be defined inside a class
- It is therefore written as a method
- We have to be careful to select the appropriate class in which to place these methods
- Methods are often called “member functions” because they are functions that belong to a class

n^n table

- Problem
  - Print out a table comprising the integers n and n^n, for n = 1, 2, ..., 10

1	1
2	4
3	27
4	256
.	.
.	.
.	.

Solution

Problem

- Problem: How to calculate n^n? (i.e. the power function or pow)
- Solutions:
  - Use a library function
  - Or, write your own function

Using a pre-defined library function

- Java method Math.pow( ) takes two real numbers as arguments and returns a real number

```
public class PowerTable
{
    // Computes the values of n^n for n = 1,
    // 2, ..., 10 using the predefined function pow
    public static void main(String[ ] args)
    {
        int n = 0;
        do
        {
            n = n+1;
            System.out.println(n + " " + Math.pow(n,n));
        }
        while (n < 10);
    }
}
```

Using a user-defined function

```
// Returns x to the power of y
// Pre-condition: y >= 0
public static double myPow(double  x, int  y)
{
    double z;
    int counter;
    if (y == 0)
    {
        z = 1;
    }
    else
    {
        counter = 1;
        z = x;
        while (counter < y)
        {
            z = z * x;
            counter = counter + 1;
        }
    }
    return z;
}
```

Class exercise: max( ) function

- Problem
  - Write pseudocode for a function max( ) that takes two integers as arguments and returns the bigger of the two

Solution

Program using max( ) function

- Problem
  - Write an algorithm that uses function max( ) to compute the maximum of four numbers

Solution

Why use functions?

- Functions greatly assist algorithm development
  - By breaking tasks into smaller sub-tasks
- Functions re-use code
  - Library code (APIs in Java)
  - User code
  - Independent of algorithm in which they are used
  - Reduces redundant code
- Functions enhance maintainability

Why use functions?

- Functions make algorithms easier to read and understand
- Functions can be implemented by separate teams
- Libraries of useful functions can be developed