# Module 3
# Problem Solving and Algorithms

# Introduction

*In this module* we will learn how to break problems down into step-by-step solutions called algorithms.

Then, we will look algorithms from a human readable representation called pseudocode. Then we will translate this into a visually or diagramming perspective by using a <u>system called flowcharts</u>.

# Definitions

*An algorithm* is a set of instructions for solving a problem. *For example*, navigating from your home to the RRC campus would be an algorithm as follow:

1. Leave through the front door.
2. Walk down to the end of the driveway.
3. Turn right.
4. Walk until you reach a east-bound bus stop.
5. Wait for the bus.
6. When bus arrives board and pay.
7. *Etc.*

***Creating algorithms - requirements***

## Requirements of a Computer Algorithm

1. The algorithm is well-ordered (order executes matters).
2. The algorithm must be no ambiguous (simplified to highest).
3. All steps are computable (avoid things like division by zero).
4. The algorithm must produce a result (solves a problem).
5. The algorithm will always finish in a finite amount of time.

## *Creating algorithms - efficiency*

*Analyzing algorithms' efficiency:*

*For example, a common algorithm process* performed by computers is sorting. *Since computers can compare a large number of items quickly, they can sort large collections at incredible speeds.*

*The Efficiency of Algorithms here relates with* two important considerations:

- *Time Efficiency:* How long it will take to solve the problem?
- *Space Efficiency:* How much memory will it use while solving the problem?

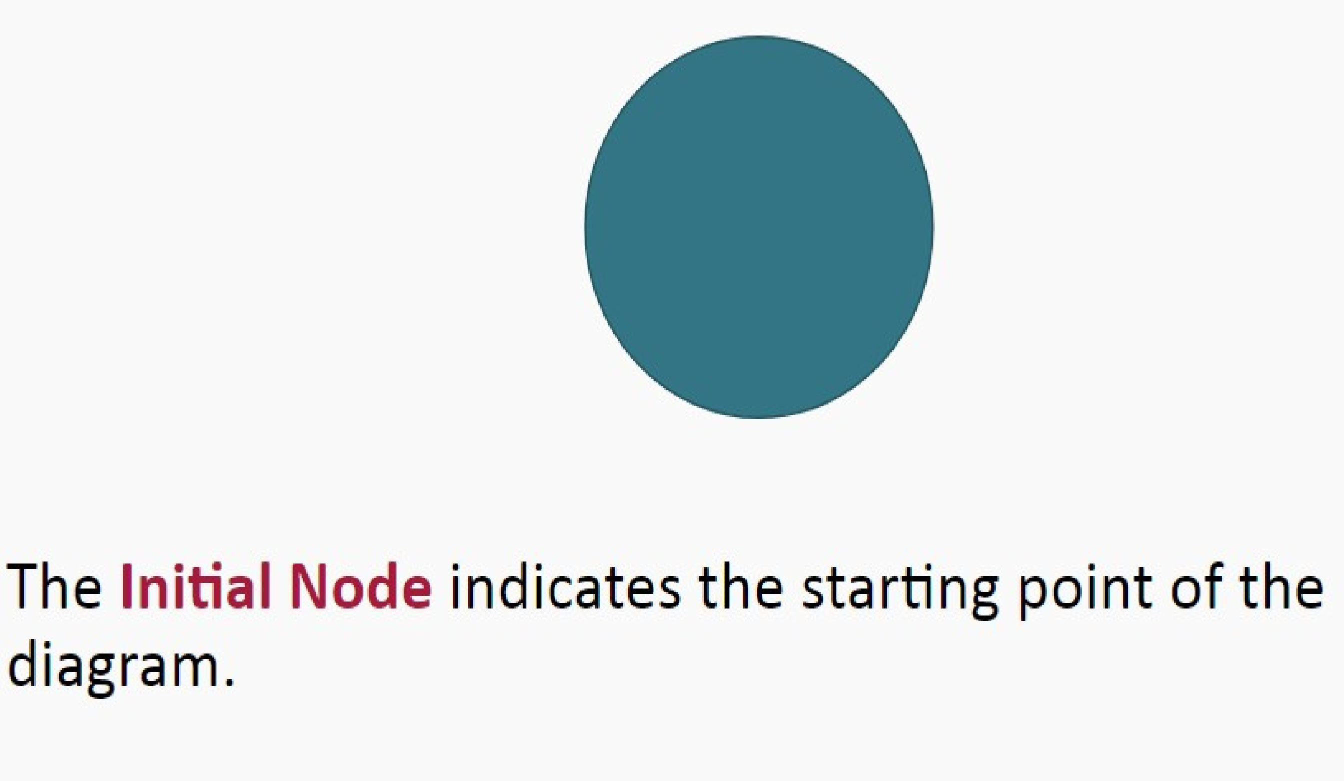
*Creating algorithms - pseudocode*

## Steps to create an algorithm:

1. **Understand the Problem** (identify the inputs, outputs, and constraints).
2. **Identify the Approach** (strategy use to solve the problem; searching, sorting, etc)
3. **Break It Down** (divide the problem into smaller sub-problems or steps)
4. **Define Input and Output** (data types, ranges, and formats)
5. **Design the Logic** (detailing the sequence of steps to be executed, core of algorithm)
6. **Use pseudocode and flowchart** (human readable and graphic representations; no programming language syntax)
7. **Consider Efficiency** (analyze the algorithm's efficiency in terms of time and space complexity).

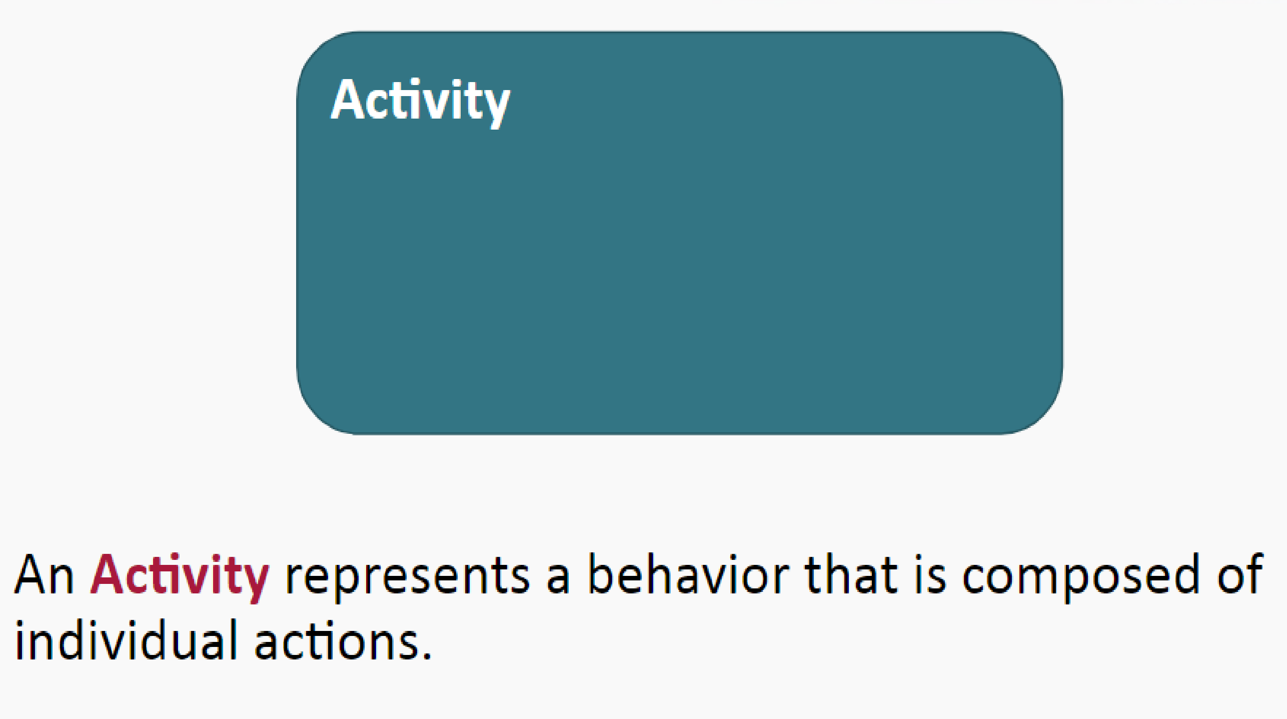
# Flowcharts

Flowcharts illustrate instructions or actions in a sequence way to solve problems by using symbols. Each *symbol* has a unique meaning in Unified Modeling Language (UML), as follow:



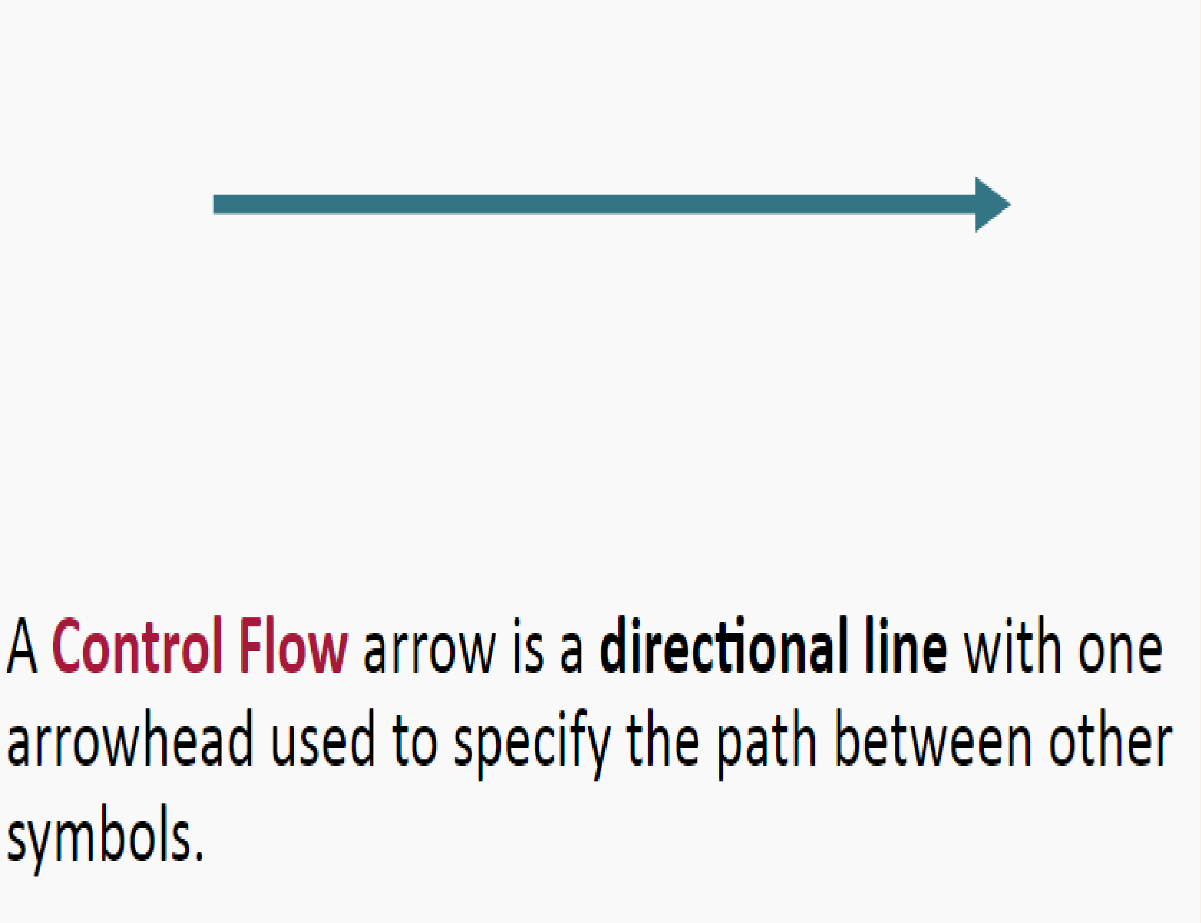
The **Initial Node** indicates the starting point of the diagram.

# Flowcharts



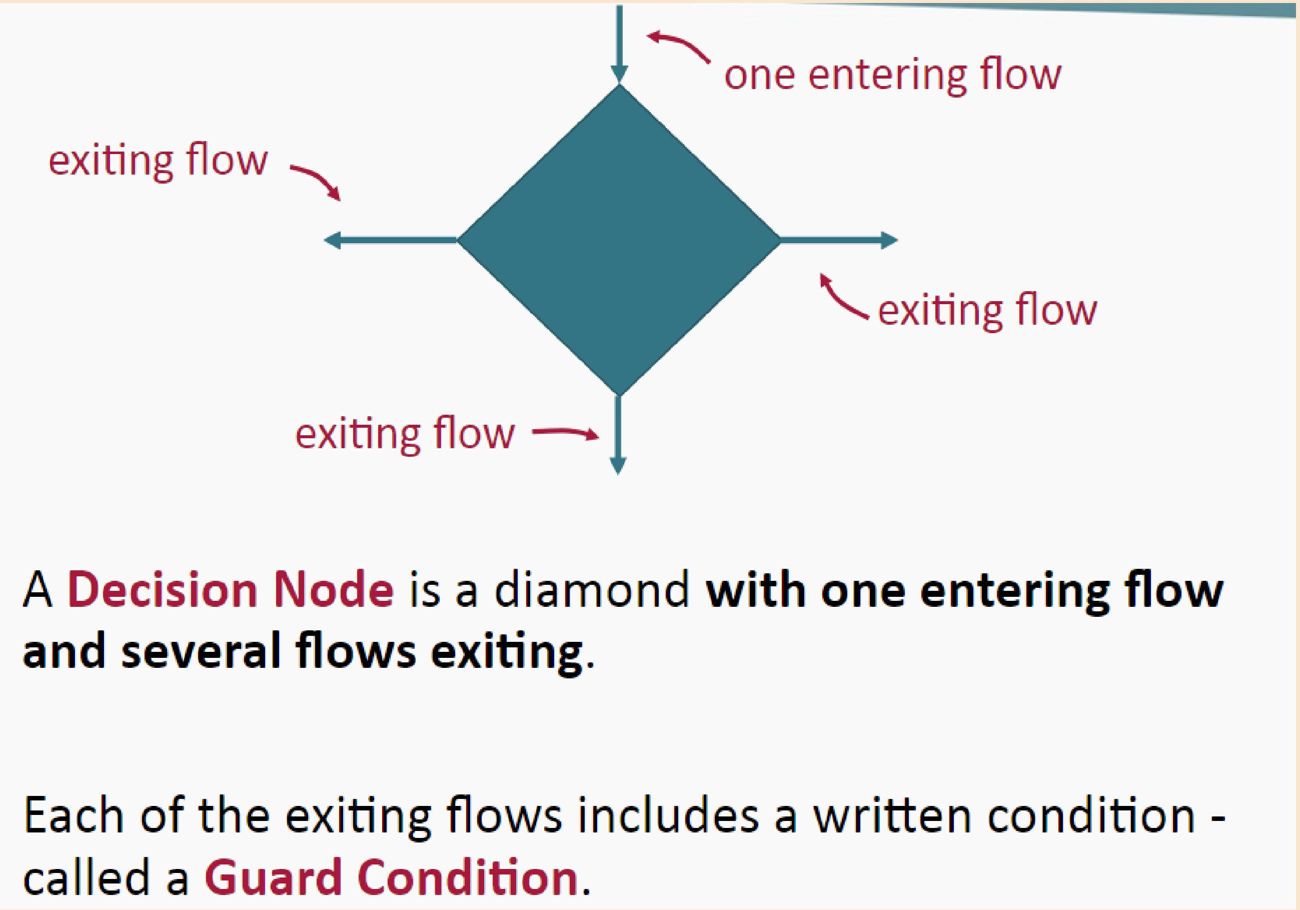
An **Activity** represents a behavior that is composed of individual actions.

# Flowcharts



A **Control Flow** arrow is a **directional line** with one arrowhead used to specify the path between other symbols.

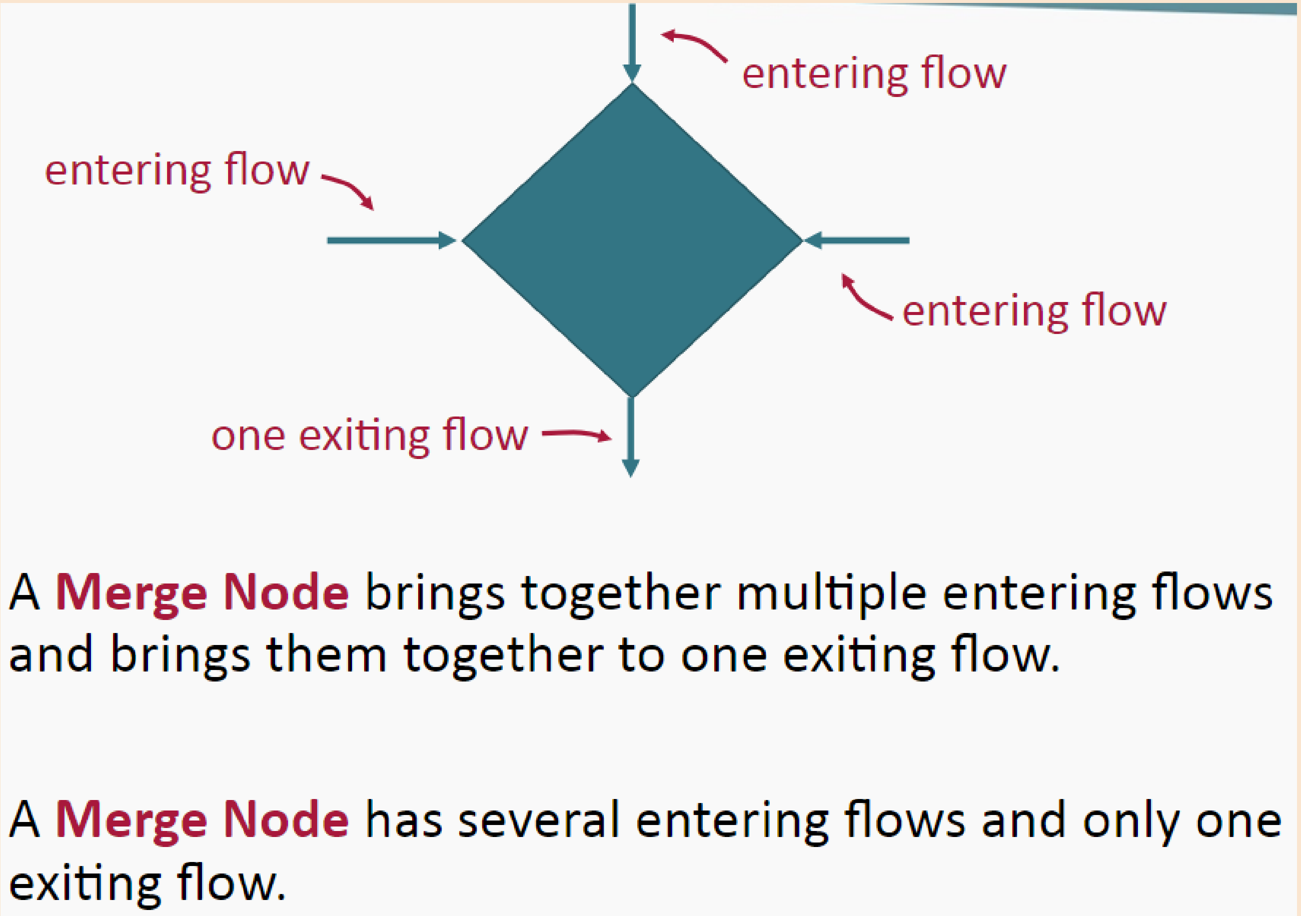
# Flowcharts



A **Decision Node** is a diamond **with one entering flow and several flows exiting**.

Each of the exiting flows includes a written condition - called a **Guard Condition**.

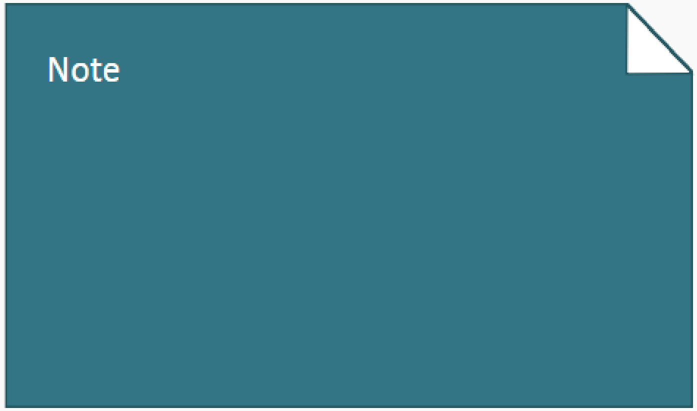
# Flowcharts



A **Merge Node** brings together multiple entering flows and brings them together to one exiting flow.

A **Merge Node** has several entering flows and only one exiting flow.

# *Flowcharts*

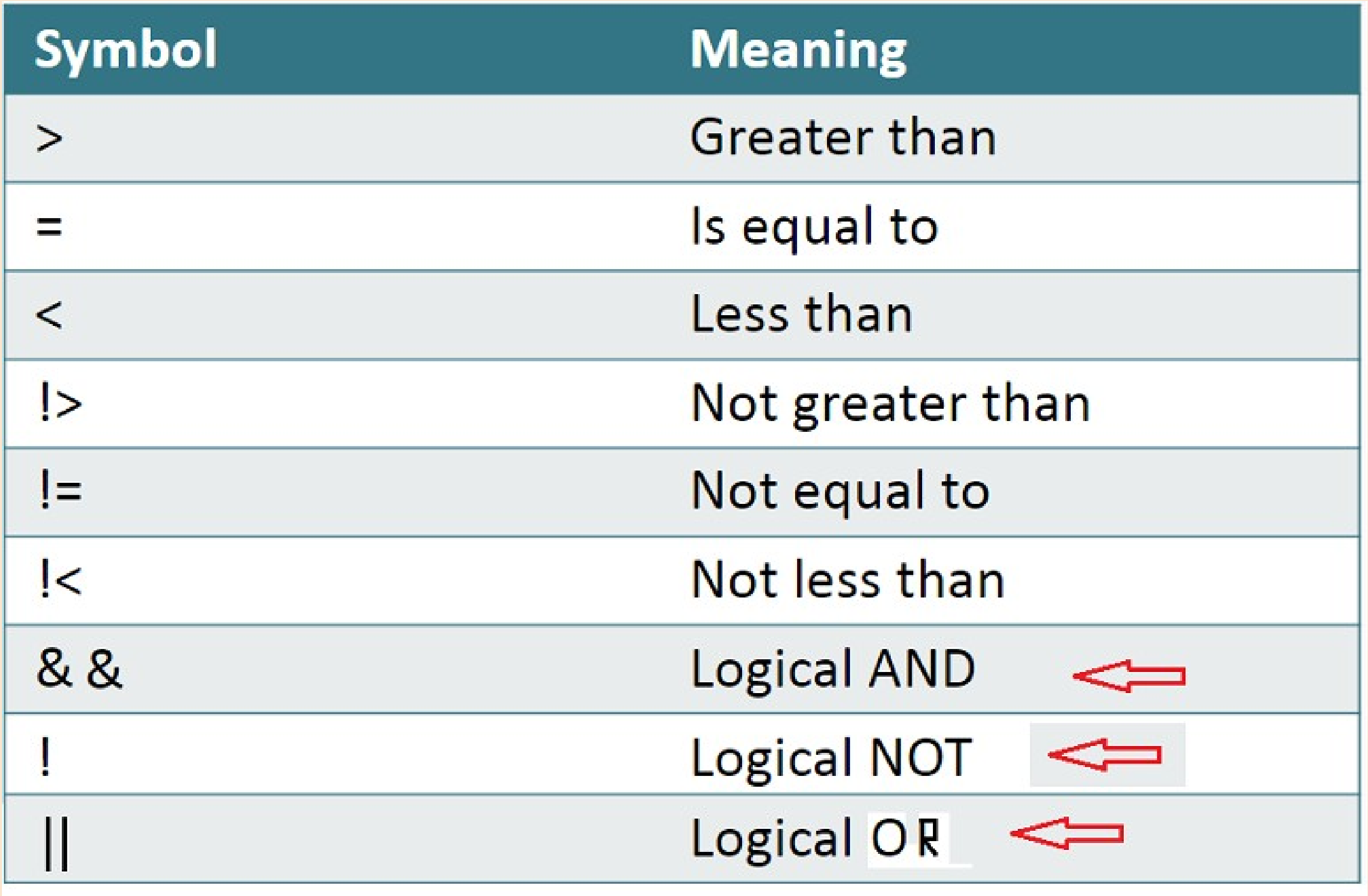


A **Note** (comment) gives the ability to attach various remarks to elements.

A comment does not represent code, but may contain information that is useful to a modeler.

## Flowcharts

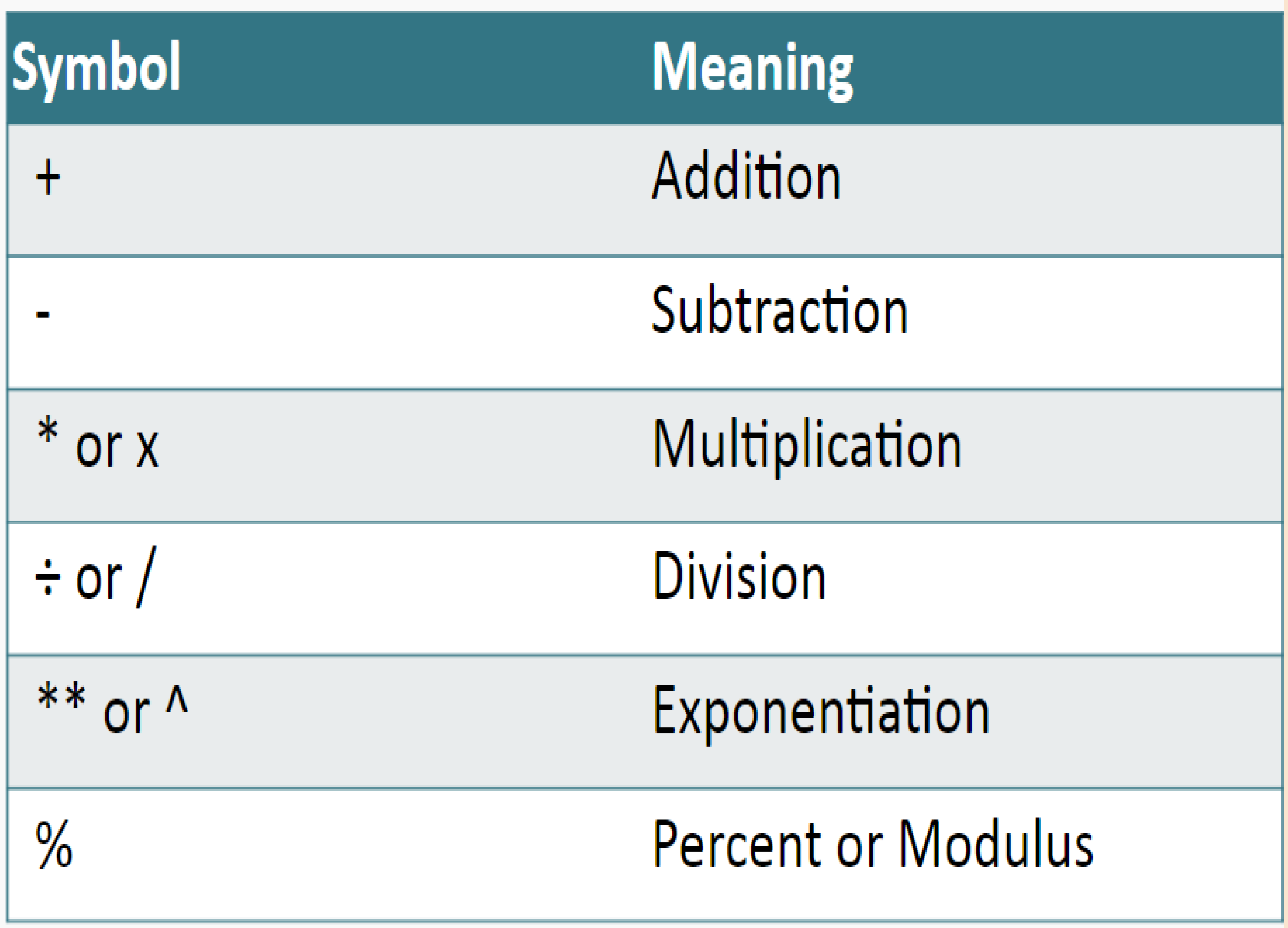
To include Boolean expressions in your flowcharts, you will use the following symbols. Each *symbol* has a unique meaning in Unified Modeling Language (UML), as follow:



| Symbol | Meaning |
|---|---|
| > | Greater than |
| = | Is equal to |
| < | Less than |
| !> | Not greater than |
| != | Not equal to |
| !< | Not less than |
| & & | Logical AND |
| ! | Logical NOT |
| \|\| | Logical OR |

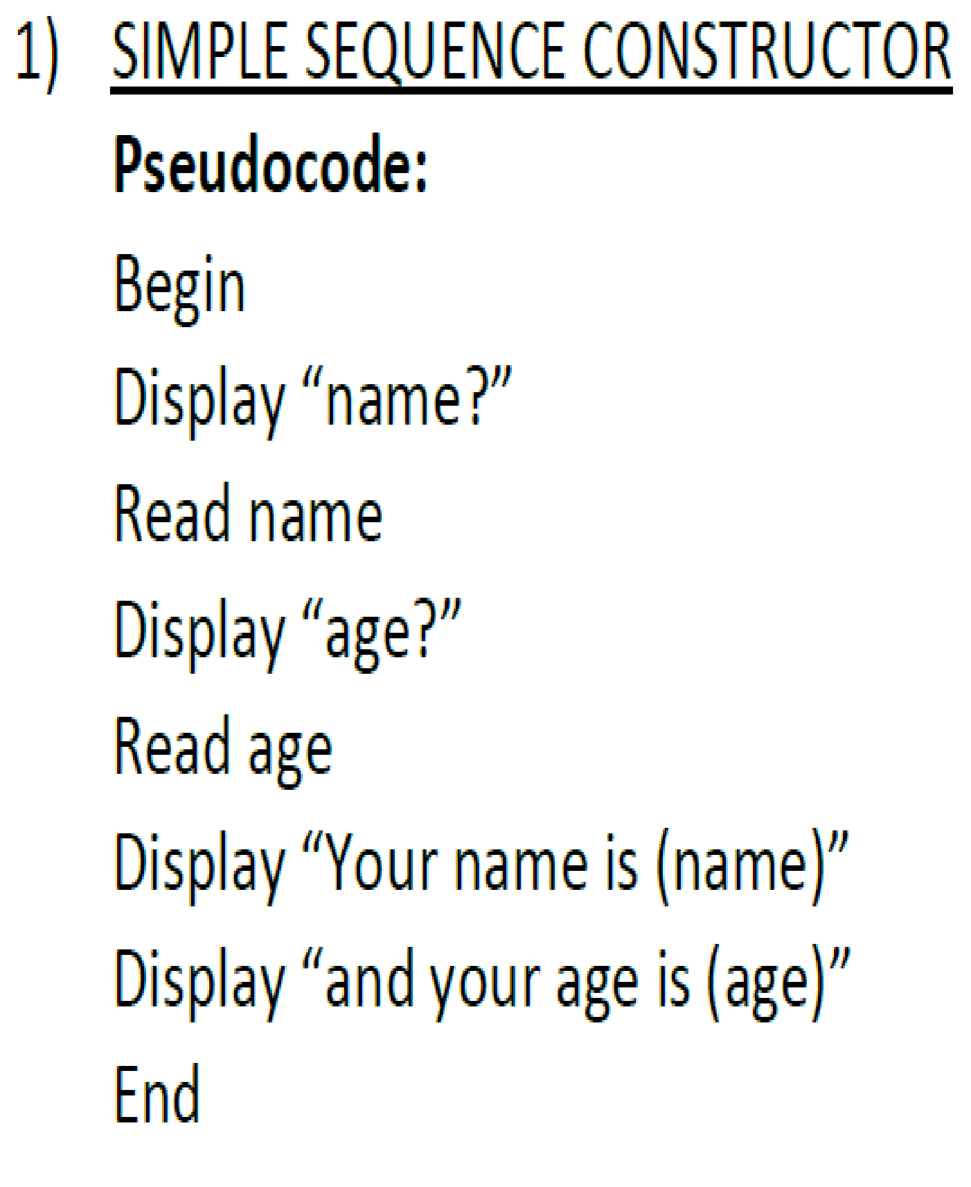
## Flowcharts

To include mathematical expressions in your flowcharts, you will use the following symbols:



| Symbol | Meaning |
|---|---|
| + | Addition |
| - | Subtraction |
| * or x | Multiplication |
| ÷ or / | Division |
| ** or ^ | Exponentiation |
| % | Percent or Modulus |

## *Sequential Structure - pseudocode*

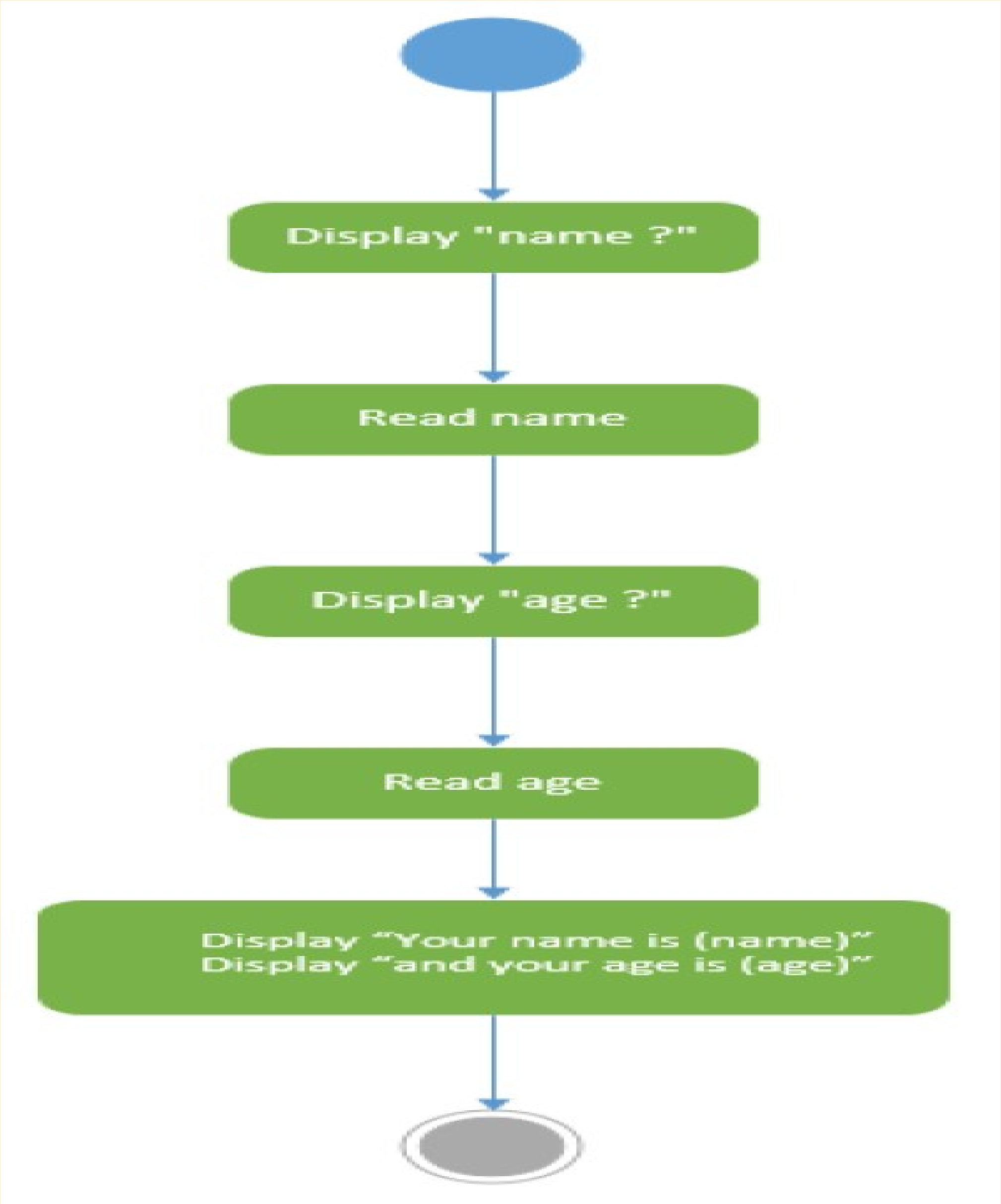


1) SIMPLE SEQUENCE CONSTRUCTOR

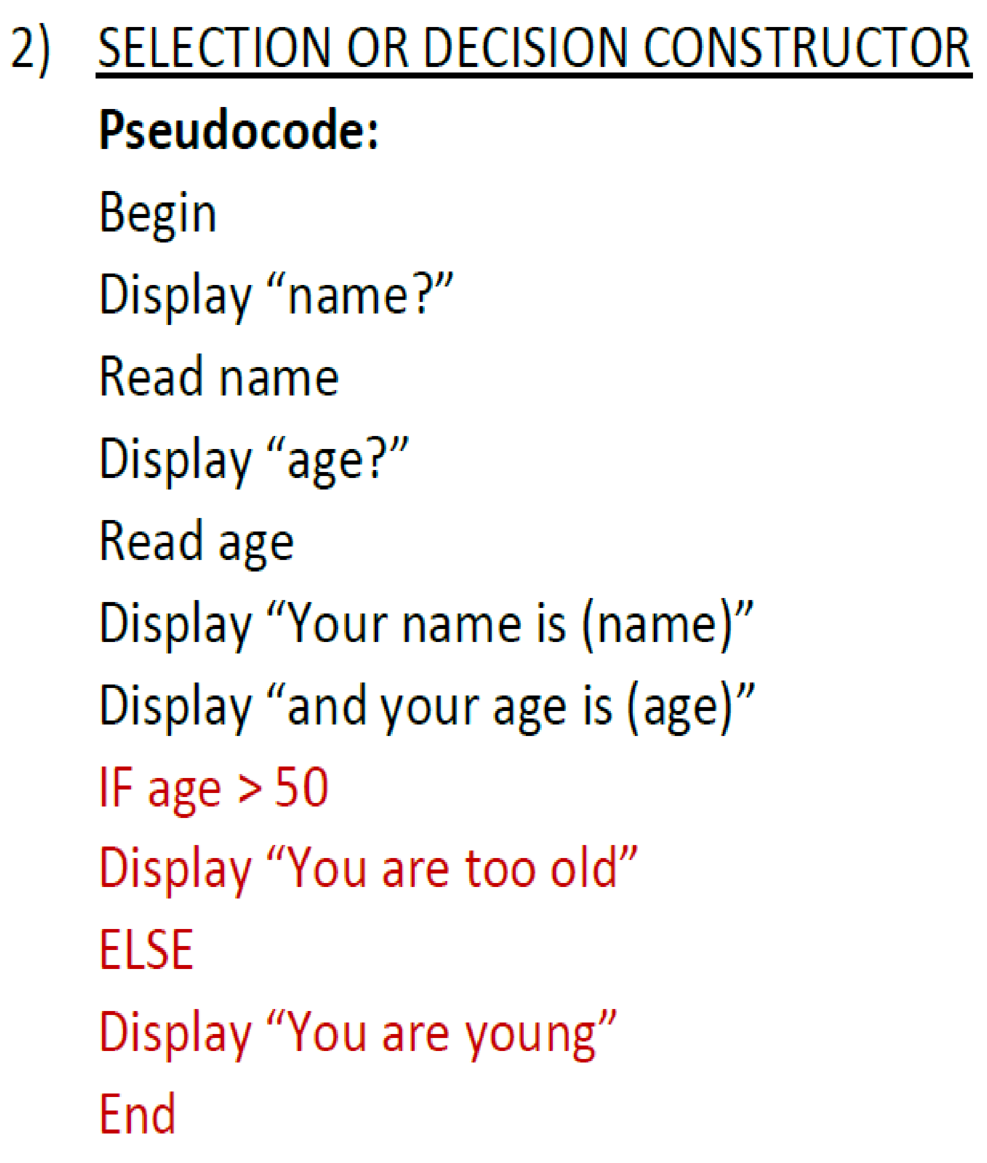
**Pseudocode:**

```
Begin
Display "name?"
Read name
Display "age?"
Read age
Display "Your name is (name)"
Display "and your age is (age)"
End
```

# *Sequential Structure – flowchart*


Display "name ?"
Read name
Display "age ?"
Read age
Display "Your name is (name)"
Display "and your age is (age)"

# *Decision Structure – pseudocode*

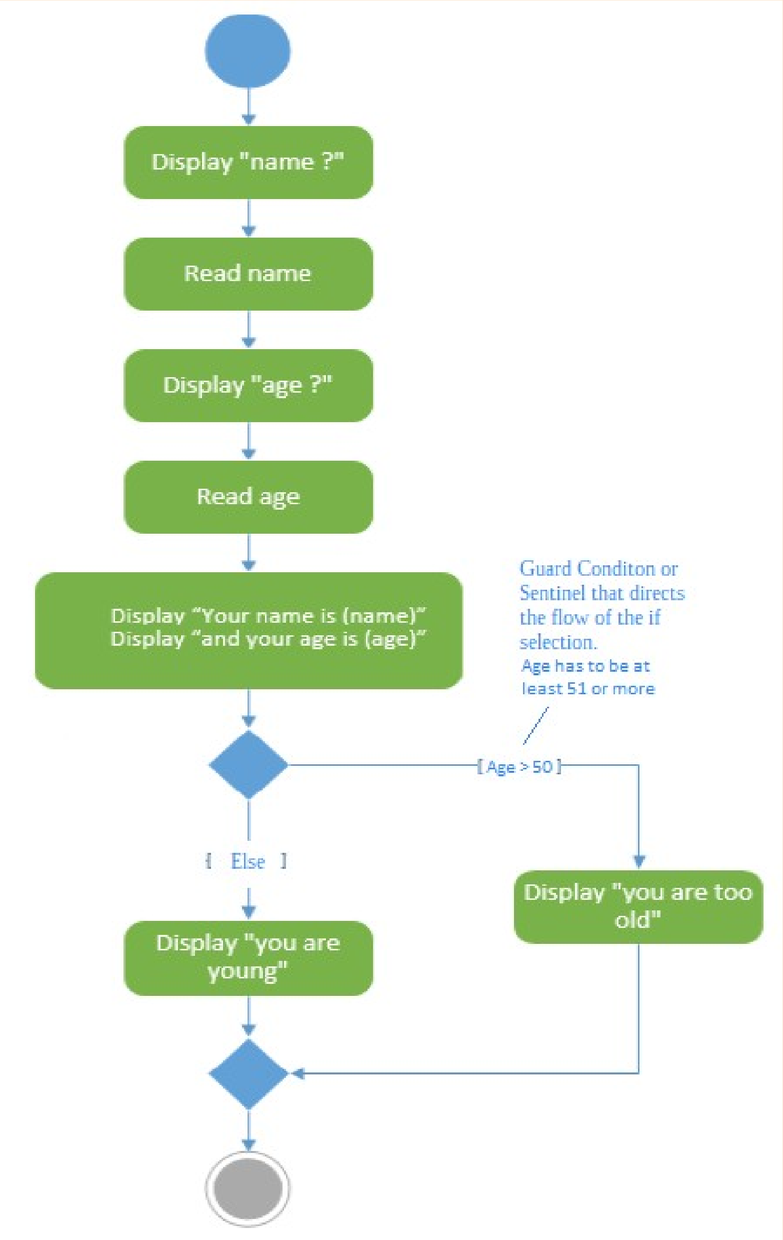


2) SELECTION OR DECISION CONSTRUCTOR

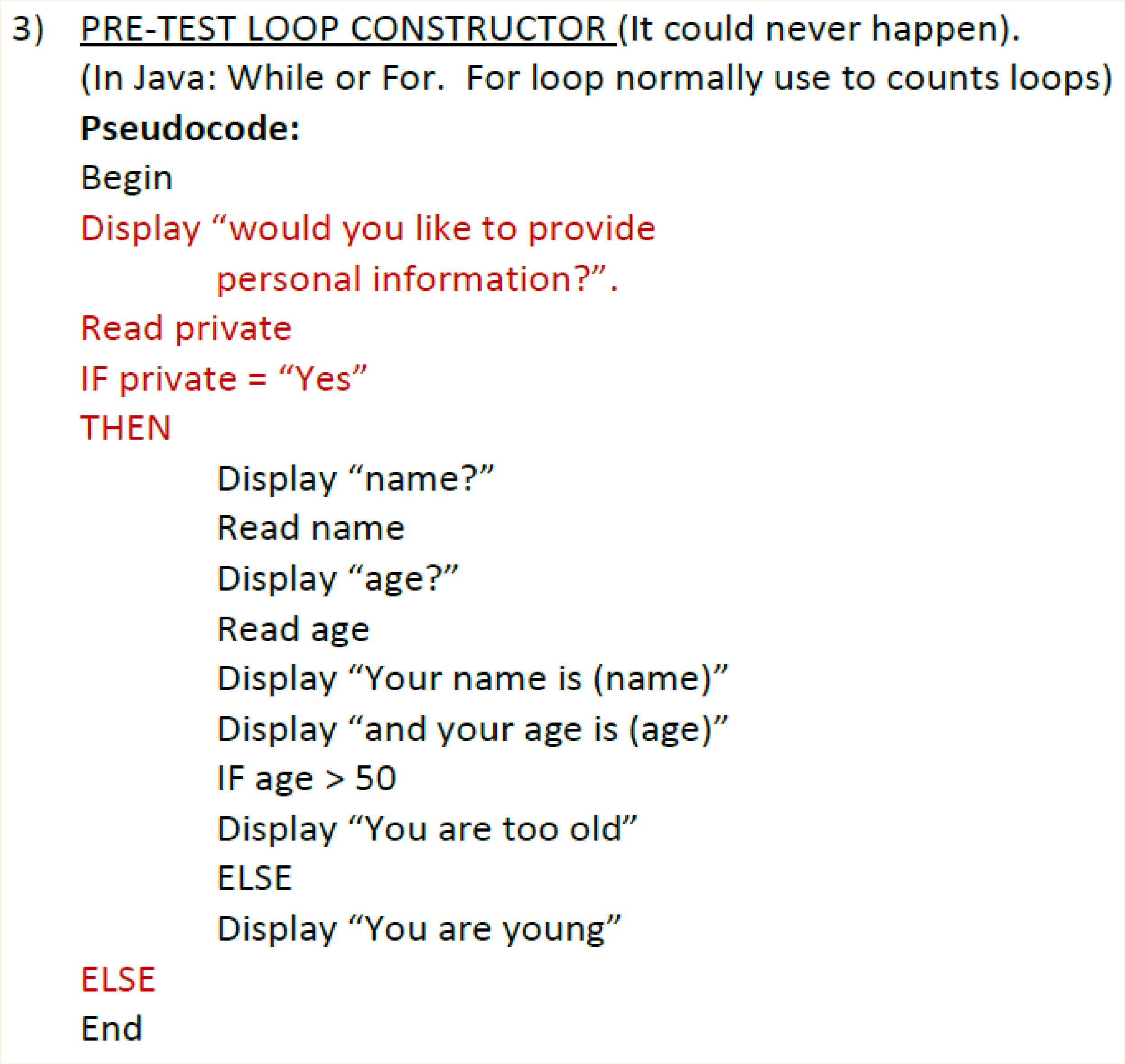
**Pseudocode:**

```
Begin
Display “name?”
Read name
Display “age?”
Read age
Display “Your name is (name)”
Display “and your age is (age)”
IF age > 50
Display “You are too old”
ELSE
Display “You are young”
End
```

# *Decision Structure – flowchart*


Display "name ?"
Read name
Display "age ?"
Read age
Display "Your name is (name)"
Display "and your age is (age)"
Guard Conditon or Sentinel that directs the flow of the if selection.
Age has to be at least 51 or more
[ Age > 50 ]
[ Else ]
Display "you are too old"
Display "you are young"

# Loop Structure – pseudocode



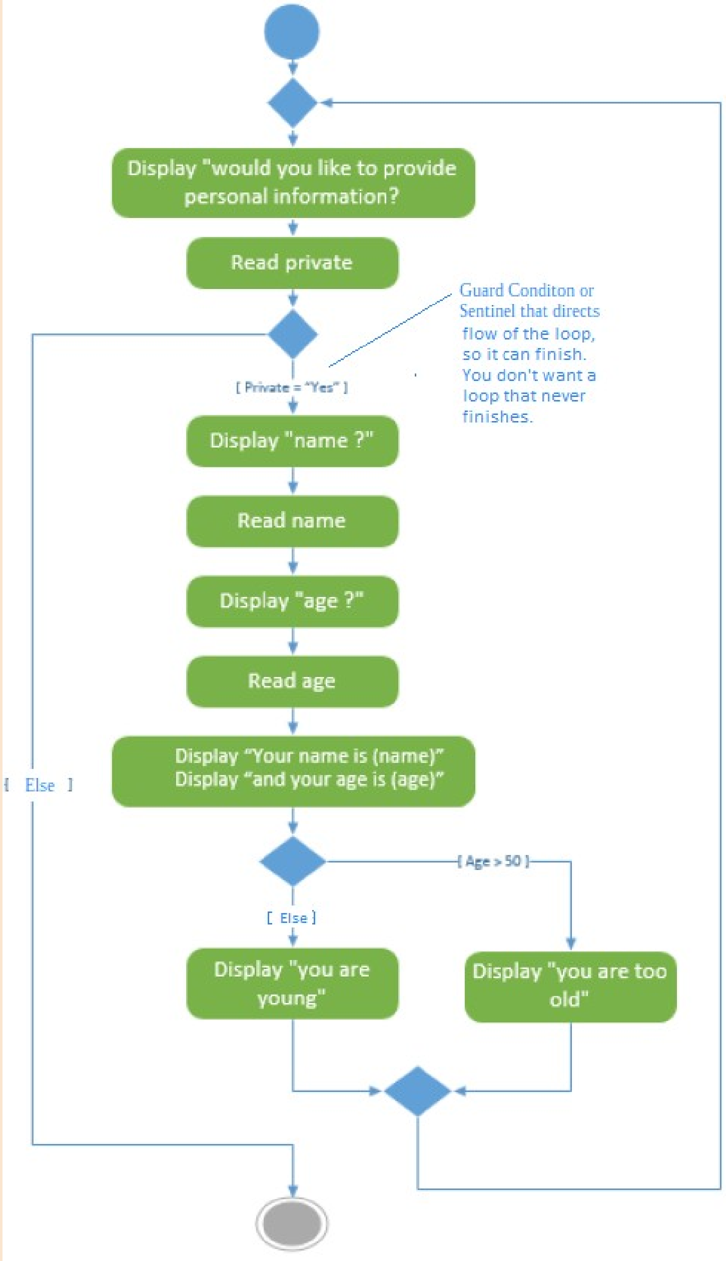
3) PRE-TEST LOOP CONSTRUCTOR (It could never happen).

(In Java: While or For. For loop normally use to counts loops)

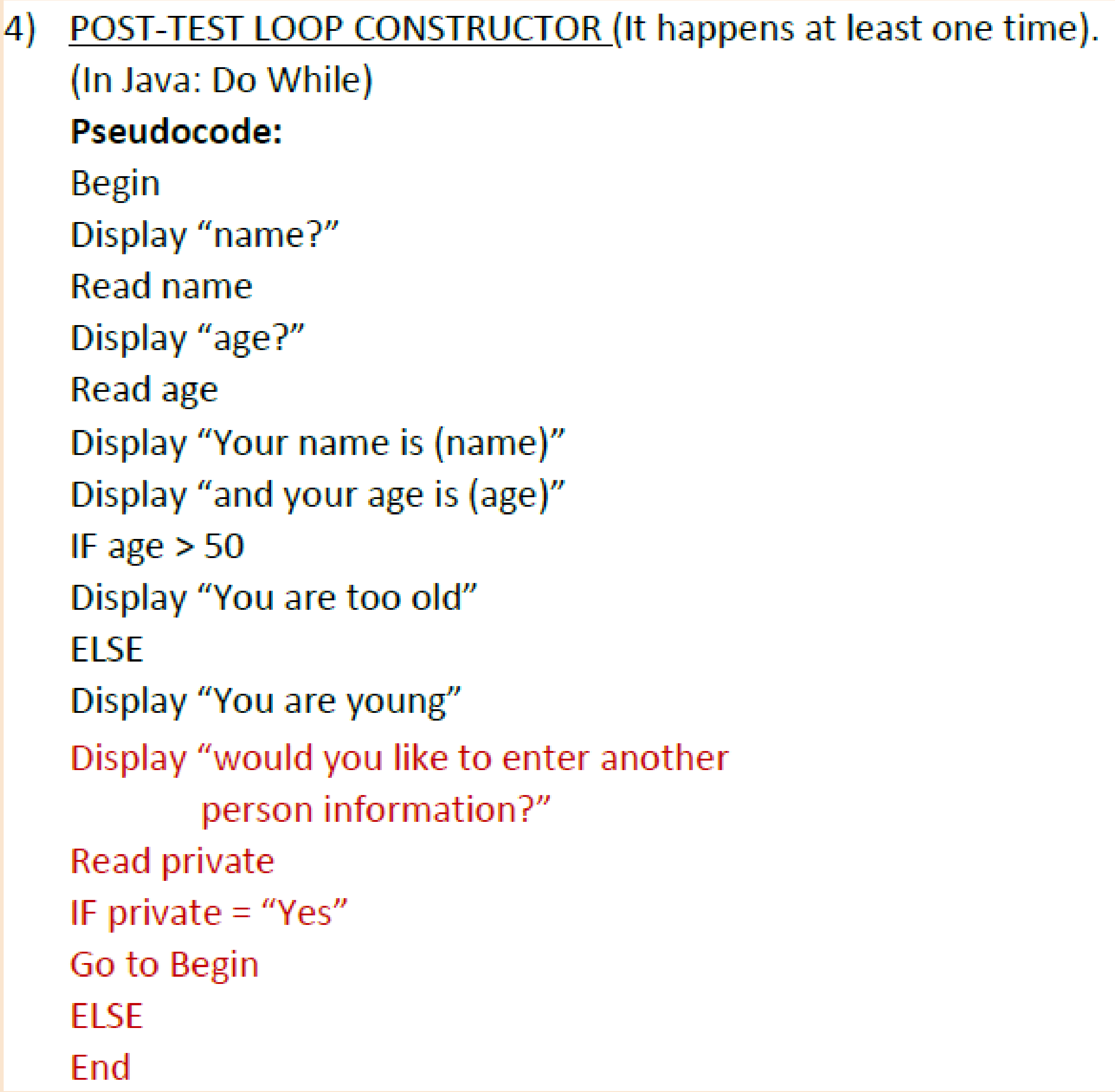
**Pseudocode:**

```
Begin
Display "would you like to provide
        personal information?".
Read private
IF private = "Yes"
THEN
        Display "name?"
        Read name
        Display "age?"
        Read age
        Display "Your name is (name)"
        Display "and your age is (age)"
        IF age > 50
        Display "You are too old"
        ELSE
        Display "You are young"
ELSE
End
```

# Loop Structure – flowchart


Display "would you like to provide personal information?
Read private
[ Private = "Yes" ]
Guard Conditon or Sentinel that directs flow of the loop, so it can finish. You don't want a loop that never finishes.
Display "name ?"
Read name
Display "age ?"
Read age
Display "Your name is (name)" Display "and your age is (age)"
[ Else ]
[ Age > 50 ]
[ Else ]
Display "you are young"
Display "you are too old"

# Loop Structure – pseudocode

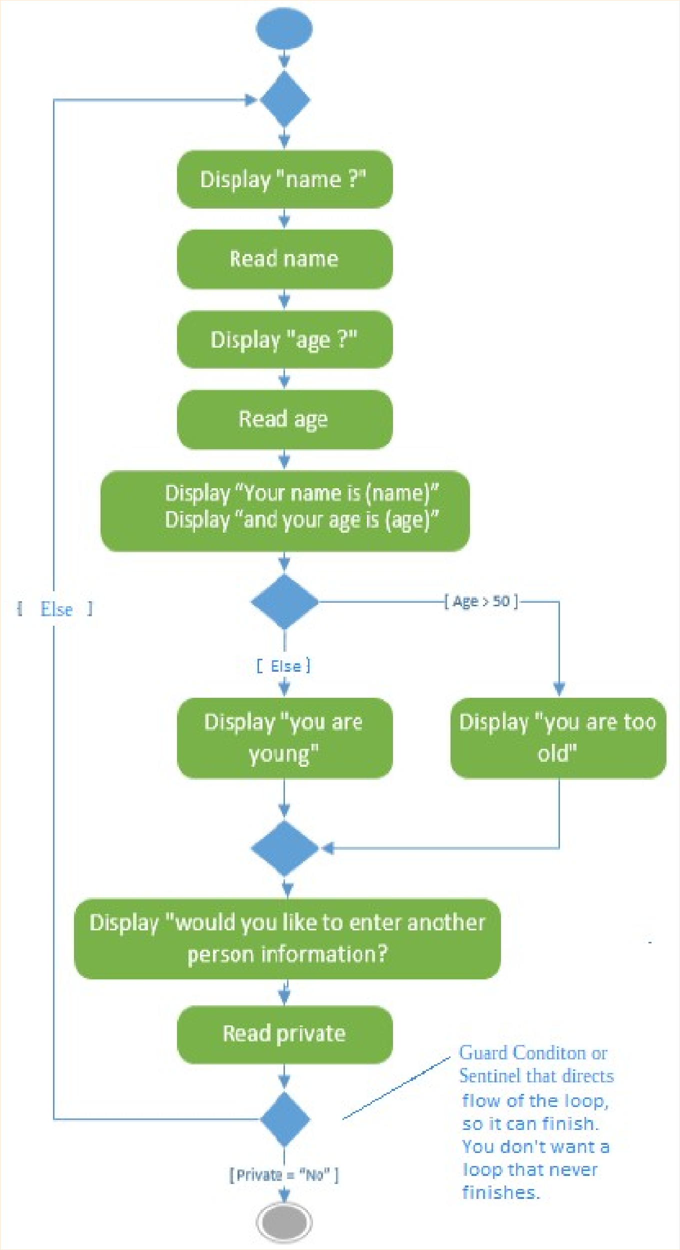


4) POST-TEST LOOP CONSTRUCTOR (It happens at least one time).
(In Java: Do While)

**Pseudocode:**

```
Begin
Display “name?”
Read name
Display “age?”
Read age
Display “Your name is (name)”
Display “and your age is (age)”
IF age > 50
Display “You are too old”
ELSE
Display “You are young”
Display “would you like to enter another
          person information?”
Read private
IF private = “Yes”
Go to Begin
ELSE
End
```

# *Loop Structure – flowchart*


Display "name ?"
Read name
Display "age ?"
Read age
Display "Your name is (name)"
Display "and your age is (age)"
[ Age > 50 ]
[ Else ]
Display "you are young"
Display "you are too old"
Display "would you like to enter another person information?
Read private
[ Else ]
[ Private = "No" ]
Guard Conditon or Sentinel that directs flow of the loop, so it can finish. You don't want a loop that never finishes.

Activity

## *Balance program*

Design a program that:

1. Defines an overdrawn penalty.
2. Asks user for balance.
3. If balance is greater than zero, adds 2% interest into balance.
4. If balance is less than zero, subtracts overdraw penalty from balance.
5. If balance is zero, display message “$0”
6. Display final balance
7.

There isn’t dropbox for this activity, just show to your instructor the pseudocode and the flowchart as a checkpoint.