Algorithmic Thinking and Structured Programming (in Greenfoot)

Teachers:

Renske Smetsers-Weeda

Sjaak Smetsers

Ana Tanase

Today’s Lesson plan (5) Jan 8th

- 10 min Looking back
  - Quiz: graded, will be discussed next week
  - What did we learn before/during vacation?

- Theory for assignment 4
- Assignment 4 print-outs: who doesn’t have one yet?
- Work on assignment 4

- 10 min Wrapping up

Retrospective assignment 3

- Nesting
- Optimization
- Submethods
- Run as an ‘Act’ loop
- (Greenfoot.stop)
- Generic solutions

Retrospective: Optimization

- Redundancy: why do we care?

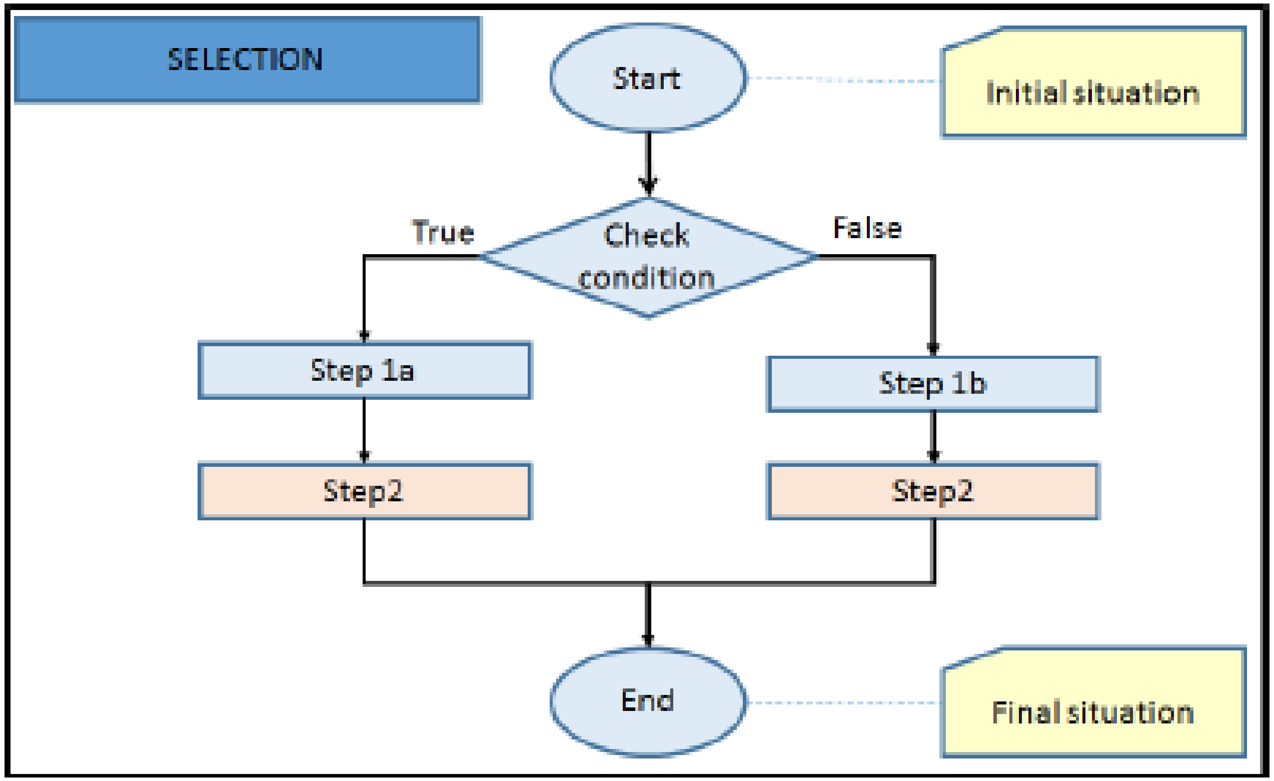
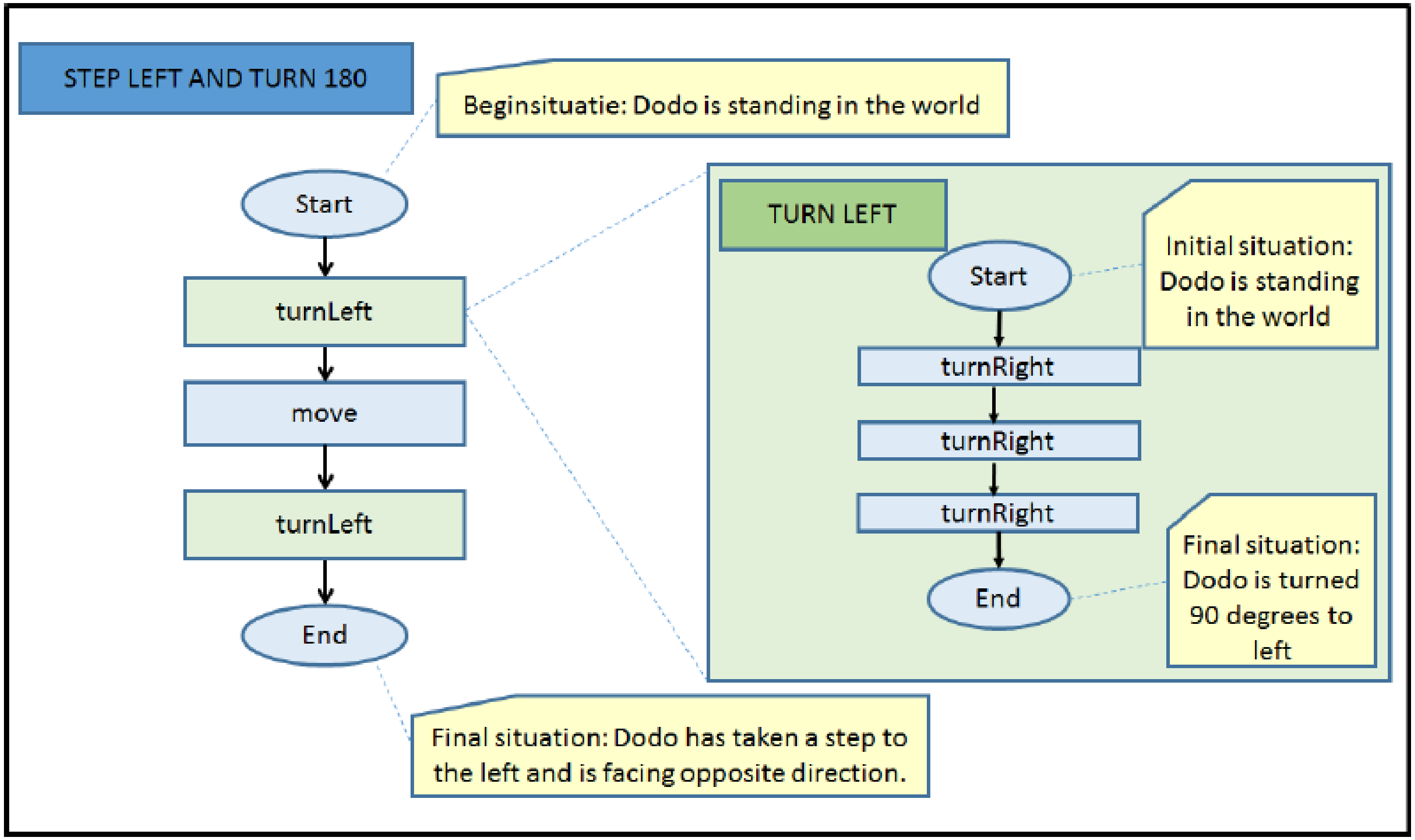


Figure 2: Flowchart with a redundant activity

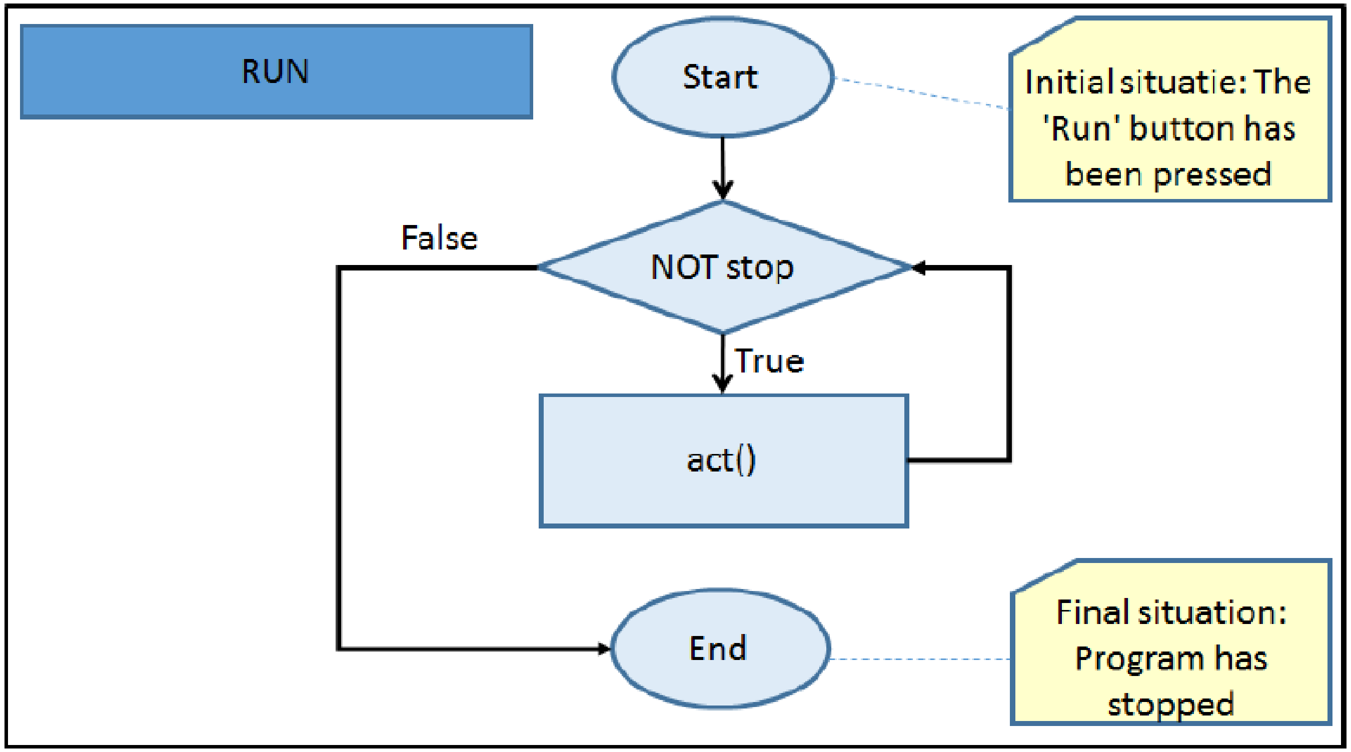
Retrospective: submethods

- Submethods: why do we bother?

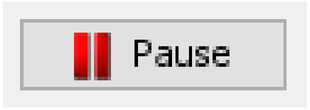

STEP LEFT AND TURN 180
Beginsituatie: Dodo is standing in the world
Start
turnLeft
move
turnLeft
End
Final situation: Dodo has taken a step to the left and is facing opposite direction.
TURN LEFT
Start
Initial situation: Dodo is standing in the world
turnRight
turnRight
turnRight
End
Final situation: Dodo is turned 90 degrees to left

Retrospective: Run

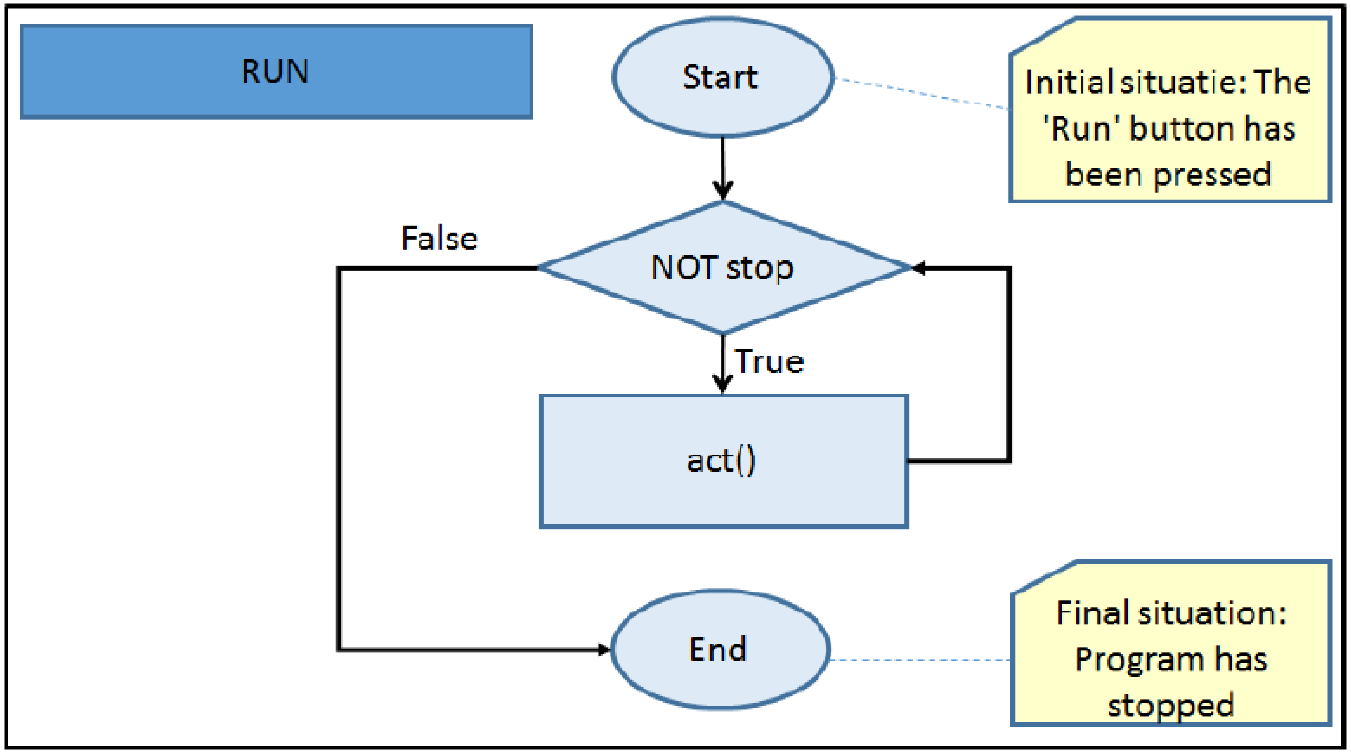
- Greenfoot Run: a while loop
- When does this stop?


RUN
Start
Initial situatie: The 'Run' button has been pressed
False
NOT stop
True
act()
End
Final situation: Program has stopped

Retrospective: Run

- Greenfoot Run: a while loop
- Only stops if:
  - User presses 

  - Calling Greenfoot.stop( ); in the code


RUN
Start
Initial situatie: The 'Run' button has been pressed
NOT stop
False
True
act()
End
Final situation: Program has stopped

Retrospective: Generic solutions

Topics assignment 4

- Conditionals:
  - boolean methods
  - logical operators: ||, &&, !
- Return statements
- Nested if-then-else
- Modularization: Breaking problem down, solving subproblems (using exsiting solutions), and combining to solve the whole problem
  - Method calls (from within other methods)
  - Advantageous when testing
- Quality criteria for programs and code

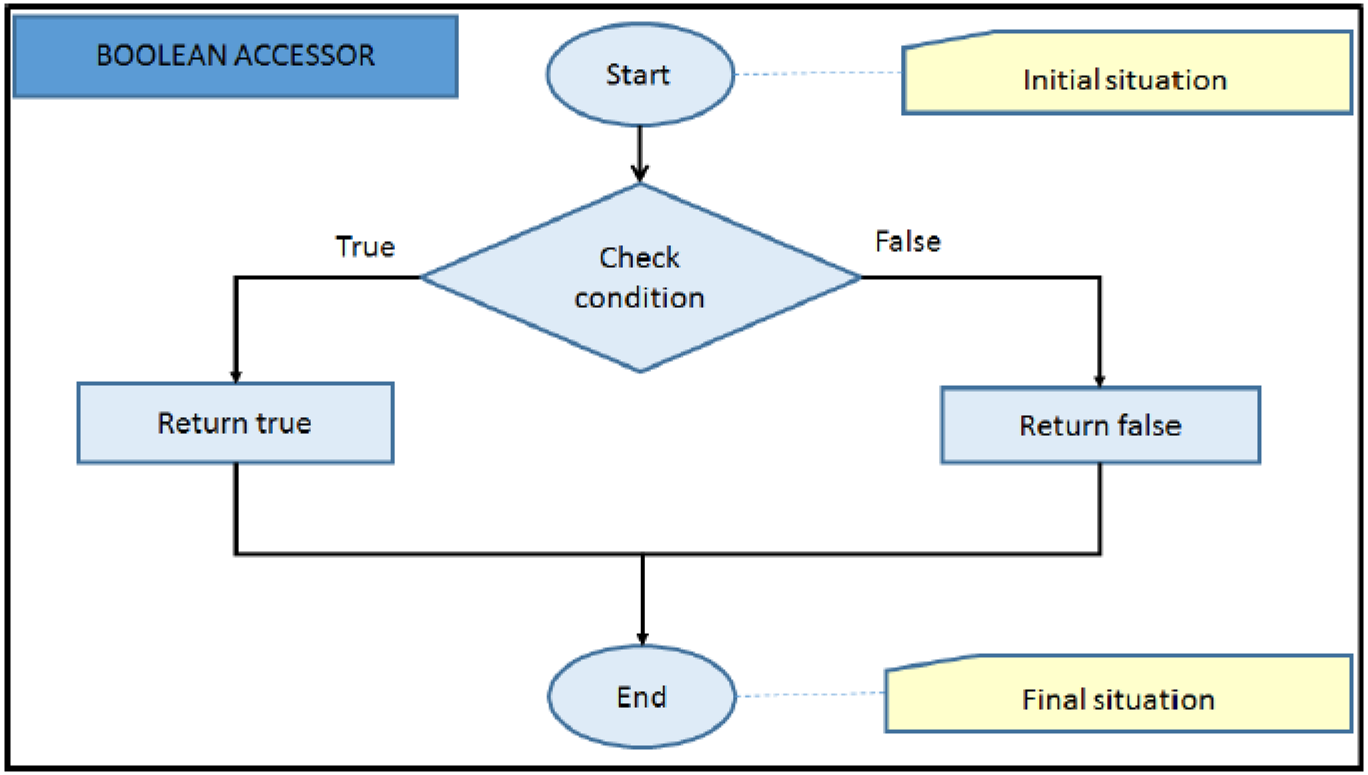
Conditionals

- Conditionals:
  - boolean methods
  - logical operators: ||, &&, !

  - || means OR
    - fenceAhead ( ) || borderAhead ( )
  - && means AND
    - canMove ( ) && eggAhead ( )
  - ! Means NOT
    - ! eggAhead ( )

Return Reminder

- Return:
  - After a return, End follows immediately
  - No more steps executed after a return



Jump Joyfully

Example with:

- Nested if-then-else
- Using return statements
- Complex Boolean statements

Jump Joyfully

Example with:

- Nested if-then-else
- Using return statements
- Complex Boolean statements

Jump up and down joyfully

If Mimi has a nest on each side,

she jumps up and down joyfully

MyDodo methods:

boolean nestAhead ( )	// returns true if nest in cell ahead
void turnLeft ( )	// turns 90 degrees clockwise
void turnRight ( )	// turns 90 degrees counterclockwise
void move ( )	// step forward if possible

Strategy:

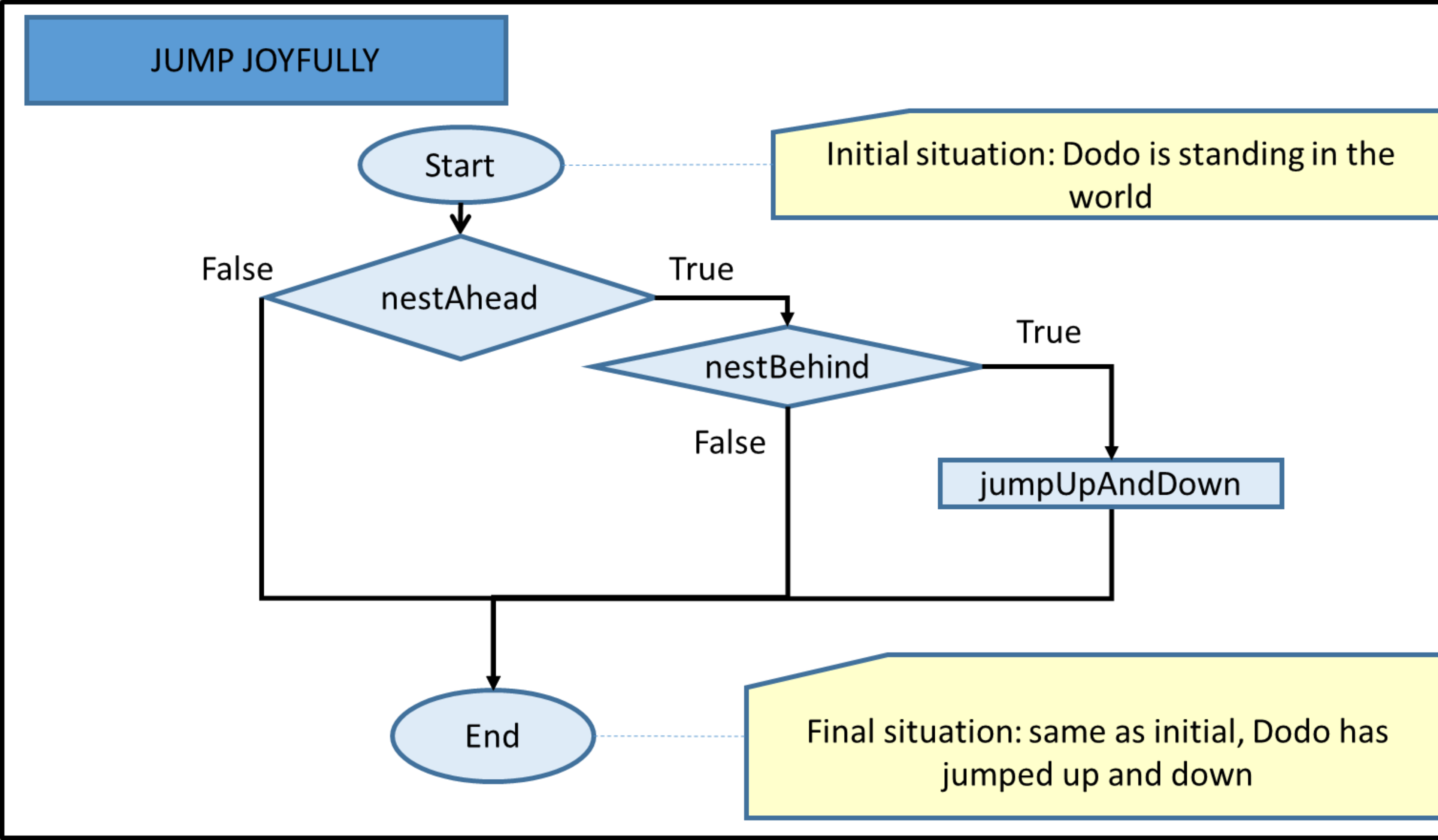
Sketch a **high-level** flowchart for jumpJoyfully

Tip:

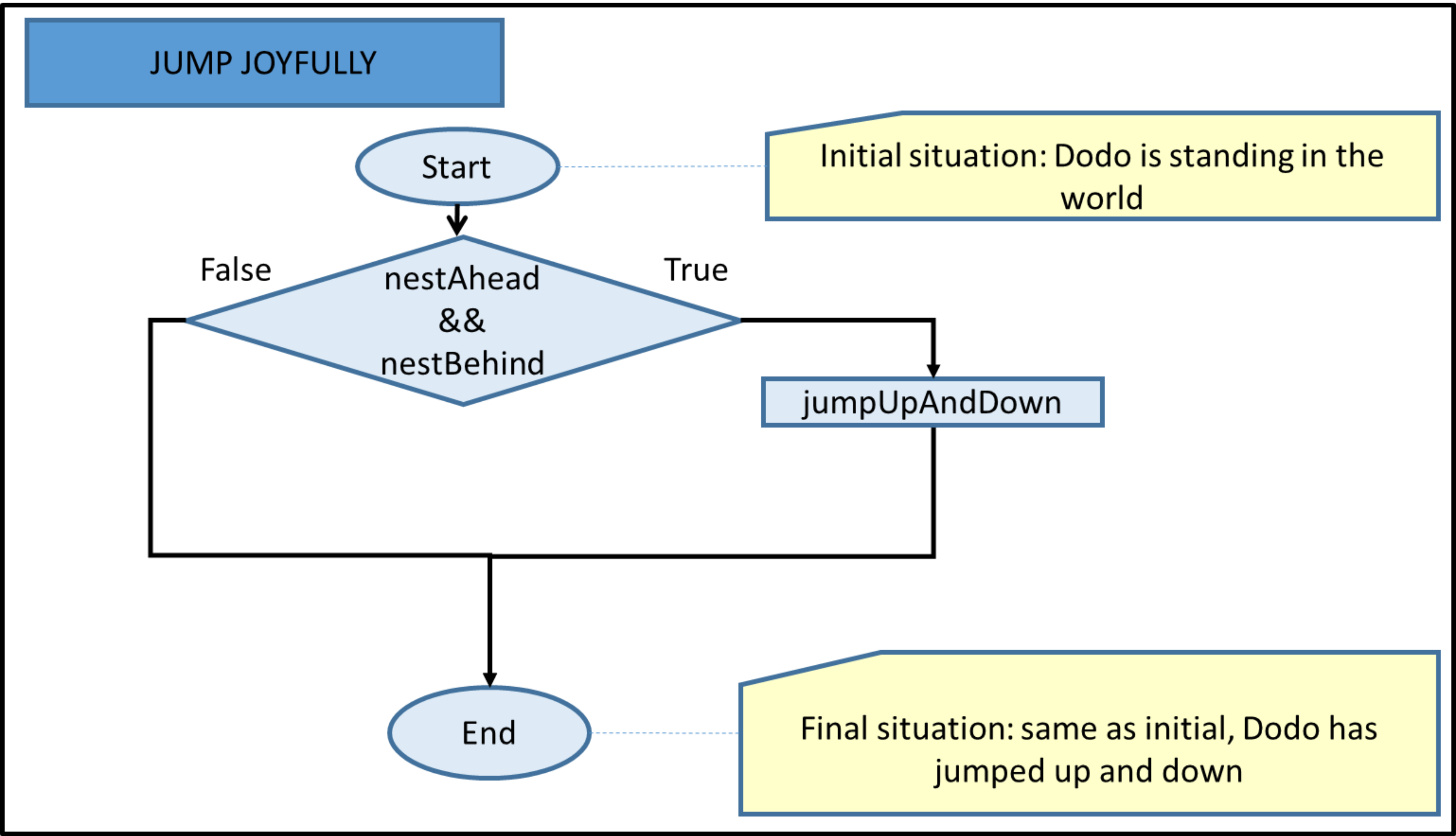
- First assume nestBehind and jumpUpAndDown exist
- Then: design, implement & test them separately

Sketch high-level flowchart

Test using: Nested if..then..else

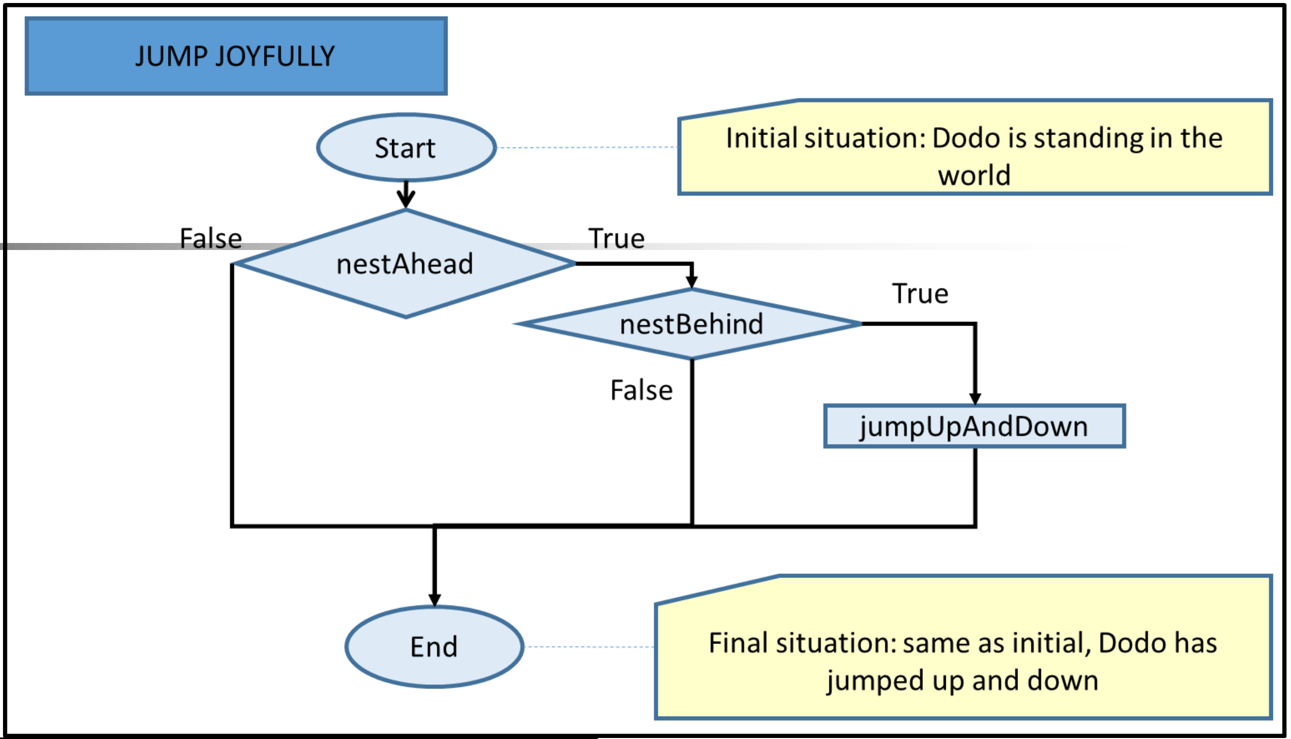

JUMP JOYFULLY
Start
Initial situation: Dodo is standing in the world
False
nestAhead
True
nestBehind
True
False
jumpUpAndDown
End
Final situation: same as initial, Dodo has jumped up and down

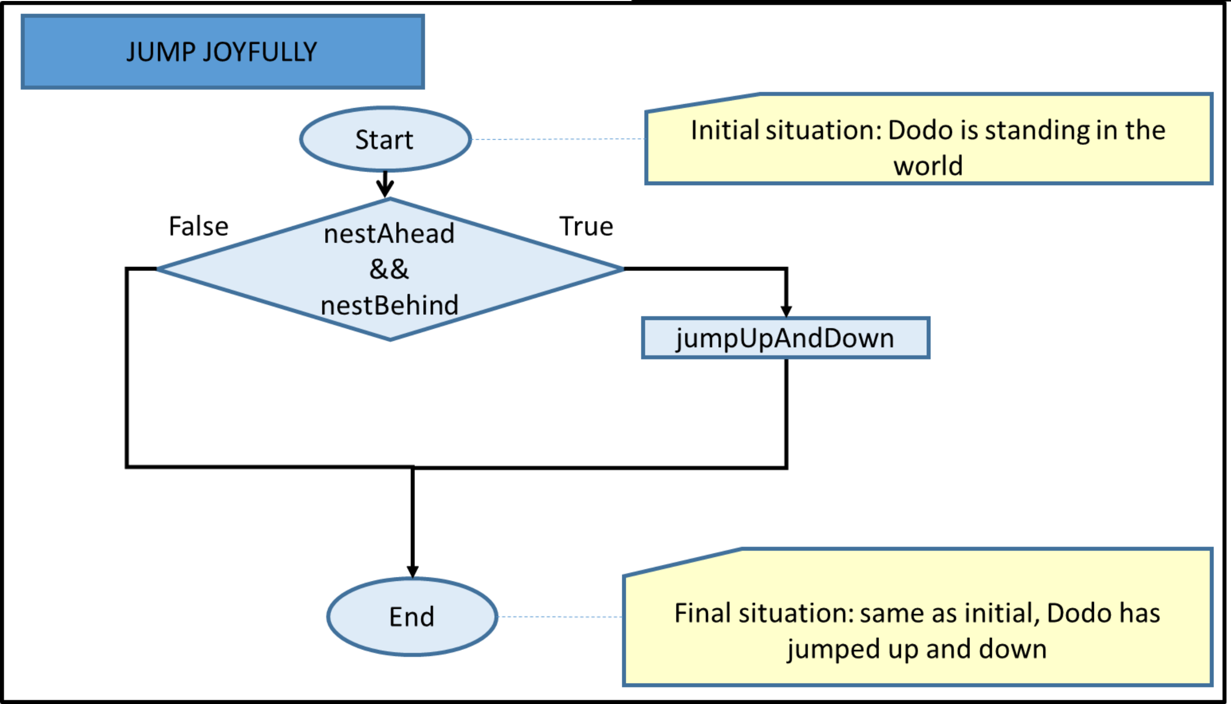
Test using: conjugated Boolean &&


JUMP JOYFULLY
Start
Initial situation: Dodo is standing in the world
False
nestAhead && nestBehind
True
jumpUpAndDown
End
Final situation: same as initial, Dodo has jumped up and down

Compare:

- Which do you prefer?
- Why?


JUMP JOYFULLY
Start
Initial situation: Dodo is standing in the world
False
nestAhead
True
nestBehind
True
False
jumpUpAndDown
End
Final situation: same as initial, Dodo has jumped up and down

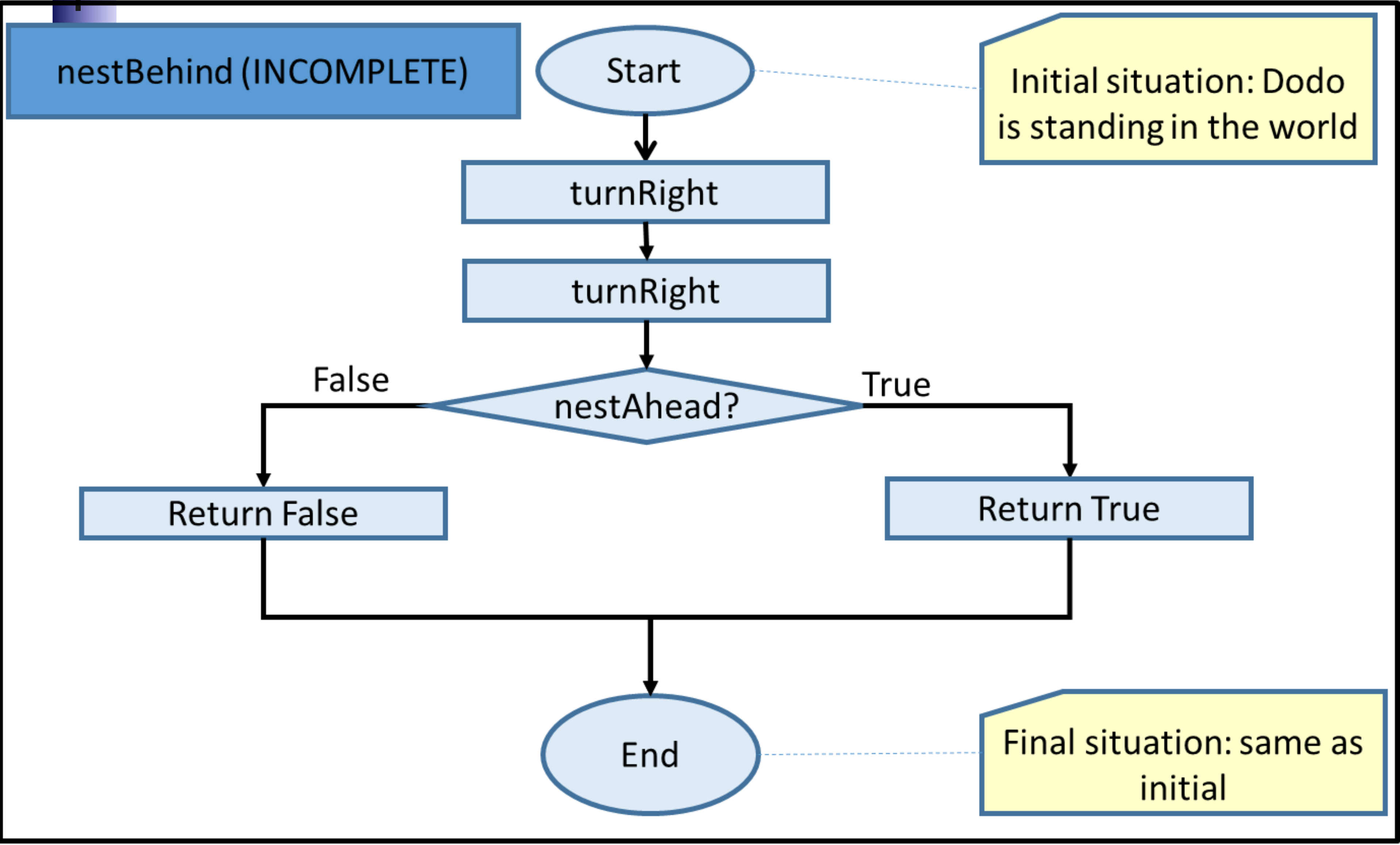

JUMP JOYFULLY
Start
Initial situation: Dodo is standing in the world
False
nestAhead && nestBehind
True
jumpUpAndDown
End
Final situation: same as initial, Dodo has jumped up and down

Now: design nestOnLeft

- ✓ Finished high-level flowchart
- .. Now the Boolean nestBehind()

Draw the flowchart

Boolean nestBehind



Now: test nestOnLeft ( )

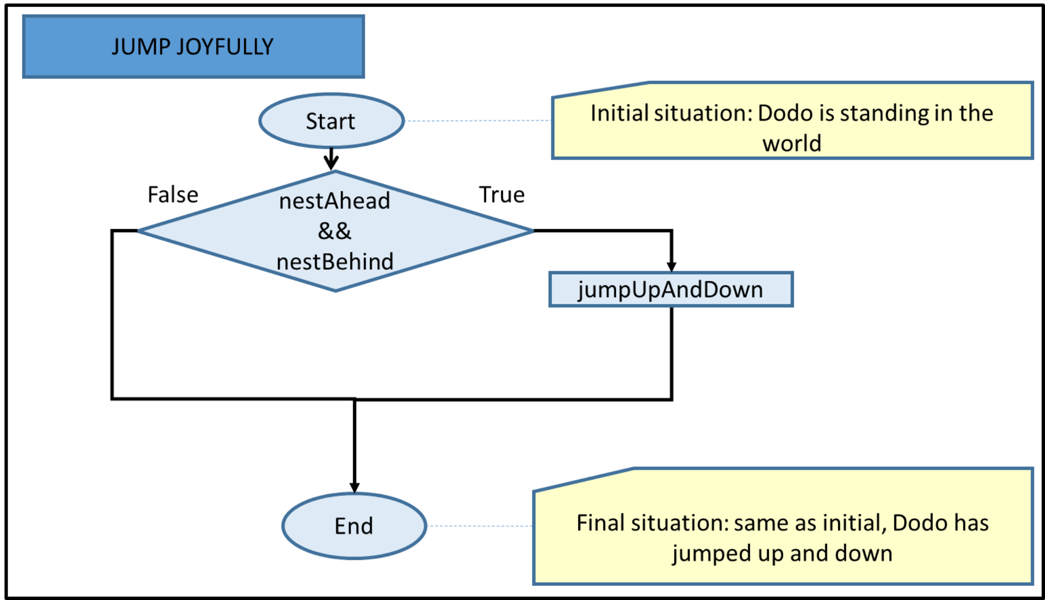
✓Finished high-level flowchart

✓Designed nestBehind( )

❑ … now test nestBehind( )

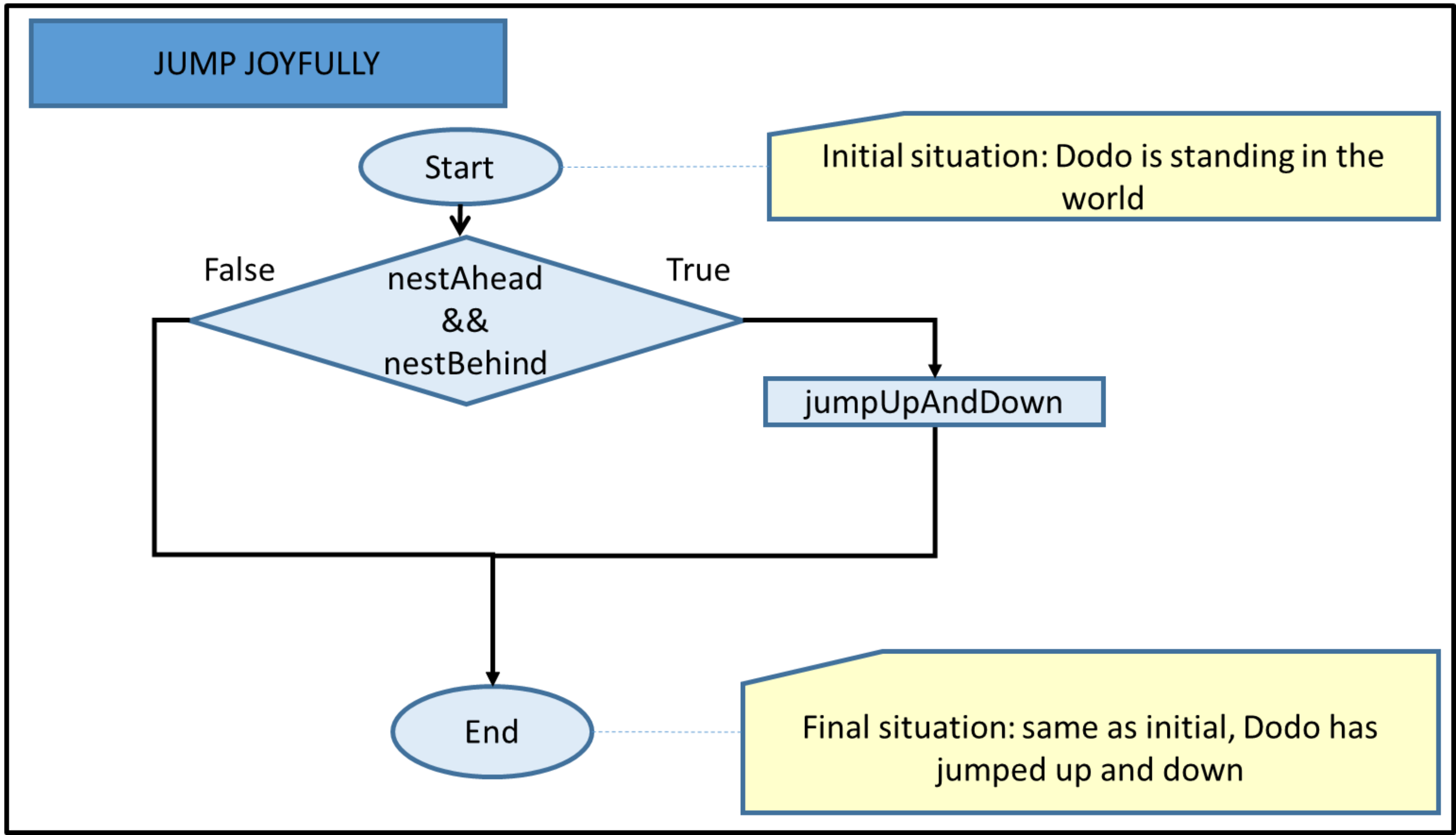
What are we doing:

Testing small pieces before we use them!


JUMP JOYFULLY
Start
Initial situation: Dodo is standing in the world
False
nestAhead && nestBehind
True
jumpUpAndDown
End
Final situation: same as initial, Dodo has jumped up and down

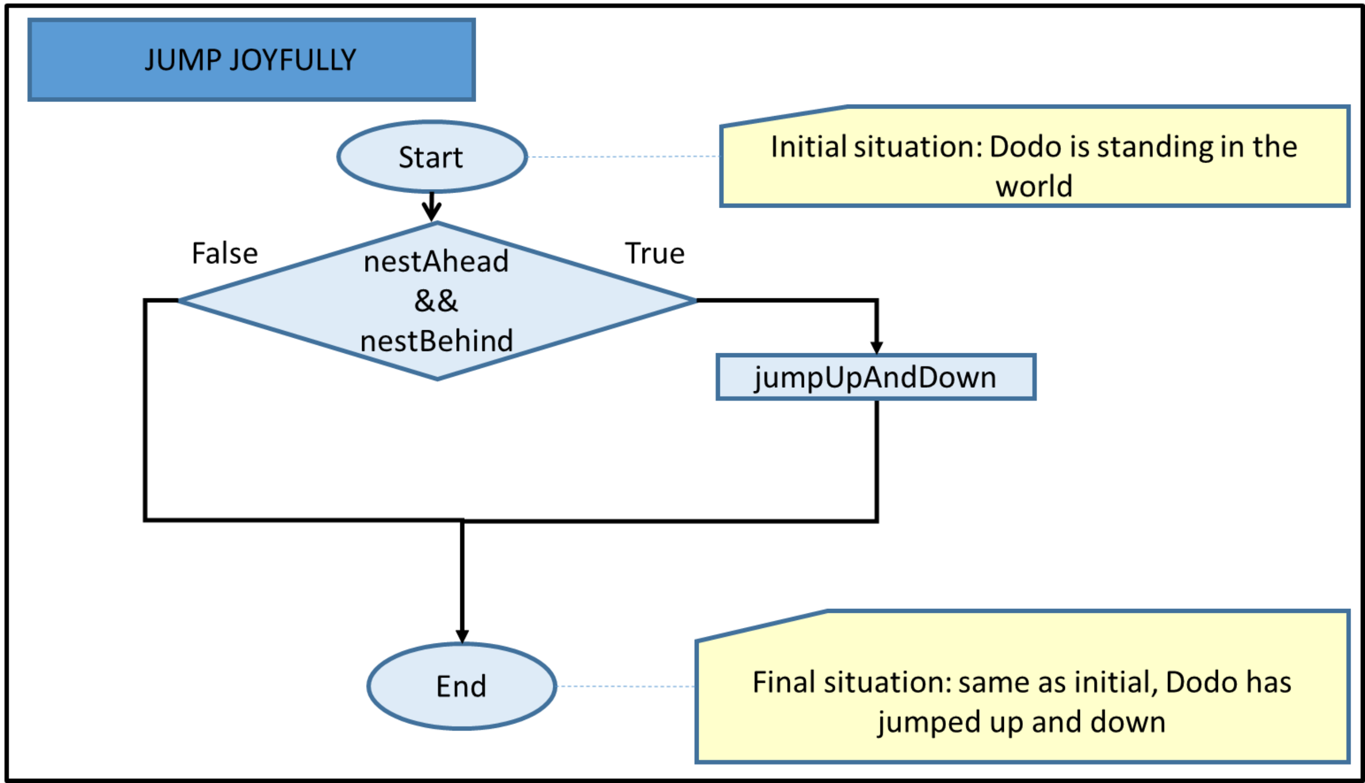
Now: design and test jumpUpAndDown

- ✓Finished high-level flowchart
- ✓Designed and tested nestBehind( )
- … now design and test jumpUpAndDown ( )


JUMP JOYFULLY
Start
Initial situation: Dodo is standing in the world
False
nestAhead && nestBehind
True
jumpUpAndDown
End
Final situation: same as initial, Dodo has jumped up and down

Now: test the whole thing

- ✓Finished high-level flowchart
- ✓Designed and tested nestBehind( )
- ✓Designed and tested jumpUpAndDown ( )
- .. Now combine parts and test whole thing: jumpJoyfully


JUMP JOYFULLY
Start
Initial situation: Dodo is standing in the world
False
nestAhead && nestBehind
True
jumpUpAndDown
End
Final situation: same as initial, Dodo has jumped up and down

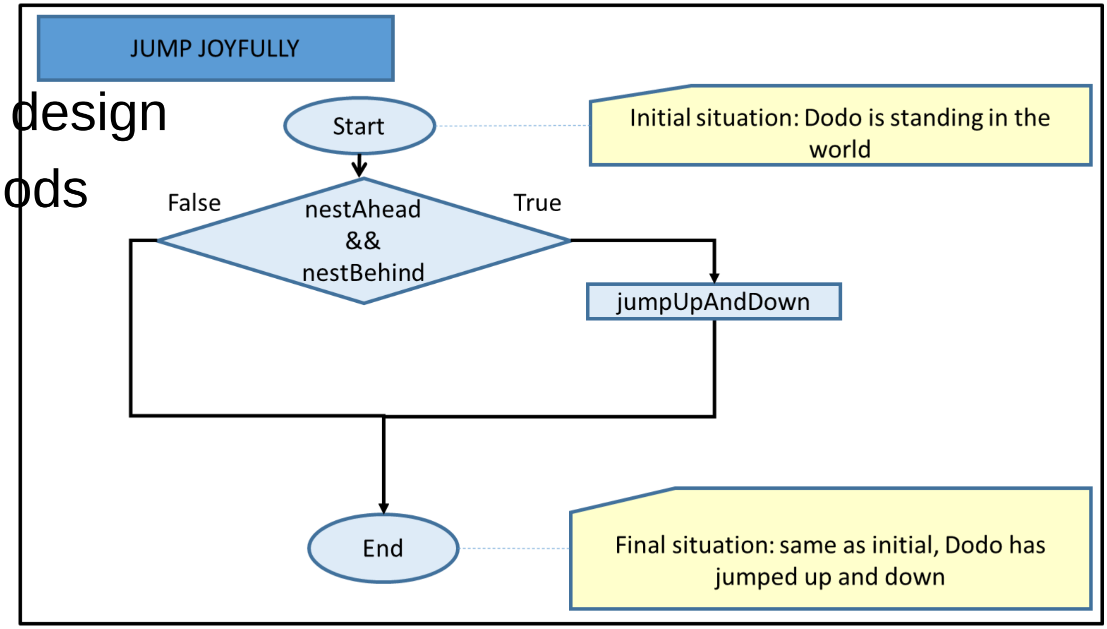
Now: enjoy and be proud

- ✓Finished high-level flowchart
- ✓Designed and tested nestBehind( )
- ✓Designed and tested jumpUpAndDown ( )
- ✓Combined parts and tested whole thing: jumpJoyfully

So, first start with high level design

Then implement small methods

Then test the whole thing


JUMP JOYFULLY
Start
Initial situation: Dodo is standing in the world
False
nestAhead && nestBehind
True
jumpUpAndDown
End
Final situation: same as initial, Dodo has jumped up and down

What did we just practice?

- Conditionals:
  - boolean methods
  - logical operators: ||, &&, !
- Return statements
- Nested if-then-else
- Modularization: Breaking problem down, solving subproblems (using exsiting solutions), and combining to solve the whole problem
  - Method calls (from within other methods)
  - Advantageous when testing

Computational thinking

- **Working in a structured manner:**
  - Breaking problems down into subproblems
  - Design, solve and test solutions to subproblems
  - Combing these (sub)solutions to solve problem
- **Analyzing** the quality of a solution
- **Reflecting** about the solution chosen and proces
- **Generalizing** and re-use of existing solutions

Work on Assignment 4

Quality criteria of solution

- Correctness: does what it should, doesn't what it shouldn't
- Efficiency: scale of (processor/memory/network) use is in proportion to problem/solution
- Elegancy/smart: generic (can be used for more problems)
- Scalability / adaptability: easily adjusted (modules/abstraction)
- Reliability: no crashes
- Maintainability: use of modules, comments, naming conventions, logical initial/final situations
- Usability: user-friendly (error messages)

Quality criteria of code

- Readible: namingconventions, modules
- Testability: test modules separately
- Flexibility: easy add-on/replace modules
- Correctness: does what is expected
- Efficiency: economical with time/memory resources

Questions?

Wrapping up

Homework for Wednesday 8:30 January 13th:

- Assignment 4:
    - All exercises
    - ZIP code and ‘IN’ and **email** to **Renske.weeda@gmail.com**

- Reflection/Evaluation: Tips & Tops