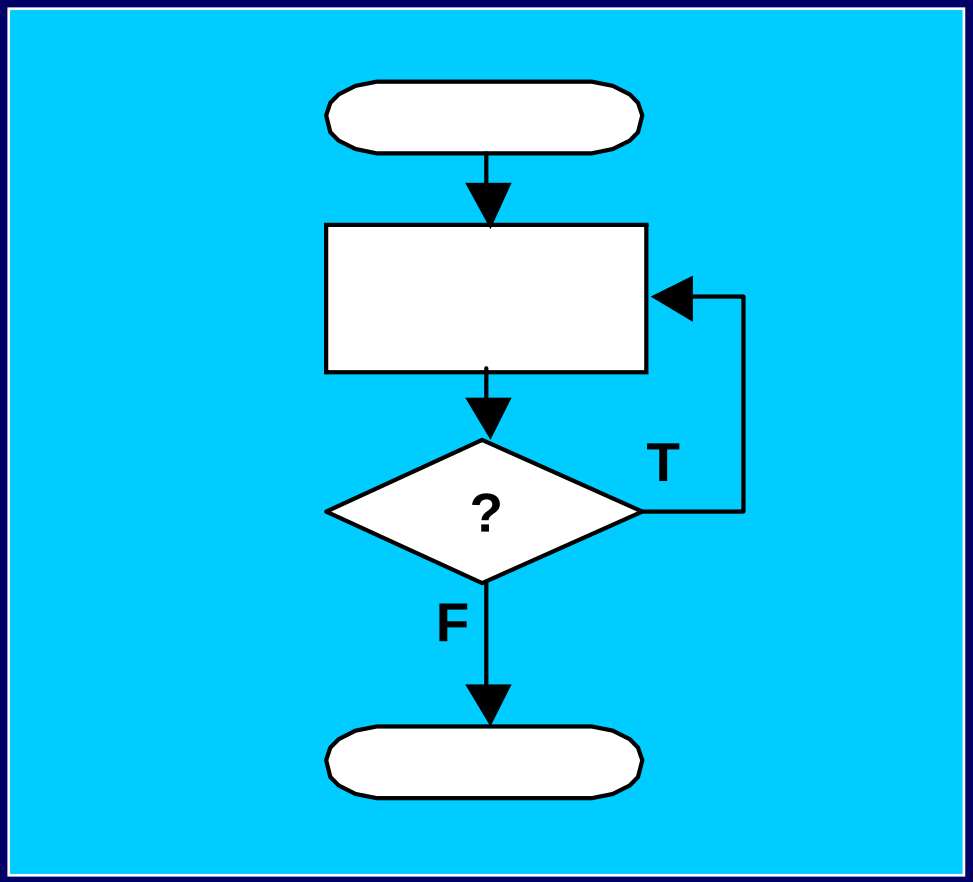
do/while Iteration Structure

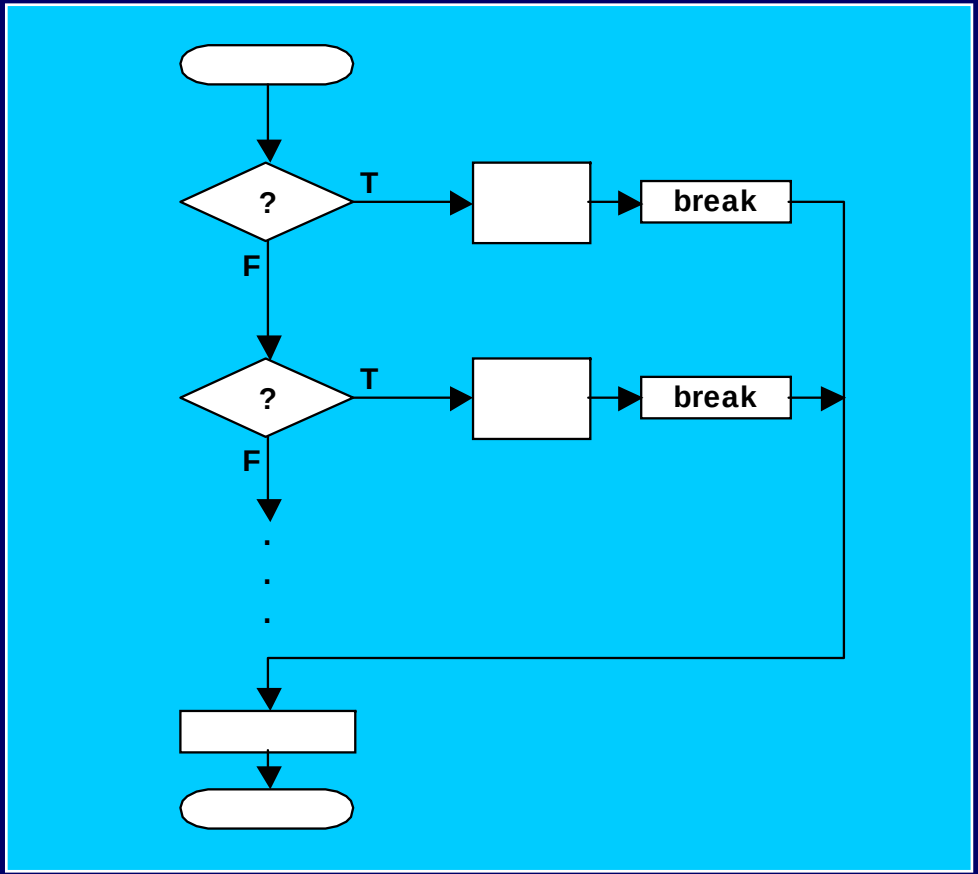


Implementing `do/while`

```
do {
    statements;
} while (boolean expression);
```

- Notes:
  - Always executes at least once
    - Good for user input checking
  - Don’t forget the semicolon at the end

switch Selection Structure



Implementing `switch`

```
switch (controlling integral expression) {
    case constant integral expression:
            statements;
            break;  // important
    case constant integral expression:
            statements; break;
    ...
    default:
            statements to do if no case matches;
}
```

- See updated GradeBook.java (Fig 5.9, pp. 171-173)

`switch` Notes

- Do NOT forget the `breaks`!
- Integral types only:
  - Just `byte`, `short`, `int`, `char` (but not `long`)
  - And new Java 5 feature – enumeration types
    - e.g, `enum Section {FIELD, LOGE, PAVILION};`
- *Constant* integral expression (a.k.a., `case` "label") :
  - No ranges, but can stack, like:

    ```
    case 1: case 2: case3:
    ```
- Can *always* rewrite as nested `if` statements
  - Safer, more structured, recommended in most cases

`break` and `continue`

- Ways to get around the strict structures
  - i. e., not really structures anymore
  - `break` – completely exits the structure
  - `continue` – skips the rest of current iteration (`while`, `for`, or `do/while` structures only)
- Also labeled versions for nested structures
- Usual advice is to find a better way
  - i.e., *should* look for a structured alternative

Boolean operators: &&, ||, !

- For combining simple boolean expressions into more complex expressions
  - Operands are boolean expressions
  - e.g., `grade == 'A' && weight > 10`
    - Note: relational operators have higher precedence
- Truth tables – whole result given partial results
  - `op1 && op2` - true if *both* operands are true
  - `op1 || op2` - true if *either* operand is true
  - `!op` - true if operand is false
  - See LogicalOperators.java (Fig. 5.18, pp. 184-185)
- Note: `&&` has greater precedence than `||`

Quiz - Logical Expressions

- Actually just a self-test *(but please try anyway)*
- Say `int x=2, y=8, z=17;`
  - What is (`x<y || z>x && y>z`)? ⇐ **`true`**
  - What is (`x < y-z`)? ⇐ **`false`**
  - What is (`x+y > y + z / y`)? ⇐ **`false`**
- What is (`--z == x * y++`)? ⇐ **`true`**
  - And after that statement executes, what is

    (`z > y * x`)? ⇐ **`false`**

How did you do?

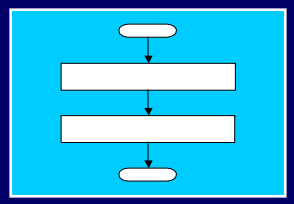
Number correct	Interpretation
5	Very savvy
4	Expected after about ½ of CS 5JA
3	Lagging
2 or less	Might as well flip a coin!

More boolean expressions

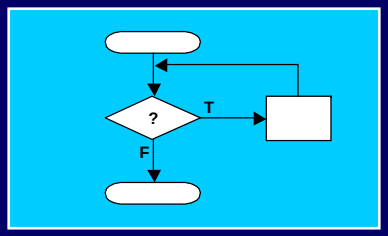
- Note a difference from math descriptions:
  - In math: `(0 < amount < 1000)`
  - In Java: `(0 < amount && amount < 1000)`
- De Morgan's Law – has 2 forms, both useful to simplify boolean expressions
  - Let `A` and `B` represent boolean values
    1. `!(A && B)` is the same as `!A || !B`
    2. `!(A || B)` is the same as `!A && !B`
- Q: How say `not( 0 < amount < 1000)`?

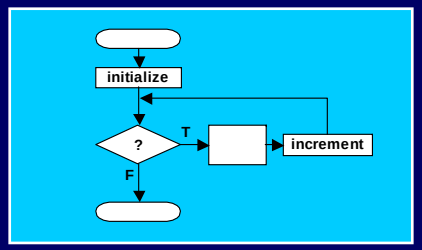
Review: 7 control structures

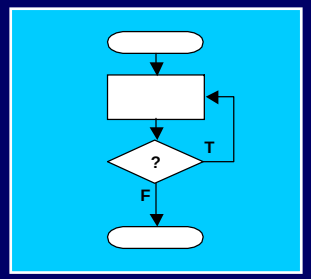
Sequence



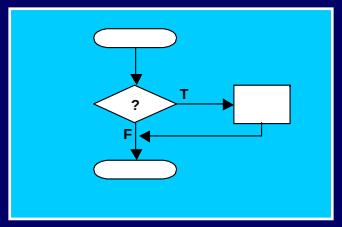
Iteration

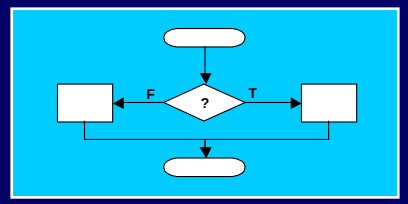


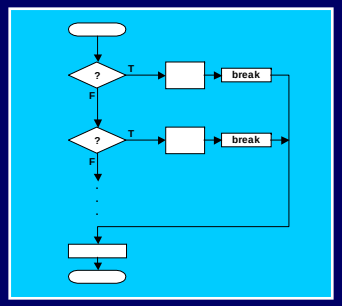




Selection





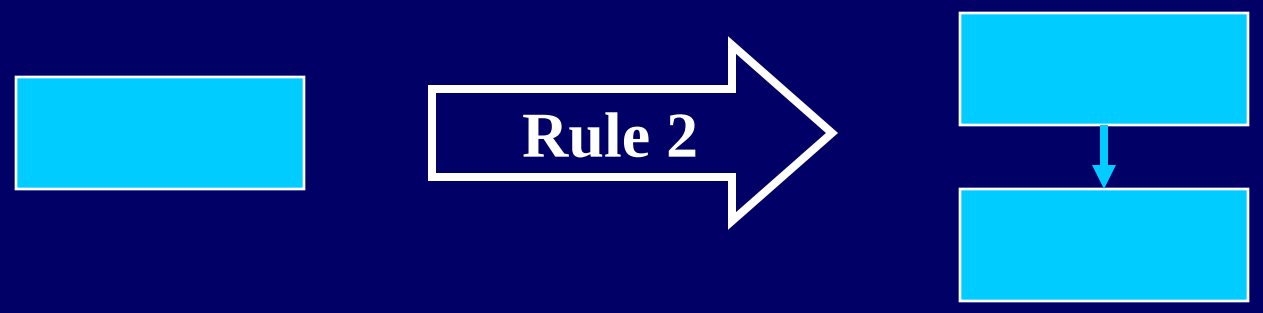


Structure “rule” #1: start with the simplest flowchart

- One rectangle
- A good (and widely applicable) example:

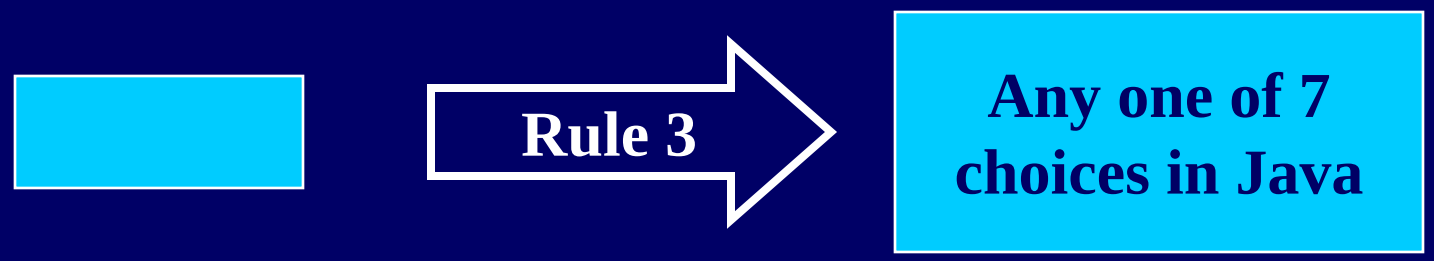
  get some data, calculate and show some results
- Really just a way to start; clarifies the “big picture”

Rule #2: replace any rectangle by two rectangles in sequence



- This "stacking rule" can apply repeatedly: one→two, two→three, … For example:
  1. Get data
  2. Process
  3. Show results

Rule #3: replace any rectangle by any control structure



- This "nesting rule" also applies repeatedly, as each control structure has rectangles
- e.g., nest a `while` loop in an `if` structure:

```
if  (n > 0)
    while (i < n)
        System.out.println(i++);
```

Rule #4: apply rules #2 and #3 repeatedly, and in any order

- Stack, nest, stack, nest, nest, stack, … gets more and more detailed as one proceeds
  - Think of control structures as building blocks that can be *combined in two ways only*.
  - Captures the essence of stepwise refinement: keep adding details as they arise
    - Basically means keep adding control structures as long as they are needed
- Top-down design: start with forest, do trees later

Programming graphics

- Need a window – `javax.swing.JFrame`
  - Several essential steps to use (necessary "plumbing"):
    - Set the size – width and height in pixels
    - Set a title (optional), and a close operation
    - Make it visible
  - e.g., see lines 20-25 of ShapesTest (Fig. 5.27, p. 193)
- Add `javax.swing.JComponent`s to window
  - Note: `JPanel` *is a* subclass of `JComponent`
  - Draw shapes, colors, … on these components
- That's all there is to it!
  - Except for the painstaking labor, of course