Comparing objects, like Strings

- Do NOT use `==` to test equality
  - That just compares references! For example,

    ```
    String s1 = “dog”;
    String s2 = “DOG”.toLowerCase();
    s1 == s2
    ```
    // false! – *different objects*
- Use `equals` method instead (if defined by class)

    `s1.equals(s2)` // true – *same contents*
  - But not all classes define equals method. Be careful.
- Some objects (like `String`s) are `Comparable`, so

    `s3.compareTo(s4)` // returns -1, 0, or 1

`boolean` variables

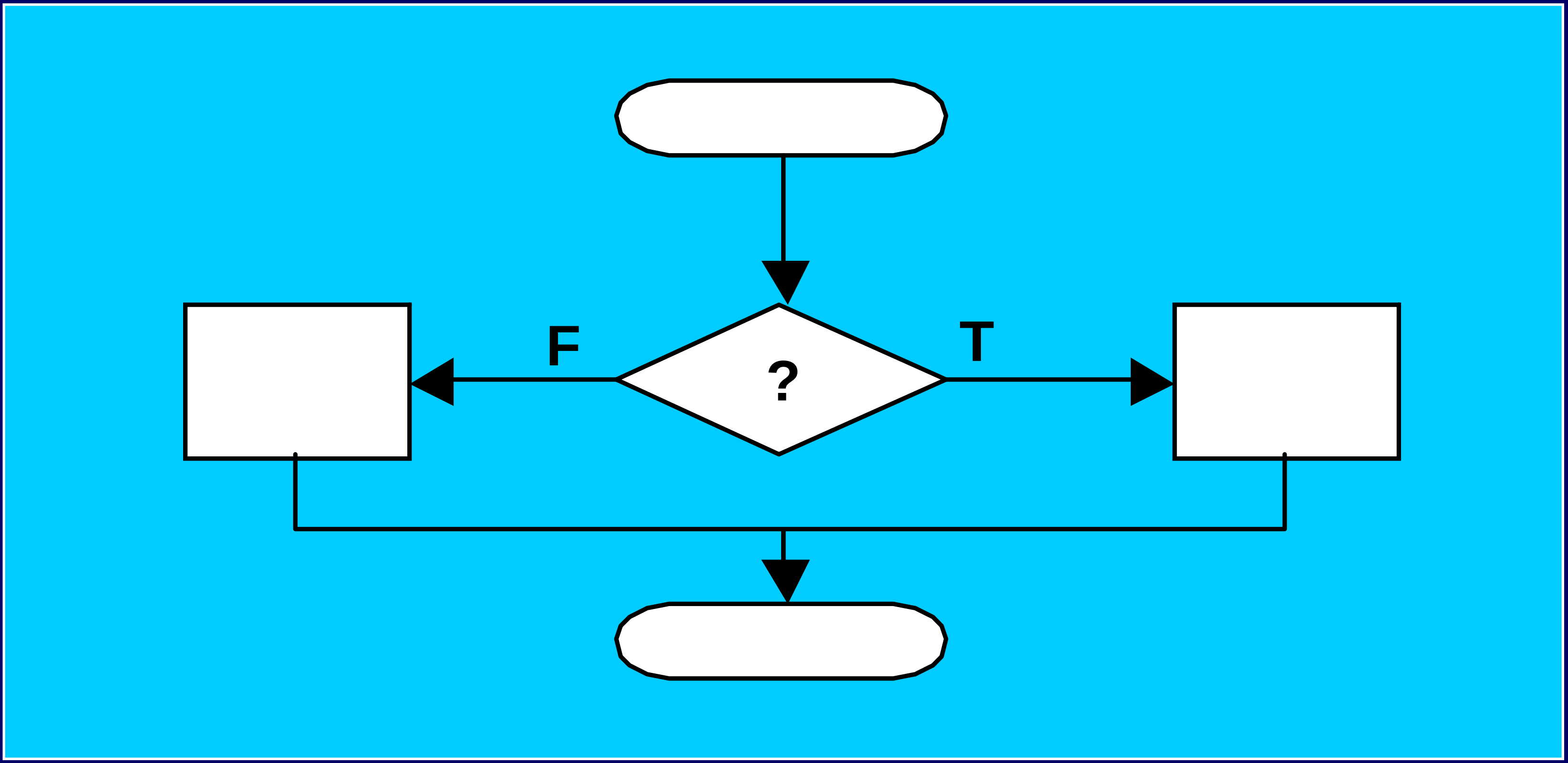
- A primitive type to store true or false
  - e.g.,

```
boolean done = false;
...
if (!done) {
   ...
   done = true;
}
```

- Often used just for readability:

```
boolean pass = grade >= 70;
if (pass) ...
```

if/else Selection Structure



Implementing `if/else`

- General way – use `if` and `else`:

```
if (grade >= 60)
    message = “Pass”;
else
   message = “Fail”;
```

  - Either clause can be a block – i.e., `{…}`
- Sometimes – use selection operator:

```
message = grade >= 60 ? “Pass” : “Fail”;
```

    // same result as `if/else` above

  - Applications are much more limited though

Nesting & indenting

- No such thing as multiple else blocks – others actually *nested* inside else block
  - e.g.,

```
if (grade >= 90)
  message = “Excellent”;
else
  if (grade >= 60)
    message = “Pass”;
  else
    message = “Fail”;
```

  - Gets messy, so usually else/if on same line:

```
else if (grade >= 90) …
```

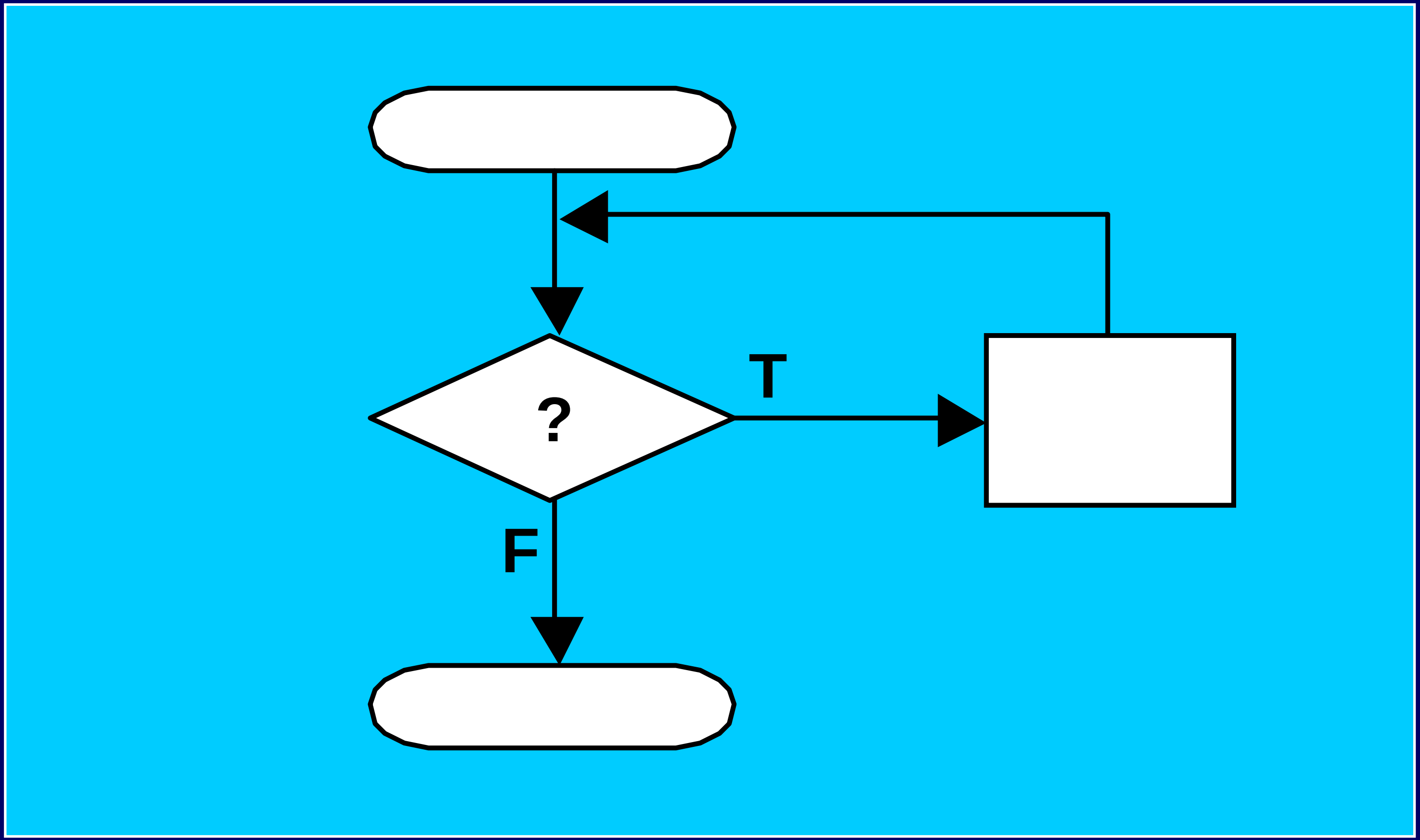
Nesting/indenting (cont.)

- Critical to test relations in the correct order
  - Sometimes means stating the *negative condition*
- Also watch out for “dangling else” problems

```
if (first-level condition)
    if (second-level condition)
        do something;
else (what level?) …
```

→ | this else should be indented to here

while Iteration Structure



Implementing/applying `while`

`while (`*boolean expression*`)`
    operation`;` *// or a block, delimited by* `{ }`

- Can be used for counter-controlled loops:

```
int counter = 0;     // initialize
while (counter < 10) {   // compare to limit
    System.out.println(counter*counter);
    counter = counter + 1;   // increment
}
```

  - Must: (1) initialize, (2) check against limit, (3) increment
  - See related version of GradeBook.java (Fig. 4.6, pp. 119-121)

Applying `while` (cont.)

- Processing unlimited amounts of input data
  - e.g., better GradeBook.java (Fig. 4.9, pp. 127-128) – reads grades until sentinel entered by user
- Special note: watch out for endless loops!
  - i.e., boolean expression never becomes false
    - Use `ctrl^C` at command line to interrupt
  - But some situations call for it – in such cases:

    `while (true) ...` // intention is clear this way

Notes about type conversions

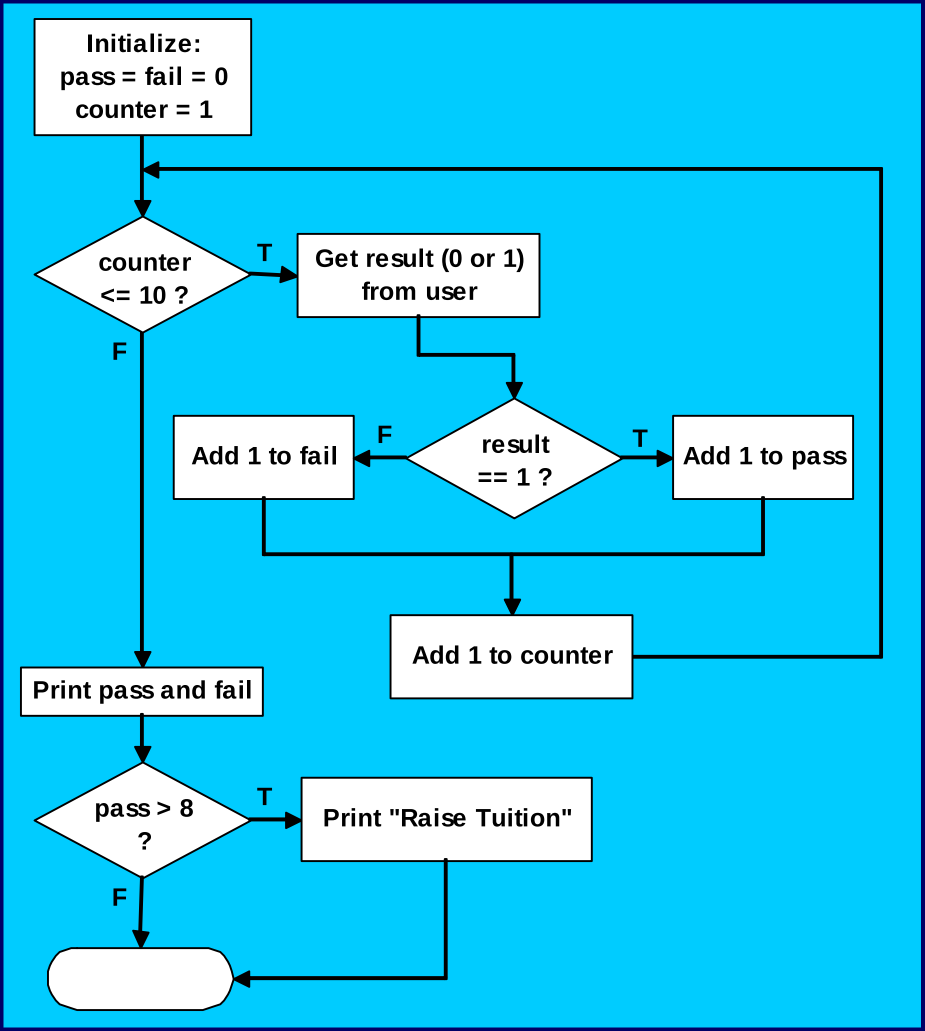
- Automatically applies to *promotions* only:
  - e.g., `int n = 5; double d = n;` // okay
    - `n` is "promoted" to `double` before assignment happens
  - e.g., `int n = 5; double d = n/2.0;` // okay
    - `n` promoted to `double` before division; result is `double`
- Must "cast" to force other conversions:
  - e.g., `double d = 5.; int n = d;` // error

    `double d = 5.; int n = (int)d;` // okay
  - But not all casts are legal (basically must make sense):

    `String s = "dog"; int n = (int)s;` // error

Combining control structures

- Two ways only:
  - *Stack* – in sequence
  - *Nest* – one inside other
- Analysis.java (Fig. 4.12, p. 134) shows both ways
  - An if/else structure inside a while loop
  - And an if structure in sequence after the while loop



Aside – simple drawings

- Really just a preview of upcoming topic
- Need a `Graphics` object to draw on
  - Any subclass of `JComponent` – e.g., `JPanel` – can be passed one by the windowing system
    - Inherits method: `paintComponent(Graphics g)`
  - See DrawPanel.java (Fig. 4.19, p. 142)
- And a window to show it – e.g., a `JFrame`
  - See DrawPanelTest (Fig. 4.20, p. 143)

Assignment with arithmetic

- Assignment operators

  e.g., `a += 5;`

  // same as: `a = a + 5;`

  – Also `-=`, `*=`, `/=`, and `%=`
- Special forms for `+=` and `-=`, called increment and decrement operators, respectively
  - `++` increments by 1 (same as `+= 1`)

    `--` decrements by 1 (same as `-= 1`)
  - e.g. `counter++;` // same as counter = counter + 1;

Pre/post versions of ++ and - -

- Post-increment is not exactly the same as pre-increment (same goes for decrement)
- Post version changes after used in expression

  e.g., say `x = 7`, then

  ```
  System.out.println(x++);
  ```

  would print 7
- Pre version changes before it is used

  ```
  System.out.println(++x);
  ```

  would print 8.
  - In either case, `x` equals 8 after the print.

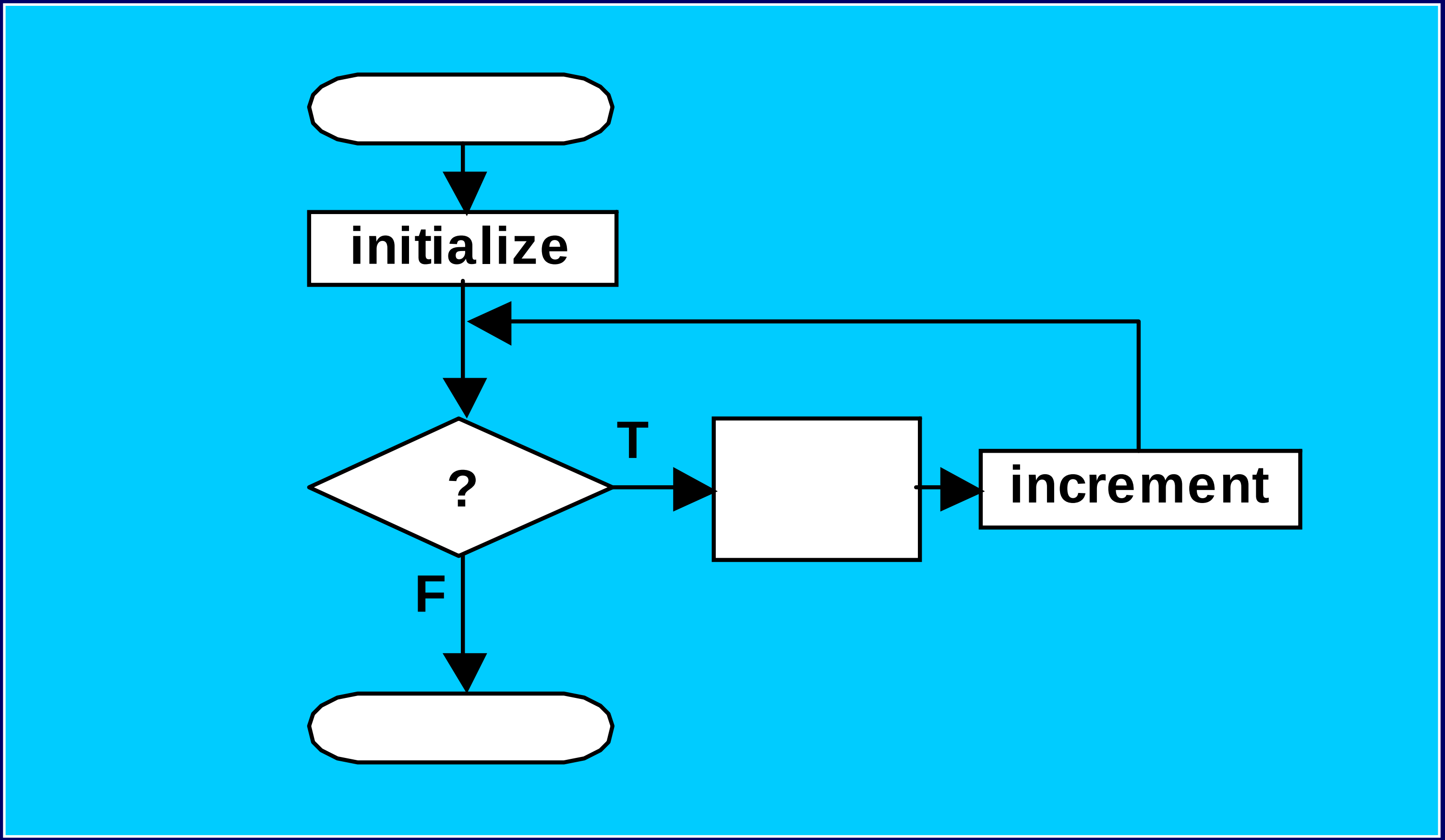
Operator precedence update

1. ( )
2. ++, --
3. *, /, %
4. +, -
5. =, +=, -=, *=, /=, %=

More iteration structures

- Remember: 3 ways to implement "loops" in Java
  - `while`, `for`, and `do/while`
- `while` loop is most basic
  - i.e., can always replace a `for` loop or `do/while` loop with `while` alone
    - But other forms are handy, and recommended sometimes
- Exam tip:
  - Translating a loop is a favorite exam problem

for Iteration Structure



`for` purpose: counter-controlled loops

- Recall the 3 steps with `while`:

```
int c = 0;  // initialize control variable
while (c < 10) {  // continuation condition
    System.out.println(c*c);
    c = c + 1;  // increment control variable
}
```

- One `for` does all:

```
      initialize       condition      increment
for (int c=0; c<10; c++)
   System.out.println(c*c);
```

`for` Notes

- Header *requires* three fields
  - i.e., always two "`;`" – but can leave one or more blank
- Manipulate control variable in the header
  - Manipulate other variables in loop body
  - Also best to NOT change control variable in body
- "Increment" not limited to `++`
  - Can decrement too: `for (int i=10; i>0; i--)`
  - Or use any amount: `for (int i=0; i<100; i+=5)`
- *Scope* of control variable limited to loop
  - Unless it is declared outside the loop

Applying `for` loops

- Find the sum of even integers from 2 through 20

```
int total = 0;
for (int num = 2; num <= 20; num += 2)
     total += num;
```

- Print digits (0 to 9) with spaces between

```
for (int i = 0; i < 10; i++)
     System.out.print(i + " ");
```

  // prints `"0 1 2 … 9 "`

- Use to do any operation a *fixed* number of times
  - e.g., `Interest.java` (Fig. 5.6, p. 167)