

Chapter 5

Comparison operators

- Operate on two numeric values; result in a boolean value.
- (Again, note operator precedences.)
- You know most of these from secondary school algebra.

Basic comparisons

Java	name
<	less than
>	greater than
==	equal to
<=	less than or equal to
>=	greater than or equal to
!=	not equal to

Example

```
public static boolean areEqual(int x, int y) {  
    return x == y;  
}
```

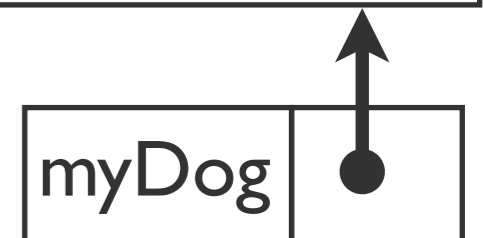
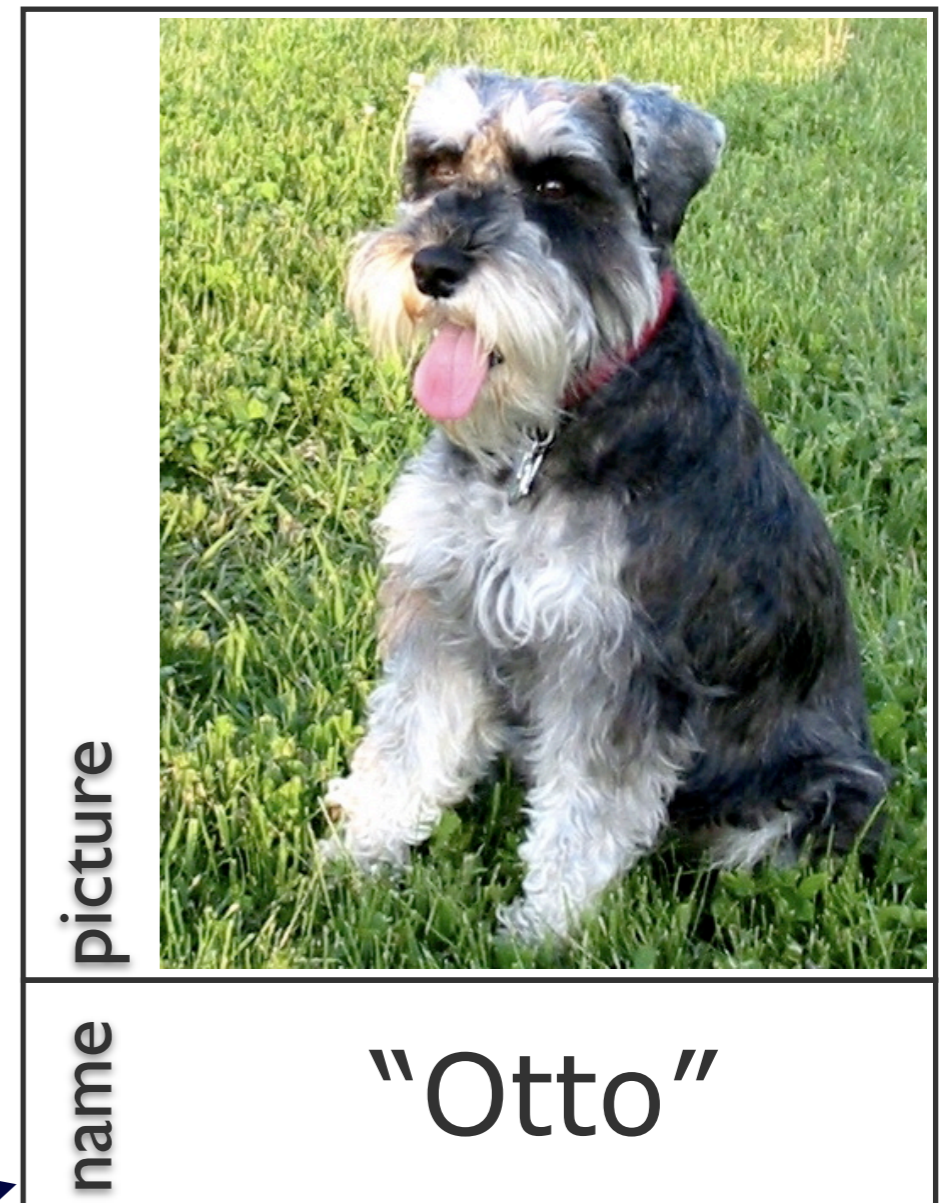
(Of course, a real program
would just use `x==y` to get the
same result!)

Comparing objects

- Three ways to compare objects
 - `==`
 - `.equals()`
 - `.compareTo()`
- Each is useful for different things!

Referential equality

- A test for objects
- `x == y` is true iff `x` and `y` refer to the exact same object
- In this case,
`otto == myDog`
is true
- Why is this kind of test useful?



Warning!

- Two different objects will never have referential equality, even if they have the same contents!
- Why is this the case?

```
public boolean refTest() {  
    Dog otto =  
        new Dog("Otto");  
    Dog myDog =  
        new Dog("Otto");  
    // always false!  
    return otto == myDog;  
}
```

Comparing contents

- All objects support a method called `equals`
- `equals` takes an `Object` and returns a `boolean`: `true` if the base object is "equal to" the parameter, and `false` otherwise
- What does "equal to" mean?

Comparing contents

- What does “equal to” mean?
- Answer: whatever we want it to!
- By default, “equal to” means referential equality; we can provide an equals method that does better than that.

Example

```
public class NumHolder {
    private int num;

    public NumHolder(int n) {
        num = n;
    }

    // ... other methods

    public boolean equals(Object o) {
        if (o instanceof NumHolder) {
            return ((NumHolder)o).num == this.num;
        }
        return false;
    }
}
```

Self-check questions

- Say you have the following block of code:

```
x = y;
```

```
return x == y;
```

Assuming it compiles, will it always return true?

- How about if that second line reads

```
return x.equals(y);
```

Comparing objects

- Many objects support a `compareTo` method.
- `x.compareTo(y)` returns
 - the `int` 0 if `x` and `y` have the same contents
 - some `int` `i`, $i < 0$ if `x` is “less than” `y`
 - some `int` `i`, $i > 0$ if `x` is “greater than” `y`

Logic basics

- We're often interested in combining boolean values in interesting ways
- Logical connectives provide us with various means to do this

Connectives

- AND: $x \text{ AND } y$ is true
iff x is true and y is true
- OR: $x \text{ OR } y$ is true
iff x is true or if y is true
- NOT: NOT x is true
iff x is false
- Note difference between logical OR
and vernacular or!

DeMorgan's Law

“A negated conjunction is equivalent to a disjunction of negations, and a negated disjunction is equivalent to a conjunction of negations.”

(Say that five times quickly!)

DeMorgan's Law

- $\text{NOT } (x \text{ AND } y) == (\text{NOT } x) \text{ OR } (\text{NOT } y)$
- $\text{NOT } (x \text{ OR } y) == (\text{NOT } x) \text{ AND } (\text{NOT } y)$

**There are many other useful
logical equivalences like
this; how can we find them?**

Truth tables provide a systematic, exhaustive way to explore the results of boolean functions.

x	y	$x \ \&\& \ y$	$x \ \ y$	$! \ y$
F	F	F	F	T
F	T	F	T	F
T	F	F	T	T
T	T	T	T	F

Self-check

- Write the following expressions in equivalent ways (p , q , and r are booleans, x and y are ints):
- $p = !(q \ \&\& \ r) \ \&\& \ !(\!q \ \&\& \ \!r)$
- $p = !(x < 4) \ \&\& \ !(y > 5)$
- $p = !((x < 2) \ \&\& \ (y \leq 4))$

Binary boolean connectives
don't always evaluate both
their arguments.

“I’m going to buy a cola if it costs less than ten cents and if I can climb Mt. Everest in under two weeks.”

If an and expression is *doomed to be false*, Java won't evaluate the right hand side.

```
false && someExpensiveMethod()
```

Likewise, if an or expression is guaranteed to be true...

```
true || someExpensiveMethod()
```

This property is called *short-circuited evaluation*.

**Boolean expressions can be
used to make decisions in
your programs.**

Java programs can make decisions in a few ways: with an expression, or with one of several statements.

The *ternary operator* is an unusual 3-valued operator.

$exp_{test} ? exp_{cons} : exp_{alt}$

if exp_{test} is true

if exp_{test} is false

```
int x, y;
```

```
// ...
```

```
System.out.print("The larger number is: ");  
System.out.println((x > y) ? x : y);
```

The ternary operator is also called the “conditional operator” and the “selection operator.”

The *if statement* executes a statement if its condition is true

if (exp_{test}) $stmt_{cons}$

if exp_{test} is true

What is a statement?

A statement is either

- a single line of Java code terminated by a semicolon, or
- a sequence of statements enclosed in curly braces

```
x++;
```

```
System.out.println("foo");
```

```
{
```

```
    int y = 4;
```

```
    System.out.println(y);
```

```
}
```

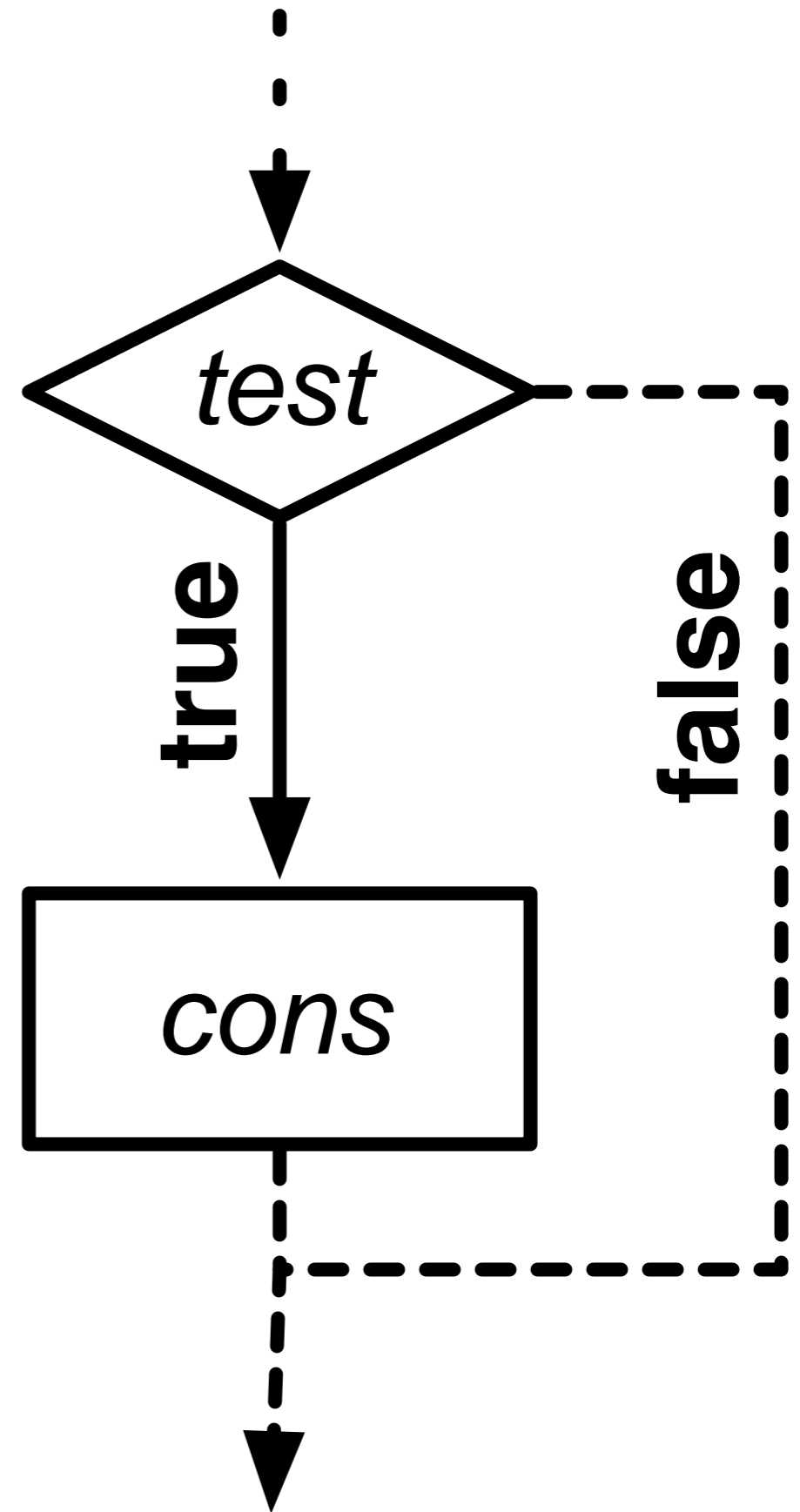
```
{  
  { foo(); }  
  System.out.println(5);  
}
```

The *if statement* executes a statement if its condition is true

if (exp_{test}) $stmt_{cons}$

if exp_{test} is true

if (*exp_{test}*)
stmt_{cons}



The *if statement* executes a statement if its condition is true

if (*exp_{test}*)

stmt_{cons}

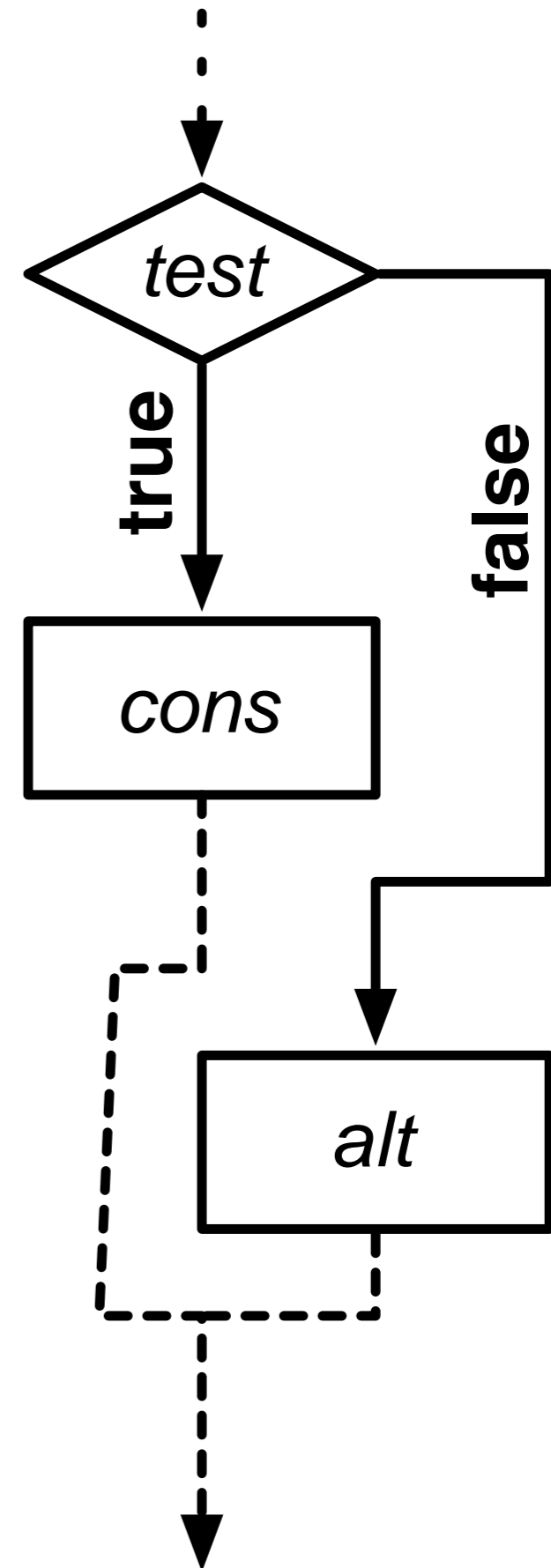
else

stmt_{alt}

if *exp_{test}* is true

if *exp_{test}* is false

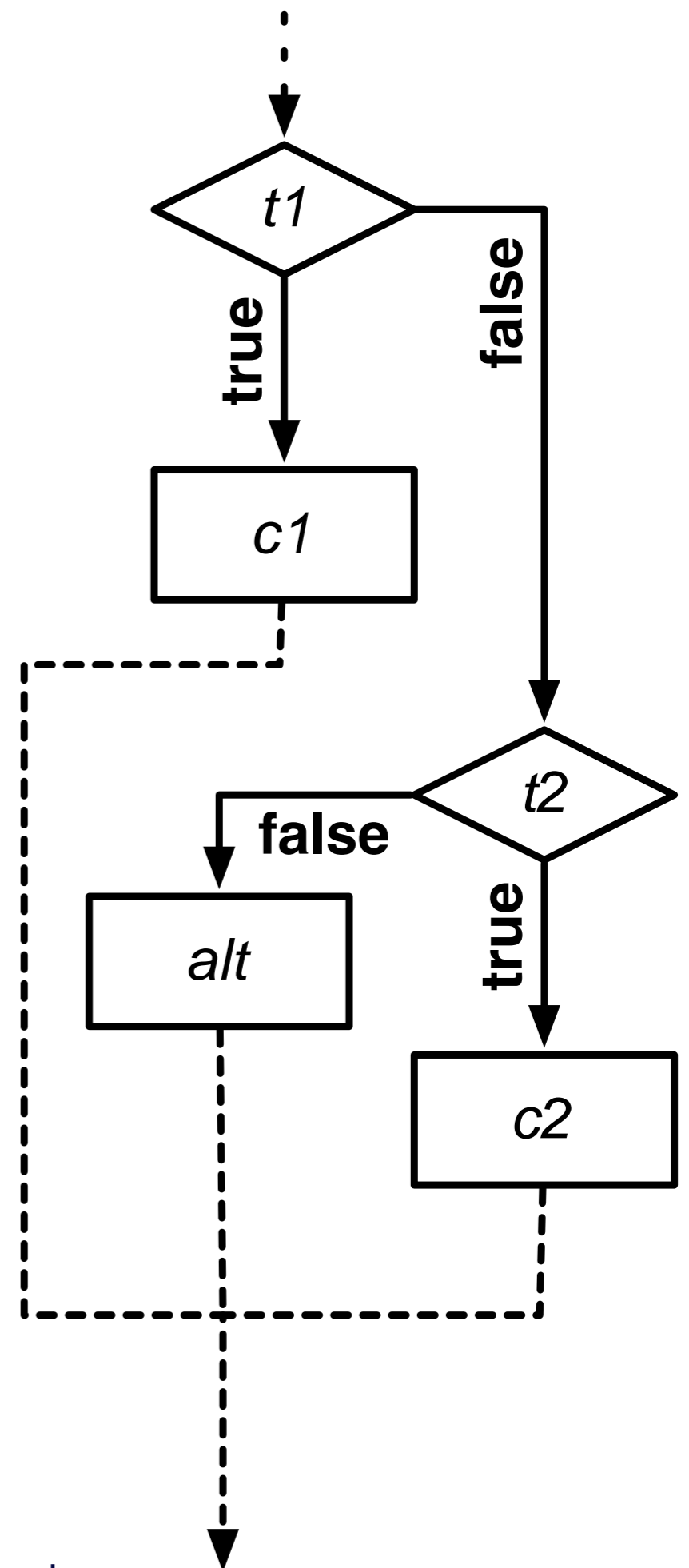
if (exp_{test})
 $stmt_{cons}$
else
 $stmt_{alt}$



The *if statement* executes a block of code if its condition is true

```
if ( $exp_{test1}$ ) stmtcons1  
else if ( $exp_{test2}$ ) stmtcons2  
else if ( $exp_{test3}$ ) stmtcons3  
else stmtalt
```

if (exp_{t1})
stmt_{c1}
else if (exp_{t2})
stmt_{c2}
else
stmt_{alt}



switch is useful when you need to execute one (or more) of several statements depending on the value of an expression

```
switch (exp) {  
    case  $v_1$ : stmt1  
    case  $v_2$ : stmt2  
    case  $v_3$ : stmt3  
    default: stmtd  
}
```

If *exp* has value v_1 , then the switch statement will start executing at *stmt*₁ (and so on).

Where does execution stop?

Think of `switch` statements
as *jumping into the middle*
of a block of code.

(You can escape with `break`.)

Self-check

Translate the following code into equivalent code that uses an if statement:

```
switch (x % 2) {  
    case 0:    System.out.println("E");  
    case 1:    System.out.println("O");  
    default:   System.out.println("D");  
}
```

Review: switch

- Are there any restrictions on the type of the expression?
- Choices in `switch` statements aren't mutually exclusive; why not?
- What is the term for the behavior that enables `switch` statements to execute more than one statement?