Set/get, Subclasses, Loops

18 January 2008
CMPT166
Dr. Sean Ho
Trinity Western University

TRINITY WESTERN UNIVERSITY

Review of last time

- Declaring classes in OO-M2, C++, Java
- Declaring and instantiating objects in M2, C++, Java
- Access control
  - Header vs. implementation files
  - public/private/protected
- Java packages, jar

Review: operator precedence

- In order from most tightly bound first:
  - Parentheses: ()
  - Unary postfix (r to l): x++, x--
  - Unary prefix (r to l): ++x, --x, +x, -x, (type) x
  - Multiplicative: *, /, %
  - Additive: +, -
  - Relational: <, >, <=, >=
  - Equality: ==, !=,
  - Conditional (r to l): ?:
  - Assignment (r to l): =, +=, -=, *=, /=, %=, etc.

Java primitive types

- boolean (1 byte): true, false
- char (2 bytes): Unicode, '\u0000' to '\uFFFF'
- byte (1 byte): -128 to +127
- short (2 bytes): -32768 to +32767
- int (4 bytes): -2^{31} to $+2^{31}-1$
- long (8 bytes): -2^{65} to $+2^{65}-1$
- float (4 bytes): +/- 1.40129846432481707e–45 to 3.40282346638528860e+38
- double (8 bytes): +/- 4.94065645841246544e–324 to 1.79769313486231570e+308

public/private keywords

- So far most of our classes/attributes/methods have been declared public
- The private keyword specifies that only methods within this class can access this entity:

```
class Student {
    private String name;
}
Student s1 = Student();
s1.name;        // error!
```

- This is for information hiding: prevent others from directly accessing/modifying an entity.

Set/get methods

- A common idiom is to declare instance variables private but provide public set/get methods:

```
class Student {
    private String name;
    public String getName() { return name; }
    public setName(String n) { name = n; }
}
```

- Advantages of set/get over just declaring public?
  - Control access to the instance variable
    - Can add error checking
  - Hides underlying storage type of variable
    - Can upgrade to different data structure later

Subclasses, instances, attributes

- Recall classes are user-defined container types
- A subclass inherits attributes and methods from the superclass
- Subclasses should be seen as specializations of the superclass: “A is a kind of B”
- Instances should be seen as examples of a class: “A is a B”
- Attributes should be seen as components or parts of a class: “A has a B”

Example

- class Mammal { Heart h; }
- class Dog extends Mammal { void bark(); }
- class Cat extends Mammal { void meow(); }
- Dog fido = new Dog();
- Cat smokey = new Cat();

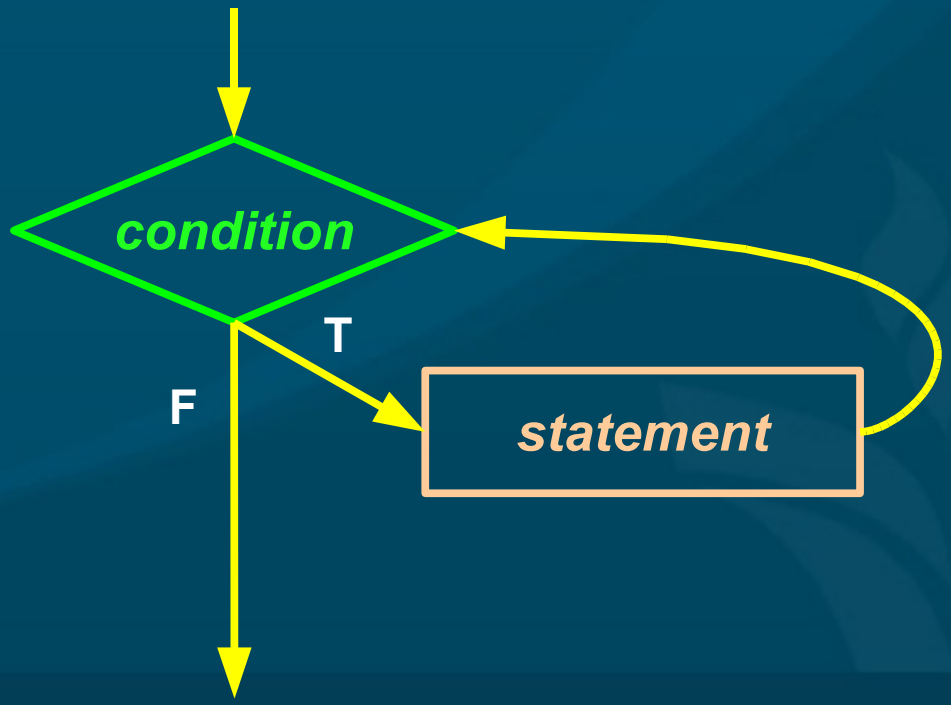
- "A Dog is a kind of Mammal."
- "fido is a Dog."
- "fido is a Mammal."
- "fido has a Heart."
- "smokey can meow()."

Interfaces

- An interface is a set of methods that a class implements
  - public interface Speaker { public void speak(); }
  - class Dog extends Mammal implements Speaker {
    - void bark() { System.out.println("Woof!"); }
    - public void speak() { bark(); }
  - }
  - class Cat extends Mammal implements Speaker {
    - void meow() { System.out.println("Meow!"); }
    - public void speak() { meow(); }
  - }
- Compare fido.speak() with smokey.speak()

While loops

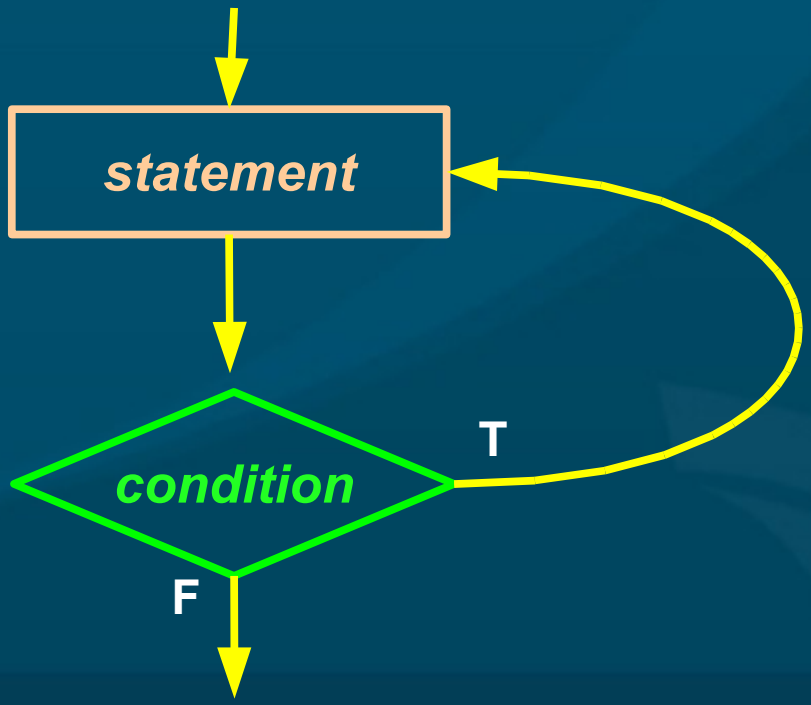
- while (*condition*) *statement*;
- As usual, statement can be a {} block
- *condition* evaluates to a boolean
- Top-of-loop testing



do/while loops

- do *statement* while (*condition*);

- As usual, statement can be a {} block
- *condition* evaluates to a boolean
- Bottom-of-loop testing



For loops as while loops

- Pretty much every for loop:
  - for (*init*; *condition*; *increment*) *statement*;
- ... can be expressed as an equivalent while loop:
  - *init*;
  - while (*condition*) {
    - *statement*;
    - *increment*;
  - }

break/continue

- Use break to terminate a loop early:
  - for (i=0; i<10; i++) {
    - if (i==5) break; // quit at 5
  - }
- Use continue to skip to the next iteration of the loop:
  - for (i=0; i<10; i++) {
    - if (i==5) continue; // don't print 5
    - System.out.print(i);
  - }

Switch statement

- switch (*expression*) {
  - case *val1*: *statement; ...;* break*;*
  - case *val2*: *statement; ...;* break*;*
  - *...*
  - default: *statement; ...;*
- }

- Similar to a nested if/else structure
  - But *expression* is only evaluated once
- If omit a break, execution continues to next case:
  - case *val1:*
  - case *val2*: *statement; ...;* break*;*

Labeled blocks

- Blocks can be named
- break/continue can specify a name:
  - Go to start/end of named block

```
main: {
    for (row=0; row<n_rows; row++) {
        for (col=0; col<n_cols; col++) {
            if (row+col == 12) break main;
        }
    }
}
```