Control Statements, Part 2

October 6, 2003

Compound boolean expressions

- Sometimes we want to evaluate more complex expressions
  - if age is greater than 30 but less than 65, print out “You are an old geezer”
  - if y is greater than 50 or equal to 25, multiply y by 2
  - if a, b, and c are all less than zero, print out “All three numbers are negative”

Boolean operators

- Format:
  \(<boolean expression> <boolean operator> <boolean expression>\)

- Three operators:
  - && and binary \(p \& \& q\)
  - || or binary \(p \| q\)
  - ! not unary \(!p\)

Using boolean operators

- &&: True if both operands are true
- ||: True if either operand is true
- !: Take the opposite of the operand
Truth table

| p  | q  | p&&q | p||q | !p | !q |
|----|----|------|------|----|----|
| true | true | true  | true  | false | false |
| true | false | false | true  | false | true |
| false | true | false | true  | true  | false |
| false | false | false | false | true  | true |

Order of operations (revised again)

1. Parenthesis
2. Unary operators (-, !)
3. *, /, %
4. +, -
5. Relational operators (>, >=, <, <=)
6. Equality operators (==, !=)
7. Boolean AND (&&)
8. Boolean OR (||)
9. Assignment (=)

Examples

```java
int a=10, b=13, c=20, d=7;
boolean test1 = a < b && b < c || d > c;
boolean test2 = c % b <= c % d || !(a >= b);
```

• Q: What are the values of test1 and test2?

Answers

```java
int a=10, b=13, c=20, d=7;
boolean test1 = a < b && b < c || d > c;
boolean test2 = c % b <= c % d || !(a >= b);
```

• A:
  - test1 = true
  - test2 = true
DeMorgan's Law

- Describes how to negate a logical statement without evaluating the statement first
- Idea:
  - negate both of the operands
  - change the operator to the opposite operator
- Example:
  - !(p && q) becomes !p || !q
  - !(p || q) becomes !p && !q
- The statements are logically equivalent

Truth table

| p   | q   | !(p&&q) | !p||!q | !(p||q) | !p&&!q |
|-----|-----|---------|--------|--------|-------|
| true| true| false   | false  | false  | true  |
| true| false| true    | true   | true   | true  |
| false| true| true    | true   | false  | false |
| false| false| true    | true   | true   | true  |

Compound boolean expressions

- if age is greater than 30 but less than 65, print out “You are an old geezer”
  ```java
  if (age > 30 && age < 65) {
    System.out.println("You are an old geezer");
  }
  ```
- if y is greater than 50 or not equal to 25, multiply y by 2
  ```java
  if (y > 50 || y != 25) {
    y = y*2;
  }
  ```

Compound boolean expressions

- if a, b, and c are all less than zero, print out “All three numbers are negative”
  ```java
  if (a < 0 && b < 0 && c < 0) {
    System.out.println("All three numbers are negative");
  }
  ```
Evaluating complex boolean expressions

- Q: Does Java evaluate the entire expression, every time?
- A: No!
  - example: if (a < 0 && b < 0 && c < 0)
    - a = 2: first part of the expression is false, therefore the whole expression is false
  - example: if (age > 30 || age < 65)
    - age = 35: first part of the expression is true, therefore the whole expression is true
- Once you know what the expression evaluates to, stop evaluating. --> *short-circuit evaluation*

Another usage for boolean variables

- Boolean variables are often used as flags
  - keep track of some option the user set, or indicate what state the program is in
- Typically, variables used as flags start with the word *is*
  - isValid
  - isAnInteger
  - isRunning

Example

```java
int count = 0, sum = 0, limit = 100;
boolean isAtLimit = false;
... // calculate the sum, somehow
if (sum >= limit) {
    isAtLimit = true;
}
... if (isAtLimit) {
    System.out.println("Limit reached or exceeded.");
}
```

Nested if statements

- Sometimes, our conditions will be even more complex, and depend on previous conditions
  - i.e., if <statement1> is true, then check if <statement2> is true, and if so, do something
  - Checking for valid interest rate input in the Bank program is an example of this
- Use *nested if* statements to do this
  - just place one if statement in the statement block ({...}) of another if statement
Interest rate (in pseudocode)

Get interest rate from the user.
If interest rate is less than zero or greater than 100, print an error message.
Else, if interest rate is greater than 1, divide interest rate by 100.0; calculate the new balance.
Print out the account’s balance.

Interest rate (in Java code)

SavingsCustomer c1;
...
... // initialize savings customer
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
System.out.print("Enter interest rate: ");
double rate = Double.parseDouble(br.readLine());
if (rate > 100 || rate < 0) {
    System.out.println("Invalid interest rate entered.");
} else {
    if (rate > 1) {
        rate = rate/100.0;
    }
    c1.acct.calculateInterest(rate);
}

Increment and decrement operators

• Shorthand for saying “add one to this variable” and “subtract one from this variable”
• Example:
  int i = 10;
i++; // equivalent to i = i + 1;
System.out.println(i); // i = 11
i--; // equivalent to i = i - 1;
System.out.println(i); // i = 10;

Comparing objects

• When we compare two numbers, we use the equality (==) operator
  – if (a == b)
  – if (x == 56)
  – if (y == 28.3)
• Q: What if we need to compare two non-numbers?
  – Strings, CheckingAccounts, Books, etc.
Example: 2 CheckingAccounts

CheckingAccount acct1 = new CheckingAccount(1000, 123456);
CheckingAccount acct2 = new CheckingAccount(1000, 123456);
if (acct1 == acct2) {
    System.out.println("Accounts 1 and 2 are equal");
} else {
    System.out.println("Accounts 1 and 2 are not equal");
}

What is the result of the previous code?

- Initial thought: The accounts are equal
  - same balance
  - same account number
- In reality: The accounts are not equal!
  - equality for objects means “at the same location in memory”
  - to compare objects, we need to write comparison methods!

Example: CheckingAccount with comparison

public class CheckingAccount {
    double balance;
    int number;
    ...
    public boolean equals(CheckingAccount chk) {
        if (chk.balance == balance && chk.number == number) {
            return true
        } else {
            return false
        }
    }
    ...
}

Comparing 2 CheckingAccounts

CheckingAccount acct1 = new CheckingAccount(1000, 123456);
CheckingAccount acct2 = new CheckingAccount(1000, 123456);
if (acct1.equals(acct2)) {
    System.out.println("Accounts 1 and 2 are equal");
} else {
    System.out.println("Accounts 1 and 2 are not equal");
}
Comparing Strings

- The String class also has an equals() method
  - also: equalsIgnoreCase()

Example:
```java
String s1 = "this is a string";
String s2 = "This is a STRING";
if (s1.equals(s2)) {
    System.out.println("match");
} else if (s1.equalsIgnoreCase(s2)) {
    System.out.println("match, ignoring case");
} else {
    System.out.println("no match");
}
```