equals, else, switch, and Boolean Math

CSC 116 – Section 002
February 9, 2004

Comparing Objects

• When two values are compared we are looking at contents.
• Primitive data types (int, double) have the actual value of the variable as the contents of the variable
• Objects have the address in memory where the data for the object is stored as the contents of the variable
Comparing Objects (2)

• Using == to compare objects compares the addresses where the objects are stored
• Using .equals compares the data of the object

The Object Class

• The Object class is the top-most class in the Java class hierarchy.
• Every class that you make has the Object class as a superclass. This means that all classes you make automatically have the methods of the Object Class.
• One of these methods is .equals
.equals Method

- The .equals method in the Object class only checks to see if Object x == Object y.
- It only checks to see if the addresses where the two objects are the same
- If you wish to determine equality of objects in another way, you may override the .equals Method
- Most classes do this (Ex: Rectangle)

.equals Method (2)

- The .equals Method looks similar to this:
  public boolean equals(<Class Name> <var>) {
      return true;
  }
Boolean Math

- Boolean Operator: “takes boolean values as its operands and returns a boolean value” [Wu]

<table>
<thead>
<tr>
<th>Boolean Operator</th>
<th>Java Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>&amp;&amp;</td>
</tr>
<tr>
<td>OR</td>
<td></td>
</tr>
<tr>
<td>NOT</td>
<td>!</td>
</tr>
</tbody>
</table>

Boolean Math (2)

| P   | Q   | P&&Q | P||Q | !P   |
|-----|-----|------|------|------|
| False | False | False | False | True |
| False | True  | False | True  | True |
| True  | False | False | True  | False |
| True  | True  | True  | True  | False |
Short-Circuit Evaluation

- OR
  - If the left operand evaluates true, the right operand is not evaluated
  - If the left operand is true, the whole expression is true
- AND
  - If the left operand evaluates false, the right operand is not evaluated
  - If the left operand is false, the whole expression is false

Example: Boolean Math (&&)

```java
public void showTestScoreBooleanMath(int score) {
    if(score <= 100 && score >= 90)
        System.out.println("A");
    if(score < 90 && score >= 80)
        System.out.println("B");
    if(score < 80 && score >= 70)
        System.out.println("C");
    if(score < 70 && score >= 60)
        System.out.println("D");
    if(score < 60)
        System.out.println("FAIL - no one should do this!");
}
```
Example: Boolean Math (||)

```java
public void passOrFail(String lg) {
    if(lg.equals("A") || lg.equals("B") || lg.equals("C")) {
        System.out.println("PASS");
    }
    if(lg.equals("D") || lg.equals("F")) {
        System.out.println("FAIL");
    }
}
```

Example: Boolean Math (!)

```java
public boolean failClass(String grade) {
    if(!grade.equals("F"))
        return false;
    return true;
}
```
DeMorgan’s Law

- Used to distribute a NOT to AND or OR expressions
- !(P && Q) = !P || !Q
- !(P || Q) = !P && !Q

Boolean Math (3)

- !(x > y) = (x <= y)
- !(x < y) = (x >= y)
- !(x >= y) = (x < y)
- !(x <= y) = (x > y)
Multiple if-else Statements

- If you want to have multiple decision points in a program you can use multiple if-else statements

Example: Multiple if-else Stmts.

```java
public void showTestScoreMultipleIfs(int score) {
    if (score >= 90)
        System.out.println("A");
    else if (score >= 80)
        System.out.println("B");
    else if (score >= 70)
        System.out.println("C");
    else if (score >= 60)
        System.out.println("D");
    else
        System.out.println("FAIL - no one should do this!");
}
```
Nested if Statements

• An if statement is nested if there is another if statement in the then or else blocks
• Be careful to make sure that ifs and elses match up properly

Example: Nested if Statements

```java
public void showTestScoreNestedIfs(int score) {
    if (score >= 60) {
        if (score >= 70) {
            if (score >= 80) {
                if (score >= 90)
                    System.out.println("A");
                else
                    System.out.println("B");
            } else
                System.out.println("C");
        } else
            System.out.println("D");
    } else {
        System.out.println("FAIL - no one should do this!");
    }
}
```
switch Statements

- Also used to make decisions in the program
- Change control flow
- Syntax:
  switch (<arithmetic expression>) {
    <case label 1> : <case body 1>
    ...
    <case label n> : <case body n>
  }
  //Where case label i has the form: case <constant>
  //or default

Example: switch Statement

```java
public void showDigit(int digit) {
    switch (digit) {
        case 1: System.out.println("one");
            break;
        case 2: System.out.println("two");
            break;
        case 3: System.out.println("three");
            break;
        default: System.out.println("unknown");
            break;
    }
}
```

switch Statements (2)

- break is used to break out of the switch statement and skip execution on the remaining statements
- Without a break statement, all other statements will be executed
- Example:
  ```java
  int selection = 1;
  switch (selection) {
      case 0: System.out.println(0);
      case 1: System.out.println(1);
      case 2: System.out.println(2);
  }
  ```

Example 2: switch Statement

```java
public void showPlusMinusScore(int score) {
    switch (score) {
        case 100:
        case 99:
        case 98: System.out.println("A+");
            break;
        case 97:
        case 96:
        case 95:
        case 94:
        case 93: System.out.println("A");
            break;
        default: System.out.println("unknown");
            break;
    }
```
References

- Jason Schwarz’s Lecture 9 & 10 slides: http://courses.ncsu.edu/csc116/
- Java API: Object, Rectangle: http://java.sun.com/j2se/1.4.2/docs/api/