Boolean Expressions
Looping
Lecture 14

Borrowing from slides by Alan Hu, Kurt Eiselt, Paul Carter, and Tamara Munzner

Reading Assignments
- Reading for this week: looping/iterations
  - Edition 2: Ch. 6.1-6.5
  - Edition 3: Ch. 7.1-7.5

Objectives for Today
- Understand how to rewrite and simplify logical expressions
- Understand the concept of looping and how to write simple while loops

Recap: Java’s if statement with else
- Syntax:
  ```java
  if ( condition ) statement;
  else statement;
  ```
- Example:
  ```java
  if (age < 19)
      System.out.println("You may not buy alcohol.");
  else
      System.out.println("Buy beer!");
  System.out.println("This lecture sponsored by Molson");
  ```

Recap: Boolean Expressions
- Boolean expression: test which returns either true or false when evaluated
- Consists of operands and operators, like arithmetic expression
  - but operators only return true or false when applied to operands
- Two different kinds of operators
  - relational
  - logical

Recap: Relational Operators
- Compares two values (operands)
- Operators (See Appendix F.)
  - == equal
    - returns true if they are equal, false otherwise
    - note: do not confuse this with =
  - != not equal
    - returns true if they are not equal, false otherwise
  - < less than
  - <= less than or equal to
  - > greater than
  - >= greater than or equal to
Recap: Equality Example

```java
int a = 3;
int b = 6;
int c = 10;
if (a == b)
    System.out.println("these two values are equal");
if ((b - a) == a)
    System.out.println("that was a silly example");
if (a != b)
    System.out.println("nope!");
```

Note we can use arithmetic operator inside boolean expression

Recap: Logical Operators

- Way to combine results from relational operators into single test
- AND, OR, and NOT
  - in terms from math or philosophy class
  - Operators
    - && logical AND
    - || logical OR
    - ! logical NOT

Recap: Boolean Tables

- **AND:**
  - a b a && b
  - false false false
  - false true false
  - true false false
  - true true true

- **OR:**
  - a b a || b
  - false false false
  - false true true
  - true false true
  - true true true

- **NOT:**
  - a ! a
  - false true
  - true false

Recap: Precedence and Associativity of Logical Operators

- Logical operators have different precedence:
  - ! has highest precedence
  - && has 2nd highest precedence
  - || has lowest precedence
- All logical operators are left associative, i.e.
  - a && b && c is identical to (a && b) && c

Recap: Equivalence of Complex Boolean Expressions

- Using boolean tables, we can transform complex expressions, and possibly simplify them
- For example:
  - Can we simplify !(a && b)?

- Answer:
  - Yes, equivalent to (a || b)

Recap: Equivalence of Complex Boolean Expressions

- Using boolean tables, we can transform complex expressions, and possibly simplify them
- For example:
  - Can we simplify !(a && b)?

- Answer:
  - Yes, equivalent to (a || b)
Equivalence of Complex Boolean Expressions

- **In general:**
  - \(!(!a)\) is identical to \(a\)
  - \(! (a \&\& b)\) is identical to \((!a) \mid (!b)\)
  - \(! (a \mid\mid b)\) is identical to \((!a) \&\& (!b)\)
  - \(a \&\& (b \mid\mid c)\) is identical to \((a \&\& b) \mid\mid (a \&\& c)\)
  - \(a \mid\mid (b \&\& c)\) is identical to \(a \mid\mid b \&\& c\)
  - but is not identical to \((a \mid\mid b) \&\& (a \mid\mid c)\)

De Morgan’s Laws

- **Therefore:**
  - \(!\) is a bit like the unary
  - \(&\&\) is a bit like *
  - \(\mid\mid\) is a bit like +

Equivalence of Complex Boolean Expressions

- **In general:**
  - \(!(!a)\) is identical to \(a\)
  - \(! (a \&\& b)\) is identical to \((!a) \mid (!b)\)
  - \(! (a \mid\mid b)\) is identical to \((!a) \&\& (!b)\)
  - \(a \&\& (b \mid\mid c)\) is identical to \((a \&\& b) \mid\mid (a \&\& c)\)
  - \(a \mid\mid (b \&\& c)\) is identical to \(a \mid\mid b \&\& c\)
  - but is not identical to \((a \mid\mid b) \&\& (a \mid\mid c)\)

Loops

- **One more key idea that makes computers powerful:**
  - Remembering things: variables
  - Calculating things: expressions
  - Making decisions: **if**
  - Repeating things: **while**
  - **Loops** allow repetitive operations in programs
  - aka *iteration, repetition*

Climbing Stairs

- Am I at the top of the stairs?
- No.
- Climb up one step.
while Statement

```
while ( boolean expression )
  body
```

- Simplest form of loop in Java
- Body of loop can be
  - single statement
  - whole block of many statements in curly braces
- Meaning is similar to English:
  "While you’re still hungry, have some more fries."
  ```
  while ( still hungry ) {
    eat fries;
  }
```
if Versus while Statements

How if statement works

boolean expression

true
false

statement

how while statement works

boolean expression

true
false

statement

• How can boolean change from false to true?

These diagrams called flowcharts

Using while Statements

public class WhileDemo
{
    public static void main (String[] args)
    {
        int limit = 3;
        int counter = 1;

        while (counter <= limit)
        {
            System.out.println("The square of "+ counter + " is "+ (counter * counter));
            counter = counter + 1;
        }
        System.out.println("End of demonstration");
    }
}

• while statement

Using while Statements

public class WhileDemo
{
    public static void main (String[] args)
    {
        int limit = 3;
        int counter = 1;

        while (counter <= limit)
        {
            System.out.println("The square of "+ counter + " is "+ (counter * counter));
            counter = counter + 1;
        }
        System.out.println("End of demonstration");
    }
}

• boolean expression
**Using while Statements**

```java
public class WhileDemo {
    public static void main(String[] args) {
        int limit = 3;
        int counter = 1;
        while (counter <= limit) {
            System.out.println("The square of " + counter + " is " + (counter * counter));
            counter = counter + 1;
        }
        System.out.println("End of demonstration");
    }
}
```

- `while` statement body

- trace what happens when execute

```
limit 3  counter 1
```

- `while` statement body

- statement after `while`
  - control flow resumes here when boolean is false

```
limit 3  counter 1
```

- `while` statement body

- Is counter <= limit? yes
Using while Statements

public class WhileDemo
{
    public static void main (String[] args)
    {
        int limit = 3;
        int counter = 1;
        while (counter <= limit)
        {
            System.out.println("The square of " + counter + " is " + (counter * counter));
            counter = counter + 1;
        }
        System.out.println("End of demonstration");
    }
}

limit 3 counter 1 Is counter <= limit? yes
"The square of 1 is 1" printed on monitor

Using while Statements

public class WhileDemo
{
    public static void main (String[] args)
    {
        int limit = 3;
        int counter = 1;
        while (counter <= limit)
        {
            System.out.println("The square of " + counter + " is " + (counter * counter));
            counter = counter + 1;
        }
        System.out.println("End of demonstration");
    }
}

limit 3 counter 2 Is counter <= limit? yes
"The square of 2 is 4" printed on monitor

Using while Statements

public class WhileDemo
{
    public static void main (String[] args)
    {
        int limit = 3;
        int counter = 1;
        while (counter <= limit)
        {
            System.out.println("The square of " + counter + " is " + (counter * counter));
            counter = counter + 1;
        }
        System.out.println("End of demonstration");
    }
}

limit 3 counter 3 Is counter <= limit? yes
Using \textit{while} Statements

```java
public class WhileDemo {
    public static void main (String[] args) {
        int limit = 3;
        int counter = 1;
        while (counter <= limit) {
            System.out.println("The square of "+ counter + " is "+ (counter * counter));
            counter = counter + 1;
        }
        System.out.println("End of demonstration");
    }
}
```

Limit: 3, Counter: 1
Is counter <= limit? yes
"The square of 3 is 9" printed on monitor

Limit: 3, Counter: 4
Is counter <= limit? NO!
"End of demonstration" printed on monitor

What if boolean expression is already false?

```java
public class WhileDemo {
    public static void main (String[] args) {
        int limit = 3;
        int counter = 1;
        while (counter >= limit) {
            System.out.println("The square of "+ counter + " is "+ (counter * counter));
            counter = counter + 1;
        }
        System.out.println("End of demonstration");
    }
}
```

Limit: 3, Counter: 4
Is counter <= limit? NO!
Body of loop never executed

- change \textit{termination condition}
What if boolean expression is always true?

public class WhileDemo
{
    public static void main(String[] args)
    {
        int limit = 3;
        int counter = 1;
        while (counter >= counter)
        {
            System.out.println("The square of "+counter +
            " is " +(counter * counter));
            counter = counter + 1;
        }
        System.out.println("End of demonstration");
    }
}

- change termination condition
- always true

Infinite Loops

public class WhileDemo
{
    public static void main(String[] args)
    {
        int limit = 3;
        int counter = 1;
        while (counter >= counter)
        {
            System.out.println("The square of "+counter +
            " is " +(counter * counter));
            counter = counter + 1;
        }
        System.out.println("End of demonstration");
    }
}

- if termination condition always true, loop never ends
  - infinite loop goes forever

Infinite Loops

public class WhileDemo
{
    public static void main(String[] args)
    {
        int limit = 3;
        int counter = 1;
        while (counter >= limit)
        {
            System.out.println("The square of "+counter +
            " is " +(counter * counter));
            counter = counter + 2;
        }
        System.out.println("End of demonstration");
    }
}

- process gets closer to termination condition
- but never satisfies condition, keeps going past it

Quasi-Infinite Loops

public class WhileDemo
{
    public static void main(String[] args)
    {
        int limit = 3;
        int counter = 1;
        while (counter <= limit)
        {
            System.out.println("The square of "+counter +
            " is " +(counter * counter));
            counter = counter - 1;
        }
        System.out.println("End of demonstration");
    }
}

- good termination condition
- but process never gets closer to condition

Example: Loan Amortization

- How about computing your account balance after some number of years?
  
  double balance = initBalance;
  int years = 0;
  while (years < term) {
      intAccrued = balance*intRate/100;
      balance += (intAccrued - payment);
      years++;
  }

  - loan amortization