CS 120 Lecture 16

Java Loops
(Java: An Eventful Approach, Ch 7 and 13),

6 November 2012

Slides Credit: Bruce, Danyluk and Murtagh

Programs Involving Repetition

- Drawing Grass

<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>----------------</td>
</tr>
</tbody>
</table>

- Drawing grids
- Printing marks on a ruler
- Repeatedly rolling dice in craps game
Recognizing a Pattern

```java
public void begin() {
    // add the blades of grass
    new Line(0, GRASS_TOP, 0, GROUND_LINE, canvas);
    new Line(4, GRASS_TOP, 4, GROUND_LINE, canvas);
    new Line(8, GRASS_TOP, 8, GROUND_LINE, canvas);
    new Line(12, GRASS_TOP, 12, GROUND_LINE, canvas);
    new Line(16, GRASS_TOP, 16, GROUND_LINE, canvas);
    ...
}
```

Making a Pattern Explicit

```java
// add the blades of grass
bladePosition = 0;
new Line(bladePosition, GRASS_TOP, bladePosition, GROUND_LINE, canvas);
bladePosition = bladePosition + GRASS_SPACING;
new Line(bladePosition, GRASS_TOP, bladePosition, GROUND_LINE, canvas);
bladePosition = bladePosition + GRASS_SPACING;
new Line(bladePosition, GRASS_TOP, bladePosition, GROUND_LINE, canvas);
bladePosition = bladePosition + GRASS_SPACING;
new Line(bladePosition, GRASS_TOP, bladePosition, GROUND_LINE, canvas);
bladePosition = bladePosition + GRASS_SPACING;
...
```
Eliminating Code Repetition

private int bladePosition=0;
public void onMouseClick(Location point) {
    // grow a blade of grass with each mouse click
    if (bladePosition < canvas.getWidth()) {
        new Line(bladePosition, GRASS_TOP,
                 bladePosition, GROUND_LINE,
                 canvas);
        bladePosition = bladePosition + GRASS_SPACING;
    }
}

• First approach tedious for programmer
• Second approach tedious for user

The **while** Loop
(Indefinite loop)

• A control construct for specifying repetition
• General Structure:

```
while (condition) {
    // Statements to be repeated
}
```
Drawing Grass with `while`

```java
public void begin() {
    // add the blades of grass
    double bladePosition = 0;
    while (bladePosition < canvas.getWidth()) {
        new Line(bladePosition, GRASS_TOP,
                 bladePosition, GROUND_LINE,
                 canvas);
        bladePosition = bladePosition +
                       GRASS_SPACING;
    }
}
```

Drawing a Grid

```java
while (verticalCorner.getX() < canvas.getWidth() || horizontalCorner.getY() < canvas.getHeight()) {
    new FilledRect(verticalCorner, 5, canvas.getHeight(), canvas);
    new FilledRect(horizonalCorner, canvas.getWidth(), 5, canvas);
    verticalCorner.translate(10, 0);
    horizontalCorner.translate(0, 10);
}
```
The Counting **while** loop

• Counting up
  
  ```java
  int i=initialValue;
  while(i<endValue){
    //statements to be repeated
    i++;
  }
  ```

Drawing a Number of Bricks

• Might want to draw exactly 10 bricks

  ```java
  private static final int BRICKS_TOTAL=10;
  int brickPosition=0;
  int brickCount=0;
  while ( brickCount < BRICKS_TOTAL ) {
    new FilledRect(brickPosition, BRICK_TOP,
                   BRICK_WIDTH, BRICK_HEIGHT,
                   canvas);
    brickPosition = brickPosition + BRICK_WIDTH + BRICK_SPACING;
    brickCount++;
  }
  ```
Suppose we want to draw a brick wall

```java
int level = 0;
while (level < WALL_HEIGHT) {
    ...//draw one row of bricks
    brickY = brickY + BRICK_HEIGHT;
    level ++;
}
```
• Already know how to draw a row of bricks

• Nest 1 while loop inside another
  while (condition1) {
    //moves to draw a row of bricks
    while (condition2) {
      //draws one row of bricks
    }
  }

Putting Things Together

int level = 0;
double brickY = WALL_Y;
while (level < WALL_HEIGHT) {
  brickInLevel = 0;
  brickX = WALL_X;

  //draw one row of bricks
  while (brickInLevel < WALL_WIDTH) {
    new FilledRect (brickX, brickY, 
                    BRICK_WIDTH, BRICK_HEIGHT, 
                    canvas);
    brickX = brickX + BRICK_WIDTH+1;
    brickInLevel ++;
  }
  brickY = brickY - BRICK_HEIGHT-1;
  level ++;
}
Making Code Simple and Clear

• Avoid empty if-parts

<table>
<thead>
<tr>
<th>No</th>
<th>Yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>if ( box.contains(point)) { //do nothing } else { counter++; }</td>
<td>If ( !box.contains (point) ) { counter++; }</td>
</tr>
</tbody>
</table>

• Use Boolean expressions in assignments

if ( box.contains (point) ) { boxGrabbed = true; } else { boxGrabbed = false; }

boxGrabbed = box.contains.(point);

• Don’t use true or false in conditionals

if ( boxGrabbed == true ) { ... } else { ... }

if (boxGrabbed) { ... }
Simplifying Code with DeMorgan’s Laws

• DeMorgan’s Laws

\[ ! ( A \&\& B ) = !A \| !B \]

\[ ! ( A \| B ) = !A \&\& !B \]

Applying DeMorgan’s Laws

• Simplify: \(! ( x < 0 \| x \geq 100 )\)

using \(! ( A \| B ) = !A \&\& !B\)

\[ ! ( x < 0 ) \&\& !( x \geq 100 ) \]

\[ ( x \geq 0 ) \&\& ( x < 100 ) \]
Curly Braces

Curly braces bracketing multiple lines of code are necessary

```java
if ( targetContains(pt) ) {
    target.hide();
    score++;
}
```

In the second version, score is updated despite the conditional

```
if ( targetContains (pt) )
    target.hide();
    score++;
```

Curly Braces

A single line of code runs the same with and without curly braces

```java
if ( temperature >= 100 ) {
    display.setText("Water is in a gaseous phase");
}
```

is the same as

```java
if ( temperature >= 100 )
    display.setText("Water is in a gaseous phase");
```
Curly Braces

Which interpretation is correct?

```java
if ( temperature >= 80 )
  if (raining)
    display.setText("Bring an Umbrella");
  else
    display.setText("T-shirt Weather");

if ( temperature >= 80 )
  if (raining)
    display.setText("Bring an Umbrella");
  else  // WRONG!! This else matches the nearest if
    display.setText("Bring a coat!");
```

This is called the “Dangling else” problem.

Recognizing Patterns

• Counting: continually updating a value by a fixed amount

• Counting raindrops
  ```java
  int dropCount = 0;  //Raindrop counter
  while (dropCount < MAX) {
    new Raindrop(...);
    dropCount++;
  }
  ```
Counting Bricks

while ( count < TOTAL ) {
    new Brick(...);
    count++;
}

The Counting while Loop

int i = initialValue;   // initialize
while (i < stopVal) {   // test
    ...               // do stuff
    i++;              // increment
}

“Counter-Controlled Loop Pattern”
The **for** loop
(Definite Loop)

- Especially useful for counting
- Ex:

```java
for (int i=initialVal;  //initialize
     i<stopVal;        //test
     i++;) {          //increment
    ...               //do stuff
}
```

Counting Raindrops with **for** Loop

```java
for (int dropCount = 0;
     dropCount <MAX;
     dropCount++) {
    new Raindrop (...);
}
```
More General Start and End Points

- Loops can take whatever starting point, end point, and increment
  Ex:
  ```java
  for (int i=23; i <= 1728; i=i+591;){
      //do stuff
  }
  - But one should avoid using a double for any of the three values

Counting Backwards with for Loop

Ex: Printing a countdown
  ```java
  for (int count = 10; count >= 1; count--) {
      System.out.println(count);
  }
```
Update Values

- Can increment loop index by any value
- Ex: Drawing grass blades

```java
for (int pos = 0;
     pos < WIDTH;
     pos = pos + GAP) {
    new Line (pos, TOP, pos, GROUND, canvas);
}
```

General Syntax of `for` Loop

- `for` (initialization; condition; update) {
  //Do something
}

Initialization: gen’ly creates a counting variable
Condition: a boolean expression to stop the loop
Updating: updates the variable created
Nested Loops

- Any loop body can contain another loop

Ex: for ( … ) {
    while (…) {
        while (…) {
            while (…) {
                for(…) {
                }
            }
        }
    }
}

The do while Loop

- Syntax:
  do {
    <code to repeat>
  } while (<condition>)

(see Craps Example online)
**do while Loop vs while Loop**

- **do while**
  - Condition checked at the end
  - Loop body executed at least once

- **while**
  - Condition checked at the beginning
  - Loop body may never execute

**Avoiding Loop Errors**

- Easier to find errors if you know where to look

- Common loop errors include:
  - Off by 1 in counting loops
  - Infinite loops
Off by one errors

Suppose we want to run a for loop 5 times:

```
for(int i=0;i<=5; i++){
}
```

```
for(int i=0;i<5;i++) {
}
```

The left hand version will run it 6 times, not 5.

Infinite Loops

Ex:

```
while ( count< TOTAL ) {
    new Brick (...);
}
```

Since value of count is not updated, the condition in while will stay true forever.
Student To Do’s

• HW07
  – Exercise 5.8.2 (DNA Generator)
  – Exercise 5.8.3 (Morse Code)
  – Due **Monday** 11/12 by 11:59pm

• Read *Java: An Eventful Approach*
  – Ch. 7 and 13 (Today)