Loops

CSE160, Computer Science A: Honors
Stony Brook University

http://www.cs.stonybrook.edu/~cse160
Motivation

• Suppose that you need to print a string (e.g., "Welcome to Java!") a user-defined times $N$:

```
System.out.println("Welcome to Java!");
```

• While loop:

```
int count = 0;
while (count < N) {
    System.out.println("Welcome to Java");
    count++;
}
```
What is Iteration?

- Repeating a set of instructions a specified number of times or until a specific result is achieved
- How do we repeat steps?
  - Imagine 3 instructions A, B, & C:
    Instruction A
    Instruction B
    Instruction C can be jump A (meaning go back to A)
  - Iteration might result in:
    Execute A
    Execute B
    Execute C
    Execute A
    Execute A
    Execute B
    ...
    ...
Why use Iteration?

- To make our code more practical and efficient
- To make our code more flexible and dynamic

Example:

- How would we write code to print N! (factorial), where N is a number entered by the user?
- Without iteration (or recursion) this would be impractical
  - We do not know N, when we are about to write the program
Without iteration or recursion

System.out.print("Enter N: ");
int N = input.nextInt();
int factorial = 1;
if ((N == 1) || (N == 0)) factorial = 1;
else if (N == 2) factorial = 2 * 1;
else if (N == 3) factorial = 3 * 2 * 1;
else if (N == 4) factorial = 4 * 3 * 2 * 1;
else if (N == 5) factorial = 5 * 4 * 3 * 2 * 1;
...
System.out.println(factorial);

Inefficient coding (repetition)!
With iteration

```java
System.out.print("Enter N: ");
int N = input.nextInt();
int factorial = 1;
int i = 1;
while (i <= N)
    factorial *= i++;
System.out.println(factorial);

Works! No matter what N is!
```
Java and iteration

- We have 3 types of iterative statements
  - a while loop
  - a do…while loop
  - a for loop
- All 3 can be used to do similar things
- Which one should you use?
  - a matter of individual preference/convenience
while (loop-continuation-condition) {
    // loop-body;
    Statement(s);
}

int count = 0;
while (count < 100) {
    System.out.println("Welcome to Java!");
    count++;
}
Trace while Loop

```java
int count = 0;
while (count < 2) {
    System.out.println("Welcome to Java!");
    count++;
}
```

Initialize count
int count = 0;

while (count < 2) {
    System.out.println("Welcome to Java!");
    count++;
}

(count < 2) is true
int count = 0;
while (count < 2) {
    System.out.println("Welcome to Java!");
    count++;
}

Print Welcome to Java
int count = 0;
while (count < 2) {
    System.out.println("Welcome to Java!");
    count++;  
}

Increase count by 1
count is 1 now
int count = 0;
while (count < 2) {
    System.out.println("Welcome to Java!");
    count++;}

(count < 2) is still true since count is 1
int count = 0;
while (count < 2) {
    System.out.println("Welcome to Java!");
    count++;
}

Print Welcome to Java
Trace while Loop

```java
int count = 0;
while (count < 2) {
    System.out.println("Welcome to Java!");
    count++;
}
```

Increase count by 1
count is 2 now
int count = 0;

while (count < 2) {
    System.out.println("Welcome to Java!");
    count++;
}

(count < 2) is false since count is 2 now
int count = 0;
while (count < 2) {
    System.out.println("Welcome to Java!");
    count++;
}

The loop exits. Execute the next statement after the loop.
Caution: equality for reals

• Don’t use floating-point values for equality checking in a loop control - floating-point values are approximations for some values
• Example: the following code for computing \( \text{sum} = 1 + 0.9 + 0.8 + \ldots + 0.1 \):

```java
double item = 1; double sum = 0;
while (item != 0) { // No guarantee item will be 0
    sum += item;
    item -= 0.1;
}
System.out.println(sum);
```

• Variable item starts with 1 and is reduced by 0.1 every time the loop body is executed
• The loop should terminate when item becomes 0
• There is no guarantee that item will be exactly 0, because the floating-point arithmetic is approximated
• It is actually an infinite loop!
do-while Loop

d o  {
   // Loop body;
   Statement(s);
} while (loop-continuation-condition);

(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)
Why use do ... while?

- For when you have a loop body that must execute at least once
- Example: a program menu
Scanner in = new Scanner(System.in);
String selection;
int counter = 0;

do{
    System.out.println("Choose a Menu Option:");
    System.out.println("P) Print Counter");
    System.out.println("Q) Quit");
    System.out.print("ENTER: ");
    selection = in.nextLine();
    if (selection.toUpperCase().equals("P"))
        System.out.println("Counter: " + counter++);
}while(!selection.toUpperCase().equals("Q"));
System.out.println("Goodbye!");
• An Example Session

Choose a Menu Option:
P) Print Counter
Q) Quit
ENTER: P
Counter: 0
Choose a Menu Option:
P) Print Counter
Q) Quit
ENTER: A
Choose a Menu Option:
P) Print Counter
Q) Quit
ENTER: P
Counter: 1
Choose a Menu Option:
P) Print Counter
Q) Quit
ENTER: Q
Goodbye!
for (initial-action;
    loop-continuation-condition;
    action-after-each-iteration) {
    // loop body;
    Statement(s);
}

int i;
for (i = 0; i < 100; i++){
    System.out.println("Welcome to Java");
}

Loop Continuation Condition?
true false
Statement(s) (loop body)
Action-After-Each-Iteration

(i < 100)?
true false
System.out.println("Welcome to Java");
i++

(A) (B)
for loops and counting

- for loops are popular for counting loops
  - through the indices of a string
  - through the indices of an array (later)
  - through iterations of an algorithm
- Good for algorithms that require a known number of iterations
  - counter-controlled loops
int i;
for (i = 0; i < 2; i++) {
    System.out.println("Welcome to Java!");
}
int i;
for (i = 0; i < 2; i++) {
   System.out.println("Welcome to Java!");
}
int i;
for (i = 0; i < 2; i++) {
    System.out.println("Welcome to Java!");
}

(i < 2) is true since i is 0
Trace for Loop

```java
int i;
for (i = 0; i < 2; i++) {
    System.out.println(
        "Welcome to Java!";
}
```

Print Welcome to Java
int i;
for (i = 0; i < 2; i++) {
    System.out.println("Welcome to Java!");
}

Execute adjustment statement
i now is 1
int i;
for (i = 0; i < 2; i++) {
    System.out.println("Welcome to Java!");
}
int i;
for (i = 0; i < 2; i++) {
    System.out.println("Welcome to Java!");
}
int i;
for (i = 0; i < 2; i++) {
    System.out.println("Welcome to Java!");
}

Execute adjustment statement
i now is 2
int i;
for (i = 0; i < 2; i++) {
    System.out.println("Welcome to Java!");
}
int i;
for (i = 0; i < 2; i++) {
    System.out.println("Welcome to Java!");
}
Exit the loop. Execute the next statement after the loop
for loops

The **initial-action** in a **for** loop can be a list of zero or more comma-separated expressions.

The **action-after-each-iteration** in a **for** loop can be a list of zero or more comma-separated statements.

```java
for (int i = 0, j = 0; (i + j < 10); i++, j++) {
    // Do something
}
```

The loop can be empty:

```java
for (int i = 1; i < 100; System.out.println(i++));
```
Infinite loops

If the loop-continuation-condition in a `for` loop is omitted, it is implicitly true

```java
for ( ; ; ) {
  // Do something
}

(a) Equivalent

while (true) {
  // Do something
}

(b)
```
Caution;

Adding a semicolon at the end of the `for` clause before the loop body is a common mistake:

```java
int i;
for (i=0; i<10; i++) {
    System.out.println("i is " + i);
}
```

Logic Error
Caution;

Adding a semicolon at the end of the while clause before the loop body is a common mistake:

```java
int i=0;
while (i < 10);
{
    System.out.println("i is " + i);
    i++;  
}
```

Logic Error
Which Loop to Use?

while, do-while, and for loops are expressively equivalent

while (loop-continuation-condition) {
  // Loop body
}

Equivalent

for (; loop-continuation-condition; ) {
  // Loop body
}

(a)  
(b)  
Equivalent

for (initial-action;
   loop-continuation-condition;
   action-after-each-iteration) {
  // Loop body;
}

(a)  
(b)
Examples of loops

int sum = 0;
for (int j=1; j<=4; j++) {
    sum = sum + j;
    j++;
}
Be careful not to double the update of your counting variable
Using a flag

• A flag is a **boolean** loop control

```java
boolean moreWorkFlag = true;
int factorial = 1;
while (moreWorkFlag){
    factorial *= N;
    N--;
    if (N == 1) moreWorkFlag = false;
}
```

• How does it work?
  • flag used as loop condition
  • inside the loop, test for ending condition
  • when condition is reached, turn flag off
  • once turned off, loop ends
```c
int sum = 0;
for (int i=1; i<=4; i++)
    sum = sum + i;
```

<table>
<thead>
<tr>
<th>sum</th>
<th>i</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>
Nested Loops

```java
for (int i = 1; i <= 10; i++){
    for (int j = 1; j <= 10; j++){
        int product = i*j;
        System.out.print(product + " ");
    }
    System.out.print("\n");
}
```

```
1 2 3 4 5 6 7 8 9 10
2 4 6 8 10 12 14 16 18 20
3 6 9 12 15 18 21 24 27 30
... 10 20 30 40 50 60 70 80 90 100
```
Local Variables and Blocks

• A variable declared inside a block is known only inside that block
  • it is *local* to the block, therefore it is called a *local variable*
  • when the block finishes executing, local variables disappear
    • references to it outside the block cause a compiler error
  
• That includes *Init field* of *for* loops:

```
for(int i=0; i < 10; i++) {...}
```
Java **Good programming Practice**

- Do not declare variables inside loops it takes time during execution to create and destroy variables, so it is better to do it just once for loops)
Keywords break and continue

- You can also use **break** in a loop to immediately terminate the loop:

```java
public static void main(String[] args) {
    int sum = 0;
    int number = 0;
    while (number < 20) {
        number++;
        sum += number;
        if (sum >= 100) // increments until the sum is greater than 100
            break;
    }
    System.out.println("The number is " + number);
    System.out.println("The sum is " + sum);
}
```

The number is 14
The sum is 105
Keywords break and continue

- You can also use `continue` in a loop to end the current iteration and program control goes to the end of the loop body (and continues the loop):

```java
public static void main(String[] args) {
    int sum = 0;
    int number = 0;
    while (number < 20) {
        number++;
        if (number == 10 || number == 11)
            continue;
        sum += number;
    }
    System.out.println("The number is " + number);
    System.out.println("The sum is " + sum);
}
```

The number is 20
The sum is 189