Loops

CSE 114, Computer Science 1
SUNY Korea

Pradipta De
pradipta.de@sunykorea.ac.kr

Acknowledgement for the slides: Dr. Paul Fodor (CS Stony Brook)
Motivation

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

```java
System.out.println("Welcome to Java!");
...
N?
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 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 = Keyboard.readInt();
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)!
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++;
}
int count = 0;

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

Initialize count
Trace while Loop, cont.

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

```java
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++;
}
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 \(1 + 0.9 + 0.8 + \ldots + 0.1\):

```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!
Is this code correct?

Print to console “Welcome to Java” 100 times

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

Careful when choosing the loop condition

Avoid the “off-by-one” error
do-while Loop

do {
    // Loop body;
    Statement(s);
} while (loop-continuation-condition);
Why use do ... while?

- For when you have a loop body that must execute at least once.
- Example: a program menu
String selection;
PrintStream out = System.out;
Scanner in = new Scanner(System.in);
int counter = 0;
do{
    out.println("Choose a Menu Option:");
    out.println("P) Print Counter");
    out.println("Q) Quit");
    out.print("ENTER: ");
    selection = in.nextLine();
    if (selection.toUpperCase().equals("P"))
        out.println("Counter: " + counter++);
}while(!selection.toUpperCase().equals("Q"));
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!
Loop Design Strategies

- Step 1: Identify the statements that need to be repeated.

- Step 2: Wrap these statements in a loop like this:
  ```java
  while (true) {
    Statements;
  }
  ```

- Step 3: Code the `loop-continuation-condition` and add appropriate statements for controlling the loop.
  ```java
  while (loop-continuation-condition) {
    Statements;
    Additional statements for controlling the loop;
  }
  ```
Keywords “break” and “continue”

```java
public class TestBreak {
    public static void main(String[] args) {
        int sum = 0;
        int number = 0;

        while (number < 20) {
            number++;
            sum += number;
            if (sum >= 100)
                break;
        }
        System.out.println("The number is " + number);
        System.out.println("The sum is " + sum);
    }
}
```

```java
int number = 0;

while (number < 20) {
    number++;
    if (number == 10 || number == 11)
        continue;
    sum += number;
}

System.out.println("The sum is " + sum);
```

The number is 14
The sum is 189
The sum is 105
for Loops

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

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

(A) Action-After-Each-Iteration
(B) Loop Continuation Condition?
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
Trace for Loop

```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 initializer
i is now 0
Trace for Loop, cont.

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

(i < 2) is true since i is 0
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 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!"seudt);
}

Execute adjustment statement
i now is 2
Trace for Loop, cont.

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

(i < 2) is false since i is 2
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 = 1; i < 100; System.out.println(i++));

for (int i = 0, j = 0; (i + j < 10); i++, j++) {
    // Do something
}
```
Do the code snippets generate same output?

```java
for (int i = 0; i < 10; ++i) {
    sum += i;
}
```

Output: 45

```java
for (int i = 0; i < 10; i++) {
    sum += i;
}
```

Output: 45
Infinite loops

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

The infinite loop (a) is correct.

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

Equivalent

```
while (true) {
  // Do something
}
```
Caution;

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

```java
for (int 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
}

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

Equivalent

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

While and for loop constructs are pre-test loops ➔ condition is checked before executing loop body

Do-while loop construct is a post-test loop ➔ condition is checked after loop body is executed at least once
Using a flag

- A flag is a boolean loop control
  ```java
  boolean flag = true;
  ```

- 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

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

```c
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
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("
```

```
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```
Code review

How many times the loop action is executed?

```java
for (int i = 0; i < 10000; i++)
    for (int j = 0; j < 10000; j++)
        for (int k = 0; k < 10000; k++)
            Perform an action
```

$10^{12}$

```java
for (int i = 0; i < 10; i++)
    for (int j = 0; j < i; j++)
        System.out.println(i * j)
```

45
Local Variables and Blocks

- A block (a compound statement) is the set of statements between a pair of matching braces (curly brackets).
- 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.
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.
- OK to declare loop counters in the *Init field* of for loops:
  ```java
  for(int i=0; i < 10; i++)
  ...
  ```