Software Development (CS2500)

Lecture 10: Iteration

M. R. C. van Dongen

October 14, 2013

The `for` Statement

- Mainly used for *bounded iteration*.

Java

```
for (⟨initialisation⟩; ⟨condition⟩; ⟨update⟩) {
    ⟨stuff⟩
}
```

1. The statement starts by carrying out ⟨`initialisation`⟩.
2. Carries out ⟨`stuff`⟩ while ⟨`condition`⟩ holds.
3. After each iteration ⟨`update`⟩ is carried out.

The `for` Statement

- Mainly used for *bounded iteration*.

Java

```
for (⟨initialisation⟩; !⟨done⟩; ⟨update⟩) {
    ⟨stuff⟩
}
```

1. The statement starts by carrying out ⟨`initialisation`⟩.
2. Carries out ⟨`stuff`⟩ while not ⟨`done`⟩.
3. After each iteration ⟨`update`⟩ is carried out.

The `for` Statement

Java

```
int digit; // Declare induction variable.
for (digit = 0; digit <= 1; digit++) {
    System.out.print( "Next binary digit is " );
    System.out.println( digit );
}
```

The `for` Statement

Java

```
int digit; // Declare induction variable.
for (digit = 0; digit <= 1; digit++) {
    System.out.print( "Next binary digit is " );
    System.out.println( digit );
}
```

The `for` Statement

Java

```
int digit; // Declare induction variable.
for (digit = 0; digit <= 1; digit++) {
    System.out.print( "Next binary digit is " );
    System.out.println( digit );
}
```

The `for` Statement

Java

```
int digit; // Declare induction variable.
for (digit = 0; digit <= 1; digit++) {
    System.out.print( "Next binary digit is " );
    System.out.println( digit );
}
```

The `for` Statement

Java

```
int digit; // Declare induction variable.
for (digit = 0; digit <= 1; digit++) {
    System.out.print( "Next binary digit is " );
    System.out.println( digit );
}
```

The `for` Statement

Java

```
int digit; // Declare induction variable.
for (digit = 0; digit <= 1; digit++) {
    System.out.print( "Next binary digit is " );
    System.out.println( digit );
}
```

The `for` Statement

Java

```
int digit; // Declare induction variable.
for (digit = 0; digit <= 1; digit++) {
    System.out.print( "Next binary digit is " );
    System.out.println( digit );
}
```

The `for` Statement

Java

```
int digit; // Declare induction variable.
for (digit = 0; digit <= 1; digit++) {
    System.out.print( "Next binary digit is " );
    System.out.println( digit );
}
```

Don't Try This at Home

```
...
int helper;
for (helper = 0; helper <= 1; helper++ ) {
    System.out.println( "Next binary digit is " + helper );
}
...
for (helper = 0; helper <= 1; helper++ ) {
    System.out.println( "Next binary digit is " + helper );
}
```

Don't Try This at Home

```
int helper;
int helper;
for (helper = 0; helper <= 1; helper++ ) {
    System.out.println( "Next binary digit is " + helper );
}
...
for (helper = 0; helper <= 1; helper++ ) {
    System.out.println( "Next binary digit is " + helper );
}
```

Don't Try This at Home

```
int helper;
int helper;
for (helper = 0; helper <= 1; helper++ ) {
    System.out.println( "Next binary digit is " + helper );
}
...
for (helper = 0; helper <= 1; helper++ ) {
    System.out.println( "Next binary digit is " + helper );
}
```

The `for` Statement: Keep Yer Variables Local

Java

```
for (int digit = 0; digit <= 1; digit++) {
    System.out.print( "Next binary digit is " + digit );
}
```

The `while` Statement

- Mainly used for *unbounded iteration*.

Java

```
while (⟨condition⟩) {
    ⟨stuff⟩
}
```

- This carries out ⟨`stuff`⟩ while ⟨`condition`⟩ holds.

The `while` Statement

Java

```
final double initialBalance = 10000.0;
final double targetBalance = 20000.0;
final double interestRate = 5.00;

double balance = initialBalance;
int years = 0;
while (balance < targetBalance) {
    years++;
    final double interest = balance * interestRate / 100.0;
    balance = balance + interest;
}

System.out.println( "initial balance: " + initialBalance );
System.out.println( "target balance: " + targetBalance );
System.out.println( "years: " + years );
System.out.println( "balance: " + balance );
```

Input Validation

Java

```
final Scanner scanner = new Scanner( System.in );
boolean done = false;
int input = 0;

while (!done) {
    System.err.print( "Please enter a positive number: " );
    input = scanner.nextInt( );
    done = input > 0;
    if (!done) {
        System.err.println( "That's not a positive number." );
    }
}

System.out.println( "Your number is: " + input );
```

Variable Must Have a Value

Java

```
final Scanner scanner = new Scanner( System.in );
boolean done = false;
int input = 0;

while (!done) {
    System.err.print( "Please enter a positive number: " );
    input = scanner.nextInt( );
    done = input > 0;
    if (!done) {
        System.err.println( "That's not a positive number." );
    }
}

System.out.println( "Your number is: " + input );
```

Make Sure We Enter the Loop

Java

```
final Scanner scanner = new Scanner( System.in );
boolean done = false;
int input = 0;

while (!done) {
    System.err.print( "Please enter a positive number: " );
    input = scanner.nextInt( );
    done = input > 0;
    if (!done) {
        System.err.println( "That's not a positive number." );
    }
}

System.out.println( "Your number is: " + input );
```

We're Done if Input is Correct

Java

```
final Scanner scanner = new Scanner( System.in );
boolean done = false;
int input = 0;

while (!done) {
    System.err.print( "Please enter a positive number: " );
    input = scanner.nextInt( );
    done = input > 0;
    if (!done) {
        System.err.println( "That's not a positive number." );
    }
}

System.out.println( "Your number is: " + input );
```

Deal With Error if Needed

Java

```
final Scanner scanner = new Scanner( System.in );
boolean done = false;
int input = 0;

while (!done) {
    System.err.print( "Please enter a positive number: " );
    input = scanner.nextInt( );
    done = input > 0;
    if (!done) {
        System.err.println( "That's not a positive number." );
    }
}

System.out.println( "Your number is: " + input );
```

The do-while Statement

Java

```
do {
    ⟨statement⟩
} while (⟨condition⟩);
```

Java

```
⟨statement⟩
while (⟨condition⟩) {
    ⟨statement⟩
}
```

The do-while Statement

Java

```
do {
    ⟨statement⟩
} while (⟨condition⟩);
```

Java

```
⟨statement⟩
while (⟨condition⟩) {
    ⟨statement⟩
}
```

The do-while Statement

Java

```
do {
    ⟨statement⟩
} while (⟨condition⟩);
```

Java

```
⟨statement⟩
while (⟨condition⟩) {
    ⟨statement⟩
}
```

The do-while Statement

⟨condition⟩ is true

Java

```
do {
    ⟨statement⟩
} while (⟨condition⟩);
```

Java

```
⟨statement⟩
while (⟨condition⟩) {
    ⟨statement⟩
}
```

The do-while Statement

Java

```
do {
    ⟨statement⟩
} while (⟨condition⟩);
```

Java

```
⟨statement⟩
while (⟨condition⟩) {
    ⟨statement⟩
}
```

The do-while Statement

⟨condition⟩ is true

Java

```
do {
    ⟨statement⟩
} while (⟨condition⟩);
```

Java

```
⟨statement⟩
while (⟨condition⟩) {
    ⟨statement⟩
}
```

The do-while Statement

Java

```
do {
    ⟨statement⟩
} while (⟨condition⟩);
```

Java

```
⟨statement⟩
while (⟨condition⟩) {
    ⟨statement⟩
}
```

The do-while Statement

⟨condition⟩ is false

Java

```
do {
    ⟨statement⟩
} while (⟨condition⟩);
```

Java

```
⟨statement⟩
while (⟨condition⟩) {
    ⟨statement⟩
}
```

The do-while Statement

Done

Java

```
do {
    ⟨statement⟩
} while (⟨condition⟩);
```

Java

```
⟨statement⟩
while (⟨condition⟩) {
    ⟨statement⟩
}
```

Input Validation

Java

```
final Scanner scanner = new Scanner( System.in );
boolean done;
int input;

do {
    System.err.print( "Please enter a positive number: " );
    input = scanner.nextInt( );
    done = input > 0;
    if (error) {
        System.err.println( "That's not a positive number." );
    }
} while (!done);

System.out.println( "Your number is: " + input );
```

Input Validation

Java

```
final Scanner scanner = new Scanner( System.in );
boolean done;
int input;

do {
    System.err.print( "Please enter a positive number: " );
    input = scanner.nextInt( );
    done = input > 0;
    if (error) {
        System.err.println( "That's not a positive number." );
    }
} while (!done);

System.out.println( "Your number is: " + input );
```

Input Validation

Java

```
final Scanner scanner = new Scanner( System.in );
boolean error;
int input;

do {
    System.err.print( "Please enter a positive number: " );
    input = scanner.nextInt( );
    error = input <= 0;
    if (error) {
        System.err.println( "That's not a positive number." );
    }
} while (error);

System.out.println( "Your number is: " + input );
```

Adding Numbers

Java

```
int i, sum;

i = 0;
sum = 0;
while (i < 100) {
  i = i + 1;
  sum = sum + i;
} // sum == 1 + 2 + ... + 100
```

Invariants

- *Invariants* relate the values of the variables in your program.
  - Concretize: Makes relationships explicit (documentation).
    - This helps when writing the program.
  - Correctness: They may help you prove the program is correct.
  - Maintenance: They help you maintain your program.
- Good programmers state invariants as comments in programs.

Not Meaningful

Don't Try This at Home

```
// variable declaration.
int x;

// assign zero to x.
x = 0;

// add two to x.
x = x + 2;

// increment x.
x++;
```

Useful Relationship

Java

```
if (⟨condition⟩) {
    // ⟨condition⟩
    .
    .
    .
} else {
    //
    .
    .
    .
}
```

Useful Relationship

Java

```
if (⟨condition⟩) {
    // ⟨condition⟩
    .
    .
    .
} else {
    // ! ⟨condition⟩
    .
    .
    .
}
```

Another Useful Relationship

Java

```
// ⟨condition⟩1
while (⟨condition⟩2) {
    .
    .
    .
    // ⟨condition⟩1
}
    //
```

Another Useful Relationship

Java

```
// ⟨condition⟩1
while (⟨condition⟩2) {
    .
    .
    .
    // ⟨condition⟩1
}
    //                ! ⟨condition⟩2
```

Another Useful Relationship

Java

```
// ⟨condition⟩1
while (⟨condition⟩2) {
    .
    .
    .
    // ⟨condition⟩1
}
    // ⟨condition⟩1
```

Another Useful Relationship

Java

```
// ⟨condition⟩1
while (⟨condition⟩2) {
    .
    .
    .
    // ⟨condition⟩1
}
    // ⟨condition⟩1 && ! ⟨condition⟩2
```

Developing the Invariant

Java

```
int i, sum;

i = 0;
sum = 0;
while (i < 100) {
  i = i + 1;
  sum = sum + i;
}     // i >= 100
```

Developing the Invariant

Java

```
int i, sum;

i = 0;
sum = 0;
while (i < 100) {
  i = i + 1;
  sum = sum + i;
}     // i >= 100              && sum == 0 + 1 + ... + i
```

Developing the Invariant

Java

```
int i, sum;

i = 0;
sum = 0;
while (i < 100) {
  i = i + 1;
  sum = sum + i;
}     // i >= 100              && sum == 0 + 1 + ... + i
                  // i == 100 && sum == 0 + 1 + ... + i
```

Developing the Invariant

Java

```
int i, sum;

i = 0;
sum = 0;
while (i < 100) {
  i = i + 1;
  sum = sum + i;
}     // i >= 100              && sum == 0 + 1 + ... + i
                  // i == 100 && sum == 0 + 1 + ... + i
                              // sum == 0 + 1 + ... + 100
```

Developing the Invariant

Java

```
int i, sum;

i = 0;
sum = 0;
while (i < 100) {
  i = i + 1;
  sum = sum + i;
}     // i >= 100 && i <= 100 && sum == 0 + 1 + ... + i
                  // i == 100 && sum == 0 + 1 + ... + i
                              // sum == 0 + 1 + ... + 100
```

Developing the Invariant

Java

```
int i, sum;

i = 0;
sum = 0;          // i <= 100 && sum == 0 + 1 + ... + i
while (i < 100) {
  i = i + 1;
  sum = sum + i;  // i <= 100 && sum == 0 + 1 + ... + i
}     // i >= 100 && i <= 100 && sum == 0 + 1 + ... + i
                  // i == 100 && sum == 0 + 1 + ... + i
                              // sum == 0 + 1 + ... + 100
```

Developing the Invariant

Java

```
int i, sum;

i = 0;
sum = 0;           // i <= 100 && sum == 0 + 1 + ... + i
while (i < 100) {
  i = i + 1;       // i <= 100 && sum == 0 + 1 + ... + i-1
  sum = sum + i;   // i <= 100 && sum == 0 + 1 + ... + i
}     // i >= 100 && i <= 100 && sum == 0 + 1 + ... + i
                  // i == 100 && sum == 0 + 1 + ... + i
                               // sum == 0 + 1 + ... + 100
```

Developing the Invariant

Java

```
int i, sum;

i = 0;
sum = 0;           // i <= 100 && sum == 0 + 1 + ... + i
while (i < 100) { // i <  100 && sum == 0 + 1 + ... + i
  i = i + 1;       // i <= 100 && sum == 0 + 1 + ... + i-1
  sum = sum + i;  // i <= 100 && sum == 0 + 1 + ... + i
}     // i >= 100 && i <= 100 && sum == 0 + 1 + ... + i
                  // i == 100 && sum == 0 + 1 + ... + i
                                 // sum == 0 + 1 + ... + 100
```

Loop the Loop

Puzzler: How Many Iterations?

Don't Try This at Home

```
public class LoopDeLoop {
    public static void main( String[] args ) {
        final int LAST_INDEX = Integer.MAX_VALUE;
        final int FIRST_INDEX = Integer.MAX_VALUE - 10;
        int count = 0;
        for (int index = FIRST_INDEX; index <= LAST_INDEX; index++) {
            count++;
        }
        System.out.println( count );
    }
}
```

For Wednesday

- Study Sections 5.1, 5.2, and 5.4.

Acknowledgements

- This lecture corresponds to [*Big Java, Early Objects*, 5.1–5.4].
- The puzzler is based on Bloch, and Gafter 2005, Puzzles 26.

Bibliography


Bloch, Joshua, and Neal Gafter [2005]. *Java Puzzlers Traps, Pitfalls, and Corner Cases.* Addison–Wesley. ISBN: 0-321-33678-x.


Horstfmann, Cay S. *Big Java, Early Objects.* International Student Version. Wiley. ISBN: 978-1-118-31877-5.

About this Document

- This document was created with `pdflatex`.
- The LaTeX document class is `beamer`.