

Programming in Java: lecture 3

- Control structures
 - Blocks
 - Loops
 - while
 - do ... while
 - for
 - Branches
 - if – else
 - switch
- Algorithm development
 - Pseudocode
 - Stepwise refinement

Slides made for use with "Introduction to Programming Using Java, Version 5.0" by David J. Eck
Some figures are taken from "Introduction to Programming Using Java, Version 5.0" by David J. Eck
Lecture 3 covers Section 3.1 to 3.6

Blocks

- Grouping things together
- Simple statements and compound statement
- Statements end with ; or }

```
{  
    <statements>  
}
```

```
{ // This block exchanges the values of x and y  
  int temp; // A temporary variable for use in this block.  
  temp = x; // Save a copy of the value of x in temp.  
  x = y; // Copy the value of y into x.  
  y = temp; // Copy the value of temp into y.  
}
```

Loops

- `While` loop
- `Do while` loop
- `For` loop
- We only need one of these to have a complete language
- We have several for convenience

While loop

- Two variants

```
while (<boolean-expression>
    <statement>
```

```
while (<boolean-expression> {
    <statements>
}
```

While examples

- Declare variables
- Prime loop
- Iterate

```
int number;    // The number to be printed.
number = 1;    // Start with 1.
while ( number < 6 ) { // Keep going as long as number is < 6.
    System.out.println(number);
    number = number + 1; // Go on to the next number.
}
System.out.println("Done!");
```

Do ... while loop

- Two variants

```
do
```

```
    <statement>
```

```
while (<boolean-expression>);
```

```
do {
```

```
    <statements>
```

```
} while (<boolean-expression>);
```

Do .. while examples

- Always executes loop body once

```
boolean wantsToContinue; // True if user wants
                          // to play again.
do {
    Checkers.playGame();
    TextIO.put("Do you want to play again? ");
    wantsToContinue = TextIO.getlnBoolean();
} while (wantsToContinue == true);
```

For loops

- Why yet another type?
- Standard form

```
<initialization>  
while (<continuation-condition>) {  
    <statements>  
    <update>  
}
```

```
for (<init>;<continuation-cond>; <update>) {  
    <statements>  
}
```

```
// Also possible without block statement
```


For loop examples

■ Simplification

```
years = 0; // initialize the variable years
while ( years < 5 ) { // condition for continuing loop
    interest = principal * rate; //
    principal += interest; // do three statements
    System.out.println(principal); //
    years++; // update the value of the variable, years
}
```

Becomes

```
for ( years = 0; years < 5; years++ ) {
    interest = principal * rate;
    principal += interest;
    System.out.println(principal);
}
```

For loop

- **Standard form**

```
for (<variable> = <min>;<variable> <= <max>;<variable>++ ) {  
    <statements>  
}
```

There is an error below

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

Off by one errors

```
for (int i = 1; i < 10; i++) {  
    System.out.println(i);  
}
```

Several counters

- More advanced for loops

```
for ( i=1, j=10; i <= 10; i++, j-- ) {  
    TextIO.printf("%5d", i); // Output i in a 5-character wide column.  
    TextIO.printf("%5d", j); // Output j in a 5-character column  
    TextIO.putln();          // and end the line.  
}
```

```
1    10  
2     9  
3     8  
4     7  
5     6  
6     5  
7     4  
8     3  
9     2  
10    1
```

Iterating over chars

- Printing out the English alphabet
- Unfortunately its more complex for Danish characters

```
// Print out the alphabet on one line of output.  
char ch; // The loop control variable;  
        //           one of the letters to be printed.  
for ( ch = 'A'; ch <= 'Z'; ch++ ) {  
    System.out.print(ch);  
}  
System.out.println();
```

Nested for loops

```
for ( rowNumber = 1; rowNumber <= 12; rowNumber++ ) {  
    for ( N = 1; N <= 12; N++ ) {  
        // print in 4-character columns  
        System.out.printf( "%4d;", N * rowNumber ); // No new line!  
    }  
    System.out.println(); // Add a carriage return at end of the line.  
}
```

```
1;   2;   3;   4;   5;   6;   7;   8;   9;  10;  11;  12;  
2;   4;   6;   8;  10;  12;  14;  16;  18;  20;  22;  24;  
3;   6;   9;  12;  15;  18;  21;  24;  27;  30;  33;  36;  
4;   8;  12;  16;  20;  24;  28;  32;  36;  40;  44;  48;  
5;  10;  15;  20;  25;  30;  35;  40;  45;  50;  55;  60;  
6;  12;  18;  24;  30;  36;  42;  48;  54;  60;  66;  72;  
7;  14;  21;  28;  35;  42;  49;  56;  63;  70;  77;  84;  
8;  16;  24;  32;  40;  48;  56;  64;  72;  80;  88;  96;  
9;  18;  27;  36;  45;  54;  63;  72;  81;  90;  99; 108;  
10; 20; 30; 40; 50; 60; 70; 80; 90; 100; 110; 120;  
11; 22; 33; 44; 55; 66; 77; 88; 99; 110; 121; 132;  
12; 24; 36; 48; 60; 72; 84; 96; 108; 120; 132; 144;
```

Enums and for each loops

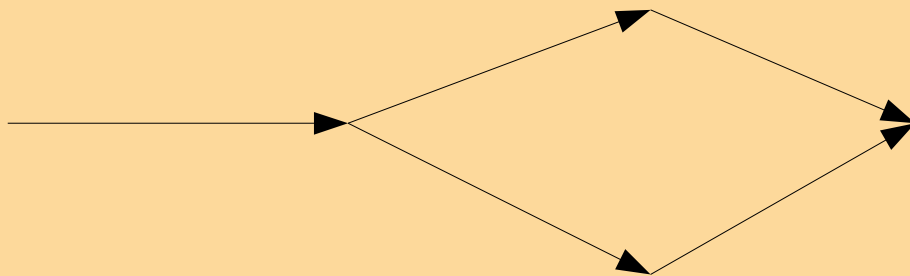
```
for ( <enum-type-name> <variable-name> : <enum-type-name>.values() ) {  
    <statements>  
}
```

- Could have been called for each
- Printing out days and their ordinal number

```
public class DayIterator {  
    // The days of the week  
    enum Day { MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY }  
  
    public static void main(String[] args) {  
        //  
        for (Day d : Day.values() ) {  
            System.out.println(d + " is day number " + d.ordinal());  
        }  
    }  
}
```

Branches

- The `if` statement
- The `switch` statement
- Branches the computation tree



The if statement

- **Basic form: Selects one of two actions**

```
if (<boolean-expression> )
    <statement-1>
else
    <statement-2>
```

or

```
if (<boolean-expression>) {
    <statement-1>
} else {
    <statement-2>
}
```


The dangling else problem

- When is the second case executed

```
if ( x > 0 )
    if (y > 0)
        System.out.println("First case");
else
    System.out.println("Second case");
```

The computer reads this as

```
if ( x > 0 )
    if (y > 0)
        System.out.println("First case");
else
    System.out.println("Second case");
```

The dangling else problem

- Solution:
 - use block statements

```
if ( x > 0 ) {  
    if (y > 0) {  
        System.out.println("First case");  
    }  
}  
else {  
    System.out.println("Second case");  
}
```

If...else if construction

- Used to choose between more than two things
- Example: Exactly one of the three statements are executed

```
if (<boolean-expression-1>) {  
    <statement-1>  
} else if (<boolean-expression-2>) {  
    <statement-2>  
} else {  
    <statement-3>  
}
```

If...else if construction

- Can be extended indefinitely
- Last else is optional

```
if ( boolean-expression-1 ) {
    statement-1
} else if ( boolean-expression-2 ) {
    statement-2
} else if ( boolean-expression-3 ) {
    statement-3
    .
    . // (more cases)
    .
} else if ( boolean-expression-N ) {
    statement-N
} else {
    statement-(N+1)
}
```

Conditional operator

- Leftover from lecture 2
- If then else on a single line
 - `<boolean-expression> ? <expr1> : <expr2>`
 -
 - Example
 - `max = (x > y) ? x : y;`

The switch statement

- Most common form

```
switch ( expression ) {  
    case constant-1 :  
        statements-1  
        break;  
    case constant-2 :  
        statements-2  
        break;  
    .  
    .    // (more cases)  
    .  
    case constant-N :  
        statements-N  
        break;  
    default: // optional default case  
        statements-(N+1)  
}  
// end of switch statement
```

Switch example

```
switch ( N ) { // (Assume N is an integer variable.)
  case 1:
    System.out.println("The number is 1.");
    break;
  case 2:
  case 4:
  case 8:
    System.out.println("The number is 2, 4, or 8.");
    System.out.println("(That's a power of 2!)");
    break;
  case 3:
  case 6:
  case 9:
    System.out.println("The number is 3, 6, or 9.");
    System.out.println("(That's a multiple of 3!)");
    break;
  case 5:
    System.out.println("The number is 5.");
    break;
  default:
    System.out.print("The number is 7 or is outside");
    System.out.println(" the range 1 to 9.");
}
```

Enums and switch

```
switch ( currentSeason ) {
    case WINTER:      // ( NOT Season.WINTER ! )
        System.out.println("December, January, February");
        break;
    case SPRING:
        System.out.println("March, April, May");
        break;
    case SUMMER:
        System.out.println("June, July, August");
        break;
    case FALL:
        System.out.println("September, October, November");
        break;
}
```


Definite assignment

- A variable must be assigned before it is used
- It must be assigned on all possible paths

```
String computerMove;
switch ( (int) (3*Math.random()) ) {
    case 0:
        computerMove = "Rock";
        break;
    case 1:
        computerMove = "Scissors";
        break;
    case 2:
        computerMove = "Paper";
        break;
}
System.out.println("Computer's move is " + computerMove); // ERROR!
```

Definite assignment

- Use default

```
String computerMove;  
switch ( (int)(3*Math.random()) ) {  
    case 0:  
        computerMove = "Rock";  
        break;  
    case 1:  
        computerMove = "Scissors";  
        break;  
    default:  
        computerMove = "Paper";  
        break;  
}  
System.out.println("Computer's move is " + computerMove);
```

Definite assignment

- Example 2
- No definite assignment Definite assignment

```
String computerMove;
int rand;
rand = (int)(3*Math.random());
if ( rand == 0 ) {
    computerMove = "Rock";
} else if ( rand == 1 ) {
    computerMove = "Scissors";
} else if ( rand == 2 ) {
    computerMove = "Paper";
}
```

```
String computerMove;
int rand;
rand = (int)(3*Math.random());
if ( rand == 0 ) {
    computerMove = "Rock";
} else if ( rand == 1 ) {
    computerMove = "Scissors";
} else
    computerMove = "Paper";
}
```

Break and continue

- We can exit a loop prematurely

```
while (true) { // looks like it will run forever!
    TextIO.put("Enter a positive number: ");
    N = TextIO.getlnInt();
    if (N > 0) { // input is OK; jump out of loop
        break;
    }
    TextIO.putln("Your answer must be > 0.");
}
// continue here after break
```

Labeled break

- For use in nested loops

```
boolean nothingInCommon;
nothingInCommon = true; // Assume s1 and s2 have no chars in common.
int i,j; // Variables for iterating through the chars in s1 and s2.
i = 0;
bigloop: while (i < s1.length()) {
    j = 0;
    while (j < s2.length()) {
        if (s1.charAt(i) == s2.charAt(j)) { //s1 and s2 have a common char.
            nothingInCommon = false;
            break bigloop; // break out of BOTH loops
        }
        j++; // Go on to the next char in s2.
    }
    i++; //Go on to the next char in s1.
}
```

Continue

- Jumps to the top of the loop and starts the next iteration
- Also exists in a labeled version

Algorithm development

- Pseudocode
- Stepwise refinement

Pseudocode

- English description in steps
- Can be more mathematical

Get the user's input

while there are more years to process:

 Compute the value after the next year

 Display the value

Stepwise refinement

- Get the user's input

 - Ask the user for the initial investment

 - Read the user's response

 - Ask the user for the interest rate

 - Read the user's response

3N+1 Problem

“Given a positive integer, N , define the '3N+1' sequence starting from N as follows: If N is an even number, then divide N by two; but if N is odd, then multiply N by 3 and add 1. Continue to generate numbers in this way until N becomes equal to 1. For example, starting from $N = 3$, which is odd, we multiply by 3 and add 1, giving $N = 3*3+1 = 10$. Then, since N is even, we divide by 2, giving $N = 10/2 = 5$. We continue in this way, stopping when we reach 1, giving the complete sequence: 3, 10, 5, 16, 8, 4, 2, 1.

“Write a program that will read a positive integer from the user and will print out the 3N+1 sequence starting from that integer. The program should also count and print out the number of terms in the sequence.”

3N+1 Problem

- Get a positive integer N from the user
- Compute, print, and count each number in the sequence;
- Output the number of terms;
-
- The second step is still very complex

3N+1 Problem

- Get a positive integer N from the user;
- while N is not 1;
 - Compute $N = \text{next term}$;
 - Output N ;
 - Count this term;
- Output the number of terms;

3N+1 Problem

- Branch on even

```
Get a positive integer N from the user;
while N is not 1:
    if N is even:
        Compute  $N = N/2$ ;
    else
        Compute  $N = 3 * N + 1$ ;
    Output N;
    Count this term;
Output the number of terms;
```

3N+1 Problem

- Adding counter

```
Get a positive integer N from the user;  
Let counter = 0;  
while N is not 1:  
    if N is even:  
        Compute  $N = N/2$ ;  
    else  
        Compute  $N = 3 * N + 1$ ;  
    Output N;  
    Add 1 to counter;  
Output the counter;
```

3N+1 Problem

- Handling incorrect input

Ask user to input a positive number;

Let N be the user's response;

while N is not positive:

- Print an error message;

- Read another value for N;

Let counter = 0;

while N is not 1:

- if N is even:

 - Compute $N = N/2$;

- else

 - Compute $N = 3 * N + 1$;

- Output N;

- Add 1 to counter;

Output the counter;

```

public class ThreeN1 {
    public static void main(String[] args) {
        int N;          // for computing terms in the sequence
        int counter;   // for counting the terms
        TextIO.put("Starting point for sequence: ");
        N = TextIO.getlnInt();
        while (N <= 0) {
            TextIO.put("The starting point must be positive. Please try again:");
            N = TextIO.getlnInt();
        }
        // At this point, we know that N > 0
        counter = 0;
        while (N != 1) {
            if (N % 2 == 0) {
                N = N / 2;
            } else {
                N = 3 * N + 1;
            }
            TextIO.putln(N);
            counter = counter + 1;
        }
        TextIO.putln();
        TextIO.putln("There were " + counter + " terms in the sequence.");
    } // end of main()
} // end of class ThreeN1

```


Debugging

- Debugging statements
- `System.out.println("x=" + x + " before the loop");`