Control Flow

Computer Engineering Department

Java Course

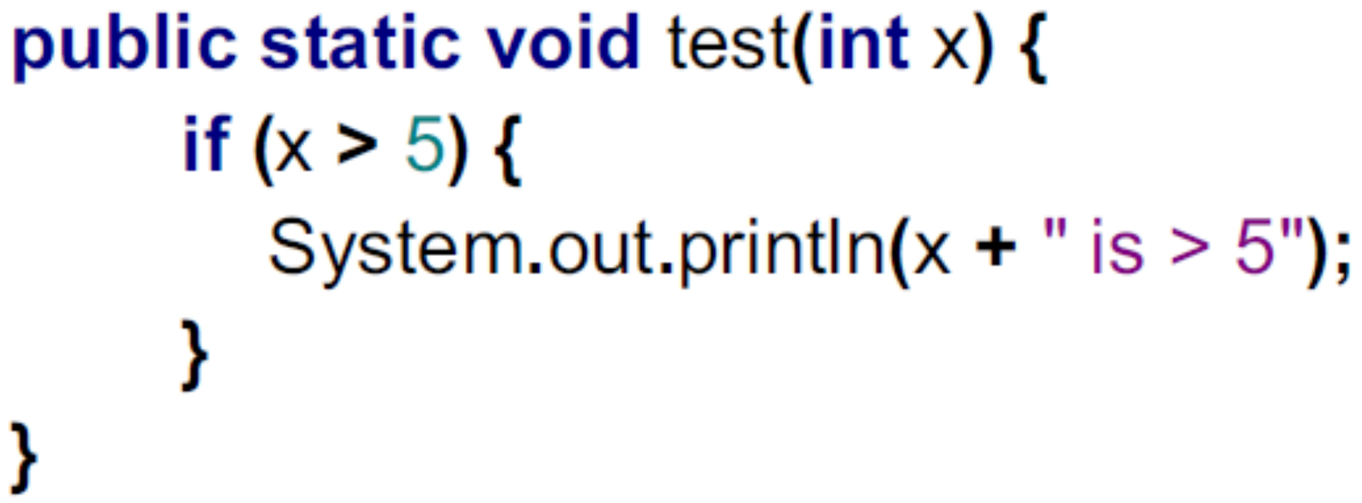
Asst. Prof. Dr. Ahmet Sayar

Kocaeli University - Fall 2014

IF Statement

If Statement

```
if (CONDITION) {
    STATEMENTS
} else {
    STATEMENTS
}
```



```
public static void test(int x) {
        if (x > 5) {
            System.out.println(x + " is > 5");
        }
}
```

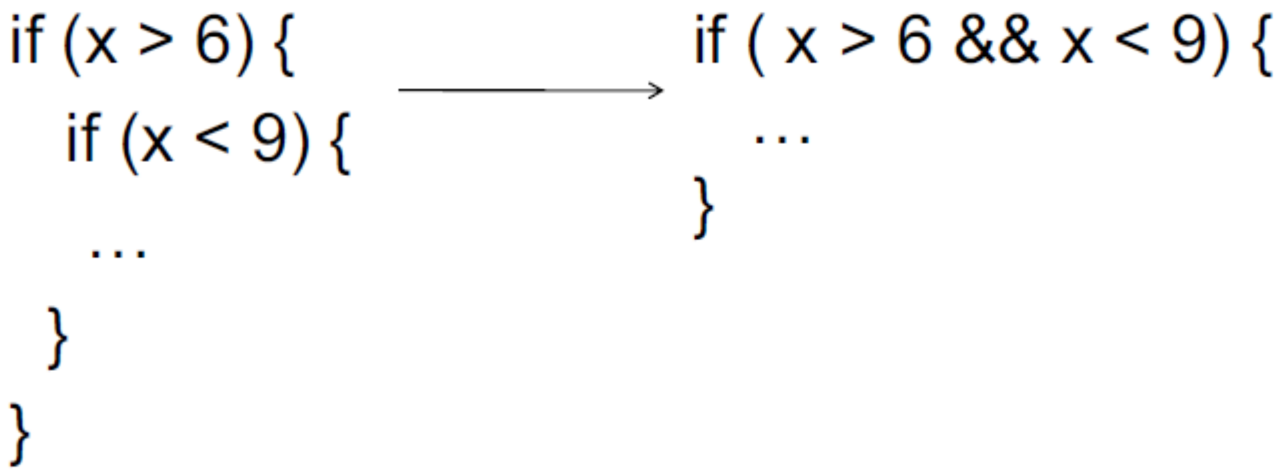
Comparison operators

- x > y: x is greater than y
- x < y: x is less than y
- x >= y: x is greater than or equal to x
- x <= y: x is less than or equal to y

- x == y: x equals y
- (equality: ==, assignment: =)

Boolean operators

- &&: logical AND
- ||: logical OR



```
if (x > 6) {
   if (x < 9) {
    ...
  }
}
```

$\longrightarrow$

```
if ( x > 6 && x < 9) {
    ...
}
```

Multibranch `if-else` Statements

- syntax

  if (*Boolean_Expression_1*)

      Statement_1

  else if (*Boolean_Expression_2)*

      Statement_2

  else if (*Boolean_Expression_3)*

      Statement_3

  else if …

  else

      Default_Statement

Compound Boolean Expressions

- Boolean expressions can be combined using the “and” (&&) operator.
- example

```
if ((score > 0) && (score <= 100))
...
```

- not allowed

```
if (0 < score <= 100)
...
```

Negating a Boolean Expression

- A boolean expression can be negated using the **“not” (!) operator**.
- syntax

  `!(Boolean_Expression)`
- example

  `(a || b) && !(a && b)`

  which is the *exclusive or*

Using ==, cont.

- == is not appropriate for determining if two objects have the same value.
  - if (s1 == s2), where s1 and s2 refer to strings, determines only if s1 and s2 refer the a common memory location.

```
String bir = new String("1"); //"1";
String iki = new String(“1"); //"1";
if (bir==iki) {
    System.out.println("esitler");
}
else{
    System.out.println("esit degiller");
}
```

Using ==, cont.

- To test the equality of objects of class String, use method equals.

  s1.equals(s2)

  or

  s2.equals(s1)

- To test for equality ignoring case, use method equalsIgnoreCase.

  (“Hello”.**equalsIgnoreCase**(“hello”))

Nested Statements

- An if-else statement can contain any sort of statement within it.
- In particular, it can contain another if-else statement.
    - An if-else may be nested within the “if” part.
    - An if-else may be nested within the “else” part.
    - An if-else may be nested within both parts.

Nested Statements, cont.

- syntax

```
if (Boolean_Expression_1)
    if (Boolean_Expression_2)
        Statement_1)
    else
        Statement_2)
else
    if (Boolean_Expression_3)
        Statement_3)
    else
        Statement_4);
```

The `switch` Statement

- The switch statement is a mutltiway branch that makes a decision based on an *integral* (integer or character) expression.
- The switch statement begins with the keyword switch followed by an integral expression in parentheses and called the *controlling expression.*

The `switch` Statement, cont.

- The action associated with a matching case label is executed.
- If no match is found, the case labeled default is executed.
  - The default case is optional, but recommended, even if it simply prints a message.
- Repeated case labels are not allowed.

The `switch` Statement, cont.

```java
import java.util.*;

public class MultipleBirths
{
    public static void main(String[] args)
    {
        int numberOfBabies;
        System.out.print("Enter number of babies: ");
        Scanner keyboard = new Scanner(System.in);
        numberOfBabies = keyboard.nextInt();

        switch (numberOfBabies)
        {
            case 1:
                System.out.println("Congratulations.");
                break;
            case 2:
                System.out.println("Wow. Twins.");
                break;
            case 3:
                System.out.println("Wow. Triplets.");
                break;
            case 4:
            case 5:
                System.out.println("Unbelieveable.");
                System.out.println(numberOfBabies + " babies");
                break;
            default:
                System.out.println("I don't believe you.");
                break;
        }
    }
}
```

controlling expression

case label

`break` *statement*

Sample Screen Dialog 1

```
Enter number of babies: 1
Congratulations.
```

Sample Screen Dialog 2

```
Enter number of babies: 3
Wow. Triplets.
```

Sample Screen Dialog 3

```
Enter number of babies: 4
Unbelievable.
4 babies
```

Sample Screen Dialog 4

```
Enter number of babies: 6
I don't believe you.
```

Display 3.5

A switch Statement

Switch Example

```
int i=1;    // outputs for differing values of i  : (0, 1, 2, 3)
switch (i) {
    case 0:
        System.out.println("zero");
        break;
    case 1:
        System.out.println("one");
    case 2:
        System.out.println("two");
    default:
        System.out.println("default");
}
```

The Conditional Operator

```
if (n1 > n2)
    max = n1;
 else
    max = n2;
```

can be written as

```
max = (n1 > n2) ? n1 : n2;
```

- The `?` and `:` together are call the *conditional operator* or *ternary operator.*

LOOPS

Java Loop Statements: Outline

- The while Statement
- The do-while Statement
- The for Statement

the `while` Statement, cont.

```
import java.util.*;

public class WhileDemo
{
    public static void main(String[] args)
    {
        int count, number;

        System.out.println("Enter a number");
        Scanner keyboard = new Scanner(System.in);
        number = keyboard.nextInt();

        count = 1;
        while (count <= number)
        {
            System.out.print(count + ", ");
            count++;
        }

        System.out.println();
        System.out.println("Buckle my shoe.");
    }
}
```

Sample Screen Dialog 1

```
Enter a number:
2
1, 2,
Buckle my shoe.
```

Sample Screen Dialog 2

```
Enter a number:
3
1, 2, 3,
Buckle my shoe.
```

Sample Screen Dialog 3

```
Enter a number:
0

Buckle my shoe.
```

The loop body is iterated zero times.

Display 3.6

A while Loop

The `do-while` Statement

- Also called a do-while loop
- Similar to a while statement, except that the loop body is executed at least once
- syntax

  do

  Body_Statement

  while (*Boolean_Expression*);

  – **don’t forget the semicolon!**

The `do-while` Statement, cont.

```
import java.util.*;

public class DoWhileDemo
{
    public static void main(String[] args)
    {
        int count, number;

        System.out.println("Enter a number");
        Scanner keyboard = new Scanner(System.in);
        number = keyboard.nextInt();

        count = 1;
        do
        {
            System.out.print(count + ", ");
            count++;
        }while (count <= number);

        System.out.println();
        System.out.println("Buckle my shoe.");
    }
}
```

Sample Screen Dialog 1

```
Enter a number:
2
1, 2,
Buckle my shoe.
```

Sample Screen Dialog 2

```
Enter a number:
3
1, 2, 3,
Buckle my shoe.
```

Sample Screen Dialog 3

```
Enter a number:
0
1,
Buckle my shoe.
```

The loop body is always executed at least one time.

Display 3.8

A do–while Loop

The `for` Statement

- A for statement executes the body of a loop a fixed number of times.
- Example

```
for (count = 1; count < 3; count++)
        System.out.println(count);
System.out.println("Done");

//watch out the usage of '{' and '}'
```

Multiple Initialization, etc.

- example

```
for (n = 1, p = 1; n < 10; n++)
    p = p * n ;
```

- Only one boolean expression is allowed, but it can consist of &&s, ||s, and !s.
- Multiple update actions are allowed, too.

```
for (n = 1, p = 1; n < 10; n++, p=p * n)
```

rarely used

Example

```
for (n = 20, p = 1; p < n; n++, p=p * 5){
        System.out.println("n: " +n+"  "+"p: " +p);
}
```

What is the output of this code?

The Empty `for` Statement

- What is printed by

```
int product = 1, number;
for (number = 1; number <= 10; number++);
     product = product * number;

System.out.println(product);
```

?

- The last semicolon in

```
for (number = 1; number <= 10; number++);
```

produces an empty `for` statement.

- for(;;){

```
      System.out.println("infinite loop");
 }
```

Choosing a Loop Statement

- If you know how many times the loop will be iterated, use a for loop.
- If you don’t know how many times the loop will be iterated, but
  - it could be zero, use a while loop
  - it will be at least once, use a do-while loop.
- Generally, a while loop is a safe choice.

The `break` Statement in Loops

- A break statement can be used to end a loop immediately.
- The break statement ends only the **innermost** loop or switch statement that contains the break statement.
- break statements make loops more difficult to understand.
- Use break statements sparingly (if ever).

Example

```
for(int i=0; i<2; i++){
        System.out.println("i: "+i);
        for (int j = 0; j < 3; j++) {
            System.out.println("j: "+j);
            if (j==1) {
                break;
            }
        }
}
```

The `break` Statement in Loops, cont.

```java
import java.util.*;

public class BreakDemo
{
    public static void main(String[] args)
    {
        int itemNumber;
        double amount, total;
        Scanner keyboard = new Scanner(System.in);

        System.out.println("You may buy ten items, but");
        System.out.println("the total price must not exceed $100.");

        total = 0;
        for (itemNumber = 1; itemNumber <= 10; itemNumber++)
        {
            System.out.print("Enter cost of item #"
                                        + itemNumber + ": $");
            amount = keyboard.nextDouble();
            total = total + amount;
            if (total >= 100)
            {
                System.out.println("You spent all your money.");
                break;
            }
            System.out.println("Your total so far is $" + total);
            System.out.println("You may purchase up to "
                          + (10    itemNumber) + " more items.");
        }
        System.out.println("You spent $" + total);
    }
}
```

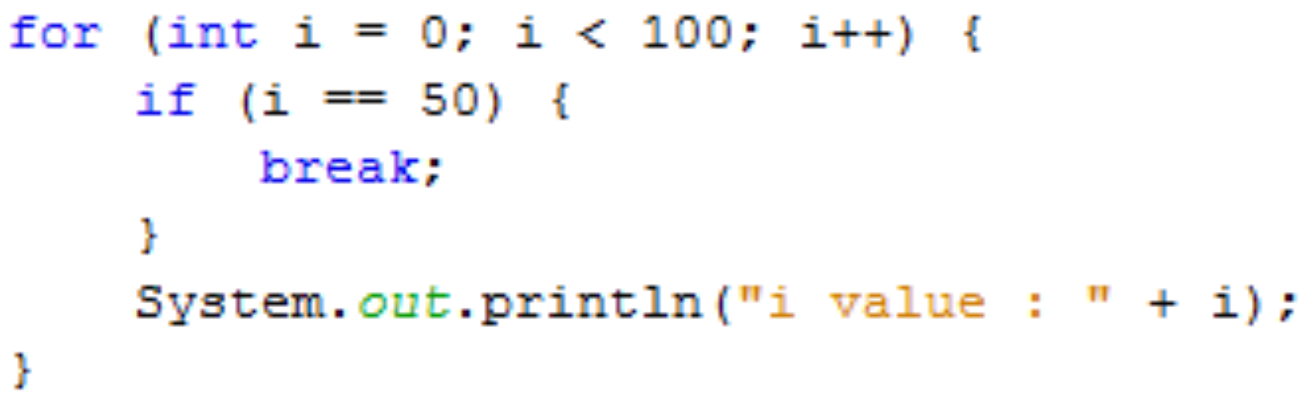
Sample Screen Dialog

```
You may buy ten items, but
the total price must not exceed $100.
Enter cost of item #1: $90.93
Your total so far is $90.93
You may purchase up to 9 more items.
Enter cost of item #2: $10.50
You spent all your money.
You spent $101.43
```

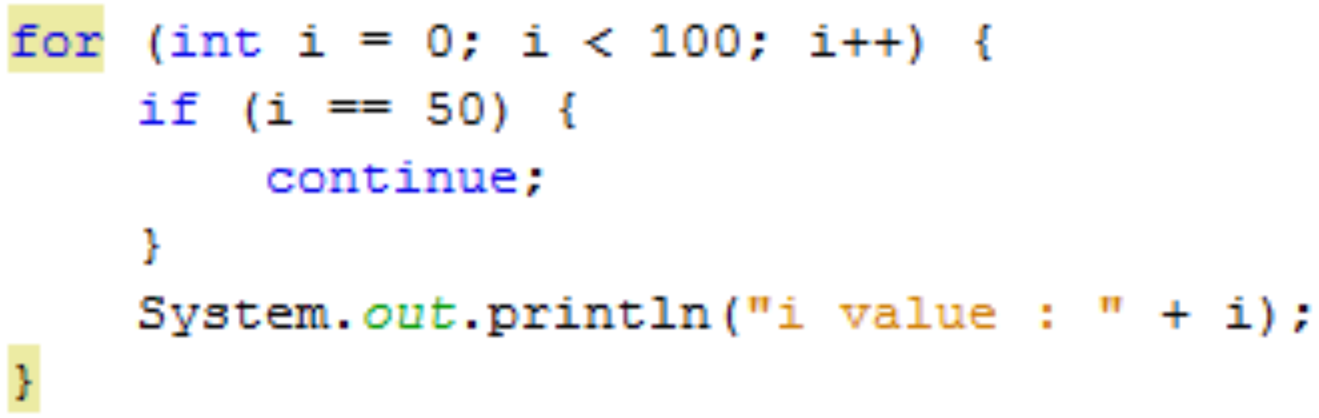
Display 3.13

Ending a Loop with a break Statement

Branching Statements



```
for (int i = 0; i < 100; i++) {
    if (i == 50) {
        break;
    }
    System.out.println("i value : " + i);
}
```



```
for (int i = 0; i < 100; i++) {
    if (i == 50) {
        continue;
    }
    System.out.println("i value : " + i);
}
```

The `exit` Method

- Sometimes a situation arises that makes continuing the program pointless.
- A program can be terminated normally by `System.exit(0).`
- Example

```
if (numberOfWinners == 0)
{
   System.out.println(“/ by 0”);
   System.exit(0);
}
```

Embedded Loops

```
for (int i = 0; i < 3; i++) {
    for (int j = 2; j < 4; j++) {
        System.out.println (i + " " + j);
    }
}
```

Ending a Loop

- If the number of iterations is known before the loop starts, the loop is called a *count-controlled loop.*
  - use a for loop.
- Asking the user before each iteration if it is time to end the loop is called the *ask-before-iterating technique.*
  - appropriate for a small number of iterations
  - Use a while loop or a do-while loop.

Ending a Loop, cont.

- For large input lists, a *sentinel value* can be used to signal the end of the list.
  - The sentinel value must be different from all the other possible inputs.
  - A negative number following a long list of nonnegative exam scores could be suitable.

```
90
0
10
-1
```

Example: `class ExamAverager`

```java
import java.util.*;

/**
 Determines the average of a list of (nonnegative) exam scores.
 Repeats for more exams until the user says she/he is finished.
*/
public class ExamAverager
{
    public static void main(String[] args)
    {
        System.out.println("This program computes the average of");
        System.out.println("a list of (nonnegative) exam scores.");
        double sum;
        int numberOfStudents;
        double next;
        String answer;
        Scanner keyboard = new Scanner(System.in);
        do
        {
            System.out.println();
            System.out.println("Enter all the scores to be averaged.");
            System.out.println("Enter a negative number after");
            System.out.println("you have entered all the scores.");
            sum = 0;
            numberOfStudents = 0;
            next = keyboard.nextDouble();
            while (next >= 0)
            {
                sum = sum + next;
                numberOfStudents++;
                next = keyboard.nextDouble();
            }
            if (numberOfStudents > 0)
                System.out.println("The average is "
                                    + (sum/numberOfStudents));
           else
                System.out.println("No scores to average.");

            System.out.println("Want to average another exam?");
            System.out.println("Enter yes or no.");
            answer = keyboard.next();
        }while (answer.equalsIgnoreCase("yes"));
    }
}
```

Sample Screen Dialog

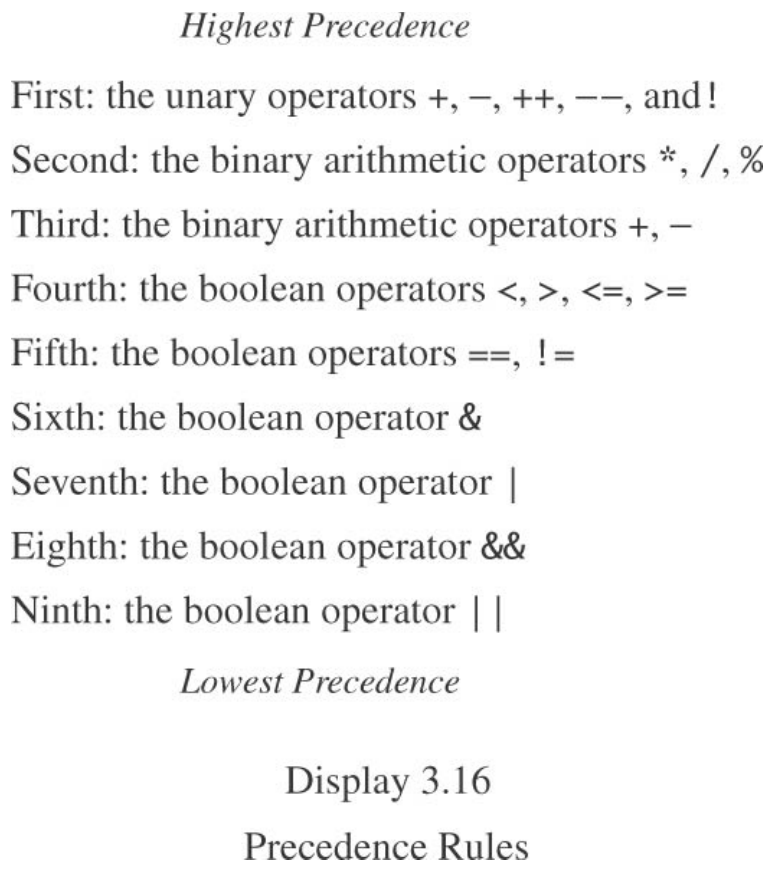
```
This program computes the average of
a list of (nonnegative) exam scores.

Enter all the scores to be averaged.
Enter a negative number after
you have entered all the scores.
100
90
100
90
-1
The average is 95.0
Want to average another exam?
Enter yes or no.
yes

Enter all the scores to be averaged.
Enter a negative number after
you have entered all the scores.
90
70
80
-1
The average is 80.0
Want to average another exam?
Enter yes or no.
no
```

Display 3.14
Nested Loops

Naming Boolean Variables

- Choose names such as `isPositive` or `systemsAreOk`.
- Avoid names such as `numberSign` or `systemStatus`.

Precedence Rules in Boolean Expressions



Highest Precedence

First: the unary operators +, −, ++, −−, and !

Second: the binary arithmetic operators *, /, %

Third: the binary arithmetic operators +, −

Fourth: the boolean operators <, >, <=, >=

Fifth: the boolean operators ==, !=

Sixth: the boolean operator &

Seventh: the boolean operator |

Eighth: the boolean operator &&

Ninth: the boolean operator ||

Lowest Precedence

Display 3.16

Precedence Rules

Precedence Rules, cont.

- In what order are the operations performed?

```
score < min/2 - 10 || score > 90
score < (min/2) - 10 || score > 90
score < ((min/2) - 10) || score > 90
(score < ((min/2) - 10)) || score > 90
(score < ((min/2) - 10)) || (score > 90)
```