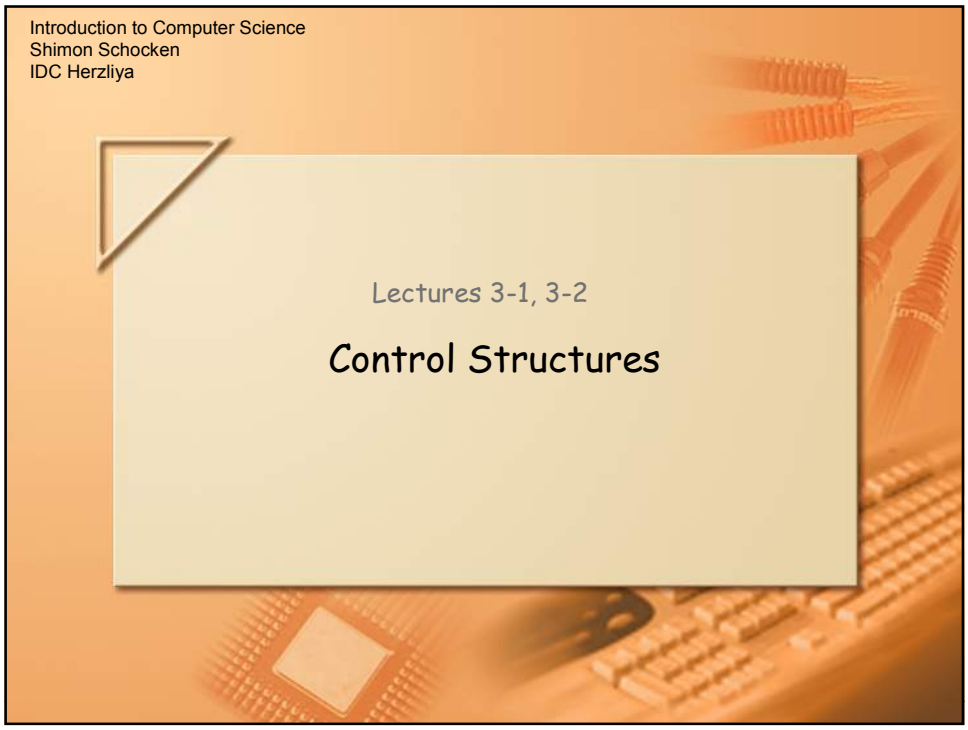


Shorthand operators (increment / decrement)

```
k++
```

Is equivalent to

```
k = k + 1
```

```
k--
```

Is equivalent to

```
k = k - 1
```

k must be a numeric variable.

Shorthand operators (increment / decrement)

```
int k = 10, n = 0;

k = k + 1;    // k=11
k++;          // k=12
++k;          // k=13

n = k++;      // k=14, n=13
n = ++k       // k=15, n=15
```

The rules of the game:

Expressions	Operation	Value Of expression
count++	add 1	old value
++count	add 1	new value
count--	subtract 1	old value
--count	subtract 1	new value

```
sum = 50
System.out.println(sum +" "+ sum++ +" "+ ++sum +" "+ sum +" "+ sum-- +" "+ sum)
```

What will it print?

Shorthand operators (assignment)

Often we perform an operation on a variable, and then assign the result to the same variable. Example: `sum = + x`

```
sum = sum + x
```

is equivalent to:

```
sum += x
```

The rules of the game:

Operation	Equivalent to
x += y	x = x + y
x -= y	x = x - y
x *= y	x = x * y
x /= y	x = x / y
x %= y	x = x % y

The right hand side, which can be a complex expression, is evaluated first, and then combined with the rest of the assignment:

```
x /= x - y
```

is equivalent to:

```
x = x / (x - y)
```

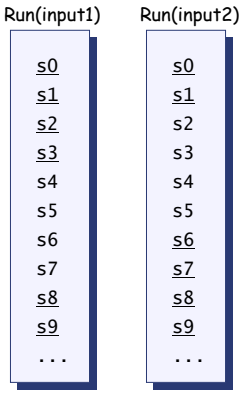
Control structures

A program is a sequence of statements.

On the right we see the same program, in two separate runs

In each run some statements execute (the underlined ones); the other statements are skipped

In order to affect such different flows of execution, we need programming mechanisms that tell the computer to "branch" to different code segments.



- Low level programming languages affect branching using GOTO statements
- High level programming languages affect branching using high-level abstractions like `if`, `while`, `switch`, and so on
- Taken together, these statements are called *control structures*.

Control structures: lecture outline

- `If`
- Boolean expressions and variables
- `While`
- `switch`
- `do`
- `for`

The if statement (example)

```
System.out.println("Enter your age:");
int age = scan.nextInt();
if (age < AGELIMIT)
    System.out.println("Sorry, " + age + " is under the allowed age.");
// Code continues ...
```

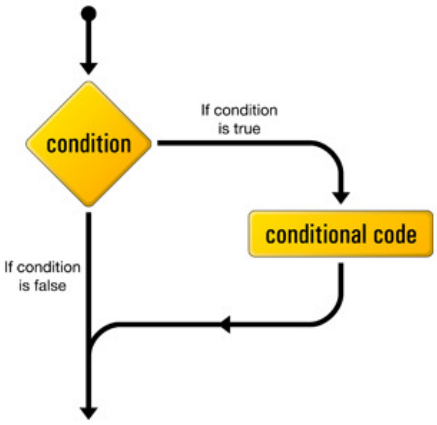
The body of the IF can be either a single statement or a a block of statements:

```
    if (age < AGELIMIT) {
      System.out.println("Sorry, " + age + " is under the allowed age.");
      System.out.println("Try again when you are old enough.");
    }
    // Code continues ...
```


The if statement

Syntax:

```
if (condition)
   conditional code
```



- `if`: the most basic branching mechanism
- Condition: a Boolean expression that evaluates to either `true` or `false`
- The condition's value determines which branch to take

(Beautiful flow chart images from http://articles.sitepoint.com/article/mysql-3-getting-started-php/3)

The if … else statement (example)

```
System.out.println("Enter your age:");
int age = scan.nextInt();
if (age < AGELIMIT)
    System.out.println("Sorry, you are under the allowed age.");
else
    System.out.println("Welcome! Wait while the game is loading ...");
// code continues ...
```

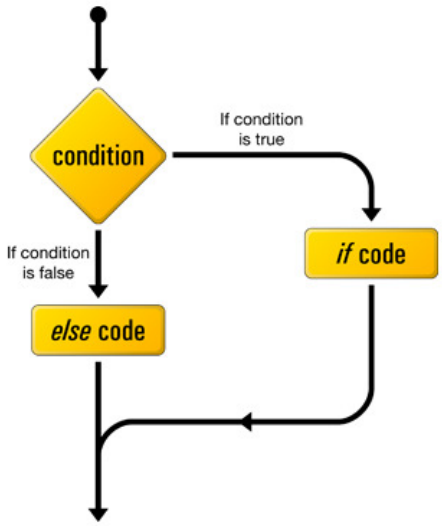
Compare to:

```
// bug
System.out.println("Enter your age:");
int age = scan.nextInt();
if (age < AGELIMIT)
    System.out.println("Sorry, you are under the allowed age.");
System.out.println("Welcome! Wait while the game is loading ...");
```


The if … else statement

Syntax:

```
if (condition)
   if code
else
   else code
```



Nested if (example)

```
System.out.println("Enter your age:");
int age = scan.nextInt();
if (age < AGELIMIT)
    System.out.println("Sorry, you are under the allowed age.");
else {
    System.out.println("Enter your credit card:");
    int cardNumber = scan.nextInt();
    if verify(cardNumber)     // calling a Boolean method
        System.out.println("Welcome! ...");
    else
        System.out.println("Sorry, bad card");
}
```


Nested if (example)

```
// Reads two integers and compares them
import java.util.Scanner;

class Compare2Numbers {

  public static void main(String[] args) {
    Scanner scan = new Scanner(System.in);

    System.out.print("Enter a number:");
    int a = scan.nextInt();
    System.out.print("Enter another number");
    int b = scan.nextInt();

    if (a != b)
      if (a > b)
        System.out.println(a + " is greater");
      else
        System.out.println(b + " is greater");
    else
      System.out.println("the numbers are equal");
  }
}
```

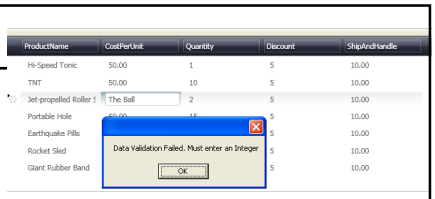
- The nesting can go as deep as you want
- Use with caution: telescoped code is hard to read and comprehend.

What's the difference between these two solutions?

```
if (a > b) System.out.println(a + " is greater");
if (a < b) System.out.println(b + " is greater");
if (a = b) System.out.println("the numbers are equal");
```

Example: using if to validate inputs

- Whenever we request users to enter inputs, we must assume that an invalid value can be entered
- Valid / invalid: according to the application's requirements.




```
System.out.println("Enter number of items:")
int n =  scan.nextInt();
if (n < 0)
    System.out.println("Number of items must be non-negative!");
else {
    double total = n * ITEM_PRICE;
    System.out.println("The total price is:" + total);
}
// Code continues
```

True / false Conditions are expressed as *Boolean expressions*

- That's a naïve solution to input validation. Can you see why?

Control structures: lecture outline

- If
- Boolean expressions and variables
- While
- switch
- do
- for

Control structures are based on evaluating some *condition*:

```
if ( (age >= 21) && (driverLicenseIsValid() )
   // allow car rental
```

condition

Where do these conditions come from?

- Algorithmic needs
- Requirements analysis

- "Only drivers older than 21 can rent a car"
- Continue the loop until Math.abs(r-x*x) < epsilon
- "If the package weights more than 10 kg or the shipping distance is more than 100 km we charge 10% extra" (ambiguous …)
- "Three numbers x, y, z are said to be Pythagorean if $x^2 + y^2 = z^2$ "

Such statements are expressed in computer programs as **Boolean expressions**.

Boolean expressions in action

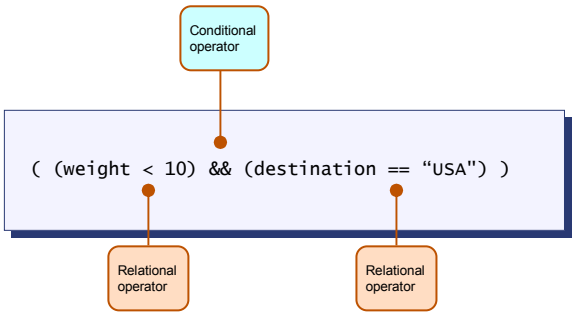
```
// Some electronic commerce application:
if ((weight < 10) && (destination == "USA"))
   shippingCost = 0
else
   shippingCost = weight * COST_PER_POUND;
```

```
if ( (x < 0) || (x > 100) )
   System.out.println("x must be between 0 and 100");

// Logically equivalent to:

if ( !((x >= 0) && (x <= 100)) )
   System.out.println("x must be between 0 and 100");
```

Boolean expressions anatomy



A relational operator computes a Boolean function on it's operand(s) and returns true or false

A conditional operator computes a Boolean operation on it's Boolean operand(s) and returns true or false.

Relational operators

Operator	Meaning
==	equal to
!=	not equal to
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to

Examples:

```
age > 18
city == "LA"
x != 100
gpa <= 3.75
tailIsDown() == true
bob.bandWidth > alice. bandWidth
```

- Each *operator* operates on two *operands* and returns a Boolean value
- Called "relational operators" since they test how the operands relate to each other
- Return a Boolean value.

Conditional operators

Boolean expressions can be combined using conditional operators, of which Java offers three:

Operator	Operation
`!`	Logical **not**
`&&`	Logical **and**
`||`	Logical **or**

Examples:

```
!(tailIsDown() == true)

! tailIsDown()

tailIsDown() && (n < 100)

(city == "Tel Aviv") || (city == "Haifa")

!dubim && !yaar
```

Each operator operates on one or two Boolean operands and returns a Boolean result.

Truth tables

NOT (!)

a	!a
false	true
true	false

- A "truth table" shows the output of a Boolean function for every possible input
- The truth table can be viewed as the definition of the Boolean function.

AND (&&)

a	b	a && b
false	false	false
false	true	false
true	false	false
true	true	true

OR (||)

a	b	a || b
false	false	false
false	true	true
true	false	true
true	true	true

- `!a` is true if `a` is false and false otherwise
- `(a && b)` is true if both `a` and `b` are true and false otherwise
- `(a || b)` is true if either `a` or `b` or both are true and false otherwise.

Boolean expressions and Boolean variables in action

Boolean expressions can be made as complex as necessary:

```
if  (!((a > 0) && ((a % 2) == 0)))
     System.out.println("The input must be a positive even number!");
```

```
// Determines if a given triangle is right.

class RightTriangle {

    public static void main(String[] args) {
        // Get the triangle's three edges a,b,c  (omitted)

        boolean rightTriangle = (a*a + b*b == c*c) ||
                                (a*a + c*c == b*b) ||
                                (b*b + c*c == a*a);

        if (rightTriangle)
            System.out.println("It's a right triangle");
        else
            System.out.println("It's not a right triangle");
    }
}
```


The conditional operator: a shorthand if/else mechanism

Example

```
int max = (a > b) ? a : b;
```

Equivalent to:

```
int max;
if (a > b) max = a; else max = b;
```

Syntax

```
condition ? expression1 : expression2 ;
```

- If the ***condition*** is true, the expression returns the value of ***expression1***; else, the expression returns the value of ***expression2***
- Similar to an `if/else` statement
- However, the conditional is not a statement; it's an expression that can returns a value

Example

```
Systems.out.println("Thanks! you bought " + count +
                    ((count == 1) ? "item" : "items");
```

Control structures: lecture outline

- `If`
- Boolean expressions and variables
- `While`
- `switch`
- `do`
- `for`

Example

```
// Counts from 1 to n
int count = 0;
while (count < n) {
  System.out.println(count);
  count++;
}
```

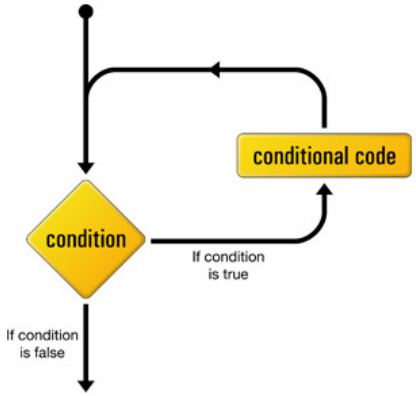
What will the program print when . . .

- `n = 5` ?
- `n = 1` ?
- `n = 0` ?
- `n = -3` ?

The while statement

Syntax:

```
while (condition)
   conditional code
```



A `while` loop executes zero or more times.

Example: factors

```
// Prints the factors of a given integer
public static void main (String args[]) {
   Scanner scan = new Scanner(System.in);
   System.out.print("Enter a number: ");
   int n = scan.nextInt();
   System.out.println("The divisors of " + n + " are:");

   int i = 1;
   while (i <= n) {
       if (n % i == 0)
           System.out.println(i);
       i++;
   }

}
```

Output:

```
Enter a number: 12
The divisors of 12 are:
1
2
3
4
6
12
```

Example: input testing

```
public static void main (String args[]) {
   final int ITEM_PRICE = 100;
   Scanner scan = new Scanner(System.in);
   // Gets an input and makes sure it's non-negative
   System.out.print("Enter number of items: ");
   int n =  scan.nextInt();

   while (n <= 0) {
       System.out.println("Number of items must be positive!");
       System.out.print("Enter number of items: ");
       n =  scan.nextInt();
   }

   System.out.println("The total price is: " + (n * ITEM_PRICE));
}
```

Output:

```
Enter number of items: -10
Number of items must be positive!
Enter number of items: -2
Number of items must be positive!
Enter number of items: 0
Number of items must be positive!
Enter number of items: 5
The total price is: 500
```

This code is "bullet-proof":
there is no way to enter an invalid input.

Example: palindrome tester

- radar
- kayak
- drab bard
- able was I ere I saw elba
- Dennis and Edna sinned
- Rise to vote, sir
- Doom an evil deed, liven a mood
- Go hang a salami; I'm a lasagna hog
- A man, a plan, a canal, Panama

```
import java.util.Scanner;

public class PalindromeTester {
  public static void main (String[] args) {

    Scanner scan = new Scanner (System.in);
    System.out.print("Enter some text: ");
    String txt = scan.nextLine();

    int left = 0;
    int right = txt.length() - 1;
    while (txt.charAt(left) == txt.charAt(right) && left < right) {
      left++;    // shorthand notation for left = left + 1;
      right--;   // shorthand notation for right = right - 1;
    }

    if (left < right)
      System.out.println ("The text is NOT a palindrome.");
    else
      System.out.println ("The text IS a palindrome.");
   }
}
```

Control structures: lecture outline

- `if`
- Boolean expressions and variables
- `while`
- `switch`
- `do`
- `for`

The switch statement

Syntax

```
switch expression {
  case value1:
       statement1
  case value2:
       statement2
  ...
  default:
       statement
}
```

- The expression value is compared to each case value; if there's a match, the corresponding statement is executed
- If none of the values matches, the default statement is executed
- `default` is optional
- The types of the expression and the values must be either integral or character
- The flow of control "falls through" all the cases, even after a match; Can be terminated with the `break` keyword.

Example (typical ATM application)

```
final int WITHDRAWAL = 1;
final int DEPOSIT = 2;
final int BALANCE = 3;
// ...

// The user selects some banking operation, which is represented in
// this program by an int variable named "operation" that can be
// either 1, 2, or 3.

switch (operation) {
    case WITHDRAWAL:
         doWithdrawal();
   case BALANCE:
         showBalance();
   case DEPOSIT:
         doDeposit();
}
```


Example (from the Java tutorials)

```
// Returns the number of days for a given month / year

switch (month) {
    case 1:
    case 3:
    case 5:
    case 7:
    case 8:
    case 10:
    case 12:
          numDays = 31;
          break;
    case 4:
    case 6:
    case 9:
    case 11:
          numDays = 30;
          break;
    case 2:
          if ( ((year % 4 == 0) && !(year % 100 == 0)) || (year % 400 == 0) )
              numDays = 29;
          else
              numDays = 28;
          break;
    default:
          System.out.println("Invalid month.");
          break;
}
```

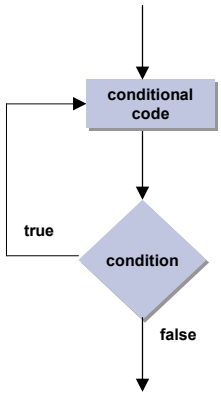
Control structures: lecture outline

- `If`
- Boolean expressions and variables
- `While`
- `switch`
- `do`
- `for`

The do statement

Syntax:

```
do
  conditional code
while (condition);
```

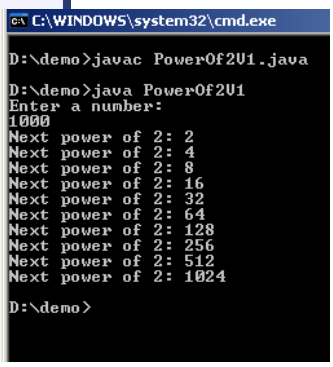


Similar to `while`, except that the condition evaluates at the end of the loop

- `while` loop executes 0 or more times
- `do` loop executes 1 or more times.

Example

```
// Prints the powers of 2 up to a limit
...
// Get the value of limit somehow
...
int n = 1;
do {
   n = n * 2;
   System.out.println("Next power of 2: " + n);
} while (n <= limit);
```



The for statement

Syntax:

```
for (initialization ; condition ; increment)
    conditional code;
```

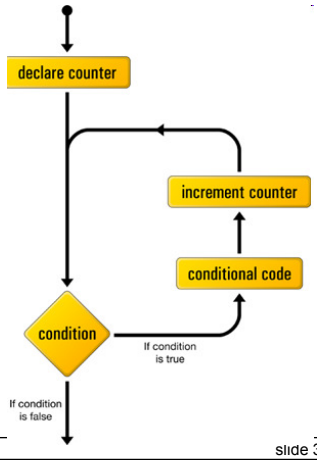
Equivalent to:

```
initialization
while condition {
    statement;
    increment;
}
```

- Many loops have this common pattern
- Comes to play when the number of iterations is known in advance.

Example:

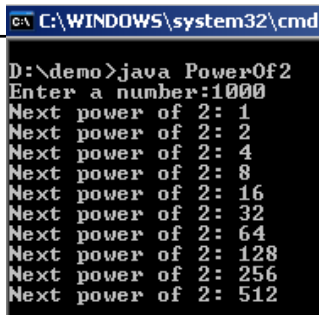
```
for (int count = 0; count <= 10; count++) {
      System.out.println (count);
   }
```



Example

Prints the powers of 2 up to a limit

```
for(int n = 1; n <= limit; n = n * 2) {
  System.out.println("Next power of 2: " + n);
}
```



Equivalent to:

```
do {
  n = n * 2;
  System.out.println("Next power of 2: " + n);
} while (n <= limit);
```

- `for` yields tighter and clearer code than `do` and `while`, and is preferred when suitable.

The for statement

`for` loop without initialization

```
for ( ; condition ; increment )
  statement;
```

- Any one of the three expressions in the `for` header is optional
- The two semicolons are always required.

Example:

```
for (;;) { // an infinite loop
  System.out.println ("beep");
}
```

`for` loop without increment:

```
for ( initialization; condition ; )
  statement;
```

Infinite `for` loop:

```
for ( initialization; ; increment )
  statement;
```

Example

```
// 10 by 10 multiplication table
class MultiplicationTable {
  public static void main(String[] args) {
    for(int j = 1 ; j <= 10 ; j++) {
      for(int k = 1 ; k <= 10 ; k++)
        System.out.print(j * k);
      System.out.println();
    }
  }
}
```


break and continue

- `break` can be used to terminate a loop; Execution continues after the loop's end
- `continue` can be used to terminate the current iteration;
  Execution continues by jumping to evaluate the loop's condition

```
// Reads a series of positive numbers from the user, until
// 0 is read. Computes and prints the series' average.
class Average {
  public static void main(String[] args){
    Scanner scan = new Scanner(System.in);
    int x, sum = 0; count = 0;
    while(true) {
      System.out.print("Enter a positive number:");
      x = scan.nextInt();
      if (x == 0) break;
      if (x < 0) {
        System.out.println("Only positive numbers!");
        continue;
      }
      sum += x;
      count++;
    }
    System.out.println("The average is " + (sum/count));
  }
}
```

Ugly code . . .

ERROR: undefined
OFFENDING COMMAND:

STACK: