AP CSA

zhang si 张思

zhangsi@rdfz.cn

ICC 609

2-Control Structures

- **Control structures in CS**
  - Selection-Unit 3
  - Iteration -Unit 4
  - sequencing
- **Evaluating expressions**
  - Boolean expressions
  - Operator proderce

Control Structures

- Control structures are the mechanism by which you make the statements of a program run in a nonsequential order.
- Unless specified otherwise, the order of statement execution through a method is linear: one statement after the other in sequence
- Some programming statements modify that order, allowing us to:
    - Selection-decide whether or not to execute a particular statement, or
    - Iteration-perform a statement over and over, repetitively
- These decisions are based on a **boolean expression** (also called a condition) that evaluates to true or false
- The order of statement execution is called the flow of control

Selection

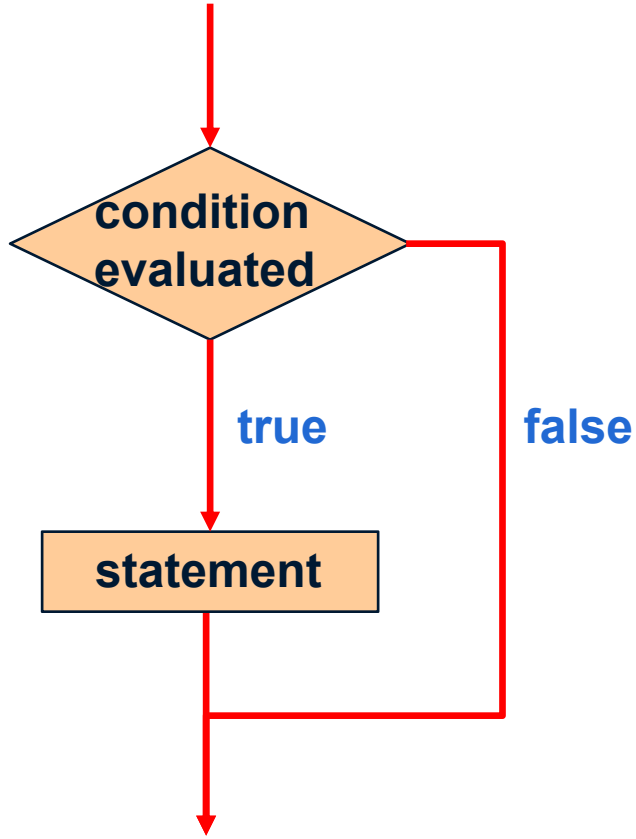
- A one-way selection (if statement)

is a Java reserved word

The *condition* must be a boolean expression. It must evaluate to either true or false.

```
if ( condition )
   statement;
```

If the *condition* is true, the *statement* is executed.
If it is false, the *statement* is skipped.



flow chart

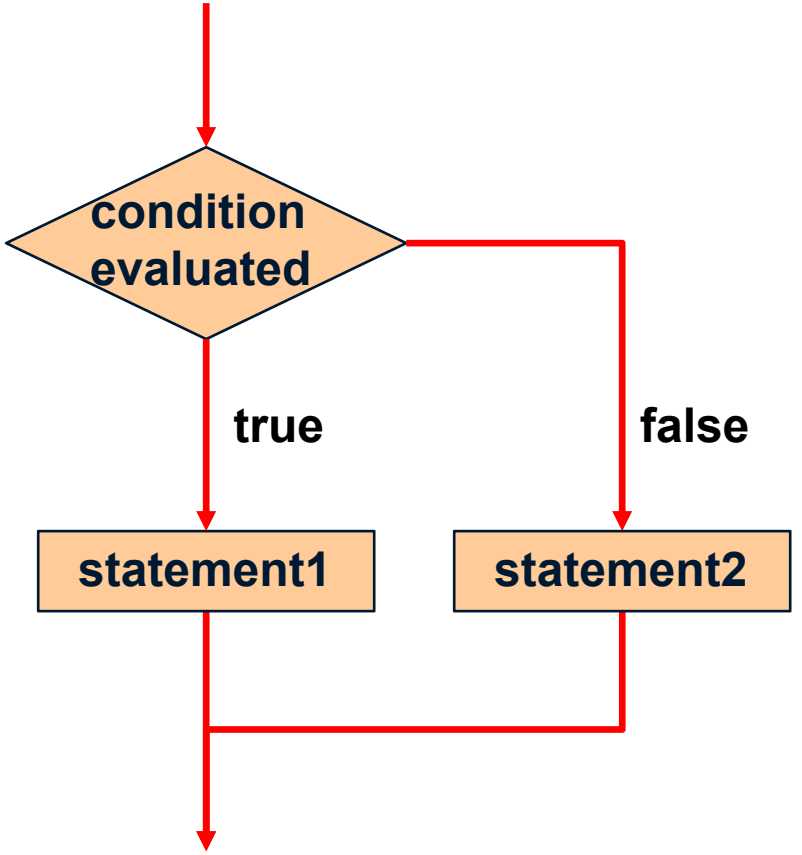
Selection

- A two-way selection (*if-else statement*)

```
if ( condition )
   statement1;
else
   statement2;
```

If the *condition* is true, *statement1* is executed; if the *condition* is false, *statement2* is executed

One or the other will be executed, but not both

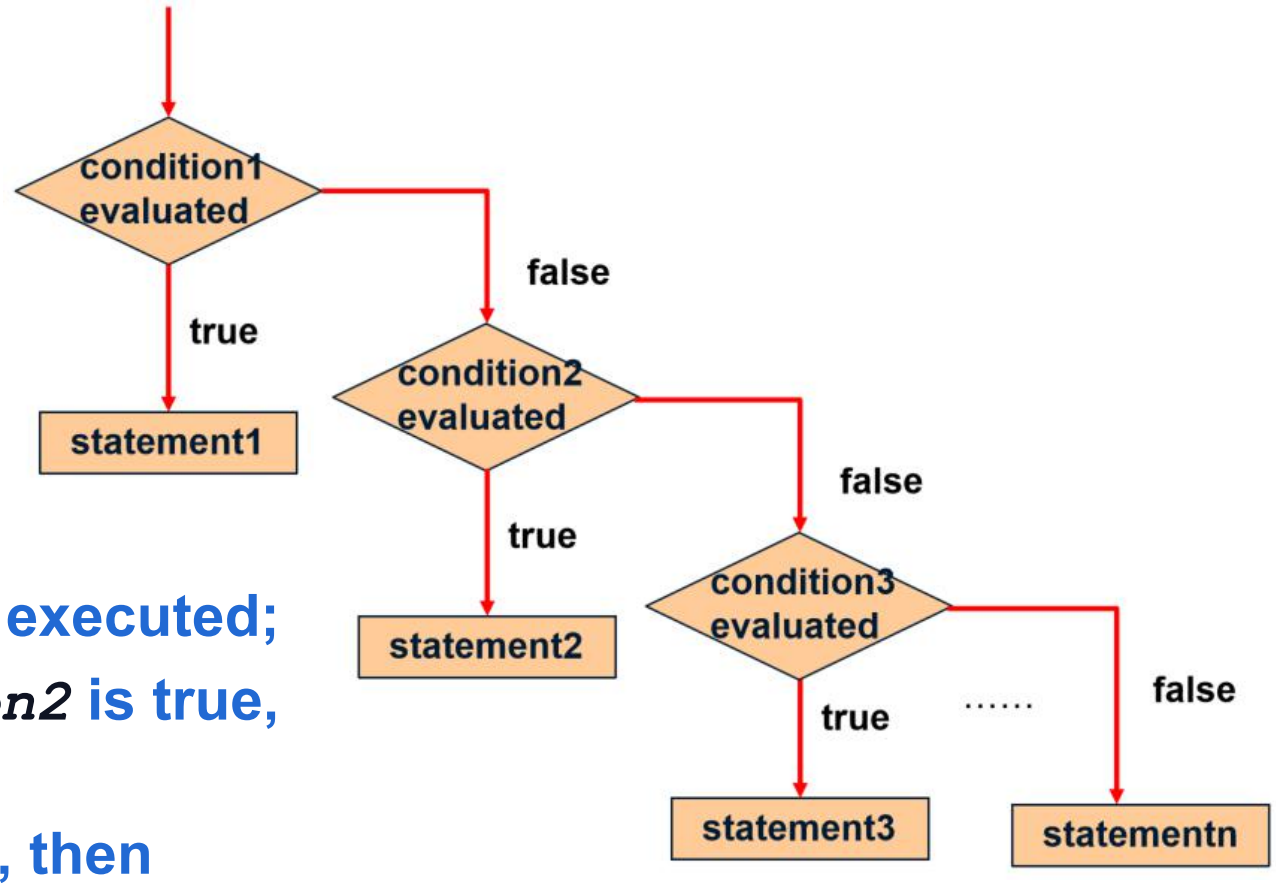


flow chart

Selection

- A multi-way selection (*if-else if -else statement*)

```
if ( condition1 )
    statement1;
else if (condition2)
    statement2;
else if (condition3)
    statement3;
        ……
else
    statementn;
```



flow chart

- ✓ If the *condition1* is true, *statement1* is executed;
- ✓ if the *condition1* is false and *condition2* is true, *statement2* is executed ……
- ✓ if *all of the condition* above is false , then *statementn* is executed

Selection

- **Block Statements**
  - using braces { … } to group statements together into a *block statement*
- **Nested if Statements**
  - The statement executed as a result of an **`if`** statement or **`else`** clause could be another statement
  - An **`else`** clause is matched to the last unmatched **`if`** (no matter what the indentation implies)
  - Braces can be used to specify the **`if`** statement to which an **`else`** clause belongs

```
if ( condition1 )
      if(condition2)
          statement1;
else statement2;
```

V.S.

```
if ( condition1 )
     {if(condition2)
         statement1;}
else statement2;
```

Iteration

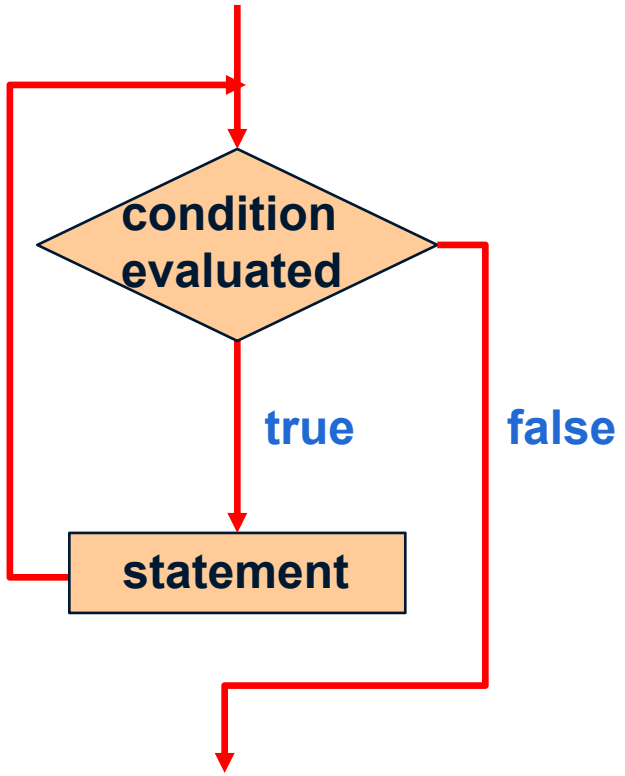
- **while loop:** When you can't determine how many times you want to execute the loop body, use a `while` statement

while is a reserved word

```
while ( condition )
    statement;
```

If the *condition* is true, the *statement* is executed.
Then the *condition* is evaluated again.

The *statement* is executed repeatedly until the *condition* becomes false.



flow chart

Iteration

- **for loop:** If you can determine how many times you want to execute the loop body, use a `for` statement

Reserved word

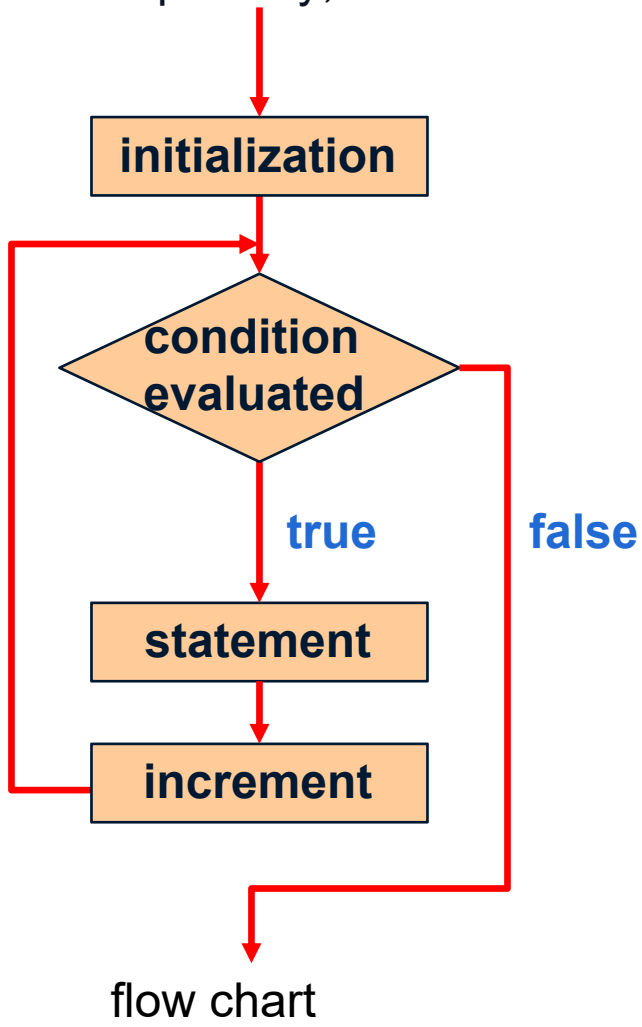
The *initialization* is executed once before the loop begins

The *statement* is executed until the *condition* becomes false

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

The *increment* portion is executed at the end of each iteration

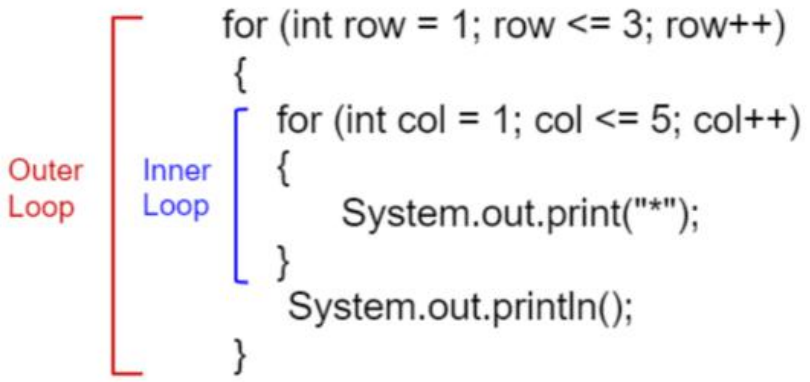
The *condition-statement-increment* cycle is executed repeatedly



flow chart

Iteration

- Infinite Loops
  - The body of a `while` loop eventually must make the condition false.If not, it is an ***infinite loop***, which will execute until the user interrupts the program
  - This is a **common logical error**,You should always double check to ensure that your loops will terminate normally
- Nested Loops
  - Similar to nested if statements, loops can be nested as well
  - That is, the body of a loop can contain another loop
  - Each time through the outer loop, the inner loop goes through its full set of iterations



Result:

```
*****
*****
*****
```

break and continue statement

- break statement: end the loop immediately

```
public class HelloWorld
{

    public static void main(String[]
args){
        System.out.println("Loop starts!");
        for(int a=0; a<10;a++){

            if(a==5){
            break;
            }
       System.out.println(a);
        }
        System.out.println("Loop is end!");
    }
}
```

- continue statement: the loop condition is evaluated again, and the loop body is executed again if it is still true.

```
public class HelloWorld
{

    public static void main(String[]
args){
        System.out.println("Loop starts!");
        for(int a=0; a<10;a++){

            if(a==5){
            continue;
            }
       System.out.println(a);
        }
        System.out.println("Loop is end!");
    }
}
```

Boolean Expressions

- A condition often uses one of Java's *equality operators* or ***relational operators***, which all return **boolean results**:

Operator	Meaning	Example
==	equal to	`if (x == 100)`
!=	not equal to	`if (age != 21)`
>	greater than	`if (salary > 30000)`
<	less than	`if (grade < 65)`
>=	greater than or equal to	`if (age >= 16)`
<=	less than or equal to	`if (height <= 6)`

Relational operators

- Note the difference between the equality operator (==) and the assignment operator (=)
- Relational operators should generally be used only in the comparison of primitive types (i.e., int, double, or boolean).
- Do not routinely use == to test for equality of floating-point numbers.

Logical operators

- Boolean expressions can use the following ***logical operators***:

Operator	Meaning	Example
!	NOT	`if (!found)`
&&	AND	`if (x < 3 && y > 4)`
||	OR	`if (age < 2 || height < 4)`

logical operators

&&	T	F
T	T	F
F	F	F

||	T	F
T	T	T
F	T	F

!	
T	F
F	T

truth tables

- **Short-circuit evaluation**.
  - The subexpressions in a compound boolean expression are evaluated from left to right, and evaluation automatically stops as soon as the value of the entire expression is known.
  - eg: expression A || B, where A and B are some boolean expressions. If A is true, then the expression is true irrespective of the value of B.

Logical Operators

- **De Morgan's Laws** can be applied to Boolean expressions.
- Equivalent Boolean expressions will evaluate to the same value in all cases.

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

- Questions:

What is printed when the following code executes and `x` equals `4` and `y` equals `3`?

```
if (!(x < 3 || y > 2)) System.out.println("first case");
else System.out.println("second case");
```

- ○ `first case`
- ○ `second case`

Operators precedence

Operator Precedence

highest precedence	→	(1)	`!, ++, --`	→ unary operators
		(2)	`*, /, %`	} Arithmatic operators
		(3)	`+, -`	
		(4)	`<, >, <=, >=`	} Relational operators
		(5)	`==, !=`	
		(6)	`&&`	} Logical operators
		(7)	`||`	
lowest precedence	→	(8)	`=, +=, -=, *=, /=, %=`	→ Assignment operators

Remeber:

1.parentheses () is the highest in the order of operations

2.casting have higher precedence than * / % recall:(double) 3/4=0.75