Accelerating Information Technology Innovation

http://aiti.mit.edu

Cali, Colombia
Summer 2012
Lección 03 – Control Structures
Agenda

1. Block Statements

2. Decision Statements

3. Loops
What are Control Structures?

• Without control structures, a computer would evaluate all instructions in a program sequentially

• Allow you to control:
  – the order in which instructions are evaluated
  – which instructions are evaluated
  – the “flow” of the program

• Use pre-established code structures:
  – block statements (anything contained within curly brackets)
  – decision statements (if, if-else, switch)
  – Loops (for, while)
Block Statements

• Statements contained within curly brackets

```java
{
    statement1;
    statement2;
}
```

• Evaluated sequentially when given instruction to “enter” curly brackets

• Most basic control structure (building block of other control structures)
The “if” decision statement causes a program to execute a statement conditionally*

    if (condition) {
        statement;
    }

    next_statement;

*Executes a statement when a condition is true
Dissecting if-then

```java
if (condition) {
    statement;
}
next_statement;
```

- The `condition` must produce either `true` or `false`, also known as a `boolean` value.

- If `condition` returns `true`, `statement` is executed and then `next_statement`.

- If `condition` returns `false`, `statement` is not executed and the program continues at `next_statement`.
if (condition) {
    statement;
}
next_statement;

*if-then* Statement Flow Chart
if-then Example

```java
int price = 5;
if (price > 3) {
    System.out.println("Too expensive");
}
//continue to next statement
```

Output:

Too expensive
if-then-else Statements

- The basic “if” statement can be extended by adding the “else” clause in order to do something if expression is false
  ```java
  if (condition) {
    statement1;
  }
  else {
    statement2;
  }
  next_statement;
  ```

- Again, the condition must produce a boolean value

- If condition returns true, statement1 is executed and then next_statement is executed.

- If condition returns false, statement2 is executed and then next_statement is executed.
if (condition){
    statement1;
}
else {
    statement2;
}
next_statement;

execute statement1

execute statement2

execute next_statement

condition TRUE?

yes

no
if-then-else Example

```java
int price = 2;

if (price > 3) {
    System.out.println("Too expensive");
}
else {
    System.out.println("Good deal");
}
//continue to next statement
```

Output:

Good deal
Chained if-then Statements

- Note that you can combine if-else statements below to make a chain to deal with more than one case

```java
if (grade == 'A')
    System.out.println("You got an A.");
else if (grade == 'B')
    System.out.println("You got a B.");
else if (grade == 'C')
    System.out.println("You got a C.");
else
    System.out.println("You got an F.");
```
Chained \texttt{if-then-else} Statement Flow Chart

```java
if (condition1) {
    statement1;
} else if (condition2) {
    statement2;
} else if (condition3) {
    statement3;
} else {
    statement\_else;
} 
next\_statement;
```
switch Statements

• The `switch` statement is another way to test several cases generated by a given expression.

• The expression must produce a result of type `char`, `byte`, `short` or `int`, but not `long`, `float`, or `double`.

```java
switch (expression) {
    case value1:
        statement1;
        break;

    case value2:
        statement2;
        break;

    default:
        default_statement;
        break;
}
```

• The `break;` statement exits the switch statement
switch (expression) {
  case value1:
    // Do value1 thing
    break;
  case value2:
    // Do value2 thing
    break;
  ...
  default:
    // Do default action
    break;
}
// Continue the program
Remember the Example…

Here is the example of chained if-else statements:

```java
if (grade == 'A')
    System.out.println("You got an A.");

else if (grade == 'B')
    System.out.println("You got a B.");

else if (grade == 'C')
    System.out.println("You got a C.");

else
    System.out.println("You got an F.");
```
Chained if-then-else as switch

- Here is the previous example as a switch

```java
switch (grade) {
    case 'A':
        System.out.println("You got an A.");
        break;
    case 'B':
        System.out.println("You got a B.");
        break;
    case 'C':
        System.out.println("You got a C.");
        break;
    default:
        System.out.println("You got an F.");
}
```
What if there are no break statements?

- Without break, switch statements will execute the first statement for which the expression matches the case value AND then evaluate all other statements from that point on.

- For example:

```java
switch (expression) {
    case value1:
        statement1;

    case value2:
        statement2;

    default:
        default_statement;
}
```

- NOTE: **Every statement** after the true case is executed.
switch (expression) {
    case value1:
        // Do value1 thing
    case value2:
        // Do value2 thing
    ... 
    default:
        // Do default action
}
// Continue the program

Switch Statement Flow Chart w/o breaks
Loops

- A loop allows you to execute a statement or block of statements repeatedly.

- There are 4 types of loops in Java:
  1. while loops
  2. do-while loops
  3. for loops
  4. foreach loops (coming soon!)
The **while** Loop

```java
while (condition) {
    statement
}
```

- This **while** loop executes as long as `condition` is true. **When condition is false, execution continues with the statement following the loop block.**

- The condition is tested at the beginning of the loop, so if it is initially `false`, the loop will not be executed at all.
while Loop Flow Chart

while (expression) {
    statement
}

Test condition is true?

The while loop

Execute loop statement(?)

Next statement
Example

```c
int limit = 4;
int sum = 0;
int i = 1;

while (i < limit){
    sum += i;
    i++;
}
```

- What is the value of `sum`?
  6

```
i = 1   sum = 1
i = 2   sum = 3
i = 3   sum = 6
i = 4
```
do-while Loops

- Similar to while loop but guarantees at least one execution of the body

```c
do {
    statement;
} while(condition)
```
do-while Flowchart

do {
    statement;
}
while(condition)
next_statement;

execute statement

condition true?

execute next_statement
do-while Example

boolean test = false;

do {
    System.out.println("Hey!");
} while(test)

Output:
Hey!
**for Loop**

- Control structure for capturing the most common type of loop

```plaintext
i = start;
while (i <= end)
{
    ...
    i++;
}
```

```plaintext
for (i = start; i <= end; i++)
{
    ...
}
```
Dissecting the `for` Loop

```
for (initialization; condition; update)
{
    statement;
}
```

The control of the `for` loop appear in parentheses and is made up of three parts.

1. The first part, the `initialization`, sets the initial conditions for the loop and is executed before the loop starts.

2. Loop executes so long as the `condition` is true and exits otherwise.

3. The third part of the control information, the `update`, is used to increment (update) the loop counter. This is executed at the end of each loop iteration.
for Loop Flow Chart

The for loop

for (initialization;
   condition == true;
   update)
{
   //statements
}
next_statement;
Example

```java
int limit = 4;
int sum = 0;

for(int i = 1; i<=limit; i++ )
{
    sum += i;
}
```

- What is the value of `sum`?
  - 10
Another Example

for ( int div = 0; div < 1000; div++ ) {

    if ( div % 12 == 0 ){

        System.out.println(div+"is divisible by 12");

    }
}

• This loop will display every number from 0 to 999 that is evenly divisible by 12.
Other Possibilities

• If there is more than one variable to set up or increment they are separated by a comma.
  
  ```java
  for (i=0, j=0; i*j<1000; i++, j+=2) {
      System.out.println(i+"*"+j+"="+i*j);
  }
  ```

• You do not have to fill every part of the control of the for loop but you must still have two semi-colons.
  
  ```java
  for (int i=0; i<100; ) {
      sum += i;
      i++;
  }
  ```

  *Straying far from convention may make code difficult to understand and thus is not common*
Using the `break` Statement in Loops

- We have seen the use of the `break` statement in the switch statement.
- In loops, you can use the `break` statement to exit the current loop you are in. Here is an example:

```java
int index = 0;
while (index <= 4) {
    index++;
    if (index == 3)
        break;
    System.out.println("The index is "+index);
}
```

The index is 1
The index is 2
The index is 3
Using the `continue` Statement in Loops

- Continue statement causes the loop to jump to the next iteration
- Similar to break, but only skips to next iteration; doesn’t exit loop completely

```java
int index = 0;
while (index <= 4) {
    index++;
    if (index == 3)
        continue;
    System.out.println("The index is " + index);
}
```

```
index = 1   The index is 1
index = 2   The index is 2
index = 3   -- --
Index = 4   The index is 4
```
Nested Loops – Example

- Printing a triangle

```java
for (int i=1; i<=5; i++){
    for (int j=1; j<=i; j++){
        System.out.println("*");
    }
}
```

* * * * *
You are withdrawing money from a savings account. How do you use an If Statement to make sure you do not withdraw more than you have?

```java
if ( amount < balance )
{
    balance = balance - amount;
}
//next statement
```
Which Control Structure?

- As a programmer, you will never be asked something like: “Write a for loop to…”

- You will need to implement logic in your program that meets your specification and requirements

- With experience, you will know which control structure to use.