HBO Graduaat Informatica Optie Programmeren

Java Basics

Control structures

cvo leerstad

volwassenenonderwijs

Flow Control

- A flow control statement evaluates an expression and executes another statement as a consequence
- A boolean expression must evaluate to `true` or `false`
- Use braces, `{` and `}`, to group one or more statements into a *block*
  - A block of statements is allowed wherever a single statement is allowed
  - Sometimes called a *body* (e.g. in loops)
- Any variable declared within a block remains in scope only until the next closing brace

Flow Control: The `if` Statement

- Simplest form of conditional processing
- Syntax:

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

```
if ( boolean_expression ) {
   statement1;
   statement2;
} else {
   statement3;
   statement4;
}
```

use parentheses to group statements

Flow Control: The `if` Statement

- Example

```
if (c < b){
   a = b;
   b = c;
   c = a - b;
 } else if (c > b) {
   a = c;
   c = a - b;
 } else {
   solution = b;
 }
```

- There is no elseif statement (like in PL/SQL, Perl, ...)
- A common coding error is to use the assignment operator (=) rather than the equality operator (==) in the control expression!

Flow Control: The `if` Statement

Quiz: what does this code display?

- A
- B C
- Something else

```
int a = 4, b = 7;
if (a < b)
    System.out.print("A ");
else
    System.out.print("B ");
    System.out.print("C ");
```

Flow Control: The `if` Statement

- You can nest `if` statements, but be careful.
- Combining `if` and `else` (`else if`) is recommended

- Nesting: not generally recommended because you have to maintain a mental stack of the decisions being made, which becomes difficult once if statements are nested to a level of three or more.
- Also, it is very easy to forget that the statement following an else always binds to the immediately preceding if that is not already matched with an else (even if the indentation suggests otherwise!)!

Flow Control: The `if` Statement

- What if... you remove the first `else`?

```
if (speed > 30)
    if (speed > 70)
        System.out.println("Speed is over 70");
    else
        System.out.println("Speed is over 30");
else
    System.out.println("Speed is under 30");
```

```
if (speed > 70)
    System.out.println("Speed is over 70");
else if (speed > 30)
    System.out.println("Speed is over 30");
else
    System.out.println("Speed is under 30");
```

Flow Control: The Conditional Operator ? :

- Can be useful alternative to `if ... else ...`
- Syntax

```
(boolean_expression) ? expression1 : expression2
```

  - If `boolean_expression` evaluates to `true`, the result of the `?:` operator is the value of `expression1`
  - Otherwise, it is the value of `expression2`

`expression1` and `expression2` must have compatible types

Flow Control: The Conditional Operator ? :

- Example

```
if (n >= 0) {

  String s1 = (n > 1) ? "are " : "is ";
  System.out.print("There " + s1);

  String s2 = (n > 1) ? "es" : "";
  System.out.print(n + " box" + s2);

} else {

  System.out.println("Enter a non-negative number");

}
```

Flow Control: The `switch` statement

- Select from a number of alternatives
- Syntax:

```
switch ( integer_expression or String)
{
    case constant_expression1:
        statement1;
        break;
    case constant_expression2:
        statement2;
        break;
    default:
        statement3;
        break;
}
```

1. First, the `integer_expression or String` is evaluated
2. Then, control passes to the `case` label that matches
3. Flow of execution continues until a `break` statement is reached (or the end of the `switch` block)
4. If no `case`s match, control passes to the optional `default` label
5. If no default label, entire switch statement is skipped

- Note that the statement will execute *zero* or more times
- Don't forget to update the loop counter (i)

Flow Control: The `switch` statement

- The `switch` statement compares the `String` object in its expression with the expressions associated with each `case` label as if it were using the `String.equals` method; consequently, the comparison of `String` objects in `switch` statements is case sensitive. The Java compiler generates generally more efficient bytecode from `switch` statements that use `String` objects than from chained `if-then-else` statements.

Flow Control: The `switch` statement

- labels must be known at compile time
  - Use literals or `final` variables (see later chapter)
- a label does *not* force a break out of a switch statement
  - Insert `break` statements to skip to end of switch statement
- the `default` label must not necessarily be the last label!

Flow Control: The `while` Loop

- Simplest form of loop:

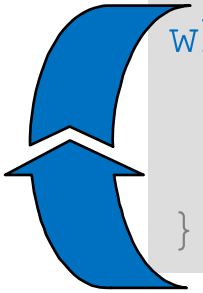
```
while (boolean_expression)
    statement;
```

```
while (boolean_expression) {
    statement1;
    statement2;
    ...
}
```

  - If `boolean_expression` evaluates to `true`, the statement is executed. Then, `boolean_expression` is re-evaluated
  - The statement is executed repeatedly until the `boolean_expression` evaluates to `false`

Flow Control: The `while` Loop

- Example



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

- Note that the statement will execute *zero* or more times
- Don't forget to update the loop counter (i)

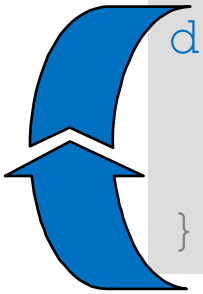
Flow Control: The `do while` Loop

- Alternative form of loop
  - Similar to `while` loop except that `boolean_expression` is evaluated after statement is executed
  - Loop executes at least once

```
do {
    statement1;
    statement2;
    ...
} while (boolean_expression);
```

Flow Control: The `do while` Loop

- Example

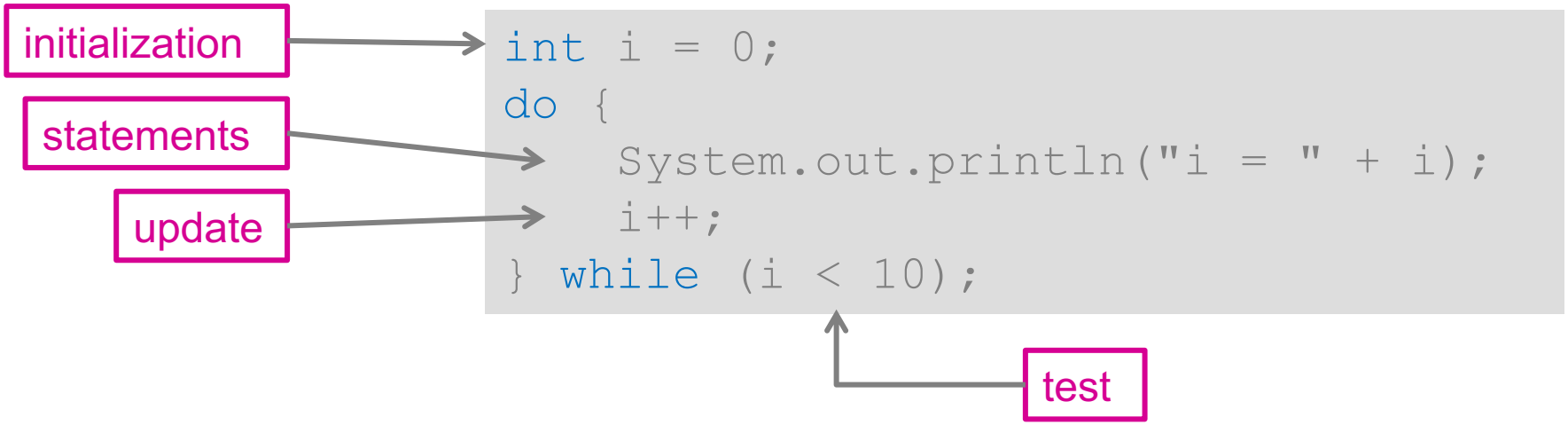


```
int i = 0;
do {
    System.out.println("i = " + i);
    i++;
} while (i < 10);
```

- Note that the statement will execute *one* or more times
- Don't forget to update the loop counter (i)

Flow Control: The Components of a Loop

- The `while` and `do while` loops have four components:



- Don't forget the initialization!
- Don't forget to update the loop counter

Flow Control: The `for` Loop

- Versatile form of while loop

```
for (init_expr; boolean_expr; update_expr)
    statement;
```

```
for (init_expr; boolean_expr; update_expr) {
    statement1;
    statement2;
}
```

- Example:

```
int counter;  // loop counter must be declared
for (counter = 0; counter < 10; counter++)
    System.out.println(counter);
```

Flow Control: More on the `for` Loop

- Initialisation can be replaced by a declaration:

```
for (int counter = 0; counter < 10; counter++)
    System.out.println(counter);
```

- You can use more than one counter

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

Flow Control: More on the `for` Loop

- The body of the loop can even be empty:

```
int sum = 0;
for (int i = 0; i < 10; sum += i++)
{}  // note the empty curly braces
```

- Never-ending loop

```
for (;;) {
    System.out.println(".");
}
```

Any of the components of the `for` loop can be omitted

Flow Control: The `break` Statement

- Can be used to 'break out' of a loop or `switch` statement
  - Transfers control to the first statement following the loop body or switch statement
  - Not strictly necessary, but can simplify code in certain situations

Flow Control: The `break` Statement

- Example

```
...
while (age <= 65) {
    balance = (balance+payment) * (1 + interest));
    if (balance >= 250000)
        break;
    age++;
}
...
```

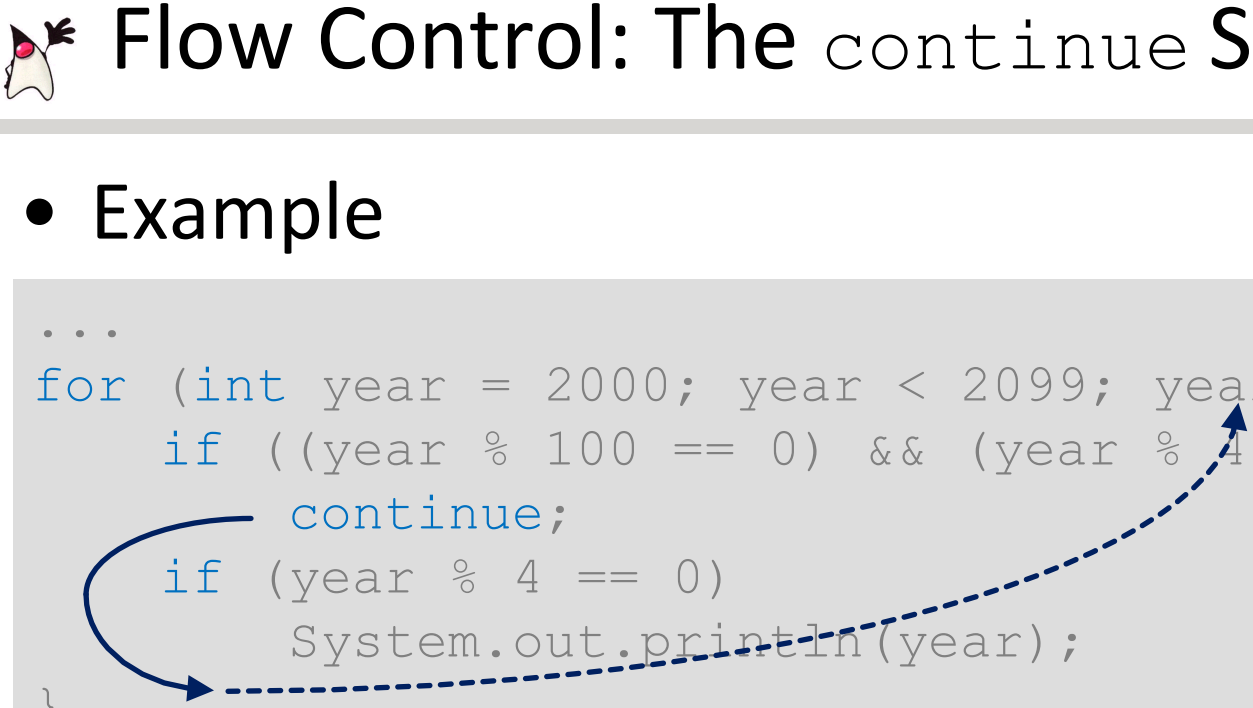
Flow Control: The `continue` Statement

- Can only be used in loops
  - Terminates the current iteration
  - Causes next iteration of loop
  - Useful to skip over a value or range of values within loop

Flow Control: The `continue` Statement

- Example

```
...
for (int year = 2000; year < 2099; year++) {
    if ((year % 100 == 0) && (year % 400 != 0))
        continue;
    if (year % 4 == 0)
        System.out.println(year);
}
...
```



Flow Control: Labeled `break` and `continue`

- Used to break out of nested loops or continue a loop outside the current loop

```
...
outer_loop:
for (int i = 0; i < 10; i++) {
    for (int j = 0; j < 5; j++) {
        System.out.println(i, j);
        if (i + j > 8)
            break outer_loop;
    }
}
...
```

`break outer_loop` transfers control to the *end* of the loop labelled `outer_loop`

Flow Control: The `for` Loop and Visibility

- Visibility of variables declared in the initializer:
  - Local variables have a scope of the `for` statement

```
...
for(int i = 0;i<6;i++){
 System.out.println(i);
}
int i = 3;                    // OK, no overlap
...
```

```
...
int i = 3;                    // Scopes would overlap
for(int i = 0;i<6;i++){       // Error, cannot declare
 System.out.println(i);       // variable twice
}
...
```

Flow Control: The `for/in` Statement

There is a special syntax for looping over arrays and collections of objects

```
...
int[] primes = new int[] {2, 3, 5, 7, 11, 13, 17};
for (int pr : primes) {
    System.out.println(pr);
}
...
```

See collections

Methods

- A *method* is equivalent to a function or subroutine in other languages
- A method can only be defined within a class definition

Methods

- Syntax:

```
modifier returnType methodName(argumentList) {
    // method body
    . . .
}
```

- The key components of a method are:
  - A *modifier* such as `public` or `private` (as well as `static` and `final`)
  - If the method returns a value, its primitive type or class; otherwise specify `void`
  - The name of the method
  - An optional *argument list* inside parenthesis
  - The method body, which contains the statements that are executed when the method is called

Methods arguments

- In the method definition:
  - The argument list contains zero or more *formal arguments* separated by commas and enclosed ()
  - Each formal argument is similar to a variable declaration
  - A formal argument can be a primitive type or an object reference
- When the method is called:
  - Each formal argument becomes a *local variable* of the method
    - Not visible outside the method
  - Each variable is initialized with the value specified in the call

Methods arguments

- Primitive arguments are passed by copy, objects are passed by reference

```
public boolean checkPassword(String password){
    try {
      String pwdMD = DAOAccount.getInstance().getPwdMD(this);
      MessageDigest md = MessageDigest.getInstance("SHA");
      String passwordString =
          Base64.encodeBytes(md.digest(password.getBytes()));
      return pwdMD.equals(passwordString);
    } catch (Exception e){
        e.printStackTrace();
    }
    return false;
}
```

Returning a Value from a Method

- In the method definition:
  - If a return type other than `void` is specified, body must `return` a value or expression of the appropriate type
  - If a return type of `void` is specified, the method automatically returns at the closing brace of its body
  - Can exit body prematurely using an explicit `return` (with no value)

Returning a Value from a Method

- When the method is called:
  - The method can be used in the same way as any other expression, or
  - The return value can be ignored

```
public boolean checkPassword(String password){
    if ("Peter".equals(password))
        return true;
    else
        return false;
}
```

Only static methods can be called from static ones
(More on this in the other chapters)

Summary (1)

- Variables, assignments, expressions and operators are the fine grains that allow to build a program.
- Primitive types are also important, even in an object-oriented programming language such as Java; operations on these primitives are so intuitive...

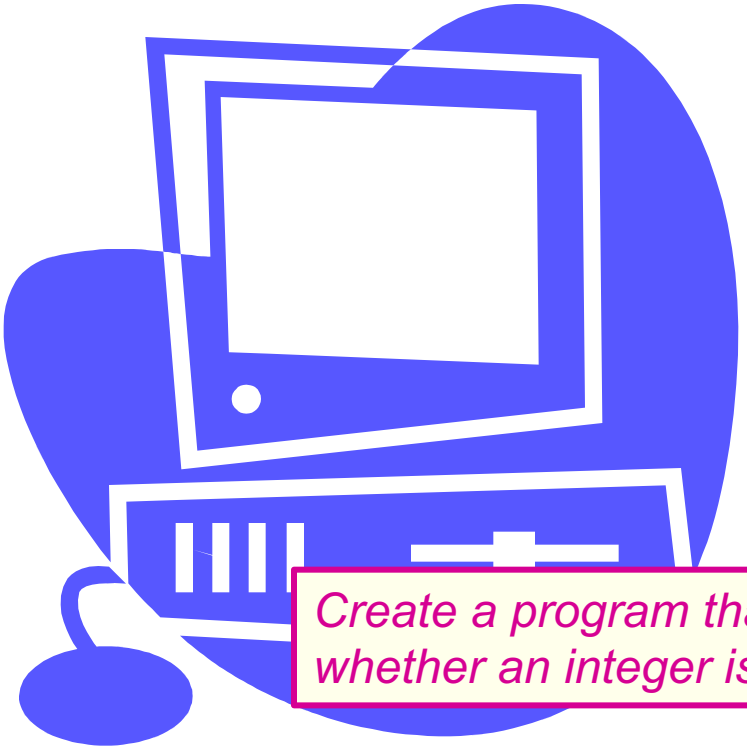
Summary (2)

- Conditional execution: based on the `if/else` statement as well as with the `switch` statement.
- The ternary operator (`?:`) also allows to evaluate expressions based on a condition
- The `for` loop is a versatile iteration construct; the `while` and `do/while` loops are less general.

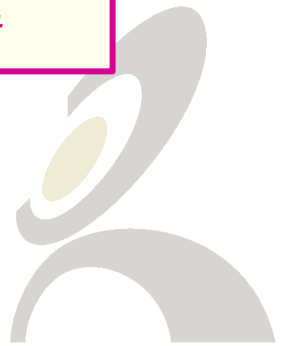
Summary (3)

- Methods allow to group a logical unit of processing.
- The interface with the "outside world" of the method are
  - the argument list
  - the return type (or void)

Exercise



Create a program that determines whether an integer is prime or not



Exercise

Create a program that determines the GCD of two numbers

Questions ??