

Programació I - Tema 2 - Llenguatge Java,
Composicions seqüencial, alternativa i iterativa

Universitat de Barcelona
Grau en Enginyeria Informàtica

18 de setembre de 2012



- 1 2.1. Introducció
- 2 2.2. Estructura genèrica d'un programa
- 3 2.3. Entitats i expressions
- 4 2.4. Sentències i Composicions algorísmiques

Llenguatge Java

- Plataforma **Java** (Sun-Oracle)
 - Codificació en llenguatge de programació **Java**
 - Compilació a llenguatge intermig **bytecode**
 - Execució en la **màquina virtual**
 - **Muntatge** de tots els blocs necessaris
 - **Interpretació** o **Traducció** al llenguatge màquina real
 - Programari lliure: **classpath** & **kaffe**
- Plataforma **.NET** (Microsoft)
 - > 1 **llenguatge** de programació (C#, C++, VB, PHP, Perl...)
 - Estàndard ECMA ; Programari lliure: **mono**



Concepte de programa

Programa: Descripció NO AMBIGUA de les accions que cal realitzar per tal de donar la solució CORRECTA a un problema en un temps FINIT.

Estructura de Dades: Descripció de les entitats utilitzades en el transcurs d'un programa per emmagatzemar-hi la informació.



Estructura d'un programa

- **inclusió de biblioteques:** `import`
- **declaració de classes:** `class`
 - **declaració d'atributs:** contingut intern de l'objecte (variables i constants)
 - **declaració de mètodes:** interfície de l'objecte (cal que com a mínim una de les classes del programa tingui un **mètode principal** `main`, com a punt d'inici d'execució)

Estructura d'un programa

Listing 1: Generic.java

```
importacions

public class Generic {
    public static void main(String [] args) {
        Declaracions Entitats
        Sentències o Instruccions
    }
    Declaracions Mètodes
}
```

2.2.3. Convencions de codi

- importants per al bon manteniment de codi, per a la compartició de codi entre programadors.
- afecten tant la utilització de majúscules i minúscules, com a la situació dels símbols (, , ;, ...), com a situació del codi dins del fitxer java
- Per a més detalls veure l'enllaç
<http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html>



Exemple 1: Exemple programa. Càlcul de la longitud i l'àrea d'una circumferència I

Listing 2: Longitud.java

```
/*  
1. Calcul de la longitud d'una circumferencia i de l'area d'un cercle.  
*/  
  
public class Longitud {  
    public static void main (String [] args) {  
        double radi = 1.; // variable que conte el radi que vulguem  
  
        // sortida del resultat per pantalla;  
        System.out.println("La_longitud_d'una_circumferencia_de_" +  
            "radi_" + radi +  
            "_es_" + 2. * Math.PI * radi);  
        System.out.println("L'area_d'un_cercle_de_radi_" + radi +  
            "_es_" + Math.PI * radi * radi);  
    }  
}
```


Listing 3: Convencions.java

```
/*  
 * Nom de la classe  
 *  
 * Informació de la versió  
 *  
 * Data  
 *  
 * Copyright notice  
 */  
  
import nompaquet;  
  
/**  
 * Descripció de la classe  
 *  
 * @version      nombre data  
 * @author      nom cognom  
 */  
  
public class Convencions {  
    /* Comentari de la classe */
```

Convencions de codi: Exemple de comentaris II

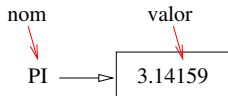
```
/**
 * ... comentari del programa principal
 */
public static void main (string [] args) {
    /* Per a cada entitat declarada:
       comentari de la variable o constant que s'està definint */

    /** classVar1 es el comptador de caràcters */
    public static int classVar1;

    // ... aquí van les sentències ...
}

/**
 * ...documentacio del metode fesAlgunaCosa ...
 */
public void fesAlgunaCosa() {
    // ...sentències del mètode ...
}
}
```

Concepte



- **Entitats:** elements que permeten representar les dades (valors) amb les quals treballarà un programa.
- Característiques de les entitats:
 - Nom : identificador de l'entitat
 - Tipus : conjunt de valors i operacions que es poden realitzar
 - Valor : dada que representa

Característiques de les entitats: Nom

- comença per una lletra, un subratllat (_) o un símbol \$
- els següents caràcters poden ser lletres o dígitos
- es distingeixen minúscules i majúscules
- no es poden usar blancs entremig
- no hi ha longitud màxima
- no poden ser paraules clau de Java o reservades

Exemples:

identificador

nomUsuari

MAX_LONGITUD

_variablesistema

Característiques de les entitats: Tipus

- determina:
 - el conjunt de valors que es poden representar (Rang)
 - l'espai de memòria ocupat
 - la codificació interna utilitzada
 - les operacions que es poden efectuar (Operadors)
 - els literals o valors que contindrà l'entitat
- variants:
 - **tipus valor** (bàsics): byte, short, int, long, float, double, char, boolean
 - **tipus referència**: objectes, p.ex. String

(les pàgines següents mostren exemples de tipus, podeu consultar els que no estan exemplificats a la referència de Java i les seves llibreries)



Característiques de les entitats: Tipus

- tipus valor: valor atòmic (o únic)

Tipus	Ocupació (en bytes)	Codificació	Rang	Valors
byte	1	complement a 2	−128...127	5
short	2	complement a 2	−32768...32767	24563
int	4	complement a 2	−214748364...214748363	234234
long	8	complement a 2	−9223372036854775808... 9223372036854775807	40358
float	4	IEEE 745		5.0f
double	8	IEEE 745		5.0d
char	2	Unicode UTF-16	'\u0000' (o 0) ... (o 65535) '\uffff' (o 65535)	'c'
boolean	1		false, true	true

- tipus referència:
 - String**: cadenes de caràcters



Operadors

- **unaris:**

- numèrics: -, ++, --
- lògics: !

- **binaris:**

- numèrics: +, -, *, /, %
- lògics: &&, ||
- relacionals: ==, <, >, <=, >=, !=
- assignació: +=, -=, *=, %=, /=, ...
- ...

- **n-aris:** mètodes...

(hi ha més operadors dels citats en aquesta pàgina)

Característiques de les entitats: Valor

- La dada que conté l'entitat pot ser modificable o no al llarg del programa:
 - **Constants:** El valor no pot variar durant el programa.
 - Tot el seu identificador es posa en majúscules
 - Si està formada per diferents paraules, es separen pel símbol del subratllat (-)
 - Exemple: MITJANA_NOTES
 - **Variables:** El valor pot ser modificat durant el programa.
 - la primera lletra de l'identificador és una minúscula, no comencen mai per subratllat ni per \$
 - el nom d'una variable ha de ser curt però entenedor
 - les variables d'una sola lletra s'eviten excepte per a variables temporals (*i, j, k, m, n* s'usen de tipus enter; *c, d, e* per a tipus caràcters)
 - si està formada per diferents paraules, les paraules internes comencen en majúscules
 - Exemples: nomAlumne, dniAlumne, i, c

Declaració de constants

- **Sintaxi:**

```
public static final tipusConstant NOM_CONSTANT=valorConstant;
```

- Exemple: `public static final float PI=3.14f;`

Declaració de variables

- **Sintaxi:**

```
tipusVariable nomVariable [=valorInicial];
```

- **Exemples:**

- **tipus valor**

- char a;
 - int b, c;
 - boolean d = false;

- **tipus referència** (exemple particular String)

- String t, u;
 - String v = new String("Hola");
 - String s;
s = new String();

Exemple 2: Definició de les entitats per a calcular la longitud d'una circumferència I

Listing 4: LongitudCircumferencia0.java

```
/*  
 * LongitudCircumferencia0  
 *  
 * 1.0  
 *  
 * 22-09-2009  
 *  
 * Copyright notice  
 */  
  
/**  
 * Longitud circumferència  
 *  
 * @version      1.0 22-09-2009  
 * @author      Anna Puig  
 */
```

Exemple 2: Definició de les entitats per a calcular la longitud d'una circumferència II

```
public class LongitudCircumferencia0 {  
    /** FACTOR es el factor multiplicatiu */  
    public final static double FACTOR = 2.;  
    /** PI es la constant matematica */  
    public final static double PI = 3.14159;  
  
    public static void main (String [] args) {  
        /** radi es el radi de la circumferencia */  
        double radi;  
        /** longitud es la longitu de la circumferència a calcular */  
        double longitud;  
  
        .....  
    }  
}
```

2.3.2. Expressions

- Una expressió és:
 - Una constant
 - Una variable
 - Una crida a una funció
 - la combinació de qualsevol dels anteriors amb operadors aritmètics i lògics
- **si A és una expressió**
 - (A) és una expressió
 - operadorUnari A és una expressió
 - A operadorBinari B és una expressió



Procés de càlcul d'expressions

- Passos per a calcular expressions
 - **1. Anàlisi sintàctica** (ben escrita)
 - **2. Anàlisi semàntica** (concordància de tipus i operadors)
 - **3. Avaluació de l'expressió**: substitució de les variables i constants pels seus valors i càlcul del valor final de l'expressió
- Regles per **avaluar** expressions:
 - *variable*: es substitueix pel valor que guarda en aquell moment
 - *constant*: es substitueix pel valor que guarda en aquell moment
 - *funció*: crida i càlcul del valor de retorn de la funció
 - *operadors*: s'avaluen segons la prioritat + associativitat d'esquerra a dreta

Sentències i composicions algorísmiques:Estructura d'un programa

Listing 5: Generic.java

```
importacions

public class Generic {
    public static void main(String [] args) {
        Declaracions Entitats
        Sentències o Instruccions
    }
    Declaracions Mètodes
}
```



2.4.1 Sentències elementals (Statements)

- Sentències:

- Comentaris: `//, /* ... */`

- Assignacions: `<identificador> = <expressió>;`

- Crides a mètodes:

- Entrada: `import java.util.Scanner; Scanner lectura;
lectura = new Scanner(System.in); nom_variable =
lectura.nextXXXXX();`

- Sortida: `System.out.println(<cadena> + nom_variable);`

- Acaben sempre en `;`

- Formen part sempre d'un bloc (o conjunt) `{ <sentències> }`



2.4.1. Assignació de variables

- Assignació:
 - acció elemental que permet de donar valor a una variable
 - **Sintaxi:**

`nomVariable = expressió`

- S'avalua l'expressió i el valor del resultat es posa com a contingut nou de la variable
- El tipus de la part esquerra ha de coincidir amb el tipus de la part dreta.



Assignació de variables. Exemples

- Exemples:
 - tipus valor
 - `a = 1;`
 - `b = 2 + 3;`
 - `c = 4 + b;`
 - `d = (float) c;`
 - tipus referència (exemple particular String)
 - `s = new String("Hola");`
 - `t = "Ho" + "la";`
 - `u = s + "!";`
 - `String t = u.substring(1, 2);`



Assignació de variables amb operadors aritmètics

- L'assignació:

`nomVariable += expressió`

és equivalent a:

`nomVariable = nomVariable + expressió`

- Hi han d'altres operadors que funcionen igual -=, *=, /=, %=
- Abreviacions d'assignacions:

`nomVariable++;`

és equivalent a:

`nomVariable = nomVariable + 1`



2.4.2. Composició seqüencial

- En el procés de disseny d'un programa, el problema (S) es descompon en una o varies sentències (o instruccions) de forma seqüencial (una darrera l'altra $S \equiv S_0 S_1 S_2 \dots S_n$)

$$S \equiv \begin{array}{c} S_0 \\ S_1 \\ S_2 \\ S_3 \\ \dots \\ S_n \end{array}$$

- S_k sempre s'executa després de S_{k-1}
- L'execució és incondicional: sempre es fa en l'ordre establert i cada S_k s'executa només un cop.

Exemple 2: Longitud d'una circumferència I

Listing 6: LongitudCircumferencia.java

```
import java.util.Scanner;
/* carreguem la biblioteca estàndard per poder fer servir
   el mètode d'entrada per teclat */
public class LongitudCircumferencia {

    public final static double FACTOR = 2.;

    public static void main (String [] args) {
        double radi, longitud;
        Scanner sc;
        sc = new Scanner(System.in);
        System.out.println("Radi?");
        radi = sc.nextDouble();

        longitud = FACTOR * Math.PI * radi;

        System.out.println("Longitud_circumferència_" +
            "radi(" + radi + ")_és_" +
            longitud); // sortida per pantalla
    }
}
```

Exemple 3: Intercanvi de dues variables I

Listing 7: Intercanvi.java

```
import java.util.Scanner;
public class Intercanvi {
    public static void main (String [] args) {
        int x;
        int y;
        int tmp;
        Scanner sc;
        sc = new Scanner(System.in);
        System.out.println("X?"); // entrada per teclat
        x = sc.nextInt();
        System.out.println("Y?");
        y = sc.nextInt();

        tmp = x;
        x = y;
        y = tmp;

        System.out.println("X:_" + x); // sortida per pantalla
        System.out.println("Y:_" + y);
    }
}
```

Assignacions

- Pre-increments i post-increments:

```
x = ++nomVariable;
```

és equivalent a:

```
nomVariable = nomVariable + 1; x = nomVariable
```

```
x = nomVariable++;
```

és equivalent a:

```
x = nomVariable; nomVariable = nomVariable + 1
```

Depuració

- **Taula d'execució**: model teòric de l'execució del programa.
- **Bolcat** (*log*): bolcar el valor de les variables a mida que s'executa el programa.
- **Depurador** (*debugger*): utilitat per executar el codi pas a pas i poder explorar les estructures de dades.

Exemple Taula d'execució I

```
import java.util.Scanner;
public class LongitudCircumferenciaTaula {
    public final static double FACTOR = 2.;
    public static void main (String [] args) {
        double radi, longitud;
        Scanner sc;
        sc = new Scanner(System.in);
        System.out.println("Radi?");

        radi = sc.nextDouble();
        longitud = FACTOR * Math.PI * radi;
        System.out.println("Longitud_=" + longitud);
    }
}
```

ent	radi	longitud	sort
⊥1	-	-	-
1⊥	1	-	Radi?
1⊥	1	6.28	Radi?
1⊥	1	6.28	Radi?↔Longitud=6.28

Exemple Bolcat I

```
import java.util.Scanner;

public class LongitudCircumferenciaBolcat {

    public final static double FACTOR = 2.;

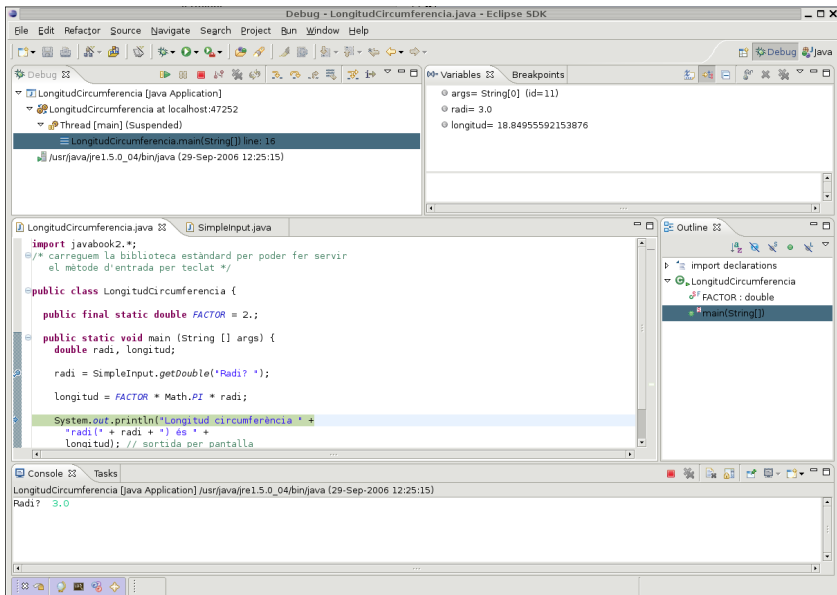
    public static void main (String [] args) {
        double radi = 0;
        double longitud = 0;
        System.out.println("radi (" + radi + ") _ longitud (" + longitud + ")");

        Scanner sc;
        sc = new Scanner(System.in);
        System.out.println("Radi?");
        radi = sc.nextDouble();
        System.out.println("radi (" + radi + ") _ longitud (" + longitud + ")");

        longitud = FACTOR * Math.PI * radi;
        System.out.println("radi (" + radi + ") _ longitud (" + longitud + ")");

    }
}
```

Exemple Depurador: Eclipse



Problemes sobre seqüencial proposats I

- Donats dos enters x , y realitzeu l'intercanvi dels seus valors sense utilitzar cap variable temporal.
- Donat un valor de temps en segons, calculeu el nombre d'hores, minuts i segons que representa aquest valor.



2.4.3. Composició alternativa

- Es parla de composició alternativa quan en l'anàlisi d'un problema cal executar unes o altres accions segons si una expressió booleana és certa o no.
- **Anàlisi del problema per casos:**
 - S'analitzen tots els casos possibles de l'expressió booleana
 - Tots els casos han de ser disjunts
 - Tots els casos han de cobrir totes les possibilitats

Composició alternativa

- **Anàlisi per casos:**

- si

$\langle \text{cond_booleana}_1 \rangle \rightarrow S_1$

$\langle \text{cond_booleana}_2 \rangle \rightarrow S_2$

$\langle \text{cond_booleana}_3 \rangle \rightarrow S_3$

...

$\langle \text{cond_booleana}_N \rangle \rightarrow S_N$

fsi



Traducció directa: Aniuament d'alternatives

```
• if (<condició_1>) { <S_1> }  
  else { if (<condició_2>) <S_2> }  
  ...  
  else { if (<condició_N>) <S_N> }  
  else {  
  }
```



Composició alternativa directa I

- Traducció literal

```
import java.util.Scanner;

public class AlternativaMultipleAniuament {

    public static void main (String [] args) {
        int edat;
        String resposta;
        Scanner sc;
        sc = new Scanner(System.in);
        System.out.println("Quina_edat_tens?"); // entrada per
        edat = sc.nextInt();
        /* Anàlisi per casos:
           si
               edat >= 18 -> resposta = "universitaris";
               14<= edat && edat >18 -> resposta = "secundaris";
               edat < 14 ->  resposta = "primaris";
           fsi
        */
```


Composició alternativa directa II

```
if (edat >= 18) {
    resposta = "universitaris";
} else {
    if (edat >= 14 && edat < 18) {
        resposta = "secundaris";
    } else {
        if (edat < 14) {
            resposta = "primaris";
        } else {
            /* Ultim else de tots.
               Aquí no hauria de passar mai! */
            resposta = "";
        }
    }
}
System.out.println("Pots_cursar_estudis_" + resposta);
}
```

Composició alternativa: 1^a Simplificació: Últim else i condicions I

- L'últim else desapareix i la última condició també.

```
import javabook2.*;
public class AlternativaMultipleAniuamentI {

    /* Simplificació de l'últim else */

    public static void main (String [] args) {
        int edat;
        String resposta;
        Scanner sc;

        sc = new Scanner(System.in);
        System.out.println("Quina_edat_tens?");
        edat = sc.nextInt();
        /* Anàlisi per casos:
           si
           edat >= 18 -> resposta = "universitaris";
           14<= edat && edat >18 -> resposta = "secundaris";
```

Composició alternativa: 1^a Simplificació: Últim else i condicions ll

```
        edat < 14 -> resposta = "primaris";
    fsi
*/
if (edat >= 18) {
    resposta = "universitaris";
} else {
    if (edat >= 14) {
        resposta = "secundaris";
    } else {
        resposta = "primaris";
    }
}
System.out.println("Pots_cursar_estudis_"+resposta);
}
```

Composició alternativa: 2^a Simplificació: Cas de dos casos I

- `if (<condició>) { <sentències> }`
`else { <sentències> }`

```
import java.util.Scanner;
```

```
public class AlternativaBifurcacio {
```

```
    public static void main (String [] args) {
```

```
        int edat;
```

```
        Scanner sc;
```

```
        sc = new Scanner(System.in);
```

```
        System.out.println("Quina_edat_tens?"); // entrada per tec
```

```
        edat = sc.nextInt();
```

```
        /* Anàlisi de casos:
```

```
        si
```

```
            edat >= 18 -> escriure per pantalla "Ja pots entrar a .
```

```
            edat < 18 -> escriure per pantalla "Encara no pots en
```

Composició alternativa: 2^a Simplificació: Cas de dos casos

II

```
        fsi
    */

    if (edat >= 18) {
        System.out.println("Ja_pots_entrar_a_la_universitat");
    } else {
        System.out.println("Encara_no_pots_entrar_a_la_universitat");
    }

    System.out.println("A_reveure!");
}
}
```

Composició alternativa: 3^a Simplificació: Cas del continuar I

- `if (<condició>) { <sentències> }`
- `<condició>` = tota expressió que avalua a tipus booleà

```
import java.util.Scanner;
```

```
public class AlternativaSimple {
```

```
    public static void main (String [] args) {
```

```
        int edat;
```

```
        Scanner sc;
```

```
        sc = new Scanner(System.in);
```

```
        System.out.println("Quina_edat_tens?"); // entrada per tec
```

```
        edat = sc.nextInt();
```

```
        /* Anàlisi de casos:
```

```
            si
```

```
                edat >= 18 -> escriure per pantalla
```

```
                        "Ja pots entrar a la universitat"
```

```
                edat < 18 -> CONTINUAR
```

Composició alternativa: 3^a Simplificació: Cas del continuar II

```
        fsi
    */

    if (edat >= 18) {
        System.out.println("Ja_pots_entrar_a_la_uni");
    }

    System.out.println("A_reveure!");
}
}
```

Curiositat: Alternativa Abreujada I

- (<condició>)?<sentènciesSÍ>:<sentènciesNO>

```
import java.util.Scanner;
```

```
public class AlternativaAbreujada {
```

```
    public static void main (String [] args) {
```

```
        int edat;
```

```
        String resposta;
```

```
        Scanner sc;
```

```
        sc = new Scanner(System.in);
```

```
        System.out.println("Quina_edat_tens?"); // entrada per tec
```

```
        edat = sc.nextInt();
```

```
        resposta = (edat >= 18) ? "Sí" : "No";
```

```
        System.out.println(resposta + "_pots_entrar_a_la_universitat");
```

```
        System.out.println("A reveure!");
```

```
    }
```

```
}
```


Múltiple - instrucció dedicada

- Només és utilitzable si l'expressió és un char, un byte, un short o un int.
- Implementa casos de comparacions per igualtat.
- Sintaxi:

```
switch (<expressió>) {  
    case <valor1>: <sentències>; [break;]  
    <...>  
    case <valorN>: <sentències>; [break;]  
    default: <sentències>;  
}
```



Múltiple - instrucció dedicada - exemple I

```
import java.util.Scanner;
public class AlternativaMultipleInstruccio {
    public static void main (String [] args) {
        int edat; String resposta;
        Scanner sc;

        sc = new Scanner(System.in);
        System.out.println("Quina_edat_tens?"); // entrada per teclat
        edat = sc.nextInt();
        /* analisi de casos
           si
               edat <= 0 -> error
               edat == 1 || edat == 2 -> escriure(Escola Bressol)
               edat == 3 || edat == 4 || edat == 5 -> escriure(Primaria)
               edat >= 6 && edat <= 11 -> escriure(primaria)
               edat == 12 -> escriure (1er ESO)
               edat == 13 -> escriure (2er ESO)
               edat == 14 -> escriure (3er ESO)
               edat == 15 -> escriure (4er ESO)
               edat > 15 -> escriure (secundaris o universitaris)
           fsi
        */
```

Múltiple - instrucció dedicada - exemple II

```
switch(edat) {  
    case 12: resposta = "1r_ESO"; break;  
    case 13: resposta = "2n_ESO"; break;  
    case 14: resposta = "3r_ESO"; break;  
    case 15: resposta = "4t_ESO"; break;  
    case 1: case 2: resposta = "escola_bressol"; break;  
    case 3: case 4: case 5: resposta = "llar_d'infants"; break;  
    default:  
        if (edat >= 6 && edat <=11) {  
            resposta = "primària";  
        } else {  
            if (edat>15) {  
                resposta = "estudis_secundaris/superiors";  
            } else {  
                resposta = "res";  
            }  
        }  
    }  
    System.out.println("Pots_cursar_"+resposta);  
}
```

Simplificació d'anàlisi de casos: són necessaris aquests anàlisis? I

```
/* Anàlisi de casos 1:
si
  x>3 --> mesgran3 = true;
  x <= 3 --> mesgran3 = false;
fsi

*/

/* Anàlisi de casos 2:
si
  trobat --> perdut = false;
  !trobat --> perdut = true;
fsi

*/
```

Simplificació d'anàlisi de casos: són necessaris aquests anàlisis? II

```
/* Anàlisi de casos 3:
si
    trobat == true    --> System.out.println("Trobat");
    trobat == false   --> System.out.println("Perdut");
fsi

*/

/* Anàlisi de casos 4:
si
    hies -->
        si (trobat && a>5) --> j = true;
        !(trobat && a>5) --> j = false;
    !hies --> j = true;
fsi

*/
```

2.4.4. Composició iterativa

- Quan s'ha de repetir l'execució d'una acció o conjunt d'accions segons si es compleix una propietat (o condició booleana), s'utilitza la composició iterativa.
- **Sintaxi bàsica:**
 - S_i (Sentències inicials)

```
while ( condició booleana ) {  
    <  $S$  ( sentències ) >  
}
```

 S_f (Sentències finals)



Composició iterativa

● Mecanisme:

- Inicialment s'executen les sentències S_i , després s'avalua l'expressió booleana del while. Si és certa s'executen les sentències S . Es torna a analitzar la condició després d'executar S .
 - Les accions del cos del while (S) només s'executen si la condició booleana és certa.
 - Quan la condició booleana és falsa es passen a executar les sentències finals de després del mentre (S_f)
- **Correctesa:** S'han de garantir l'estat inicial abans del while, la condició booleana de control del while i les sentències internes per a que la composició iterativa acabi en algun moment.



Exemples: Analitzar les taules d'execució (o traça) dels següents programes I

```
public class Escriu100Enters {  
  
    public static void main (String [] args) {  
        int num;  
  
        num = 1;  
        while (num <= 100) {  
            System.out.println("_"+num+"_");  
            num = num + 1;  
        }  
    }  
}
```


Exemples: Analitzar les taules d'execució (o traça) dels següents programes II

```
public class Escriu100EntersInf {  
    public static void main (String [] args) {  
        int num;  
  
        num = 1;  
        while (num <= 100) {  
            System.out.println("_"+num+"_");  
        }  
    }  
}
```

```
public class CondicionsInicialsIterativa {  
    public static void main (String [] args) {  
        int num = 0;  
        /* I num = 3? i num = -1? i num = 4? */  
        while (num != 0) {  
            num = num - 2;  
        }  
    }  
}
```