UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escola Tècnica Superior d'Enginyeria Informàtica

etsinf

Tema 6. Estructures de control: iteració

Introducció a la Informàtica i a la Programació (IIP)

Curs 2019/20

Departament de Sistemes Informàtics i Computació

Continguts

Duració: 5 sessions

1. Introducció: la necessitat d'usar estructures de repetició per a programar
2. Instruccions d'iteració en Java: `for`, `while` i `do...while`
3. La iteració com estratègia de disseny: fases i elements
   - Fase 1: Descobrir l'estructura iterativa del problema
   - Fase 2: Determinar els elements d'una iteració correcta: cos, inicialització i guarda
   - Fase 3: Traduir la iteració dissenyada a un bucle Java
   - Fase 4: Comprovar la correcció i terminació del bucle dissenyat
4. Exemples i exercicis amb:
   - Validació de dades
   - Recurrències
   - Iteracions niuades

- **Crea** una carpeta ***Tema 6*** dins de la teua carpeta ***W:\IIP***
- **Descarrega** (del Tema 6 de PoliformaT) els fitxers **BlueJ *exemplesT6.jar*** i ***exercicisT6.jar***
- Des de l'opció **Projecte** de ***BlueJ***, usa l'opció **Open ZIP/JAR…** per tal d'obrir-los com projectes ***BlueJ*** i prepara't per usar-los

Introducció: la necessitat d'usar estructures de repetició per a programar

- Independentment de la complexitat d'un problema, la seua resolució mitjançant un programa ha de complir les condicions següents:
  - Les instruccions que la componen han d'escriure's de manera finita, i
  - la seua execució també ha de ser finita.
- Per a resoldre molts problemes senzills (com els dels temes anteriors) basta amb una simple seqüència d'instruccions.
- Però en altres problemes resulta imprescindible que aquesta seqüència d'instruccions es repetisca per a poder resoldre'ls.
- En ocasions, el nombre de vegades que ha de repetir-se la seqüència es coneix per endavant; però altres vegades no és així.

Per què instruccions de repetició?

- Mostrar per pantalla n vegades “Hola des de BlueJ”
  - n instruccions `System.out.println(“Hola des de BlueJ”);`
  - I si n és una dada?
- Llegir les dades d’hora i minuts fins que siguen vàlids
  - Llegir les dades i si no són vàlides tornar a llegir
  - Quantes vegades?
- Com es calcula l’arrel quadrada d’un número real x >= 0?
  - Mètode iteratiu de Newton: calcular els termes de la següent sèrie fins que la diferència entre els dos darrers termes calculats siga molt menuda:

$$t_1 = \frac{1+x}{2}$$

$$t_{i+1} = \frac{1}{2}\left(t_i + \frac{x}{t_i}\right)$$

```
Math.sqrt(7)
   2.6457513110645907   (double)
```

```
double x = 7.0;
double t1 = (1 + x) / 2;
t1
   4.0   (double)
double t2 = (1 / 2.0) * (t1 + x / t1);
t2
   2.875   (double)
double t3 = (1 / 2.0) * (t2 + x / t2);
t3
   2.654891304347826   (double)
double t4 = (1 / 2.0) * (t3 + x / t3);
t4
   2.64576704419029   (double)
double t5 = (1 / 2.0) * (t4 + x / t4);
t5
   2.6457513111113693   (double)
t4 - t5
   1.5733078920554533E-5   (double)
```

Per què instruccions de repetició?

BlueJ:pract1Bluej

```
public class HolaBlueJ {
    public static void main(String[] args) {
        System.out.println("Hola des de BlueJ");
    }
}
```

```
import java.util.Scanner;
public class HolaBlueJ {
  public static void main(String[] args) {
     Scanner teclat = new Scanner(System.in);
     System.out.println("Quantes vegades vols repetir el missatge? ");
     int n = teclat.nextInt();
     REPETIR DES DE 1 FINS n vegades ...
     System.out.println("Hola des de BlueJ");
  }
}
```

Necessitem instruccions que ens permeten iterar

Per què instruccions de repetició?

BlueJ:iip [pract4]

```
public class TimeInstant {
    private int hours, minutes;
    ...
    /** PRECONDICIÓ: 0 <= h < 24, 0 <= m < 60
     *  crea un TimeInstant de h hores i m minuts */
    public TimeInstant(int h, int m) { hours = h; minutes = m; }
    ...
}
```

BlueJ:iip [pract4]

```
public class Test4 {
  private Test4() { }
  public static void main(String[] args) {
     Scanner teclat = new Scanner(System.in);
     System.out.println("Lectura de teclat d'una hora.");
     REPETIR...
     System.out.print(" -> Introduiu les hores (entre 0 i 23): ");
     int hor = teclat.nextInt();
     System.out.print(" -> Introduiu els minuts (entre 0 i 59): ");
     int min = teclat.nextInt();
     MENTRE hor i min NO són vàlids
     TimeInstant hUser = new TimeInstant(hor, min);
     ...
  }
}
```

Necessitem instruccions que ens permeten iterar

Per què instruccions de repetició?

BlueJ:iip [pract6]

```
public class IIPMath {
    private IIPMath() { }

    public static double sqrt(double x, double epsilon) {
        double term = (1 + x) / 2;

        REPETIR MENTRE QUE LA DIFERÈNCIA ENTRE EL
        TERME CALCULAT I L'ANTERIOR SIGA "GRAN"...
        anterior = term;
        term = (anterior + x / anterior) / 2;
        dif = anterior - term;

        return term;
    }
}
```

Necessitem instruccions que ens permeten iterar

Per què instruccions de repetició?

- Els llenguatges de programació disposen d'instruccions per representar les estructures de repetició: les instruccions iteratives o bucles.
- En Java hi ha tres instruccions iteratives:
  - El bucle `while`
  - El bucle `for`
  - El bucle `do...while`
- Les estructures iteratives (bucles) permeten expresar de forma finita una execució tan llarga com es requerisca.
- El número d'instruccions a executar pot estar determinat a priori (ser un número fixe) o ser un número variable que depén de les dades.

- Dos aspectes a tenir en compte:
  - la terminació: s'ha de garantir que el nombre de repeticions siga finit.
  - l'eficiència: s'ha d'executar el menor nombre possible d'instruccions.

Instruccions d'iteració en Java: instrucció for

Sintaxi:

```
for ([Inici]; B; [Avanç]) {
    S;
}
```

Si **S** és una seqüència d'instruccions **{ } obligatòries**

Si **S** és una única instrucció **{ }** NO obligatòries però **RECOMANADES** segons les **convencions** d'escriptura de **codi Java** (**Checkstyle**)

on:

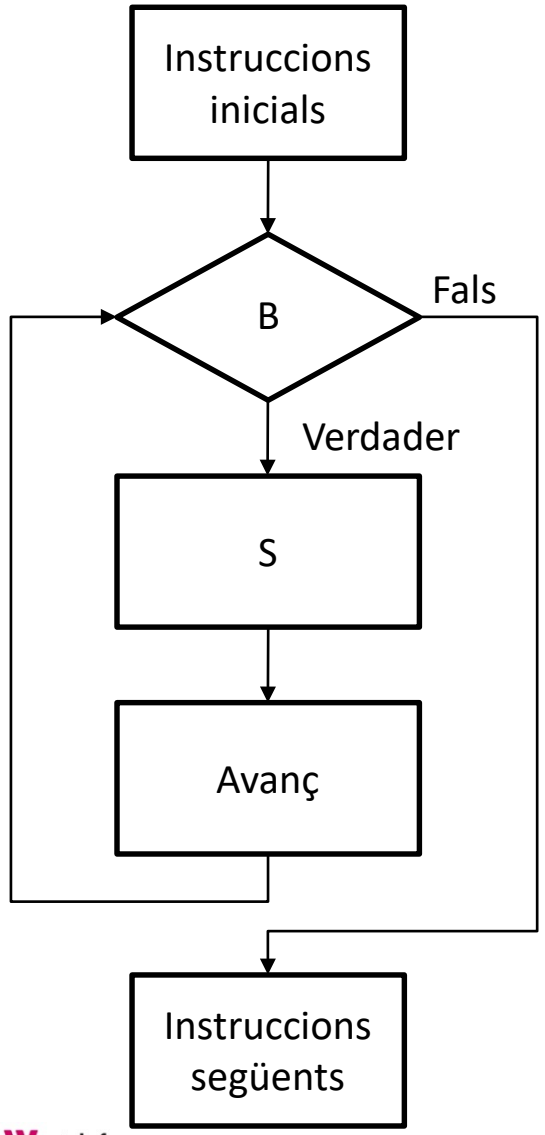
- els elements entre claudàtors [ ] són optatius,
- **B** és una expressió lògica,
- **S** és una instrucció qualsevol (o una seqüència d'instruccions),
- `Inici` i `Avanç` són llistes d'instruccions `i1`, `i2`,..., `in` i `a1`, `a2`,..., `am` acabades per comes, que defineixen la inicialització i la progressió del bucle, respectivament.

Instruccions d'iteració en Java: instrucció for

Execució:

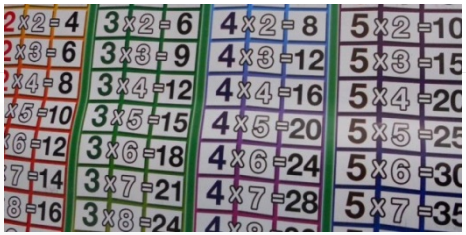
- S'executen les instruccions inicials.
- Mentre que la condició B siga certa s'executen les instruccions de S seguides de les instruccions d'avanç.
- Quan la condició B deixa de ser certa s'acaba l'execució del bucle i es continua en la següent instrucció a la iterativa.

Pot ser que les instruccions **S** i les d'avanç no s'executen.



Instruccions d'iteració en Java: instrucció for
Exemple: la taula de multiplicar

BlueJ:exemplesT6



```
import java.util.Scanner;
public class TaulaDeMultiplicar {
    private TaulaDeMultiplicar() { }

    public static void main(String[] args) {
        Scanner teclat = new Scanner(System.in);
        System.out.print("Dis-me un número positiu: ");
        int num = teclat.nextInt();
        if (num > 0) {
            mostrarTaulaDel(num);
        } else { System.out.println("El número no és vàlid"); }
    }

    public static void mostrarTaulaDel(int n) {
        System.out.printf("Taula del %2d\n", n);
        for (int i = 1; i <= 10; i++) {
            System.out.printf("%2d x %2d = %2d\n", n, i, n * i);
        }
    }
}
```

Instruccions d'iteració en Java: instrucció do...while

Sintaxi:

```
do {
     S;
} while (B);
```

Si **S** és una seqüència d'instruccions **{ }** **obligatòries**

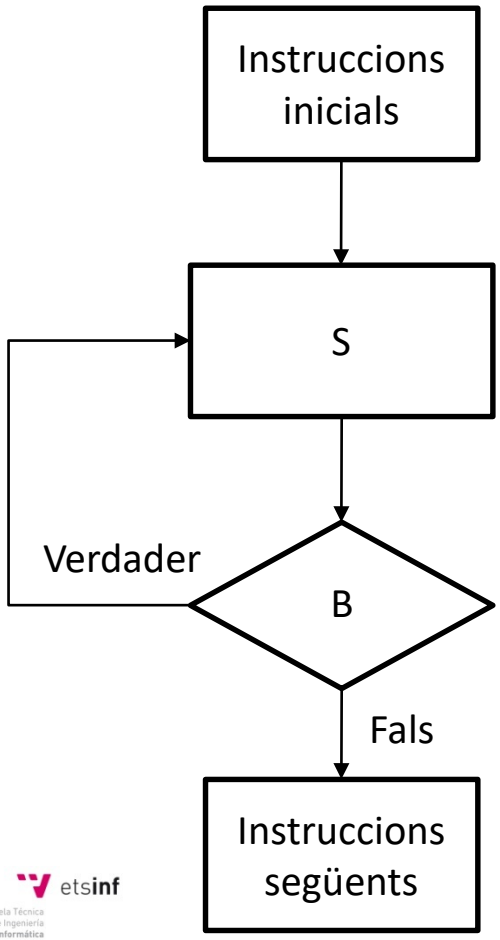
Si **S** és una única instrucció **{ }** NO obligatòries però **RECOMANADES** segons les **convencions** d'escriptura de **codi Java** (**Checkstyle**)

on:

- **B** és una expressió lògica,
- **S** és una instrucció qualsevol o una seqüència d'instruccions.

Execució:

- S'executen les instruccions inicials.
- Es repeteix l'execució de les instruccions S fins que s'arriba a un estat en què la condició B és falsa.
- En acabar, es continua en la instrucció següent a la iterativa.



➡ **S** s'executa almenys una vegada.

Instruccions d'iteració en Java: instr. do...while
Exemple: validar una dada

BlueJ:exemplesT6

```
import java.util.Scanner;
public class TaulaDeMultiplicar2 {
    private TaulaDeMultiplicar2() { }

    public static void main(String[] args) {
        Scanner teclat = new Scanner(System.in);
        do {
            System.out.print("Dis-me un número positiu: ");
            int num = teclat.nextInt();
            if (num <= 0) {
                System.out.println("El número no és vàlid");
            }
        } while (num <= 0);
        mostrarTaulaDel(num);
    }

    public static void mostrarTaulaDel(int n) {
        System.out.printf("Taula del %2d\n", n);
        for (int i = 1; i <= 10; i++) {
            System.out.printf("%2d x %2d = %2d\n", n, i, n * i);
        }
    }
}
```

Instruccions d'iteració en Java: instrucció while

Sintaxi:

```
while (B) {
    S;
}
```

Si **S** és una seqüència d'instruccions { } **obligatòries**

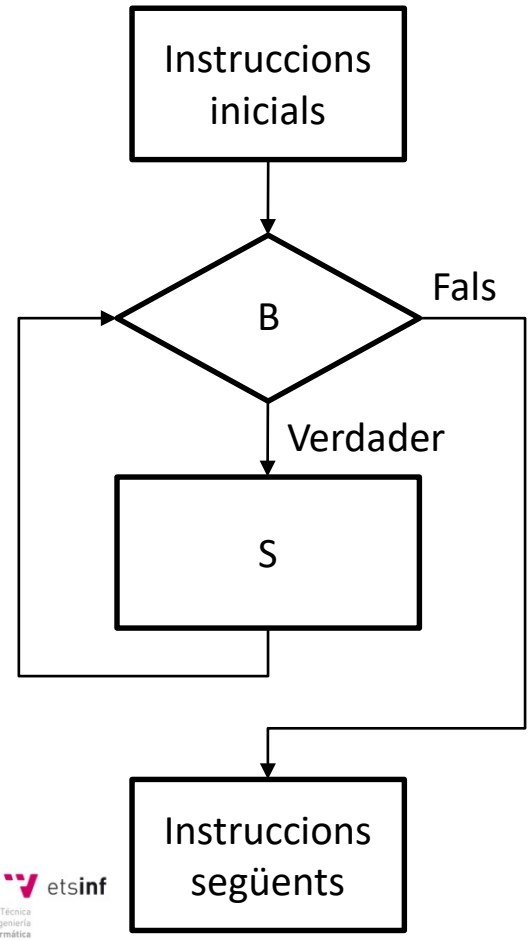
Si **S** és una única instrucció { } NO obligatòries però **RECOMANADES** segons les **convencions** d'escriptura de **codi Java** (**Checkstyle**)

on:

- **B** és una expressió lògica,
- **S** és una instrucció qualsevol o una seqüència d'instruccions.

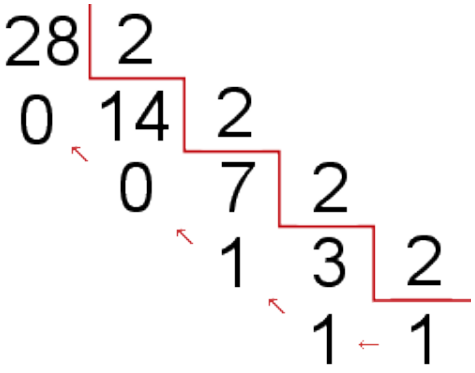
Execució:

- S'executen les instruccions inicials.
- Mentre que la condició B siga certa s'executen les instruccions S.
- Quan la condició B deixa de ser certa, s'acaba l'execució del bucle i es continua en la següent instrucció posterior a la iterativa.



Pot ser les instruccions de **S** no s'executen.

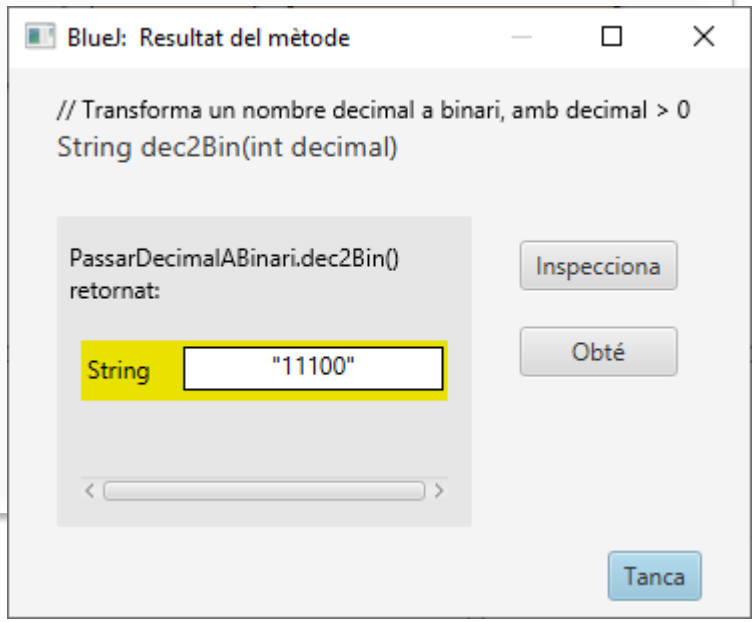
Instruccions d'iteració en Java: instrucció `while`
Exemple: passar de decimal a binari



$28 = 11100_2$

BlueJ:exemplesT6

```
// decimal > 0
public static String dec2Bin(int decimal) {
    String binari = "";
    int dividend = decimal;
    while (dividend > 0) {
        binari = dividend % 2 + binari;
        dividend /= 2;
    }
    return binari;
}
```



La iteració com estratègia de disseny: **elements**

- Els components estructurals bàsics de tota iteració o bucle són:
  - La guarda del bucle: condició que mentre s'avalue a verdader implicarà la repetició de l'execució del cos del bucle (bloc d'instruccions).
  - El cos del bucle: les instruccions que permeten avançar pas a pas cap a la solució del problema.
  - La inicialització de variables: abans d'entrar en el bucle s'ha d'assignar valors ben definits a totes les variables involucrades en l'execució del bucle.

BlueJ:exemplesT6

```
// decimal > 0
public static String dec2Bin(int decimal) {
    String binari = "";
    int dividend = decimal;
    while (dividend > 0) {
        binari = dividend % 2 + binari;
        dividend /= 2;
    }
    return binari;
}
```

Inicialització de les variables del bucle

Guarda del bucle

Cos del bucle

La iteració com estratègia de disseny: **elements**

```
for ([i1, i2, ..., in]; B; [a1, a2, ..., am]) {
        S;
}
```

```
[i1; i2; ...; in;]      // Inici
do {
    S;
   [a1; a2; ...; am;]   // Avanç
} while (B);
```

```
[i1; i2; ...; in;]      // Inici
while (B) {
    S;
   [a1; a2; ...; am;]   // Avanç
}
```

Inicialització de les variables del bucle

```
[i1 i2 ... in]
```

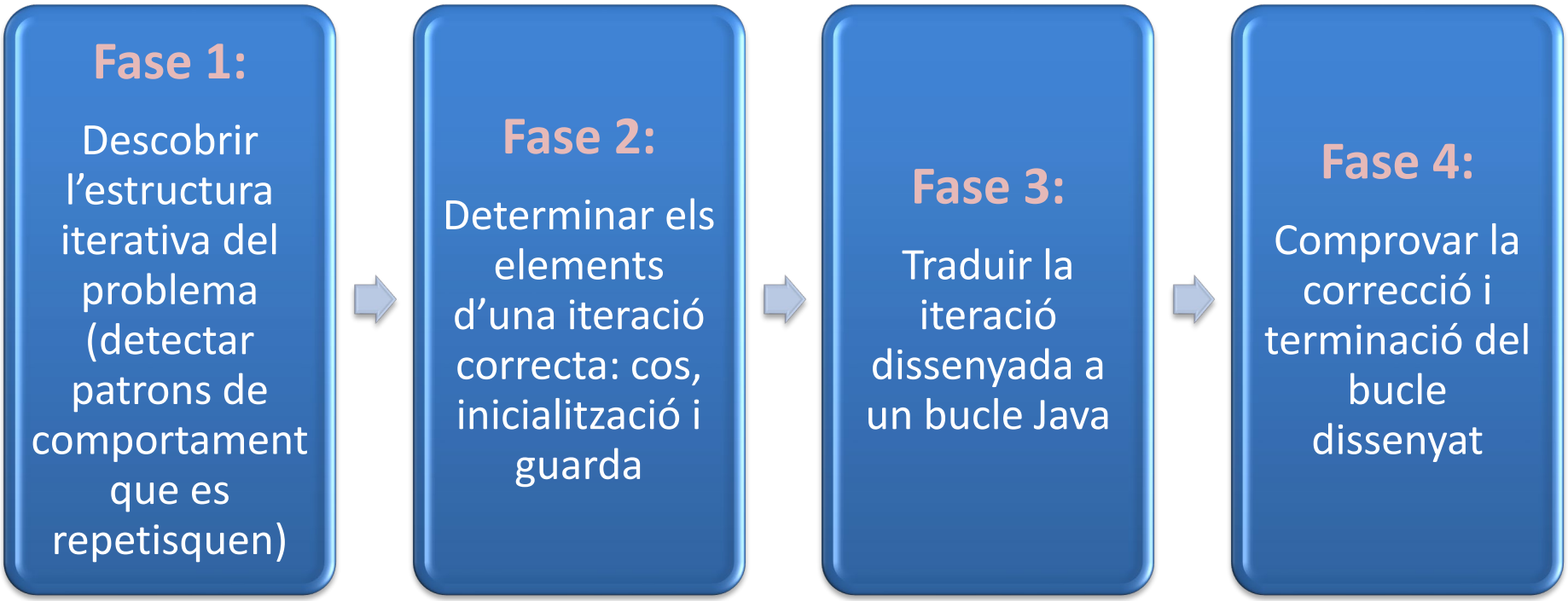
Guarda del bucle

```
B
```

Cos del bucle

```
S;
[a1 a2 ... am]
```

La iteració com estratègia de disseny: **fases**



La iteració com estratègia de disseny: **exemple**

- Suposem que no disposem de l'operador de multiplicació i hem d'implementar el càlcul del producte d'a per n (a * n), basant-nos en sumes (sent n un valor no negatiu).

mètode (static) **producteAperN**

DADES/PARÀMETRES `int a, int n` tal que $a \geq 0 \wedge n \geq 0$;

RESULTATS `int producte` tal que `producte = a * n`

Estratègia: com multiplicar sense usar l'operador *?

Repetir n vegades producte = producte + a

```
a*n == a + a + a + . . . + a      si n>0
                 n vegades
a*n == 0                          si n==0
```

La iteració com estratègia de disseny: **exemple**

Fase 1: Descobrir l'estructura iterativa del problema
(detectar patrons de comportament que es repetisquen)

- Cos, o quines instruccions hi ha que repetir:

```
producte = producte + a;  // producte és un acumulador
```

- Quantes vegades hi ha que repetir l'execució del cos: n vegades

```
producte = 0;
```

sume 0 vegades

```
producte = producte + a;
producte = producte + a;
...
producte = producte + a;
```

sume n vegades

sume n vegades

La iteració com estratègia de disseny: **exemple**

Fase 1: Descobrir l'estructura iterativa del problema
(detectar patrons de comportament que es repetisquen)

Alternativament:

- Cos, o quines instruccions hi ha que repetir:

```
producte = producte + a;  // producte és un acumulador
```

- Quantes vegades hi ha que repetir l'execució del cos: n−1 vegades

```
producte = a;
```

sume 1 vegada

```
producte = producte + a;
producte = producte + a;
...
producte = producte + a;
```

sume n-1 vegades

sume n vegades

La iteració com estratègia de disseny: **exemple**

Fase 1: Descobrir l'estructura iterativa del problema
(detectar patrons de comportament que es repetisquen)

- Cos, o quines instruccions hi ha que repetir:

```
producte = producte + a;  // producte és un acumulador
```

- Quantes vegades hi ha que repetir l'execució del cos: n vegades

Com les compte?

```
producte = 0;
cont = 0;
```

```
producte = producte + a;
cont++;
producte = producte + a;
cont++;
...
producte = producte + a;
cont++;
```

```
// he sumat 0 vegades
// he repetit 0 vegades
// he sumat 1 vegada
// he repetit 1 vegada
// he sumat 2 vegades
// he repetit 2 vegades


// he sumat n vegades
// he repetit n vegades
```

➔PARAR

He sumat n vegades

He repetit n vegades

La iteració com estratègia de disseny: **exemple**

Fase 2: Determinar els elements d'una iteració (cos, inicialització i guarda), refinant la Fase 1

- **Condició de parada ➔ Guarda**

com l'execució del cos *acaba quan* `(cont == n)`

Parada

el cos *s'executa mentre* `(cont != n)`

!Parada = Guarda

- **Cos:**

```
producte = producte + a;
cont++;
```

- **Inicialització de les variables que intervenen en la iteració:**

abans de començar *a repetir*, `producte = 0; cont = 0;`

La iteració com estratègia de disseny: **exemple**

Fase 3: Traduir la iteració dissenyada a un bucle Java

- Expressar algorísmicament la iteració:

```
int producte = 0, cont = 0;
MENTRE (cont != n) FER {
    producte = producte + a;
    cont++;
}
```

- Traduir a un bucle Java (`while`, `for` o `do…while`)

```
int producte = 0;
for (int cont = 0; cont != n; cont++) {
   producte = producte + a;
}
```

```
int producte = 0, cont = 0;
while (cont != n) {
    producte = producte + a;
    cont++;
}
```

```
int producte = 0, cont = 0;
if (cont != n) {
    do {
        producte = producte + a;
        cont++;
    } while (cont != n);
}
```

La iteració com estratègia de disseny: **exemple**

Fase 4: Comprovar la correcció i terminació del bucle dissenyat

```
int producte = 0, cont = 0;
while (cont != n) {
    producte = producte + a;
    cont++;
}
```

Consideracions sobre la correcció, la terminació i l'eficiència:

- Per construcció el codi és correcte si $a \geq 0 \wedge n \geq 0$
- El bucle acaba ja que $n \geq 0$ i `cont` comença valent 0 i s'incrementa en 1 en cada passada del bucle: en n iteracions deixarà de cumplir-se la guarda i el bucle acabarà.
- La guarda del bucle s'avalua $n + 1$ vegades i les instruccions del cos del bucle s'executaran n vegades.

La iteració com estratègia de disseny: **exemple**

- Són correctes els següents fragments de codi per a calcular `a * n` ?

```
int producte = 0, cont = n;
while (cont != 0) {
    producte = producte + a;
    cont--;
}
```

```
int producte = 0, cont = 0;
while (cont >= 0) {
    producte = producte + a;
    cont++;
}
```

La iteració com estratègia de disseny

BlueJ:exercicisT6

Exercici: suma dels n primers naturals

$$s = \sum_{k=0}^{n} k$$

```
s = 0 + 1 + 2 + 3 + . . . + (n-2) + (n-1) + n
```

```
// Primera versió: suma ascendent 1+2+3+4+...+(n-2)+(n-1)+n
int suma = 0,        // acumula el resultat de les successives sumes
    k = 1;           // indica el nº que es va a sumar a suma
while (      ) {


}
```

Traça per a n = 3

iteració	k	suma

```
// Segona versió: suma ascendent 1+2+3+4+...+(n-2)+(n-1)+n
int suma = 0,        // acumula el resultat de les successives sumes
    k = 0;           // indica el darrer nº que s'ha sumat a suma
while (      ) {


}
```

Traça per a n = 3

iteració	k	suma

La iteració com estratègia de disseny

BlueJ:exercicisT6

Exercici: suma dels n primers naturals

$$s = \sum_{k=0}^{n} k$$

```
s = 0 + 1 + 2 + 3 + . . . + (n-2) + (n-1) + n
```

```
// Primera versió: suma descendent n+(n-1)+(n-2)+...+4+3+2+1
int suma = 0,        // acumula el resultat de les successives sumes
    k = n;           // indica el nº que es va a sumar a suma
while (      ) {


}
```

Traça per a n = 3

iteració	k	suma

```
// Segona versió: suma descendent n+(n-1)+(n-2)+...+4+3+2+1
int suma = n,        // acumula el resultat de les successives sumes
    k = n;           // indica el darrer nº que s'ha sumat a suma
while (      ) {


}
```

Traça per a n = 3

iteració	k	suma

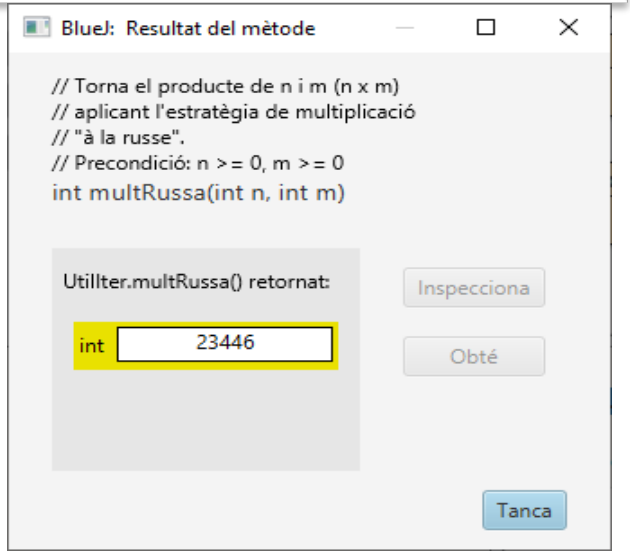
La iteració com estratègia de disseny
Exemple: multiplicació "à la russe"

BlueJ:exemplesT6

- El mètode **`multRussa`** de la classe d'utilitats **`UtilIter`** torna el resultat de multiplicar els seus dos arguments utilitzant l'estratègia de la multiplicació "à la russe".

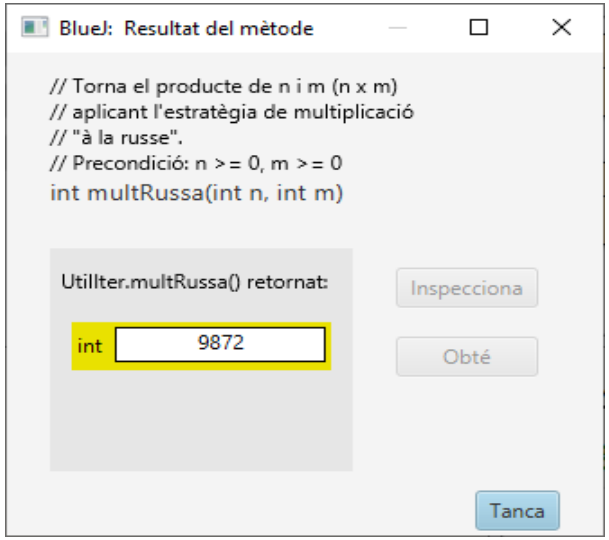
multiplicand	*multiplicador*	*producte*
19	1234 ✓	1234
9	2468 ✓	2468
4	~~4936~~	
2	~~9872~~	
1	19744 ✓	19744

23446



multiplicand	*multiplicador*	*producte*
8	~~1234~~	
4	~~2468~~	
2	~~4936~~	
1	9872 ✓	9872

9872



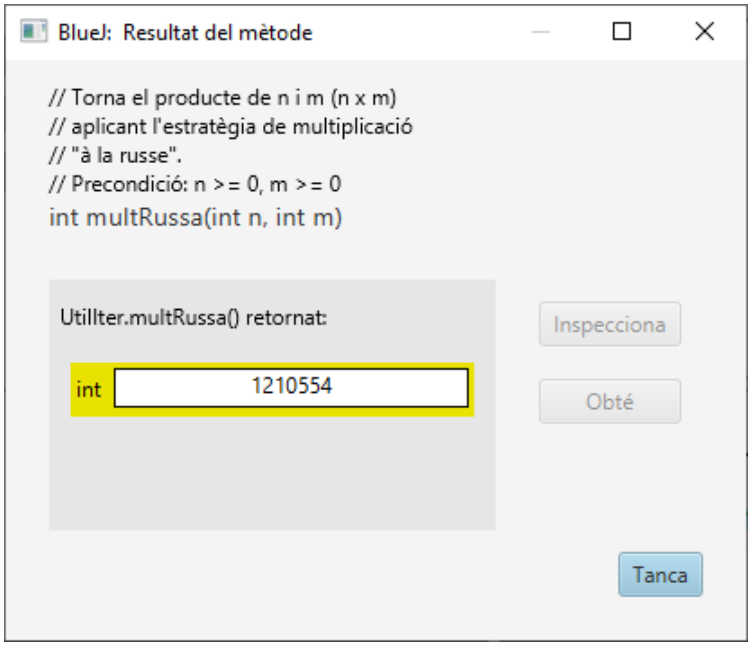
La iteració com estratègia de disseny
Exemple: multiplicació “à la russe”

BlueJ:exemplesT6

- El mètode `multRussa` de la classe d’utilitats `UtilIter` torna el resultat de multiplicar els seus dos arguments utilitzant l’estratègia de la multiplicació “à la russe”.

multiplicand	*multiplicador*	*producte*
981	1234 ✓	1234
490	~~2468~~	
245	4936 ✓	4936
122	~~9872~~	
61	19744 ✓	19744
30	~~39488~~	
15	78976 ✓	78976
7	157952 ✓	157952
3	315904 ✓	315904
1	631808 ✓	631808

1210554



La iteració com estratègia de disseny
Exemple: multiplicació “à la russe”

BlueJ:exemplesT6

- El mètode **`multRussa`** de la classe d’utilitats **`UtilIter`** torna el resultat de multiplicar els seus dos arguments utilitzant l’estratègia de la multiplicació “à la russe”

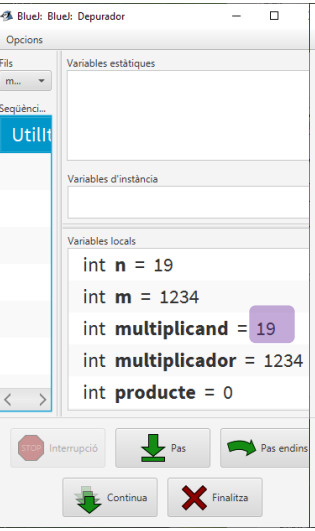
```
/** Torna el producte de n i m (n x m)
 *  aplicant l'estratègia de multiplicació
 *  "à la russe".
 *  Precondició: n >= 0, m >= 0
 */
public static int multRussa(int n, int m) {
    int multiplicand = n, multiplicador = m, producte = 0;
    while (multiplicand != 0) {
        if (multiplicand % 2 != 0) {
            producte = producte + multiplicador;
        }
        multiplicand = multiplicand / 2;
        multiplicador = multiplicador + multiplicador;
    }
    return producte;
}
```

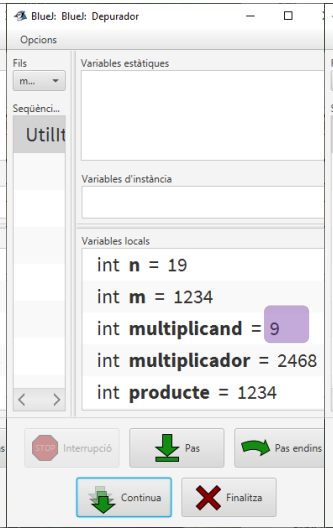
La iteració com estratègia de disseny
Exemple: multiplicació “à la russe”

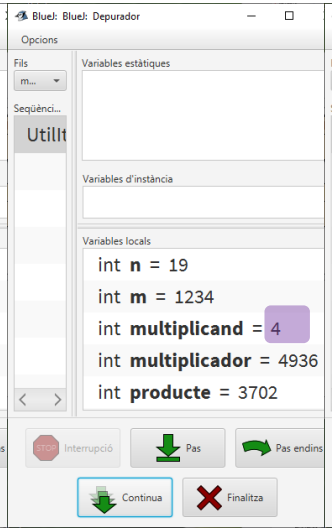
BlueJ:exemplesT6

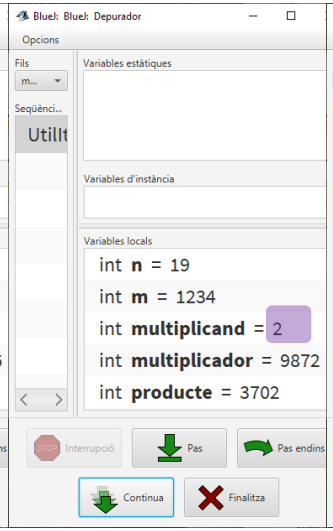
- Ficant un punt de ruptura al començament del bucle del mètode **`multRussa`** s’obté la traça següent per a `n = 19` i `m = 1234`. Observa com es modifiquen les variables en cada iteració, fins aplegar a la darrera on la guarda es fa falsa (`multiplicand` val 0) i l’execució del mètode acaba.

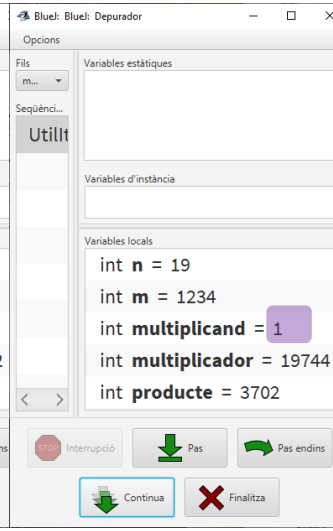
```
public static int multRussa(int n, int m) {
    int multiplicand = n, multiplicador = m, producte = 0;
    while (multiplicand != 0) {
        if (multiplicand % 2 != 0) {
            producte = producte + multiplicador;
        }
        multiplicand = multiplicand / 2;
        multiplicador = multiplicador + multiplicador;
    }
    return producte;
}
```

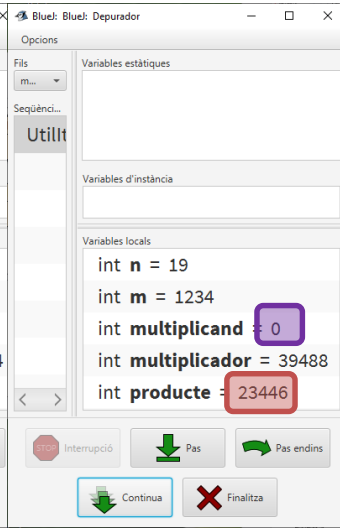










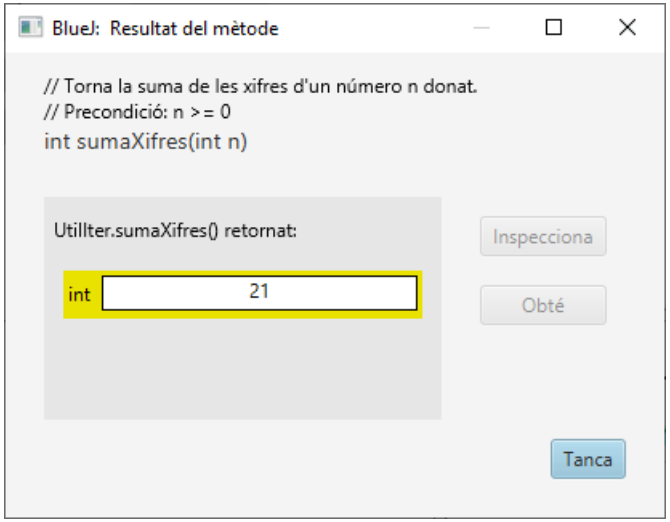


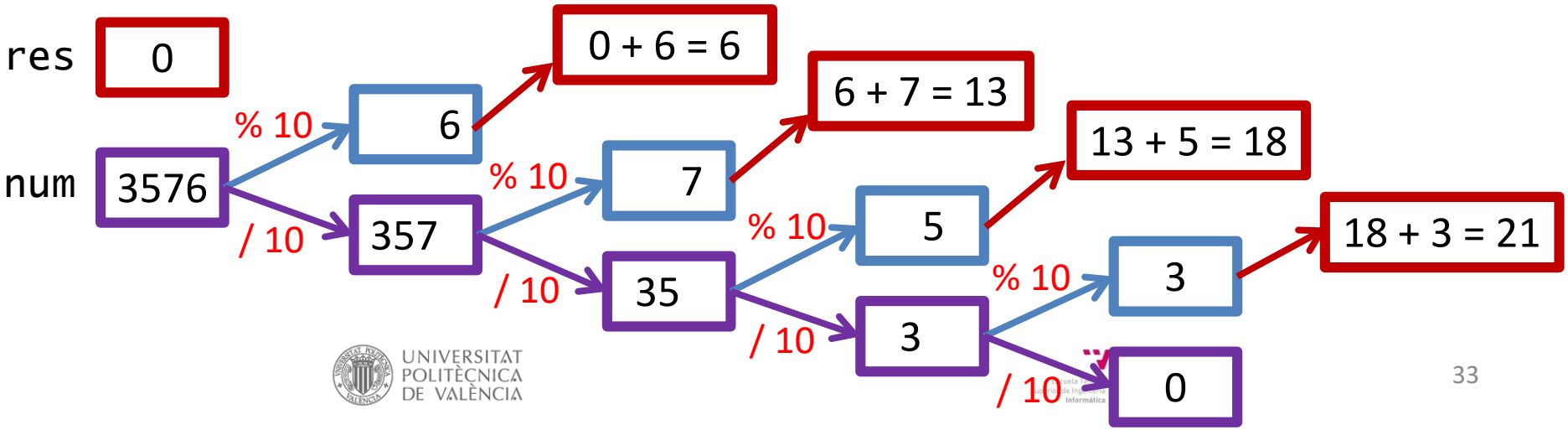
La iteració com estratègia de disseny
Exemple: suma xifres d'un natural

BlueJ:exemplesT6

- El mètode **sumaXifres** de la classe d'utilitats **UtilIter**, donat un número natural torna la suma de les seues xifres.

```
/** 0 <= n */
public static int sumaXifres(int n) {
    int res = 0, num = n;
    while (num > 0) {
        res = res + num % 10;
        num = num / 10;
    }
    return res;
}
```



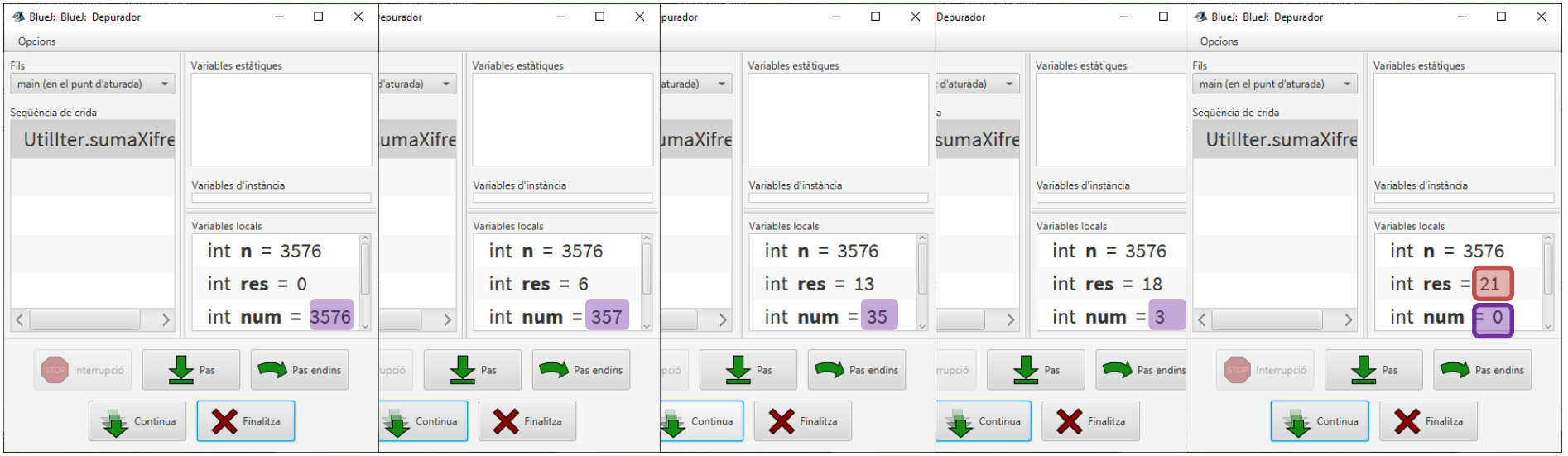


La iteració com estratègia de disseny
Exemple: suma xifres d'un natural

BlueJ:exemplesT6

- Fica un punt de ruptura al començament del bucle del mètode **`sumaXifres`** i observa com es modifiquen les variables en cada iteració, fins aplegar a la darrera en la que la guarda es fa falsa (num val 0) i l'execució del mètode acaba.

```
12  public static int sumaXifres(int n) {
13      int res = 0, num = n;
        while (num > 0) {
15          res = res + num % 10;
16          num = num / 10;
17      }
18      return res;
19  }
```



La iteració com estratègia de disseny
Exemple: MCD

- Determinar el Màxim Comú Divisor de dos números enters: **x**, **y** majors que 0.

➢ Algorisme MCD (Euclides 300 a.C.): siguen **x** i **y** els valors originals de les variables a i b, llavors :

Mentre a i b no siguen iguals, canviar el major dels dos per la diferència entre el major i el menor. Quan siguen iguals, qualsevol d'ells és el MCD.

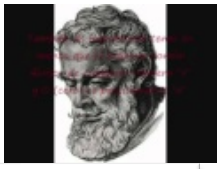
```
/** x>0, y>0 */
public static int mcd(int x, int y) {
    int a = x, b = y;
    while (a != b) {
        if (a > b) { a = a - b; }
        else { b = b - a; }
    }
    return a;
}
```

- Propietats en les que està basat aquest algorisme:
  - Al finalitzar cada iteració: `MCD(x,y) = MCD(a,b)`
  - Aquesta propietat és una conseqüència de la propietat matemàtica:

    `MCD(a,b) = MCD(a-b,b)`, si `a>b`

    `MCD(a,b) = MCD(a,b-a)`, si `a<b`
  - Quan finalment `a = b`, `MCD(x,y) = MCD(a,b) = a = b`

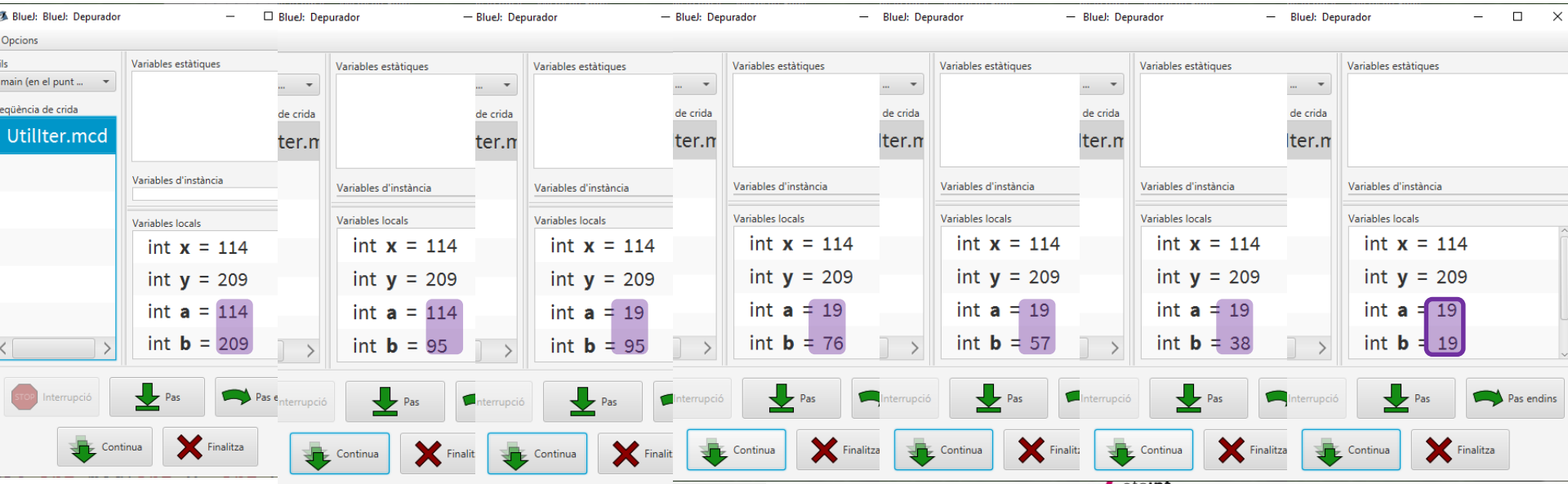
La iteració com estratègia de disseny
Exemple: MCD



BlueJ:exemplesT6

- Fica un punt de ruptura al començament del bucle del mètode **mcd** i observa com es modifiquen les variables en cada iteració, fins aplegar a la darrera en la que la guarda es fa falsa (a és igual a b) i l'execució del mètode acaba.

```
24  public static int mcd(int x, int y) {
25      int a = x, b = y;
        while (a != b) {
27          if (a > b) { a = a - b; }
28          else { b = b - a; }
29      }
30      return a;
31  }
```

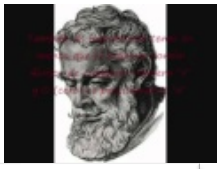


La iteració com estratègia de disseny
Exemple: MCD millorat

- Si x és múltiple de y --> x % y=0 --> MCD(x, y) = y.
- Si x % y> 0 --> x = q * y + r on q ≥ 0 i y > r > 0.
- Equivalentment: x – q * y = r > 0.
- Qualsevol divisor y de x, és divisor de x – q * y i també de r.
- El càlcul del MCD(x,y) es pot reduir al càlcul del MCD(y,r).

```
/** x>0, y>0 */
public static int mcd2(int x, int y) {
    int a = x, b = y;
    int r = a % b;
    while (r > 0) {
        a = b;
        b = r;
        r = a % b;
    }
    return b;
}
```

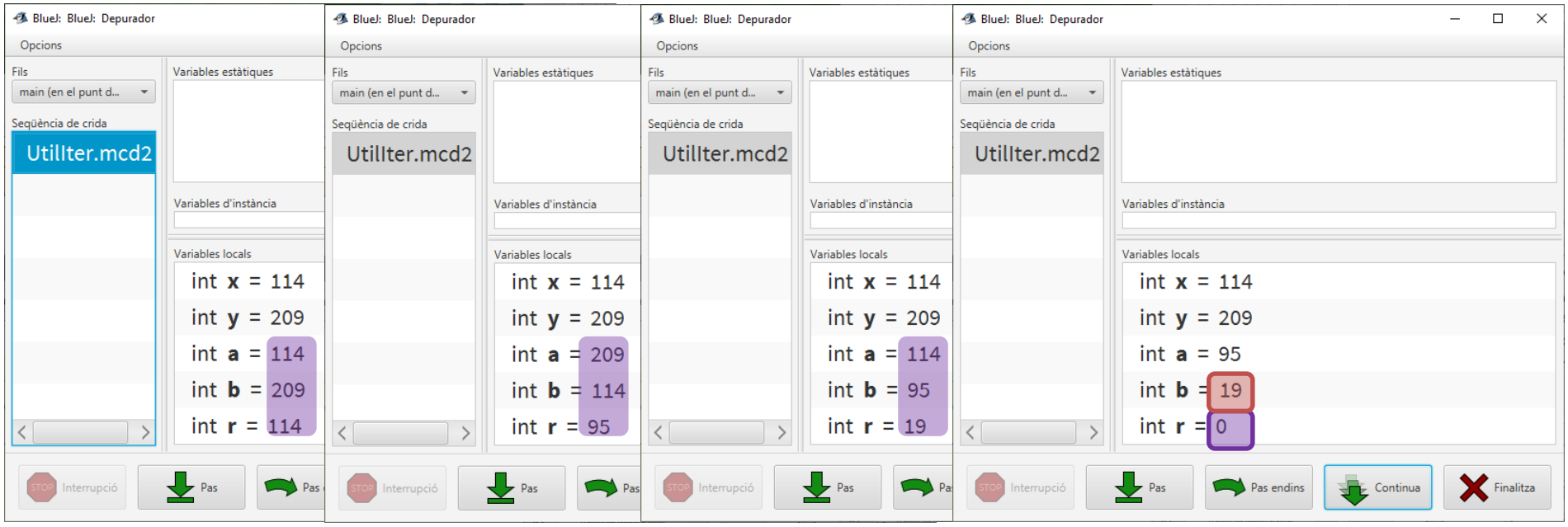
La iteració com estratègia de disseny
Exemple: MCD millorat



BlueJ:exemplesT6

- Fica un punt de ruptura al començament del bucle del mètode **mcd2** i observa com es modifiquen les variables en cada iteració, fins aplegar a la darrera en la que la guarda es fa falsa (r és 0) i l'execució del mètode acaba.

```
37      public static int mcd2(int x, int y) {
38          int a = x, b = y;
39          int r = a % b;
            while (r > 0) {
41              a = b;
42              b = r;
43              r = a % b;
44          }
45          return b;
46      }
```



La iteració com estratègia de disseny
Exercici: factorial

BlueJ:exercicisT6

- Dissenya en la classe **`FactorialMeu`** el mètode **`factorial`** per tal que, donat un `int n >= 1`, calcule `factorial(n) == 1 * 2 * ... * n`. Per a això, has de:
  1. Descobrir l'estructura iterativa del problema
  2. Determinar el cos, les instruccions d'inicialització i la guarda del bucle
  3. Escriure el mètode en Java utilitzant una instrucció `for`
  4. Comprovar la correcció i terminació del bucle dissenyat

- Utilitza el mètode dissenyat per a calcular el factorial de 12. Quin és el resultat?
- Utilitzant la calculadora del sistema, calcula 13! Quin resultat obtens?
- Utilitza el mètode dissenyat per a calcular el factorial de 13. Has obtingut el mateix resultat que amb la calculadora? Per què?
- Executa el mètode **`factorial`** utilitzant com argument un valor negatiu. Què passa?
- Modifica el codi del **`main`** de la classe **`FactorialMeu`** per tal que valide la dada d'entrada amb la finalitat d'assegurar que l'enter llegit està comprès entre 0 i 12.

Validació de dades
Exercici: lectura validada

BlueJ:exercicisT6

- El mètode **llegirIntPos** de la classe d'utilitats **Lectura**, donat un missatge i un objecte de tipus Scanner, llig des de teclat un valor enter positiu:

```
public static int llegirIntPos(String msg, Scanner tec) {
    int n;
    do {
        System.out.print(msg);
        n = tec.nextInt();
    } while (n < 0);
    return n;
}
```

- El codi del **main** de la classe **SumaFinsN** valida la dada d'entrada amb la finalitat d'assegurar que l'enter llegit és positiu.

```
public static void main(String[] args) {
    Scanner teclat = new Scanner(System.in);
    int num = Lectura.llegirIntPos("Introdueix el número: ", teclat);
    System.out.printf("La suma dels primers números naturals "
                      + "fins %d és %d\n", num, sumaFins(num));
}
```

Validació de dades
Exercici: lectura validada

BlueJ:exercicisT6

- Completa en la classe d'utilitats **`Lectura`** el mètode **`llegirIntEntre`** per a llegir un enter comprès en un interval [x , y].

- Completa el **`main`** de la classe programa **`TestLectura`** per tal que:
  - Sol.licite del teclat l'any actual, com un valor comprès entre 1900 i el del sistema.
  - Sol.licite del teclat un any en el futur (màxim 3000)
  - Mostre per pantalla la diferència entre els dos anys.

- Completa la classe d'utilitats **`Lectura`** amb mètodes homònims als dissenyats per a validar la lectura de números reals: **`llegirDoublePos`**, **`llegirDoubleEntre.`**

Validació de dades
Exercici: càlcul de la nota mitjana

BlueJ:exercicisT6

- El **main** de la classe **NotaMitjana** llig notes des del teclat i mostra per pantalla la nota mitjana. L'usuari indica que ha finalitzat l'entrada de notes teclejant -1.

```
public static void main(String[] args) {
     Scanner tec = new Scanner(System.in);
     System.out.println("Introdueix les notes, per a acabar -1:");
     double suma = 0; int num = 0;
     double nota = tec.nextDouble();
     while (nota != -1) {
          suma = suma + nota; num++;
          nota = tec.nextDouble();
     }
     if (num > 0) {
          System.out.printf("La mitjana de les %2d notes és %.3f\n",
                                num, suma / num);
     } else { System.out.println("No s'han introduit notes"); }
 }
```

- Què passaria si es prescindeix de l'última instrucció condicional?
- És necessari fer un càsting en el càlcul de la mitjana?

Validació de dades
Exercici: càlcul de la nota mitjana

BlueJ:exercicisT6

- Seria correcta aquesta versió?

```
public static void main(String[] args) {
     Scanner tec = new Scanner(System.in);
     System.out.println("Introdueix les notes, per a acabar -1:");
     double suma = 0; int num = 0;
     double nota;
     do {
          nota = tec.nextDouble();
          suma = suma + nota;
          num++;
     } while (nota != -1);
     if (num > 0) {
          System.out.printf("La mitjana de les %2d notes és %.3f\n",
                                num, suma / num);
     } else { System.out.println("No s'han introduit notes"); }
 }
```

Validació de dades
Exercici: càlcul de la nota mitjana

BlueJ:exercicisT6

- Modifica el codi del **main** per tal que valide que les notes introduïdes estan en [0,10].

```
public static void main(String[] args) {
     Scanner tec = new Scanner(System.in);
     System.out.println("Introdueix les notes, per a acabar -1:");
     double suma = 0; int num = 0;
     double nota;
     do {
          nota = tec.nextDouble();
          if (nota != -1) { suma = suma + nota; num++; }
     } while (nota != -1);
     if (num > 0) {
          System.out.printf("La mitjana de les %2d notes és %.3f\n",
                                num, suma / num);
     } else { System.out.println("No s'han introduit notes"); }
 }
```

Exercici: endevinar un número d'un interval en un nombre màxim d'intents

BlueJ:exercicisT6

- En el Tema 1, utilitzant *PSeInt*, vam escriure l'algorisme per resoldre aquest problema:

```
Algoritmo EndevinaONo
    Escribir "Estic pensant un número entre 0 i 50"
    numSecret <- azar(51)
    intents <- 5
    Escribir "Tens ", intents, " per endevinar-lo"
    Repetir
        Escribir "Quin número és?"
        Leer num
        Si num  = numSecret Entonces
            Escribir "Has encertat! El número és el ", numSecret
        SiNo
            Si num < numSecret Entonces
                Escribir "No has encertat. El número a endevinar és major"
            SiNo
                Escribir "No has encertat. El número a endevinar és menor"
            Fin Si
            intents <- intents - 1
            Escribir "Te queden ", intents, " per endevinar-lo"
        Fin Si
    Hasta Que num = numSecret O intents = 0
    Si intents = 0 Entonces
        Escribir "El número a endevinar era el ", numSecret
    SiNo
        Escribir "Has endevinat en ", (6 - intents), " intents"
    Fin Si
FinAlgoritmo
```

- Completa el **`main`** de la classe **`EndevinaONo`** per tal que resolga el mateix problema.

Recurrències
Exemple: terme n-èsim d'una sèrie

BlueJ:exercicisT6

- El mètode **terme** de la classe **Serie1** torna el terme n-èsim de la sèrie:

$$a_1 = 7, \quad a_i = a_{i-1} + i \,, \; i \geq 2$$

```
/** n>=1 */
public static int terme(int n) {
    int term = 7; // valor de l'últim terme calculat
    int i = 1;    // índex de l'últim terme calculat
    while (i < n) {
       i++;
       term = term + i;
    }
    return term;
}
```

iteració	i	term
0	1	7
1	2	9
2	3	12
3	4	16
4	5	21
5	6	27

i<n

i==n

n = 6

$a_1 = 7$

$a_2 = a_1 + 2 = 7 + 2 = 9$

$a_3 = a_2 + 3 = 9 + 3 = 12$

$a_4 = a_3 + 4 = 12 + 4 = 16$

$a_5 = a_4 + 5 = 16 + 5 = 21$

$a_6 = a_5 + 6 = 21 + 6 = 27$

Recurrències
Exemple: terme n-èsim d'una sèrie

BlueJ:exercicisT6

- El mètode **terme** de la classe **Serie1** torna el terme n-èsim de la sèrie:

$$a_1 = 7, \quad a_i = a_{i-1} + i \,, i \geq 2$$

```
/** n>=1 */
public static int terme(int n) {
    int term = 7; // valor de l'últim terme calculat
    int i = 2;    // índex del següent terme a calcular
    while (i <= n) {
       term = term + i;
       i++;
    }
   return term;
}
```

iteració	i	term
0	2	7
1	3	9
2	4	12
3	5	16
4	6	21
5	7	27

i<=n

i==n+1

```
/** n>=1 */
public static int terme(int n) {
    int term = 7;
    for (int i = 2; i <= n; i++) { term = term + i; }
    return term;
}
```

Recurrències
Exemple: suma dels n primers termes d'una sèrie

BlueJ:exercicisT6

- El mètode **suma** de la classe **Serie1** torna la suma dels n primers termes de la sèrie:

$a_1 = 7,$
$a_i = a_{i-1} + i, i \geq 2$

$$\text{suma} = \sum_{i=1}^{n} a_i = a_1 + a_2 + a_3 + \ldots + a_{(n-1)} + a_n$$

```
/** n>=1 */
public static int suma(int n) {
    int term = 7, suma = 7;
    for (int i = 2; i <= n; i++) {
        term = term + i;
        suma = suma + term;
    }
    return suma;
}
```

n = 6
a_1 = 7, suma = 7
a_2 = a_1+2 = 9, suma = 7+9 = 16
a_3 = a_2+3 = 12, suma = 16+12 = 28
a_4 = a_3+4 = 16, suma = 28+16 = 44
a_5 = a_4+5 = 21, suma = 44+21 = 65
a_6 = a_5+6 = 27, suma = 65+27 = 92

iteració	i	term	suma
0	2	7	7
1	3	9	16
2	4	12	28
3	5	16	44
4	6	21	65
5	7	27	92

Recurrències
Exercici: successió de Fibonacci

BlueJ:exercicisT6

- Completa el mètode **fibonacci** de la classe **FibonacciMeu** de forma que torne el terme n-èsim de la sèrie de Fibonacci (n>=0):

```
/** n>=0 */
public static int fibonacci(int n) {
   int penultim = 0, // valor del penúltim terme calculat
       ultim = 1,    // valor de l'últim terme calculat
       i = 1,        // índex de l'últim terme calculat
       term = 1;     // valor del terme i-èsim
   while (      ) {




   }

   return ultim;
}
```

$a_0 = 0, \quad a_1 = 1,$
$a_n = a_{n-1} + a_{n-2}, \; n \geq 2$

0 1 1 2 3 5 8 13 21 ...

iteració	term	penultim	ultim	i

$n = 5$
$a_0 = 0$
$a_1 = 1$
$a_2 = a_1 + a_0 = 1 + 0 = 1$
$a_3 = a_2 + a_1 = 1 + 1 = 2$
$a_4 = a_3 + a_2 = 2 + 1 = 3$
$a_5 = a_4 + a_3 = 3 + 2 = 5$

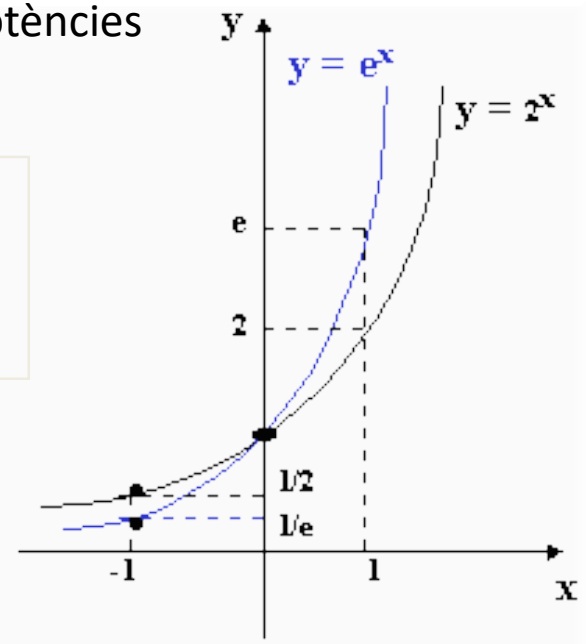
Recurrències
Exercici: càlcul aproximat d'e^x

BlueJ:exercicisT6

- La funció exponencial e^x pot definir-se com una sèrie de potències (**Desenvolupament en Sèrie de Taylor** - **AMA**):

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots$$

Si x = 1: $$e = \sum_{n=0}^{\infty} \frac{1}{n!}$$

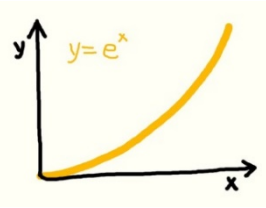


$$e = \frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \cdots$$

$e \approx 2{,}718281828459045235360...$

- Es vol dissenyar una iteració per tal de realizar el càlcul del terme n-ésim de la sèrie i, sumant aquestos termes, obtindre una aproximació al valor d' e^x

Recurrències
Exercici: càlcul aproximat d'e^x



$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots$$

Cada terme es pot calcular en funció del terme anterior

BlueJ:exercicisT6

- Variables del bucle:

  `i`: comptador d'iteracions, al principi val 0
  ---> índex de l'últim terme calculat
  `term`: valor del terme i-ésim, el primer val 1

- Elements del bucle:

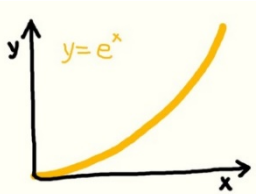
```
Inici:

Cos:


Guarda:
```

i	term	en funció de l'anterior
0	$x^0/0! = 1$	
1	$x^1/1! = x$	= 1*x
2	$x^2/2!$	= x*(x/2)
3	$x^3/3!$	= x*(x/2)*(x/3)
4	$x^4/4!$	= x*(x/2)*(x/3)*(x/4)
…	…	…
i	$x^i/i!$	= x*(x/2)*..*(x/(i-1))*(x/i)
…	…	…
n	$x^n/n!$	= x*(x/2)*..*(x/(n-1))*(x/n)

Recurrències
Exercici: càlcul aproximat d'e^x

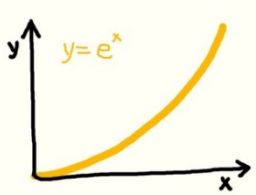


BlueJ:exercicisT6

- Completa els següents mètodes en la classe **CalculEx** :
  - **mostraTerms**, per a mostrar per pantalla els n primers termes d'aquesta sèrie
  - **sumaExpFins**, per a calcular la suma dels n primers termes d'aquesta sèrie
- Executa en el *CodePad* el mètode **sumaExpFins** per a obtenir una aproximació al valor del número **e** mitjançant la suma de 5, 10, 20 i 30 termes i observa com les diferències respecte al valor de **Math.E** se van fent cada vegada més menudes.

```
Math.E
   2.718281828459045   (double)
CalculEx.sumaExpFins(1, 5)
   2.7166666666666663   (double)
CalculEx.sumaExpFins(1, 10)
   2.7182818011463845   (double)
CalculEx.sumaExpFins(1, 20)
   2.7182818284590455   (double)
CalculEx.sumaExpFins(1, 30)
   2.7182818284590455   (double)
```

Recurrències
Exercici: càlcul aproximat d'e^x



BlueJ:exercicisT6

- Completa el mètode **exp** en la classe **CalculEx** de manera que, donat un valor real `x` i un valor real `epsilon` molt menut (`epsilon` > 0), calcule el valor d'e^x sumant tots els termes generats fins que l'últim calculat siga menor que `epsilon`.

- Executa en el *CodePad* el mètode **exp** amb distintes dades i comprova que el resultat obtingut és el correcte comparant-lo amb el resultat de **Math.exp(x)**.

- Executa en el *CodePad* una expressió per a calcular la diferència entre el valor que torna el mètode exp (amb `epsilon` 10^{-5}) i el valor de la constant **Math.E**.

- Fica un punt de ruptura en l'avaluació de la guarda del mètode **exp** i executa'l en el *CodePad* per a calcular $e^{0.0025}$ i $e^{2.5}$ i observa el valor de les variables locals. Quantes iteracions fa el bucle en cada cas?

Bucles niuats
Exemple: les taules de multiplicar

BlueJ:exemplesT6

- El mètode **mostrarTaules** de la classe **TaulaDeMultiplicar2**, mostra per pantalla les taules de multiplicar de l'1 al 9, una a continuació de l'altra, utilitzant el mètode **mostrarTaulaDel**:

```
for (int i = 1; i <= 9; i++) {
    mostrarTaulaDel(i);
}
```

- O també, sense cridar al mètode **mostrarTaulaDel**:

```
for (int i = 1; i <= 9; i++) {
    System.out.printf("Taula del %2d\n", i);
    for (int j = 1; j <= 10; j++) {
        System.out.printf("%2d x %2d es %2d\n", i, j, i * j);
    }
}
```

Bucles niuats
Exercici: traça

BlueJ:exercicisT6

- Què mostra per pantalla el següent codi?

```
int nfil = 4, ncol = 3;
for (int i = 1; i <= nfil; i++) {
    for (int j = 1; j <= ncol; j++) {
        System.out.print(i + "-" + j + "  ");
    }
    System.out.println();
}
```

- Executa el mètode **traza1** de la classe **Traces** i comprova la teua resposta.

Bucles niuats
Exercici: traça

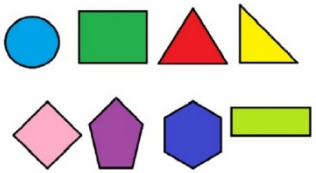
BlueJ:exercicisT6

- Què mostra per pantalla el següent codi?

```
for (int i = 0; i <= n; i++) {
    // Seqüència de Z de longitud i:
    for (int j = 1; j <= i; j++) {
        System.out.print('Z');
    }
    // Seqüència de A de longitud n:
    for (int j = 1; j <= n; j++) {
        System.out.print('A');
    }
    // Seqüència de Z de longitud n-i:
    for (int j = 1; j <= n - i; j++) {
        System.out.print('Z');
    }
    System.out.println();
}
```

- Executa el mètode **traza2** de la classe **Traces** i comprova la teua resposta.

Bucles niuats
Exercici: dibuixar un rectangle amb *



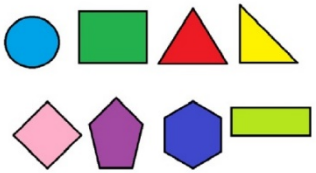
BlueJ:exercicisT6

- El mètode **rectangle** de la classe **Dibuixa**, mostra per pantalla un rectangle de base i altura donades utilitzant *:

```
public static void rectangle(int base, int altura) {
    for (int i = 0; i < altura; i++) {
        for (int j = 0; j < base; j++) {
            System.out.print("*");
        }
        System.out.println();
    }
}
```

- Executa el mètode amb base 3 i altura 2, base 6 i altura 3 i base 4 i altura 5. Què passa si es crida al mètode amb base 0 i altura 0?

Bucles niuats
Exercici: dibuixar altres figures amb *



BlueJ:exercicisT6

- Completa en la classe **`Dibuixa`** el mètode **`triangleRec`** per tal que, donada una base, mostre per pantalla un triangle rectangle utilitzant *.

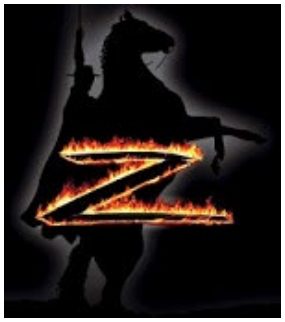
```
Dibuixa.triangleRec(3);
Dibuixa.triangleRec(4);
Dibuixa.triangleRec(5);
```

BlueJ: BlueJ: Fines...

Opcions

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

Bucles niuats
Exercici: dibuixar altres figures amb *



BlueJ:exercicisT6

- Completa en la classe **`Dibuixa`** el mètode **`zeta`** per tal que, donat un enter n>=0, mostre per pantalla una zeta utilitzant el caràcter 'Z' en n línies:

```
Dibuixa.zeta(5);
System.out.println();
Dibuixa.zeta(7);
```

BlueJ: BlueJ: Finestra ...

Opcions

```
 ZZZZZ
   Z
  Z
 Z
ZZZZZ

ZZZZZZZ
      Z
     Z
    Z
   Z
  Z
ZZZZZZZ
```

Instruccions iteratives

Exercicis de CAP

- Realització en CAP, en l'ordre indicat, dels exercicis següents:
  - Dibujar Rectángulo con * (clau CCDIJ4ai)
  - Dibujar Triángulo con * (clau CCDIK4ai)
  - Dibujar Árbol de Navidad con * (clau CCDIL4ai)

Solució visible des del 25/11