UF2 – DISSENY MODULAR
CURS 2013-14

A01

MÈTODES

Introducció

- Un **mètode** és un mòdul de programa que conté una sèrie de sentències que realitzen una tasca.

- Els **mètodes** poden ser utilitzats per a definir codi reutilitzable, així com organitzar i simplificar codi

- Per a executar un mètode, l'has d'**invocar** o **cridar.**

Introducció

```java
public class PrimerMetode {

    public static void main(String[] args) {

        System.out.println("Companyia Carles Vallbona");
        System.out.println("C. de les Moreres, 14");
        System.out.println("08526 Granollers");
        System.out.println("Aquest és el meu primer programa de mètodes!");
    }
}
```

El mètode *main()* en aquesta classe conté dues sentències. La primera és una crida al mètode *dadesCompanyia()*

```java
public class PrimerMetode {

    public static void main(String[] args) {

        dadesCompanyia();
        System.out.println("Aquest és el meu primer programa de mètodes!");
    }

    public static void dadesCompanyia() {

        System.out.println("Companyia Carles Vallbona");
        System.out.println("C. de les Moreres, 14");
        System.out.println("08526 Granollers");
    }
}
```

Introducció

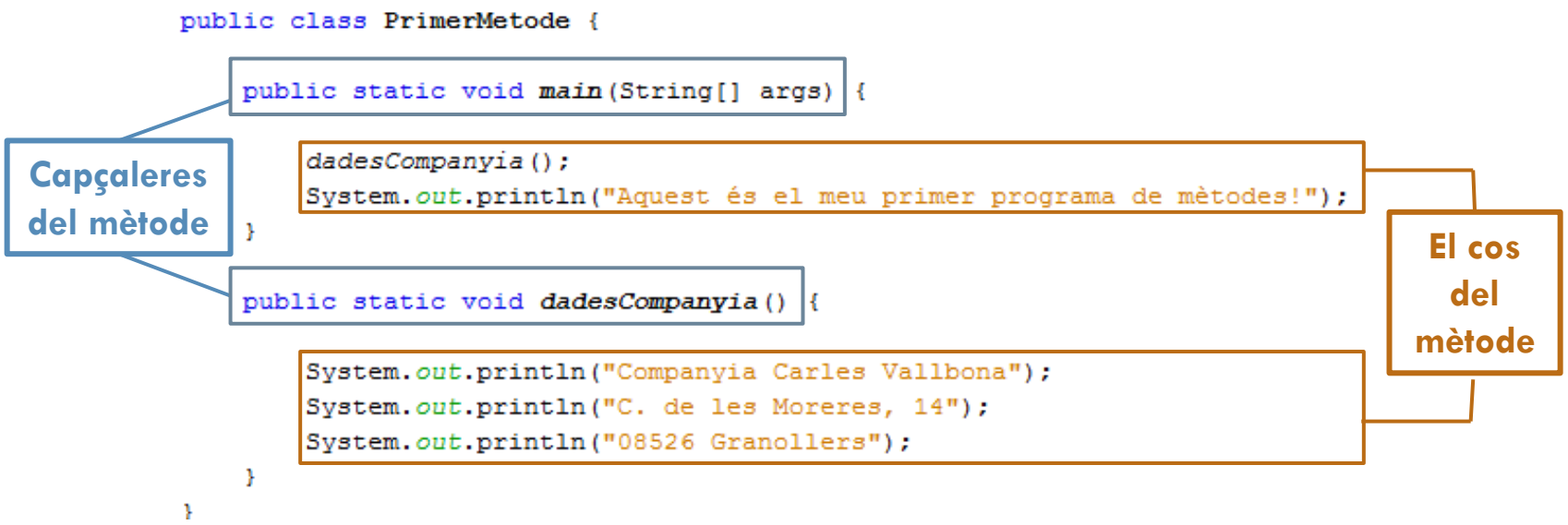
```
public static int sumar(int i1, int i2) {
    int resultat = 0;
    for (int i = i1; i <= i2; i++) {
        resultat = resultat + i;
    }
    return resultat;
}

public static void main(String[] args) {

    System.out.printf("La suma d'1 fins a 10 és: %d. \n", sumar(1,10));
    System.out.printf("La suma de 20 fins a 37 és: %d. \n", sumar(20,37));
    System.out.printf("La suma de 35 fins a 49 és: %d. \n", sumar(35,49));
}
```

UN MÈTODE ÉS UNA COL.LECCIÓ DE SENTÈNCIES AGRUPADES JUNTES PER A EXECUTAR UNA OPERACIÓ.

"EN ALTRES PARAULES: FAS EL TREBALL UN COP I EL POTS FER SERVIR MOLTES VEGADES"

Definir un mètode

- Cada mètode ha d'incloure dues parts:



Capçalera d'un mètode

- La capçalera d'un mètode és la primera línia del mètode i conté el següent:
  - Especificadors d'accés opcionals
  - Un tipus de retorn
  - Un identificador o nom del mètode
  - Parèntesi o paràmetres del mètode

Capçalera del mètode – Especificadors

- L'especificador d'accés per a un mètode a Java pot ser un dels següents **modificadors**:
  - public
  - private
  - protected

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

        public static void dadesCompanyia()
```

Capçalera del mètode – Retorn

- Un mètode pot o no retornar un valor.

```
public static void dadesCompanyia()
```

- Un tipus de retorn descriu el tipus de dada que el mètode envia al mètode que l'ha cridat.

```
public static int sumar(int i1, int i2)
```

Capçalera del mètode – Nom

- L'identificador conté el nom del mètode.

```
public static void dadesCompanyia()

public static int sumar(int i1, int i2)
```

- L'identificador del mètode ha de ser una paraula sense espais i segueix les mateixes regles que els identificadors de variables (*camelStyle*).

Capçalera del mètode – Parèntesi

- El parèntesi poden contenir dades que s'enviaran al mètode. Aquestes dades s'anomenen paràmetres.
- Els paràmetres són opcionals i, per tant, poden existir mètodes que no en continguin cap.
  - Mètode amb paràmetres:

```
public static int sumar(int i1, int i2)
```

  - Mètode sense paràmetres:

```
public static void dadesCompanyia()
```

Cridar un mètode

- Cridar un mètode vol dir executar el codi que hi haurà dins d'aquest mètode. Hi ha dues maneres de cridar un mètode, depenent de si el mètode retorna o no valors.
  - Si el mètode retorna un valor, la crida al mètode es tractada com un **valor:**
    ```
    int resultat = sumar(20,37);
    ```
    o
    ```
    System.out.printf("La suma d'1 fins a 10 és: %d. \n", sumar(1,10));
    ```
  - Si el mètode retorna *void*, la crida al mètode ha de ser una **sentència:**
    ```
    dadesCompanyia();
    ```

Modularització

- La modularització divideix un gran problema en subproblemes fent que el codi sigui fàcil de mantenir i depurar.
  - En l'exemple anterior per a sumar dos números si ho fem a través de mètodes tenim els següents avantatges:
    - Aïlla el problema per al càlcul de la suma de la resta del codi en el mètode principal. Per tant, la lògica es torna clara i el programa és més fàcil de llegir.
    - Es redueix l'àmbit de depuració en els possibles errors en el càlcul de la suma.
    - El mètode suma pot ser reutilitzat per un altre programa.