Programació I - T5 - Encapsulament: classes, objectes i mètodes

Universitat de Barcelona
Grau en Enginyeria Informàtica

14 de novembre de 2012

Introducció
Definició de classes i objectes
Encapsulació: gets i sets
Encapsulació: constructors
Mètodes de classe i mètodes d'objecte
Exemples de classes ja existents
Pas de paràmetres: tipus valor, objectes immutables i objectes mutables

1. Tema 5. Encapsulament: classes, objectes i mètodes
   - Introducció
   - Definició de classes i objectes
     - Definició d'atributs
     - Definició de mètodes
   - Encapsulació: gets i sets
   - Encapsulació: constructors
   - Mètodes de classe i mètodes d'objecte
   - Exemples de classes ja existents
   - Pas de paràmetres: tipus valor, objectes immutables i objectes mutables

UNIVERSITAT DE BARCELONA

Introducció
Definició de classes i objectes
Encapsulació: gets i sets
Encapsulació: constructors
Mètodes de classe i mètodes d'objecte
Exemples de classes ja existents
Pas de paràmetres: tipus valor, objectes immutables i objectes mutables

Filosofies de programació: diferents mètodes d'encapsulament

- **Disseny descendent**: divisió del problema original en **subproblemes** més senzills, de forma que solucionant els subproblemes acabarem solucionant el problema original.
- **Programació modular**: divisió del programa desitjat en **subprogrames** més senzills, de forma que combinant els subprogrames senzills podem crear el programa desitjat.
- **Programació orientada a objectes**: divisió del programa desitjat en **objectes** (fragments de codi dedicats a manipular un subconjunt de les dades del programa desitjat), de forma que la interacció (**missatges**) entre objectes resol el problema original.
- **Objecte**: **encapsulament** de dades + codi. Les dades que conté (**atributs**) només es poden manipular a través del codi

Introducció
Definició de classes i objectes
Encapsulació: gets i sets
Encapsulació: constructors
Mètodes de classe i mètodes d'objecte
Exemples de classes ja existents
Pas de paràmetres: tipus valor, objectes immutables i objectes mutables

Programació orientada a objectes:Atributs

- **Classe**: *definició* d'atributs i *mètodes comuns* (o operacions).
- **Objecte**: *instància* d'una classe; *encapsulació* de:
  - **Propietats**: dades concretes
  - **Mètodes**: codi per manipular-les

- Els atributs d'una classe són els conjunts de valors que representa la classe i valors que serveixen per controlar l'estat intern d'un objecte d'una classe.
- Poden ser publics o privats, depenent de la visibilitat que es permet des d'altres parts del codi.
- Avantatges de ser *privats*: Independència de la implementació per part de l'usuari de la classe.

Introducció
Definició de classes i objectes
Encapsulació: gets i sets
Encapsulació: constructors
Mètodes de classe i mètodes d'objecte
Exemples de classes ja existents
Pas de paràmetres: tipus valor, objectes immutables i objectes mutables

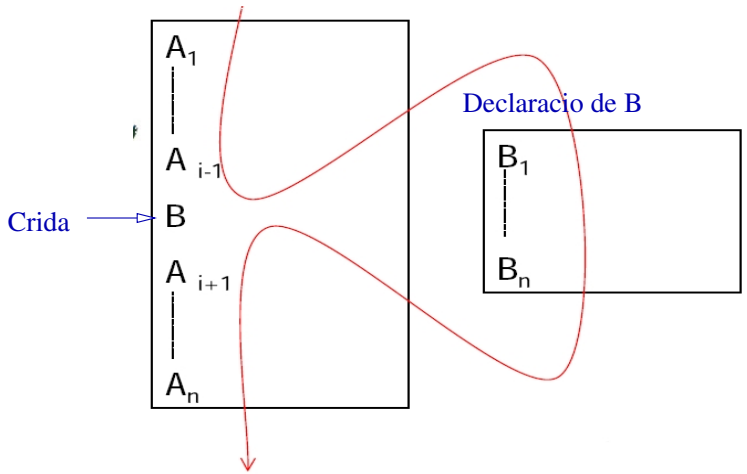
Programació orientada a objectes: Mètodes

- Sovint, es troba que trossos del programa s'han d'executar diferents vegades o que sempre s'han d'executar quan es declara una variable d'un cert tipus referència.
- Els mètodes són el mecanisme per AGRUPAR un conjunt d'instruccions i REFERENCIAR-lo sota un únic NOM.
- Poden ser publics o privats, depenent de la visibilitat que es permet des d'altres parts del codi.
- Avantatges: Possible reutilització, Flexibilitat, Claretat, Robustesa del codi i permet fer proves per separat.
- Permeten donar funcionalitats de les dades d'una classe concreta.
- Permeten inicialitzar els atributs d'una classe (mètode especial anomenat constructor de la classe)

Introducció
Definició de classes i objectes
Encapsulació: gets i sets
Encapsulació: constructors
Mètodes de classe i mètodes d'objecte
Exemples de classes ja existents
Pas de paràmetres: tipus valor, objectes immutables i objectes mutables

Mecanisme: Com es declara i s'utilitza un mètode?

- **Declaració del mètode**: és la definició del mètode mitjançant el conjunt d'instruccions que encapsula.
- **Crida al mètode**: és la utilització del mètode per part d'un altre conjunt d'instruccions (per exemple: quan s'utilitza el Math.sqrt(x)).



Introducció
Definició de classes i objectes
Encapsulació: gets i sets
Encapsulació: constructors
Mètodes de classe i mètodes d'objecte
Exemples de classes ja existents
Pas de paràmetres: tipus valor, objectes immutables i objectes mutables

Sintaxi: Mètodes d'objecte

- **Mètode d'objecte**: és un bloc de sentències etiquetat amb un *identificador*, que pot rebre dades d'entrada i generar dades de sortida (el mateix esquema que un programa complet).
- **Sintaxi**:

Declaració:

```
public <tipus_retorn>
<nom_metode>(<paràm_formals>) {
    <declaració_variables>
    <sentències>
    return <expressió>;
}
```

Introducció
Definició de classes i objectes
Encapsulació: gets i sets
Encapsulació: constructors
Mètodes de classe i mètodes d'objecte
Exemples de classes ja existents
Pas de paràmetres: tipus valor, objectes immutables i objectes mutables

Mètodes d'objecte: crida

Crida: Utilització

```
<id_variable>.<nom_mètode>(<paràmetres_actuals>)
```

- Els paràmetres són el mecanisme per a comunicar dades entre la crida i el mètode concret.
- Els paràmetres de la crida s'anomenen paràmetres actuals i les de la definició o declaració del mètode s'anomenen paràmetres formals.

Mètode sense paràmetres o sense retorn I

- Pot ser que un mètode no rebi cap paràmetre d'entrada i/o no retorni cap tipus de valor.
- Els mètodes que **no reben paràmetres**: no contenen res entre els parèntesis de la capçalera.
- Els mètodes que **no retornen res**: porten indicat **void** com a `<tipus_retorn>` i no han de contenir la instrucció `return`.

JugadorPrivats.java

```
import java.util.Scanner;

public class JugadorPrivats {
  private int   any;
  private float alcada;

  public void  setAny(int n)       { any = n; }
  public int   getAny()            { return any; }

  public void  setAlcada(float f) { alcada = f; }
  public float getAlcada()        { return alcada; }
}
```

Mètode d'objecte sense paràmetres o sense retorn I

- La crida al mètode equival al valor retornat, per tant es pot utilitzar en totes les ocasions que es podria posar una expressió del `<tipus_retorn>` indicat.

Mètode d'objecte sense paràmetres o sense retorn II

TuplaPrivatsBasic.java

```
import java.util.Scanner;

public class TuplaPrivatsBasic {
  public static void main (String [] args) {
    int a; float h;
    JugadorPrivats tupla = new JugadorPrivats();

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

    System.out.println("Any?");
    a = sc.nextInt();
    System.out.println("Alçada?");
    h = sc.nextInt();

    tupla.setAny(a);
    tupla.setAlcada(h);

    System.out.println("Any="     + tupla.getAny());
    System.out.println("Alçada="  + tupla.getAlcada());
  }
}
```

Introducció
Definició de classes i objectes
Encapsulació: gets i sets
Encapsulació: constructors
Mètodes de classe i mètodes d'objecte
Exemples de classes ja existents
Pas de paràmetres: tipus valor, objectes immutables i objectes mutables

Paràmetres: Sintaxi:

- **Paràmetres formals** [*a la declaració del mètode*]: llista de `<tipus> <id_param>` separada per comes si n'hi ha més d'un.
- **Paràmetres actuals** [*a la crida del mètode*]: llista d'`<expressions>` separada per comes si n'hi ha més d'una.
- S'ha de garantir que:
  - Coincideixi el nombre de paràmetres actuals i el nombre de paràmetres formals
  - Coincideixi en ordre, el tipus i el significat dels paràmetres formals i dels paràmetres actuals

Pas de paràmetres I

- d'**entrada**: són aquells que el mètode que és cridat UTILITZA però NO modifica:
  - Es fan servir dins del mètode
  - es consulta el seu valor

JugadorPrivats2.java

```
public class JugadorPrivats2 {
    private int   any;
    private float alcada;

    public void  setAny(int n)       { any = n; }
    public void  setAlcada(float f) { alcada = f; }
}
```

Pas de paràmetres II

TuplaPrivatsBasic2.java

```
import java.util.Scanner;

public class TuplaPrivatsBasic2 {
  public static void main (String [] args) {
    int a; float h;
    JugadorPrivats2 tupla = new JugadorPrivats2();

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

    System.out.println("Any?");
    a = sc.nextInt();
    System.out.println("Alçada?");
    h = sc.nextInt();

    tupla.setAny(a);
    tupla.setAlcada(h);

  }
}
```

Pas de paràmetres I

- de **sortida**: són paràmetres que el mètode cridat MODIFICA.
  - el mètode dóna un resultat sobre ells
  - abans de la crida al mètode no tenen un valor que s'hagi de mantenir

JugadorPrivats3.java

```
public class JugadorPrivats3 {
    private int   any;
    private float alcada;

    public void  setAny(int n)       { any = n; }
    public void  setAlcada(float f) { alcada = f; }
    public JugadorPrivats3 copia() {
        JugadorPrivats3 aux;
        aux = new JugadorPrivats3();
        aux.any = this.any;
        aux.alcada = this.alcada;
        return aux;
    }
}
```

Pas de paràmetres II

TuplaPrivatsBasic3.java

```
import java.util.Scanner;

public class TuplaPrivatsBasic3 {
  public static void main (String [] args) {
    int a; float h;
    JugadorPrivats3 tupla = new JugadorPrivats3();
    JugadorPrivats3 tupla2;

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

    System.out.println("Any?");
    a = sc.nextInt();
    System.out.println("Alçada?");
    h = sc.nextInt();

    tupla.setAny(a);
    tupla.setAlcada(h);

    tupla2 = tupla.copia();
  }
}
```

Mètode amb paràmetres de sortida I

- d'**entrada/sortida**: són aquells paràmetres que tenen un valor que és consultat dins del mètode i alhora sobre ells s'expressa un resultat.

Àmbit de validesa I

- Anomanem **àmbit de validesa** d'un identificador a la zona del codi en que esta disponible.
- Les variables declarades a cada mètode no són **visibles** des d'altres mètodes.
- els paràmetres d'un mètode només són visibles dins del mètode.
- els atributs privats són visibles dins de la classe on han estat definits.
- la forma de compartir la informació entre classes diferents és enviant-la a través dels paràmetres i recollint-la a través del valor de sortida.

Àmbit de validesa II

MetodeMultiple.java

```
import java.util.Scanner;
public class MetodeMultiple {
  public static void main (String [] args) {
       Triangle t;
       Scanner sc;
       sc = new Scanner(System.in);

       t = new Triangle();
       System.out.println("a?");
       t.setAmplada(sc.nextDouble());
       System.out.println("b?");
       t.setAlcada(sc.nextDouble());
       System.out.println("h:␣"+ t.hipotenusa());
  }
}
```

Àmbit de validesa III

Triangle.java

```
public class Triangle {
    double ampl;
    double alc;

    public void setAmplada(double a) {
        ampl = a;
    }
    public void setAlcada(double a) {
        alc = a;
    }
    public double hipotenusa() {
        double total;
        total = Math.sqrt( quadrat(ampl) + quadrat(alc) );
        return total;
    }
    public  double quadrat(double n) {
        double resultat;
        resultat = n*n;
        return resultat;
    }
}
```

Introducció
Definició de classes i objectes
Encapsulació: gets i sets
Encapsulació: constructors
Mètodes de classe i mètodes d'objecte
Exemples de classes ja existents
Pas de paràmetres: tipus valor, objectes immutables i objectes mutables

Temps de vida de les variables

- Una entitat existeix i és accessible en l'àmbit del mètode on es declara
- En acabar-se un mètode, desapareixen totes les variables que pertànyen al seu àmbit de visibilitat.

Paràmetres de mètodes de tipus array: Entrada I

- Inicialitzar una taula d'enters de la classe ConjuntEnters.

ConjuntEnters.java

```
public class ConjuntEnters {
    private int[] taula;
    private int numElems;

    public void iniTaula (int [] t, int n) {
        taula = new int[n];
        for (int i=0; i<n; i++) {
            taula[i] = t[i];
        }
        numElems = n;
    }
}
```

Paràmetres de mètodes de tipus array: Entrada II

TaulesEntrada.java

```
public class TaulesEntrada {
  public static void main (String [] args) {
      int n;
      ConjuntEnters cj;

      int taula[] = {1,3,2,5,10};
      n = 5;
      cj = new ConjuntEnters();

      cj.iniTaula(taula, n);
  }
}
```

Paràmetres de mètodes de tipus array: Sortida I I

- Extreure un subconjunt d'elements d'un conjunt d'enters en una taula.

ConjuntEnters2.java

```
public class ConjuntEnters2 {
    private int[] taula;
    private int numElems;

    public void iniTaula (int [] t, int n) {
        taula = new int[n];
        for (int i=0; i<n; i++) {
            taula[i] = t[i];
        }
        numElems = n;
    }
    public int[] subConjunt(int idx1, int idx2) {
        int[] t;
        if (idx2>idx1 && idx1>=0 && idx1<=numElems && idx2>=0 && idx1<=numElems) {
            t = new int[idx2-idx1+1];
            for (int i=0; i<idx2-idx1+1; i++) {
                t[i] = taula[idx1+i];
            }
        } else t = null;
        return t;
    }

}
```

Paràmetres de mètodes de tipus array: Sortida I II

TaulesSortidaI

```
public class TaulesSortidaI {
  public static void main (String [] args) {
      int[] t = {1,5,8,12};
      ConjuntEnters2 cj;
      cj = new ConjuntEnters2();
      cj.iniTaula(t, 4);

      t = cj.subConjunt(2,3);

      if (t!=null) {
          System.out.println("Posicio␣0␣de␣t␣" + t[0]);
          System.out.println("Posicio␣1␣de␣t␣" + t[1]);
      }
  }
}
```

Paràmetres de mètodes de tipus array: Sortida II I

- Extreure un subconjunt d'elements d'un conjunt d'enters en una taula.

ConjuntEnters3.java

```
public class ConjuntEnters3 {
    private int[] taula;
    private int numElems;

    public void iniTaula (int [] t, int n) {
        taula = new int[n];
        for (int i=0; i<n; i++) {
            taula[i] = t[i];
        }
        numElems = n;
    }
    public void subConjunt(int idx1, int idx2, int[] t) {
        if (idx2>idx1 && idx1>=0 && idx1<=numElems && idx2>=0 && idx1<=numElems) {
            for (int i=0; i<idx2-idx1+1; i++) {
                t[i] = taula[idx1+i];
            }
        }
    }
}
```

Paràmetres de mètodes de tipus array: Sortida II II

TaulesSortidaII

```
public class TaulesSortidaII {
  public static void main (String [] args) {
      int[] t = {1,5,8,12};
      int[] t2;
      ConjuntEnters3 cj;
      cj = new ConjuntEnters3();
      cj.iniTaula(t, 4);

      t2 = new int[2];
      cj.subConjunt(2,3, t2);

      System.out.println("Posicio␣0␣de␣t2␣" + t2[0]);
      System.out.println("Posicio␣1␣de␣t2␣" + t2[1]);
  }
}
```

Paràmetres de mètodes de tipus tupla: Entrada/Sortida I

- Sigui una taula d'enters amb valors entre el 0 i el 9, es vol tenir en una taula les freqüències d'aparició de cada valor. Aquesta taula està inicialitzada a 0, des del programa principal.

ConjuntEnters4.java

```
public class ConjuntEnters4 {
    private int[] taula;
    private int numElems;

    public void iniTaula (int [] t, int n) {
        taula = new int[n];
        for (int i=0; i<n; i++) {
            taula[i] = t[i];
        }
        numElems = n;
    }
    public void frequencies(int[] t) {
        for (int i=0; i<numElems; i++) {
            t[taula[i]]++;
        }
    }
}
```

Paràmetres de mètodes de tipus tupla: Entrada/Sortida II

TaulesEntradaSortida.java

```
public class TaulesEntradaSortida {
  public static void main (String [] args) {
      int[] t = {1,5,8,9,1,2,5,5,5,9,9};
      int[] taulaFreq;

      ConjuntEnters4 cj;
      cj = new ConjuntEnters4();
      cj.iniTaula(t, 11);

      taulaFreq = new int[10];
      for (int i=0; i<10; i++) {
          taulaFreq[i] = 0;
      }

      cj.frequencies(taulaFreq);

      for (int i=0; i<10; i++) {
          System.out.println("Posicio␣"+i+"␣de␣taulaFreq␣" + taulaFreq[i]);
      }
  }
}
```

Introducció
Definició de classes i objectes
Encapsulació: gets i sets
Encapsulació: constructors
Mètodes de classe i mètodes d'objecte
Exemples de classes ja existents
Pas de paràmetres: tipus valor, objectes immutables i objectes mutables

Encapsulació

Encapsulació: no deixarem manipular les dades internes d'un objecte directament, sinó que es farà a través dels **mètodes d'objecte** que ofereixi l'objecte. D'aquesta forma aïllem l'estructura interna de l'objecte del codi que l'utilitza, de forma que es pot canviar independentment sempre que es conservi la **interfície**.

- **Classe**:

```
class <id_classe> {
  private <id_tip> <id_atr>;
  <...>
  public void set<id_atr>(<id_tip> <id_par>) {<...>}
  public <id_tip> get<id_atr>() {<...>}
  <...>

}
```

- **Assignació**: `<id_var>.set<id_atribut>(<valor>)`
- **Consulta**: `<id_var>.get<id_atribut>()` equival a una expressió

Proteccions amb atributs privats I

- S'utilitzen els sets i els gets per impedir accesos incorrectes als atributs de les classes.

Jugador3.java

```
public class Jugador3 {
  private int    any;

  public void  setAny(int n) {
    if (n > 1900 && n < 1990) {
      any = n;
    }
  }
  public int    getAny() { return any; }
}
```

Proteccions amb atributs privats II

TuplaPrivatsProteccio.java

```
import java.util.Scanner;

public class TuplaPrivatsProteccio {
  public static void main (String [] args) {
    Jugador3 tupla = new Jugador3();

    tupla.setAny(1980);
    System.out.println("Any=" + tupla.getAny());

    tupla.setAny(2001);
    System.out.println("Any=" + tupla.getAny());
  }
}
```

Introducció
Definició de classes i objectes
Encapsulació: gets i sets
Encapsulació: constructors
Mètodes de classe i mètodes d'objecte
Exemples de classes ja existents
Pas de paràmetres: tipus valor, objectes immutables i objectes mutables

Constructors

- **Constructor**: mètode dedicat construir un objecte d'una classe determinada. L'utilitzem per incloure les inicialitzacions de l'objecte, que poden dependre de paràmetres subministrats en el moment d'efectuar la crida a través de `new`.
  - **Classe**:

```
class <id_classe> {
  <...>
  public <id_classe>(<id_tip> <id_atr> <...>) {<...>}
  <...>

}
```

  - **Construcció**: `<id_var> = new <id_classe>(<valors>);`
- **Sobre-càrrega**: podem tenir més d'un mètode amb el mateix nom però amb una llista diferent de paràmetres formals. En el nostre cas, ho utilitzarem per a tenir un constructor que no necessita cap informació, i un altre que rep la informació per inicialitzar tots els atributs.

Constructors I

```
class Jugador {
  private int    any;
  public Jugador()       { any = 1985; }
  public Jugador(int n) { setAny(n);  }
  public void  setAny(int n) {
    if (n > 1900 && n < 1990) {
      any = n;
    }
  }
  public int    getAny() { return any; }
}

public class TuplaPrivatsConstructor {
  public static void main (String [] args) {
    Jugador juvenil = new Jugador();
    Jugador senior  = new Jugador(1975);

    System.out.println("juvenil=" + juvenil.getAny());
    System.out.println("senior␣=" +  senior.getAny());
  }
}
```

Mètodes de classe i mètodes d'objecte I

- **Mètode de classe**: és el mateix per a tota una classe d'objectes i no depèn de les dades concretes de cap instància [no cal que existeixi cap objecte per a poder-lo aplicar].
- **Mètode d'objecte**: pot ser el mateix per a tota una classe d'objectes i depèn de les dades concretes d'una instància [cal que existeixi un objecte al qual poder-lo aplicar].
- Les *aplicacions en Java* són la declaració d'una classe que conté un mètode de classe anomenat `main`, el qual s'executa com a punt d'entrada.

Exemple de classe ja existent: String I

- La classe String disposa de mètodes de classe per a obtenir l'equivalent alfanumèric de diversos tipus de dades.
- Els objectes String disposen de mètodes d'objecte per a treballar amb el valor del seu contingut.

```
public class ClasseString {
  public static void main (String [] args) {
    String s1 = String.valueOf(1);
    String s2 = String.valueOf(2);
    System.out.println("s1+s2="           + s1 + s2);
    String s3 = new String("Hola");
    String s4 = "Hola";
    System.out.println("s3.length="       + s3.length());
    System.out.println("s3.majúscules="   + s3.toUpperCase());
    System.out.println("s3.endsWith(a)="  + s3.endsWith("a"));
    System.out.println("s3.startsWith(a)="+ s3.startsWith("a"));
    System.out.println("substring(1,2)="  + s3.substring(1,2));
    System.out.println("s3␣equals␣s4="    + s3.equals(s4));
    System.out.println("s3␣==␣s4="        + (s3 == s4));
    System.out.println("Caràcter␣a␣la␣posició␣0:␣"     + s3.charAt(0));
    System.out.println("minuscules␣"     + s3.toLowerCase());
  }
}
```

Exemple de classe ja existent: Character I

- Existeixen unes classes especials per als tipus valor (Integer, Float, Character, Boolean), per a poder proporcionar mètodes de classe bàsics per a operar amb ells.
- Són classes immutables: un cop s'han inicialitzat, no poden canviar el seu valor
- En qualsevol moment és possible convertir un dels tipus valor al seu equivalent en objecte (tipus referència).

```
public class ClasseCharacter {
  public static void main (String [] args) {
    // Mètodes de classe
    int codiUNICODE = Character.getNumericValue('A');
    System.out.println("Unicode(A)="+codiUNICODE);

    boolean digit    = Character.isDigit('A');
    System.out.println("isDigit(A)="+digit);

    char minuscula = Character.toLowerCase('A');
    System.out.println("Minuscula(A)="+minuscula);
```

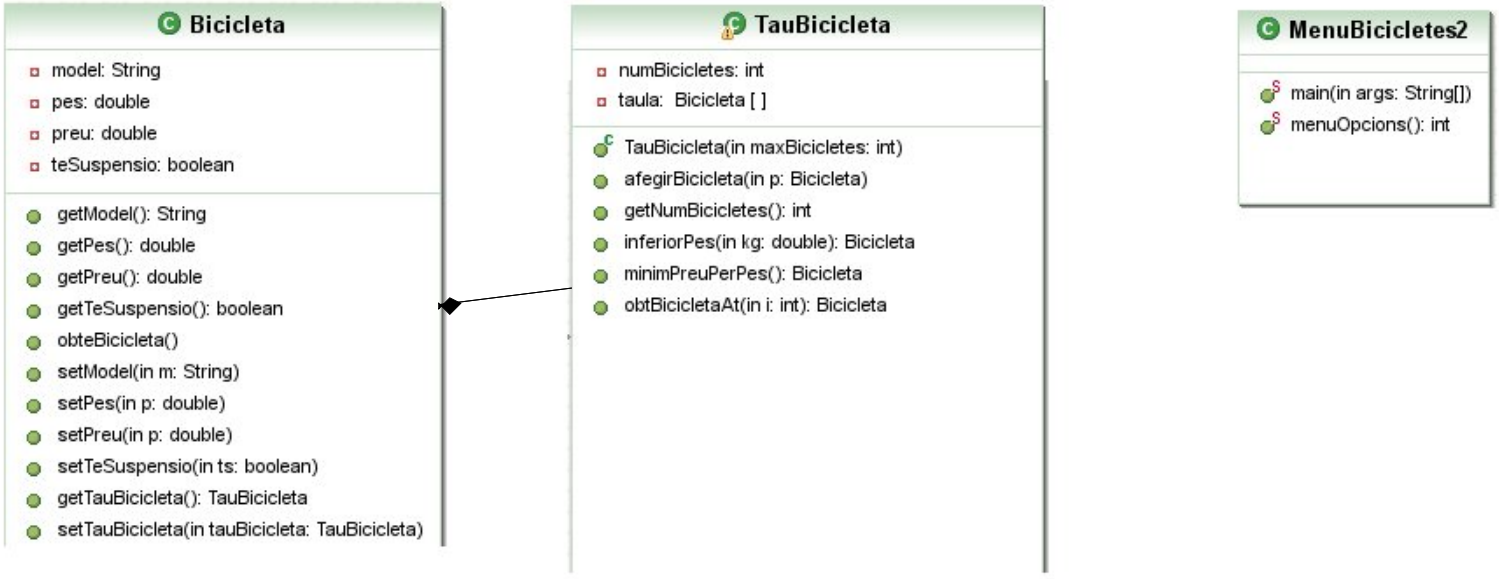
Exemple de classe ja existent: Character II

```
    // Mètodes d'objecte
    Character x = new Character('B');
    Character y = new Character('C');

    boolean equalToC = x.equals(y);
    System.out.println("equals(C)="+equalToC);

    String s = x.toString();
    System.out.println("toString="+s);

    Class c = x.getClass();
    System.out.println("classe="+c.getName());
  }
}
```

Problema: Gestió de bicicletes I

- Es vol gestionar la informació de com a molt 25 bicicletes. Cada bicicleta té un conjunt de dades: model, pes, si té suspensió i preu. Es vol programar una solució que permeti:
  1. Afegir bicicletes
  2. Obtenir el model de la bicicleta que tingui mínim cost per kilogram.
  3. Obtenir la bicicleta de pes inferior a un pes entrat per l'usuari, si és que n'hi ha alguna.
  4. Sortir
- Es proposa la solució usant atributs privats i mètodes dins de les classes (implementades amb fitxers diferents).

Problema Bicicletes: Solució I



1. Identificació de la seqüència principal: Seqüència d'opcions
   entrada per teclat:
   `Primer element:` $opcio = menuOpcions()$
   `Següent element:` $opcio = menuOpcions()$
   `FinalSeq()=` $(opcio == 4)$

Problema Bicicletes: Solució II

2. Identificació de l'esquema: Recorregut.

```
import java.util.Scanner;
public class MenuBicicletes2 {
    public static int menuOpcions(Scanner sc) {
        System.out.println("\n(1) Inserir/afegir una bicicleta");
        System.out.println("(2) Bicicleta de cost preu per kg minim");
        System.out.println("(3) Cerca de la bicicleta inferior a un cert pes");
        System.out.println("(4) Sortir");
        System.out.println("Escull una de les opcions:");
        return sc.nextInt();
    }

    public static void main(String [] args) {
        Bicicleta     bici;
        TauBicicleta  tauBicis;
        int opcio;
        Scanner sc;
        sc = new Scanner(System.in);

        tauBicis = new TauBicicleta(25);
        opcio = menuOpcions(sc);
        while (opcio!=4) {
            // tractar element
            switch (opcio) {
            case 1: // Afegir bicicleta
                bici = new Bicicleta();
                bici.obteBicicleta(sc);
                tauBicis.afegirBicicleta(bici);
                break;
            case 2: // minim_preu_per_pes
```

Problema Bicicletes: Solució III

```
                bici = tauBicis.minimPreuPerPes();
                System.out.println("Bicicleta de minim preu per kg "
                                   + bici);

                break;

            case 3: // N'hi ha alguna de pes inferior a 1kg?
                System.out.println("Llindar de pes?");
                bici = tauBicis.inferiorPes(sc.nextInt());
                if ( bici != null ) {
                    System.out.println("Bicicleta trobada\n " + bici);
                } else {
                    System.out.println("Bicicleta no trobada ");
                }
                break;
            default:
                System.out.println("opcio erronia");
                break;
            }

            // obtencio seguent element
            opcio = menuOpcions(sc);
        }
        System.out.println("Programa acabat correctament!");
    }
}
```

Problema Bicicletes: Solució I

- Classe TauBicicleta: dins del fitxer TauBicicleta.java

```
// Tupla que guardarà un conjunt de bicicletes
public class TauBicicleta {
    private int       numBicicletes;
    private Bicicleta[] taula;

    public TauBicicleta(int maxBicicletes) {
        taula = new Bicicleta[maxBicicletes];
        numBicicletes = 0;
    }
    public void afegirBicicleta(Bicicleta p) {
        if (numBicicletes < taula.length) {
            taula[numBicicletes] = p;
            numBicicletes = numBicicletes + 1;
        } else {
            System.out.println ("Estas excedint el nombre màxim de bic
        }
    }
    public int getNumBicicletes() {
        return numBicicletes;
    }
    public Bicicleta obtBicicletaAt(int i) {
```

Problema Bicicletes: Solució II

```
        return taula[i];
    }
    public Bicicleta minimPreuPerPes() {
        int iminim;
        int i;
        double minim;
        double aux;
        i = 0;
        minim = Integer.MAX_VALUE;
        iminim = 0;
        while ( i < numBicicletes ) {
            aux = taula[i].getPreu() / taula[i].getPes();
            if ( minim > aux) {
                minim = aux;
                iminim = i;
            }
            i = i + 1;
        }
        return (taula[iminim]);
    }
    public Bicicleta inferiorPes(double kg) {
        boolean   trobat;
        int       i;
```

Problema Bicicletes: Solució III

```
        Bicicleta b;

        trobat = false;
        i = 0;
        while ( i < numBicicletes && !trobat) {
            if (taula[i].getPes() < kg) {
                trobat = true;
            } else {
                i = i + 1;
            }
        }
        if (trobat) {
            b = taula[i];
        } else {
            b = null;
        }
        return(b);
    }
}
```

Problema Bicicletes: Solució I

- Classe Bicicleta: dins del fitxer Bicicleta.java

```
import java.util.Scanner;
public class Bicicleta {
    private String  model;
    private double  pes;
    private boolean teSuspensio;
    private double  preu;

    public void setModel(String m) {
        model = m;
    }
    public String getModel() {
        return (model);
    }

    public void setPes(double p) {
        pes = p;
    }
    public double getPes() {
        return (pes);
    }

    public void setTeSuspensio(boolean ts) {
        teSuspensio = ts;
    }
    public boolean getTeSuspensio() {
        return (teSuspensio);
    }
```

Problema Bicicletes: Solució II

```
    public void setPreu(double p) {
        preu = p;
    }
    public double getPreu() {
        return (preu);
    }
    public void obteBicicleta(Scanner sc) {
        System.out.println("Model?");;
        model = sc.next();
        System.out.println("Pes?");;
        pes = sc.nextDouble();
        System.out.println("Te suspensió (s/*)?");
        teSuspensio = sc.next().equals("s");
        System.out.println("Preu?");;
        preu = sc.nextDouble();
    }
    public String toString() {
        return ("Bicicleta: Model: "+model+"\n\t Pes: "+pes
                +"\n\t Suspensio:"+teSuspensio+"\n\t Preu: " +preu);
    }
}
```

Introducció
Definició de classes i objectes
Encapsulació: gets i sets
Encapsulació: constructors
Mètodes de classe i mètodes d'objecte
Exemples de classes ja existents
Pas de paràmetres: tipus valor, objectes immutables i objectes mutables

Pas de paràmetres: tipus valor, objectes immutables i objectes mutables

- d'**entrada**: són aquells que el mètode que és cridat UTILITZA però NO modifica:
  - de **tipus valor**: es passa al mètode una *còpia de les dades*.
  - **objectes immutables**: es passa al mètode una *referència a unes dades* que no es poden modificar. (els objectes immutables són aquells als quals no se'ls poden modificar les dades un cop s'han inicialitzat; exemple: `String`, `Character`, etc.).

[tots els exemples vistos fins ara són d'aquest tipus]

- de **sortida**: són paràmetres que el mètode cridat MODIFICA.

En llenguatge Java es proporciona una referència a un objecte mutable buit de contingut inicialment a la crida del mètode.

Pas de paràmetres II

- d'**entrada/sortida**: són aquells paràmetres que tenen un valor que és consultat dins del mètode i alhora sobre ells s'expressa un resultat.
  - **objectes mutables**: es passa al mètode una *referència a unes dades que es poden modificar*. (els objectes mutables són aquells als quals se'ls poden modificar les dades; exemple: `Jugador`, `Triangle`, `StringBuffer`).

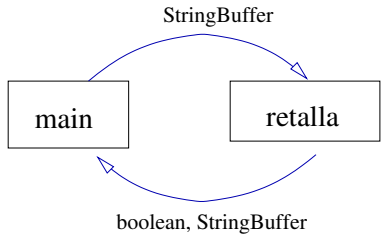
Pas de paràmetres d'entrada I

Què passaria en aquest cas?

```
public class MetodeEstaticParamEntSortImmutable {
  public static void main (String [] args) {
    String nom = new String("Maria␣del␣Mar");

    System.out.println("Nom:␣"+nom);

    if (retalla(nom)) {
      System.out.println("Era␣massa␣llarg:␣"+nom);
    }
  }
  public static boolean retalla(String cad) {
    boolean retallat;
    if (cad.length() > 5) {
      cad = cad.substring(0,5);
      retallat = true;
    } else {
      retallat = false;
    }
    return retallat;
  }
}
```

Pas de paràmetres d'entrada-sortida I

- Exemple de pas de paràmetres d'entrada-sortida:



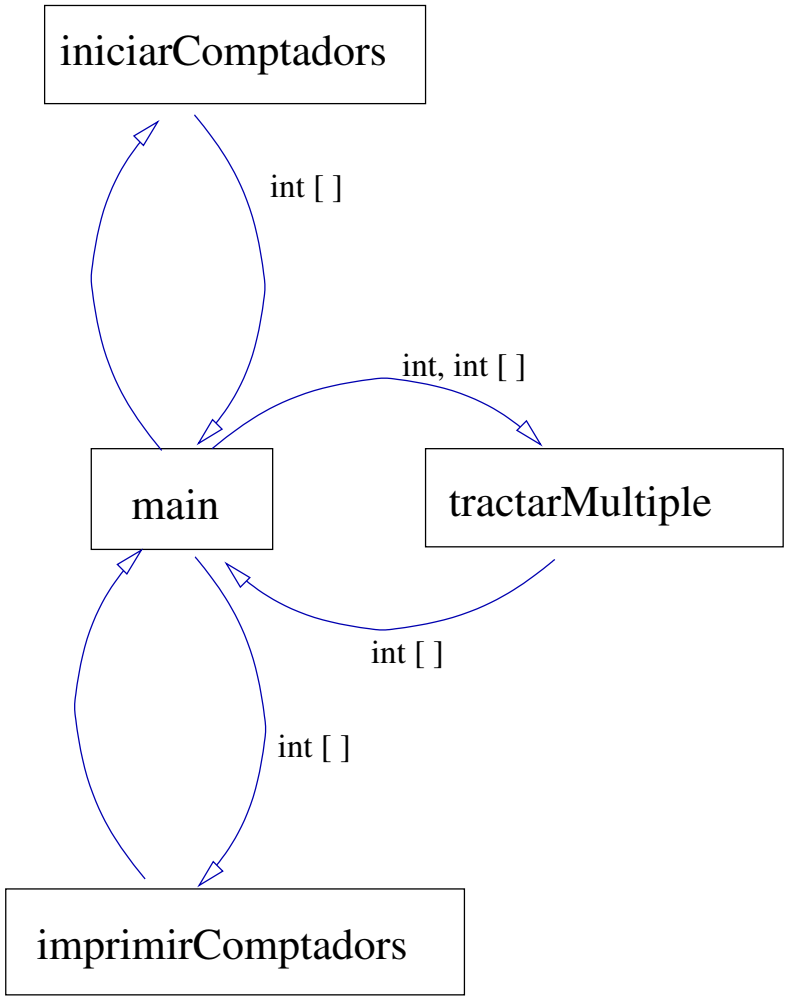

```
public class MetodeEstaticParamEntSort {
  public static void main (String [] args) {
    StringBuffer nom = new StringBuffer("Maria del Mar");

    System.out.println("Nom: "+nom);

    if (retalla(nom)) {
      System.out.println("Era massa llarg: "+nom);
    }
  }
  public static boolean retalla(StringBuffer cad) {
    boolean retallat;
    if (cad.length() > 5) {
      cad.delete(5, cad.length());
      retallat = true;
    } else {
      retallat = false;
    }
    return retallat;
  }
}
```

Exemple:Mètodes estàtics o de classe I

Dígits continguts en els múltiples de 7: donats els múltiples de 7 inferiors a 10000, donar quants dígits 0's, quants dígits 1's, . . . , quants dígits 9 contenen.

- Identificació de la seqüència:
  - Primer element: m= 7
  - Següent element: m = m + 7
  - Final de seqüència: $m > 10000$
- Identificació de l'esquema: Recorregut

Exemple:Mètodes estàtics o de classe II



Exemple:Mètodes estàtics I

```
public class ComptarDigits {
  public static int TAMANY = 10;
    public static void main (String [] args) {
        int [] comptador;
        int m;
        comptador = iniciarComptadors();
        m = 7;
        while (m<10000) {
            tractarMultiple(m, comptador);
            m = m + 7;
        }
        imprimirComptadors(comptador);
    }

    public static int[] iniciarComptadors() {
        int [] taula = new int[TAMANY];

        /* Identificacio de la sequencia:
           1er element: i = 0
           Seguent element:  i  = i + 1
           Final de sequencia: i >= 10
           Identificacio de l'esquema: Recorregut
        */
```

Exemple:Mètodes estàtics II

```
        for (int idx=0; idx < TAMANY ; idx++) {
            taula[idx] = 0;
        }
        return taula;
    }
    public static void tractarMultiple( int m, int [] comptador) {
        int d;

        /* Identificacio de la sequencia:
           1er element: d = m % 0
           Seguent element: m = m / 10; d = m % 10;
           Final de sequencia: m == 0
           Identificacio de l'esquema: Recorregut
        */
        System.out.println("Multiple␣" + m);
        d = m%10;
        while (m!=0) {
            System.out.println("␣␣␣␣␣␣digit:␣"+d);
            comptador[d] = comptador[d]+1;
            m = m/10;
            d = m%10;
        }
    }
```

Exemple:Mètodes estàtics III

```
    public static void imprimirComptadors(int[] t) {
        /* Identificacio de la sequencia:
           1er element: i = 0
           Seguent element:  i  = i + 1
           Final de sequencia: i >= 10
           Identificacio de l'esquema: Recorregut
        */
        for (int i=0; i<10; i=i+1) {
            System.out.println ("comptador␣"+i+":␣"+t[i]);
        }
    }
}
```

Disseny de tupla I

- **Descripció**: Es desitja dissenyar i implementar una estructura de dades que permeti guardar un conjunt de com a molt 30 enters que permeti inserir un enter en qualsevol posició i esborrar la primera aparició d'un enter en la taula.
- **Estratègia**: Es definirà una tupla que conté una taula i el nombre d'elements i els mètodes **setIntAt(idx, nouEnter)** i **deleteValue(enter)**
- **setIntAt(idx, nouEnter)**:
  - Identificació seqüència: Primer Element: $i = nElems - 1$; Següent: $i = i - 1$; Fi de seq: $i < idx$
  - Identificació de l'esquema: Recorregut
- **deleteValue(enter)**:
  - Primer es cerca el valor i es troba l'índex de la seva primera aparició,
  - després es recorre des de l'índex fins al final esborrant els elements.

Exemple de tupla: insercions i esborrats a qualsevol posició I

```
public class TaulaGen {
    private int[] taula;
    private int nElems;

    public TaulaGen(int max) {
        taula = new int[max];
        nElems = 0;
    }

    public int obtIntAt(int idx) {
        return (taula[idx]);
    }

    public int getNumElems() {
        return nElems;
    }

    public void setIntAt (int idx, int nouEnter){
        if (nElems < taula.length) {

            for (int i= nElems-1; i>= idx; i = i-1) {
                taula[i+1] = taula[i];
            }
```

Introducció
Definició de classes i objectes
Encapsulació: gets i sets
Encapsulació: constructors
Mètodes de classe i mètodes d'objecte
Exemples de classes ja existents
Pas de paràmetres: tipus valor, objectes immutables i objectes mutables

Exemple de tupla: insercions i esborrats a qualsevol posició II

```
            taula[idx] = nouEnter;
            nElems = nElems + 1;
        }
    }

    public void deleteValue (int x) {
        int idx;
        boolean trobat;

        // cerca del primer enter igual al paràmetre

        trobat = false;
        idx = 0;

        while (idx<nElems && !trobat) {
            trobat = (taula[idx]==x);
            idx = idx + 1;
        }
        if (trobat) {
            // S'esborra
            for (int i = idx; i<nElems; i=i+1) {
```

Introducció
Definició de classes i objectes
Encapsulació: gets i sets
Encapsulació: constructors
Mètodes de classe i mètodes d'objecte
Exemples de classes ja existents
Pas de paràmetres: tipus valor, objectes immutables i objectes mutables

Exemple de tupla: insercions i esborrats a qualsevol posició III

```
                taula[i-1] = taula[i];
            }
            nElems = nElems -1;
        }
    }

    public void imprimirTaula() {
        for (int i=0; i<nElems; i=i+1) {
            System.out.println("␣" + taula[i]);
        }
    }
    public String toString() {
        String s="";
        for (int i=0; i<nElems; i=i+1) {
            s = s + "␣"+ taula[i];
        }
        return s;
    }
}
```

Introducció
Definició de classes i objectes
Encapsulació: gets i sets
Encapsulació: constructors
Mètodes de classe i mètodes d'objecte
Exemples de classes ja existents
Pas de paràmetres: tipus valor, objectes immutables i objectes mutables

Exemple d'ús de la tupla: insercions i esborrats a qualsevol posició I

```
class ProvaTaula {

  public static void main(String[] args) {
   TaulaGen t = new TaulaGen(30);

   for (int i=0; i < 10; i=i+1) {
       t.setIntAt(i, i);
   }
   t.imprimirTaula();
   t.setIntAt(2, 5);
   t.imprimirTaula();
   t.deleteValue(3);
   t.imprimirTaula();
   System.out.println(t);
  }
}
```