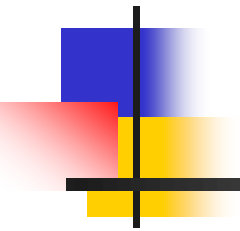


Programació Avançada

Aclariments Pràctica 2

Aprendre: C++

1.- Llegir els 2 dossiers de C++ penjats a l'aula virtual.

Dossier 1 → Programació imperativa

Dossier 2 → Programació Orientada a Objectes i Variable dinàmica en C++

Excepcions en C++

2.- Diferències destacables:

→ Parametrització (Pràctica 1)

→ **Gestió de la variable dinàmica** (Pràctica 2)

→ Sobrecarrega d'operadors (Pràctica 1)

→ **Herència múltiple** (Pràctica 2)

→ **Polimorfisme** (Pràctica 2)

→ **Destructor** (Pràctica 2)

Pràctica 2

Objectius:

→**1.- Seqüències enllaçades.**

→**2.- POO: Herència, polimorfisme, redefin**
sobrecarrega de mètodes, constructo

→**3.- Excepcions.**

Lliurament:

Amb el format indicat: portada, índex, ...
Participació de cada membre del grup

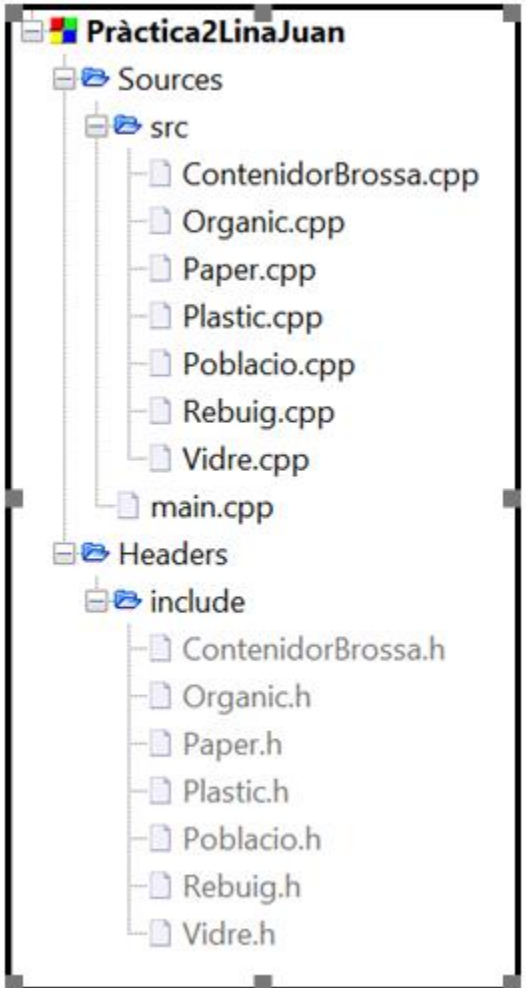
→ 1.- Llistat imprès dels fonts (**Inici sessió 3**)

→ 2.- Penjar el projecte al Moodle **(2/Novembr**

Consideracions:

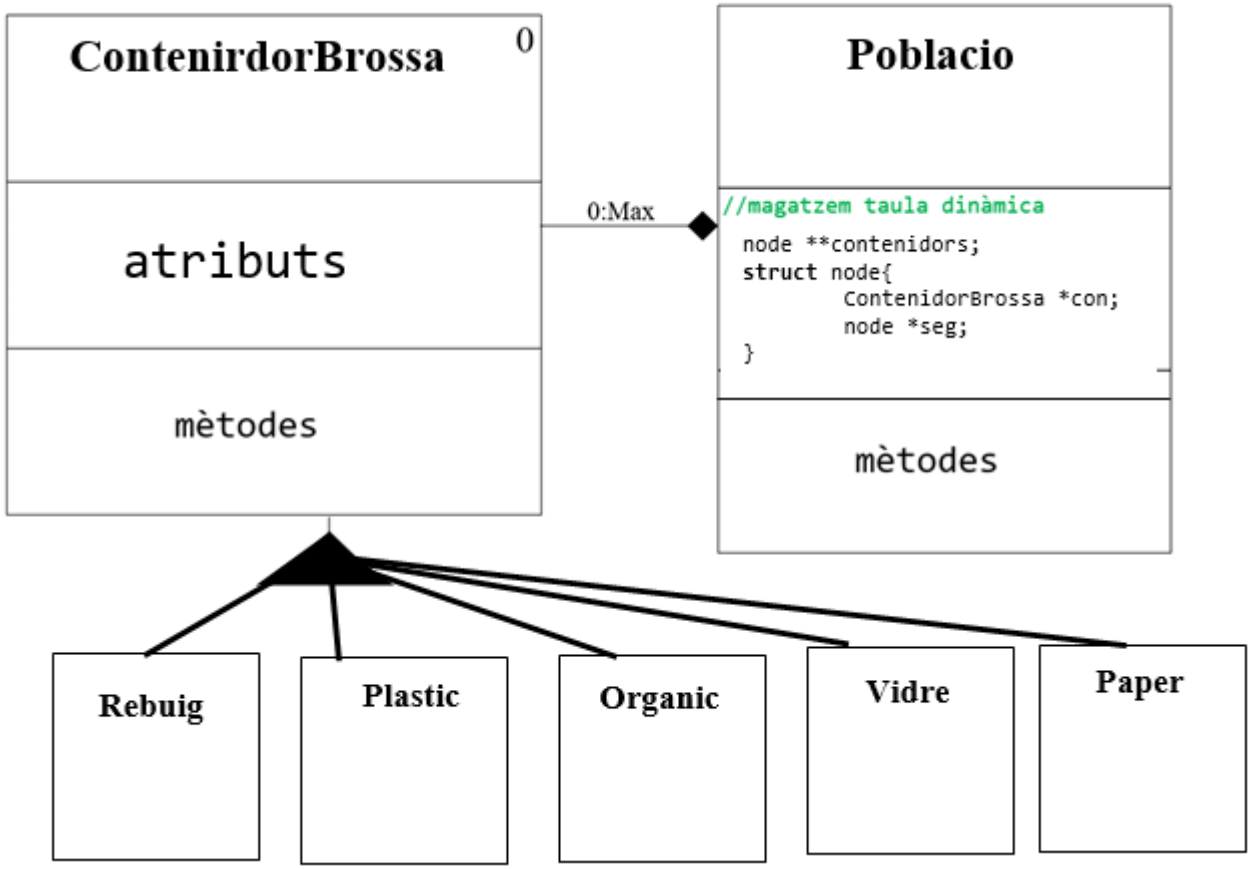
→1.- **Crear** un nou projecte. Identificar-lo:
Pràctica2Cognom1Nom1&&Cognom2Nom2

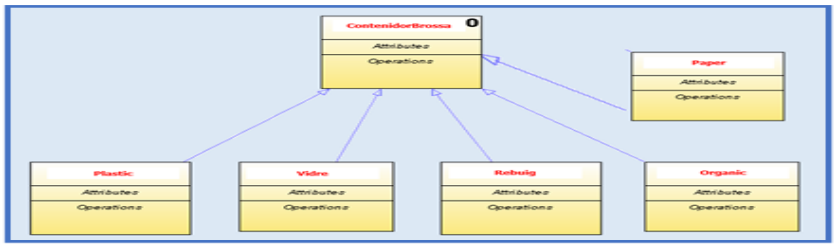
→2.- **Estructura** del projecte:



Enunciat

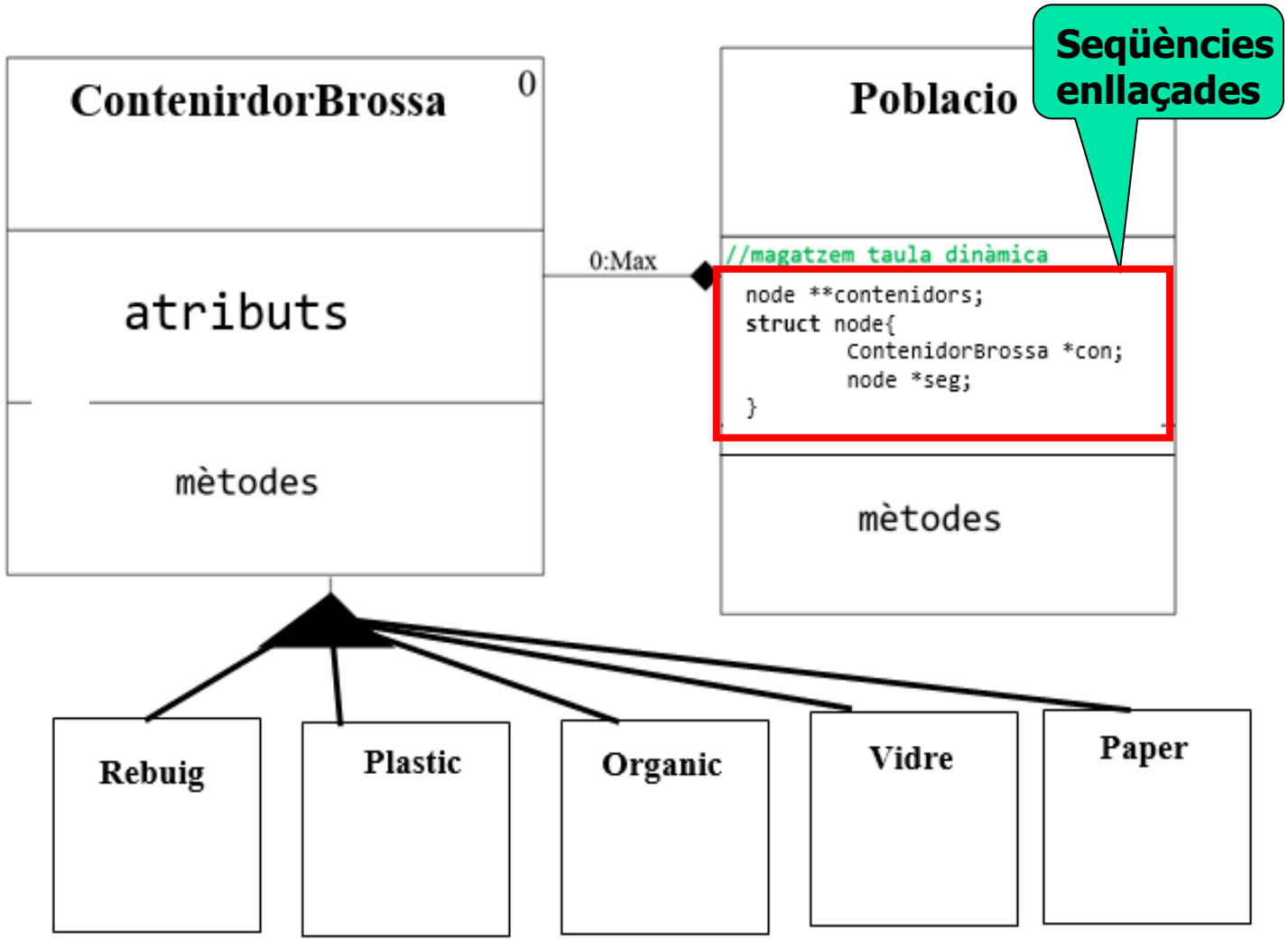
Pràctica 2



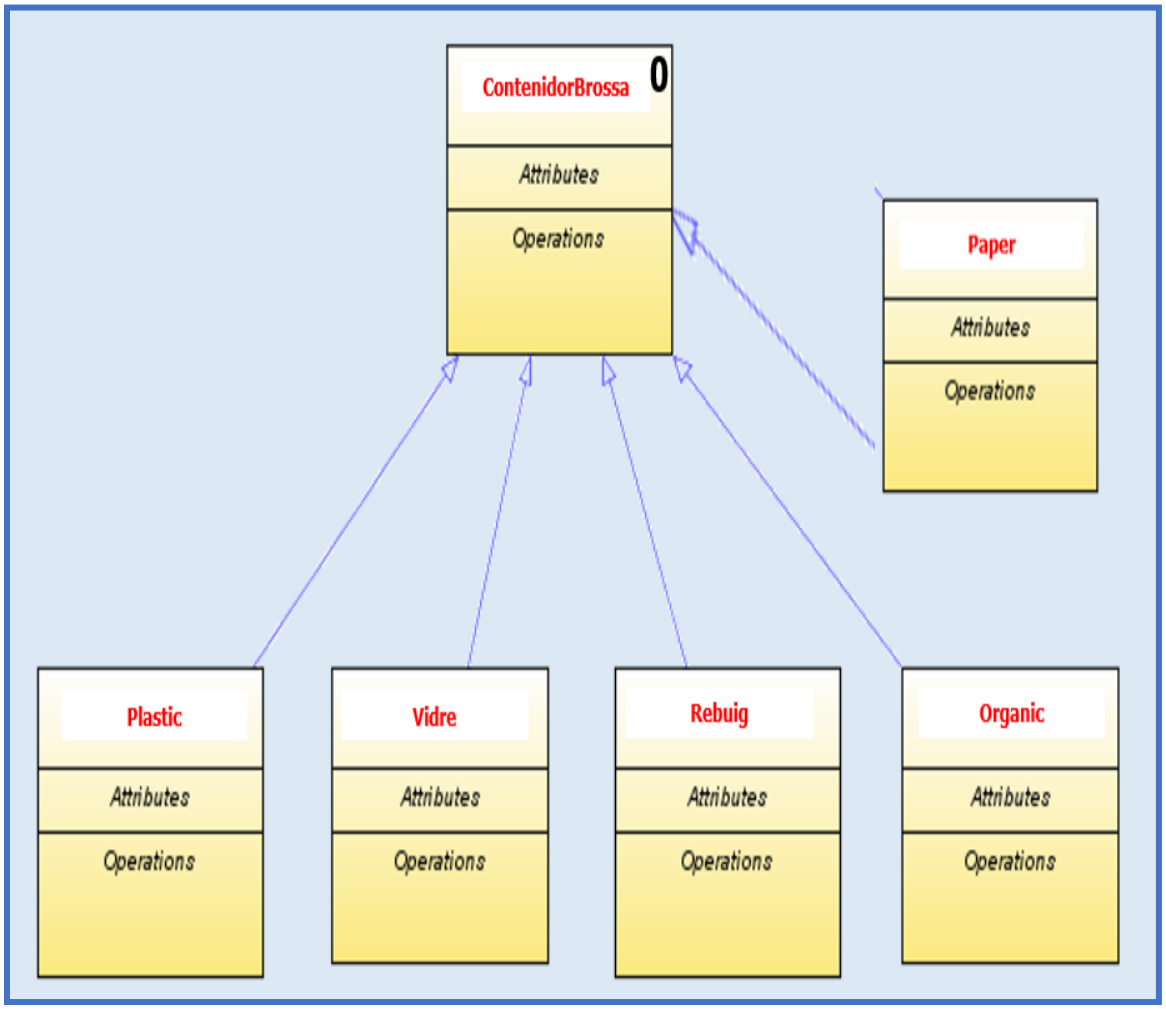


Enunciat

Pràctica 2

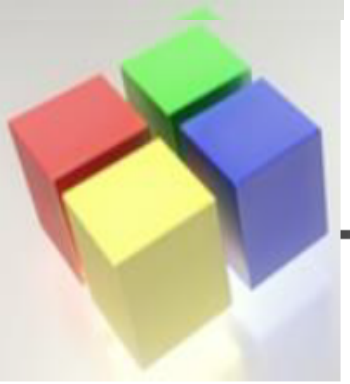


Pràctica 2. Part 1



Treballar l'herència:

Constructors
Polimorfisme
Redefinició
Sobrecarrega
Destructor
Mètodes i classe abstracta



Pràctica 2

Polimorfisme:

→En C++ **no és automàtic**.

→El programador ha d'indicar si vol o no que actui el polimorfisme.
Sense cap indicació no hi ha polimorfisme.

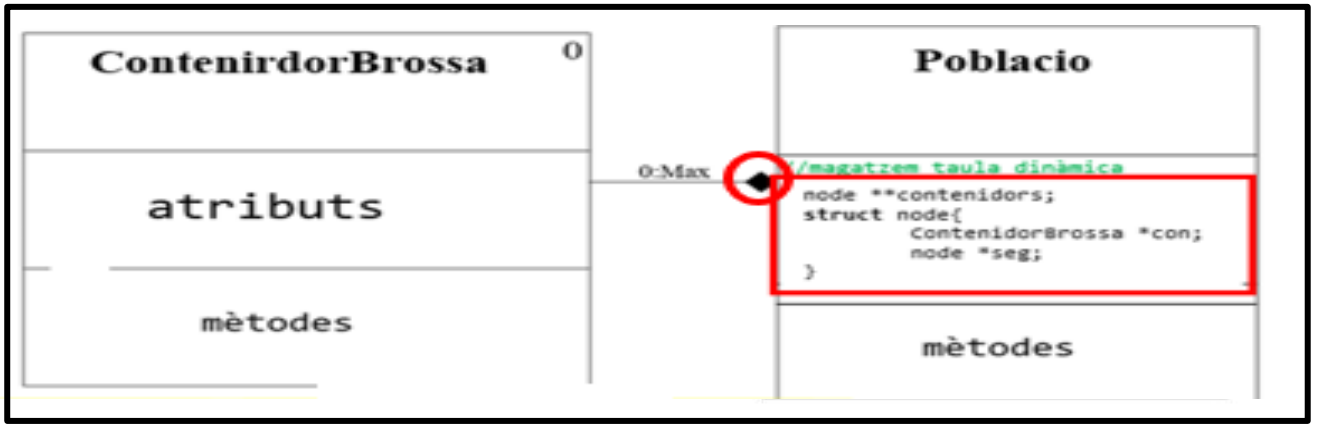
→El modificador **virtual** en un mètode indica que es vol un comportament polimòrfic. Cal indicar-lo en la definició del mètode però no en la seva implementació.

```
virtual tipus_retorn metode(paràmetres); //*.h
tipus_retorn nomClasse::metode(paràmetres){ //*.cpp
                    //sentències
}
```

→ Els mètodes abstractes a més de portar el modificador de virtual han de rebre **una assignació de zero**.

```
virtual tipus_retorn nom_mètode(paràmetres) = 0;
```

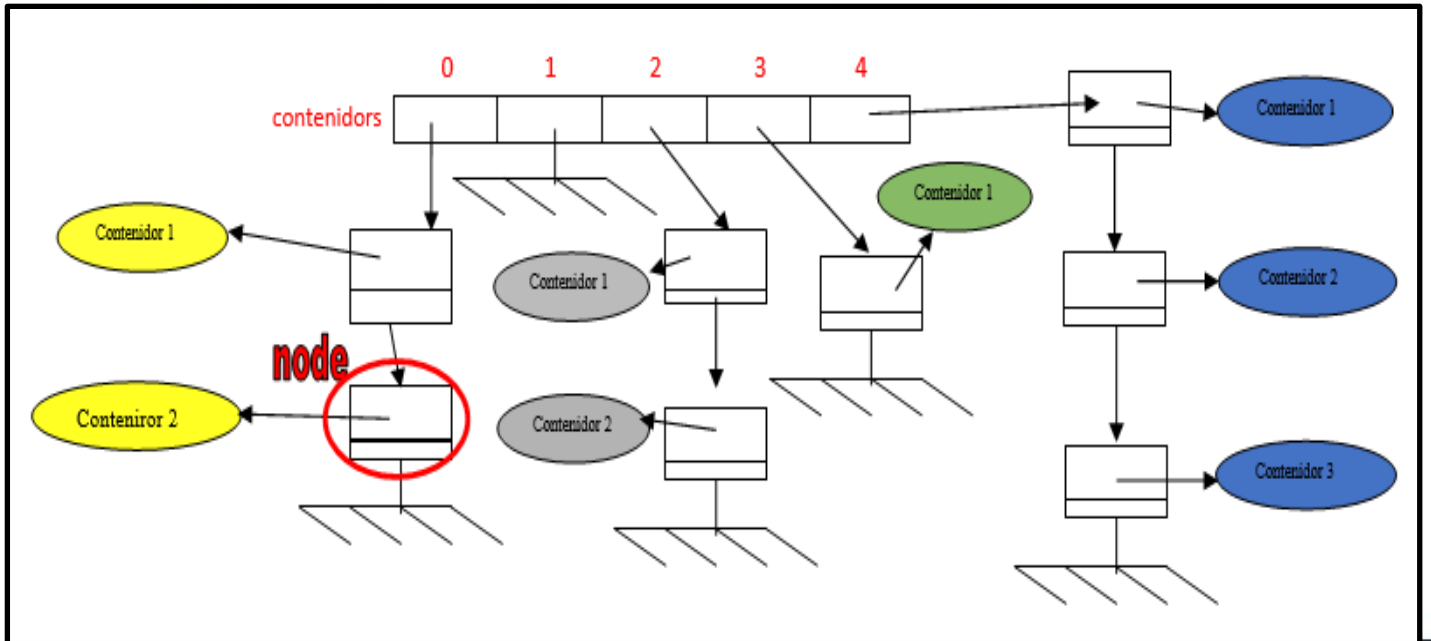
Pràctica 2. Part 2



Contenidor:

Vector dinàmic de 5 components

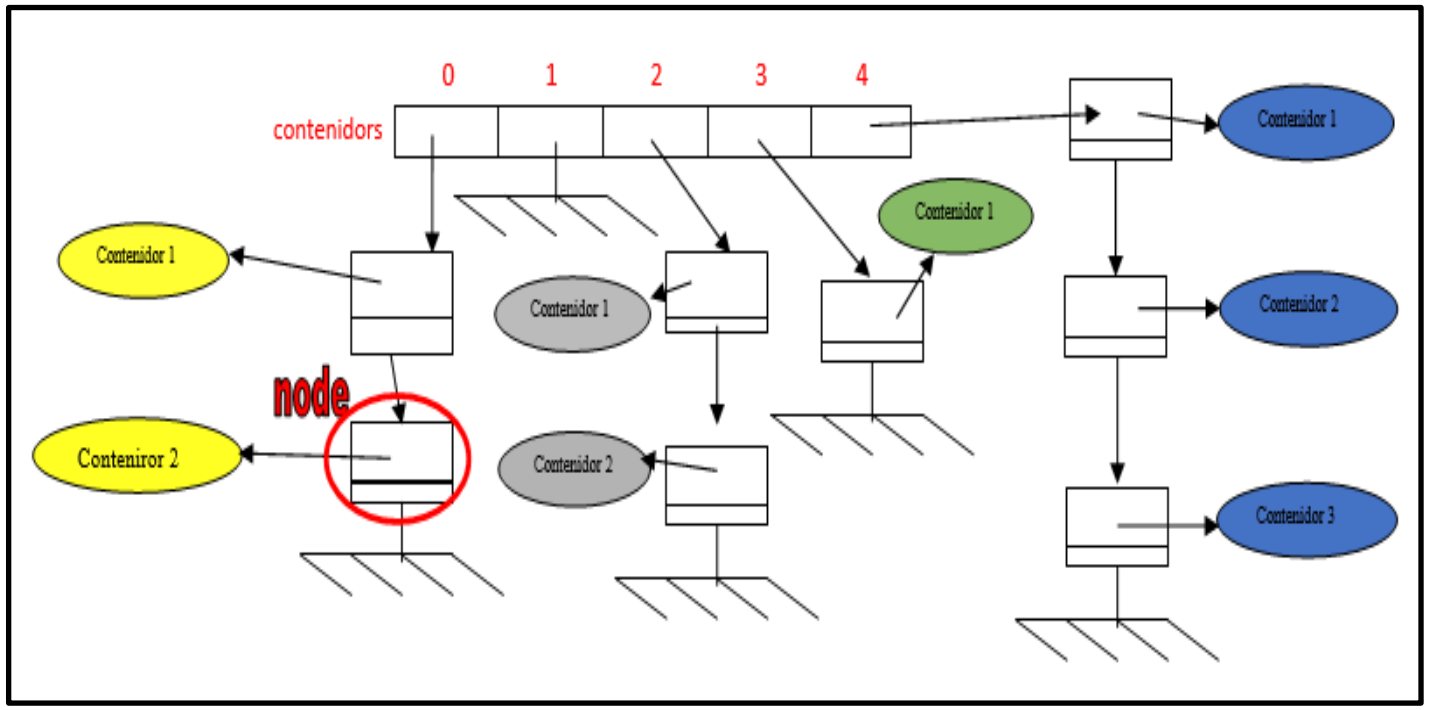
Agrupa en una seqüència enllaçada contenidors de brossa



Composició en temps execució.

Els objectes continguts seran dinàmics (necessari perquè la classe base és abstracta i està especialitzada)

Pràctica 2. Part 2



Seqüència enllaçada:

→ **sense** node capçalera

→ **lineal** (darrer node té una referència nul·la)

Node

→ tipus **struct** (com una classe però **només** amb atribu
de visibilitat pública anomenats camps)

→ visibilitat **private**(dins de la classe Poblacio)

```
struct node{
        ContenidorBrossa *con;
        node* seg;
};
```

Pràctica 2

Contenidor:

Vector dinàmic de 5 components DINÀMIC → es crea amb new
Agrupa en una seqüència enllaçada contenidors de brossa

```
tipus* variable;
variable = new tipus[dimensio];
//ús de la variable amb la sintaxi de vector variable[índex]
....
delete variable; //imprescindible
```

Exemple 1. Vector d'enters:

```
int*  taula;
taula = new int[20];
taula[0]=34;
.....
delete taula;
```

Exemple 2. Taula d'objectes dinàmics:

```
ContenidorBrossa**  taula;
taula = new ContenidorBrossa*[20];
taula[0]=new Organic(<paràmetres>);
.....
delete taula[0];
delete taula;
```

Pràctica 2

Tipus de les components de la taula

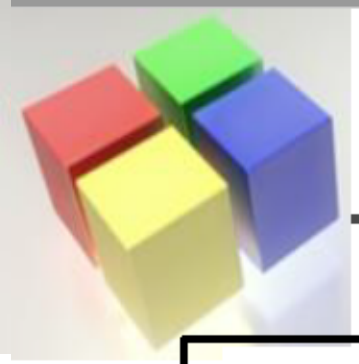
```
node** contenidor;
```

Per crear la taula dinàmicament

```
struct node{
        ContenidorBrossa *con;
        node* seg;
};
```

Un node referencia a un contenidor dinàmic. Tot objecte d'una derivada és objecte de la base, per tant aquest camp pot apuntar a qualsevol tipus de contenidor

Imprescindible que els nodes i els objectes derivats de ContenidorBrossa siguin creats dinàmicament.



Herència en C++

Observeu! Una classe pot estendre més d'una classe. Herència múltiple.

```
class nomClasse [: mode nomClass1, mode nomClass2,...]{
        // tots els membres de nomClass1, nomClass2, hi són!
        //s'afegeixen atributs i/o mètodes i/o es redefineixen mètodes
};
```

Mode d'herència:

- **private**

El que no és privat en la base s'hereta amb accés privat

- **protected**

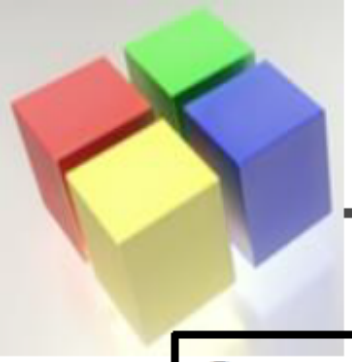
El que no és privat en la base s'hereta amb accés protected

- **public**

El que no és privat en la base s'hereta amb **la mateixa visibilitat** de la base

classe base	classe derivada
public	private
protected	private
private	**inaccessible**
public	protected
protected	protected
private	**inaccessible**
public	public
protected	protected
private	**inaccessible**

En Java sempre es fa una herència public



Excepcions en C++

Controlar les situacions anormals mitjançant l'ús de Excepcions:

1.- La filosofia de funcionament és la mateixa que Java.

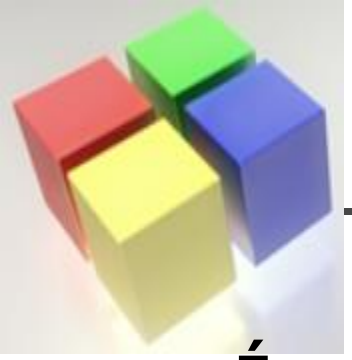
2.- Recalcar:

→ en Java les excepcions han de ser objectes de la classe Exception (també derivades), **en C++ es poden llançar excepcions de tipus primitius (int, char ....)**

→ El procediments/mètodes que llancen excepcions ho han d'indicar **obligatòriament** en la seva signatura, en Java no és sempre necessari, depèn del tipus d'excepció a llençar.

→ En C++ la paraula reservada per indicar que el mètode/procediment llança una excepció és **throw**, en Java és **throws** (prototipus procediment).

3.- Pràctica 2: llanceu excepcions de tipus primitiu



Pràctica 2. Mètode destructor

- **És un mètode especial d'una classe que serveix para eliminar un objecte d'aquesta classe.**
- És el mètode que realitza la tasca contraria al constructor.
- Tenen el mateix nom que la classe a la que pertanyen, però tenen el símbol ~ davant. No tenen tipus de retorn ni paràmetres.

  Prototipus: **~nomClasse()**
- No pot estar sobrecarregat. Ha de ser d'àmbit públic i no s'hereten.
- Es pot invocar explícitament. També ho fa automàticament quan es destrueix un objecte (quan s'executa l'operador **delete**).
- En general, serà necessari definir un destructor quan la classe tingui atributs de tipus punter, però no és una regla estricta.