Universitat de Barcelona

Tema 1 **Introducció a les estructures de dades**

Dra. Maria Salamó Llorente

Estructura de Dades

Grau d'Enginyeria Informàtica

Facultat de Matemàtiques i Informàtica,

Universitat de Barcelona

Universitat de Barcelona

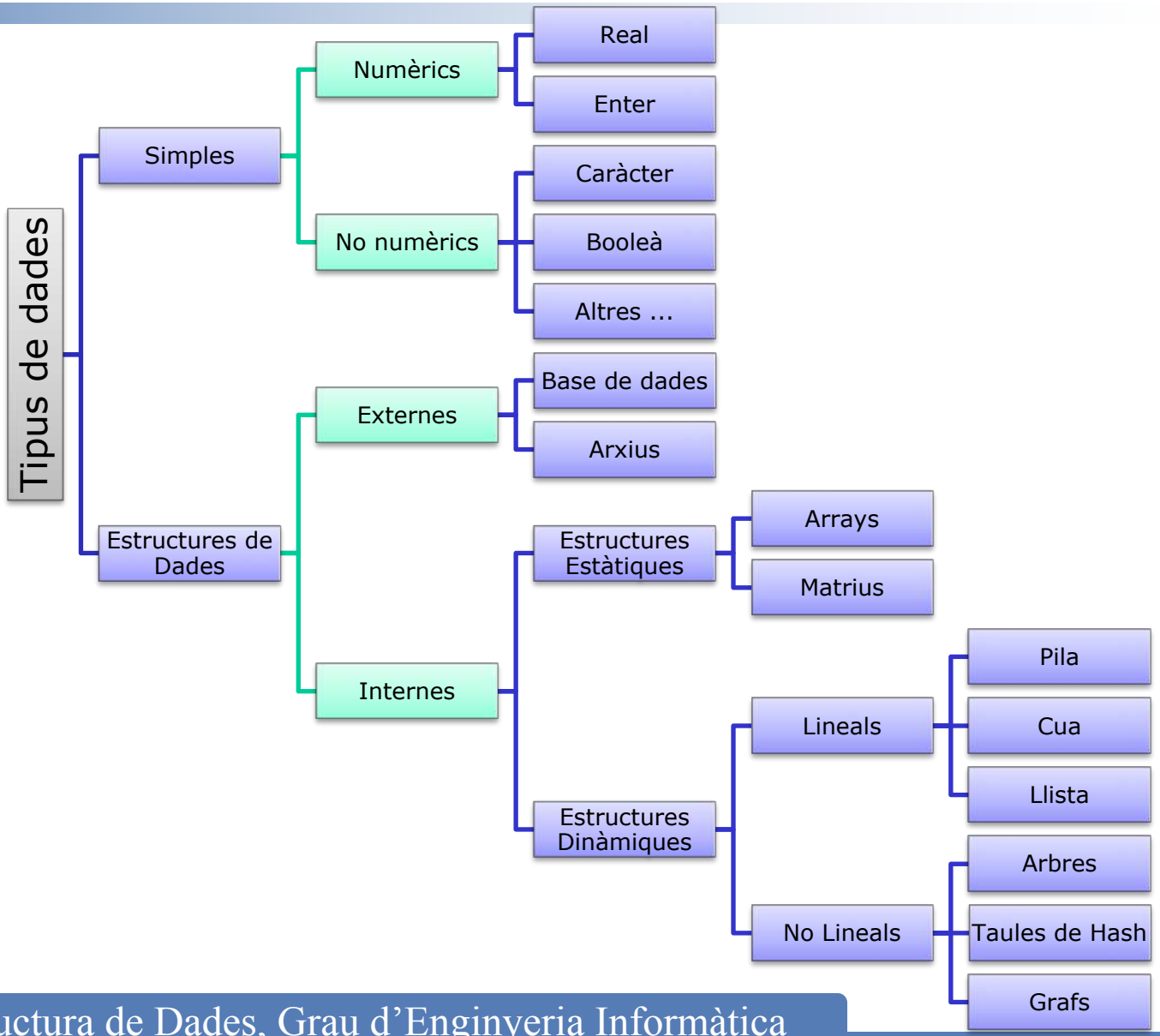
Contingut

1.1 Introducció

Definició

Una estructura de dades és una forma d'**organitzar les dades** en un computador per a que puguin ser usades de forma **eficient**

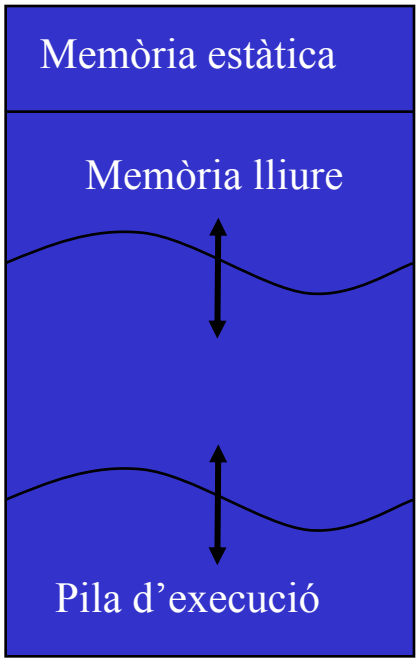
Tipus de dades



On es guarden les dades?

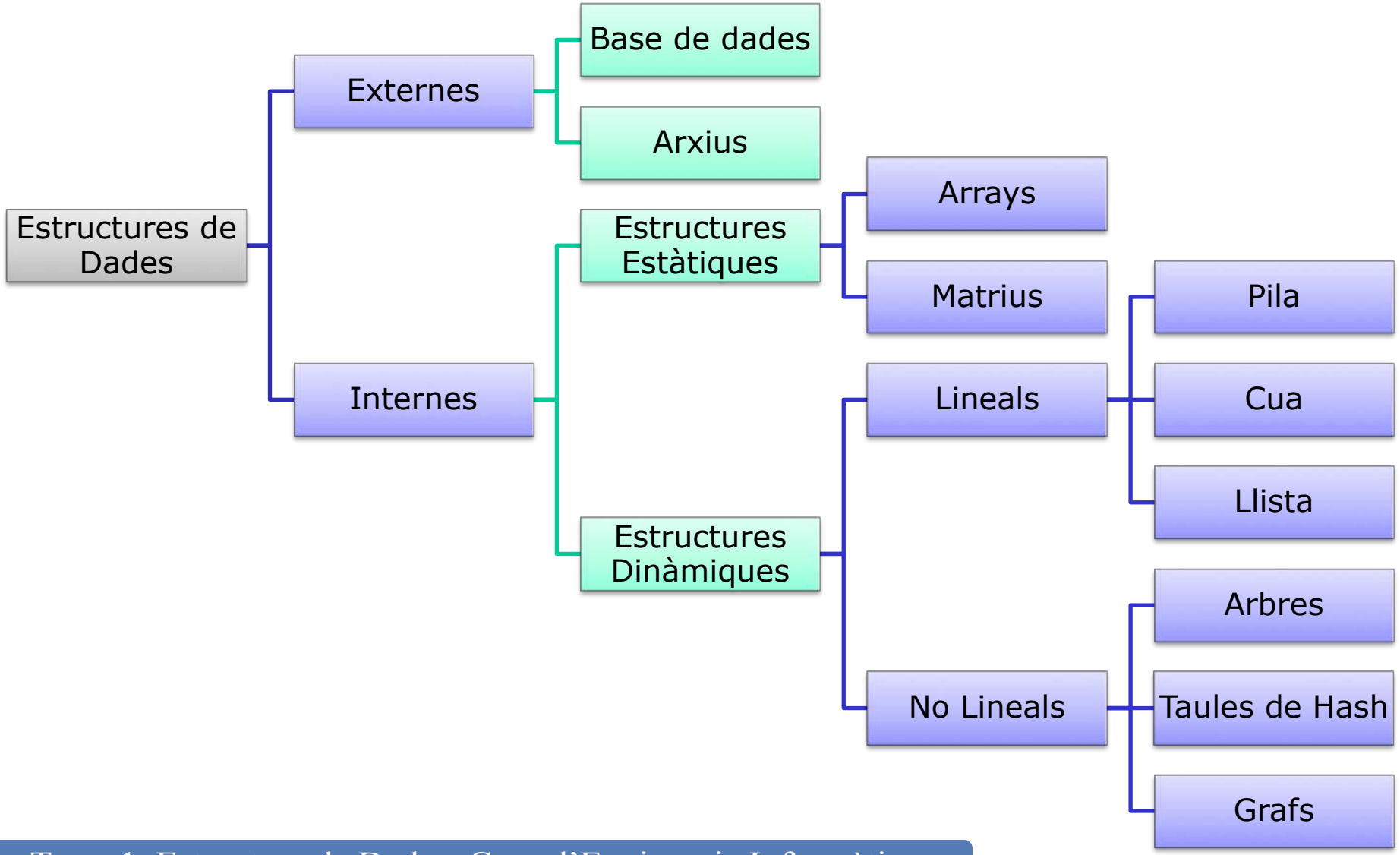
- Memòria d'emmagatzematge secundari o extern
  - Les estructures de dades que s'emmagatzemen aquí les hi crida **estructures de dades externes**
- Memòria principal
  - **Estructures de dades internes**
- Classificació d'Estructures de Dades segons tipus de memòria:
  - Parlem d'una **Estructura de dades estàtica** quan se li assigna una quantitat fixa de memòria per a aquesta estructura abans de l'execució del programa. La quantitat d'espai assignat per a la memòria estàtica es determina durant la compilació i no varia
  - Parlem d'una **Estructura de dades dinàmica** quan se li assigna memòria a mesura que és necessitada, durant l'execució del programa. En aquest cas la memòria no queda fixa durant la compilació. L'acte d'assignar memòria durant l'execució es diu **assignació dinàmica de la memòria**

Com s'organitza la memoria?



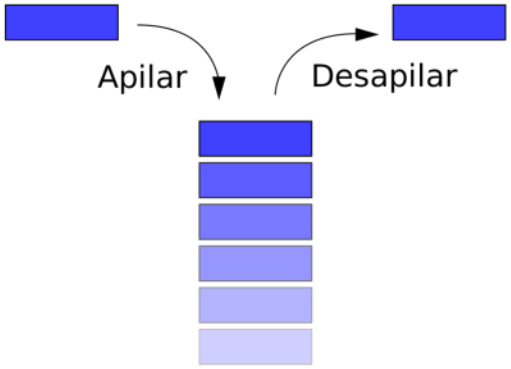
- El sistema de la computadora manté una pila d'execució, la quantitat de la qual d'espai que ocupa, variarà durant l'execució del programa
  - Cada vegada que un procés és invocat en un programa, es crea un registre d'activació i s'emmagatzema en la pila d'execució.
  - El registre conté espai per a totes les variables declarades en un procediment i una còpia als paràmetres que estan sent passats a un procediment. A més de la indicació per continuar on ha de seguir l'execució del programa, una vegada es completi la funció
- La memòria dinàmica s'emmagatzema en la memòria lliure, la que creix cap a la zona baixa
- Es produeixen errors quan:
  - Massa memòria és assignada dinàmicament
  - Es creen massa registres d'activació

Tipus d'estructures de dades

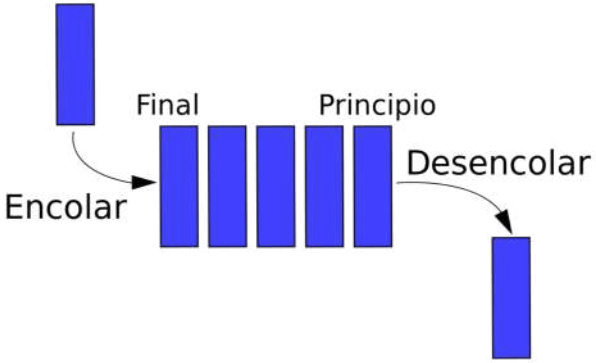


Exemples d'estructures de dades

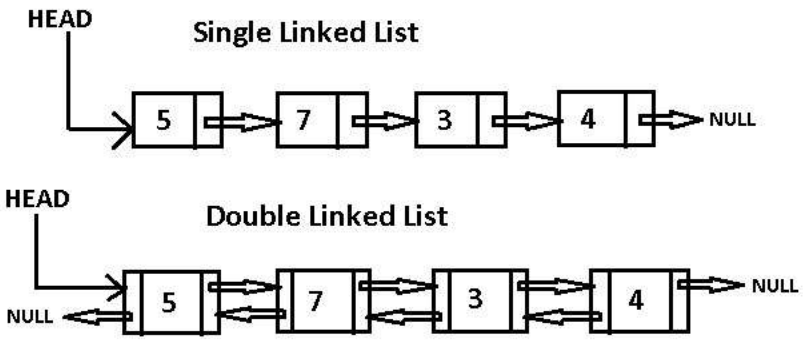
Pila



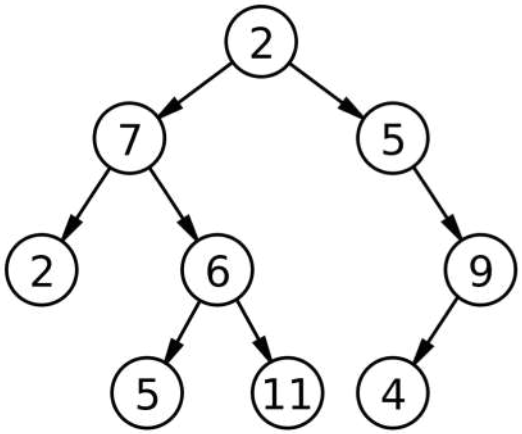
Cua



Llista



Arbre binari



Perquè són útils?

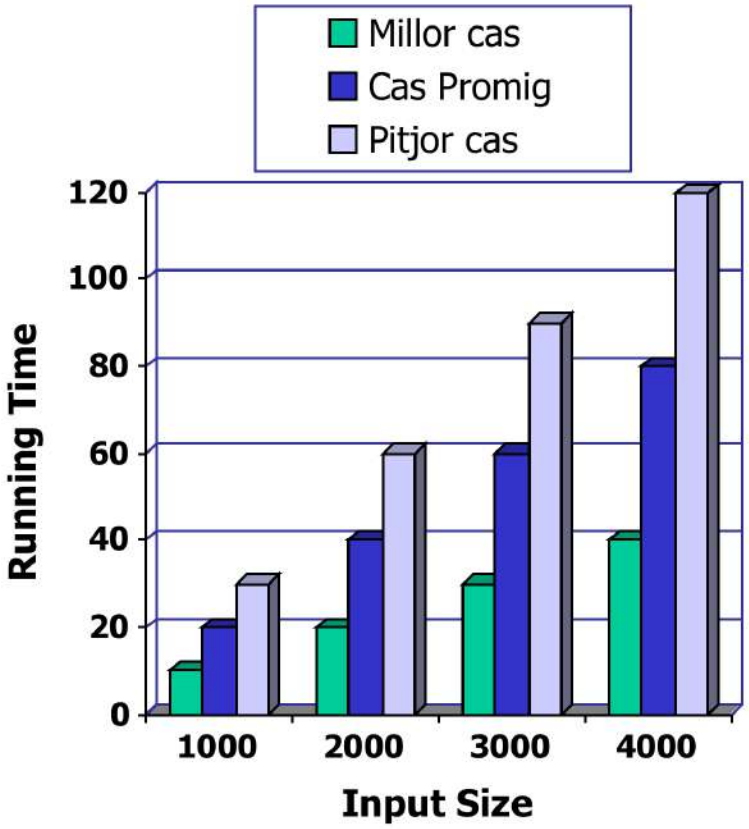
Una estructura de dades ens permet fer el software més eficient

L'eficiència s'ha de mesurar i depèn de:

- El temps d'execució
- Espai d'emmagatzematge

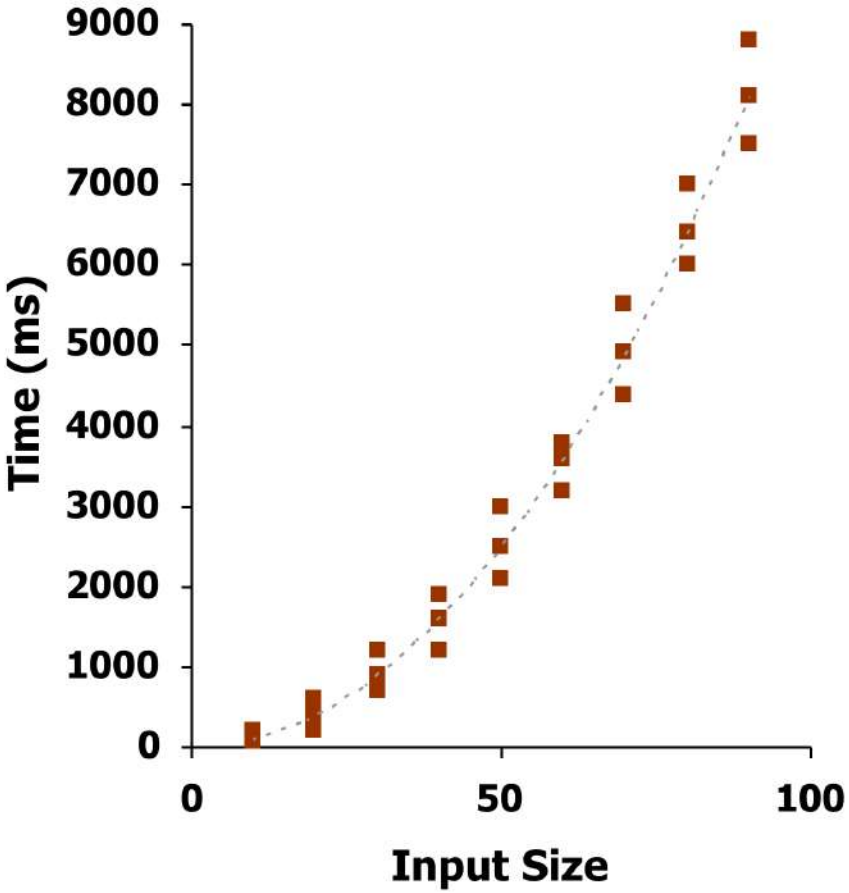
Quant de temps necessitem per processar les dades?

- La majoria d'algorismes transformen objectes d'entrada en objectes de sortida
- El temps de procés d'un algorisme **creix segons la mida** de les dades d'entrada
- El temps del cas promig és difícil de determinar
- Normalment s'enfoca des del **pitjor cas**
  - Més fàcil d'analitzar
  - Crucial per aplicacions com jocs, finances i robots



Mètode empíric de l'eficiència

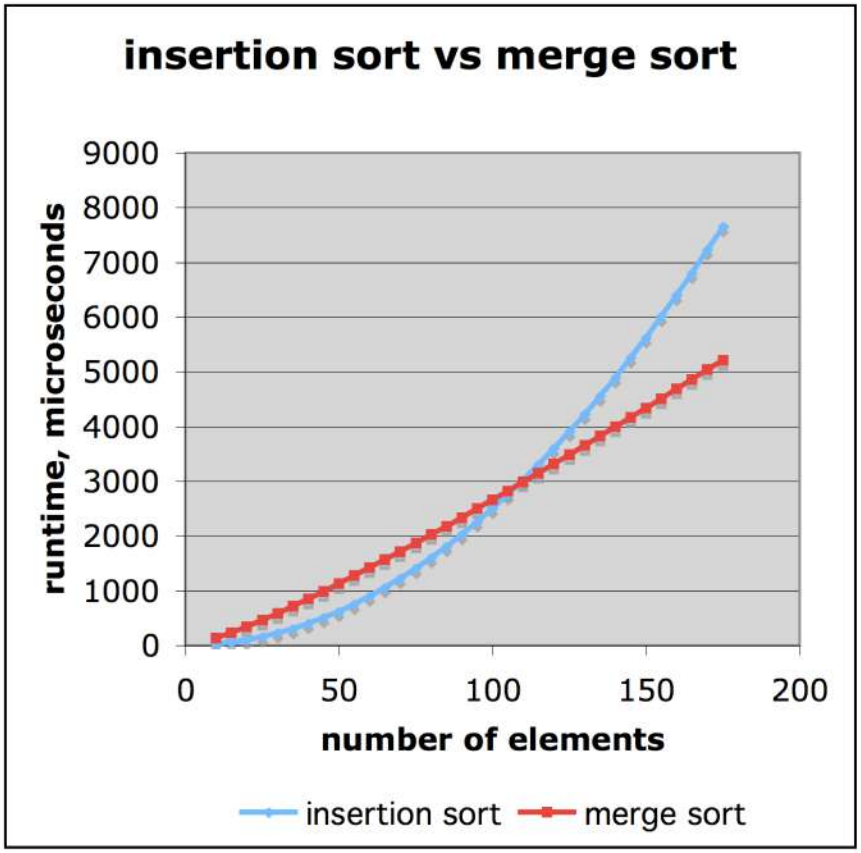
- Escriu un programa implementant l'algorisme
- Executa el programa amb una entrada que canviï de mida i tingui diferent composició
- Usa un mètode com clock() per obtenir una mesura acurada del temps actual d'execució
- Mostra els resultats en un gràfic



Limitacions del mètode empíric

- És **necessari implementar l'algorisme**, i pot ser difícil
- Els resultats no indicaran el temps per altres entrades no incloses en l'experiment
- Per poder comparar dos algorismes, s'ha d'usar el mateix entorn hardware i software

Comparativa de dos algorismes



insertion sort: $n^2 / 4$

merge sort: $2 n \log n$

Si ordenem un milió d'ítems?

- **insertion sort** triga unes **70 hores**

mentre que

- **merge sort** triga uns **40 segons**

En una màquina més ràpida, com 100 vegades més ràpida,

- **insertion sort** triga uns **40 minuts**

mentre que

- **merge sort** triga uns **0.5 segons**

Mètode analític de l'eficiència

- El mètode analític també es coneix com **anàlisi teòric** de l'eficiència
- Usa una descripció a més alt nivell de l'algorisme enlloc de la implementació
  - Resultat independent de la màquina i del llenguatge
- Caracteritza el temps d'execució com una funció, **f**, depenent de la mida de les dades, ***n***.
- Considera totes les possibles entrades
- Ens evitem programar el/s algorisme/s

Complexitat computacional

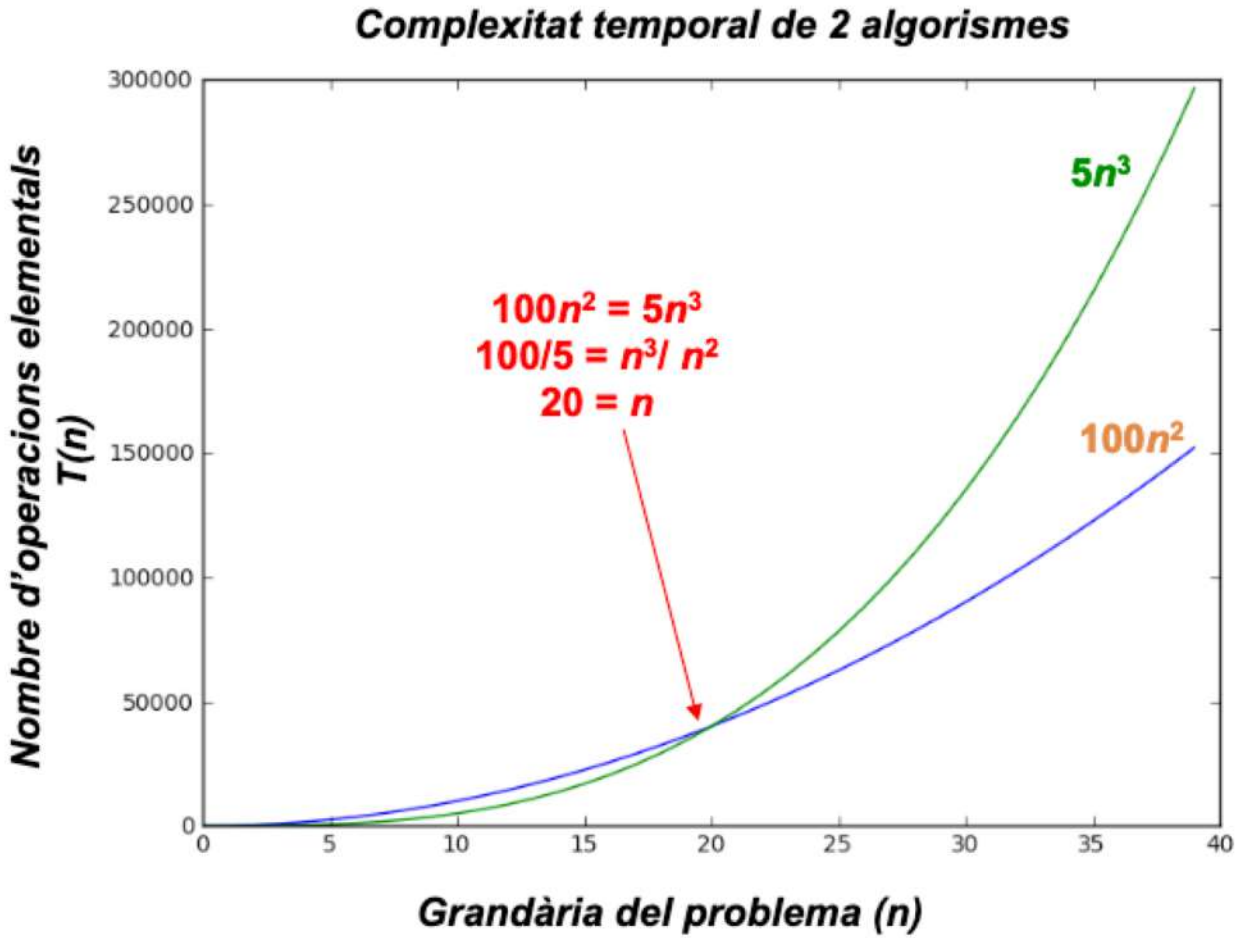
Com calculem la complexitat temporal d'un algorisme?

- f(n): **Funció de la grandària del problema** que proporciona el **temps d'execució** que inverteix una implementació d'un algorisme per a un problema de grandària **n**

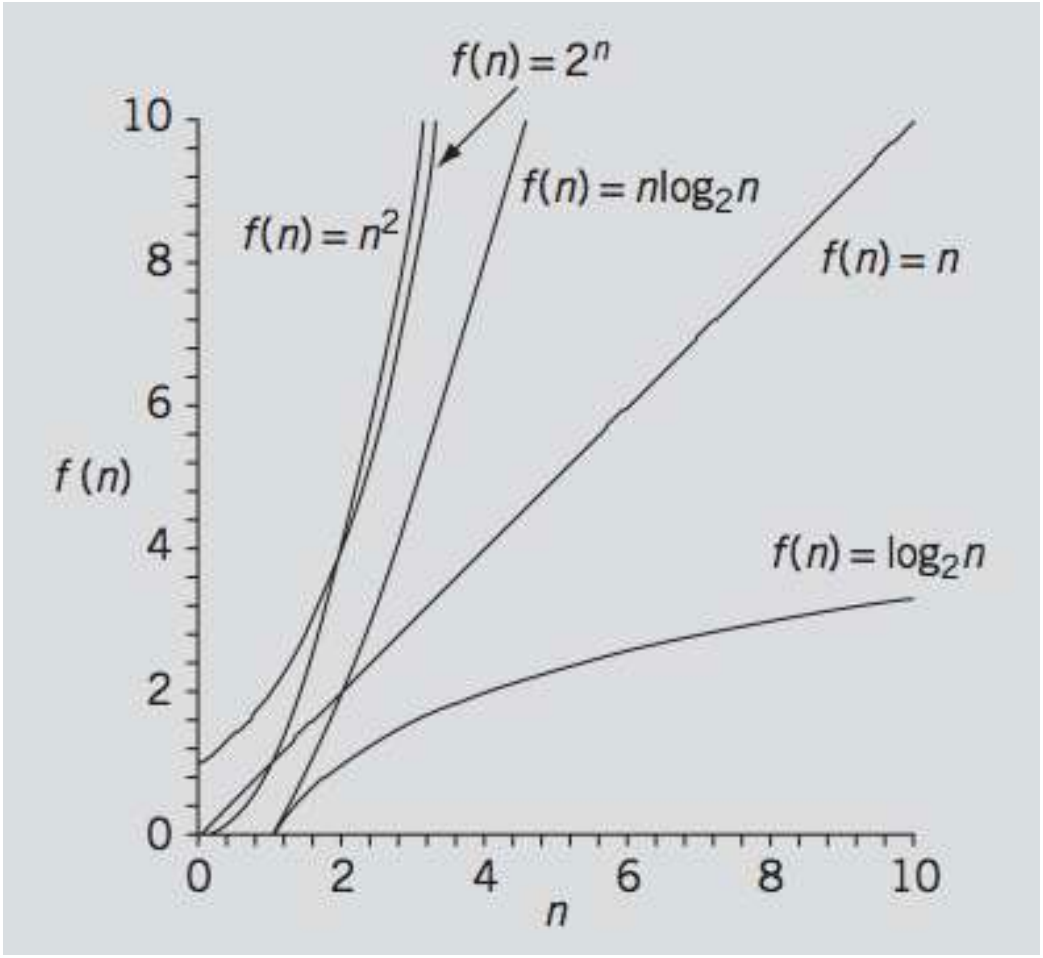
- f(n) es calcularà comptant el nombre d'operacions elements o **passos del programa** que realitza l'algorisme
  - Per tant, el resultat de f(n) és independent de la màquina concreta utilitzada

Quin algorisme és més eficient per a resoldre un mateix problema, un que triga $\mathbf{f(n) = 100n^2}$ o un que triga $\mathbf{f(n) = 5n^3}$?

n	$100n^2$	$5n^3$
1	100	5
…	…	…
10	10000	5000
…	…	…
20	40000	40000
…	…	…
30	90000	135000
…	…	…
40	160000	320000
…	…	…



7 Functions Importants



- Funcions que ens indiquen el cost temporal de diferents ordres de complexitat:
  - Constant ≈ **1**
  - Logarítmica≈ **log n**
  - Lineal ≈ **n**
  - N-Log-N ≈ **n log n**
  - Quadràtica ≈ $\mathbf{n^2}$
  - Cúbica ≈ $\mathbf{n^3}$
  - Exponencial ≈ $\mathbf{2^n}$

Cost temporal dels passos d'un programa

- Assignació, lectura, escriptura, retorn, expressió lògica, expressió aritmètica
  - Totes tenen un cost de 1 pas
- Seqüències d'instruccions: {S1;S2}
  - el cost serà el cost de S1 + cost de S2
- Sentències de selecció: if (condició) S1 else S2,
  - cost(condició) + MAXIM (cost (S1), cost (S2)}
- Bucle amb condició inicial: while(condició){S}
  - [Cost (condició) + cost (S)] * no. iteracions + cost(condició)
- Bucle amb condició final: do {S} while(condició)
  - [Cost(S) + Cost (condició)] * no. iteracions
- Bucle amb comptador: for(init;cond;incr) {S}
  - Cost(init) + [Cost(cond) + Cost (S)+ Cost(incr)] * no. Iteracions + Cost(cond)

```
main() {
  int  a, n, c;

  cin >> n;          ---------> 1
  a = 1;             ---------> 1
  while (a <= n) {
        c=calcula(n);
        a = a+1;     ---------> 1
  }
}

int calcula(int n) {
  int cal, i;
                     ---------> 1
  cal = 1;
  for (i=1; i <= n; i++) {
    cal := cal * n;  --------> 1
  }
  return(cal);       ---------> 1
}
```

$f(n)$ = Cost(main) = 2 + Cost(while) = $2+3n^2+7n+1 = 3n^2+9n+3 \rightarrow O(n^2)$

Cost(while) = [1+1+Cost(calcula)+1]*n+1 =

[1+1+3n+4+1]*n + 1 = $3n^2+7n+1$

Cost(calcula) = 2 + Cost(for) = 2+3n+2 = 3n+4

Cost(for) = 1+[1+1+1]*n +1 = 3n+2

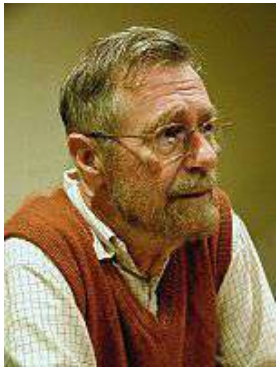
1.2 Ciència de la Computació i Programació

Ciència de la Computació

- **El seu objectiu NO és: estudiar els ordinadors!**

  ”Els ordinadors són per a la informàtica el que són els telescopis per l’astronomia" – Edsger Dijkstra (1930 - 2002)

  Dijkstra fou un científic informàtic holandès que va rebre el premi Turing per les seves contribucions fonamentals en el desenvolupament de llenguatges de programació al 1972



- **La ciència de la computació estudia**:
  - Com resoldre problemes (independentment de les màquines)
  - Com estudiar problemes sense solucions
  - Com fer els problemes computacionals (desenvolupar els algorismes)
  - Com usar els ordinador com a mitjà per resoldre problemes

Àrees en la ciència de la computació

La Ciència de la Computació se centra en respondre a les preguntes fonamentals sobre el que es pot calcular de forma computacional i quina quantitat de recursos són necessaris per a fer aquests càlculs

4 grans àrees

- La teoria de la computació
- **Algorismes** i **estructures de dades**
- Metodologia de programació i llenguatges
- Elements informàtics i arquitectura

Què és la programació?

- La programació és el procés de codificar un algorisme en un llenguatge de programació per a que sigui executable
- La **ciència de la computació ≠ programació**
- Els algorismes expressen la solució a través de:
  - Els passos necessaris per obtenir el resultat
  - La representació de les dades (tipus de dades)

- Els tipus de dades permeten interpretar les cadenes de bits representades en l'ordinador
- La complexitat dels problemes porta a necessitar tipus de dades complexes i sofisticats

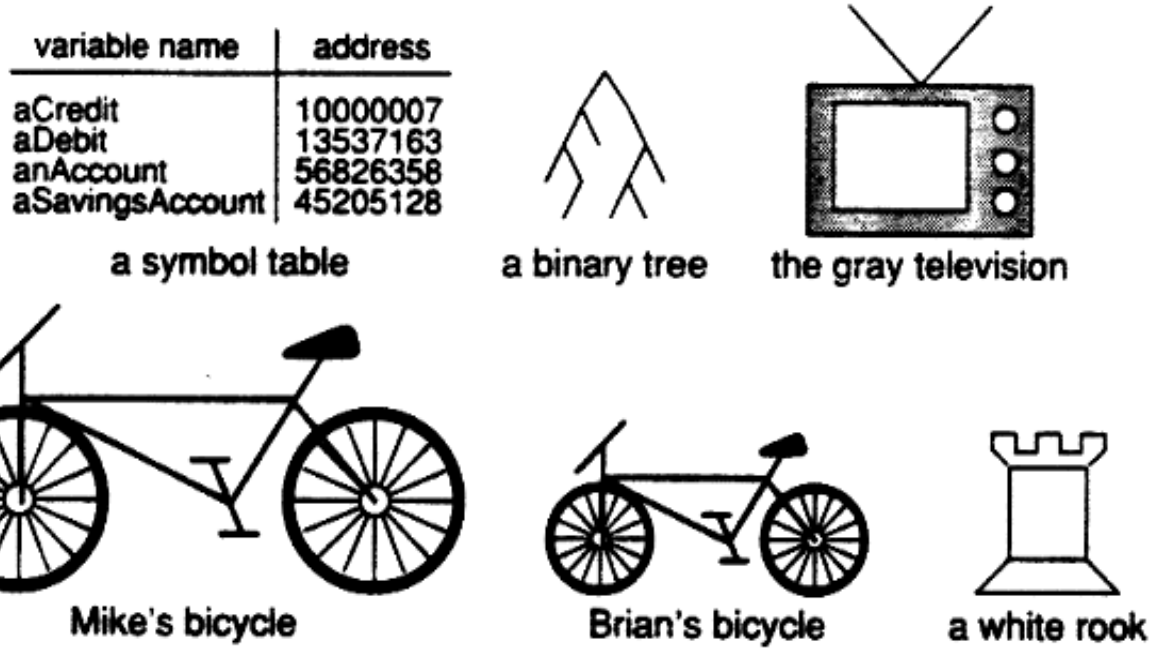
Programació Orientada a Objectes

- Orientació a Objectes (OO) consisteix en organitzar el software com una col·lecció discreta d'objectes que incorporen tant les **dades** com el **comportament**.
- La OO estructura el software en dades i el conjunt d'operacions associades a aquestes dades, en unes entitats anomenades **classes**.

- **Un programa és un conjunt d'objectes** (que pertanyen a diferents classes) que interactuen entre si per tal de resoldre el problema

Programació Orientada a Objectes

- La OO és una filosofia, no està lligada a cap llenguatge de programació
- Java i C++ són llenguatges de programació orientat a objectes (POO)



Objectes, classes i mètodes

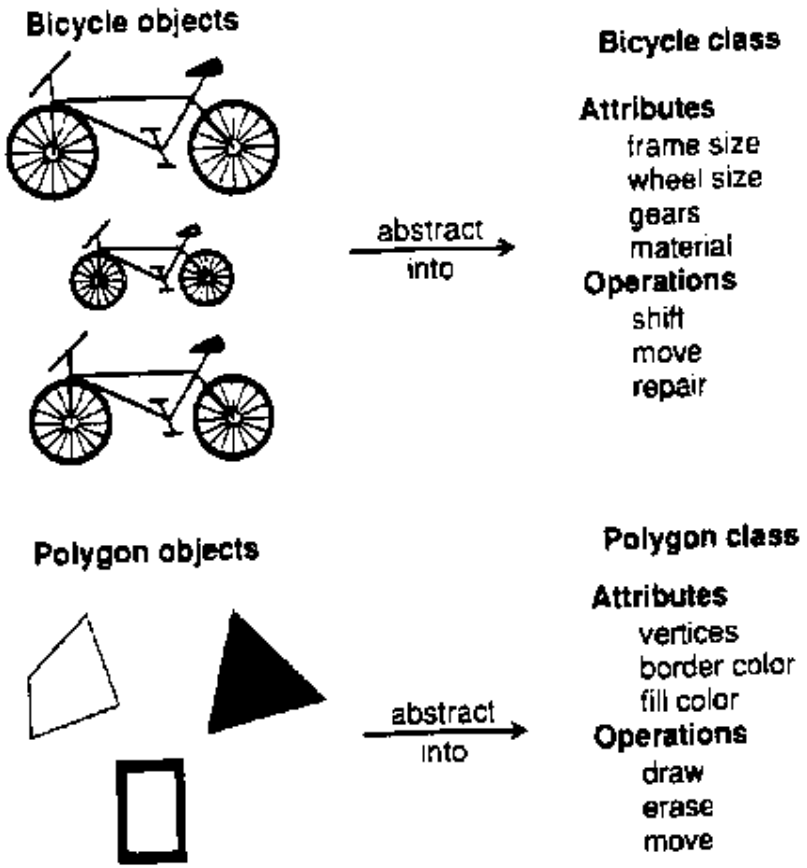
Objectes

- Els objectes es poden usar per representar entitats del mon real
- Un objecte està format per:
  - Un conjunt de dades (**atributs o estat**)
  - Un conjunt d'operacions (**mètodes o comportament**)
- El comportament d'un objecte pot modificar el seu estat
- Cada objecte té un identificador únic pel qual pot ser referenciat

Objectes, classes i mètodes

Classes

- Una **classe** és una abstracció d'un cert concepte
- Un **objecte** es defineix mitjançant una classe
- Es diu que un objecte és una **instància** d'una classe
- La classe representa un concepte i l'objecte representa la materialització d'aquest concepte



Objectes, classes i mètodes

Classes

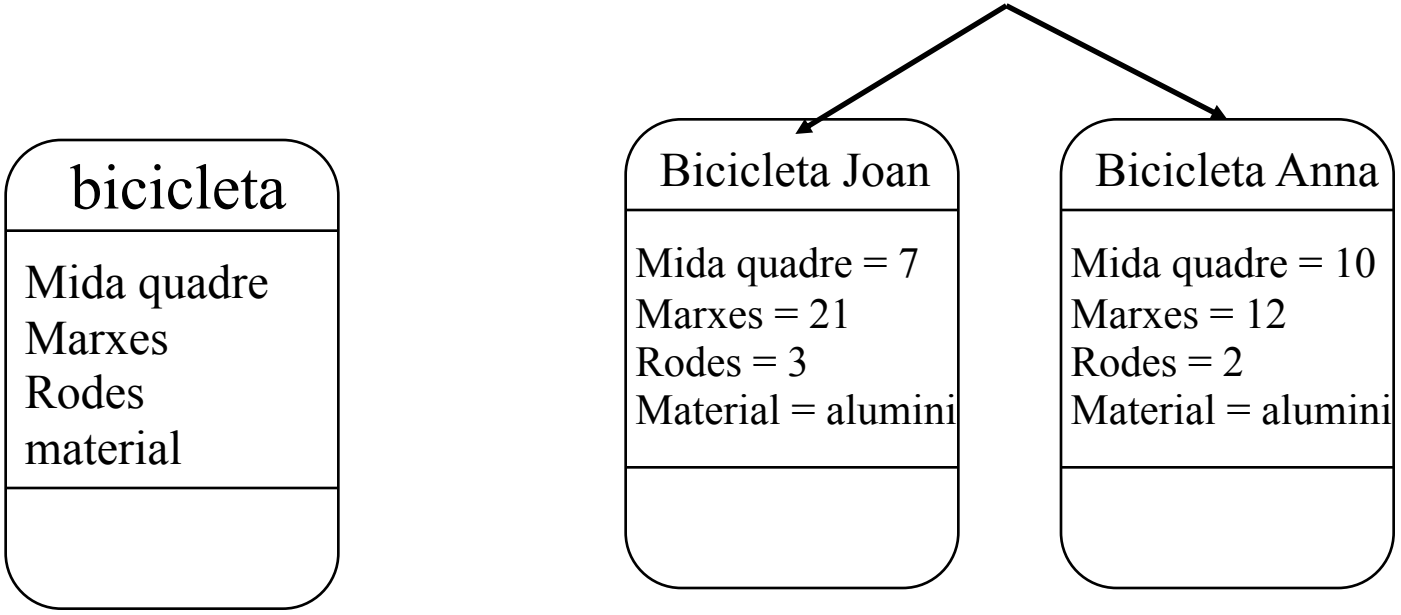
Una classe és un pla (o pla d'obra) d'un objecte

- La classe utilitza els mètodes per definir els comportaments de l'objecte
- Es poden crear múltiples objectes d'una mateixa classe
- Els objectes comparteixen el nom dels atributs i els mètodes però cada objecte té un valor concret per cada atribut
- La classe que conté el mètode **main** d'un programa JAVA o C++ representa el programa complet

Objectes, classes i mètodes

Objectes i Classes

Múltiples casos de la mateixa classe



1.3 Característiques de la OO

Característiques de la OO

Abstracció

- Consisteix en agafar una informació i extreure'n les característiques més representatives

Encapsulament

- Amaga a l'usuari la implementació interna de l'objecte

Característiques de la OO

Herència

- Defineix una relació entre classes.
- Una classe (**superclasse**) defineix un conjunt de propietats comuns a altres classes (**subclasses**).
- Les classes es poden organitzar en jerarquies

Polimorfisme

- Canvi de comportament d'una operació depenent de l'objecte al qual s'aplica.

1.4 Tipus Abstractes de Dades (TAD)

Tipus Abstractes de Dades (TADs)

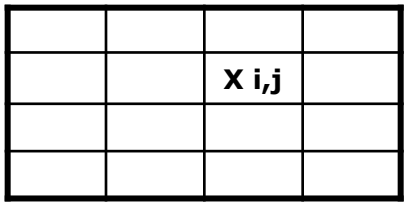
Definició

- "Un TAD és un ens **tancat i autosuficient**, que no requereix d'un context específic perquè pugui ser utilitzat en un programa, la qual cosa garanteix **portabilitat** i **reutilització** del programari i minimitza els efectes que puguin produir un canvi a l'interior del TAD"
  - Ocultar la representació del tipus de dades respecte de les aplicacions es coneix com a **encapsulament de les dades**
  - Aquesta manera de treballar té l'avantatge de la **reutilització de codi**

- La implementació d'un TAD es composa de:
  - Les variables necessàries per definir les dades
  - Les rutines d'accés a aquestes variables

Tipus Abstractes de Dades (TADs)

- **Exemple**:
  - TAD Matriu
  - Objecte abstracte:



  - Restriccions: n>0, m>0
  - Llista d'Operacions :
    - CrearMatriu(i, j)
    - AssignarMatriu(M, i, j, valor)
    - ObtenirDadaMatriu(M, i, j)
    - SumaMatriu(M1, M2)
    - MatriuNegativa(M)
    - RestarMatriu(M1, M2)
    - MatriuInversa(M)
    - MostrarMatriu(M)

Definició d'una classe

- La implementació d'un TAD es composa de:
  - Les variables necessàries per definir les dades → **ATRIBUTS**
  - Les rutines d'accés a aquestes variables → **MÈTODES**

- Una classe és un tipus de dades especial (TAD) que defineix com construir un cert tipus d'objecte

- La "**classe**" també emmagatzema la part de les dades que són compartides per totes les instàncies d'aquesta classe

- "**Instàncies**" són objectes que es creen i que segueixen la definició donada en l'interior de la classe

Universitat de Barcelona

Tema 1 **Introducció a les estructures de dades**

Grau d'Enginyeria Informàtica

Facultat de Matemàtiques i Informàtica,

Universitat de Barcelona