

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Tema 1. Introducció (Part 1)

Llenguatges, Tecnologies i Paradigmes de Programació (LTP)

DSIC, ETSInf



Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

- 1 Motivació
- 2 Conceptes essencials en llenguatges de programació
 - Tipus i sistemes de tipus
 - Polimorfisme
 - Reflexió
 - Procediments i control de flux
 - Gestió de memòria
- 3 Principals paradigmes de programació: imperatiu, funcional, lògic, orientat a objectes, concurrent
 - Paradigma imperatiu
 - Paradigma declaratiu
 - Paradigma orientat a objectes
 - Paradigma concurrent
- 4 Altres paradigmes: basat en interacció, emergents
 - Paradigma basat en interacció
- 5 Bibliografia

Objectius del tema

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

- Conèixer l'evolució dels llenguatges de programació (LP) i quins han sigut les seues aportacions més importants quant a l'impacte en el disseny d'altres llenguatges.
- Entendre els principals paradigmes de programació disponibles avui en dia i les seues característiques principals.
- Comprendre els diferents mecanismes d'abstracció (genericitat, herència i modularització) i pas de paràmetres.
- Identificar aspectes fonamentals dels LP: abast estàtic/dinàmic, gestió de memòria.
- Entendre els criteris que permeten triar el paradigma/llenguatge de programació més adequat en funció de l'aplicació, envergadura i metodologia de programació.
- Entendre les característiques dels LP en relació al model subjacent (paradigma) i als seus components fonamentals (sistemes de tipus i classes, model d'execució, abstraccions).
- Entendre les implicacions dels recursos expressius d'un LP quant a la seua implementació.

Una història que va començar en 1950

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

ANYS 50:

- Temps programador barat, màquines cares:
keep the machine busy
- Quan no es programava directament el hardware, el programa es compilava a mà per a obtenir la màxima eficiència, per a un hardware concret:
connexió directa entre llenguatge i hardware

ACTUALITAT:

- Temps programador car, màquines barates:
keep the programmer busy
- El programa es construeix per a ser eficient i es compila automàticament per a generar codi portable que siga, alhora, eficient:
connexió directa entre disseny del programa i llenguatges: objectes, concurrència, etc.

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Ensenyament dels LP

Tres aproximacions

- 1 Programació com un ofici
- 2 Programació com una branca de les matemàtiques
- 3 Programació en termes de conceptes

1. Programació com un ofici

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

- S'estudia en un paradigma únic i amb un únic llenguatge
- Pot ser contraproductiu. Per exemple, aprendre a manipular llistes en certs llenguatges pot portar a la conclusió errònia que el maneig de llistes és sempre tan complicat i costós:

1. Programació com un ofici

ZIP lists en Java

```
class Pair<A, B> {
    private A left;
    private B right;

    public Pair(A left, B right) {
        this.left = left;
        this.right = right;
    }

    public class MyZip {
        public static <A, B> List<Pair<A, B>> zip(List<A> as, List<B> bs) {
            Iterator<A> it1 = as.iterator();
            Iterator<B> it2 = bs.iterator();
            List<Pair<A, B>> result = new ArrayList<>();
            while (it1.hasNext() && it2.hasNext()) {
                result.add(new Pair<A, B>(it1.next(), it2.next()));
            }
            return result;
        }

        public static void main(String[] args) {
            List<Integer> x = Arrays.asList(1, 2, 3);
            List<String> y = Arrays.asList("a", "b", "c");
            List<Pair<Integer, String>> zipped = zip(x, y);
            System.out.println(zipped);
        }
    }
}
```

Eixida

```
[(1,a), (2,b), (3,c)]
```

```
public A left() { return left; }
public B right() { return right; }
public String toString() {
    return "(" + left + "," +
        right + ")";
}
```

ZIP lists en Haskell

```
zip :: [a] -> [b] -> [(a,b)]
```

```
zip [] xs           = []
zip (x:xs) []       = []
zip (x:xs) (y:ys) = (x,y):zip xs ys
```

Ús

```
: zip [1,2,3] ["a","b","c"]
[(1,"a"), (2,"b"), (3,"c")]
```

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusion

Reflexió

Procediments i control de flux

Pas de paramètres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

00

Concurrent

Altres paradigmes

Basat en
interacció

Bibliografia

2. Programació com una branca de les matemàtiques

- O bé s'estudia en un llenguatge *ideal*, restringit (Dijkstra) o el resultat és massa teòric, allunyat de la pràctica.

2. Programació com una branca de les matemàtiques

Exemple: verificació formal (d'un programa d'una línia)

El programa

```
while (x<10) x:=x+1;
```

La prova

Partim de l'expressió (*Hoare triple*)

```
{ $x \leq 10$ } while (x<10) x:=x+1 { $x=10$ }
```

La condició del bucle és $x < 10$. Usem el invariant de bucle $x \leq 10$ i amb aquestes assumpcions podem provar l'expressió

```
{ $x < 10 \wedge x \leq 10$ } x:=x+1 { $x \leq 10$ }
```

Aquesta expressió es deriva formalment de les regles de la lògica de Floyd-Hoare, però també pot justificar-se de forma intuïtiva: *La computació comença en un estat on es compleix $x < 10 \wedge x \leq 10$, la qual cosa és equivalent a dir que $x < 10$. La computació afeg 1 a x, per la qual cosa tenim que $x \leq 10$ és cert (en el domini dels enters)*

Sota aquesta premissa, la regla per al bucle while ens permet traure la conclusió

```
{ $x \leq 10$ } while (x<10) x:=x+1 { $\neg(x < 10) \wedge x \leq 10$ }
```

I podem veure que la postcondició d'aquesta expressió és lògicament equivalent a $x=10$.

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

3. Programació en termes de conceptes

- S'estudia un conjunt de **conceptes semàntics** i **estructures d'implementació** en termes dels quals es descriuen de forma natural diferents llenguatges i les seues implementacions

3. Programació en termes de conceptes

Un llenguatge de programació pot combinar característiques de diferents blocs

Llenguatge funcional

- (+) Polimorfisme
- (+) Estratègies
- (+) Ordre superior

Llenguatge lògic

- (+) No determinisme
- (+) Variables lògiques
- (+) Unificació

Llenguatge *kernel*

- (+) Abstracció de dades
- (+) Recursió
- (+) ...

Llenguatge imperatiu

- (+) Estats explícits
- (+) Modularitat
- (+) Components

Llenguatge OO

- (+) Classes
- (+) Herència

Llenguatge *dataflow*

- (+) Concurrencia

Conceptes essencials

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Destaquem els següents conceptes:

- Tipus i sistemes de tipus
- Polimorfisme
- Reflexió
- Pas de paràmetres
- Àmbit de les variables
- Gestió de memòria

Tipus i sistemes de tipus

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Un **tipus** representa el conjunt de valors que pot adoptar una variable o expressió. Els tipus:

- Ajuden a detectar **errors** de programació
*Solament els programes que utilitzen les expressions segons el tipus que tenen aplicant les funcions permeses són **legals***
- Ajuden a **estructurar** la informació
Els tipus poden veure's com a col·leccions de valors que comparteixen certes propietats
- Ajuden a manejar estructures de **dades**
Els tipus indiquen com utilitzar les estructures de dades que comparteixen el mateix tipus mitjançant certes operacions

Tipus i sistemes de tipus

Llenguatges tipificats

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

- En els llenguatges **tipificats**, les variables tenen un tipus associat (e.g., C, C++, C#, Haskell, Java, Maude).
- Els llenguatges que no restringeixen el rang de valors que poden adoptar les variables són **no tipificats** (e.g., Lisp, Prolog).
 - També pot entendre's que tots els valors tenen un tipus únic o universal

Tipus i sistemes de tipus

Llenguatges tipificats

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme
Sobrecàrrega
Coerció
Genericitat
Inclusió
Reflexió
Procediments i control de flux
Pas de paràmetres
Abast de les variables
Gestió de memòria

Paradigmes de programació

Imperatiu
Declaratiu
OO
Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

El sistema de tipus estableix què tipus d'associació de variables és possible:

- El **valor** associat a la variable ha de tenir el tipus d'aquesta (e.g., C, Haskell)
- El **valor** associat a la variable pot ser d'altres tipus *compatibles* relacionats amb el tipus de la variable (e.g., C++, C#, Java).
- De forma ortogonal, a més, la persistència del tipus del valor pot canviar:
 - **Estàtic**: el tipus no canvia durant l'execució
 - **Dinàmic**: el tipus pot canviar en canviar el valor associat

Tipus i sistemes de tipus

Llenguatges tipificats

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

$$\begin{array}{ccccc} \text{Llenguatges} & & & & \text{Expressions} \\ \text{tipificats} & = & & \text{de programa} & + & \text{Sistemes} \\ & & & & & \text{de tipus} \end{array}$$

- En els llenguatges amb tipificació **explícita**, els tipus formen part de la sintaxi.
- En els llenguatges amb tipificació **implícita**, els tipus **no** formen part de la sintaxi.

Tipus i sistemes de tipus

Exemples de tipificació

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

- Llenguatge **no tipificat**: Prolog

```
objecte(clau) .
objecte(pilota) .
cosa(X) <- objecte(X) .
```

La variable x no té tipus associat.

- Llenguatge amb **tipificació explícita**: Java

```
int x;
x = 42;
```

Totes les variables han de ser declarades, i en la declaració ha d'especificar-se el seu tipus explícitament

- Llenguatge amb **tipificació implícita**: Haskell

```
fac 0 = 1
fac x = x * fac (x-1)
```

El sistema de tipus infereix automàticament el tipus de la variable x

Tipus i sistemes de tipus

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme
Sobrecàrrega
Coerció
Genericitat
Inclusió
Reflexió
Procediments i control de flux
Pas de paràmetres
Abast de les variables
Gestió de memòria

Paradigmes de programació

Imperatiu
Declaratiu
OO
Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Per a definir el tipus de les variables o expressions usem un *llenguatge d'expressions de tipus*.

Exemple de llenguatge d'expressions de tipus

- Tipus bàsics o primitius: `Bool`, `Char`, `Int`, ...
- Variables de tipus: `a`, `b`, `c`, ...
- Constructors de tipus:
 - \rightarrow per a definir funcions,
 - \times per a definir parells,
 - `[]` per a definir llistes
 - ...
- Regles de construcció de les expressions:

$$\tau ::= \text{Bool} \mid \text{Char} \mid \text{Int} \mid \dots \mid \tau \rightarrow \tau \mid \tau \times \tau \mid [\tau]$$

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Tipus i sistemes de tipus

Tipus monomòrfics y tipus polimòrfics

- Els tipus en l'expressió de tipus dels quals **no** apareix cap variable de tipus es denominen **monotipus** o **tipus monomòrfics**.
- Els tipus en l'expressió de tipus dels quals apareix alguna variable de tipus es denominen **polítipus** o **tipus polimòrfics**.
- Un tipus polimòrfic representa un conjunt infinit de monotips

Tipus i sistemes de tipus

Exemples de expressions de tipus

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

- Expressió de tipus predefinit. Tipus bàsics `Bool`, `Int`, ...

`Bool` és el tipus dels valors booleans `True` i `False`

- Expressió de tipus funcional.

`Int` \rightarrow `Int` és el tipus de la funció `fact` vista abans, que retorna el factorial d'un nombre.

- Expressió de tipus parametritzat.

`[a]` \rightarrow `Int` és el tipus de la funció `length`, que calcula la longitud d'una llista.

Tipus i sistemes de tipus

Exemples de expressions de tipus

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

B
in

Bibliografia

- Expressió de tipus predefinit. Tipus bàsics `Bool`, `Int`, ...

`Bool` és el tipus dels valors booleans `True` i `False`

tipus monomòrfics

- Expressió de tipus funcional.

`Int → Int` és el tipus de la funció `fact` vista abans, que retorna el factorial d'un nombre.

tipus polimòrfic

- Expressió de tipus parametritzat.

`[a] → Int` és el tipus de la funció `length`, que calcula la longitud d'una llista.

variable de tipus

constructor de tipus

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

- És una característica dels llenguatges que permet manejar valors de diferents tipus usant una interfície uniforme
- S'aplica tant a funcions com a tipus:
 - Una funció pot ser polimòrfica pel que fa a un o varis dels seus arguments.

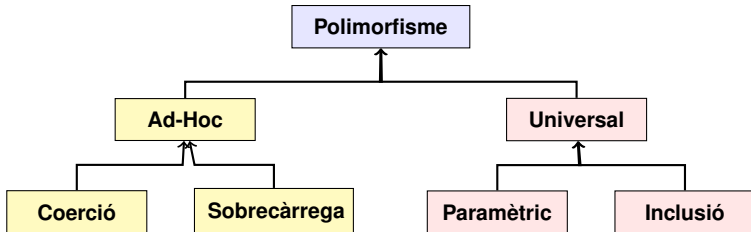
La summa (+) pot aplicar-se a valors de diferents tipus com a sencers, reals, ...

- Un tipus de dades pot ser polimòrfic pel que fa als tipus dels elements que conté.

Una llista amb elements d'un tipus arbitrari és un tipus polimòrfic

Polimorfisme

Tipus de polimorfisme



- Ad-hoc o aparent: treballa sobre un nombre finit de tipus no relacionats
 - Sobrecàrrega
 - Coerció
- Universal o vertader: treballa sobre un nombre potencialment infinit de tipus *amb certa estructura comuna*
 - paramètric (genericitat)
 - d'inclusió (herència)

- **Sobrecàrrega:** existència de diferents funcions amb el mateix nom.

- Els operadors aritmètics $+$, $-$, $*$, $/$, ... solen estar sobrecarregats:

$$(+) :: \text{Int} \rightarrow \text{Int} \rightarrow \text{Int}$$

```
(+) :: Float -> Float -> Float
```

```
(+) :: Complex -> Complex -> Complex
```

```
(+) :: Int -> Float -> Float
```

corresponen a diferents usos de +

- L'operador + no pot rebre el politipus

$$(+) :: a \rightarrow a \rightarrow a$$

perquè significaria dotar de significat (i implementació) a *la suma* de caràcters, funcions, llistes, etc., la qual cosa pot interessar-nos o no

Polimorfisme. Sobrecàrrega

Exemple de sobrecàrrega en Java (1)

En Java, la sobrecàrrega de mètodes es realitza canviant el tipus dels paràmetres:

```
/* overloaded methods */  
  
int myAdd(int x, int y, int z) {  
    ...  
}  
  
double myAdd(double x, double y, double z) {  
    ...  
}
```

Polimorfisme. Sobrecàrrega

Exemple de sobrecàrrega en Java (2)

```
public class Overload {  
    public void numbers(int x, int y) {  
        System.out.println("Method that gets integer numbers");  
    }  
    public void numbers(double x, double y, double z) {  
        System.out.println("Method that gets real numbers");  
    }  
    public int numbers(String st) {  
        System.out.println("The length of " + st + " is " +  
                           st.length());  
        return st.length();  
    }  
    public static void main(...) {  
        Overload s = new Overload();  
        int a = 1;  
        int b = 2;  
        s.numbers(a,b);  
        s.numbers(3.2,5.7,0.0);  
        a = s.numbers("Madagascar");  
    }  
}
```

No té per què haver-hi coincidència quant al nombre ni quant al tipus dels paràmetres/resultat

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Polimorfisme. Coerció

Polimorfisme Ad-Hoc: Coerció

- **Coerció:** conversió (implícita o explícita) de valors d'un tipus a un altre.
- Quan és implícita sol fer-se usant una jerarquia de tipus o de la seua representació.

Per exemple, en la majoria de llenguatges, per als arguments dels operadors aritmètics existeix coerció entre valors sencers i reals

- Alguns llenguatges permeten forçar una coerció explícita.
 - Llenguatges de la família de C (sentència *Cast*)
 - En Java és possible transformar:
 - una variable primitiva d'un tipus bàsic a un altre
 - un objecte d'una classe a una superclasse

Polimorfisme. Coerció

Exemple de Coerció en Java

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Conversió implícita en Java:

```
int num1 = 100        // 4 bytes
long num2 = num1       // 8 bytes
```

Conversió explícita en Java:

```
int num1 = 100                // 4 bytes
short num2 = (short) num1     // 2 bytes
char c = (char) num1          // 2 bytes

String s = Integer.toString(num1)
```

Polimorfisme. Genericitat

Polimorfisme Universal: Genericitat

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

- **Genericitat/Paramètric**: la definició d'una funció o la declaració d'una classe presenta una estructura que és comuna a un nombre potencialment infinit de tipus

- En Haskell podem definir i usar tipus genèrics i funcions genèriques
- En Java podem definir i usar classes genèriques i mètodes genèrics

Polimorfisme. Genericitat

Exemple de Genericitat en Haskell

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de

programació

Imperatiu

Declaratiu

OO

Concurrent

Altres

paradigmes

Basat en interacció

Bibliografia

Usant **un tipus genèric** (amb variables de tipus), podem definir una estructura de dades per a representar i manipular les entrades (de qualsevol tipus) d'un *diccionari*:

```
type Entry k v = (k,v)

getKey :: Entry k v -> k
getKey (x,y) = x

getValue :: Entry k v -> v
getValue (x,y) = y
```

Amb una **funció genèrica** podem calcular la longitud d'una llista els elements de la qual són de qualsevol tipus:

```
length :: [a] -> Integer
length [] = 0
length (x:xs) = 1 + (length xs)
```

Polimorfisme. Genericitat

Exemple de Genericitat en Java (1/2)

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de

programació

Imperatiu

Declaratiu

OO

Concurrent

Altres

paradigmes

Basat en interacció

Bibliografia

Podem usar una classe genèrica (amb paràmetres) per a definir una entrada d'un *diccionari*:

```
public class Entry<K,V>{
    private final K mKey;
    private final V mValue;

    public Entry(K k, V v){
        mKey = k;
        mValue = v;
    }
}
```

```
public K getKey(){
    return mKey;
}

public V getValue(){
    return mValue;
}
}
```

Podem definir un **mètode genèric** per a calcular la longitud d'un array de *qualsevol* tipus:

```
public static <T> int lengthA(T[] inputArray){
    ...
}
```

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Polimorfisme. Genericitat

Exemple de Genericitat en Java (2/2)

Exemple d'ús d'entrades d'un diccionari:

parametrización



```
Entry<Integer,String> elem1 = new Entry<>(3,"Programming");  
System.out.println(elem1.getValue());
```

Exemple d'ús del mètode genèric per a la longitud d'un array:

```
Integer[] intArray = {1, 2, 3, 5};  
Double[] doubleArray = {1.1, 2.2, 3.3};  
  
System.out.println("Array length =" + lengthA(intArray));  
System.out.println("Array length =" + lengthA(doubleArray));
```


Polimorfisme. Genericitat

Algunes consideracions de la genericitat Java

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

- Una classe genèrica és una classe convencional, llevat que dins de la seua declaració utilitza una **variable de tipus** (paràmetre), que serà definit quan siga utilitzat.
- Dins d'una classe genèrica es poden utilitzar altres classes genèriques
- Una classe genèrica pot tenir diversos paràmetres

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Polimorfisme. Inclusió

Polimorfisme Universal: Inclusió/Herència

- **Inclusió o Herència:** la definició d'una funció treballa sobre tipus que estan relacionats seguint una jerarquia d'inclusió.
- En l'orientació a objectes l'herència és el mecanisme més utilitzat per a permetre la **reutilització** i **extensibilitat**.

*L'herència organitza les classes en una estructura jeràrquica formant **jerarquies de classes***

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerciò

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Polimorfisme. Inclusió

Polimorfisme Universal: Inclusió/Herència

IDEA:

Una classe B heretarà d'una classe A quan volem que B tinga l'estructura i comportaments de la classe A. A més podem

- afegir nous atributs a B
- afegir nous mètodes a B

I depenent del llenguatge podem

- redefinir mètodes heretats
- heretar de diverses classes (en Java solament podem heretar d'una classe)

Polimorfisme. Inclusió

Exemple de Herència en Java (1/2)

```
public class Bicycle {
    protected int cadence;
    protected int gear;
    protected int speed;
    public Bicycle (int startCad, int startSpeed,
                    int starteGear) {
        cadence = startCad;
        speed = startSpeed;
        gear = startGear;
    }
    public void setCadence(int newValue) {
        cadence = newValue; }
    public void setGear(int newValue) {
        gear = newValue; }
    public void applyBrake(int decrement) {
        speed -= decrement; }
    public void speedUp(int increment) {
        speed += increment; }
}
```

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Polimorfisme. Inclusió

Exemple de Herència en Java (2/2)

```
public class MountainBike extends Bicycle {
    public int seatHeight;
    public MountainBike(int startHeight, int startCad,
                        int startSpeed, int starteGear){
        super(startCad, startSpeed, StartGear);
        seatHeight = startHeight;
    }
}
```

- Les subclasses es defineixen usant la paraula clau **extends**
- Es poden afegir atributs, mètodes i redefinir mètodes

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

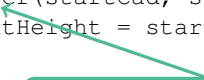
Basat en interacció

Bibliografia

Polimorfisme. Inclusió

Exemple de Herència en Java (2/2)

```
public class MountainBike extends Bicycle {  
    public int seatHeight;  
    public MountainBike(int startHeight, int startCad,  
                        int startSpeed, int startGear){  
        super(startCad, startSpeed, startGear);  
        seatHeight = startHeight;  
    }  
}
```



constructor de la classe pare

- Les subclasses es defineixen usant la paraula clau **extends**
- Es poden afegir atributs, mètodes i redefinir mètodes

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Polimorfisme. Inclusió

Algunes consideracions de la herència Java (1/3)

- En Java, la base de qualsevol jerarquia és la classe `Object`.
- Si una classe es declara com `final`, no es pot heretar d'ella
- Java solament té herència simple
- A una variable de la superclasse se li pot assignar una referència a qualsevol subclasse derivada d'aquesta superclasse, però **no** al contrari.

Exemple d'assignació vàlida

```
Bicycle b;  
MountainBike m = new MountainBike(75, 90, 25, 8);  
b = m
```

Polimorfisme. Inclusió

Algunes consideracions de la herència en Java (2/3)

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

- En Java s'usen qualificadors davant dels atributs i mètodes per a establir què variables d'instància i mètodes dels objectes d'una classe són visibles
 - **Private:** cap membre o atribut `private` de la superclasse és visible en les subclasses o altres classes.
Si s'usa per a atributs de classe, hauran de definir-se mètodes que accedisquen a aquests atributs
 - **Protected:** els membres `protected` de la superclasse són visibles en la subclasse, però no visibles per a l'exterior.
 - **Public:** els membres públics de la superclasse segueixen sent públics en la subclasse.
 - **Default:** els membres amb visibilitat `default` són visibles des de qualsevol classe que estiga en el mateix paquet

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Polimorfisme. Inclusió

Algunes consideracions de la herència en Java (3/3)

	Classe	Paquet	Subclasse	Altres
Public	Sí	Sí	Sí	Sí
Private	Sí	No	No	No
Protected	Sí	Sí	Sí	No
Default	Sí	Sí	No	No

Cuadro: Visibilitat en Java

Polimorfisme. Inclusió

Exemple de redefinició de mètode heretat en Java (1/2)

```
public class Employee {
    String name;
    int nEmployee, salary;
    static private int counter = 0;
    public Employee (String name, int salary){
        this.name = name;
        this.salary = salary;
        nEmployee = ++counter;
    }
    public void increaseSalary(int wageRaise){
        salary += (int) (salary*wageRaise/100);
    }
    public String toString(){
        return "Num. Employee " + nEmployee +
            " Name: " + name + " Salary: " + salary;
    }
}
```

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Polimorfisme. Inclusió

Exemple de redefinició de mètode heretat en Java
(2/2)

```

public class Executive extends Employee{
    int budget;
    public Executive(String name, int salary){
        super(name, salary);
    }
    void assignBudget(int b){
        budget = b;
    }
    public String toString(){
        String s = super.toString();
        s = s + " Budget: " + budget;
        return s;
    }
}

```

Exemple d'ús:

```

Executive boss = new Executive("Thomas Turner", 1000);
boss.assignBudget(1500);

```

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Polimorfisme. Inclusió

Herència en Java: Classes abstractes

- Una classe abstracta és la que es declara com `abstract`
 - Si una classe té un mètode `abstract` és obligatori que la classe siga `abstract`.
 - Per als mètodes declarats `abstract` no es dóna implementació.
 - Una classe abstracta no pot tenir instàncies.
- Totes les subclasses que hereten d'una classe abstracta, si elles no són abstractes hauran de redefinir els mètodes abstractes donant-los una implementació.

Polimorfisme. Inclusió

Exemple d'ús de classes abstractes en Java (1/2)

```
public abstract class Shape {  
    private float x, y; // Position of the shape  
    public Shape (float initX, float initY){  
        x = initX; y = initY;  
    }  
    public void move(float incX, float incY){  
        x = x+incX; y = y+incY;  
    }  
    public float getX(){ return x; }  
    public float getY(){ return y; }  
    public abstract float perimeter();  
    public abstract float area();  
}
```

Polimorfisme. Inclusió

Exemple d'ús de classes abstractes en Java (2/2)

```
public class Square extends Shape {
    private float side;
    public Square (float initX, float initY, float initSide){
        super(initX,initY); // Call to super constructor
        side = initSide;
    }
    public float perimeter(){ return 4*side; }
    public float area(){ return side*side; }
}

public class Cicle extends Shape {
    private float radius;
    public Circle(float initX, float initY, float initRadius){
        super(initX,initY); // Call to super constructor
        radius = initRadius;
    }
    public float perimeter(){ return 2*pi*radius; }
    public float area(){ return pi*radius*radius; }
}
```

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Polimorfisme. Inclusió

Exemple d'ús de classes abstractes en Java (2/2)

```

public class Square extends Shape {
    private float side;
    public Square (float initX, float initY, float initSide){
        super(initX,initY); // Call to super constructor
        side = initSide;
    }
    public float perimeter(){ return 4*side; }
    public float area(){ return side*side; }
}

public class Circle extends Shape {
    private float radius;
    public Circle(float initX, float initY, float initRadius){
        super(initX,initY);
        radius = initRadius;
    }
    public float perimeter(){ return 2*pi*radius; }
    public float area(){ return pi*radius*radius; }
}

```

I si volguérem estendre aquest Example amb una forma nova com el triangle, què cal fer?

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Polimorfisme. Inclusió

Herència en Java: Interfícies

Una interfície **declara** els atributs i operacions que han de definir-se en les classes que implementen (**implements**) aquesta interfície

```
public interface MyInterface {
    public int method1(...);
    ...}

public class MyClass implements MyInterface {
    public int method1(...) {...}
    ...}
```

- Els mètodes d'una interfície poden ser abstractes, estàtics i default (inclouen una implementació per defecte). La seua visibilitat pot ser “public”, “private” i “default”, però no “protected”.
- Els atributs són estàtics i finals (constants)
- Una classe pot implementar diverses interfícies
- Les interfícies poden heretar d'altres interfícies (**extends**)

Questió: Inclusió y genericitat

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Donades la següent definició de classes:

```
class Shape { /*...*/ }  
class Circle extends Shape { /*...*/}  
class Rectangle extends Shape { /*...*/ }  
class Node<T> { /*...*/ }
```

Compila sense error el següent fragment de codi? Per què?

```
Node<Circle> nc = new Node<Circle>();  
Node<Shape> ns = nc;
```

Qüestió: Inclusió i genericitat

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Respon a les següents qüestions:

- Pot una interfície heretar d'una classe?
- Es poden crear instàncies de les classes interfície?

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Què és la reflexió

Quan et mires en un espill pots:

- veure el teu reflex i
 - reaccionar davant el que veus
-
- En els **llenguatges de programació**, la reflexió és **la infraestructura** que, durant la seua execució, permet a un programa:
 - veure la seua pròpia estructura i
 - manipular-se a si mateix
 - La reflexió es va introduir amb el llenguatge LISP i està present en alguns llenguatges de script.

Permet, per exemple, definir programes capaços de monitoritzar la seua pròpia execució i modificar-se, en temps d'execució, per a adaptar-se dinàmicament a diferents situacions

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Reflexió

Llenguatges amb reflexió

- Es caracteritzen perquè les pròpies instruccions del llenguatge són tractades com a valors d'un tipus de dades específic; En els llenguatges sense reflexió es veuen com a simples cadenes de caràcters
- Els llenguatges amb reflexió poden veure's com **metallenguatges** del propi llenguatge.

Es diu metallenguatge a aquell amb el qual podem escriure metaprogrames (programes que manipulen programes com a compiladors, analitzadors, etc.)

- És una característica avançada però no complicada, especialment en llenguatges funcionals, gràcies a la dualitat natural entre dades i programes (homoiconicitat).

Reflexió

La Reflexió en Java

- En Java la reflexió s'usa mitjançant la biblioteca `java.lang.reflect`

La biblioteca proporciona classes per a representar de forma estructurada informació de les classes, variables, mètodes, etc.

- Es pot inspeccionar classes, interfícies, atributs i mètodes sense conèixer els noms dels mateixos en temps de compilació. Per exemple podem
 - obtenir i mostrar durant l'execució del programa el nom de totes les instàncies de la classe que s'han creat en temps d'execució.
 - llegir de teclat un `String` amb el qual poder crear un objecte amb aqueix nom, o invocar un mètode amb aqueix nom.

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Reflexió

Exemple d'us de Reflexió en Java

```
import java.lang.reflect.*;

public class MyClass {...}

...
MyClass myClassObj = new MyClass();
// get the class information:
Class<? extends MyClass> objMyClassInfo =
    myClassObj.getClass();

// get the fields:
Field[] allDeclaredVars = objMyClassInfo.getDeclaredFields();
// travel the fields:
for (Field variable : allDeclaredVars) {
    System.out.println("Name of GLOBAL VARIABLE: " +
        variable.getName());
}
```

Altres mètodes definits en la classe Class:

```
Constructor[] getConstructors();
Field[]        getDeclaredFields();
Method[]       getDeclaredMethods();

...
```

Procediments i control de flux

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Existeixen alguns conceptes relacionats amb el control de flux dels programes i la definició i cridada a procediments.

- **Pas de paràmetres.** Quan es fa una crida a un mètode o funció hi ha un canvi de context que pot fer-se de diferents formes. Veurem les principals.
- **Àmbit de les variables.** És necessari determinar si un objecte o variable és visible en un moment determinat de l'execució i aquest càlcul pot fer-se de forma estàtica o bé de forma dinàmica.

Pas de paràmetres

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Un dels mecanisme d'abstracció bàsic és organitzar les tasques d'un programa definint funcions, mètodes o procediments que resolen subtasques. Així, en un moment determinat es pot **invocar** a aquests procediments.

- CRIDA: $f(e_1, \dots, e_n)$ amb e_1, \dots, e_n expressions.
 - en executar-se la crida, el flux de control passarà al cos de la funció f i, una vegada aquest acabe, tornarà al flux des del qual es va fer la crida.
 - e_1, \dots, e_n són els anomenats **paràmetres d'entrada/reals** (*actual parameters*)
- DECLARACIÓ: $f(x_1, \dots, x_n)$ amb x_1, \dots, x_n variables.
 - x_1, \dots, x_n són els anomenats **paràmetres formals** (*formal parameters*)
 - Els paràmetres formals són variables locals al cos de la funció declarada

Pas de paràmetres

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Es distingeixen tres tipus de pas de paràmetres

- Pas per valor (*call by value*)
- Pas per referència (*call by reference/call by address*)
- Pas per necessitat (*call by need*)

Existeixen més modalitats de pas de paràmetres però aquestes tres són les més freqüents en els llenguatges de programació

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Pas de paràmetres

Pas per valor

Es calculen els valors v_i dels paràmetres d'entrada e_i en la crida i es copien en els paràmetres formals x_i

- en el cos de la funció es treballa sobre una referència a memòria diferent.

```
void inc(int v)
{
    v = v + v;
}

...
int a = 10;
inc(a);
```

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Pas de paràmetres

Pas per valor

Es calculen els valors v_i dels paràmetres d'entrada e_i en la crida i es copien en els paràmetres formals x_i

- en el cos de la funció es treballa sobre una referència a memòria diferent.

```
void inc(int v)
{
    v = v + v;
}

...
int a = 10;
inc(a);
```

a = 10

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Pas de paràmetres

Pas per valor

Es calculen els valors v_i dels paràmetres d'entrada e_i en la crida i es copien en els paràmetres formals x_i

- en el cos de la funció es treballa sobre una referència a memòria diferent.

```
void inc(int v)
{
    v = v + v;
}
...
int a = 10;
inc(a);
```

a = 10

En la crida:

Es copia el valor 10 en el paràmetre formal v

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Pas de paràmetres

Pas per valor

Es calculen els valors v_i dels paràmetres d'entrada e_i en la crida i es copien en els paràmetres formals x_i

- en el cos de la funció es treballa sobre una referència a memòria diferent.

```
void inc(int v)
{
    v = v + v;
}
...
int a = 10;
inc(a);
```

v = 10

a = 10

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Pas de paràmetres

Pas per valor

Es calculen els valors v_i dels paràmetres d'entrada e_i en la crida i es copien en els paràmetres formals x_i

- en el cos de la funció es treballa sobre una referència a memòria diferent.

```
void inc(int v)
{
    v = v + v;
}
...
int a = 10;
inc(a);
```

 $v = 20$ $a = 10$

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Pas de paràmetres

Pas per valor

Es calculen els valors v_i dels paràmetres d'entrada e_i en la crida i es copien en els paràmetres formals x_i

- en el cos de la funció es treballa sobre una referència a memòria diferent.

```
void inc(int v)
{
    v = v + v;
}
...
int a = 10;
inc(a);
```

a = 10

- La variable **a** NO es modifica perquè es treballa amb una còpia en la funció `inc`.

Pas de paràmetres

Pas per referència

Es passa **la referència** a memòria, per la qual cosa el cos de la funció treballa sobre el mateix objecte en memòria

- Per als paràmetres d'entrada e_i que **no** siguin una variable, funciona com el pas per valor
- Quan e_i és una variable (i.g., y_i), les assignacions realitzades sobre el paràmetre formal x_i alteren també el valor associat a y_i

```
void inc(int v)
{
    v = v + v;
}

...
int a = 10;
inc(a);
```

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Pas de paràmetres

Pas per referència

Es passa **la referència** a memòria, per la qual cosa el cos de la funció treballa sobre el mateix objecte en memòria

```
void inc(int v)
{
    v = v + v;
}

...
int a = 10;
inc(a);
```

a = 10

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Pas de paràmetres

Pas per referència

Es passa **la referència** a memòria, per la qual cosa el cos de la funció treballa sobre el mateix objecte en memòria

```
void inc(int v)
{
    v = v + v;
}

...
int a = 10;
inc(a);
```

a = 10

En la crida:

El paràmetre formal `v` rep l'adreça de memòria de `a`

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Pas de paràmetres

Pas per referència

Es passa **la referència** a memòria, per la qual cosa el cos de la funció treballa sobre el mateix objecte en memòria

```
void inc(int v)
```

```
{
```

v = 10

```
    v = v + v;
```

```
}
```

```
...
```

```
int a = 10;
```

```
inc(a);
```

a = 10

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Pas de paràmetres

Pas per referència

Es passa **la referència** a memòria, per la qual cosa el cos de la funció treballa sobre el mateix objecte en memòria

```
void inc(int v)
{
    v = v + v;
}

...
int a = 10;
inc(a);
```

v = 20

a = 20

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Pas de paràmetres

Pas per referència

Es passa **la referència** a memòria, per la qual cosa el cos de la funció treballa sobre el mateix objecte en memòria

```
void inc(int v)
{
    v = v + v;
}

...
int a = 10;
inc(a);
```

a = 20

- La variable *a* Sí se modifica porque se trabaja sobre la misma dirección de memoria.

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Pas de paràmetres

Pas per necessitat

- Quan es passen expressions, **no s'avaluen** fins que s'usen en el cos de la funció

```
void inc(int v)
{
    v = v + v;
}

...
int a = 10;
inc(a+1);
```

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Pas de paràmetres

Pas per necessitat

- Quan es passen expressions, **no s'avaluen** fins que s'usen en el cos de la funció

```
void inc(int v)
{
    v = v + v;
}

...
int a = 10;
inc(a+1);
```

a = 10

Pas de paràmetres

Pas per necessitat

- Quan es passen expressions, **no s'avaluen** fins que s'usen en el cos de la funció

```
void inc(int v)
{
    v = v + v;
}

...
int a = 10;
inc(a+1);
```

$a = 10$

En la crida:

S'usa una referència a una còpia local d'expressió sense avaluar, és a dir $v=a+1$.

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Pas de paràmetres

Pas per necessitat

- Quan es passen expressions, **no s'avaluen** fins que s'usen en el cos de la funció

```
void inc(int v)
```

```
{
```

```
    v = (a+1) + (a+1);
```

```
}
```

```
...
```

```
int a = 10;
```

```
inc(a+1);
```

$v = a + 1$

$a = 10$

Durant l'execució:

Si necessita l'expressió, l'avalua.

Pas de paràmetres

Pas per necessitat

- Quan es passen expressions, **no s'avaluen** fins que s'usen en el cos de la funció

```
void inc(int v)
{
    v = (a+1) + (a+1);
}

...
int a = 10;
inc(a+1);
```

$v = 22$

$a = 10$

Durant l'execució:

Si necessita l'expressió, l'avalua.

Pas de paràmetres

Pas per necessitat

- Quan es passen expressions, **no s'avaluen** fins que s'usen en el cos de la funció

```
void inc(int v)
{
    v = (a+1) + (a+1);
}

...
int a = 10;
inc(a+1);
```

a = 10

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Pas de paràmetres

Pas per necessitat

- Quan es passen expressions, **no s'avaluen** fins que s'usen en el cos de la funció

```
void inc(int v)
{
    . . .
    int a = 10;
```

- La variable `a` NO es modifica, ja que s'ha usat una còpia local.
- Normalment, gràcies a la *memoization*, `a+1` solament s'avalua una vegada.

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Pas de paràmetres

Algunes consideracions

- En pas per valor, si es passa una expressió, aquesta s'avalua per a copiar el valor resultant (a diferència del pas per necessitat)
- En pas per referència, si es passa una expressió també s'avalua i es passa el valor resultant.

Abast (àmbit) de les variables

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

- Una variable és un *nom* que s'utilitza per a accedir a una posició de memòria
- No tots els noms (de variables, funcions, constants, etc.) estan accessibles durant tota l'execució, encara que existisquen en el programa
- L'àmbit o abast d'un nom és la porció del codi on aquest nom és visible (el seu valor associat pot ser consultat/modificat).
- El moment en el qual es fa l'enllaç (l'associació) és el que es diu temps d'enllaçat.
 - Amb **abast estàtic**, es defineix en *temps de compilació*
 - Amb **abast dinàmic**, es defineix en *temps d'execució*

Abast de les variables

Exemple de càlcul del àmbit de les variables (1/2)

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

```

1 program ambits;
2 type
3   TArray : array [1..3]
4             of integer;
5 var
6   a: TArray;
7 procedure un;
8   procedure dos;
9     a : TArray;
10  begin { * dos * }
11    a[1] := 1;
12    a[2] := 2;
13    a[3] := 3;
14    canvia(1, 2);
15    writeln(a[1], ' ', a[2], ' ', a[3]);
16  end { * dos * };

```

```

17 procedure canvia(i, j:integer)
18   var aux : integer;
19   begin { * canvia * }
20     aux := a[i];
21     a[i] := a[j];
22     a[j] := aux;
23   end { * canvia * };
24 begin { * un * }
25   a[1] := 0;
26   a[2] := 0;
27   a[3] := 0;
28   dos;
29 end { * un * };
30 begin { * ambits * }
31   u;
32 end { * ambits * }

```


Abast de les variables

Exemple de càlcul del àmbit de les variables (1/2)

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

```

1 program ambits;
2 type
3   TArray : array [1..3]
4             of integer;
5 var
6   a: TArray;
7 procedure un;
8   procedure dos;
9     a : TArray;
10  begin { * dos * }
11    a[1] := 1;
12    a[2] := 2;
13    a[3] := 3;
14    canvia(1, 2);
15    writeln(a[1], ' ', a[2], ' ',
16            a[3]);
16  end { * dos * };

```

```

17 procedure canvia(i, j:integer)
18   var aux : integer;
19   begin { * canvia * }
20     aux := a[i];
21     a[i] := a[j];
22     a[j] := aux;
23   end { * canvia * };
24 begin { * un * }
25   a[1] := 0;
26   a[2] := 0;
27   a[3] := 0;
28   dos;
29 end { * un * };
30 begin { * ambits * }
31   u;
32 end { * ambits * }

```

Quins són els valors que emmagatzema l'array `a` al final de l'execució? Què s'imprimeix per pantalla en la sentència `writeln` del codi?

Abast de les variables

Exemple de càlcul del àmbit de les variables (2/2)

Considerant abast estàtic. . .

Quins són els valors que emmagatzema el `array a` al final de l'execució?
Què s'imprimeix per pantalla en la sentència `writeln` del codi?

En **temps de compilació** l'enllaç és:

- En el cos de la funció `un` (línies 25 a 27), la variable `a` està enllaçada amb la variable global de la línia 6 (`un` no té declaració de variables locals).
- En el cos de la funció `dos` (línies 11 a 15), la variable `a` està enllaçada amb la variable local `a` definida en la línia 9, ja que les variables locals amb el mateix nom que les globals oculten a aquestes últimes.
- En el cos de la funció `canvia` (línies 20 a 22), la variable `a` està enllaçada amb la variable global de la línia 6 (el procediment `canvia` està definit en l'àmbit de `un`, igual que `dos`).

Abast de les variables

Exemple de càlcul del àmbit de les variables (2/2)

Considerant abast estàtic. . .

Quins són els valors que emmagatzema el array `a` al final de l'execució?
 Què s'imprimeix per pantalla en la sentència `writeln` del codi?

Por lo tanto:

- En el cuerpo principal del programa (línea 31) se hace una llamada al procedimiento `uno`.
- Els valors del array global s'inicialitzen als valors 0, 0 i 0 (línies 25 a 27)
- Es diu a `dos`, que inicialitza un array local amb valors 1, 2 i 3 el que no modifica el array global (línies 11 a 13)
- La crida a `canvia` canvia els valors del array global, quedant 0, 0 i 0, la qual cosa no modifica el array local de `dos`.
- **S'imprimeix per pantalla els valors del array local (1, 2 i 3)**

Abast de les variables

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Considerant abast dinàmic...

¿Quins són els valors que emmagatzema el array `a` al final de l'execució?

¿Què s'imprimeix per pantalla en la sentència `writeln` del codi?

En **tiempo de ejecución** l'enllaç és:

- En el cos de la funció `un` (línies 25 a 27), la variable `a` està enllaçada amb la variable global de la línia 6 (`un` no té declaració de variables locals).
- En el cos de la funció `dos` (línies 11 a 15), la variable `a` està enllaçada amb la variable local `a` definida en la línia 9.
- En el cos de la funció `canvia` (línies 20 a 22), **com la crida a aquesta funció ocorre en l'àmbit de `dos` i `dos` té una variable local `a`, la variable `a` del cos de `canvia` s'enllaça amb la variable local de la línia 9.**

Abast de les variables

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Considerant abast dinàmic...

¿Quins són els valors que emmagatzema el array `a` al final de l'execució?

¿Què s'imprimeix per pantalla en la sentència `writeln` del codi?

Por lo tanto:

- En el cos principal del programa (línia 31) es fa una trucada al procediment `un`.
- Els valors del array global s'inicialitzen als valors 0, 0 i 0 (línies 25 a 27)
- Es crida a `dos`, que inicialitza un array local amb valors 1, 2 i 3, la qual cosa no modifica el array global (línies 11 a 13)
- La crida a `canvia` canvia els valors del array **local**, quedant 2, 1 i 3, la qual cosa no modifica el array global.
- **S'imprimeix per pantalla els valors del array local (2, 1 y 3).**

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Es refereix als diferents mètodes i operacions que s'encarreguen d'obtenir la **màxima utilitat de la memòria**, organitzant els processos i programes que s'executen en el sistema operatiu de manera tal que s'optimitze l'espai disponible.

Influeix en les decisions de disseny d'un llenguatge

A voltes els llenguatges contenen característiques o restriccions que solament poden explicar-se pel desig dels dissenyadors d'usar una tècnica o una altra de gestió de memòria

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Gestió de Memòria

Necessitat de gestionar la memòria

Elements amb requeriments d'emmagatzematge durant l'execució dels programes:

- Codi del **programa traduït**
- **Informació temporal** durant l'avaluació d'expressions i en el pas de paràmetres (e.g., en les crides a funcions els valors actuals tenen que avaluar-se i emmagatzemar-se fins a completar la llista de paràmetres)
- **Crides** a subprogrames i operacions de **tornada**
- **Buffers** per a les operacions d'entrada i eixida
- Operacions de **inserció i destrucció d'estructures de dades** en l'execució del programa (e.g., `new` en Java o `dispose` en Pascal)
- Operacions de **inserció i borrat de components** en estructures de dades (e.g., la funció `push` de Perl per a afegir un element a un array)

Gestió de Memòria

Tipus de gestió de memòria

Tipus d'assignació de l'emmagatzematge

Estàtic

Es calcula i assigna en temps de compilació

- Eficient però incompatible amb recursió o estructures de dades dinàmiques

Dinàmic

Es calcula i assigna en temps d'execució

- en pila
- en *un heap*

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Gestió de Memòria

Emmagatzematge estàtic

Emmagatzematge calculat en temps de compilació que roman fix durant l'execució del programa.

Se sol usar amb:

- **variables globals**
- **programa compilat** (instruccions en llenguatge màquina)
- **variables locals** a un subprograma
- **constants** numèriques i cadenes de caràcters
- **taules** produïdes pels compiladors i usades per a operacions d'ajuda en temps d'execució (e.g., comprovació dinàmica de tipus, depuració, ...).

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Gestió de Memòria

Emmagatzematge estàtic

Emmagatzematge calculat en temps de compilació que roman fix durant l'execució del programa.

Se sol usar amb:

- **variables globals**
- **programa compilat** (instruccions en llenguatge màquina)
- **variables locals** a un subprograma
- **constants** numèriques i cadenes de caràcters
- **taules** produïdes pels compiladors i usades per a operacions d'ajuda en temps d'execució (e.g., comprovació dinàmica de tipus, depuració, ...).

És eficient però incompatible amb recursió i amb estructures de dades, la grandària de les quals depèn de dades d'entrada o dades computades durant l'execució del programa

Gestió de Memòria

Emmagatzematg dinàmic: en pila

És la tècnica més simple **per a manejar els registres d'activació en les trucades a funcions/procediments** durant l'execució del programa (hi ha prou amb un punter al cim de la pila)

Emmagatzematge en pila

- a l'inici de l'execució s'assigna un bloc seqüencial en memòria com a espai d'emmagatzematge lliure,
- quan es requereix espai d'emmagatzematge (hi ha una crida), aquest es pren del bloc començant des del final de l'últim espai assignat (**secuencialment**)
- una vegada acabada la crida, l'espai **s'allibera en ordre invers** al que va ser assignat, per la qual cosa l'espai lliure sempre està en el cim de la pila

Gestió de Memòria

Emmagatzematge dinàmic: en *heap*

Un **heap** és una regió d'emmagatzematge en la qual els blocs de memòria s'assignen i alliberen en *moments arbitraris*

- L'emmagatzematge en *heap* és necessari quan el llenguatge permet estructures de dades (e.g., conjunts o llistes) la grandària de les quals pot canviar en temps d'execució.
- Els subblocs assignats poden ser del **mateix grandària sempre o de grandària variable**
- La desassignació pot ser
 - explícita (ex. C, C++, Pascal)
 - implícita (quan l'element assignat ja no és assolible per cap variable del programa)

Gestió de Memòria

Emmagatzematge dinàmic: en *heap*

Un **heap** és una regió d'emmagatzematge en la qual els blocs de memòria s'assignen i alliberen en *moments arbitraris*

- L'emmagatzematge en *heap* és necessari quan el llenguatge permet estructures de dades (e.g., conjunts o llistes) la grandària de les quals pot canviar en temps d'execució.
- Els subblocs assignats poden ser del **mateix grandària sempre o de grandària variable**
- La desassignació pot ser
 - explícita (ex. C, C++, Pascal)
 - implícita (quan l'element assignat ja no és assolible per cap variable del programa)
- **Garbage collector**: mecanisme del llenguatge que identifica els elements inassolibles i desassigna la memòria que ocupen, la qual passa a estar lliure

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Tema 1. Introducció (Part 2)

Llenguatges, Tecnologies i Paradigmes de Programació (LTP)

DSIC, ETSInf



Paradigmes de Programació

Factors d'èxit d'un LP

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

- **Potència expressiva**: per a generar codi clar, concís i fàcil de mantenir
- **Fàcil** d'aprendre
- **Portable** i amb garanties per a la seguretat
- Suportat per **múltiples plataformes** i eines de desenvolupament
- Suport **econòmic**
- Fàcil **migració** d'aplicacions escrites en altres llenguatges (C++ → Java)
- Múltiples **biblioteques** per a gran varietat d'aplicacions
- Disponibilitat de descàrrega de **codi obert** escrit en el llenguatge

Paradigmes de programació

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Definició de paradigma de programació

Model bàsic de disseny i desenvolupament de programes que proporciona un conjunt de mètodes i tècniques per a produir programes amb unes directrius específiques (estil i forma de plantejar la solució al problema)

Principals paradigmes:

- Imperatiu
- Declaratiu
 - funcional
 - lògic
- Orientat a objectes
- Concurrent

Existeixen també els anomenats paradigmes *emergents*

Paradigma imperatiu

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Describeix la programació com una seqüència d'instruccions o comandos que canvien l'estat del programa.

- Estableix el **com** procedir → **algorisme**
- El concepte bàsic és el **estat de la màquina**, el qual es defineix pels valors de les variables involucrades i que s'emmagatzemen en la memòria
- Les instruccions solen ser seqüencials i el programa consisteix en **construir la seqüència d'estats** de la màquina que condueix a la solució
- Aquest model està molt vinculat a l'arquitectura de la màquina convencional (*Von Neumann*)
- Programa estructurat en blocs i mòduls
- Eficient, difícil de modificar i verificar, amb **efectes laterals**

Paradigma Imperatiu

Exemple en Pascal

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Funció **length** en Pascal:

```
function length (l : list): integer
var
    b : boolean;
    aux : list;
begin
    b := is_empty(l);
    case b of
        true : length := 0;
        false : begin
                    aux := tail(l);
                    length := 1+length(aux);
                end;
    end;
end;
```

Paradigma Imperatiu

Característiques: Efectes laterals

Pot ocórrer que dues crides a funció amb els mateixos arguments donen resultats diferents

```

program proof;
var
    flag : boolean;
function f (x : integer) : integer;
begin
    flag := not flag;
    if flag then f := x else f := x+1;
end;
begin
    flag := false;
    write(f(1));
    write(f(1));
end
  
```

variable global

f cambia el valor de la variable global

Paradigma Imperatiu

Característiques: Efectes laterals

Pot ocórrer que dues crides a funció amb els mateixos arguments donen resultats diferents

```

program proof;
var
    flag : boolean;
function f (x : integer) : integer;
begin
    flag := not flag;
    if flag then f := x else f := x+1;
end;
begin
    flag := false;
    write(f(1));
    write(f(1));
end

```

Eixida del programa:

```

> proof
1
2

```

Programació imperativa

Característiques

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

- Posa l'èmfasi en el **com** resoldre un problema
- Les sentències dels programes s'executen en l'ordre en què estan escrites i dit **ordre d'execució** és crucial
- **Asignació destructiva** (el valor assignat a una variable destrueix el valor anterior d'aquesta variable) → efectes laterals que enfosqueixen el codi
- El **control** és responsabilitat del programador
- Més **complicat** del que sembla (així ho demostra la complexitat de les seues definicions semàntiques o la dificultat de les tècniques associades, e.g., de verificació formal)
- **Difícil de paral·lelitzar**
- Els programadors estan millor disposats a sacrificar les característiques avançades a canvi de poder obtenir major velocitat d'execució

Descriv les propietats de la solució cercada, deixant indeterminat l'algorisme (conjunt d'instruccions) usat per a trobar aqueixa solució

- Respon a la idea proposada per Kowalski

PROGRAMA = LÒGICA + CONTROL

- Lògica: es relaciona amb l'establiment del **Què**
- Control: es relaciona amb l'establiment del **Com**
- El programador se centra en **aspectes lògics de la solució** i deixa els aspectes de control al sistema
- Fàcil de verificar i modificar, concís i clar

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

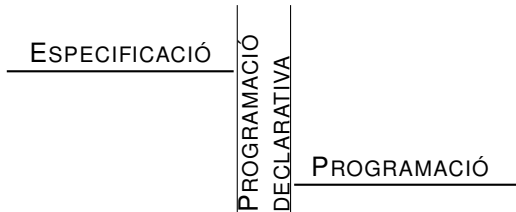
Basat en interacció

Bibliografia

Paradigma Declaratiu

Un programa declaratiu pot entendre's com **una especificació executable**.

Lleng. declaratiu = Llenguatge de **ESPECIFICACIÓ** (executable)
Llenguatge de **PROGRAMACIÓ** (alt nivell)



Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Paradigma Declaratiu

Especificació vs programació

Especificació: Definició de funció matemàtica

$$\text{fib}(0) = 1$$
$$\text{fib}(1) = 1$$
$$\text{fib}(n) = \text{fib}(n-1) + \text{fib}(n-2)$$

Paradigma Declaratiu

Especificació vs programació

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Especificació: Definició de funció matemàtica

$$\text{fib}(0) = 1$$

$$\text{fib}(1) = 1$$

$$\text{fib}(n) = \text{fib}(n-1) + \text{fib}(n-2)$$

Programa (dues versions):

Directament la especificació:

$$\text{fib}(0) = 1$$

$$\text{fib}(1) = 1$$

$$\text{fib}(n) = \text{fib}(n-1) + \text{fib}(n-2)$$

Variant optimitzada amb acumulador

$$\text{fib}(0) = 1$$

$$\text{fib}(1) = 1$$

$$\text{fib}(n) = \text{fib_aux}(1, 1, n)$$

$$\text{fib_aux}(x, y, 0) = x$$

$$\text{fib_aux}(x, y, n) = \text{fib_aux}(y, x+y, n-1)$$

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusion

Reflexió

Procediments i control de flux

Pas de paramètres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

00

Concurrent

Altres paradigmes

Basat en
interacció

Bibliografia

- **Paradigma funcional** (basat en λ el -càlcul)
 - definició d'estructures de dades i funcions que manipulen les estructures mitjançant equacions
 - **polimorfisme**
 - ordre superior
- **Paradigma lògic** (basat en la lògica de primer ordre)
 - definició de relacions mitjançant regles:

Si $C1$ i $C2$ i ... Cn , llavors A
escrit $A \leftarrow C1, C2, \dots Cn$
 - variables lògiques
 - indeterminisme

Paradigma declaratiu

Exemple en Haskell i Prolog

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

La funció `length` d'una llista:

En Haskell

```
data list a = [] | a:list a
```

```
length [] = 0
```

```
length (x:xs) = (length xs) + 1
```

En Prolog

```
length([],0).
```

```
length([X|Xs],N) :- length(Xs,M), N is M + 1.
```

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusion

Reflexió

Procediments i control de flux

Pas de paramètres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

00

Concurrent

Autres paradigmes

Basat en
interacción

Bibliografia

Programació declarativa

Caractéristiques

- Expressa **que** és la solució a un problema
- El **ordre** de les sentències i expressions **no te per què afectar a la semàntica** del programa
- Una **expressió denota un valor** independent del context (**transparencia referencial**)
- Nivell més alt de programació
 - semàntica més senzilla
 - control automàtic
 - més fàcil de paral·lelitzar
 - millor manteniment
 - major potència expressiva
 - menor grandària del codi
 - major productivitat
- Eficiència comparable a la de llenguatges com Java.
- La corba d'aprenentatge és més lenta quan s'aprèn a programar en un paradigma més convencional
- Les *impureses* de sistemes reals són difícils de modelar de manera declarativa

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Paradigma Declaratiu vs Paradigma imperatiu

Paradigma Imperatiu

PROGRAMA

INSTRUCCIONS

MODEL DE COMPUTACIÓ

VARIABLES

Transcripció d'un **algorisme**

Ordres a la **màquina**

Màquina de **estats**

Referències a **memòria**

Paradigma Declaratiu vs Paradigma imperatiu

Paradigma Declaratiu

LÒGICA com llenguatge de programació

PROGRAMA

Especificació d'un problema

INSTRUCCIONS

Fórmules lògiques

MODEL DE COMPUTACIÓ

Màquina de inferències

VARIABLES

Variables lògiques

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Paradigma Imperatiu vs Paradigma declaratiu

Un example

Què fa aquest programa imperatiu?

```

void f(int a[], int lo, hi){
    int h, l, p, t;

    if (lo<hi) {
        l = lo;
        h = hi;
        p = a[hi];
        do {
            while ((l<h)&&
                (a[l] <= p))
                l = l+1;
            while ((h>l)&&
                (a[h] >= p))
                h = h-1;
            if (l<h) {
                t = a[l];
                a[l] = a[h];
                a[h] = t;
            }
            a[hi] = a[l];
            a[l] = p;
            f(a, lo, l-1);
            f(a, l+1, hi);
        }
    }
}

```

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Paradigma Imperatiu vs Paradigma declaratiu

Un example

Què fa aquest programa declaratiu?

```
f :: Ord a => [a] -> [a]

f [] = []
f (p:xs) = (f lesser) ++ [p] ++ (f greater)
  where
    lesser = filter (< p) xs
    greater = filter (>= p) xs
```


Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Paradigma Imperatiu vs
Paradigma declaratiu

Un example

Què fa aquest programa declaratiu?

```
f :: Ord a => [a] -> [a]

f [] = []
f (p:xs) = (f lesser) ++ [p] ++ (f greater)
  where
    lesser = filter (< p) xs
    greater = filter (>= p) xs
```

- Sense assignació de variables
- Sense índex de vector
- Sense gestió de memòria

Paradigma orientat a objectes

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

oo

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Basat en la idea d'encapsular en objectes estat i operacions

- Objecte: estat + operacions
- Concepte de *classe*, *instància*, *subclasses* i herència
- Elements fonamentals:
 - abstracció
 - encapsulament
 - modularitat
 - jerarquia

Paradigma orientat a objectes

Exemple en Java

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

oo

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

La funció `length` d'una llista amb punt d'interès (PI):

```
interface ListWithIP<T> { // Llista amb PI
    void init(); // Col·loca el PI en el primer element
    void succ(); // Mou el PI al següent element
                  // (si existeix)
    boolean isLast(); // Comprova si el PI es troba
                     // en el final
}

abstract class myListWithPI<T> implements
    ListWithIP<T> {

    public int myLength(){
        int index = 0;
        for (init(); !isLast(); succ())
            index++;
        return index;
    }
}
```

Paradigma concurrent

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

- Els llenguatges de programació concurrents utilitzen per a programar la **execució simultània de múltiples tasques interactives**
- Les tasques poden consistir en un **conjunt de processos** creats per un únic programa

Accés concurrent en bases de dades, ús de recursos d'un sistema operatiu, etc.

- L'inici de la programació concurrent està en la invenció de la **interrupció** a la fi dels 50.
 - Interrupció: mecanisme maquinari que interromp el programa en execució i fa que la unitat de procés bifurque el control a una adreça donada, on resideix un altre programa que tractarà l'esdeveniment associat a la interrupció

Paradigma concurrent

Problemes associats a la concurrencia

- **Corrupció de les dades** compartides

Quan dos processos escriuen concurrentment en la pantalla pot produir-se una mescla incomprensible

- **Interbloquejos** entre processos que comparteixen recursos

A necessita dos recursos compartits (R1 i R2). Tracta d'obtenir els recursos en exclusiva (per a evitar corrupció de dades) sol·licitant R1 i després R2. Mentre, B sol·licita R2 i després R1. Cadascun obté un recurs, però cap pot obtenir el segon

- **Inanició** d'un procés que no aconsegueix un recurs donat.

Normalment el SO organitza una cua de processos per als recursos compartits en funció de la prioritat d'aquests processos. Dos processos amb alta prioritat podrien estar acaparant el recurs.

- **Indeterminisme** en l'ordre en el qual s'entrellacen les accions dels diferents processos.

Dificulta la depuració ja que els errors poden dependre d'aquest ordre

Paradigma concurrent

Conceptes propis: Primeres abstraccions (1/2)

- La manera primitiva de definir un llenguatge concurrent va consistir a afegir a un llenguatge seqüencial (Simula) primitives del SO per a la creació de processos (corutines)
 - Problema: SO nivell i falta de portabilitat
- Dijkstra va introduir (1965-71) les primeres abstraccions.
 - **Programa concurrent:** conjunt de processos seqüencials asíncrons que no fan suposicions sobre les velocitats relatives amb les quals progressen altres processos
 - Introdueix els **semàfors** com a mecanisme de sincronització

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Paradigma concurrent

Conceptes propis: Primeres abstraccions (2/2)

- Hoare introdueix la noció de **regió crítica** per a evitar interbloquejos
 - gestionar les regions crítiques era ineficient i poc modular
- En 1974 s'introdueix el concepte de **monitor** (inspirat en els TAD) per a encapsular els recursos compartits.
 - El primer llenguatge concurrent d'alt nivell amb monitors va ser Pascal concurrent (1975), després incorporat a Modula-2.
- Sorgeixen models, independents de l'arquitectura, que permeten l'anàlisi dels programes concurrents (CSP, CCS, π -càlcul, xarxes de petri, PVM)
 - aquests models influeixen en diferents llenguatges, per exemple CSP va influir en els canals de Occam i les crides remotes de ADA

Paradigma concurrent

Exemple en Java de definició de fils

Heretant de Thread

```
class MyThread extends Thread {
    public void run () {
        // cuerpo de la tarea a ejecutar
        // por el thread
    }
}
```

Implementant la interfície Runnable

```
class MyThread implements Runnable {
    public void run () {
        // cuerpo de la tarea
    }
}
```


Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Paradigma concurrent

Exemple en Java d'ús de fils

```
MyThread t1 = new MyThread();  
t1.setPriority(5)  
t1.start();  
System.out.println("Puc seguir amb les meues coses")  
// ...
```

- El mètode `start` inicia l'execució del fil (i invocarà al mètode `run`)
- L'assignació de prioritat és opcional
- El missatge es mostrarà per l'eixida independentment de l'execució del fil arrancat

Paradigma concurrent

Algunes consideracions de la concurrència en Java

- Java suporta la programació concurrent de forma **nativa** (no mitjançant biblioteques) gràcies a la classe `Thread`
- Un **fil** (*thread*) és un concepte similar al de procés. La diferència és que els fils sempre depenen d'un *programa pare* quant a recursos per a la seua execució.
 - Un procés pot mantenir el seu propi espai d'adreces i entorn d'execució
- El programador té funcions per a (per exemple) crear, arrancar, avortar, prioritzar, suspendre o reprendre fils
- La màquina virtual de java s'encarrega d'organitzar els fils, però és responsabilitat del programador evitar els problemes indesitjats de la concurrència (inanició, etc.)
- La comunicació és mitjançant **memòria compartida**. Com a ajuda, cada objecte té implícitament un bloqueig per a quan està sent utilitzat per un fil.

Programació paral·lela

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Objectiu:

Acceleració d'algorismes que consumeixen moltes hores de procés dividint el temps d'execució mitjançant l'ús de diversos processadors, **distribució de les dades** i **repartiment de la carrega**.

- Amb l'aparició dels primers microprocessadors (1975), els processos van passar a executar-se concurrentment **en diferents processadors**, per la qual cosa deixava de valdre el principi de disposar d'una memòria comuna.
 - sorgeixen noves construccions per a la comunicació entre processos, com el **pas de missatge entre processadors** *rendez-vous*.
- Primers llenguatges paral·lels: els seqüencials Fortran o C estesos amb biblioteques de pas de missatges dependents del fabricant.

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Programació paral·lela vs Programació concurrent

	PARAL·LELA	CONCURRENT
OBJECTIU	Eficiència: repartisc de càrrega	Interactivitat: diversos processos <i>simultàniament</i>
PROCESSADORS	solament es concep amb varis	és compatible amb un o amb varis
COMUNICACIÓ	pas de missatges	memòria compartida

Paradigma basat en interacció

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

- El paradigma tradicional segueix la idea de *programació com a càlcul en el model de Von Neumann*
 - un programa descriu la seqüència de passos necessaris per a produir l'eixida a partir d'una entrada
- En algunes àrees aquest model no s'adapta bé: robòtica, AI, aplicacions orientades a serveis, ...

Té més sentit la

Programació com interacció: les entrades es monitoritzen i les eixides són accions que es duen a terme dinàmicament (no hi ha un *resultat final*)

Paradigma basat en interacció

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Programa interactiu

És una comunitat d'entitats (agents, bases de dades, serveis de xarxa, etc.) que interactuen seguint certes **regles d'interacció**

- Les regles d'interacció poden estar restringides per interfícies, protocols i certes garanties del servei (temps de resposta, confidencialitat de dades, etc.)
- Instàncies del model de programació interactiva:
 - Programació conduïda por esdeveniments
 - Sistemes reactius
 - Sistemes encastats
 - Arquitectura client/servidor
 - Programari basat en agents
- Usat en aplicacions distribuïdes, disseny de GUI, programació web, disseny incremental de programes (es refinen parts d'un programa mentre està en execució)

Paradigma basat en interacció

Programació per esdeveniments

- El flux del programa està determinat per esdeveniments

Esdeveniments: senyals de sensors o, més comunament, accions d'usuari en la interfície, missatges des d'altres programes o processos, ...

- L'arquitectura típica d'una aplicació dirigida per/basada en esdeveniments (*event-driven/event-based*) consisteix en un bucle principal dividit en dues seccions independents:
 - 1 detecció o selecció d'esdeveniments (*event-detection*)
 - 2 maneig dels esdeveniments (*event-handling*)
- En el cas de *programari* encastrat, la primera secció resideix en *el maquinari* i es gestiona mitjançant interrupcions

Paradigma basat en interacció

Programació per esdeveniments

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

La programació per esdeveniments és una caracterització ortogonal a altres paradigmes:

- Es pot usar qualsevol llenguatge d'alt nivell per a escriure programes seguint l'estil *event-driven*.
- Pot o no ser orientada a objectes
- No implica programació concurrent
- Requisits:
 - poder detectar senyals, interrupcions al processador o esdeveniments de la GUI
 - poder gestionar una cua d'esdeveniments per a respondre als mateixos

Paradigma basat en interacció

Programació per esdeveniments

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

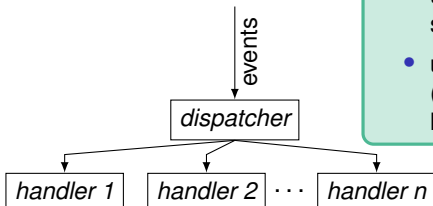
Altres paradigmes

Basat en interacció

Bibliografia

- Els *patrons de disseny* (en particular el patró *event-handler*) solen ser una ajuda que simplifica la tasca de programar aquest tipus d'aplicacions.

El patró *event-handler*



- un *dispatcher* que gestiona la seqüència d'esdeveniments
- un conjunt de manejadors (*handlers*) que implementen les accions de resposta

Paradigma basat en interacció

Programació per esdeveniments. Un exemple de *dispatcher*

bucle principal

```
do forever: // the event loop
  get an event from the input stream

  if event.type == EndOfEventStream
    quit // break out of event loop

  if event.type == ...:
    call the appropriate handler, passing it
    event information as an argument

  elseif event.type == ...:
    call the appropriate handler, passing it
    event information as an argument

  else: // unrecognized event type
    ignore the event, or raise an exception
```

eixida del bucle

selecció de handler

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

Paradigma basat en interacció

Consideracions finals

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

La programació basada en esdeveniments s'usa massivament en la programació de GUIs, principalment a causa que la majoria d'eines de desenvolupament comercials disposen de **assistents** per a la definició assistida d'aquest esquema

- Avantatge:
 - Simplifica la tasca del programador en proporcionar una implementació per defecte per al bucle principal i la gestió de la cua d'esdeveniments
- Desavantatges:
 - promou un model d'interacció excessivament simple
 - és difícil d'estendre
 - és propens a errors ja que dificulta la gestió de recursos compartits

Altres paradigmes emergents

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

- **BIO-COMPUTACIÓ:** Existeixen models de computació inspirats en la **biologia**
 - utilitzen conceptes i tècniques que s'empren en sistemes de la naturalesa com a base per a desenvolupar noves tècniques de programació
- **COMPUTACIÓ QUÀNTICA:** reemplaça els circuits clàssics per uns altres que utilitzen portes quàntiques (en compte de portes lògiques)

A quin paradigma pertanyen els llenguatges?

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

La majoria són multi-paradigma:

- **CoffeeScript** (2009): És un llenguatge orientat a objectes, basat en prototips, funcional i imperatiu. CoffeeScript es compila a Javascript.
- **Scala** (2003): Orientat a objectes, imperatiu i funcional (usat per Twitter juntament amb Ruby).
- **Erlang** (1986): funcional i concurrent (usat per HP, Amazon, Ericsson, Facebook, ...)
- **Python** (1989): funcional (llistes intensionales, abstracció lambda, fold, map) i orientat a objectes (herència múltiple)

Motivació

Conceptes

Tipus i sistemes de tipus

Polimorfisme

Sobrecàrrega

Coerció

Genericitat

Inclusió

Reflexió

Procediments i control de flux

Pas de paràmetres

Abast de les variables

Gestió de memòria

Paradigmes de programació

Imperatiu

Declaratiu

OO

Concurrent

Altres paradigmes

Basat en interacció

Bibliografia

- Cortazar, Francisco. *Lenguajes de programación y procesadores*. Editorial Cera, 2012.
- Peña, Ricardo. *De Euclides a Java: historia de algoritmos y lenguajes de programación*, Editorial Nivola, 2006.
- Pratt, T.W.; Zelkowitz, M.V. *Programming Languages: design and implementation*, Prentice-Hall, 2001 (versión de 1998 en castellano)
- Scott, M.L. *Programming Language Pragmatics*, Morgan Kaufmann Publishers, 2008 (versión revisada).
- Schildt, Herbert. *Java. The Complete Reference*. Eight Edition. The McGraw-Hill eds. 2011

Bibliografía

Aspectos de implementación

- “Programming Language Pragmatics”, M.L. Scott. (cap. 3)
- “Lenguajes de programación y procesadores”, Francisco Cortazar (cap. 1)
- “Programming Languages: design and implementation”, Pratt, T.W.; Zelkowitz, M.V. (cap. 9 y 10)