Tema 1. Introducció (Part 1)

Llenguatges, Tecnologies i Paradigmes de Programació (LTP)

DSIC, ETSInf

1. Motivació

2. Principals paradigmes de programació: imperatiu, funcional, lògic, orientat a objectes, concurrent
   - Paradigma imperatiu
   - Paradigma declaratiu
   - Paradigma orientat a objectes
   - Paradigma concurrent

3. Altres paradigmes: basat en interacció, emergents
   - Paradigma basat en interacció

4. Bibliografia

Objectius del tema

- 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 (genericidat, 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

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.

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

- S'estudia en un paradigma únic i amb un únic llenguatge
- Pot ser contraproduent. 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 A left() { return left; }
  public B right() { return right; }
  public String toString() {
    return "(" + left + "," +
                 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)]
```

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")]
```

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 afig 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 $x=10$.

3. Programació en termes de conceptes

- S'estudia un conjunt de conceptes semántics 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
(+) Estrategies
(+) Ordre superior

Llenguatge lògic

(+) No determinisme
(+) Variables lógiques
(+) Unificació

Llenguatge *kernel*

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

Llenguatge imperatiu

(+) Estados explícits
(+) Modularidat
(+) Components

Llenguatge OO

(+) Classes
(+) Herència

Llenguatge *dataflow*

(+) Concurrencia

Tema 1. Introducció (Part 2)

Llenguatges, Tecnologies i Paradigmes de Programació (LTP)

DSIC, ETSInf

Paradigmes de Programació

Factors d'èxit d'un LP

- Potencia expressiva: per a generar codi clar, concís i fàcil de mantenir
- Fácil 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++ $\rightarrow$ Java)
- Múltiples biblioteques per a gran varietat d'aplicacions
- Disponibilitat de descàrrega de codi obert escrit en el llenguatge

Paradigmes de programació

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

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

- Estableix el com procedir $\rightarrow$ 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 seqüencia 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

Example en Pascal

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

- 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'ejecució és crucial
- Asignació destructiva (el valor assignat a una variable destrueix el valor anterior d'aquesta variable) $\rightarrow$ 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

Paradigma Declaratiu

Descriu 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

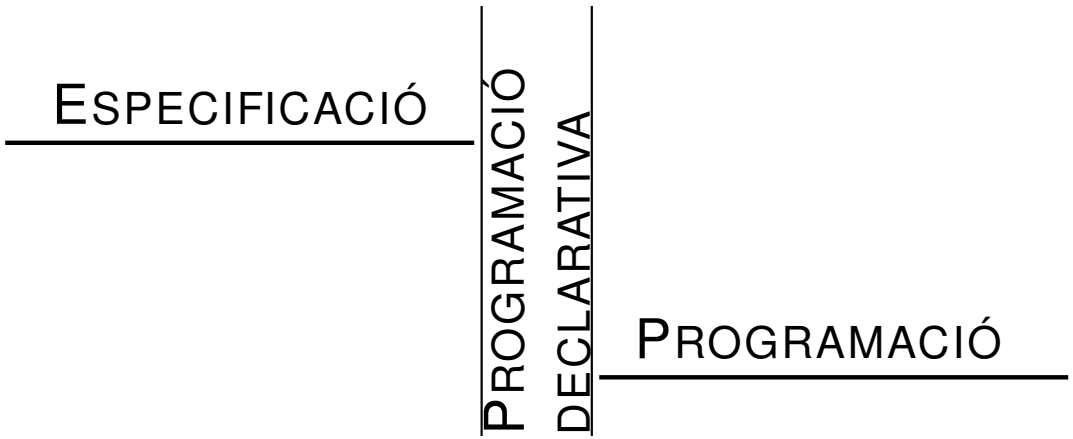
  PROGRAMA = LÒGICA + CONTROL

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

Paradigma Declaratiu

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

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



Paradigma Declaratiu

Especificació vs programació

Especificació: Definició de funció matemàtica

```
fib(0) = 1
fib(1) = 1
fib(n) = fib(n-1)+fib(n-2)
```

Paradigma Declaratiu

Especificació vs programació

Especificació: Definició de funció matemàtica

```
fib(0) = 1
fib(1) = 1
fib(n) = fib(n-1)+fib(n-2)
```

Programa (dues versions):

Directament la especificació:

```
fib(0) = 1
fib(1) = 1
fib(n) = fib(n-1)+
         fib(n-2)
```

Variant optimitzada amb acumulador

```
fib(0) = 1
fib(1) = 1
fib(n) = fib_aux(1,1,n)
fib_aux(x,y,0) = x
fib_aux(x,y,n) =
        fib_aux(y,x+y,n-1)
```

Paradigma Declaratiu

- **Paradigma funcional** (basat en λel -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 $A \leftarrow C1, C2, \ldots Cn$
  - variables lògiques
  - indeterminisme

Paradigma declaratiu

Example en Haskell i Prolog

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.
```

Programació declarativa

Característiques

- Expressa qué és la solució a un problema
- El ordre de les sentències i expressions no te per què afectar a la semàntica del programa
- Una expresió 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 saprén a programar en un paradigma més convencional
- Les *impureses* de sistemes reals són difícils de modelar de manera declarativa

Paradigma Declaratiu *vs* Paradigma imperatiu

Paradigma Imperatiu

PROGRAMA	Transcripció d'un algorisme
INSTRUCCIONS	Ordres a la màquina
MODEL DE COMPUTACIÓ	Màquina de estats
VARIABLES	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 inferencies
VARIABLES	Variables lògiques

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);
}
```

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
```

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

Paradigma orientat a objectes

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

- Objecte: estat + operacions
- Concepte de *classe*, *instància*, *subclasses* i herencia
- Elements fonamentals:
  - abstracció
  - encapsulamient
  - modularidat
  - jerarquia

Paradigma orientat a objetes

Example en Java

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

- 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 asociats 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

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

Example 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
      }
}
```

Paradigma concurrent

Example 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 concurrencia 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

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 depenents del fabricador.

Programació paral·lela *vs* Programació concurrent

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

Paradigma basat en interacció

- 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

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 conduida por esdeveníments
  - 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* encastat, la primera secció resideix en *el maquinari* i es gestiona mitjançant interrupcions

Paradigma basat en interacció

Programació per esdeveniments

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

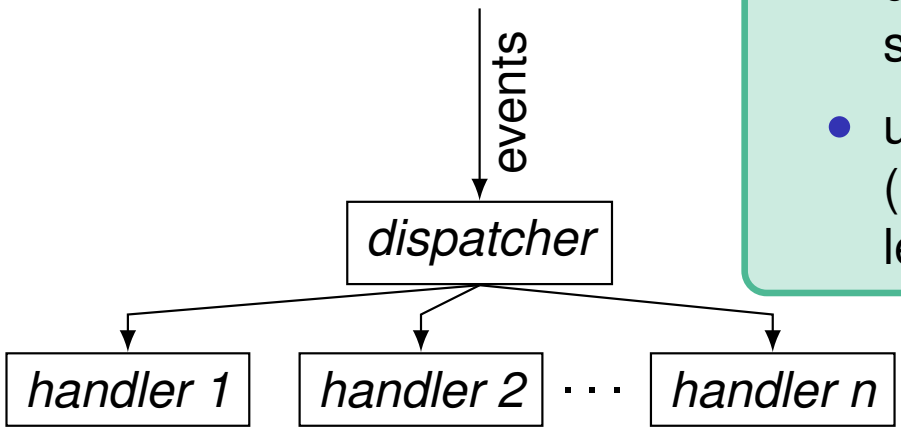
- Es pot usar qualsevol llenguatge d'alt nivell per a escriure programes seguint l'estil *event-driven*.
- Pot o no ser orientada a objectes
- No implica programació concurrent
- Requisits:
  - poder detectar senyals, interrupcions al processador o esdeveniments de la GUI
  - poder gestionar una cua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

Paradigma basat en interacció

Programació per esdeveniments

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

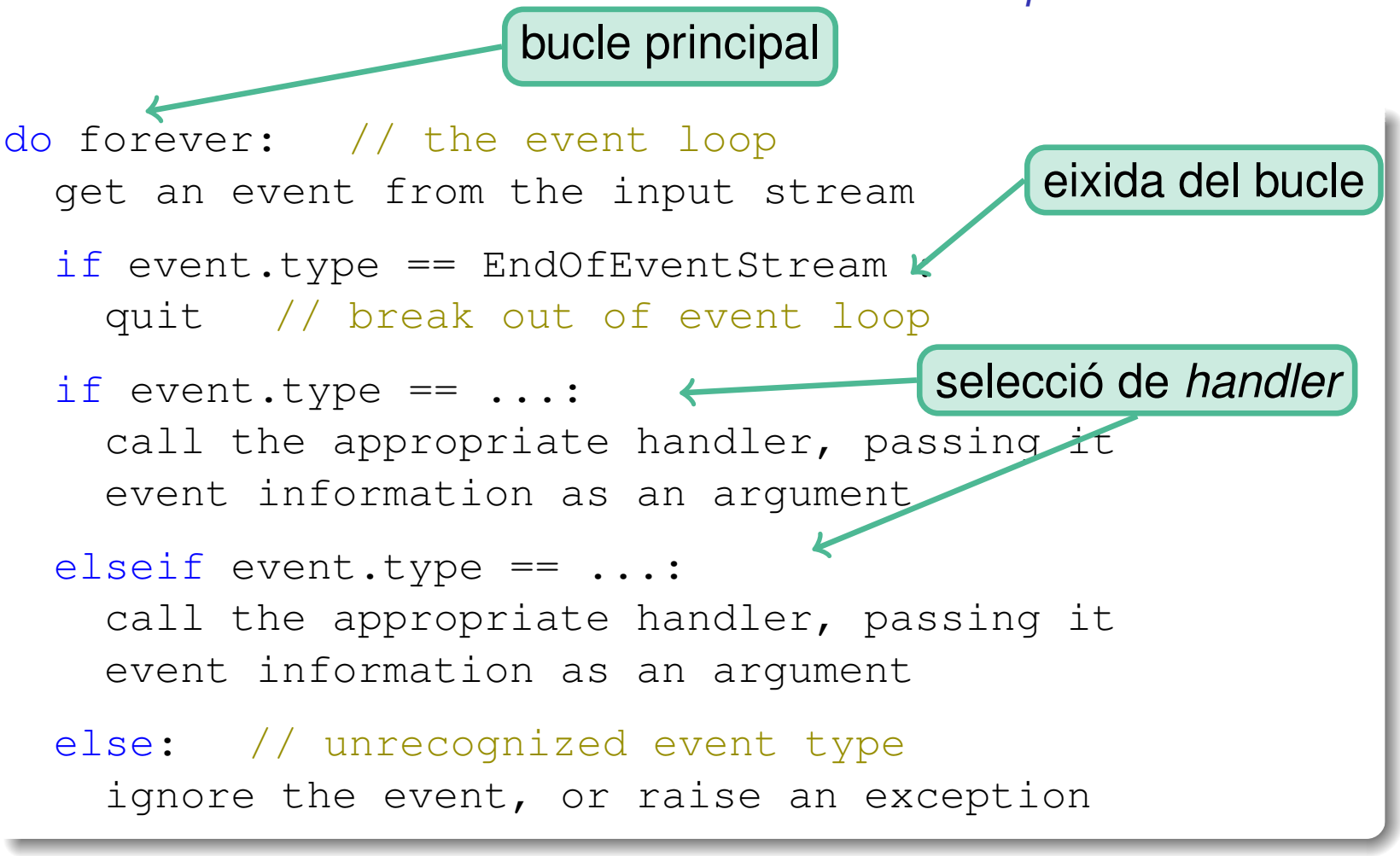
El patró *event-handler*



- 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 example de *dispatcher*



Paradigma basat en interacció

Consideracions finals

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

- 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?

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)

Bibliografia

Bàsica

- 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

Bibliografia

Aspectes d'implementació

- "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)