Motivación

Conceptos

Tipos y sistemas de tipos

Polimorfismo

Sobrecarga

Coerción

Genericidad

Inclusión

Reflexión

Procedimientos y control de flujo

Paso de parámetros

Alcance de las variables

Gestión de memoria

Paradigmas de programación

Imperativo

Declarativo

OO

Concurrente

Otros paradigmas

Basado en interacción

Bibliografía

Tema 1. Introducción (Parte II)

Lenguajes, Tecnologías y Paradigmas de Programación (LTP)

DSIC, ETSInf

DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

etsinf
Escuela Técnica Superior de Ingeniería Informática

Motivación

Conceptos

Tipos y sistemas de tipos

Polimorfismo

Sobrecarga

Coerción

Genericidad

Inclusión

Reflexión

Procedimientos y control de flujo

Paso de parámetros

Alcance de las variables

Gestión de memoria

Paradigmas de programación

Imperativo

Declarativo

OO

Concurrente

Otros paradigmas

Basado en interacción

Bibliografía

Paradigmas de Programación

Factores de éxito de un LP

- Potencia expresiva: para generar código claro, conciso y fácil de mantener
- Fácil de aprender
- Portable y con garantías para la seguridad
- Soportado por múltiples plataformas y herramientas de desarrollo
- Respaldo económico
- Fácil migración de aplicaciones escritas en otros lenguajes (C++ $\rightarrow$ Java)
- Múltiples bibliotecas para gran variedad de aplicaciones
- Disponibilidad de descarga de código abierto escrito en el lenguaje

Motivación

Conceptos

Tipos y sistemas de tipos

Polimorfismo

Sobrecarga

Coerción

Genericidad

Inclusión

Reflexión

Procedimientos y control de flujo

Paso de parámetros

Alcance de las variables

Gestión de memoria

Paradigmas de programación

Imperativo

Declarativo

OO

Concurrente

Otros paradigmas

Basado en interacción

Bibliografía

Paradigmas de programación

Definición de paradigma de programación

Modelo básico de diseño y desarrollo de programas que proporciona un conjunto de métodos y técnicas para producir programas con unas directrices específicas (estilo y forma de plantear la solución al problema)

Principales paradigmas:

- Imperativo
- Declarativo
  - funcional
  - lógico
- Orientado a objetos
- Concurrente

Existen también los llamados paradigmas *emergentes*

Motivación

Conceptos

Tipos y sistemas de tipos

Polimorfismo

Sobrecarga

Coerción

Genericidad

Inclusión

Reflexión

Procedimientos y control de flujo

Paso de parámetros

Alcance de las variables

Gestión de memoria

Paradigmas de programación

Imperativo

Declarativo

OO

Concurrente

Otros paradigmas

Basado en interacción

Bibliografía

Paradigma imperativo

Describe la programación como una secuencia de instrucciones o comandos que cambian el estado del programa.

- Establece el cómo proceder $\rightarrow$ algoritmo
- El concepto básico es el estado de la máquina, el cual se define por los valores de las variables involucradas y que se almacenan en la memoria
- Las instrucciones suelen ser secuenciales y el programa consiste en construir la secuencia de estados de la máquina que conduce a la solución
- Este modelo está muy vinculado a la arquitectura de la máquina convencional (*Von Neumann*)
- Programa estructurado en bloques y módulos
- Eficiente, difícil de modificar y verificar, con efectos laterales

Motivación

Conceptos

Tipos y sistemas de tipos

Polimorfismo

Sobrecarga

Coerción

Genericidad

Inclusión

Reflexión

Procedimientos y control de flujo

Paso de parámetros

Alcance de las variables

Gestión de memoria

Paradigmas de programación

Imperativo

Declarativo

OO

Concurrente

Otros paradigmas

Basado en interacción

Bibliografía

Paradigma imperativo

Ejemplo en Pascal

Función 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 imperativo

Características: Efectos laterales

Puede ocurrir que dos llamadas a función con los mismos argumentos den resultados diferentes

```
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

Motivación

Conceptos

Tipos y sistemas de tipos

Polimorfismo

Sobrecarga

Coerción

Genericidad

Inclusión

Reflexión

Procedimientos y control de flujo

Paso de parámetros

Alcance de las variables

Gestión de memoria

Paradigmas de programación

Imperativo

Declarativo

OO

Concurrente

Otros paradigmas

Basado en interacción

Bibliografía

Paradigma imperativo

Características: Efectos laterales

Puede ocurrir que dos llamadas a función con los mismos argumentos den resultados diferentes

```
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
```

Salida del programa:

```
> proof
1
2
```

Motivación

Conceptos

Tipos y sistemas de tipos

Polimorfismo

Sobrecarga

Coerción

Genericidad

Inclusión

Reflexión

Procedimientos y control de flujo

Paso de parámetros

Alcance de las variables

Gestión de memoria

Paradigmas de programación

Imperativo

Declarativo

OO

Concurrente

Otros paradigmas

Basado en interacción

Bibliografía

Programación imperativa

Características

- Pone el énfasis en el cómo resolver un problema
- Las sentencias de los programas se ejecutan en el orden en que están escritas y dicho orden de ejecución es crucial
- Asignación destructiva (el valor asignado a una variable destruye el valor anterior de dicha variable) $\rightarrow$ efectos laterales que oscurecen el código
- El control es responsabilidad del programador
- Más complejo de lo que parece (así lo demuestra la complejidad de sus definiciones semánticas o la dificultad de las técnicas asociadas, e.g., de verificación formal)
- Difícil de paralelizar
- Los programadores están mejor dispuestos a sacrificar las características avanzadas a cambio de poder obtener mayor velocidad de ejecución

Motivación

Conceptos

Tipos y sistemas de tipos

Polimorfismo

Sobrecarga

Coerción

Genericidad

Inclusión

Reflexión

Procedimientos y control de flujo

Paso de parámetros

Alcance de las variables

Gestión de memoria

Paradigmas de programación

Imperativo

Declarativo

OO

Concurrente

Otros paradigmas

Basado en interacción

Bibliografía

Paradigma declarativo

Describe las propiedades de la solución buscada, dejando indeterminado el algoritmo (conjunto de instrucciones) usado para encontrar esa solución

- Responde a la idea propuesta por Kowalski

  PROGRAMA = LÓGICA + CONTROL

  - Lógica: se relaciona con el establecimiento del Qué
  - Control: se relaciona con el establecimiento del Cómo
- El programador se centra en aspectos lógicos de la solución y deja los aspectos de control al sistema
- Fácil de verificar y modificar, conciso y claro

Motivación

Conceptos

Tipos y sistemas de tipos

Polimorfismo

Sobrecarga

Coerción

Genericidad

Inclusión

Reflexión

Procedimientos y control de flujo

Paso de parámetros

Alcance de las variables

Gestión de memoria

Paradigmas de programación

Imperativo

Declarativo

OO

Concurrente

Otros paradigmas

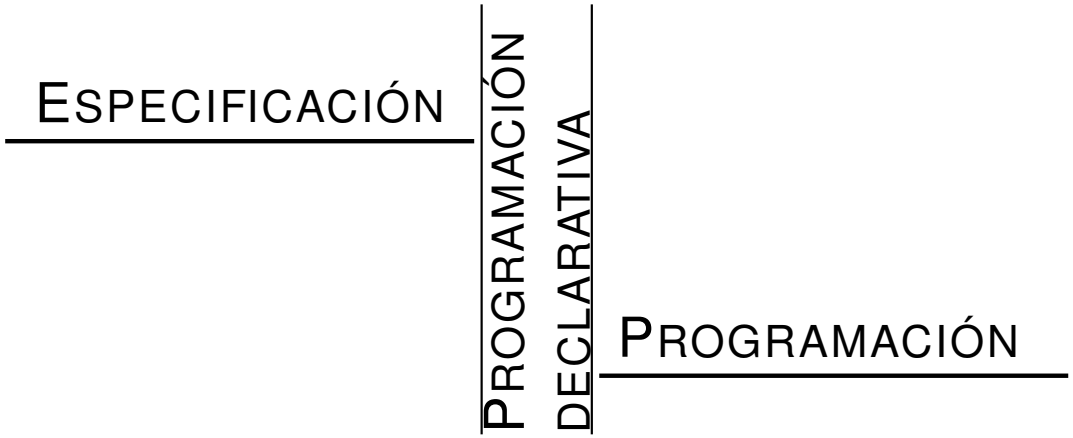
Basado en interacción

Bibliografía

Paradigma declarativo

Un programa declarativo puede entenderse como una **especificación ejecutable**.

Leng. declarativo = Lenguaje de ESPECIFICACIÓN (ejecutable)
Lenguaje de PROGRAMACIÓN (alto nivel)



Motivación

Conceptos

Tipos y sistemas de tipos

Polimorfismo

Sobrecarga

Coerción

Genericidad

Inclusión

Reflexión

Procedimientos y control de flujo

Paso de parámetros

Alcance de las variables

Gestión de memoria

Paradigmas de programación

Imperativo

Declarativo

OO

Concurrente

Otros paradigmas

Basado en interacción

Bibliografía

Paradigma declarativo

Especificación vs programación

Especificación: Definición de función matemática

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

Motivación

Conceptos

Tipos y sistemas de tipos

Polimorfismo

Sobrecarga

Coerción

Genericidad

Inclusión

Reflexión

Procedimientos y control de flujo

Paso de parámetros

Alcance de las variables

Gestión de memoria

Paradigmas de programación

Imperativo

Declarativo

OO

Concurrente

Otros paradigmas

Basado en interacción

Bibliografía

Paradigma declarativo

Especificación vs programación

Especificación: Definición de función matemática

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

Programa (dos versiones):

Directamente la especificación:

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

Variante optimizada con 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)
```

Motivación

Conceptos

Tipos y sistemas de tipos

Polimorfismo

Sobrecarga

Coerción

Genericidad

Inclusión

Reflexión

Procedimientos y control de flujo

Paso de parámetros

Alcance de las variables

Gestión de memoria

Paradigmas de programación

Imperativo

Declarativo

OO

Concurrente

Otros paradigmas

Basado en interacción

Bibliografía

Paradigma declarativo

- **Paradigma funcional** (basado en el λ-cálculo)
  - definición de estructuras de datos y funciones que manipulan las estructuras mediante ecuaciones
  - polimorfismo
  - orden superior
- **Paradigma lógico** (basado en la lógica de primer orden)
  - definición de relaciones mediante reglas:

    Si C1 y C2 y . . . Cn, entonces A
    escrito $A \leftarrow C1, C2, \ldots Cn$
  - variables lógicas
  - indeterminismo

Motivación

Conceptos

Tipos y sistemas de tipos

Polimorfismo

Sobrecarga

Coerción

Genericidad

Inclusión

Reflexión

Procedimientos y control de flujo

Paso de parámetros

Alcance de las variables

Gestión de memoria

Paradigmas de programación

Imperativo

Declarativo

OO

Concurrente

Otros paradigmas

Basado en interacción

Bibliografía

Paradigma declarativo

Ejemplo en Haskell y Prolog

La función `length` de una lista:

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.
```

Motivación

Conceptos

Tipos y sistemas de tipos

Polimorfismo

Sobrecarga

Coerción

Genericidad

Inclusión

Reflexión

Procedimientos y control de flujo

Paso de parámetros

Alcance de las variables

Gestión de memoria

Paradigmas de programación

Imperativo

Declarativo

OO

Concurrente

Otros paradigmas

Basado en interacción

Bibliografía

Programación declarativa

Características

- Expresa qué es la solución a un problema
- El orden de las sentencias y expresiones no tiene por qué afectar a la semántica del programa
- Una expresión denota un valor independiente del contexto (transparencia referencial)
- Nivel más alto de programación
  - semántica más sencilla
  - control automático
  - más fácil de paralelizar
  - mejor mantenimiento
  - mayor potencia expresiva
  - menor tamaño del código
  - mayor productividad
- Eficiencia comparable a la de lenguajes como Java.
- La curva de aprendizaje es más lenta cuando se aprendió a programar en un paradigma más convencional
- Las *impurezas* de sistemas reales son difíciles de modelar de manera declarativa

Motivación

Conceptos

Tipos y sistemas de tipos

Polimorfismo

Sobrecarga

Coerción

Genericidad

Inclusión

Reflexión

Procedimientos y control de flujo

Paso de parámetros

Alcance de las variables

Gestión de memoria

Paradigmas de programación

Imperativo

Declarativo

OO

Concurrente

Otros paradigmas

Basado en interacción

Bibliografía

Paradigma declarativo *vs* Paradigma imperativo

Paradigma imperativo

PROGRAMA	Transcripción de un algoritmo
INSTRUCCIONES	Órdenes a la máquina
MODELO DE COMPUTACIÓN	Máquina de estados
VARIABLES	Referencias a memoria

Motivación

Conceptos

Tipos y sistemas de tipos

Polimorfismo

Sobrecarga

Coerción

Genericidad

Inclusión

Reflexión

Procedimientos y control de flujo

Paso de parámetros

Alcance de las variables

Gestión de memoria

Paradigmas de programación

Imperativo

Declarativo

OO

Concurrente

Otros paradigmas

Basado en interacción

Bibliografía

Paradigma declarativo *vs* Paradigma imperativo

Paradigma declarativo

LÓGICA
como lenguaje de programación

PROGRAMA	Especificación de un problema
INSTRUCCIONES	Fórmulas lógicas
MODELO DE COMPUTACIÓN	Máquina de inferencias
VARIABLES	Variables lógicas

Motivación

Conceptos

Tipos y sistemas de tipos

Polimorfismo

Sobrecarga

Coerción

Genericidad

Inclusión

Reflexión

Procedimientos y control de flujo

Paso de parámetros

Alcance de las variables

Gestión de memoria

Paradigmas de programación

Imperativo

Declarativo

OO

Concurrente

Otros paradigmas

Basado en interacción

Bibliografía

Paradigma imperativo *vs* Paradigma declarativo

Un ejemplo

¿Qué hace este programa imperativo?

```
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);
}
```

Motivación

Conceptos

Tipos y sistemas de tipos

Polimorfismo

Sobrecarga

Coerción

Genericidad

Inclusión

Reflexión

Procedimientos y control de flujo

Paso de parámetros

Alcance de las variables

Gestión de memoria

Paradigmas de programación

Imperativo

Declarativo

OO

Concurrente

Otros paradigmas

Basado en interacción

Bibliografía

Paradigma imperativo *vs* Paradigma declarativo

Un ejemplo

¿Qué hace este programa declarativo?

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

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

Motivación

Conceptos

Tipos y sistemas de tipos

Polimorfismo

Sobrecarga

Coerción

Genericidad

Inclusión

Reflexión

Procedimientos y control de flujo

Paso de parámetros

Alcance de las variables

Gestión de memoria

Paradigmas de programación

Imperativo

Declarativo

OO

Concurrente

Otros paradigmas

Basado en interacción

Bibliografía

Paradigma imperativo *vs* Paradigma declarativo

Un ejemplo

¿Qué hace este programa declarativo?

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

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

- Sin asignación de variables
- Sin índices de vector
- Sin gestión de memoria

Motivación

Conceptos

Tipos y sistemas de tipos

Polimorfismo

Sobrecarga

Coerción

Genericidad

Inclusión

Reflexión

Procedimientos y control de flujo

Paso de parámetros

Alcance de las variables

Gestión de memoria

Paradigmas de programación

Imperativo

Declarativo

oo

Concurrente

Otros paradigmas

Basado en interacción

Bibliografía

Paradigma orientado a objetos

Basado en la idea de encapsular en objetos estado y operaciones

- Objeto: estado + operaciones
- Concepto de *clase*, *instancia*, *subclases* y herencia
- Elementos fundamentales:
  - abstracción
  - encapsulamiento
  - modularidad
  - jerarquía

Motivación

Conceptos

Tipos y sistemas de tipos

Polimorfismo

Sobrecarga

Coerción

Genericidad

Inclusión

Reflexión

Procedimientos y control de flujo

Paso de parámetros

Alcance de las variables

Gestión de memoria

Paradigmas de programación

Imperativo

Declarativo

OO

Concurrente

Otros paradigmas

Basado en interacción

Bibliografía

Paradigma orientado a objetos

Ejemplo en Java

La clase *Circle* es una abstracción de la noción de *círculo*:

```
public class Circle {      // class name
  double radius;           // variables (state)
  String color;

  double getRadius() {...} // methods (operations)
  double getArea() {...}
}
```

A partir de una clase se definen instancias que representan los objetos concretos (círculos con un radio y color específicos)

```
Circle c1, c2;
c1 = new Circle(2.0, "blue");
c2 = new Circle(3.0, "red");

Circle c3 = new Circle(1.5, "red");
```

Motivación

Conceptos

Tipos y sistemas de tipos

Polimorfismo

Sobrecarga

Coerción

Genericidad

Inclusión

Reflexión

Procedimientos y control de flujo

Paso de parámetros

Alcance de las variables

Gestión de memoria

Paradigmas de programación

Imperativo

Declarativo

OO

Concurrente

Otros paradigmas

Basado en interacción

Bibliografía

Paradigma concurrente

- Los lenguajes de programación concurrentes utilizan para programar la ejecución *simultánea* de múltiples tareas
- Las tareas pueden consistir en un conjunto de procesos creados por un único programa

> Acceso concurrente en bases de datos, uso de recursos de un sistema operativo, etc.

- El inicio de la programación concurrente está en la invención de la interrupción a finales de los 50.
  - Interrupción: mecanismo hardware que interrumpe el programa en ejecución y hace que la unidad de proceso bifurque el control a una dirección dada, donde reside otro programa que tratará el evento asociado a la interrupción

Motivación

Conceptos

Tipos y sistemas de tipos

Polimorfismo

Sobrecarga

Coerción

Genericidad

Inclusión

Reflexión

Procedimientos y control de flujo

Paso de parámetros

Alcance de las variables

Gestión de memoria

Paradigmas de programación

Imperativo

Declarativo

OO

Concurrente

Otros paradigmas

Basado en interacción

Bibliografía

Paradigma concurrente

Problemas asociados a la concurrencia

- Corrupción de los datos compartidos

  Cuando dos procesos escriben concurrentemente en la pantalla puede producirse una mezcla incomprensible

- Interbloqueos entre procesos que comparten recursos

  A necesita dos recursos compartidos (R1 y R2). Trata de obtener los recursos en exclusiva (para evitar corrupción de datos) solicitando R1 y luego R2. Mientras, B solicita R2 y luego R1. Cada uno obtiene un recurso, pero ninguno puede obtener el segundo

- Inanición de un proceso que no consigue un recurso dado.

  Normalmente el SO organiza una cola de procesos para los recursos compartidos en función de la prioridad de dichos procesos. Dos procesos con alta prioridad podrían estar acaparando el recurso.

- Indeterminismo en el orden en el que se entrelazan las acciones de los distintos procesos.

  Dificulta la depuración ya que los errores pueden depender de dicho orden

Motivación

Conceptos

Tipos y sistemas de tipos

Polimorfismo

Sobrecarga

Coerción

Genericidad

Inclusión

Reflexión

Procedimientos y control de flujo

Paso de parámetros

Alcance de las variables

Gestión de memoria

Paradigmas de programación

Imperativo

Declarativo

OO

Concurrente

Otros paradigmas

Basado en interacción

Bibliografía

Paradigma concurrente

Conceptos propios: Primeras abstracciones (1/2)

- La manera primitiva de definir un lenguaje concurrente consistió en añadir a un lenguaje secuencial (Simula) primitivas del SO para la creación de procesos (corutinas)
  - Problema: bajo nivel y falta de portabilidad
- Dijkstra introdujo (1965-71) las primeras abstracciones.
  - **Programa concurrente:** conjunto de procesos secuenciales asíncronos que no hacen suposiciones sobre las velocidades relativas con las que progresan otros procesos
  - Introduce los semáforos como mecanismo de sincronización

Motivación

Conceptos

Tipos y sistemas de tipos

Polimorfismo

Sobrecarga

Coerción

Genericidad

Inclusión

Reflexión

Procedimientos y control de flujo

Paso de parámetros

Alcance de las variables

Gestión de memoria

Paradigmas de programación

Imperativo

Declarativo

OO

Concurrente

Otros paradigmas

Basado en interacción

Bibliografía

Paradigma concurrente

Conceptos propios: Primeras abstracciones (2/2)

- Hoare introduce la noción de región crítica para evitar interbloqueos
  - gestionar las regiones críticas era ineficiente y poco modular
- En 1974 se introduce el concepto de monitor (inspirado en los TAD) para encapsular los recursos compartidos.
  - El primer lenguaje concurrente de alto nivel con monitores fue Pascal concurrente (1975), después incorporado a Modula-2.
- Surgen modelos, independientes de la arquitectura, que permiten el análisis de los programas concurrentes (CSP, CCS, π-cálculo, redes de petri, PVM)
  - estos modelos influyen en distintos lenguajes, por ejemplo CSP influyó en los canales de Occam y las llamadas remotas de ADA

Motivación

Conceptos

Tipos y sistemas de tipos

Polimorfismo

Sobrecarga

Coerción

Genericidad

Inclusión

Reflexión

Procedimientos y control de flujo

Paso de parámetros

Alcance de las variables

Gestión de memoria

Paradigmas de programación

Imperativo

Declarativo

OO

Concurrente

Otros paradigmas

Basado en interacción

Bibliografía

Paradigma concurrente

Ejemplo en Java de definición de hilos (1/2)

En Java existen dos formas de crear hilos:

- usando herencia (`extends`)
- usando interfaces (`implements`)

Usando herencia

Se define la clase `MyThread` heredando de la clase Java `Thread`

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

Motivación

Conceptos

Tipos y sistemas de tipos

Polimorfismo

Sobrecarga

Coerción

Genericidad

Inclusión

Reflexión

Procedimientos y control de flujo

Paso de parámetros

Alcance de las variables

Gestión de memoria

Paradigmas de programación

Imperativo

Declarativo

OO

Concurrente

Otros paradigmas

Basado en interacción

Bibliografía

Paradigma concurrente

Ejemplo en Java de uso de hilos (2/2)

Se crea y usa una instancia de la clase `MyThread`

```
MyThread t1 = new MyThread();
t1.setPriority(5)
t1.start();
System.out.println("Puedo seguir con mis cosas");
// ...
```

- El método `start` inicia la ejecución del hilo (e invocará al método `run`)
- La asignación de prioridad es opcional (entero entre 1 y 10, siendo 10 la mayor prioridad)
- El mensaje se mostrará por la salida independientemente de la ejecución del hilo arrancado

Motivación

Conceptos

Tipos y sistemas de tipos

Polimorfismo

Sobrecarga

Coerción

Genericidad

Inclusión

Reflexión

Procedimientos y control de flujo

Paso de parámetros

Alcance de las variables

Gestión de memoria

Paradigmas de programación

Imperativo

Declarativo

OO

Concurrente

Otros paradigmas

Basado en interacción

Bibliografía

Paradigma concurrente

Algunas consideraciones de la concurrencia en Java

- Java soporta la programación concurrente de forma nativa (no mediante bibliotecas) gracias a la clase `Thread`
- Un hilo (*thread*) es un concepto similar al de proceso. La diferencia es que los hilos siempre dependen de un *programa padre* en cuanto a recursos para su ejecución.
  - Un proceso puede mantener su propio espacio de direcciones y entorno de ejecución
- El programador tiene funciones para (por ejemplo) crear, arrancar, abortar, priorizar, suspender o reanudar hilos
- La máquina virtual de java se encarga de organizar los hilos, pero es responsabilidad del programador evitar los problemas indeseados de la concurrencia (inanición, etc.)
- La comunicación es mediante memoria compartida. Como ayuda, cada objeto tiene implícitamente un bloqueo para cuando está siendo utilizado por un hilo.

Motivación

Conceptos

Tipos y sistemas de tipos

Polimorfismo

Sobrecarga

Coerción

Genericidad

Inclusión

Reflexión

Procedimientos y control de flujo

Paso de parámetros

Alcance de las variables

Gestión de memoria

Paradigmas de programación

Imperativo

Declarativo

OO

Concurrente

Otros paradigmas

Basado en interacción

Bibliografía

Programación paralela

Objetivo:

Aceleración de algoritmos que consumen muchas horas de proceso dividiendo el tiempo de ejecución mediante el uso de varios procesadores, distribución de los datos y reparto de la carga.

- Con la aparición de los primeros microprocesadores (1975), los procesos pasaron a ejecutarse concurrentemente en distintos procesadores, por lo que dejaba de valer el principio de disponer de una memoria común.
  - surgen nuevas construcciones para la comunicación entre procesos, como el paso de mensaje entre procesadores *rendez-vous*.
- Primeros lenguajes paralelos: los secuenciales Fortran o C extendidos con bibliotecas de paso de mensajes dependientes del fabricante.

Motivación

Conceptos

Tipos y sistemas de tipos

Polimorfismo

Sobrecarga

Coerción

Genericidad

Inclusión

Reflexión

Procedimientos y control de flujo

Paso de parámetros

Alcance de las variables

Gestión de memoria

Paradigmas de programación

Imperativo

Declarativo

OO

Concurrente

Otros paradigmas

Basado en interacción

Bibliografía

Programación paralela *vs* Programación concurrente

	PARALELA	CONCURRENTE
OBJETIVO	Eficiencia: reparto de carga	Varios procesos procesos interactúan simultáneamente
PROCESADORES	solo se concibe con varios	es compatible con uno
COMUNICACIÓN	paso de mensajes y/o memoria compartida	

Motivación

Conceptos

Tipos y sistemas de tipos

Polimorfismo

Sobrecarga

Coerción

Genericidad

Inclusión

Reflexión

Procedimientos y control de flujo

Paso de parámetros

Alcance de las variables

Gestión de memoria

Paradigmas de programación

Imperativo

Declarativo

OO

Concurrente

Otros paradigmas

Basado en interacción

Bibliografía

Paradigma basado en interacción

- El paradigma tradicional sigue la idea de *programación como cálculo en el modelo de Von Neumann*
  - un programa describe la secuencia de pasos necesarios para producir la salida a partir de una entrada
- En algunas áreas este modelo no se adapta bien: robótica, AI, aplicaciones orientadas a servicios, . . .

  Tiene más sentido la

  Programación como interacción: las entradas se monitorizan y las salidas son acciones que se llevan a cabo dinámicamente (no hay un *resultado final*)

Motivación

Conceptos

Tipos y sistemas de tipos

Polimorfismo

Sobrecarga

Coerción

Genericidad

Inclusión

Reflexión

Procedimientos y control de flujo

Paso de parámetros

Alcance de las variables

Gestión de memoria

Paradigmas de programación

Imperativo

Declarativo

OO

Concurrente

Otros paradigmas

Basado en interacción

Bibliografía

Paradigma basado en interacción

Programa interactivo

Es una comunidad de entidades (agentes, bases de datos, servicios de red, etc.) que interactúan siguiendo ciertas reglas de interacción

- Las reglas de interacción pueden estar restringidas por interfaces, protocolos y ciertas garantías del servicio (tiempo de respuesta, confidencialidad de datos, etc.)
- Instancias del modelo de programación interactiva:
  - Programación conducida por eventos
  - Sistemas reactivos
  - Sistemas empotrados
  - Arquitectura cliente/servidor
  - Software basado en agentes
- Usado en aplicaciones distribuidas, diseño de GUI, programación web, diseño incremental de programas (se refinan partes de un programa mientras está en ejecución)

Motivación

Conceptos

Tipos y sistemas de tipos

Polimorfismo

Sobrecarga

Coerción

Genericidad

Inclusión

Reflexión

Procedimientos y control de flujo

Paso de parámetros

Alcance de las variables

Gestión de memoria

Paradigmas de programación

Imperativo

Declarativo

OO

Concurrente

Otros paradigmas

Basado en interacción

Bibliografía

Paradigma basado en interacción

Programación por eventos

- El flujo del programa está determinado por eventos

> Eventos: señales de sensores o, más comúnmente, acciones de usuario en la interfaz, mensajes desde otros programas o procesos, ...

- La arquitectura típica de una aplicación dirigida por/basada en eventos (*event-driven*/*event-based*) consiste en un bucle principal dividido en dos secciones independientes:
  1. detección o selección de eventos (*event-detection*)
  2. manejo de los eventos (*event-handling*)
- En el caso de *software* empotrado, la primera sección reside en el *hardware* y se gestiona mediante interrupciones

Motivación

Conceptos

Tipos y sistemas de tipos

Polimorfismo

Sobrecarga

Coerción

Genericidad

Inclusión

Reflexión

Procedimientos y control de flujo

Paso de parámetros

Alcance de las variables

Gestión de memoria

Paradigmas de programación

Imperativo

Declarativo

OO

Concurrente

Otros paradigmas

Basado en interacción

Bibliografía

Paradigma basado en interacción

Programación por eventos

La programación por eventos es una caracterización ortogonal a otros paradigmas:

- Se puede usar cualquier lenguaje de alto nivel para escribir programas siguiendo el estilo *event-driven*.
- Puede o no ser orientada a objetos
- No implica programación concurrente
- Requisitos:
  - poder detectar señales, interrupciones al procesador o eventos de la GUI
  - poder gestionar una cola de eventos para responder a los mismos

Motivación

Conceptos

Tipos y sistemas de tipos

Polimorfismo

Sobrecarga

Coerción

Genericidad

Inclusión

Reflexión

Procedimientos y control de flujo

Paso de parámetros

Alcance de las variables

Gestión de memoria

Paradigmas de programación

Imperativo

Declarativo

OO

Concurrente

Otros paradigmas

Basado en interacción

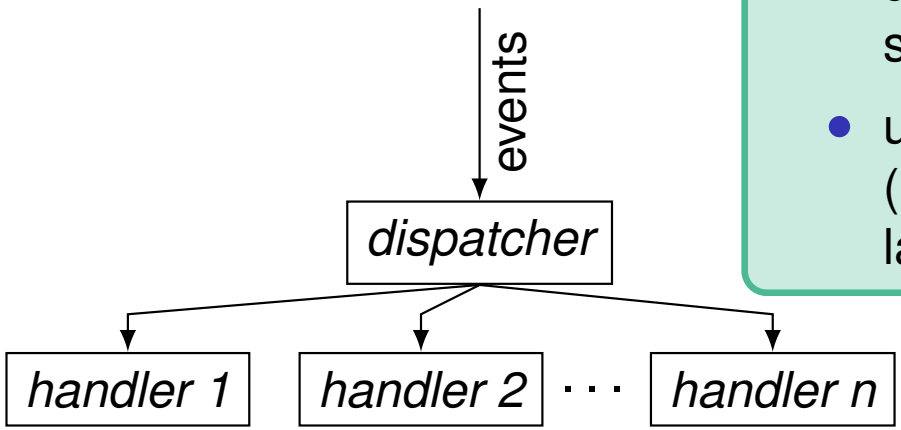
Bibliografía

Paradigma basado en interacción

Programación por eventos

- Los *patrones de diseño* (en particular el patrón *event-handler* suelen ser una ayuda que simplifica la tarea de programar este tipo de aplicaciones.

El patrón *event-handler*



- un *dispatcher* que gestiona la secuencia de eventos
- un conjunto de manejadores (*handlers*) que implementan las acciones de respuesta

Motivación

Conceptos

Tipos y sistemas de tipos

Polimorfismo

Sobrecarga

Coerción

Genericidad

Inclusión

Reflexión

Procedimientos y control de flujo

Paso de parámetros

Alcance de las variables

Gestión de memoria

Paradigmas de programación

Imperativo

Declarativo

OO

Concurrente

Otros paradigmas

Basado en interacción

Bibliografía

Paradigma basado en interacción

Programación por eventos. Un ejemplo de *dispatcher*

bucle principal

salida del bucle

selección de *handler*

```
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
```

Motivación

Conceptos

Tipos y sistemas de tipos

Polimorfismo

Sobrecarga

Coerción

Genericidad

Inclusión

Reflexión

Procedimientos y control de flujo

Paso de parámetros

Alcance de las variables

Gestión de memoria

Paradigmas de programación

Imperativo

Declarativo

OO

Concurrente

Otros paradigmas

Basado en interacción

Bibliografía

Paradigma basado en interacción

Consideraciones finales

La programación basada en eventos se usa masivamente en la programación de GUIs, principalmente debido a que la mayoría de herramientas de desarrollo comerciales disponen de asistentes para la definición asistida de este esquema

- Ventaja:
  - Simplifica la tarea del programador al proporcionar una implementación por defecto para el bucle principal y la gestión de la cola de eventos
- Desventajas:
  - promueve un modelo de interacción excesivamente simple
  - es difícil de extender
  - es propenso a errores ya que dificulta la gestión de recursos compartidos

Motivación

Conceptos

Tipos y sistemas de tipos

Polimorfismo

Sobrecarga

Coerción

Genericidad

Inclusión

Reflexión

Procedimientos y control de flujo

Paso de parámetros

Alcance de las variables

Gestión de memoria

Paradigmas de programación

Imperativo

Declarativo

OO

Concurrente

Otros paradigmas

Basado en interacción

Bibliografía

Otros paradigmas emergentes

- BIO-COMPUTACIÓN: Existen modelos de computación inspirados en la biología
  - utilizan conceptos y técnicas que se emplean en sistemas de la naturaleza como base para desarrollar nuevas técnicas de programación
- COMPUTACIÓN CUÁNTICA: reemplaza los circuitos clásicos por otros que utilizan puertas cuánticas (en vez de puertas lógicas)

Motivación

Conceptos

Tipos y sistemas de tipos

Polimorfismo

Sobrecarga

Coerción

Genericidad

Inclusión

Reflexión

Procedimientos y control de flujo

Paso de parámetros

Alcance de las variables

Gestión de memoria

Paradigmas de programación

Imperativo

Declarativo

OO

Concurrente

Otros paradigmas

Basado en interacción

Bibliografía

¿A qué paradigma pertenecen los lenguajes?

La mayoría son multi-paradigma:

- **CoffeeScript** (2009): Es un lenguaje orientado a objetos, basado en prototipos, funcional e imperativo. CoffeeScript se compila a JavaScript.
- **Scala** (2003): Orientado a objeros, imperativo y funcional (usado por Twitter junto con Ruby).
- **Erlang** (1986): funcional y concurrente (usado por HP, Amazon, Ericsson, Facebook, . . .)
- **Python** (1989): funcional (listas intensionales, abstracción lambda, fold, map) y orientado a objetos (herencia múltiple)

Bibliografía

Básica

- 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

Motivación

Conceptos

Tipos y sistemas de tipos

Polimorfismo

Sobrecarga

Coerción

Genericidad

Inclusión

Reflexión

Procedimientos y control de flujo

Paso de parámetros

Alcance de las variables

Gestión de memoria

Paradigmas de programación

Imperativo

Declarativo

OO

Concurrente

Otros paradigmas

Basado en interacción

Bibliografía

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)