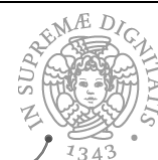




JAVA VS OCAML



Una prima analisi

- ✦ Alcune caratteristiche di Java richiedono una conoscenza dettagliata delle librerie
- ✦ Sistemi di supporto forniscono molti strumenti per programmare con Java (Eclipse è un esempio significativo)
- ✦ La nostra risposta: affrontiamo il problema in termini di problem solving

Espressioni vs Comandi



- 🐘 Ocaml è un linguaggio funzionale dove ogni espressione del linguaggio restituisce un valore
- 🐘 Java ha sia espressioni che comandi. Le espressioni restituiscono valori. I comandi operano via side effects

Metodi Statici



```
public class Max {
  public static int max (int x, int y) {
    if (x > y) {return x;}
    else { return y;}
  }
}
```

Lsimile alla
definizione di una
funzione

```
public static int max3 (int x, int y, int z) {
  return max(max(x,y), z);
}
}
```

```
public class Main {

  public static void main (String[] args) {
    System.out.println(Max.max(3, 4));
    return;
  }
}
```

Metodi statici



- ✦ Sono metodi indipendenti dall'oggetto (valgono per tutti gli oggetti della classe)
 - Non possono dipendere dai valori delle variabili di istanza
- ✦ Quando devono essere usati?
 - Per la programmazione non OO
 - Per il metodo main
- ✦ I metodi non statici sono entità dinamiche
 - Devono conoscere e lavorare sulle variabili di istanza degli oggetti.

Solita equazione



STATIC == COMPILE TIME

DYNAMIC == RUN TIME

Java: datatypes



Come possiamo programmare in Java Ocaml immutable list?

```
Type string_list = Nil | Cons of string * string_list
```

... e le funzioni ricorsive sulle liste?

```
let rec number_of_songs (pl: string_list) : int =
  begin match pl with
  | [] -> 0
  | (song :: rest) -> 1 + number_of_songs rest
  end
```

Problem solving con Java



```
interface StringList {
  public boolean isNil();
  public String hd();
  public StringList tl();
}
```

```
class Cons implements StringList {
  private String head;
  private StringList tail;
  public Cons (String h, StringList t){
    head = h; tail = t;
  }
  public boolean isNil() {
    return false; }
  public String hd () {
    return head ; }
  public StringList tl() {
    return tail; }
}
```

```
class Nil implements StringList {

  public boolean isNil() {
    return true; }
  public String hd () {
    return null; }
  public StringList tl() {
    return null; }
}
```

Operare su liste



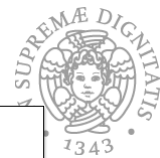
Ocaml `let x = Cons "Bunga" (Cons "Bunga", Nil)`

Java `StringList x = new Cons("Bunga", new Cons("bunga", new Nil()))`

Regole pragmatiche generali

- Per ogni tipo di dato definire la relativa interface
- Aggiungere una classe per ogni costruttore

Operare con liste



Ocaml `let rec number_of_songs (pl: strin_list) : int =
begin match pl with
| [] -> 0
| (song :: rest) -> 1 + number_of_songs rest
end`

Java `public static int numberOfSongs (StringList pl) {
if (pl.isNil()) { return 0;}
else { return 1 + numberOfSongs (pl.tail()); }
}`

Operare con liste (no recursion)



```
let rec number_of_songs (pl: string_list) : int =
  begin match pl with
  | [] -> 0
  | (song :: rest) -> 1 + number_of_songs rest
  end
```

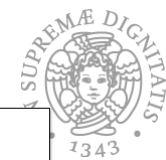
```
public static int numberOfSongs (StringList pl) {
  if (pl.isNil()) { return 0;}
  else { return 1 + numberOfSongs (pl.tail()); }
}
```

Java
(meglio)

```
public static int numberOfSongs (StringList pl) {
  int count 0 ;
  StringList curr = pl;
  while (! Curr.isNil()) {
    count = count + 1;
    curr = curr.tl();
  }
  return count ;
}
```

Usare variabili di istanza
Per value-oriented programming

Funzioni High Order?



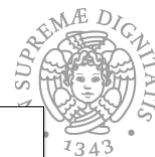
Ocaml

```
let rec map (f: string -> string)
  (pl: string_list) : string_list =
  begin match pl with
  | [] -> 0
  | (song :: rest) -> f song :: map f rest
  end
let y = ma String.uppercase (Cons "bunga bunga", Nil)
```

Java

```
public static StringList Map (??? f, StringList pl) {
  if (pl.isNil()) { return new Nil(); }
  else { return new Cons(???, map(f, pl.tail())) }
}
Public static testMap() {
  StringList x = new Cons("bunga bunga", new Nil());
  StringList y = map(???, x);
  assertEquals(y.hd(), "BUNGA BUNGA");
}
```

Usiamo le interface



```
Interface Fun { public static String apply (String x); }
Class UpperCaseFun implements Fun {
    public static String apply (String x) {
        return x.toUpperCase();
    }
}
```

```
public static StringList Map (Fun f, StringList pl) {
    if (pl.isNil()) { return new Nil(); }
    else { return new Cons(f.apply(pl.hd()), map(f, pl.tail())) }
}
Public static testMap() {
    StringList x = new Cons("bunga bunga", new Nil());
    StringList y = map(new UpperCaseFun(), x);
    assertEquals(y.hd(), "BUNGA BUNGA");
}
```

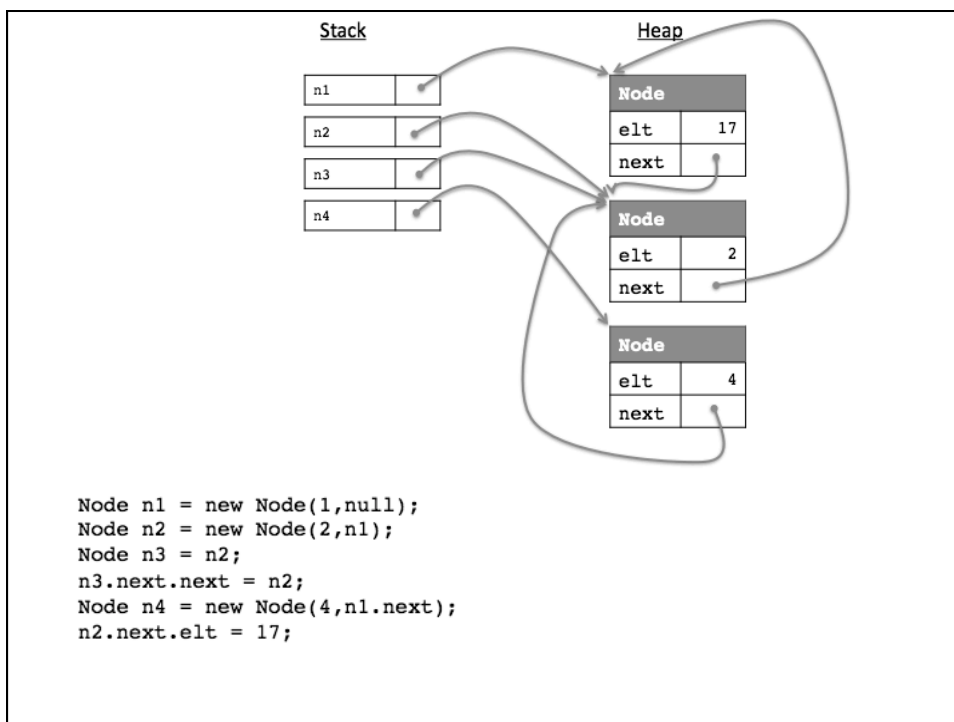
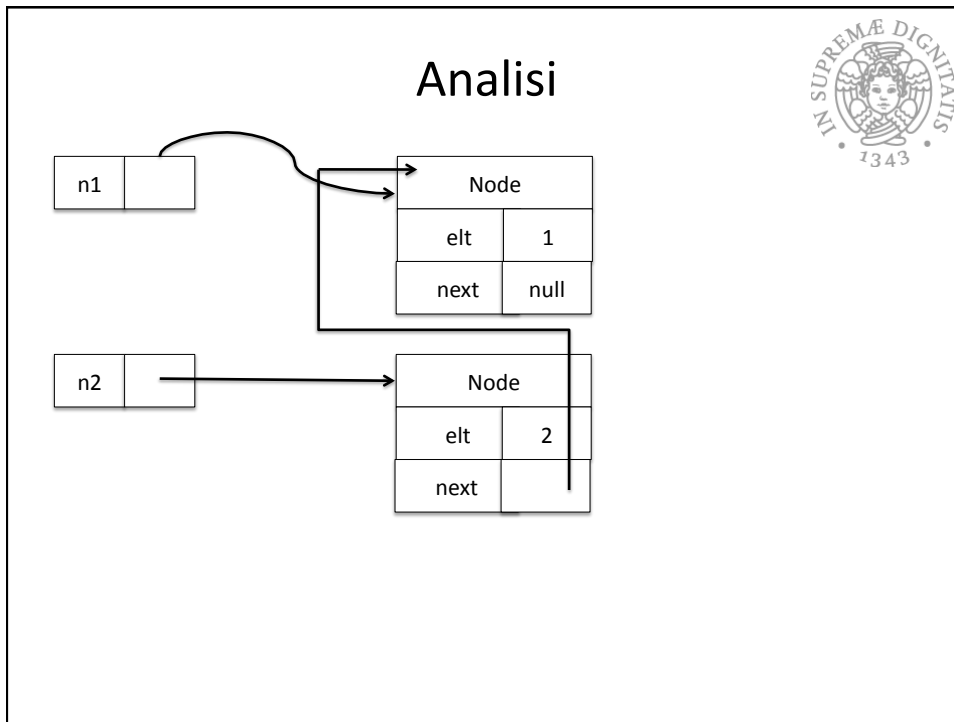
Aliasing



```
public class Node {
    public int elt;
    public Node next;
    public Node(int elt, Node next) {
        this.elt = elt;
        this.next = next;
    }
}

...
Node n1 = new Node(1,null);
Node n2 = new Node(2,n1);
Node n3 = n2;
n3.next.next = n2;
Node n4 = new Node(4,n1.next);
n2.next.elt = 17;
```

Quale è il valore di n1.elt?



Array Dinamici



```
public class DynArray {
    private int[] data = new int[0];
    private int extent = 0;

    public DynArray() { ... }

    public int get(int i) { ... }

    public void set(int index, int value) { ... }

    public int getExtent() { ... }

    public int[] values() { ... }
}

```

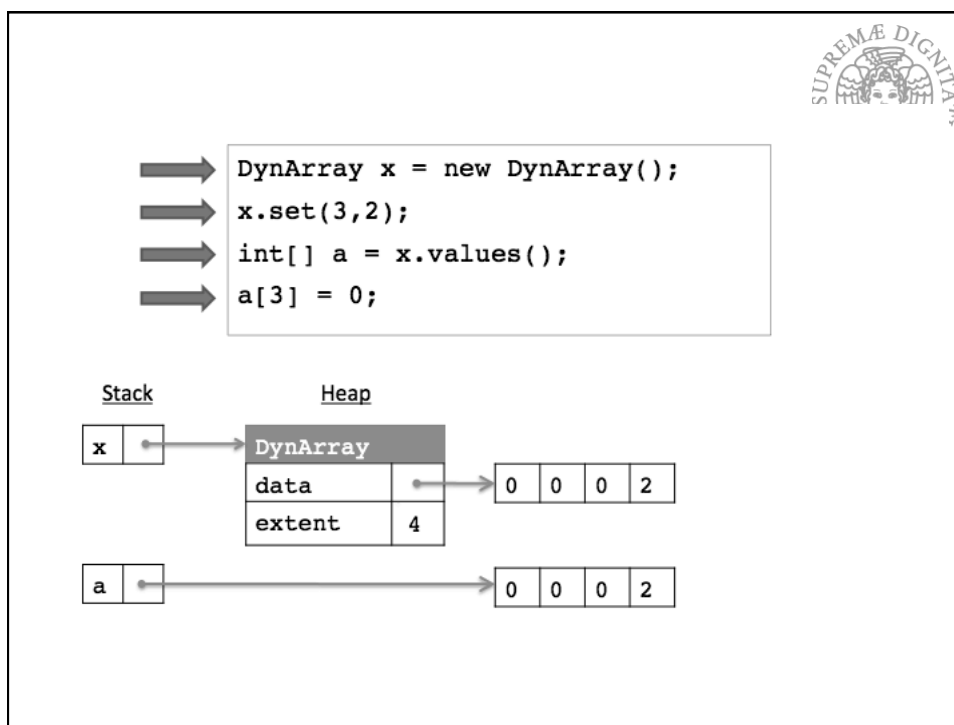
Object Invariant: extent is
always 1 past the last nonzero
value in data
(or 0 if the array is all zeros)

```
public int[] values() {
    int [] values = new int[extent];
    for (int i = 0; i < extent; i++) {
        values[i] = data[i];
    }
    return values;
}

```

```
public int[] values() {
    if (data.length == extent) {
        return data;
    }
    int [] values = new int[extent];
    for (int i = 0; i < extent; i++) {
        values[i] = data[i];
    }
    return values;
}

```



Encapsulation

- ✎ Tutte le modifiche allo stato di un oggetto devono essere fatte tramite i metodi propri dell'oggetto
- ✎ Encapsulation deve preservare le proprietà invarianti dell'oggetto
- ✎ Pertanto non si devono restituire dei valori di aliasing dai metodi

Cde Mutabili (in Ocaml)



```

module type QUEUE =
sig
  (* type of the data structure *)
  type 'a queue

  (* Make a new, empty queue *)
  val create : unit -> 'a queue

  (* Add a value to the end of the queue *)
  val enq : 'a -> 'a queue -> unit

  (* Remove the front value and return it (if any) *)
  val deq : 'a queue -> 'a

  (* Determine if the queue is empty *)
  val is_empty : 'a queue -> bool

  (* Remove the first occurrence of the value. *)
  val delete : 'a -> 'a queue -> unit
end

```

In Java



```

public interface Queue<E> {

  /** Determine if the queue is empty*/
  public boolean is_empty ();

  /** Add a value to the end of the queue */
  public void enq (E elt);

  /** Remove the front value and return it (if any) */
  public E deq ();

  /** Remove the first occurrence of the value */
  public void delete (E elt);

}

```