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;}
  }

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

Lsimile alla definizione di una funzione

```
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 volori 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.

- I metodi statici sono "funzioni" definite globalmente.
- Variabili di istanza static sono variabili globali accessibile tramite il nome della classe.
- Variabili di istanza statiche non possono essere inizializzate nel costruttore della classe (dato che non possono essere associate a oggetti istanza della classe).

Esempio

```
public class C {
  private static int big = 23;
  public static int m(int x) {
     return big + x;
  }
 }
```

Esempio

```
public class C {
  private static int big = 23;
  private int nonStaticField;
  private void setIt(int x) { nonStaticField = x + x; }

  public static int m(int x) {
    setIt(x); // Errore: un metodo static non puo' accedere a
              // metodi non statici
              // e a variabili di istanza non statiche
  }
}
```

Quindi?

- Metodi statici sono utilizzati per implementare funzionalita' che non dipendono dallo stato dell'oggetto
- Esempi significativi Java Math API che fornisce numerose funzioni matematiche come l Math.sin,
- Altri esempio sono dati dalle funzioni di conversione: Integer.toString e Boolean.valueOf.

Java: datatypes

Come possiamo programmare in Java le immutable list di OCaml?

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

… e le funzioni ricorsive sulle liste?

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

Problem solving con Java

```
interface StringList {
        public bolean 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)
```

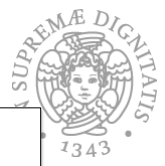
Java

```
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: strin_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 0rder?

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 = map 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);
assetEqual(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);
assetEqual(y.hd(), “BUNGA BUNGA”);
}
```

LE STRINGHE IN JAVA

Java String

- Le stringhe (sequenze di caratteri) in Java sono una classe predefinita.
- **"3" + " " + "Volte 3"** ⇒ **"3 Volte 3"** : l'operatore + e' l'operatore di concatenazione di stringhe
- Le stringhe sono oggetti immutabili (a la' Ocaml)

Uguaglianza

- Java prevede due operatori per testare l'uguaglianza
- o1 == o2 restituisce true se le due variabili o1 e o2 denotano lo stesso riferimento (pointer equality)
- o1.equals(o2) restituisce true se i due oggetti sono identici (deep equality)
- Esempio
  - String("test").equals("test") --> true
  - new String("test") == "test" --> false
  - new String("test") == new String("test") --> false

problema


```
String s1 = "java";
String s2 = "java";
```

```
s1.equals(s2)  // true --perche?
s1==s2  // true --perche?
```

problema

```
String str1 = new String("java");
String str2 = new String("java");

str1.equals(str2)  // true --stesso contenuto

str1==str2  // false -- oggetti differenti
```

String: oggetti immutabili

```
public class Main {
  public static void main(String [] args) {
  String s1 = "programmare in";
  if (s1.equals(s1.concat("java"))) {
       System.err.println("True!");
    } else {
       System.err.println("False!");
    }
  }
}
```

String: oggetti immutabili


```
public class Main {
  public static void main(String [] args) {
  String s1 = "programmare in";
  if (s1.equals(s1.concat("java"))) {
       System.err.println("True!");
    } else {
       System.err.println("False!");
    }
  }
}
```

Viene creato un nuovo Oggetto!!!