A spiral-bound notebook with a light beige, textured cover. The spiral binding is on the left side. The text is centered on the page.

Progettazione dettagliata di un
tipo di dato astratto:
l'ambiente

L'ambiente

```
# module type ENV =  
  sig  
    type env  
    val emptyenv : env  
    val bind : env * string * eval -> env  
    val bindlist : env * (string list) * (eval list)  
                  -> env  
    val applyenv : env * string -> eval  
    exception WrongBindlist  
  end  
# type eval = Int of int | Efun of expr * env  
            | Unbound;;
```

L'ambiente 1

```
public class Env {
    // OVERVIEW: un Env è una
    // funzione parziale da identificatori
    // (stringhe) a valori
    // esprimibili. Non è modificabile.
    // costruttore
    public Env ()
        // EFFECTS: costruisce un nuovo
        // Env indefinito per tutti
        // gli identificatori (emptyenv)
```

L'ambiente 2

```
public Eval apply (String s) throws  
    Undefined
```

```
// EFFECTS: se la funzione this è  
// definita per l'identificatore s  
// restituisce this(s) altrimenti  
// solleva l'eccezione (unchecked)  
// Undefined
```

L'ambiente 3

```
public Env bind (String s, Eval e)
```

```
    // EFFECTS: restituisce una funzione
```

```
    // diversa da this solo perché se
```

```
    // applicata ad s restituisce e
```

```
public Env bindlist (String[] s, Eval[] e) throws  
    WrongBindlist
```

```
    // EFFECTS: restituisce una funzione
```

```
    // ottenuta facendo il bind di stringhe
```

```
    // ed eval in posizioni corrispondenti in
```

```
    // s ed e. Se s ed e hanno lunghezze
```

```
    // diverse, solleva l'eccezione WrongBindlist
```

Di che altri tipi abbiamo bisogno?

✓ i valori esprimibili

```
# type eval = Int of int | Efun of expr * env  
| Unbound;;
```

- Eval

✓ i due exception types (unchecked)

- WrongBindlist

- Undefined

I tipi eccezione

✓ i due exception types (unchecked)

- WrongBindlist
- Undefined

```
public class WrongBindlist extends  
    RuntimeException {  
// costruttore  
    public WrongBindlist (String s)  
    {super (s); }    }
```

I tipi eccezione

✓ i due exception types (unchecked)

- NonObjectException
- Undefined

```
public class Undefined extends  
    RuntimeException {  
// costruttore  
    public Undefined (String s)  
    {super (s); }    }
```


I valori esprimibili

– Eval

```
# type eval = Int of int | Efun of expr * env  
           | Unbound;;
```

```
public class Eval {
```

```
    // un Eval è un oggetto che può
```

```
    // essere un intero,
```

```
    // una chiusura, oppure indefinito
```

Eval: i costruttori

...

// 3 costruttori overloaded per i diversi casi

```
public Eval ()
```

// EFFECTS: costruisce un Eval

// indefinito

```
public Eval (int x)
```

// EFFECTS: costruisce un Eval

// contenente l'intero x

```
public Eval (Closure x)
```

// EFFECTS: costruisce un Eval

// che contiene la chiusura x

Eval: i riconoscitori

```
public boolean isint ()  
    // EFFECTS: se this è un int  
    // restituisce true altrimenti restituisce false  
public boolean isUndefined ()  
    // EFFECTS: se this è indefinito  
    // restituisce true altrimenti restituisce false  
public boolean isClosure ()  
    // EFFECTS: se this è una chiusura  
    // restituisce true altrimenti restituisce false
```

Eval: i selettori

```
public int getInt () throws Undefined
    // EFFECTS: se this è un int lo restituisce
    // altrimenti solleva Undefined
public Closure getClosure () throws Undefined
    // EFFECTS: se this è una chiusura la restituisce
    // altrimenti solleva Undefined }
```

Di che altri tipi abbiamo bisogno?

- le chiusure
 - `Closure = Expr * Env`
- si realizzano direttamente con un record type
 - facciamo finta che le espressioni siano semplicemente stringhe

Le chiusure 1

```
public class Closure {  
    // OVERVIEW: una Closure è un record type  
    // che contiene una stringa (codifica del  
    // codice di una funzione) ed un ambiente  
    String codice;  
    Env ambiente;
```

Le chiusure 2

// costruttore

```
public Closure (String x, Env y)
```

// EFFECTS: costruisce un record

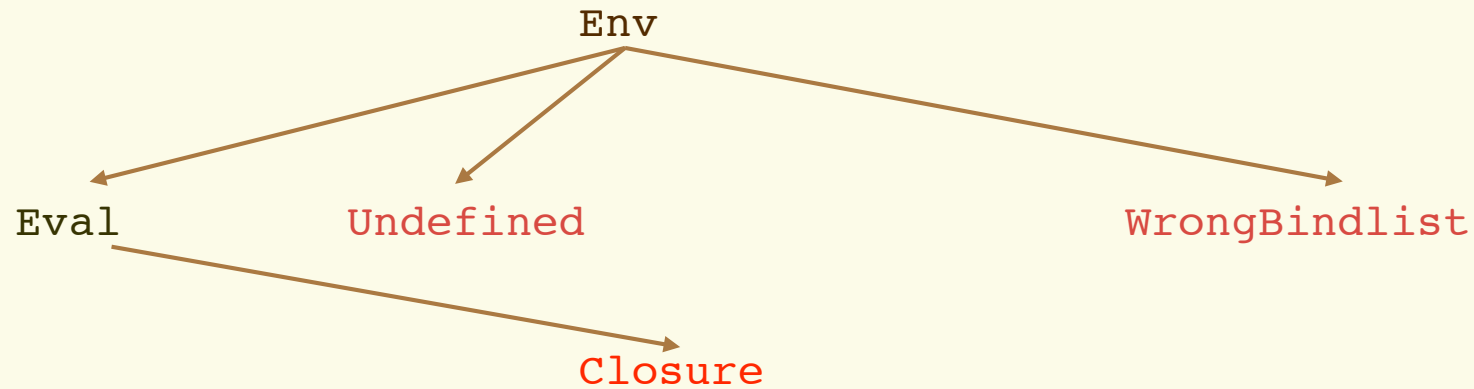
// contenente i due argomenti

```
{codice = x; ambiente = y;} }
```

Lo stato della progettazione

- ✓ classi specificate e implementate (eccezioni e record types)
 - Closure
 - WrongBindlist
 - Undefined
- ✓ classi specificate
 - Eval
 - Env

La relazione “chi usa chi” a livello di specifica



Come continuare

✓ le classi da implementare

- Eval
- Env

✓ possono essere implementate in un ordine qualunque

Implementazione di Eval 1

```
public class Eval {  
    // un Eval è un intero, una chiusura  
    // oppure indefinito  
    private Object v;
```

✓ lo stato è una variabile di tipo **Object**

Implementazione di Eval 2

```
public Eval ()  
  // EFFECTS: costruisce un Eval indefinito  
  { }  
  
public Eval (int x)  
  // EFFECTS: costruisce un Eval contenente l'intero x  
  { v = new Integer(x); }  
  
public Eval (Closure x)  
  // EFFECTS: costruisce un Eval che contiene la chiusura x  
  { v = x; }
```

Implementazione di Eval 3

```
public boolean isClosure ()  
    // EFFECTS: se this è una chiusura  
    // restituisce true altrimenti restituisce false  
    {if (isUndefined()) return false;  
    Closure c;  
    try {c = (Closure) v;}  
    catch (Exception e) {return false; }  
    return true; }
```

- ✓ utilizziamo le eccezioni (mascherate) per fare un'analisi di casi sul tipo

Implementazione di Eval 4

```
public boolean isUndefined ()  
    // EFFECTS: se this è indefinito  
    // restituisce true altrimenti  
    // restituisce false  
{ if (v == null) return true;  
  else return false; }
```

Implementazione di Eval 5

```
public boolean isint ()  
    // EFFECTS: se this è un int  
    // restituisce true altrimenti restituisce false  
    {if (isUndefined()) return false;  
    Integer l;  
    try {l = (Integer) v;}  
    catch (Exception e) {return false;}  
    return true;}
```

Implementazione di Eval 6

```
public int getInt () throws Undefined
    // EFFECTS: se this è un int lo restituisce
    // altrimenti solleva Undefined
    {if (isUndefined()) throw new(Undefined);
    Integer n;
    try {n = (Integer) v;}
    catch (Exception e) {throw new(Undefined); }
    return n.intValue(); }

public Closure getClosure () throws Undefined
    // EFFECTS: se this è una chiusura la restituisce
    // altrimenti solleva Undefined
    {if (isUndefined()) throw new(Undefined);
    Closure c;
    try {c = (Closure) v;}
    catch (Exception e) {throw New(Undefined); }
    return c; } }
```


Implementazione di Env

```
public class Env {  
    // OVERVIEW: un Env è una  
    // funzione parziale da identificatori  
    // (stringhe) a valori  
    // esprimibili. Non è modificabile.
```

✓ decidiamo di rappresentare le funzioni con arrays i cui elementi sono coppie (`String`, `Eval`)

– specifichiamo quindi per prima cosa un tipo record

PairIdeVal

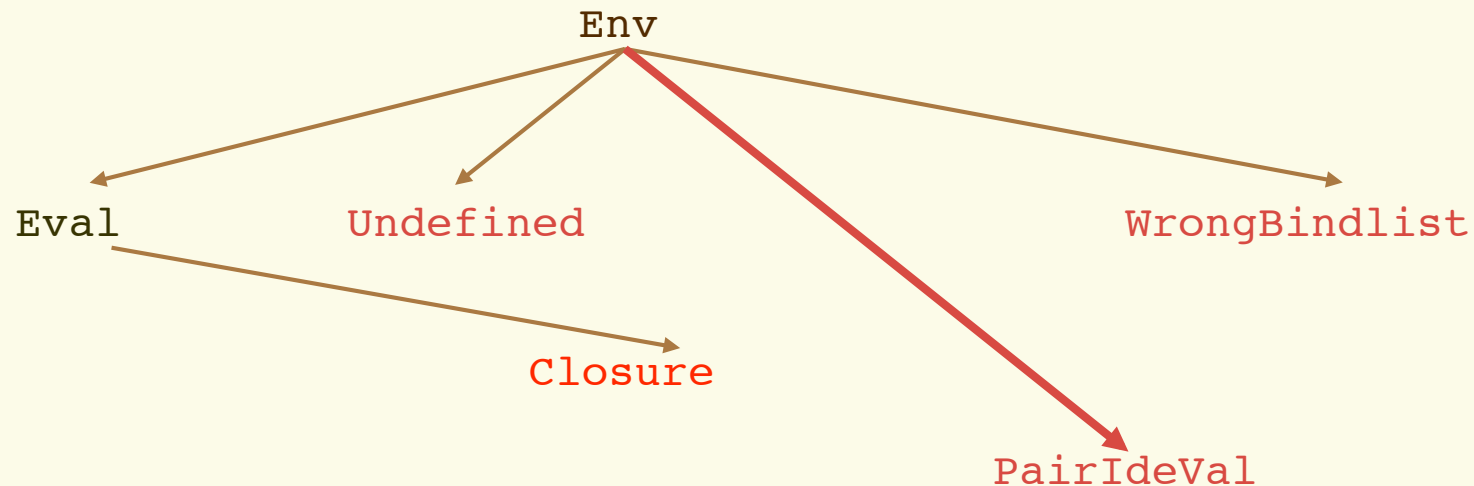
- un po' più generale di quello che ci serve

Specifica ed implementazione di PairIdeVal

```
public class PairIdeVal {  
    // OVERVIEW: un PairIdeVal è un record  
    // type che contiene una associazione  
    // tra nome e valore  
    String nome;  
    Object valore;  
    // costruttore  
    public PairIdeVal (String x, Object z)  
        // EFFECTS: costruisce un record  
        // contenente i due argomenti  
        {nome = x; valore = z;} }  
}
```

- un po' più generale di quello che ci serve
 - i valori sono Object e non Eval

La relazione “chi usa chi” a livello di specifica e di implementazione



Implementazione di Env 1

```
public class Env {  
    // OVERVIEW: un Env è una  
    // funzione parziale da identificatori  
    // (stringhe) a valori  
    // esprimibili. Non è modificabile.  
  
    private PairIdeVal[] associazioni;
```

Implementazione di Env 2

```
// costruttore pubblico
public Env ()
    // EFFECTS: costruisce un nuovo
    // Env indefinito per tutti
    // gli identificatori
{associazioni = new PairIdeVal[0];}
```

Implementazione di Env 3

```
// altri costruttori privati
private Env (PairIdeVal[] m)
    // EFFECTS: costruisce una nuova
    // funzione la cui rappresentazione
    // è m
    {associazioni =
        new PairIdeVal[m.length];
    if (m.length == 0) return ;
    for (int i = 0; i < m.length; i++) {
        associazioni[i] = m[i];}    }
```

Implementazione di Env 4

```
// altri costruttori privati
private Env (int n)
  // EFFECTS: costruisce una nuova
  // funzione la cui rappresentazione
  // è un array vuoto lungo n
  {associazioni = new PairIdeVal[n];}
```

Implementazione di Env 5

```
// altri costruttori privati
private Env (PairIdeVal[] m, int n)
  // REQUIRES: n = m.length+1
  // EFFECTS: costruisce una nuova
  // funzione la cui rappresentazione
  // è un array uguale ad m più una
  // posizione vuota
  {associazioni = new PairIdeVal[n];
  if (m.length == 0) return ;
  for (int i = 0; i < m.length; i++) {
    associazioni[i] = m[i];} }
```


Implementazione di Env 6

```
public Eval apply (String s) throws Undefined
// EFFECTS: se la funzione this è
// definita per l'identificatore s
// restituisce this(s) altrimenti
// solleva l'eccezione (unchecked)
// Undefined
{for (int i = 0; i < associazioni.length; i++) {
    PairIdeVal p = associazioni[i];
    if (s.equals(p.nome))
        return (Eval) p.valore;}
throw new Undefined("Env.apply");}
```

Implementazione di Env 7

```
public Env bind (String s, Eval m)
    // EFFECTS: restituisce una funzione
    // diversa da this solo perché se
    // applicata ad s restituisce m
    {Env nuovo;
    PairIdeVal newass = new PairIdeVal(s,m);
    if (defined(s))
        {nuovo = new Env(associazioni);
        for (int i = 0; i < nuovo.associazioni.length; i++) {PairIdeVal p =
            nuovo.associazioni[i];
            if (s.equals(p.nome))
                nuovo.associazioni[i] = newass;}
        return nuovo;}
    else {if (associazioni.length == 0)
        {nuovo = new MethodEnv(1);}
        else {nuovo = new MethodEnv(associazioni,
            associazioni.length + 1);}
    nuovo.associazioni[associazioni.length] = newass;
    return nuovo;}
```

Implementazione di Env 8

```
public Env bindlist (String[] s, Eval[] e) throws WrongBindlist
    // EFFECTS: restituisce una funzione
    // ottenuta facendo il bind di stringhe
    // ed eval in posizioni corrispondenti in
    // s ed e. Se s ed e hanno lunghezze
    // diverse, solleva l'eccezione WrongBindlist
```

Da fare per esercizio!

Implementazione di Env 9

```
private boolean defined (String s)  
    // EFFECTS: se this è definita per  
    // l'identificatore s ritorna true  
    // altrimenti ritorna false
```

Da fare per esercizio!