Progettazione gerarchica delle s-espressioni, utilizzando l'ereditarietà

Due implementazioni alternative delle s-espressioni

- ✓ alberi binari (possibilmente "vuoti", `nil`) che hanno sulle foglie atomi (stringhe)
- ✓ la definizione ricorsiva del tipo come verrebbe scritta in ML
  - `type sexpr = Nil |`
    `Atom of string |`
    `Cons of sexpr * sexpr`
- ✓ vogliamo dare due implementazioni "alternative"

Due implementazioni

```
type sexpr = Nil |
     Atom of string   |
     Cons of sexpr * sexpr
```

- ✓ vogliamo dare due implementazioni “alternative”
- ✓ una è quella che abbiamo visto
  - ogni oggetto ha 4 campi
    - solo alcuni di questi sono utilizzati nei vari casi della definizione ricorsiva
- ✓ la seconda implementazione utilizza i sottotipi per realizzare i diversi casi
- ✓ alcune operazioni possono essere comuni alle due implementazioni
  - `Sexpr` è una classe astratta e non un interfaccia

Specifica e implementazione della classe astratta `Sexpr` 1

```
public abstract class Sexpr {
// OVERVIEW: una Sexpr è un albero binario modificabile
// che ha sulle foglie atomi (stringhe)
// scompaiono i costruttori
// metodi astratti
 public abstract Sexpr cons (Sexpr s) throws
    NullPointerException;
  // EFFECTS: costruisce un nuovo albero binario che ha
  // this come sottoalbero sinistro ed s come
  // sottoalbero destro. Se s è indefinito,
  // solleva NullPointerException
 public abstract void rplaca (Sexpr s) throws
    NotANodeException;
  // EFFECTS: rimpiazza in this il sottoalbero sinistro
  // con s. Se this non è un nodo binario solleva
  // NotANodeException (checked). Se s è indefinito,
  // solleva NullPointerException
 public abstract void rplacd (Sexpr s) throws
    NotANodeException;
  // EFFECTS: rimpiazza in this il sottoalbero destro
  // con s. Se this non è un nodo binario solleva
  // NotANodeException (checked). Se s è indefinito,
  // solleva NullPointerException
```

Specifica e implementazione della classe astratta `Sexpr` 2

```
public abstract Sexpr car () throws NotANodeException;
 // EFFECTS: ritorna il sottoalbero sinistro di this.
 // Se this non è un nodo binario solleva
 // NotANodeException (checked)
public abstract Sexpr cdr () throws NotANodeException;
 // EFFECTS: ritorna il sottoalbero destro di this.
 // Se this non è un nodo binario solleva
 // NotANodeException (checked)
public abstract String getatom () throws NotAnAtomException;
 // EFFECTS: Se this non è una foglia solleva
 // NotAnAtomException (checked). Altrimenti ritorna la stringa
 // contenuta nella foglia
public abstract boolean nullS() throws NullPointerException;
 // EFFECTS: ritorna true se this è l'albero vuoto,
 // altrimenti ritorna false.
public abstract boolean atom () throws NullPointerException;
 // EFFECTS: ritorna false se this è un albero binario,
 // altrimenti ritorna true.
public abstract String toString ();
```

Specifica e implementazione della classe astratta `Sexpr` 3

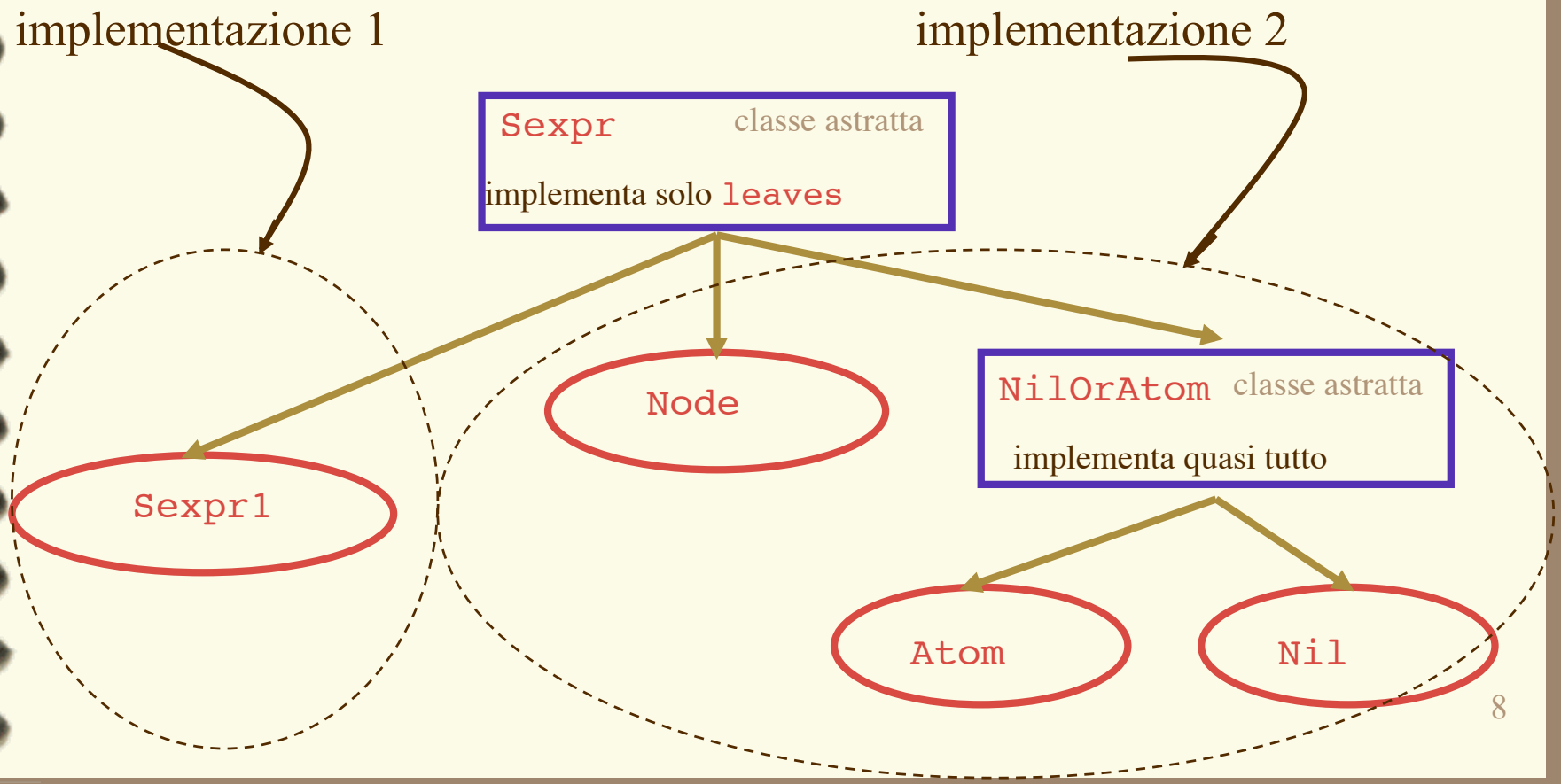
```
// Metodi concreti
 public Iterator leaves ()
  // REQUIRES: this non deve essere ciclico
  // EFFECTS: ritorna un generatore che produrrà le foglie
  // nella frontiera di this (come Strings), da sinistra a
  // destra
  // REQUIRES: this non deve essere modificato finché
  // il generatore è in uso
  {return new LeavesGen(this,numerofoglie());}

 private int numerofoglie () {
 // riscritta senza usare la rappresentazione
   if (nullS()) return 0;
   if (atom()) return 1;
   try {return (car().numerofoglie() +
      cdr().numerofoglie()); }
   catch (NotANodeException e) {return 0; } }
```

Specifica e implementazione della classe astratta `Sexpr` 4

```
// classe interna concreta (implementazione che non utilizza
// la rep di Sexpr)
 private static class LeavesGen  implements Iterator {
   private LeavesGen figlio; // sottogeneratore corrente
   private Sexpr io; // il mio albero
   private int quanti; // numero di elementi ancora da generare
   LeavesGen (Sexpr s,int n)  {
    //REQUIRES: s != null
    quanti = n;
    if (quanti > 0)
     {io = s; if (io.atom()) return;
      try {figlio = new LeavesGen(io.car(),
        io.car().numerofoglie()); }
      catch (NotANodeException e) {}
      return; }
    return;}
  public boolean hasNext() { return quanti > 0;}
  public Object next () throws NoSuchElementException {
   if (quanti == 0) throw new
       NoSuchElementException("Sexpr.leaves");
   quanti--; if (io. atom())
    try {return io.getatom();} catch (NotAnAtomException e) {}
   try {return figlio.next();} catch (NoSuchElementException
       e) {}
   try {figlio = new LeavesGen(io.cdr(),
      io.cdr().numerofoglie()); return figlio.next(); }
    catch (NotANodeException e) {
      throw new NoSuchElementException("Sexpr.leaves");}
```

Struttura della gerarchia

```
type sexpr = Nil |
     Atom of string   |
     Cons of sexpr * sexpr
```



Implementazione di Sexpr1 1

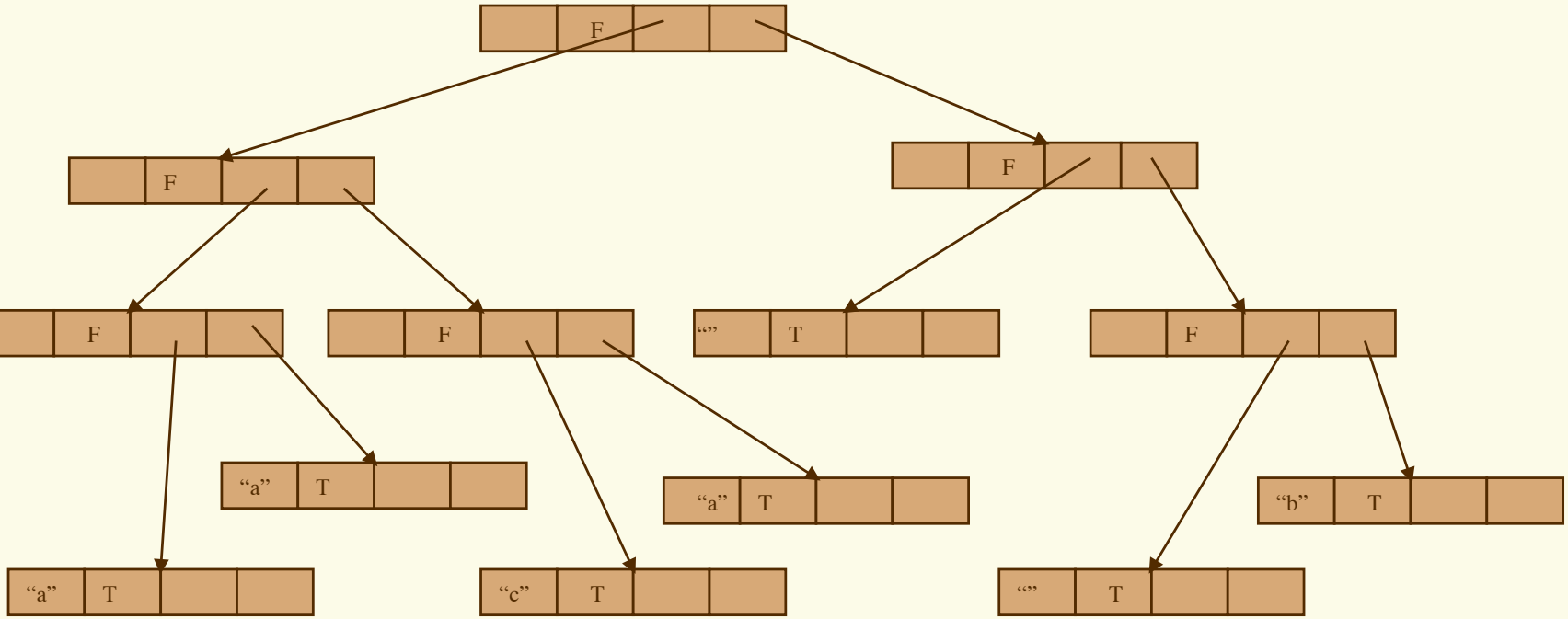
```
public class Sexpr1 extends Sexpr {
// OVERVIEW: una Sexpr è un albero binario modificabile
// che ha sulle foglie atomi (stringhe)
 private boolean foglia;
 private Sexpr sinistro, destro;
 private String stringa;
 public Sexpr1 ()
    // EFFECTS: inizializza this alla Sexpr vuota
  {foglia = true; stringa = ""; }
 public Sexpr1 (String s)
    // EFFECTS: inizializza this alla foglia contenente s
  {foglia = true; stringa = s; }
 public Sexpr cons (Sexpr s) throws NullPointerException
 {if (s == null) throw new
      NullPointerException ("Sexpr1.cons");
  Sexpr1 nuovo = new Sexpr1();
  nuovo.sinistro = this; nuovo.destro = s;
  nuovo.foglia = false; return (Sexpr) nuovo; }
 public void rplaca (Sexpr s) throws NotANodeException
    {if (foglia) throw new
           NotANodeException("Sexpr.rplaca");
     if (s == null) throw new
      NullPointerException ("Sexpr1.rplaca");

    sinistro = s; return; }
.....
```

Implementazione di Sexpr1 2

```
public class Sexpr1 extends Sexpr {
// OVERVIEW: una Sexpr è un albero binario modificabile
// che ha sulle foglie atomi (stringhe)
 private boolean foglia;
 private Sexpr sinistro, destro;
 private String stringa;
...
 public String getatom () throws NotAnAtomException
  { if (!foglia || stringa == "") throw new
        NotAnAtomException("Sexpr1.getatom");
    return stringa; }
 public String toString() {
  if (foglia) {if (stringa == "") return "nil"; else return stringa; }
  return "(" + sinistro.toString() + "." + destro.toString() + ")"; }
```

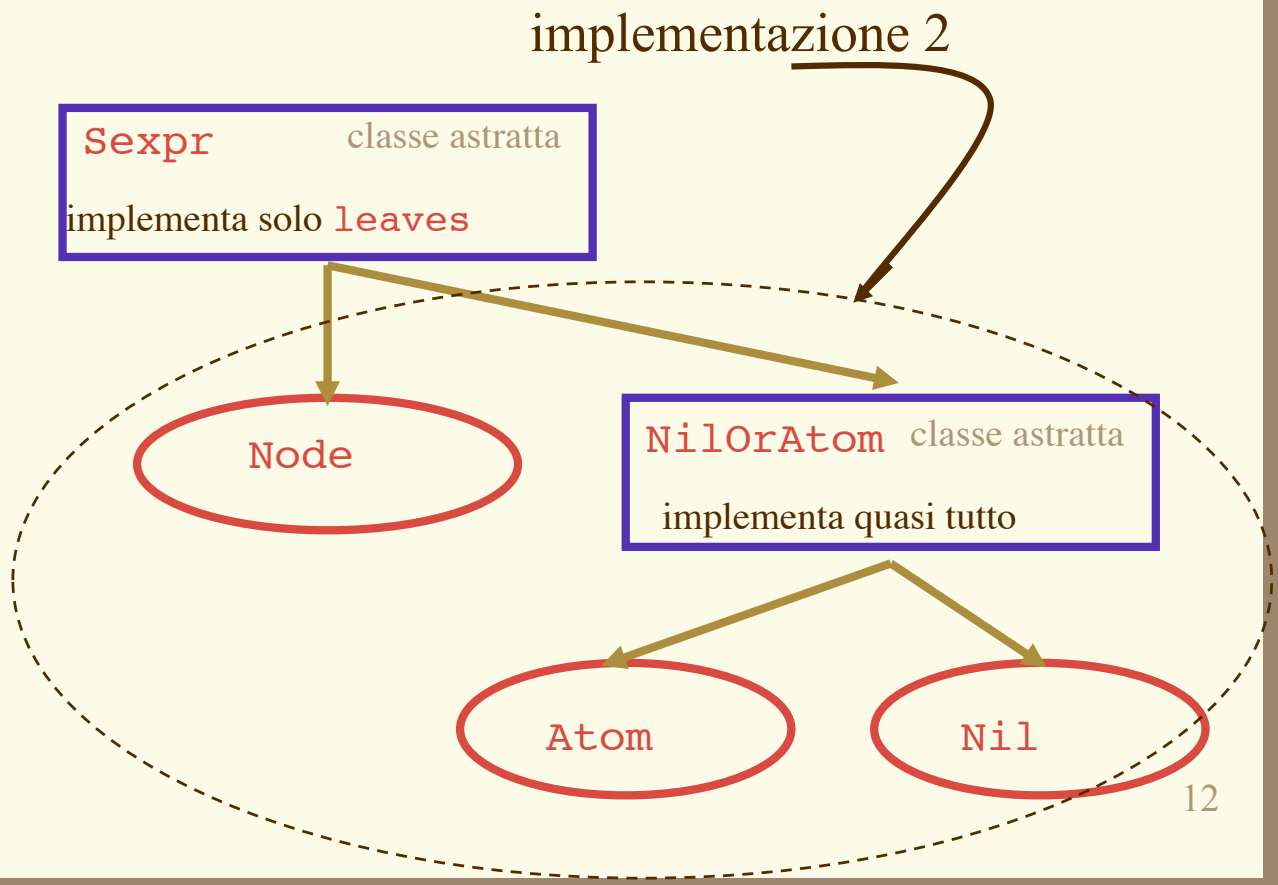
Come è fatta una `Sexpr1`

```
(((((new Sexpr1("a")).cons(new Sexpr1("a"))).
  cons((new Sexpr1("c")).cons(new Sexpr1("a")))).
   cons((new Sexpr1()).cons((new Sexpr1()).cons(new Sexpr1
   ("b"))))))
```



L'implementazione 2

```
type sexpr = Nil |
     Atom of string   |
     Cons of sexpr * sexpr
```



La classe Node

- ✓ implementa il caso ricorsivo della definizione
- ✓ notare che le specifiche di alcune operazioni astratte sono ridate perché diverse
  - definiscono un solo caso e quindi possono sollevare meno eccezioni oppure .....
- ✓ vediamo comunque insieme specifiche e implementazione

Implementazione di Node 1

```
public class Node extends Sexpr {
// OVERVIEW: una Sexpr è un albero binario modificabile
// che ha sulle foglie atomi (stringhe)
 private Sexpr sinistro, destro;
 public Node () {}
    // EFFECTS: inizializza this ad un nodo indefinito
 public Sexpr cons (Sexpr s) throws NullPointerException
 {if (s == null) throw new
     NullPointerException ("Node.cons");
  Node nuovo = new Node();
  nuovo.sinistro = this; nuovo.destro = s;
  return (Sexpr) nuovo; }
 public void rplaca (Sexpr s)
  // EFFECTS: rimpiazza in this il sottoalbero sinistro con s. Se s è indefinito,
  // solleva NullPointerException
  // Non solleva NotANodeException
    {if (s == null) throw new
      NullPointerException ("Node.rplaca");
    sinistro = s; return; }
 public void rplacd (Sexpr s)
  // EFFECTS: rimpiazza in this il sottoalbero destro con s. Se s è indefinito,
  // solleva NullPointerException
  // Non solleva NotANodeException
    {if (s == null) throw new
      NullPointerException ("Node.rplacd");
    destro = s; return; }
```

Implementazione di Node 2

```
public class Node extends Sexpr {
// OVERVIEW: una Sexpr è un albero binario modificabile
// che ha sulle foglie atomi (stringhe)
 private Sexpr sinistro, destro;
 public Sexpr car ()
  // EFFECTS: restituisce il sottoalbero sinistro di this.
  // Non solleva NotANodeException
    {
    return sinistro; }
 public Sexpr cdr ()
  // EFFECTS: restituisce il sottoalbero destro di this.
  // Non solleva NotANodeException
    {
    return destro; }
 public String getatom () throws NotAnAtomException
  // EFFECTS: Solleva NotAnAtomException
    {
    throw new NotAnAtomException("Node.getatom"); }
```

Implementazione di Node 3

```
public class Node extends Sexpr {
// OVERVIEW: una Sexpr è un albero binario modificabile
// che ha sulle foglie atomi (stringhe)
 private Sexpr sinistro, destro;
 public boolean nullS ()
  // EFFECTS: restituisce false
    {
    return false; }
 public boolean atom ()
  // EFFECTS: restituisce false
    {
    return false; }
 public String toString ()
 {return "(" + sinistro.toString() + "." + destro.toString() + ")";}  }
```

`Nil` e `Atom`

- implementano quasi tutte le operazioni nello stesso modo
  - sono diverse solo su `NullS`, `getatom` e `toString`
- facciamo una classe astratta `NilOrAtom` in cui implementiamo le operazioni comuni
  - i metodi non definiti continuano a restare astratti come nella classe `Sexpr` e verranno implementati nelle due classi concrete `Nil` e `Atom`

La classe astratta `NilOrAtom` 1

```
public abstract class NilOrAtom extends Sexpr {
// OVERVIEW: un NilOrAtom è un albero vuoto o una foglia
 public Sexpr cons (Sexpr s) throws NullPointerException
 {if (s == null) throw new
      NullPointerException (“Node.cons”);
  Node nuovo = new Node();
  nuovo.sinistro = this; nuovo.destro = s;
  return (Sexpr) nuovo; }
 public void rplaca (Sexpr s) throws NotANodeException
  // EFFECTS: Solleva NotANodeException
    {if (s == null) throw new
      NullPointerException (“NilOrAtom.rplaca”);
    throw new NotANodeException (“NilOrAtom.rplaca”);; }
 public void rplacd (Sexpr s) throws NotANodeException
  // EFFECTS: Solleva NotANodeException
    {if (s == null) throw new
      NullPointerException (“NilOrAtom.rplacd”) ;
     throw new NotANodeException (“NilOrAtom.rplacd”);}
 public Sexpr car () throws NotANodeException
  // EFFECTS: Solleva NotANodeException
    {
    throw new NotANodeException (“NilOrAtom.car”); }
```

La classe astratta `NilOrAtom` 2

```
public abstract class NilOrAtom extends Sexpr {
// OVERVIEW: un NilOrAtom è un albero vuoto o una foglia
 public Sexpr cdr () throws NotANodeException
  // EFFECTS: Solleva NotANodeException
    {
     throw new NotANodeException ("NilOrAtom.cdr");}
 public boolean atom ()
 // EFFECTS: Ritorna true
    {return true ;}  }
```

La classe `Nil`

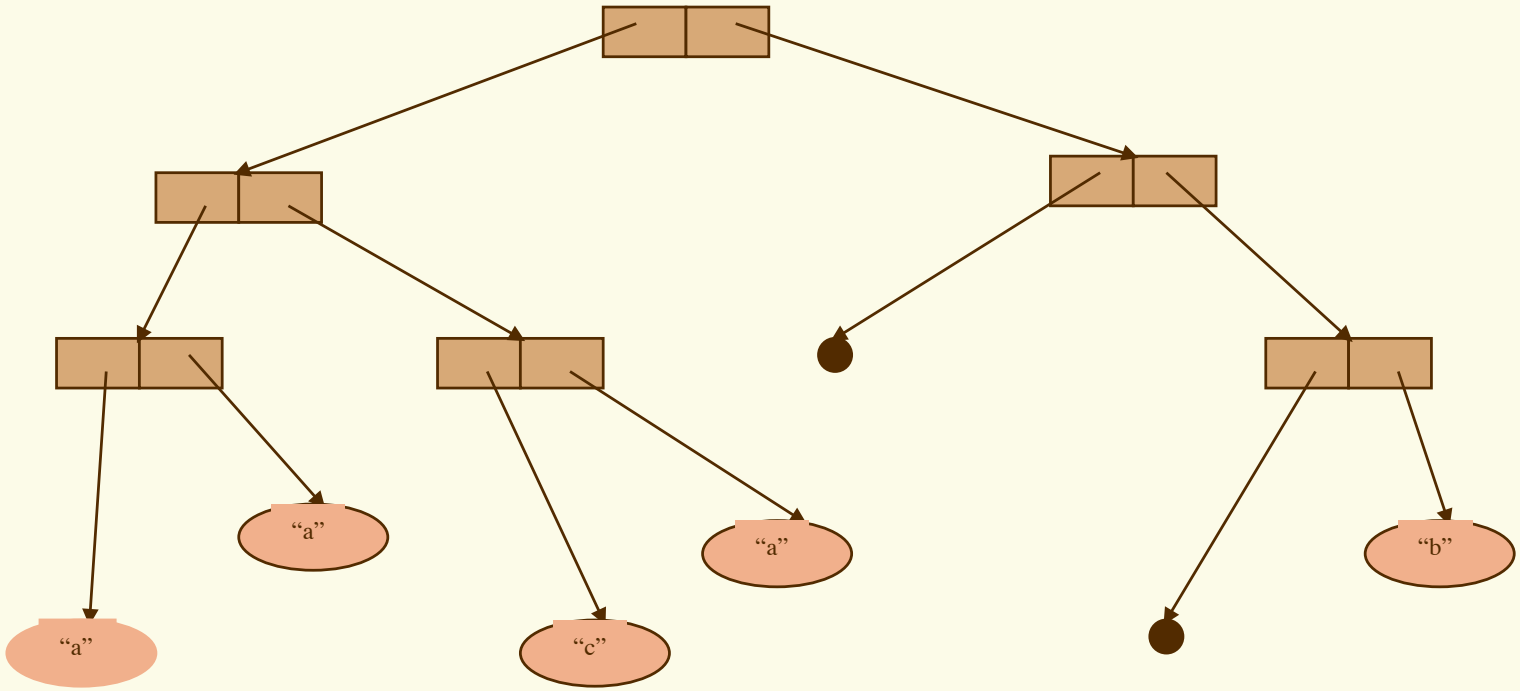
```
public class Nil extends NilOrAtom {
// OVERVIEW: un Nil è un albero vuoto
 public Nil () {}
  // EFFECTS: Crea un albero vuoto
 public boolean nullS ()
 // EFFECTS: Ritorna true
    {
     return true ;}
 public String getatom () throws NotAnAtomException
  // EFFECTS: Solleva NotAnAtomException
    {
     throw new NotAnAtomException ("Nil.getatom");}
 public String toString ()
    {return "nil" ;} }
```

La classe Atom

```
public class Atom extends NilOrAtom {
// OVERVIEW: un Atom è una foglia
 String atomo;
 public Atom (String s) { atomo = s; }
  // EFFECTS: Crea ua foglia contenente s
 public boolean nullS ()
 // EFFECTS: Ritorna false
    {
     return false ;}
 public String getatom ()
  // EFFECTS: Restituisce la stringa contenuta nella foglia. Non solleva
  // NotAnAtomException
    {return atomo ;}
 public String toString ()
    {return atomo;} }
```

Come è fatta una `Sexpr` (seconda implementazione)

```
(((((new Atom("a")).cons(new Atom("a"))).
  cons((new Atom("c")).cons(new Atom("a"))))).
   cons((new Nil()).cons((new Nil()).cons(new Atom("b"))))))
```



`Sexpr` (mescoliamo le implementazioni)

- possiamo usare insieme (e mescolare) le 4 sottoclassi concrete di `Sexpr`
  - nil e gli atomi si possono indifferentemente costruire con i costruttori di `Nil`, `Atom` e `Sexpr1`
  - la scelta fra il `cons` di `Sexpr1` e quelli (tutti uguali) di `Node`, `Atom` o `Nil` è guidata dal tipo dell'oggetto

```
(((((new Sexpr1("a")).cons(new Atom("a"))).
  cons((new Atom("c")).cons(new Atom("a")))).
   cons((new Sexpr1()).cons((new Nil()).cons(new Atom
   ("b"))))))
```

Confrontiamo le specifiche per il supertipo ed i sottotipi: `rplaca`

```
public abstract class Sexpr {
 public abstract void rplaca (Sexpr s) throws NotANodeException;
  // EFFECTS: rimpiazza in this il sottoalbero sinistro
  // con s. Se this non è un nodo binario solleva
  // NotANodeException (checked). Se s è indefinito,
  // solleva NullPointerException

public class Node extends Sexpr {
 public void rplaca (Sexpr s)
  // EFFECTS: rimpiazza in this il sottoalbero sinistro con s. Se s è indefinito,
  // solleva NullPointerException
  // Non solleva NotANodeException

public abstract class NilOrAtom extends Sexpr {
 public void rplaca (Sexpr s) throws NotANodeException
  // EFFECTS: Solleva NotANodeException
```

- ✓ ok, perché
  - in `Node`, this è sempre un nodo binario
  - in `NilOrAtom`, this non è mai un nodo binario

Confrontiamo le specifiche per il supertipo ed i sottotipi: `car`

```
public abstract class Sexpr {
 public abstract Sexpr car () throws NotANodeException;
  // EFFECTS: ritorna il sottoalbero sinistro di this.
  // Se this non è un nodo binario solleva
  // NotANodeException (checked)

public class Node extends Sexpr {
 public Sexpr car ()
  // EFFECTS: restituisce il sottoalbero sinistro di this.
  // Non solleva NotANodeException

public abstract class NilOrAtom extends Sexpr {
 public Sexpr car () throws NotANodeException
  // EFFECTS: solleva NotANodeException
```

- ✓ ok, perché
  - in `Node`, this è sempre un nodo binario
  - in `NilOrAtom`, this non è mai un nodo binario

Confrontiamo le specifiche per il supertipo ed i sottotipi: `getatom`

```
public abstract class Sexpr {
 public abstract String getatom () throws NotAnAtomException;
  // EFFECTS: Se this non è una foglia solleva
  // NotAnAtomException (checked). Altrimenti ritorna la stringa
  // contenuta nella foglia

public class Node extends Sexpr {
 public String getatom () throws NotAnAtomException
  // EFFECTS: Solleva NotAnAtomException

public class Nil extends NilOrAtom {
 public String getatom () throws NotAnAtomException
  // EFFECTS: Solleva NotAnAtomException

public class Atom extends NilOrAtom {
 public String getatom ()
  // EFFECTS: Restituisce la stringa contenuta nella foglia. Non solleva
  // NotAnAtomException
```

- ✓ ok, perché
  - in Node e in Nil, this non è mai una foglia
  - in `Atom`, this è sempre una foglia

Confrontiamo le specifiche per il supertipo ed i sottotipi: `nullS`

```
public abstract class Sexpr {
 public abstract boolean nullS() throws NullPointerException;
  // EFFECTS: ritorna true se this è l'albero vuoto,
  // altrimenti ritorna false. Se this è indefinito solleva
  // NullPointerException

public class Node extends Sexpr {
 public boolean nullS ()
  // EFFECTS: restituisce false

public class Nil extends NilOrAtom {
 public boolean nullS ()
 // EFFECTS: Ritorna true

public class Atom extends NilOrAtom {
 public boolean nullS ()
 // EFFECTS: Ritorna false
```

- ✓ ok, perché
  - in Node e in Atom, this non è mai un albero vuoto
  - in `Nil`, this è sempre un albero vuoto