Midterm Winter 2002

Karl Lieberherr

COM 1205

- has just been renamed from
  - Software Design and Development to
  - Advanced Pattern Matching and Brain Conditioning
- motivation:
  - software development will change, but
  - your pattern matching skills in your brain will be useful for the rest of your lives

Pattern matching

- Our brains are good at matching patterns
- recognize commonalities and differences
- sometimes have three-way and multi-way pattern matching

Three-way pattern matching

- Given two instances of a pattern, find the correspondences between the instances and the pattern
  - pattern = class dictionary
  - instance 1: sentence
  - instance 2: object

```
pointcut X1(S s1) : this(s);
```

sentence

```
AspectJFragment = <pointcuts> List(PointCut) EOF.
PointCut = "pointcut" PCName
  <lhs> PCList(ArgDecl) ":" <rhs> PCExp ";".
PCName = Ident.
PCList(S) ~ "(" S { "," S } ")". List(S) ~ {S}.
ArgDecl = <t> Type Variable.
Type = Ident.
Variable = Ident.
```

class dictionary

```
: AspectJFragment  (
<pointcuts> : UNKNOWN-PARSE1  {
<first> : Nonempty_PointCut_List  (
<it> : UNKNOWN-PARSE2  (
<pcname> : UNKNOWN-PARSE3  (
<ident>  : Ident "UNKNOWN-PARSE4" )
<lhs> : ArgDecl_PCList  {
<first> : Nonempty_ArgDecl_PCList  (
<it> : UNKNOWN-PARSE5  (
<t> : UNKNOWN-PARSE6  (
<ident>  : Ident "UNKNOWN-PARSE7" )
<variable> : UNKNOWN-PARSE8  (
<ident>  : Ident "UNKNOWN-PARSE9" ) ) ) }
```

object

```
pointcut X1(S s1) : this(s);
```

sentence

```
AspectJFragment = <pointcuts> List(PointCut) EOF.
PointCut = "pointcut" PCName
  <lhs> PCList(ArgDecl) ":" <rhs> PCExp ";".
PCName = Ident.
PCList(S) ~ "(" S { "," S } ")". List(S) ~ {S}.
ArgDecl = <t> Type Variable.
Type = Ident.
Variable = Ident.
```

class dictionary

```
: AspectJFragment  (
<pointcuts> : UNKNOWN-PARSE1  {
<first> : Nonempty_PointCut_List  (
<it> : UNKNOWN-PARSE2  (
<pcname> : UNKNOWN-PARSE3  (
<ident>  : Ident "UNKNOWN-PARSE4" )
<lhs> : ArgDecl_PCList  {
<first> : Nonempty_ArgDecl_PCList  (
<it> : UNKNOWN-PARSE5  (
<t> : UNKNOWN-PARSE6  (
<ident>  : Ident "UNKNOWN-PARSE7" )
<variable> : UNKNOWN-PARSE8  (
<ident>  : Ident "UNKNOWN-PARSE9" ) ) ) }
```

object

establish correspondence based on context knowledge

```
pointcut X1(S s1) : this(s);
```

sentence

```
AspectJFragment = <pointcuts> List(PointCut) EOF.
PointCut = "pointcut" PCName
  <lhs> PCList(ArgDecl) ":" <rhs> PCExp ";".
PCName = Ident.
PCList(S) ~ "(" S { "," S } ")". List(S) ~ {S}.
ArgDecl = <t> Type Variable.
Type = Ident.
Variable = Ident.
```

class dictionary

```
: AspectJFragment  (
<pointcuts> : UNKNOWN-PARSE1  {
<first> : Nonempty_PointCut_List  (
<it> : UNKNOWN-PARSE2  (
<pcname> : UNKNOWN-PARSE3  (
<ident>  : Ident "UNKNOWN-PARSE4" )
<lhs> : ArgDecl_PCList  {
<first> : Nonempty_ArgDecl_PCList  (
<it> : UNKNOWN-PARSE5  (
<t> : UNKNOWN-PARSE6  (
<ident>  : Ident "UNKNOWN-PARSE7" )
<variable> : UNKNOWN-PARSE8  (
<ident>  : Ident "UNKNOWN-PARSE9" ) ) ) }
```

object

establish correspondence based on context knowledge

```
pointcut X1(S s1) : this(s);
```

sentence

```
AspectJFragment = <pointcuts> List(PointCut) EOF.
PointCut = "pointcut" PCName
  <lhs> PCList(ArgDecl) ":" <rhs> PCExp ";".
PCName = Ident.
PCList(S) ~ "(" S { "," S } ")". List(S) ~ {S}.
ArgDecl = <t> Type Variable.
Type = Ident.
Variable = Ident.
```

class dictionary

```
: AspectJFragment  (
<pointcuts> : UNKNOWN-PARSE1  {
<first> : Nonempty_PointCut_List  (
<it> : UNKNOWN-PARSE2  (
<pcname> : UNKNOWN-PARSE3  (
<ident>  : Ident "UNKNOWN-PARSE4" )
<lhs> : ArgDecl_PCList  {
<first> : Nonempty_ArgDecl_PCList  (
<it> : UNKNOWN-PARSE5  (
<t> : UNKNOWN-PARSE6  (
<ident>  : Ident "UNKNOWN-PARSE7" )
<variable> : UNKNOWN-PARSE8  (
<ident>  : Ident "UNKNOWN-PARSE9" ) ) ) }
```

object

establish correspondence based on context knowledge

```
pointcut X1(S s1) : this(s);
```

sentence

```
AspectJFragment = <pointcuts> List(PointCut) EOF.
PointCut = "pointcut" PCName
  <lhs> PCList(ArgDecl) ":" <rhs> PCExp ";".
PCName = Ident.
PCList(S) ~ "(" S { "," S } ")". List(S) ~ {S}.
ArgDecl = <t> Type Variable.
Type = Ident.
Variable = Ident.
```

class dictionary

```
: AspectJFragment  (
<pointcuts> : UNKNOWN-PARSE1  {
<first> : Nonempty_PointCut_List  (
<it> : UNKNOWN-PARSE2  (
<pcname> : UNKNOWN-PARSE3  (
<ident>  : Ident "UNKNOWN-PARSE4" )
<lhs> : ArgDecl_PCList  {
<first> : Nonempty_ArgDecl_PCList  (
<it> : UNKNOWN-PARSE5  (
<t> : UNKNOWN-PARSE6  (
<ident>  : Ident "UNKNOWN-PARSE7" )
<variable> : UNKNOWN-PARSE8  (
<ident>  : Ident "UNKNOWN-PARSE9" ) ) ) }
```

object

establish correspondence based on context knowledge

```
pointcut X1(S s1) : this(s);
```

sentence

```
AspectJFragment = <pointcuts> List(PointCut) EOF.
PointCut = "pointcut" PCName
  <lhs> PCList(ArgDecl) ":" <rhs> PCExp ";".
PCName = Ident.
PCList(S) ~ "(" S { "," S } ")". List(S) ~ {S}.
ArgDecl = <t> Type Variable.
Type = Ident.
Variable = Ident.
```

class dictionary

```
: AspectJFragment  (
<pointcuts> : UNKNOWN-PARSE1  {
<first> : Nonempty_PointCut_List  (
<it> : UNKNOWN-PARSE2  (
<pcname> : UNKNOWN-PARSE3  (
<ident>  : Ident "UNKNOWN-PARSE4" )
<lhs> : ArgDecl_PCList  {
<first> : Nonempty_ArgDecl_PCList  (
<it> : UNKNOWN-PARSE5  (
<t> : UNKNOWN-PARSE6  (
<ident>  : Ident "UNKNOWN-PARSE7" )
<variable> : UNKNOWN-PARSE8  (
<ident>  : Ident "UNKNOWN-PARSE9" ) ) ) }
```

object

establish correspondence based on context knowledge

```
pointcut X1(S s1) : this(s);
```

sentence

```
AspectJFragment = <pointcuts> List(PointCut) EOF.
PointCut = "pointcut" PCName
  <lhs> PCList(ArgDecl) ":" <rhs> PCExp ";".
PCName = Ident.
PCList(S) ~ "(" S { "," S } ")". List(S) ~ {S}.
ArgDecl = <t> Type Variable.
Type = Ident.
Variable = Ident.
```

class dictionary

```
: AspectJFragment  (
<pointcuts> : UNKNOWN-PARSE1  {
<first> : Nonempty_PointCut_List  (
<it> : UNKNOWN-PARSE2  (
<pcname> : UNKNOWN-PARSE3  (
<ident>  : Ident "UNKNOWN-PARSE4" )
<lhs> : ArgDecl_PCList  {
<first> : Nonempty_ArgDecl_PCList  (
<it> : UNKNOWN-PARSE5  (
<t> : UNKNOWN-PARSE6  (
<ident>  : Ident "UNKNOWN-PARSE7" )
<variable> : UNKNOWN-PARSE8  (
<ident>  : Ident "UNKNOWN-PARSE9" ) ) ) }
```

object

establish correspondence based on context knowledge

```
pointcut X1(S s1) : this(s);
```

sentence

```
AspectJFragment = <pointcuts> List(PointCut) EOF.
PointCut = "pointcut" PCName
  <lhs> PCList(ArgDecl) ":" <rhs> PCExp ";".
PCName = Ident.
PCList(S) ~ "(" S { "," S } ")". List(S) ~ {S}.
ArgDecl = <t> Type Variable.
Type = Ident.
Variable = Ident.
```

class dictionary

```
: AspectJFragment  (
<pointcuts> : UNKNOWN-PARSE1  {
<first> : Nonempty_PointCut_List  (
<it> : UNKNOWN-PARSE2  (
<pcname> : UNKNOWN-PARSE3  (
<ident>  : Ident "UNKNOWN-PARSE4" )
<lhs> : ArgDecl_PCList  {
<first> : Nonempty_ArgDecl_PCList  (
<it> : UNKNOWN-PARSE5  (
<t> : UNKNOWN-PARSE6  (
<ident>  : Ident "UNKNOWN-PARSE7" )
<variable> : UNKNOWN-PARSE8  (
<ident>  : Ident "UNKNOWN-PARSE9" ) ) ) }
```

object

establish correspondence based on context knowledge

```
pointcut X1(S s1) : this(s);
```

sentence

```
AspectJFragment = <pointcuts> List(PointCut) EOF.
PointCut = "pointcut" PCName
  <lhs> PCList(ArgDecl) ":" <rhs> PCExp ";".
PCName = Ident.
PCList(S) ~ "(" S { "," S } ")". List(S) ~ {S}.
ArgDecl = <t> Type Variable.
Type = Ident.
Variable = Ident.
```

class dictionary

```
: AspectJFragment  (
<pointcuts> : UNKNOWN-PARSE1  {
<first> : Nonempty_PointCut_List  (
<it> : UNKNOWN-PARSE2  (
<pcname> : UNKNOWN-PARSE3  (
<ident>  : Ident "UNKNOWN-PARSE4" )
<lhs> : ArgDecl_PCList  {
<first> : Nonempty_ArgDecl_PCList  (
<it> : UNKNOWN-PARSE5  (
<t> : UNKNOWN-PARSE6  (
<ident>  : Ident "UNKNOWN-PARSE7" )
<variable> : UNKNOWN-PARSE8  (
<ident>  : Ident "UNKNOWN-PARSE9" ) ) ) }
```

object

establish correspondence based on context knowledge

```
pointcut X1(S s1) : this(s);
```

sentence

```
AspectJFragment = <pointcuts> List(PointCut) EOF.
PointCut = "pointcut" PCName
  <lhs> PCList(ArgDecl) ":" <rhs> PCExp ";".
PCName = Ident.
PCList(S) ~ "(" S { "," S } ")". List(S) ~ {S}.
ArgDecl = <t> Type Variable.
Type = Ident.
Variable = Ident.
```

class dictionary

```
: AspectJFragment  (
<pointcuts> : UNKNOWN-PARSE1  {
<first> : Nonempty_PointCut_List  (
<it> : UNKNOWN-PARSE2  (
<pcname> : UNKNOWN-PARSE3  (
<ident>  : Ident "UNKNOWN-PARSE4" )
<lhs> : ArgDecl_PCList  {
<first> : Nonempty_ArgDecl_PCList  (
<it> : UNKNOWN-PARSE5  (
<t> : UNKNOWN-PARSE6  (
<ident>  : Ident "UNKNOWN-PARSE7" )
<variable> : UNKNOWN-PARSE8  (
<ident>  : Ident "UNKNOWN-PARSE9" ) ) ) }
```

object

establish correspondence based on context knowledge

```
pointcut X1(S s1) : this(s);
```

sentence

```
AspectJFragment = <pointcuts> List(PointCut) EOF.
PointCut = "pointcut" PCName
  <lhs> PCList(ArgDecl) ":" <rhs> PCExp ";".
PCName = Ident.
PCList(S) ~ "(" S { "," S } ")". List(S) ~ {S}.
ArgDecl = <t> Type Variable.
Type = Ident.
Variable = Ident.
```

class dictionary

```
: AspectJFragment  (
<pointcuts> : UNKNOWN-PARSE1  {
<first> : Nonempty_PointCut_List  (
<it> : UNKNOWN-PARSE2  (
<pcname> : UNKNOWN-PARSE3  (
<ident>  : Ident "UNKNOWN-PARSE4" )
<lhs> : ArgDecl_PCList  {
<first> : Nonempty_ArgDecl_PCList  (
<it> : UNKNOWN-PARSE5  (
<t> : UNKNOWN-PARSE6  (
<ident>  : Ident "UNKNOWN-PARSE7" )
<variable> : UNKNOWN-PARSE8  (
<ident>  : Ident "UNKNOWN-PARSE9" ) ) ) }
```

object

establish correspondence based on context knowledge

```
pointcut X1(S s1) : this(s);
```

sentence

```
AspectJFragment = <pointcuts> List(PointCut) EOF.
PointCut = "pointcut" PCName
  <lhs> PCList(ArgDecl) ":" <rhs> PCExp ";".
PCName = Ident.
PCList(S) ~ "(" S { "," S } ")". List(S) ~ {S}.
ArgDecl = <t> Type Variable.
Type = Ident.
Variable = Ident.
```

class dictionary

```
: AspectJFragment  (
<pointcuts> : UNKNOWN-PARSE1  {
<first> : Nonempty_PointCut_List  (
<it> : UNKNOWN-PARSE2  (
<pcname> : UNKNOWN-PARSE3  (
<ident>  : Ident "UNKNOWN-PARSE4" )
<lhs> : ArgDecl_PCList  {
<first> : Nonempty_ArgDecl_PCList  (
<it> : UNKNOWN-PARSE5  (
<t> : UNKNOWN-PARSE6  (
<ident>  : Ident "UNKNOWN-PARSE7" )
<variable> : UNKNOWN-PARSE8  (
<ident>  : Ident "UNKNOWN-PARSE9" ) ) ) }
```

establish correspondence

object

establish correspondence using colors

```
pointcut X1(S s1) : this(s);
```

sentence

```
AspectJFragment = <pointcuts> List(PointCut) EOF.
PointCut = "pointcut" PCName
  <lhs> PCList(ArgDecl) ":" <rhs> PCExp ";".
PCName = Ident.
PCList(S) ~ "(" S { "," S } ")". List(S) ~ {S}.
ArgDecl = <t> Type Variable.
Type = Ident.
Variable = Ident.
```

class dictionary

```
: AspectJFragment  (
<pointcuts> : UNKNOWN-PARSE1  {
<first> : Nonempty_PointCut_List  (
<it> : UNKNOWN-PARSE2  (
<pcname> : UNKNOWN-PARSE3  (
<ident>  : Ident "UNKNOWN-PARSE4" )
<lhs> : ArgDecl_PCList  {
<first> : Nonempty_ArgDecl_PCList  (
<it> : UNKNOWN-PARSE5  (
<t> : UNKNOWN-PARSE6  (
<ident>  : Ident "UNKNOWN-PARSE7" )
<variable> : UNKNOWN-PARSE8  (
<ident>  : Ident "UNKNOWN-PARSE9" ) ) ) }
```

establish correspondence

object

```
pointcut X1(S s1) : this(s);
```

sentence

```
AspectJFragment = <pointcuts> List(PointCut) EOF.
PointCut = "pointcut" PCName
  <lhs> PCList(ArgDecl) ":" <rhs> PCExp ";".
PCName = Ident.
PCList(S) ~ "(" S { "," S } ")". List(S) ~ {S}.
ArgDecl = <t> Type Variable.
Type = Ident.
Variable = Ident.
```

class dictionary

```
: AspectJFragment  (
<pointcuts> : UNKNOWN-PARSE1  {
<first> : Nonempty_PointCut_List  (
<it> : UNKNOWN-PARSE2  (
<pcname> : UNKNOWN-PARSE3  (
<ident>  : Ident "UNKNOWN-PARSE4" )
<lhs> : ArgDecl_PCList  {
<first> : Nonempty_ArgDecl_PCList  (
<it> : UNKNOWN-PARSE5  (
<t> : UNKNOWN-PARSE6  (
<ident>  : Ident "UNKNOWN-PARSE7" )
<variable> : UNKNOWN-PARSE8  (
<ident>  : Ident "UNKNOWN-PARSE9" ) ) ) }
```

establish correspondence

object

```
pointcut X1(S s1) : this(s);
```

sentence

```
AspectJFragment = <pointcuts> List(PointCut) EOF.
PointCut = "pointcut" PCName
  <lhs> PCList(ArgDecl) ":" <rhs> PCExp ";".
PCName = Ident.
PCList(S) ~ "(" S { "," S } ")". List(S) ~ {S}.
ArgDecl = <t> Type Variable.
Type = Ident.
Variable = Ident.
```

class dictionary

```
: AspectJFragment  (
<pointcuts> : UNKNOWN-PARSE1  {
<first> : Nonempty_PointCut_List  (
<it> : UNKNOWN-PARSE2  (
<pcname> : UNKNOWN-PARSE3  (
<ident>  : Ident "UNKNOWN-PARSE4" )
<lhs> : ArgDecl_PCList  {
<first> : Nonempty_ArgDecl_PCList  (
<it> : UNKNOWN-PARSE5  (
<t> : UNKNOWN-PARSE6  (
<ident>  : Ident "UNKNOWN-PARSE7" )
<variable> : UNKNOWN-PARSE8  (
<ident>  : Ident "UNKNOWN-PARSE9" ) ) ) }
```

establish correspondence

object

```
pointcut X1(S s1) : this(s);
```

sentence

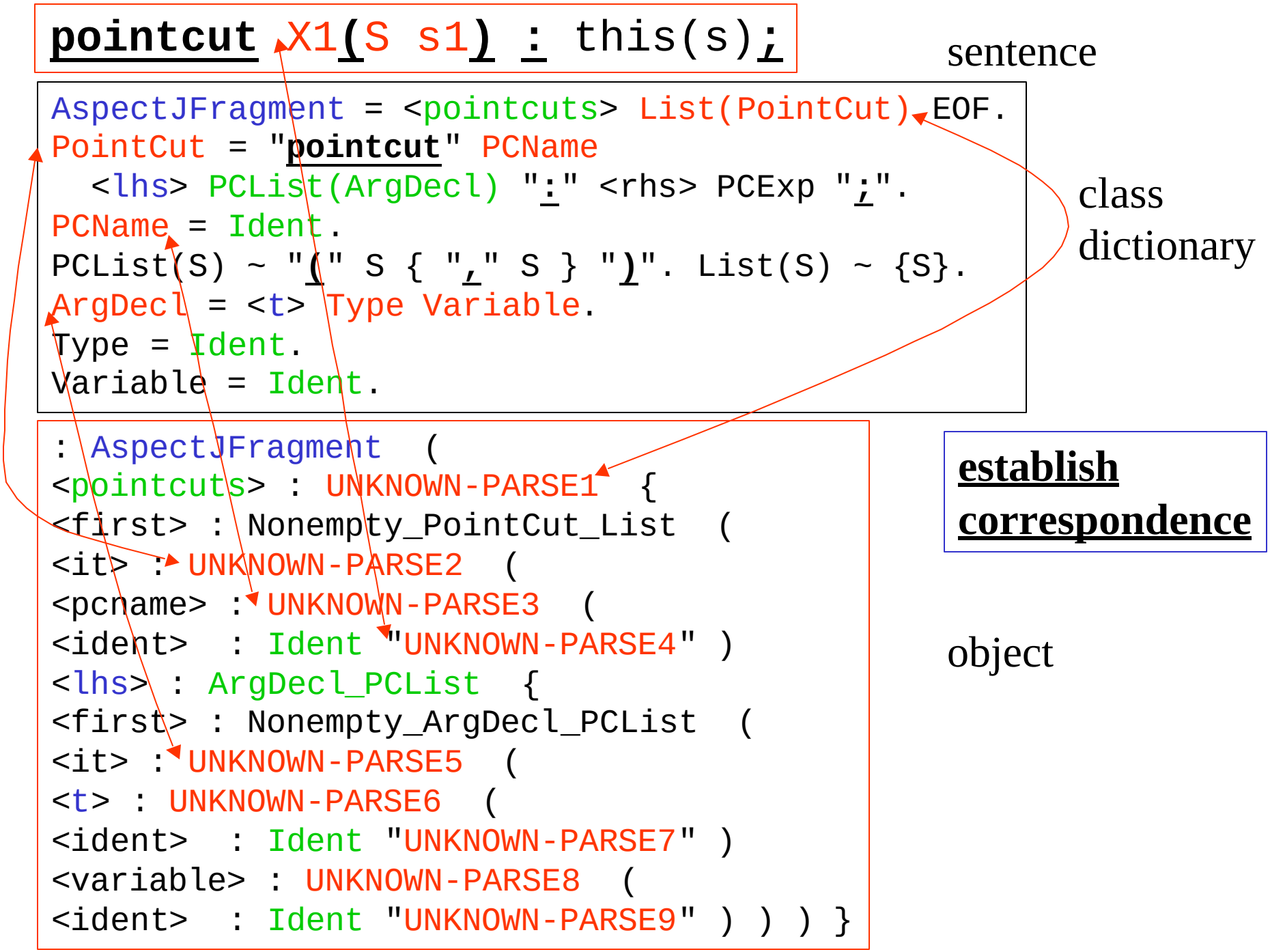
```
AspectJFragment = <pointcuts> List(PointCut) EOF.
PointCut = "pointcut" PCName
  <lhs> PCList(ArgDecl) ":" <rhs> PCExp ";".
PCName = Ident.
PCList(S) ~ "(" S { "," S } ")". List(S) ~ {S}.
ArgDecl = <t> Type Variable.
Type = Ident.
Variable = Ident.
```

class dictionary

```
: AspectJFragment  (
<pointcuts> : UNKNOWN-PARSE1  {
<first> : Nonempty_PointCut_List  (
<it> : UNKNOWN-PARSE2  (
<pcname> : UNKNOWN-PARSE3  (
<ident>  : Ident "UNKNOWN-PARSE4" )
<lhs> : ArgDecl_PCList  {
<first> : Nonempty_ArgDecl_PCList  (
<it> : UNKNOWN-PARSE5  (
<t> : UNKNOWN-PARSE6  (
<ident>  : Ident "UNKNOWN-PARSE7" )
<variable> : UNKNOWN-PARSE8  (
<ident>  : Ident "UNKNOWN-PARSE9" ) ) ) }
```

establish correspondence

object


pointcut X1(S s1) : this(s);
sentence
AspectJFragment = <pointcuts> List(PointCut) EOF.
PointCut = "pointcut" PCName
<lhs> PCList(ArgDecl) ":" <rhs> PCExp ";".
PCName = Ident.
PCList(S) ~ "(" S { "," S } ")". List(S) ~ {S}.
ArgDecl = <t> Type Variable.
Type = Ident.
Variable = Ident.
class
dictionary
: AspectJFragment (
<pointcuts> : UNKNOWN-PARSE1 {
<first> : Nonempty_PointCut_List (
<it> : UNKNOWN-PARSE2 (
<pcname> : UNKNOWN-PARSE3 (
<ident> : Ident "UNKNOWN-PARSE4")
<lhs> : ArgDecl_PCList {
<first> : Nonempty_ArgDecl_PCList (
<it> : UNKNOWN-PARSE5 (
<t> : UNKNOWN-PARSE6 (
<ident> : Ident "UNKNOWN-PARSE7")
<variable> : UNKNOWN-PARSE8 (
<ident> : Ident "UNKNOWN-PARSE9"))) }
establish
correspondence
object

```
pointcut X1(S s1) : this(s);
```

sentence

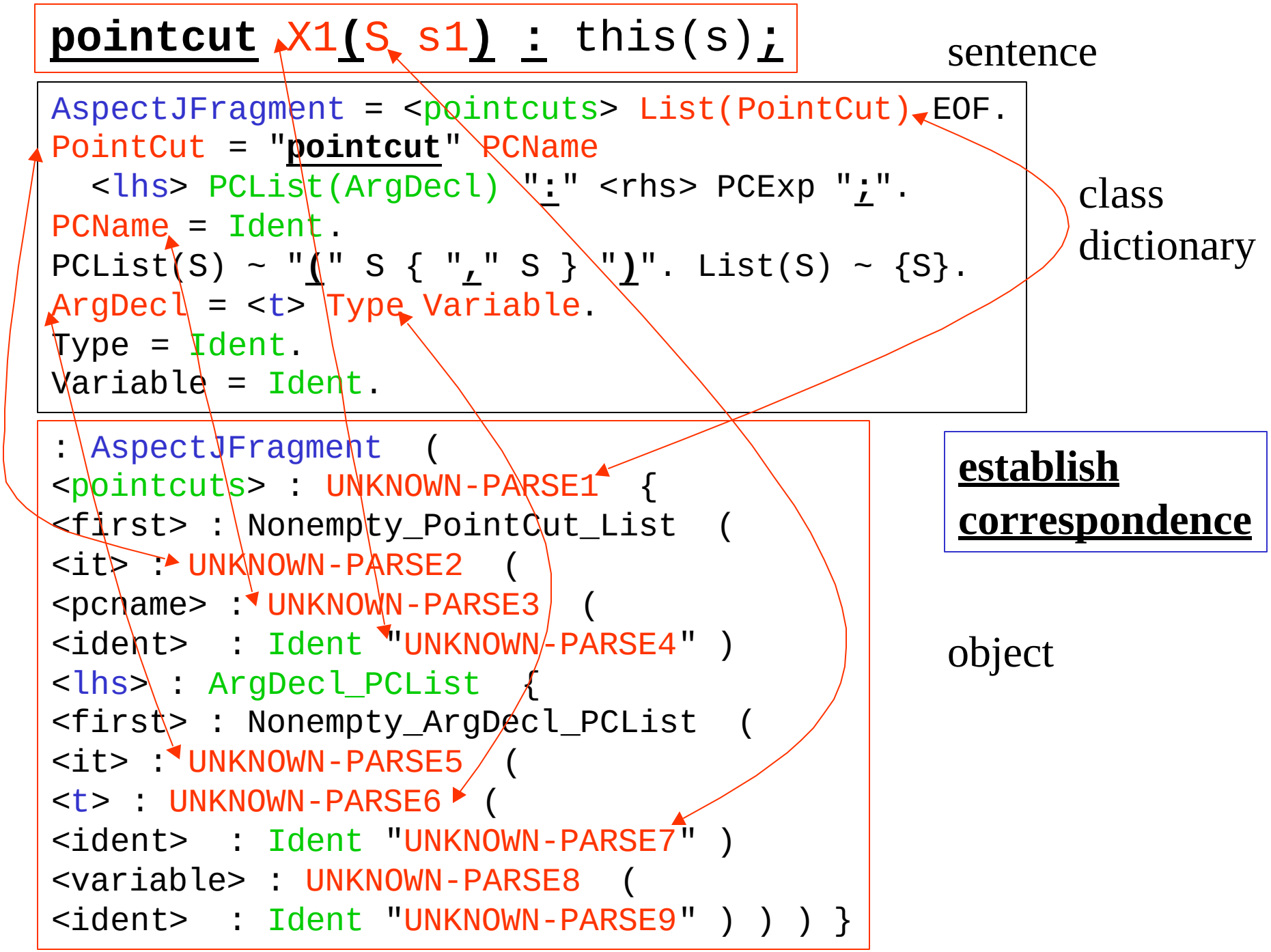
```
AspectJFragment = <pointcuts> List(PointCut) EOF.
PointCut = "pointcut" PCName
  <lhs> PCList(ArgDecl) ":" <rhs> PCExp ";".
PCName = Ident.
PCList(S) ~ "(" S { "," S } ")". List(S) ~ {S}.
ArgDecl = <t> Type Variable.
Type = Ident.
Variable = Ident.
```

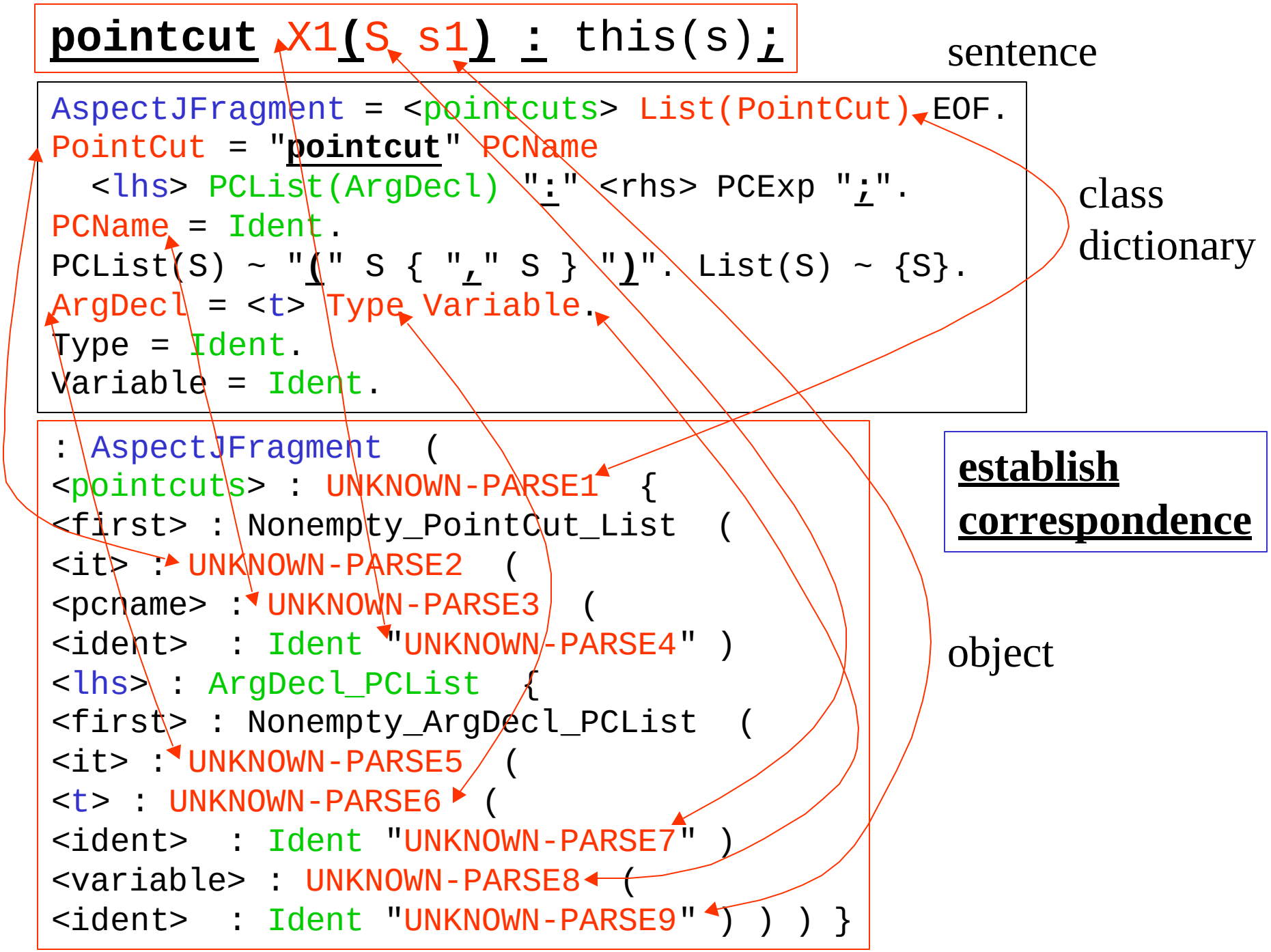
class dictionary

```
: AspectJFragment  (
<pointcuts> : UNKNOWN-PARSE1  {
<first> : Nonempty_PointCut_List  (
<it> : UNKNOWN-PARSE2  (
<pcname> : UNKNOWN-PARSE3  (
<ident>  : Ident "UNKNOWN-PARSE4" )
<lhs> : ArgDecl_PCList  {
<first> : Nonempty_ArgDecl_PCList  (
<it> : UNKNOWN-PARSE5  (
<t> : UNKNOWN-PARSE6 (
<ident>  : Ident "UNKNOWN-PARSE7" )
<variable> : UNKNOWN-PARSE8  (
<ident>  : Ident "UNKNOWN-PARSE9" ) ) ) }
```

establish correspondence

object


pointcut X1(S s1) : this(s);
sentence
AspectJFragment = <pointcuts> List(PointCut) EOF.
PointCut = "pointcut" PCName
<lhs> PCList(ArgDecl) ":" <rhs> PCExp ";".
PCName = Ident.
PCList(S) ~ "(" S { "," S } ")". List(S) ~ {S}.
ArgDecl = <t> Type Variable.
Type = Ident.
Variable = Ident.
class
dictionary
: AspectJFragment (
<pointcuts> : UNKNOWN-PARSE1 {
<first> : Nonempty_PointCut_List (
<it> : UNKNOWN-PARSE2 (
<pcname> : UNKNOWN-PARSE3 (
<ident> : Ident "UNKNOWN-PARSE4")
<lhs> : ArgDecl_PCList {
<first> : Nonempty_ArgDecl_PCList (
<it> : UNKNOWN-PARSE5 (
<t> : UNKNOWN-PARSE6 (
<ident> : Ident "UNKNOWN-PARSE7")
<variable> : UNKNOWN-PARSE8 (
<ident> : Ident "UNKNOWN-PARSE9"))) }
establish
correspondence
object


pointcut X1(S s1) : this(s);
sentence
AspectJFragment = <pointcuts> List(PointCut) EOF.
PointCut = "pointcut" PCName
<lhs> PCList(ArgDecl) ":" <rhs> PCExp ";".
PCName = Ident.
PCList(S) ~ "(" S { "," S } ")". List(S) ~ {S}.
ArgDecl = <t> Type Variable.
Type = Ident.
Variable = Ident.
class
dictionary
: AspectJFragment (
<pointcuts> : UNKNOWN-PARSE1 {
<first> : Nonempty_PointCut_List (
<it> : UNKNOWN-PARSE2 (
<pcname> : UNKNOWN-PARSE3 (
<ident> : Ident "UNKNOWN-PARSE4")
<lhs> : ArgDecl_PCList {
<first> : Nonempty_ArgDecl_PCList (
<it> : UNKNOWN-PARSE5 (
<t> : UNKNOWN-PARSE6 (
<ident> : Ident "UNKNOWN-PARSE7")
<variable> : UNKNOWN-PARSE8 (
<ident> : Ident "UNKNOWN-PARSE9"))) }
establish
correspondence
object

```
pointcut X1(S s1) : this(s);
```

sentence

```
AspectJFragment = <pointcuts> List(PointCut) EOF.
PointCut = "pointcut" PCName
  <lhs> PCList(ArgDecl) ":" <rhs> PCExp ";".
PCName = Ident.
PCList(S) ~ "(" S { "," S } ")". List(S) ~ {S}.
ArgDecl = <t> Type Variable.
Type = Ident.
Variable = Ident.
```

class dictionary

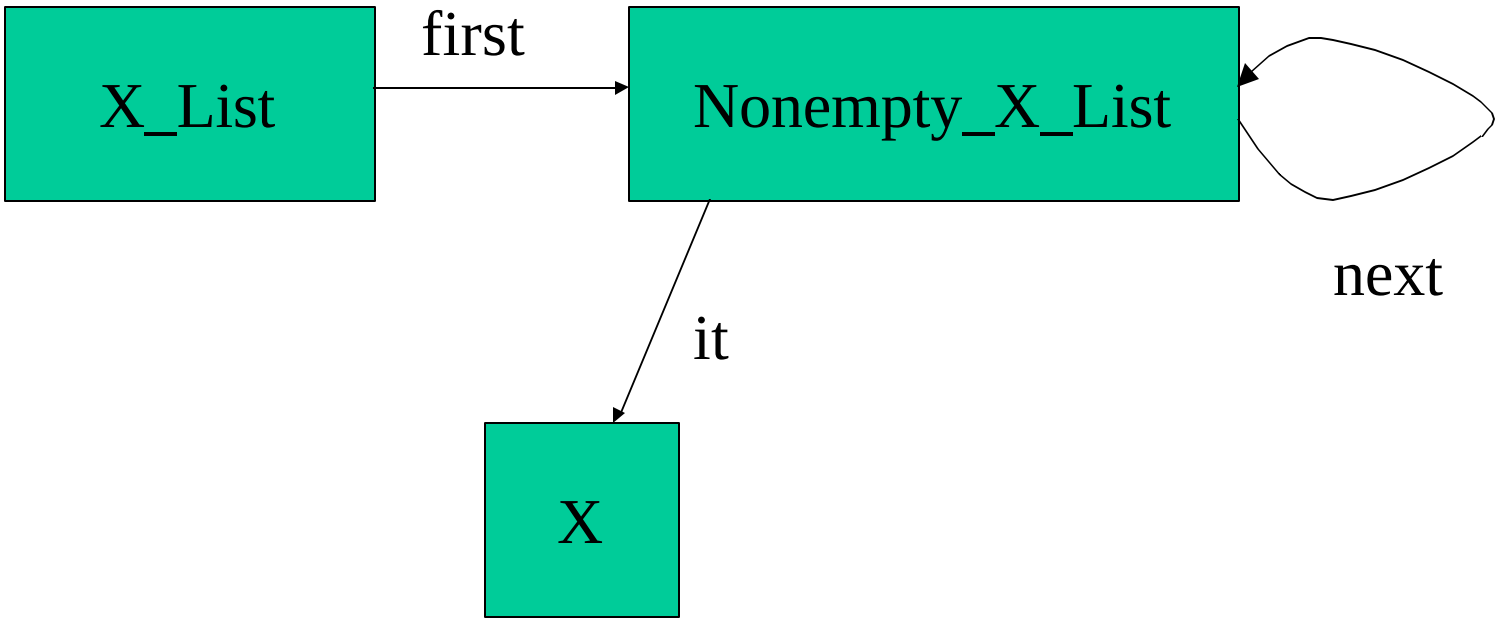
object

```
: AspectJFragment  (
<pointcuts> : UNKNOWN-PARSE1  {
<first> : Nonempty_PointCut_List  (
<it> : UNKNOWN-PARSE2  (
<pcname> : UNKNOWN-PARSE3  (
<ident>  : Ident "UNKNOWN-PARSE4" )
<lhs> : ArgDecl_PCList  {
<first> : Nonempty_ArgDecl_PCList  (
<it> : UNKNOWN-PARSE5  (
<t> : UNKNOWN-PARSE6  (
<ident>  : Ident "UNKNOWN-PARSE7" )
<variable> : UNKNOWN-PARSE8  (
<ident>  : Ident "UNKNOWN-PARSE9" ) ) ) }
```

X_List = <first> Nonempty_X_List.
Nonempty_X_List =
 <it> X
 [<next> Nonempty_X_List].

For phases 3 and 4: List Structure

- List(X)



```
pointcut UNKNOWN-PRINT1
  (A UNKNOWN-PRINT2,
   B UNKNOWN-PRINT3, C c3) :
```

sentence

```
PointCut = "pointcut" PCName
  <lhs> PCList(ArgDecl) ":" <rhs> PCExp ";".
PCName = Ident.
PCList(S) ~ "(" S { "," S } ")".
ArgDecl = <t> Type Variable.
Type = Ident.
Variable = Ident.
```

class dictionary

```
<it> : PointCut  (
<pcname> : PCName  (
<ident>  : Ident "X3" )
<lhs> : ArgDecl_PCList  {
<first> : Nonempty_ArgDecl_PCList  (
<it> : ArgDecl  (
<t> : Type  (
<ident>  : Ident "A" )
<variable> : Variable  (
<ident>  : Ident "a1" ) )
<next> : Nonempty_ArgDecl_PCList  (
```

object

```
pointcut UNKNOWN-PRINT1
  (A UNKNOWN-PRINT2,
   B UNKNOWN-PRINT3, C c3) :
```

sentence

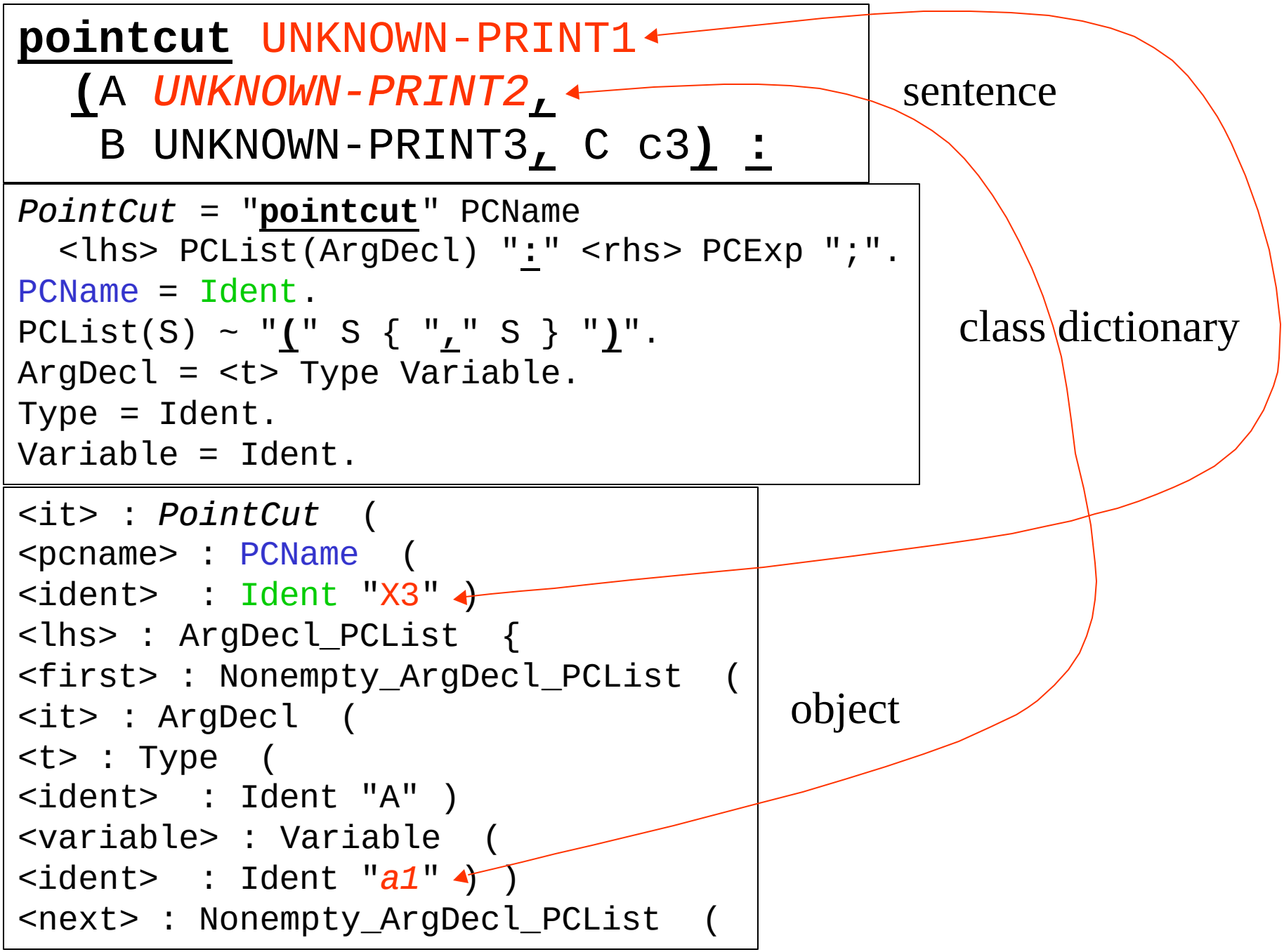
```
PointCut = "pointcut" PCName
  <lhs> PCList(ArgDecl) ":" <rhs> PCExp ";".
PCName = Ident.
PCList(S) ~ "(" S { "," S } ")".
ArgDecl = <t> Type Variable.
Type = Ident.
Variable = Ident.
```

class dictionary

```
<it> : PointCut  (
<pcname> : PCName  (
<ident>  : Ident "X3" )
<lhs> : ArgDecl_PCList  {
<first> : Nonempty_ArgDecl_PCList  (
<it> : ArgDecl  (
<t> : Type  (
<ident>  : Ident "A" )
<variable> : Variable  (
<ident>  : Ident "a1" ) )
<next> : Nonempty_ArgDecl_PCList  (
```

establish correspondence

object


pointcut UNKNOWN-PRINT1
(A UNKNOWN-PRINT2,
B UNKNOWN-PRINT3, C c3) :
sentence
PointCut = "pointcut" PCName
<lhs> PCList(ArgDecl) ":" <rhs> PCExp ";".
PCName = Ident.
PCList(S) ~ "(" S { "," S } ")".
ArgDecl = <t> Type Variable.
Type = Ident.
Variable = Ident.
class dictionary
<it> : PointCut (
<pcname> : PCName (
<ident> : Ident "X3")
<lhs> : ArgDecl_PCList {
<first> : Nonempty_ArgDecl_PCList (
<it> : ArgDecl (
<t> : Type (
<ident> : Ident "A")
<variable> : Variable (
<ident> : Ident "a1"))
<next> : Nonempty_ArgDecl_PCList (
object

Two-way pattern matching

- Given a pattern and and a use of it, find correspondences between pattern and usage.
  - pattern = method
  - usage = inlined code

```
    Iterator it = used.iterator();
    while (it.hasNext()){
       Variable curvar = (Variable) it.next();
       System.out.println(); curvar.print();
    }
```

```
static UNKNOWN-ABS3 show(UNKNOWN-ABS4) {
    UNKNOWN-ABS5
    while (it.hasNext()){
       UNKNOWN-ABS6
       System.out.println(); curvar.print();
    }
```

```
Iterator it = used.iterator();
while (it.hasNext()){
    Variable curvar = (Variable) it.next();
    System.out.println(); curvar.print();
}
```

```
static UNKNOWN-ABS3 show(UNKNOWN-ABS4) {
    UNKNOWN-ABS5
    while (it.hasNext()){
        UNKNOWN-ABS6
        System.out.println(); curvar.print();
    }
```

establish correspondence based on context knowledge

```
Iterator it = used.iterator();
while (it.hasNext()){
    Variable curvar = (Variable) it.next();
    System.out.println(); curvar.print();
}
```

```
static UNKNOWN-ABS3 show(UNKNOWN-ABS4) {
    UNKNOWN-ABS5
    while (it.hasNext()){
        UNKNOWN-ABS6
        System.out.println(); curvar.print();
    }
```

establish correspondence based on context knowledge

```
Iterator it = used.iterator();
while (it.hasNext()){
    Variable curvar = (Variable) it.next();
    System.out.println(); curvar.print();
}
```

```
static UNKNOWN-ABS3 show(UNKNOWN-ABS4) {
    UNKNOWN-ABS5
    while (it.hasNext()){
        UNKNOWN-ABS6
        System.out.println(); curvar.print();
    }
```

establish correspondence based on context knowledge

```
Iterator it = used.iterator();
while (it.hasNext()){
    Variable curvar = (Variable) it.next();
    System.out.println(); curvar.print();
}
```

```
void                    java.util.Collection used
static UNKNOWN-ABS3 show(UNKNOWN-ABS4) {
    UNKNOWN-ABS5
    while (it.hasNext()){
        UNKNOWN-ABS6
        System.out.println(); curvar.print();
    }
```

establish correspondence

make method as general as possible! Who supports iterator()?

Two-way pattern matching

- Given a pattern and and a use of it, find correspondences between pattern and usage.
  - pattern = a programming pattern
  - usage = an application in a specific context

```
AspectJFragment {
{{
  // method process finds all Compound-objects inside
  // an AspectJFragment-object and prints the message:
  // "This is a Compound expression:  "
  // for each Compound-object.
  void process() {
    UNKNOWN-AP1;
    Main.cg.UNKNOWN-AP2(UNKNOWN-AP3, UNKNOWN-AP4, cV);
  }
}}
}
```

CommandVisitor cV = new CommandVisitor();

this

"from AspectJFragment to Compound"

```
AspectJFragment {
{{
  // method process finds all Compound-objects inside
  // an AspectJFragment-object and prints the message:
  // "This is a Compound expression:  "
  // for each Compound-object.
  void process() {
    UNKNOWN-AP1;
    Main.cg.UNKNOWN-AP2(UNKNOWN-AP3, UNKNOWN-AP4, cV);
  }
}}
}
```

CommandVisitor cV = new CommandVisitor();

this

"from AspectJFragment to Compound"

DJ programming pattern:

class-graph.traverse(what, where-to-go, what-to-do)

what = an object

where-to-go = a traversal strategy

what-to-do = a visitor

```
CommandVisitor {
{{
  // prints the message:
  // "This is a Compound expression:  "
  // for each Compound-object.
   void UNKNOWN-AP5(UNKNOWN-AP6){
     System.out.println("This is a Compound expression:  ");
     host.print();
     System.out.println();
   }
}}
}
```

```
CommandVisitor {
{{
  // prints the message:
  // "This is a Compound expression:  "
  // for each Compound-object.
   void UNKNOWN-AP5(UNKNOWN-AP6){
     System.out.println("This is a Compound expression:  ");
     host.print();
     System.out.println();
   }
}}
}
```

Compound host

```
Selective visitor pattern:
Visitor {
{{
  void before (X host) { … use host … };
  void after  (X host) { … use host … };
}}
}
```

establish correspondence

```
AspectJFragment {
{{
  // Find all Variable-objects on the left-hand-side
  // (before the colon) of PointCut-objects
  static String definedVarsSpec =
```


```
Answer:
from AspectJFragment via PointCut via -> *,lhs,* to Variable";
```

```
AspectJFragment = <pointcuts> List(PointCut).
PointCut = "pointcut" PCName
  <lhs> PCList(ArgDecl) ":" <rhs> PCExp ";".
PCName = Ident.
PCList(S) ~ "(" S { "," S } ")". List(S) ~ {S}.
ArgDecl = <t> Type Variable.
Type = Ident.
Variable = Ident.
```

```
AspectJFragment {
{{
  // Find all Variable-objects on the left-hand-side
  // (before the colon) of PointCut-objects
  static String definedVarsSpec =
```


```
Answer:
from AspectJFragment via PointCut via -> *,lhs,* to Variable";
```

```
AspectJFragment = <pointcuts> List(PointCut).
PointCut = "pointcut" PCName
  <lhs> PCList(ArgDecl) ":" <rhs> PCExp ";".
PCName = Ident.
PCList(S) ~ "(" S { "," S } ")". List(S) ~ {S}.
ArgDecl = <t> Type Variable.
Type = Ident.
Variable = Ident.
```

```
AspectJFragment {
{{
  // Find all Variable-objects on the left-hand-side
  // (before the colon) of PointCut-objects
  static String definedVarsSpec =
```


```
Answer:
from AspectJFragment via PointCut via -> *,lhs,* to Variable";
```

```
AspectJFragment = <pointcuts> List(PointCut).
PointCut = "pointcut" PCName
  <lhs> PCList(ArgDecl) ":" <rhs> PCExp ";".
PCName = Ident.
PCList(S) ~ "(" S { "," S } ")". List(S) ~ {S}.
ArgDecl = <t> Type Variable.
Type = Ident.
Variable = Ident.
```

```
AspectJFragment {
{{
  // Find all Variable-objects on the left-hand-side
  // (before the colon) of PointCut-objects
  static String definedVarsSpec =
```


```
Answer:
from AspectJFragment via PointCut via -> *,lhs,* to Variable";
```

```
AspectJFragment = <pointcuts> List(PointCut).
PointCut = "pointcut" PCName
  <lhs> PCList(ArgDecl) ":" <rhs> PCExp ",".
PCName = Ident.
PCList(S) ~ "(" S { "," S } ")". List(S) ~ {S}.
ArgDecl = <t> Type Variable.
Type = Ident.
Variable = Ident.
```

```
AspectJFragment {
{{
  // Find all Variable-objects on the left-hand-side
  // (before the colon) of PointCut-objects
  static String definedVarsSpec =
```

UNKNOWN-AP7

```
Answer:
from AspectJFragment via PointCut via -> *,lhs,* to Variable";
```

Traversal strategy pattern: from A via X via Y to B

```
AspectJFragment = <pointcuts> List(PointCut).
PointCut = "pointcut" PCName
  <lhs> PCList(ArgDecl) ":" <rhs> PCExp ";".
PCName = Ident.
PCList(S) ~ "(" S { "," S } ")". List(S) ~ {S}.
ArgDecl = <t> Type Variable.
Type = Ident.
Variable = Ident.
```

touched by traversal

```
// using an inlined visitor
    UNKNOWN-AP8 v = UNKNOWN-AP9 {
      // construct a new empty set
      HashSet return_val = new HashSet();
      void UNKNOWN-AP10(Variable v1) {return_val.add(v1);}
      public Object getReturnValue() {return return_val;}
    };
}
```

```
// using an inlined visitor
    UNKNOWN-AP8 v = UNKNOWN-AP9 {
       // construct a new empty set
       HashSet return_val = new HashSet();
       void UNKNOWN-AP10(Variable v1) {return_val.add(v1);}
       public Object getReturnValue() {return return_val;}
    };
}
```

```
Selective visitor pattern (inlined):
Visitor v = new Visitor {
  // local variable declarations/definitions
  void before (X host) { … use host … };
  void after  (X host) { … use host … };
  public Object getReturnValue(){return … ;}
}
```

```
// in AspectJFragment
  HashSet collectVars(String travspec){
    // using an inlined visitor
    UNKNOWN-AP8 v = UNKNOWN-AP9 {
      // construct a new empty set
      HashSet return_val = new HashSet();
      void UNKNOWN-AP10(Variable v1) {return_val.add(v1);}
      public Object getReturnValue(){return return_val;}
    };
    return (HashSet) Main.cg.UNKNOWN-AP11(UNKNOWN-AP12,
      UNKNOWN-AP13, UNKNOWN-AP14);
 }
```

this travspec **v**

```
// in AspectJFragment
  HashSet collectVars(String travspec){
    // using an inlined visitor
    UNKNOWN-AP8 v = UNKNOWN-AP9 {
      // construct a new empty set
      HashSet return_val = new HashSet();
      void UNKNOWN-AP10(Variable v1) {return_val.add(v1);}
      public Object getReturnValue(){return return_val;}
    };
    return (HashSet) Main.cg.UNKNOWN-AP11(UNKNOWN-AP12,
      UNKNOWN-AP13, UNKNOWN-AP14);
 }
```


```
DJ programming pattern:
class-graph.traverse(what, where-to-go, what-to-do)
```

Conclusions

- Now is the time to learn those pattern matching skills
- Use color pens to help your brain detect correspondences
- Final exam will exercise similar skills
- You will exercise those skills in hw 5 and in the project