Recognising structural patterns in code

BSc project by Mathias Fuchs,

supervised by: Jan Kurš and Haidar Osman

{

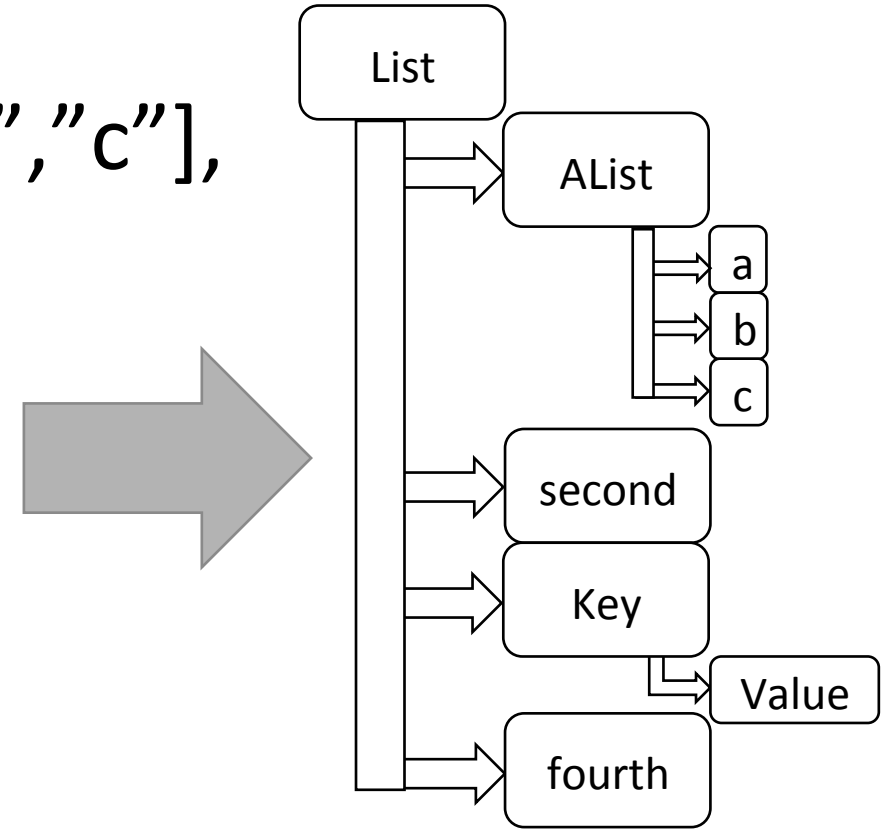
“AList”:[“a”,”b”,”c”],

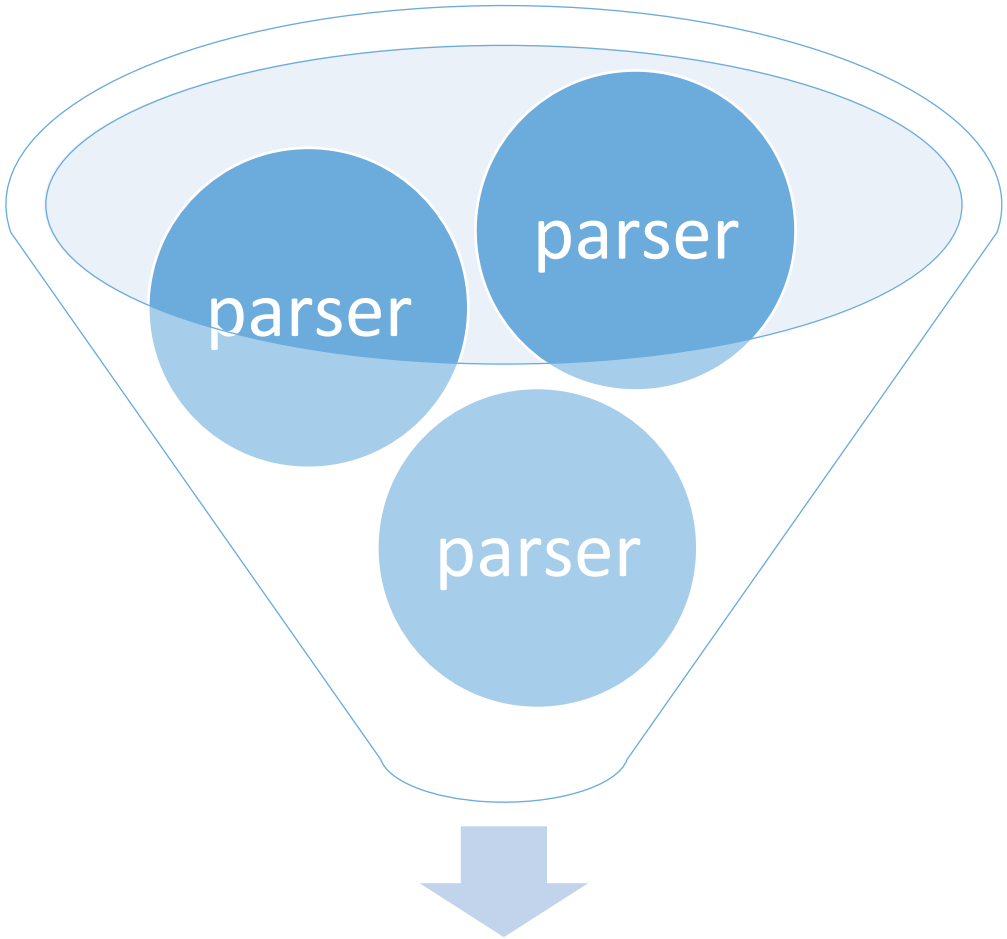
“second”,

“Key”: “value”,

“Fourth”

}





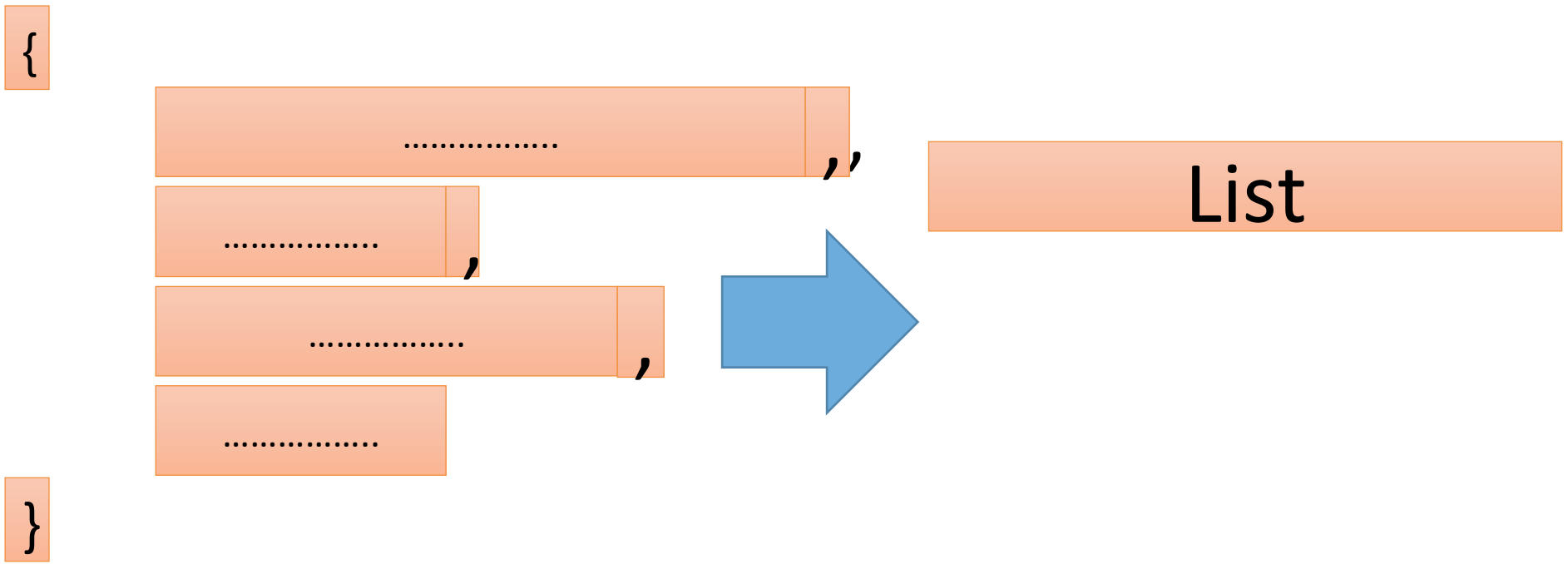
Code

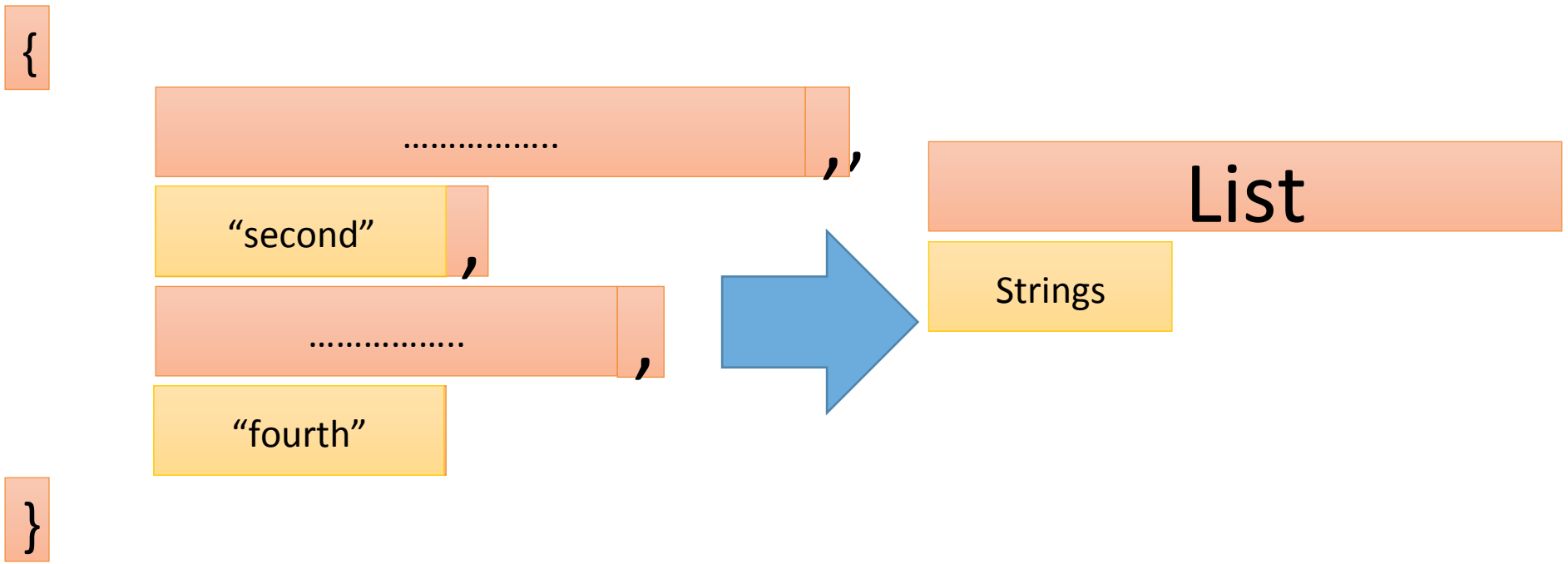
```
array
   ^ '[' asParser trim
      ,
         (((object , ',' asParser trim) star ,
object)
jsonSea
      ^ object / nameValuePair/ ((nameTag , ',' asParser trim)
star , nameTag))
      , ']' asParser trim
      map: [ :arg1 :arg2 :arg3 |
         | tmp1 |
         tmp1 := arg2 first
asOrderedCollection collect: #first.
         tmp1 add: arg2 second.
         tmp1 ]



object
   ^ '{' asParser trim , (nameValuePair ,
',' asParser trim) star
      , nameValuePair , '}' asParser trim
      map: [ :arg1 :arg2 :arg3 :arg4 |
         | tmp1 |
         tmp1 := arg2 asOrderedCollection
collect: #first.
         tmp1 add: arg3.
         tmp1 ]

start
   ^ run


letterNumber
   ^ quotationMark trim , quotationMark asParser
negate star flatten
      , quotationMark trim

jsonSea
   ^ super jsonSea
```

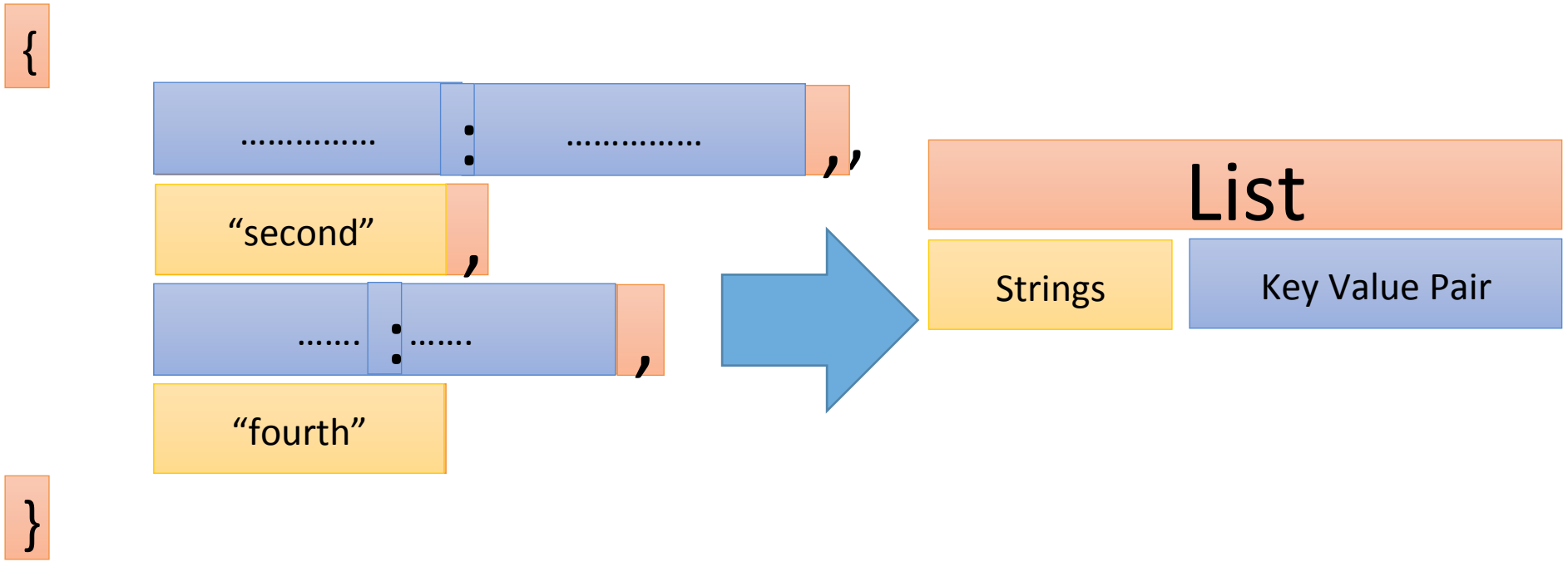
```
run
   ^ jsonSea


quotationMark
   ^ '"' asParser


nameValuePair
   ^ nameTag , ':' asParser
trim , valueTag trim


nameTag
   ^ quotationMark trim ,
#letter asParser star flatten
trim
      , quotationMark trim

nameTag
   ^ super nameTag
      ==> [ :arg1 |
         | tmp2 |
         tmp2 := RawText new.
         tmp2
            name: arg1 second;
            yourself ]



letterNumber
   ^ super letterNumber
      ==> [ :arg1 | (RawText
new name: arg1 second)
yourself ]
```

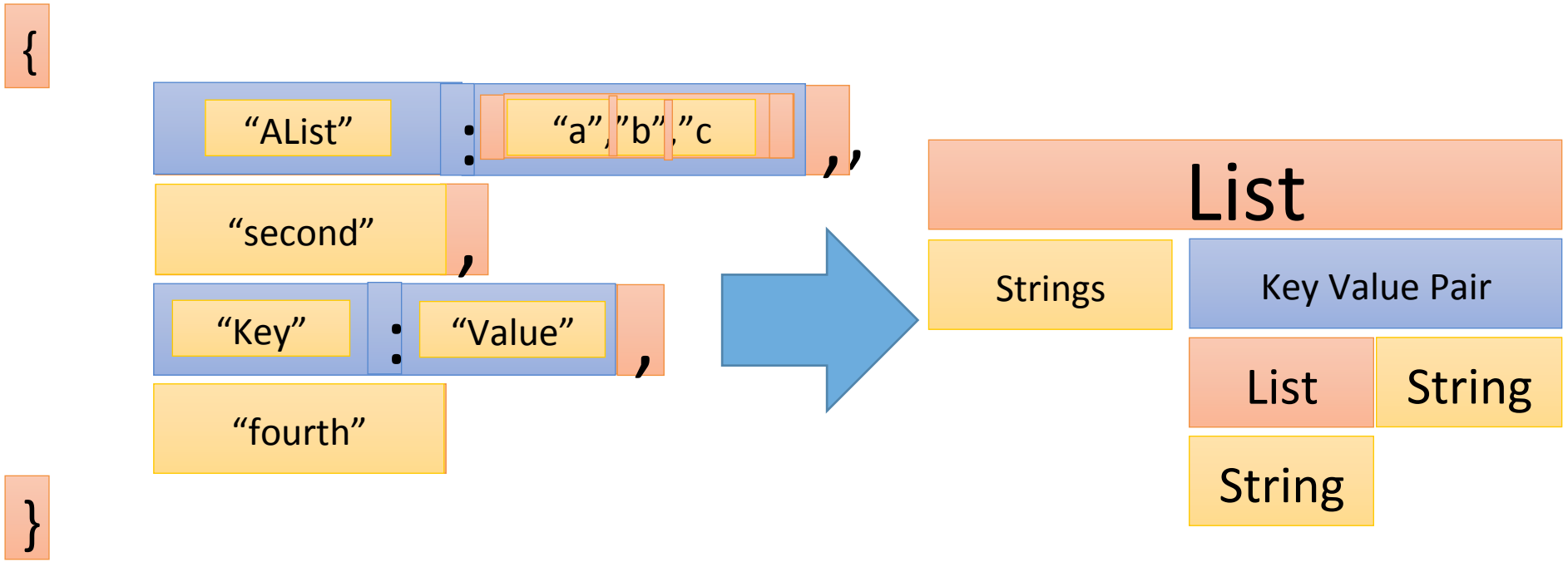
```
valueTag
   ^ array / object /
letterNumber

array
   ^ super array
      ==> [ :arg1 |
         | tmp2 |
         tmp2 := Element new.
         tmp2
            name: 'An Array';
            children: arg1;
            yourself ]



nameValuePair
   ^ super nameValuePair
      ==> [ :arg1 |
         | tmp2 |
         tmp2 := Element new.
         tmp2 name: arg1 first
name.
         (tmp2 addChild: arg1
third) yourself ]

object
   ^ super object
      ==> [ :arg1 |
         | tmp2 |
         tmp2 := Element new.
         tmp2
            name: 'An Object';
            children: arg1;
            yourself ]


valueTag
   ^ super valueTag
      ==> [ :arg1 |
         | tmp2 |
         tmp2 := Element new.
         tmp2 name: 'valueTag'.
         (tmp2 addChild: arg1)
yourself ]
```

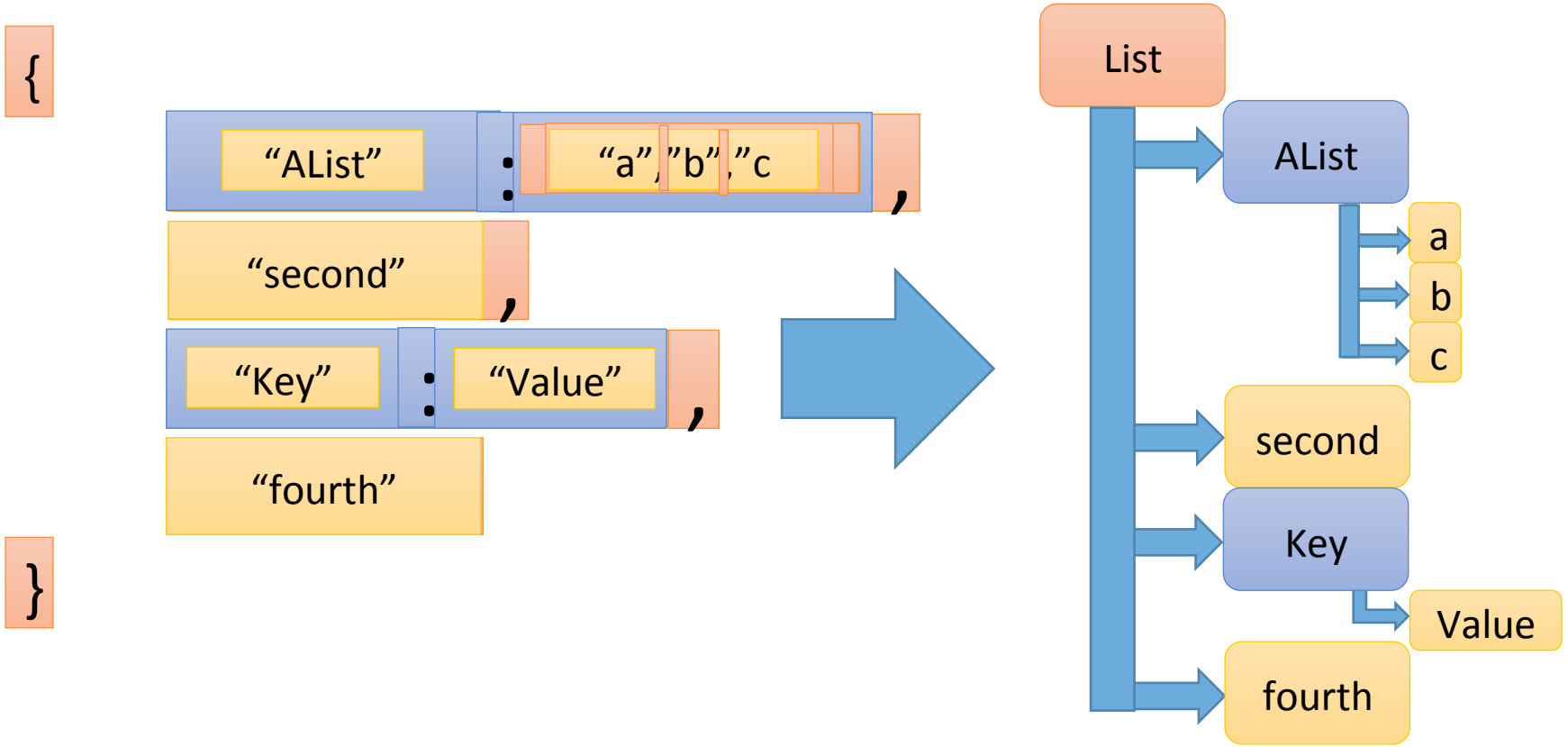
```
{
        “AList”:[ “a”,”b”,”c” ],
        “second”,
        “Key”: “value”,
        “Fourth”
}
```


{
,,
,
,
}
List

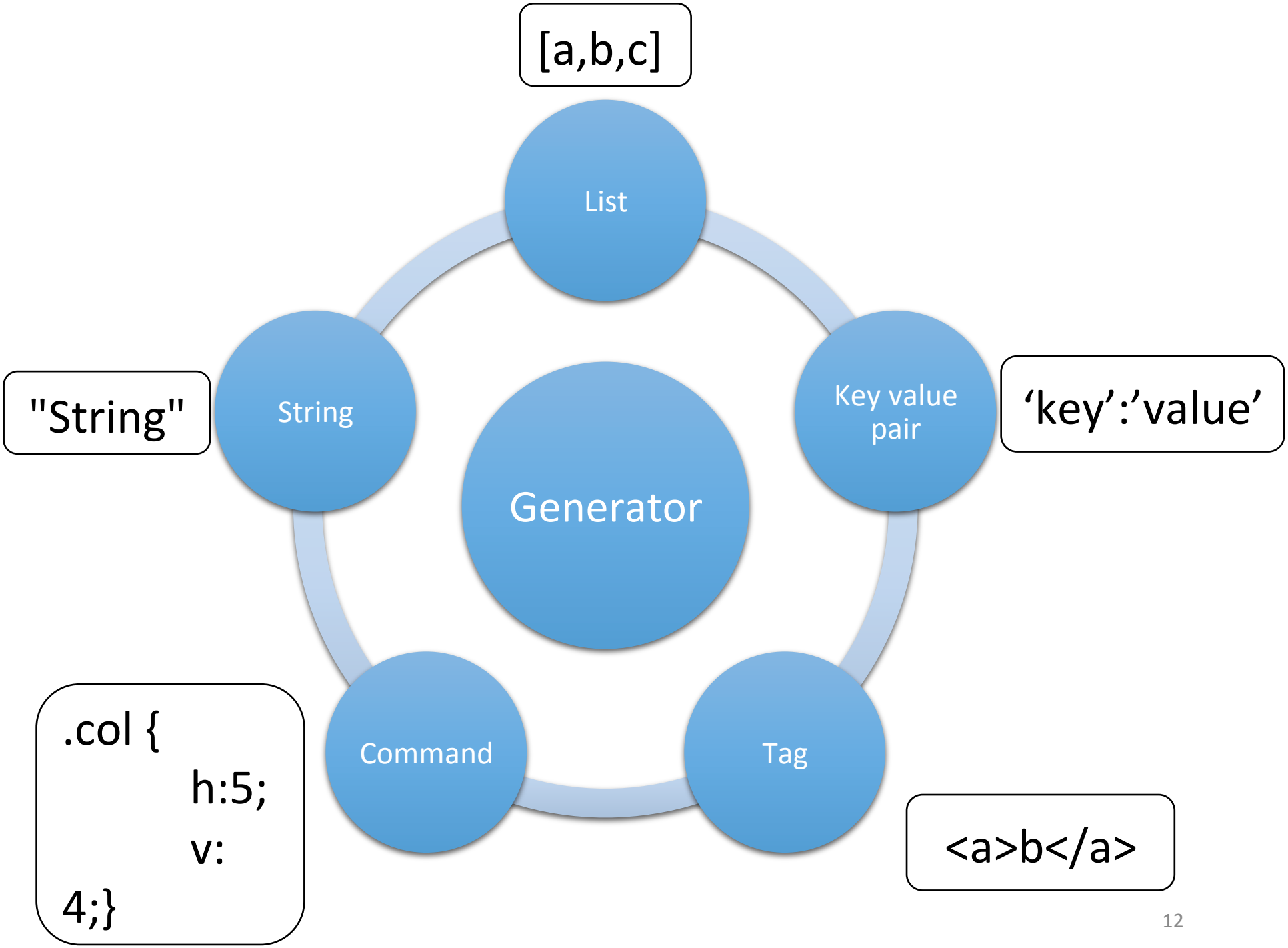

{
.....
,
"second"
,
.....
,
"fourth"
}
List
Strings

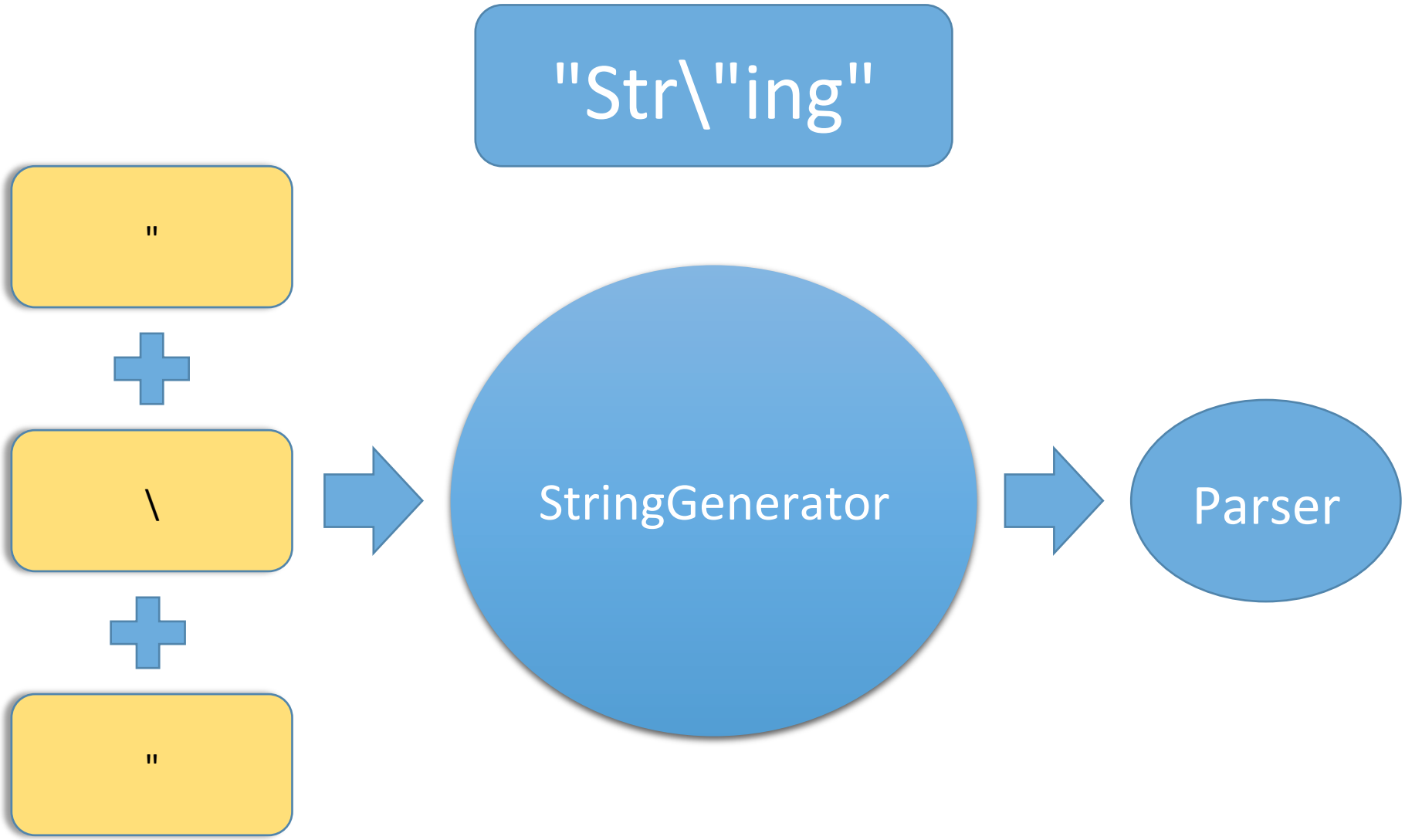

{
.....
:
.....
,
"second"
,
.....
:
.....
,
"fourth"
}
List
Strings
Key Value Pair

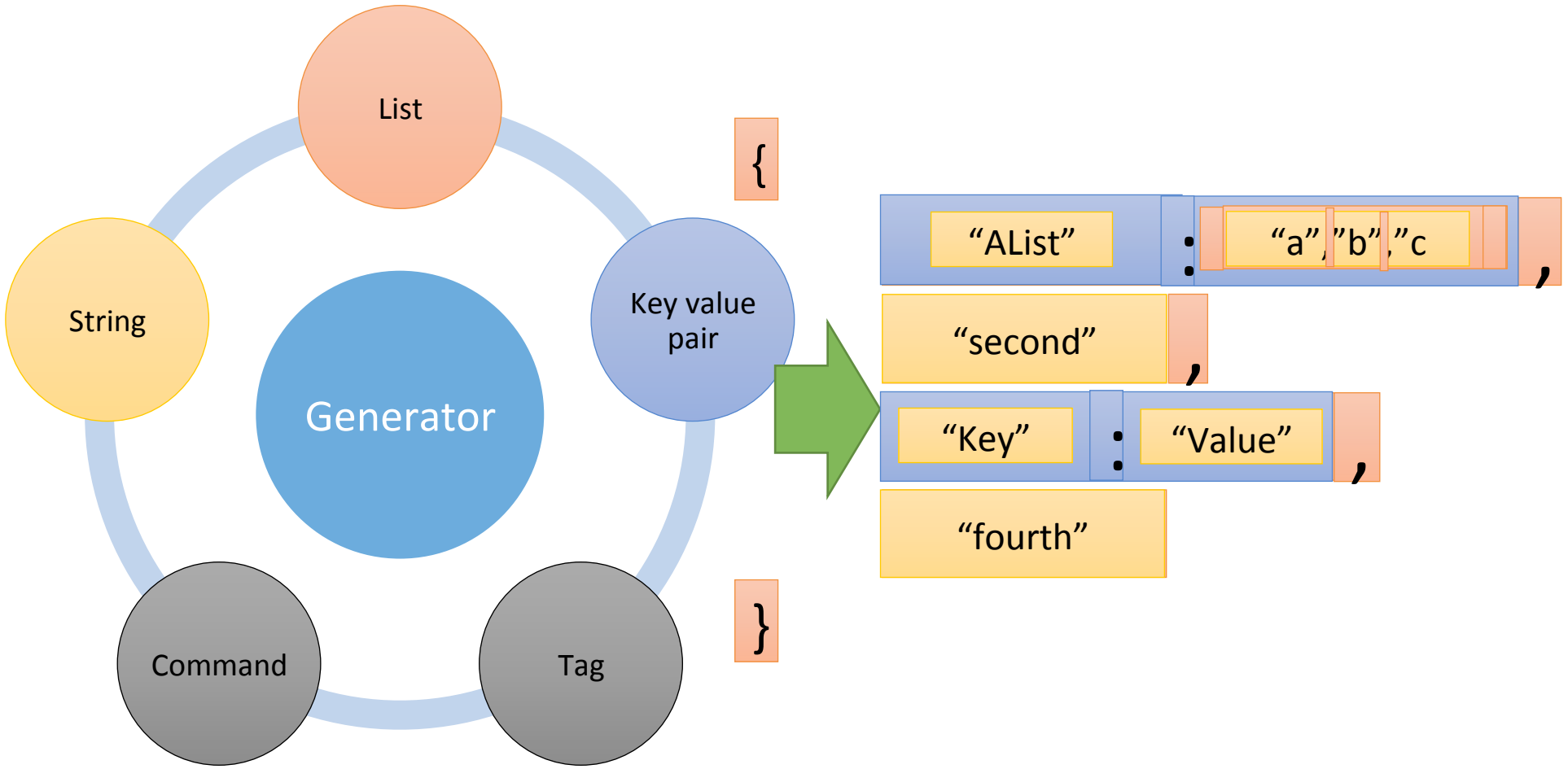

{
"AList"
:
"a"
"b"
"c
,
"second"
,
"Key"
:
"Value"
,
"fourth"
}
List
Strings
Key Value Pair
List
String
String


{
"AList"
:
"a"
"b"
"c
,
"second"
,
"Key"
:
"Value"
,
"fourth"
}
List
AList
a
b
c
second
Key
Value
fourth

```
jsonArray
    ^ (ListGenerator new)
        begin:$[ asParser token;
        end:$] asParser;
        delimiter:$, asParser;
        element: jsonObject /jsonArray/ StringGenerator doubleQuoteString / StringGenerator
identifierDigit / StringGenerator identifier ;
        generateParser
jsonObject
    ^ (ListGenerator new)
        begin:${ asParser token;
        end:$} asParser;
        delimiter:$, asParser;
        element: keywordValuePair ;
        generateParser

start
    ^ jsonObject / keywordValuePair

keywordValuePair
    ^ (KeyValueGenerator new)
        key: StringGenerator doubleQuoteString;
        delimiter: $: asParser;
        value: StringGenerator doubleQuoteString/ jsonArray /jsonObject / StringGenerator identifier /
StringGenerator identifierDigit;
        generateParser
```


[a,b,c]
List
"String"
String
Generator
Key value pair
'key':'value'
.col {
h:5;
v:
4;}
Command
Tag
<a>b


"Str\"ing"
"
\
"
StringGenerator
Parser


List
String
Generator
Key value pair
Command
Tag
{
}
“AList”
:
“a”,”b”,”c
,
“second”
,
“Key”
:
“Value”
,
“fourth”

PetitParser basic

```
array
    ^ '[' asParser trim
        ,
            (((object , ',' asParser trim) star ,
object)
        /*or*/ ((nameTag , ',' asParser trim)
star , nameTag))
        , ']' asParser trim
        map: [ :arg1 :arg2 :arg3 |
            | tmp1 |
            tmp1 := arg2 first asOrderedCollection
collect: #first.
            tmp1 add: arg2 second.
            tmp1 ]

object
    ^ '{' asParser trim , (nameValuePair ,
',' asParser trim) star
        , nameValuePair , '}' asParser trim
        map: [ :arg1 :arg2 :arg3 :arg4 |
            | tmp1 |
            tmp1 := arg2 asOrderedCollection
collect: #first.
            tmp1 add: arg3.
            tmp1 ]

start
    ^ run

letterNumber
    ^ quotationMark trim , quotationMark asParser
negate star flatten
        , quotationMark trim

jsonSea
    ^ super jsonSea
```

```
run
    ^ jsonSea

quotationMark
    ^ '"' asParser

nameValuePair
    ^ nameTag , ':' asParser
trim , valueTag trim

nameTag
    ^ quotationMark trim ,
#letter asParser star flatten
trim
        , quotationMark trim

nameTag
    ^ super nameTag
        ==> [ :arg1 |
            | tmp2 |
            tmp2 := RawText new.
            tmp2
                name: arg1 second;
                yourself ]

letterNumber
    ^ super letterNumber
        ==> [ :arg1 | (RawText
new name: arg1 second) yourself
]
```

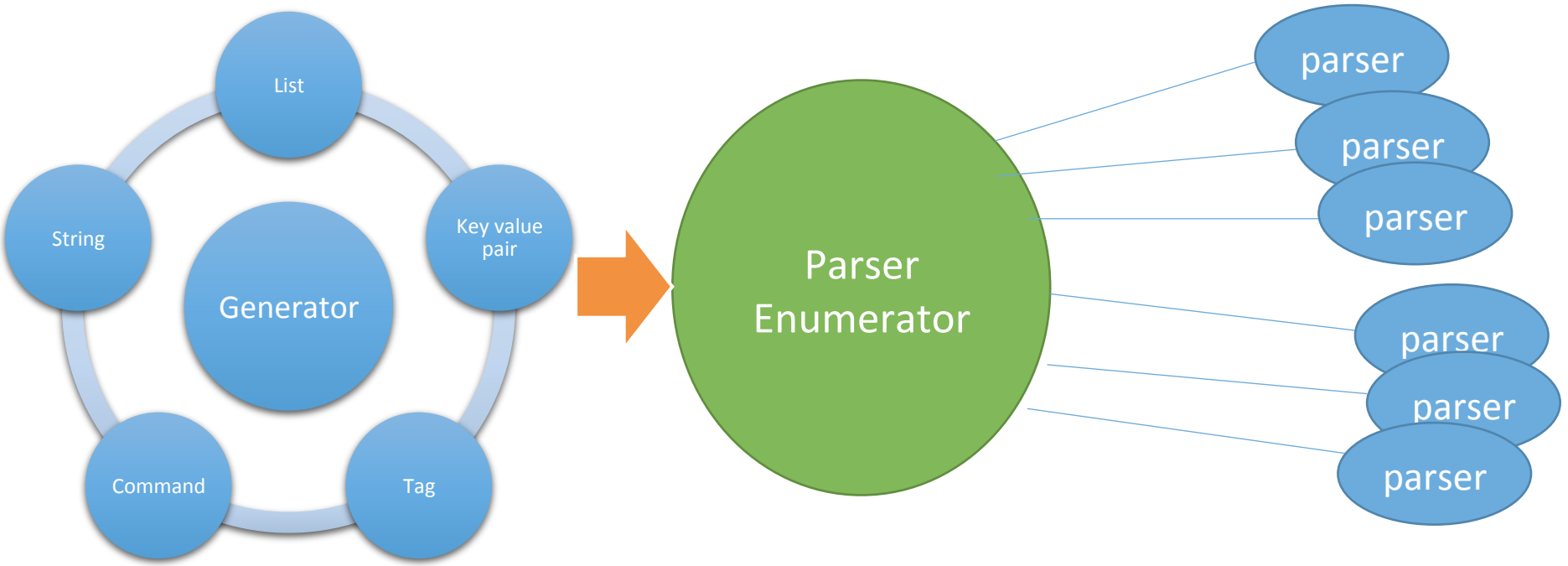
```
valueTag
    ^ array / object /
letterNumber

array
    ^ super array
        ==> [ :arg1 |
            | tmp2 |
            tmp2 := Element new.
            tmp2
                name: 'An Array';
                children: arg1;
                yourself ]

nameValuePair
    ^ super nameValuePair
        ==> [ :arg1 |
            | tmp2 |
            tmp2 := Element new.
            tmp2 name: arg1 first
name.
            (tmp2 addChild: arg1
third) yourself ]

object
    ^ super object
        ==> [ :arg1 |
            | tmp2 |
            tmp2 := Element new.
            tmp2
                name: 'An Object';
                children: arg1;
                yourself ]

valueTag
    ^ super valueTag
        ==> [ :arg1 |
            | tmp2 |
            tmp2 := Element new.
            tmp2 name: 'valueTag'.
            (tmp2 addChild: arg1)
yourself ]
```

Generators

```
jsonArray
    ^ (ListGenerator new)
        begin:$[ asParser token;
        end:$] asParser;
        delimiter:$, asParser;
        element: jsonObject /jsonArray/ StringGenerator doubleQuoteString / StringGenerator
identifierDigit / StringGenerator identifier ;
        generateParser
jsonObject
    ^ (ListGenerator new)
        begin:${ asParser token;
        end:$} asParser;
        delimiter:$, asParser;
        element: keywordValuePair ;
        generateParser
```

```
start
    ^ jsonObject / keywordValuePair
```

```
keywordValuePair
    ^ (KeyValueGenerator new)
        key: StringGenerator doubleQuoteString;
        delimiter: $: asParser;
        value: StringGenerator doubleQuoteString/ jsonArray /jsonObject / StringGenerator identifier /
StringGenerator identifierDigit;
        generateParser
```

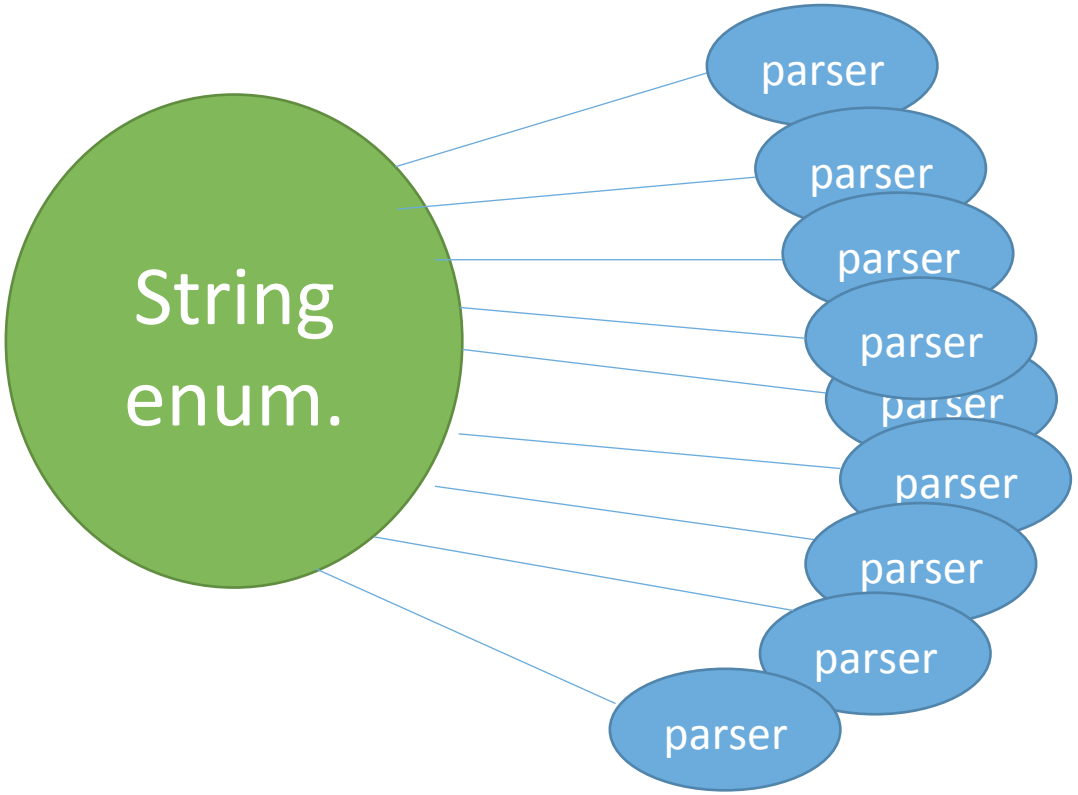

List
String
Key value pair
Generator
Command
Tag
Parser Enumerator
parser
parser
parser
parser
parser
parser

Parameters

Begins
<
‘
"

Escapes
\

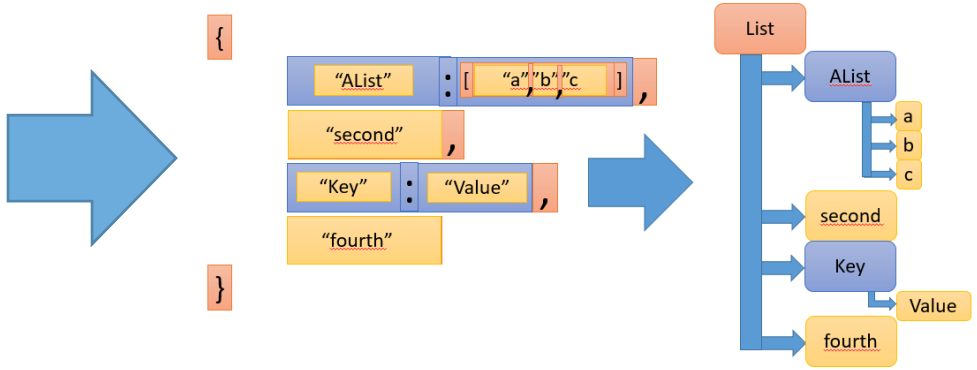
Ends
>
‘
"

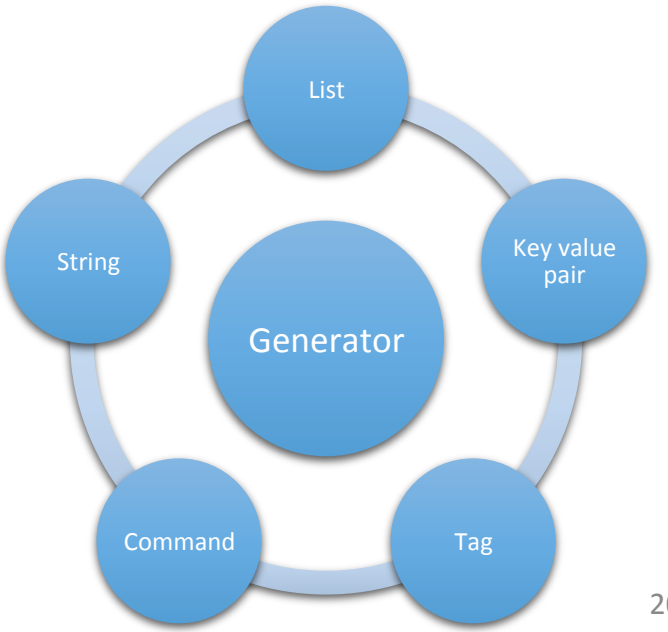


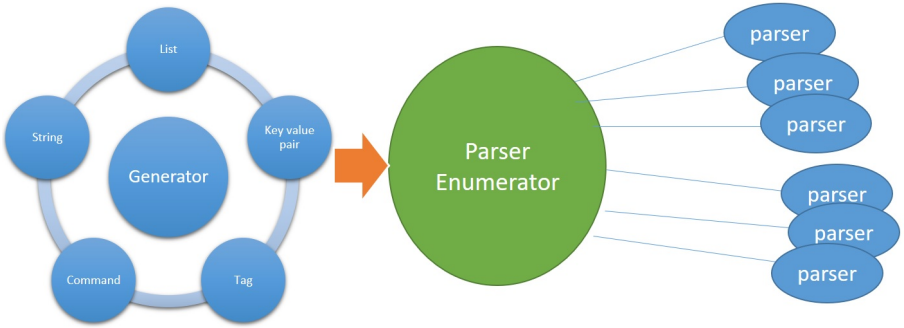

Playground
Page
input:='{
"AList": ["a","b","c"],
"second",
"Key" : "value",
"Fourth"
}'.
((CompositeEnumerator new) enumerationNamed:#list)
anySatisfy: [:parser | parser end matches: input].

Challenges

- Way to get arguments for enumerators
- Parsers only succeed or fail
- [a , b ; c , d ; e , c ; g]


{
"AList" : ["a","b","c"] ,
"second" ,
"Key" : "Value" ,
"fourth"
}
List
AList
a
b
c
second
Key
Value
fourth


List
String
Key value pair
Generator
Command
Tag


List
String
Key value pair
Generator
Command
Tag
Parser Enumerator
parser
parser
parser
parser
parser
parser