

# **Atrai: A Framework for Rapidly Implementing Programming Languages**

## Lecture 9

# Programming Language/DSL: Implementation Steps

---

- Write a parser for the language
- Parse a program to generate a simple form of the parse tree called Untyped Tree
- Write a program that interprets the Untyped Tree
  - Specify a Transformer that describes a set of rules on how to transform an untyped tree into another
  - Transformers can quickly encode the operational semantics rules/type checking rules/code generation rules

# Write Grammar in ANTLR 4

---

- Avoid using Kleene  $*$ ,  $+$ ,  $?$  for binary operators
- Left recursion is allowed
- Uses  $ALL(*)$  algorithm [out of scope of this class]
- ANTLR resolves ambiguities in favor of the alternative given first
  - allows to specify operator precedence implicitly
  - ANTLR associates operators left to right as we'd expect for  $*$  and  $+$
  - right associativity is specified manually using `<assoc=right>` option

# Sample Grammar in ANTLR 4

---

```
grammar Ex;                                     // generates class ExParser
stat: stat expr ';'
    | expr ';'
    ;
expr: expr '^' <assoc=right> expr
    | expr '*' expr
    | expr '+' expr
    | '(' expr ')'
    | num
    ;
num: INT;
INT: [0-9]+;
WS : [ \t\r\n]+ -> skip ;                     // ignore whitespace
```

# Environment API

---

- Simple and INEFFECIENT implementation

See [javadocs/index.html](http://javadocs/index.html) for documentation

See [src/main/java/atrai/interpreters/common/Environment.java](http://src/main/java/atrai/interpreters/common/Environment.java)

```
public class Environment {  
  
    public static Environment extend(String key, Object value, Environment env);  
  
    public Object get(String key);  
  
}
```

# Environment Implementation

---

- Simple and INEFFICIENT implementation

```
public class Environment {  
    private final String key;  
    private final Environment next;  
    private Object val;  
  
    private Environment(String key,  
        Object val, Environment next) {  
        this.key = key;  
        this.val = val;  
        this.next = next;  
    }  
}
```

```
public static Environment extend(String key,  
    Object value, Environment env) {  
    return new Environment(key, value, env);  
}  
  
public Object get(String key) {  
    Environment tmp = this;  
    while (tmp != null) {  
        if (key.equals(tmp.key)) {  
            return tmp.val;  
        }  
        tmp = tmp.next;  
    }  
    return null;  
}
```

# Atrai (After the name of a river in north West Bengal)

---

- A framework for matching and transforming trees
- Written in Java
- Uses three basic concepts
  - Untyped Tree
  - Pattern
  - Template
- Advanced concepts: Transformer and Visitor
- PA2 will use Atrai for rapid prototyping

# Installation and Running Interpreters

---

Inside Atrai directory execute to build Atrai and run tests:

```
mvn clean test
```

To run the LET interpreter on a sample LET program, say test.let, execute

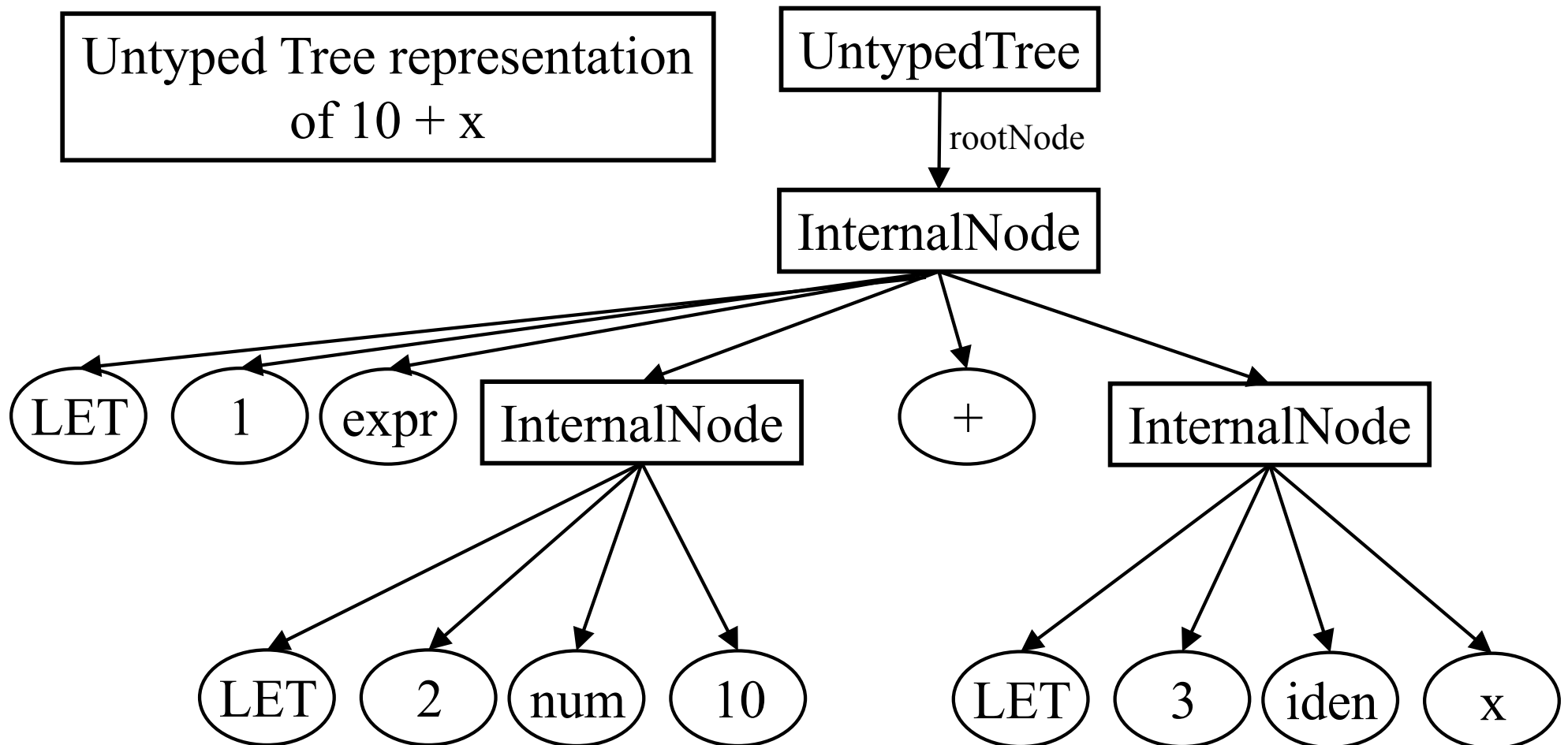
```
mvn clean package  
java -jar lib/atrai-1.0.jar interpret atrai.interpreters.LET.  
LetInterpreter test.let
```

If you have implemented your interpreter, say atrai.interpreters.LETREC.LetrecInterpreter, by extending atrai.interpreters.common.Interpreter, then you can run the interpreter as follows:

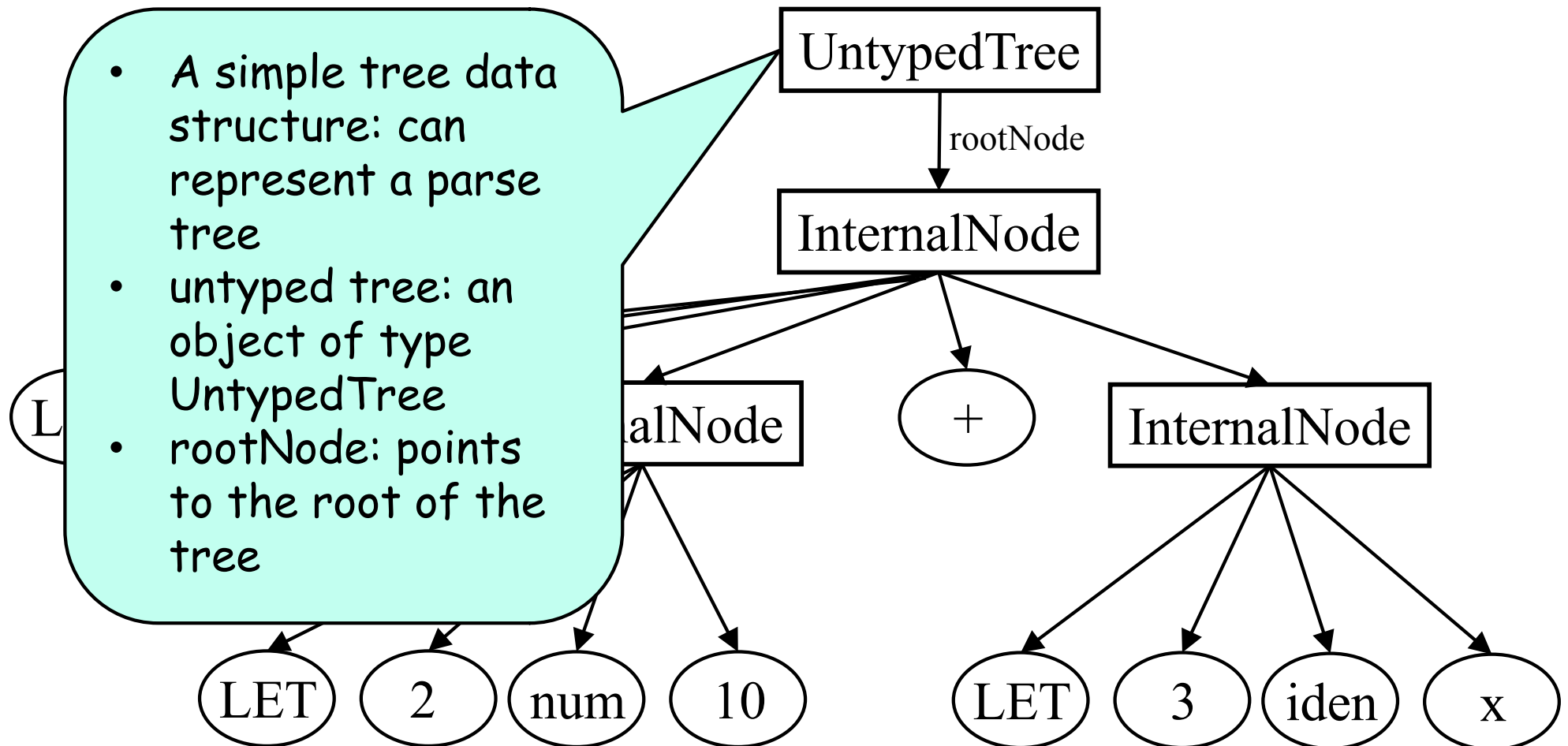
```
mvn clean package  
java -jar lib/atrai-1.0.jar interpret atrai.interpreters.  
LETREC.LetrecInterpreter test.letrec
```



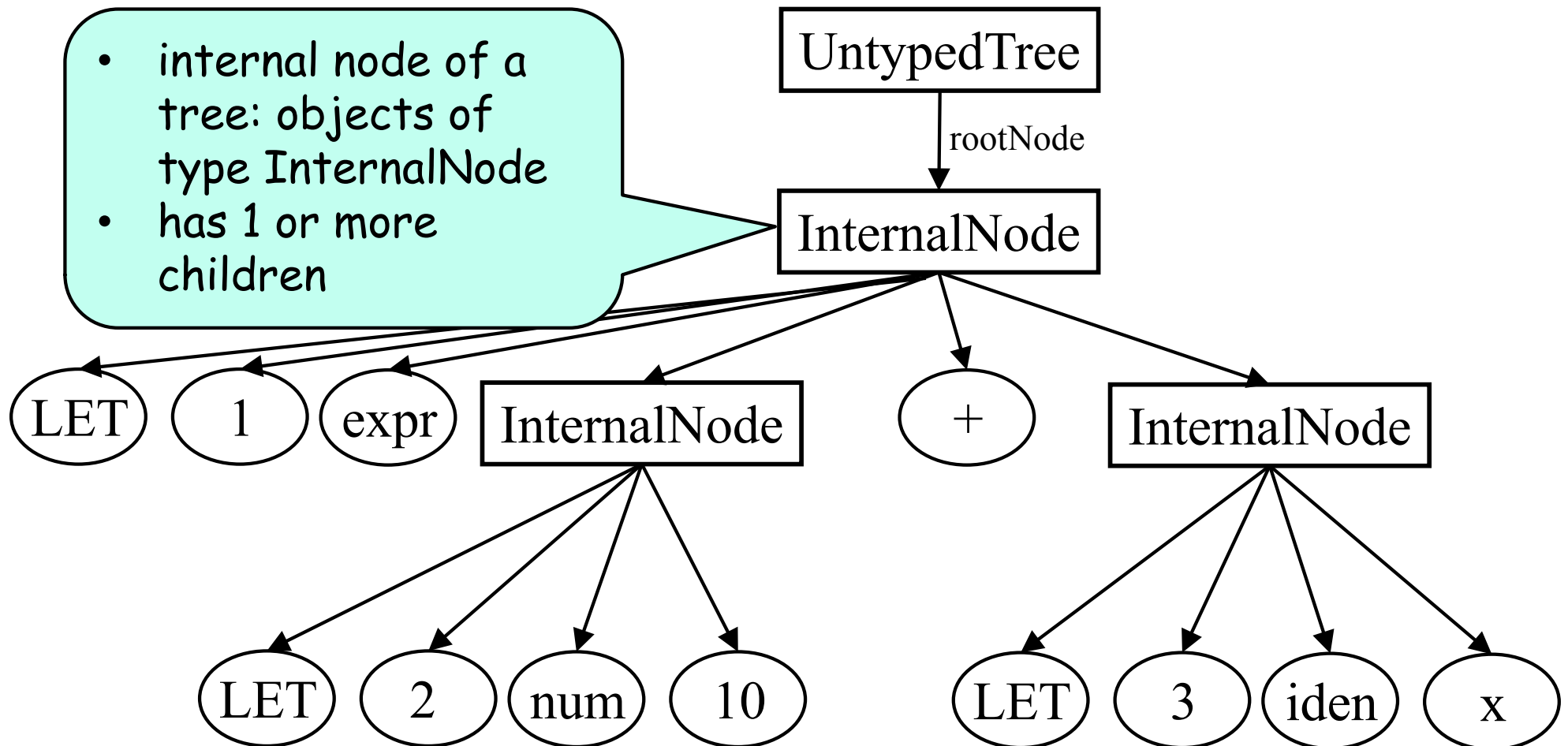
# Untyped Tree



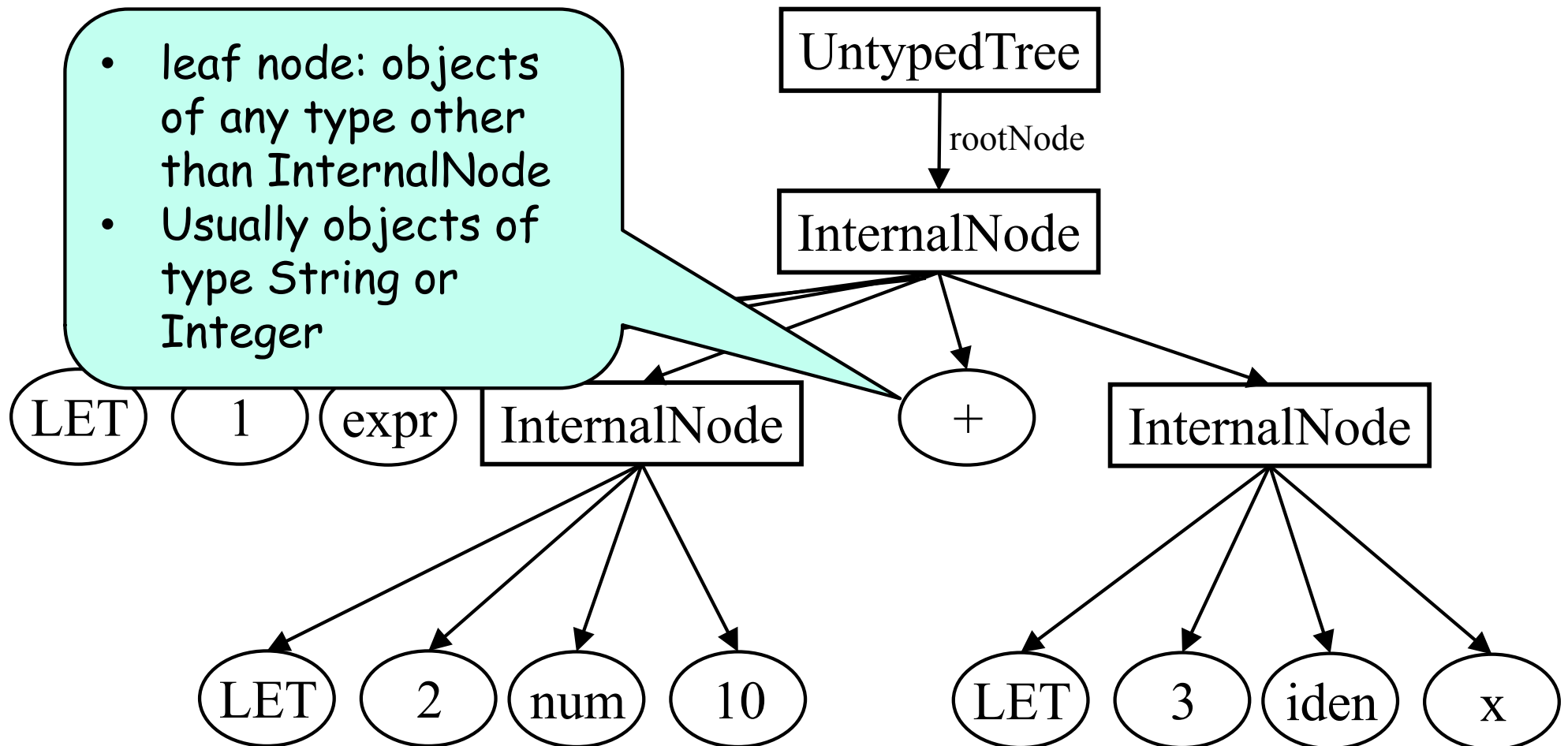
# Untyped Tree



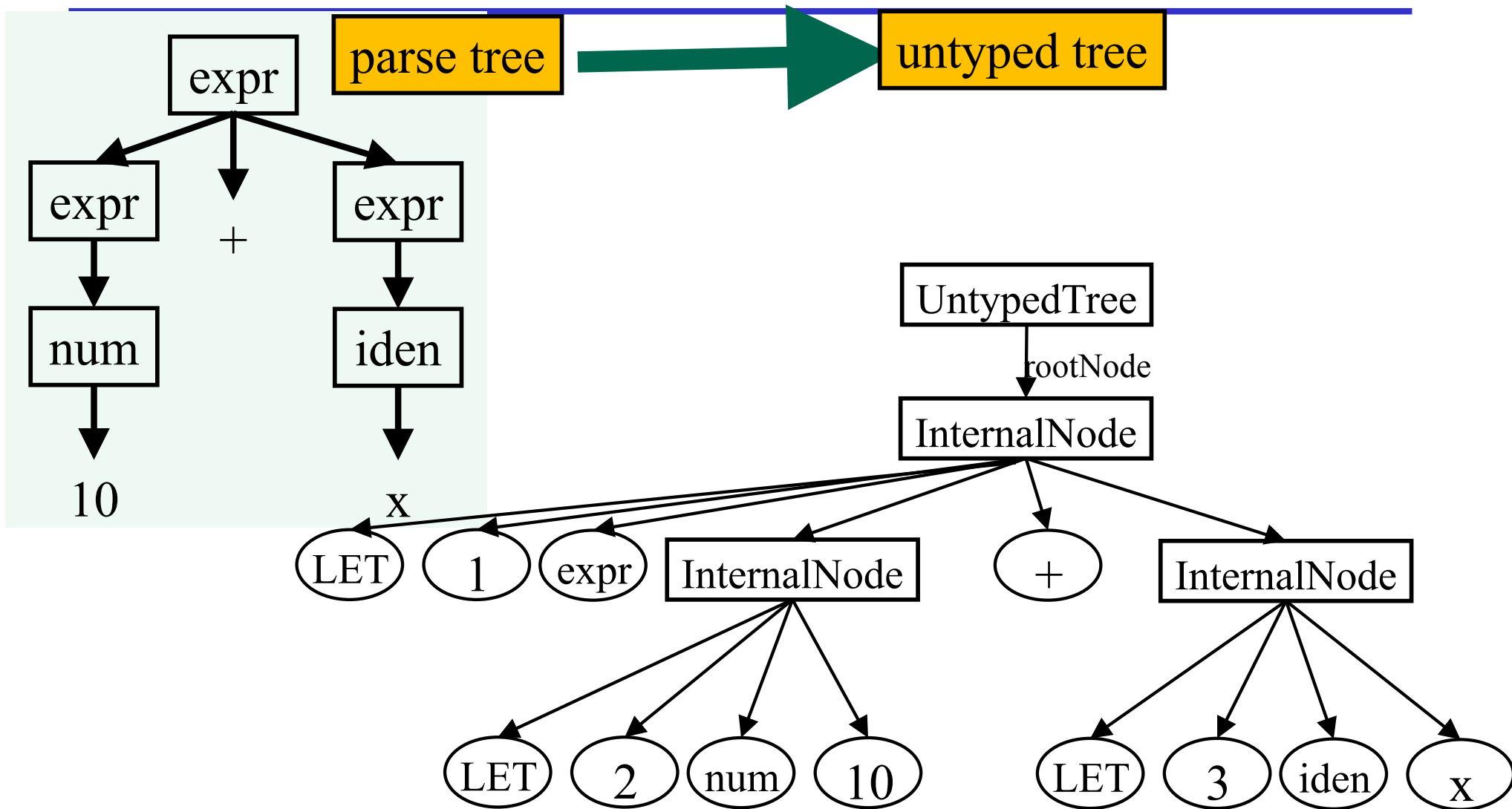
# Untyped Tree



# Untyped Tree



# Untyped tree can represent a parse tree



## General conventions in representing a parse tree using an untyped tree

---

- First children of any sub-untyped tree is the name of the language/grammar
- Second children after is an unique id
  - id maps to location in the original program string
- Third children is the name of the non-terminal which forms the root of the subtree
- We can deviate from this convention if necessary

# Create Untyped Tree from a Parse Tree

---

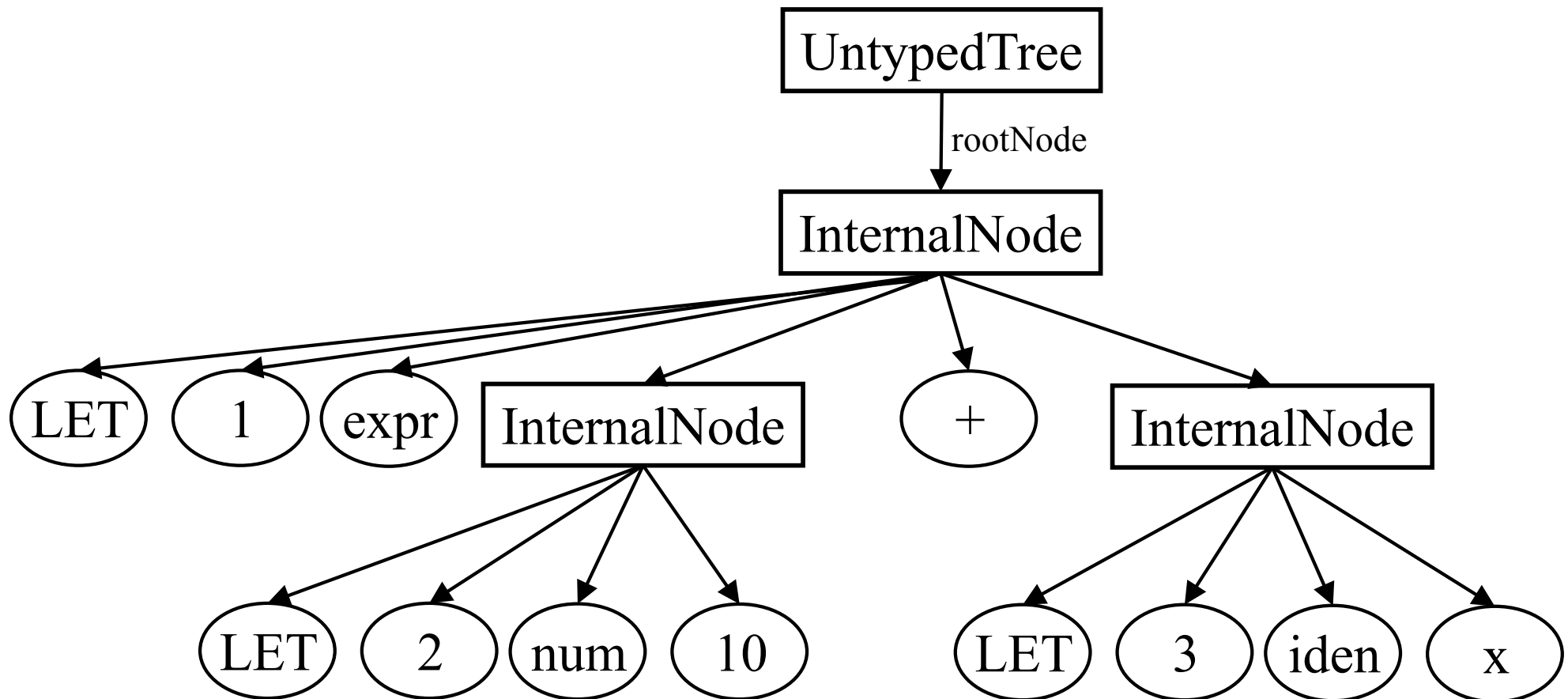
## Pseudo-code:

```
counter = 1;
languageName = "LET";
createTree(tree) { // tree is a parse tree
    ret = new InternalNode();
    ret.addChild(languageName)
    ret.addChild(counter)
    ret.addChild(tree.name);
    counter++;
    for each child of tree
        if child is a tree
            ret.addChild(createTree(child));
        else
            ret.addChild(child);
    return ret;
}
createUntypedTree(tree) {
    UntypedTree ut = new UntypedTree();
    ut.setRoot(createTree(tree));
    return ut;
}
```

# Untyped Untyped Tree

String representation of an untyped tree:

- (%LET 1 expr (%LET 2 num 10%) + (%LET 3 iden x%))%)





# Facts about a Untyped untyped tree

---

- Textual representation of a tree
  - can be used for debugging
  - can express textual patterns over untyped trees
  - can express textual templates for creating untyped trees
- Similar to S-expressions in LISP
- Has balanced (% and %) meta-characters
- Captures the syntactic structure of an untyped tree
- If we erase (% and %), and the first three leaves after each (% we get the original program except the whitespaces
- Use ` to escape any character including `

# Creating a Untyped Untyped Tree from an Untyped Tree

---

```
serialize(tree) { // tree is an untyped tree
    ret = "("%;
    for each child of tree
        if child is a tree
            ret = ret + serialize(child);
        else
            ret = ret + " " + child;
    ret = ret + "%)"
    return ret;
}
```

# Example of an Untyped Tree and the Corresponding Parse Tree

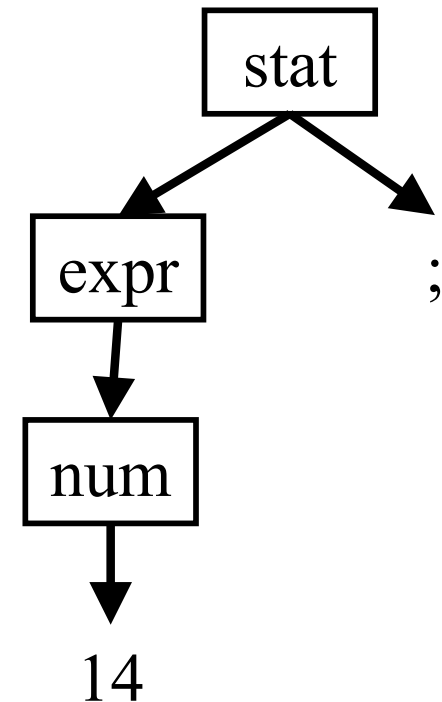
---

Given grammar Ex:

Untyped tree of the program: 14 ;

is

```
(%Ex 1 stat
  (%Ex 2 expr (%Ex 3 num 14%))%)
;
```



# Example of an Untyped Tree and the Corresponding Parse Tree

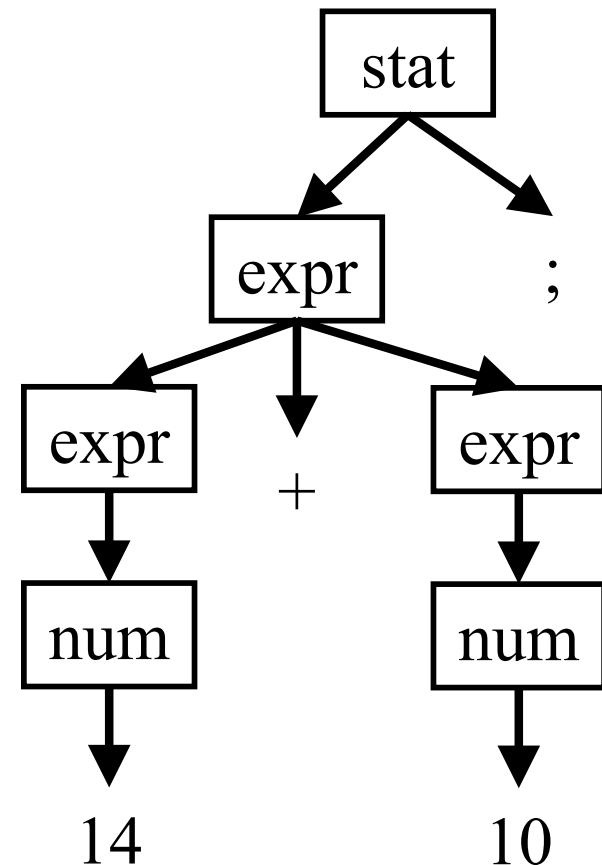
Given grammar Ex:

Serialized untyped tree of the program:

14 + 10 ;

is

```
(%Ex 1 stat
  (%Ex 2 expr
    (%Ex 3 expr (%Ex 4 num 14%))%
    +
    (%Ex 5 expr (%Ex 6 num 10%))%
  %)
  ;
%)
```



# UntypedTree class

---

A untyped tree is an instance of the class `UntypedTree` and has a reference to the root node, which is either an internal node or a leaf.

```
class UntypedTree extends Tree {  
    static UntypedTree parse(String src, Lexer lexer);  
    public Object getRoot();  
    public Location getLocationFromID(int id);  
    public String toString();  
    public String toIndentedString();  
}
```

Check [javadocs/index.html](http://javadocs/index.html) for further documentation

See [src/main/java/atra/core/UntypedTree.java](http://src/main/java/atra/core/UntypedTree.java) for source

See [src/test/java/atrai/core/TreeNodeTest.java](http://src/test/java/atrai/core/TreeNodeTest.java) for examples

# InternalNode

---

An internal node in an untyped tree is an instance of the class `InternalNode` and have a non-zero number of children.

```
class InternalNode extends TreeNode {  
    // iterates each child and applies lambda to the child  
    Object iterate(Reducer lambda, Object initialReductionValue, Object context);  
    String toString();  
    String toIndentedString();  
}
```

Check [javadocs/index.html](http://javadocs/index.html) for further documentation  
See [src/main/java/atra/core/InternalNode.java](http://src/main/java/atra/core/InternalNode.java) for source

# Generate Untyped Tree from an ANTLR 4 grammar

---

- One can use **GenericAntlrToUntypedTree** class to generate an untyped tree from a string and a grammar

```
GenericAntlrToUntypedTree p = new GenericAntlrToUntypedTree();  
UntypedTree st =
```

```
    p.parseStringToUntypedTree("Ex", "Ex", "stat", "1 + 2 * 3;");
```

```
System.out.println(st.toIndentedString());
```

```
// 1st argument is the name of the language
```

```
// 2nd argument is the name of the parser: often same as the 1st argument
```

```
// 3rd argument is the start symbol in the grammar
```

```
// 4th argument is the string to be parsed and converted to a Untyped Tree
```

```
p.parseFileToUntypedTree("Ex", "Ex", "stat", "myprogram.ex");
```

Leaves are always objects of type String except the unique ids which are Integers

# Generate Untyped Tree from an ANTLR 4 grammar

---

- Example can be found in the methods `parseFile` and `parseString` in the file `src/main/java/atrai/interpreters/LET/LetInterpreter.java`



# Convert a Serialized Untyped Tree to an Untyped Tree and vice-versa

---

```
// create a lexer which is used to tokenize strings in a Pattern  
ANTLRTokenizer lexer = new ANTLRTokenizer("Ex");
```

```
// create an Untyped Tree from a Serialized Untyped Tree  
UntypedTree st = UntypedTree.parse("(%Ex 2 expr (%Ex 3 num  
14%)%)", lexer);
```

```
// create a Serialized Untyped Tree from an Untyped Tree  
String s = st.toString();  
// or  
s = st.toIndentedString();
```

See [src/test/java/atrai/core/TreeNodeTest.java](#) for example transformations

# Patterns

---

- A pattern is like a regular expression
  - is used to match an untyped tree
  - to capture parts of the matched untyped tree
- A pattern is again a untyped Tree
  - with special symbols @ and @\_
  - @ matches any subtree or leaf
  - @\_ matches any subtree or leaf, and creates a capture group

# Example of a Pattern

---

Given grammar Ex:

An example pattern is `(%Ex @ num @_%)`

The pattern matches the following Untyped Tree:

`(%Ex 1 num 14%)`

A successful match returns `capture`, which is an array of captures:

`capture[0] = (%Ex 1 num 14%)` //the entire matched untyped tree as an instance of `UntypedTree`

`capture[1] = 14`

# Example of a Pattern

---

Given grammar Ex:

An example pattern is `(%Ex @_ expr (%Ex @ num @_ %))`

The pattern matches the following Untyped Tree:

`(%Ex 1 expr (%Ex 2 num 14 %))`

A successful match returns `capture`, which is an array of captures:

`capture[0] = (%Ex 1 expr (%Ex 2 num 14 %))`

`capture[1] = 1`

`capture[2] = 14`

# Example of a Pattern

---

Given grammar Ex:

An example pattern is `(%Ex @_ expr @_ + @_ %)`

The pattern matches the following Untyped Tree:

```
(%Ex 1 expr
  (%Ex 2 expr (%Ex 3 num 14%)%)
  +
  (%Ex 4 expr (%Ex 5 num 10%)%)
%)
```

A successful match returns `capture`, which is an array of captures:

`capture[0]` = the entire untyped tree // as an instance of `UntypedTree`

`capture[1]` = 1

`capture[2]` = `(%Ex 2 expr (%Ex 3 num 14%)%)` // as an instance of `UntypedTree`

`capture[3]` = `(%Ex 4 expr (%Ex 5 num 10%)%)` // as an instance of `UntypedTree`

# Pattern API

---

```
class Pattern extends Tree {  
    // create a pattern from a string and a lexer  
    static Pattern parse(String pattern, Lexer lexer);  
    // matches the pattern against an untyped tree  
    // returns the array of captures or null (if match fails)  
    Object[] match(UntypedTree tree);  
    // matches the pattern against a node or a leaf  
    // returns the array of captures or null (if match fails)  
    Object[] match(Object node);  
    // returns the array of captures from the last match  
    Object[] getMatches();  
    // returns the object at index from the last capture array  
    Object getMatch(int index);  
}
```

Check [javadocs/index.html](http://javadocs/index.html) for further documentation

See [src/main/java/atra/core/Pattern.java](http://src/main/java/atra/core/Pattern.java) for source

See [src/test/java/atrai/core/PatternTest.java](http://src/test/java/atrai/core/PatternTest.java) for examples

# Example of a Pattern in Atrai

---

```
// create a lexer which is used to tokenize strings in a Pattern
ANTLRTokenizer lexer = new ANTLRTokenizer("Ex");
// create a pattern from a string in serailized untyped tree form
Pattern p = Pattern.parse("(%Ex @ num @_%)", lexer);
// match the pattern against an untyped tree st
Object[] captures = p.match(st);
// returns null if match fails
```

# Example Usage of the Pattern API

---

```
String pattern = "(% if (@_) (%{ @_ }%) else @_ %)";
String source = "(%if ( (% x > 0%) ) (% { (%x = (%- x%) %) }%) else (% (% x
%) = (%x + 1%)%)%)";
Lexer lexer = new SimpleStringTokenizer();
UntypedTree s = UntypedTree.parse(source, lexer);
Pattern p = Pattern.parse(pattern, lexer);
Object[] captures = p.match(s);
```

```
// captures[0] is the untype tree corresponding to source
// captures[1] = (%x > 0%)
// captures[2] = (%x = (%- x%)%)
// captures[3] = (%(%x%) = (%x + 1%)%)
```

`SimpleStringTokenizer` is a simple `Lexer` that uses Java's `StringTokenizer` to tokenize a string.

See <src/test/java/atrait/core/PatternTest.java> for more examples



# Transformer: for matching and modifying an untyped tree, or a node or a leaf

- A set of pattern/action pairs:  $(p, a)$ 
  - where  $p$  is a pattern given as a string
  - $a$  is a lambda that takes an array of captures and a context and returns an object
- If an untyped tree or a node matches pattern  $p$  and returns the array of captures  $c$ 
  - then call  $a(c)$  and return the result
- Example:  
 $(\%Ex \ @ \ num \ @_ \ %), (captures, E) \rightarrow \{return \ captures[1]; \}$   
 $(\%Ex \ @ \ expr \ @_ \ + \ @_ \ %), (captures, E) \rightarrow \{ return \ captures[1] + captures[2]; \}$

# Transformer API

---

```
Transformer.addTransformer(PatternTree pattern, BiFunction<Object[],  
Object, Object> action);
```

```
Transformer transformer = new Transformer(lexer);
```

```
transformer.addTransformer("(%Ex @ num @_ %)", (captures, E) -> {  
    return captures[1]; });
```

```
transformer.addTransformer("(%Ex @ expr @_ + @_ %)", (captures, E) -> {  
    return captures[1] + captures[2]; }
```

```
st = transformer.transform(st, E); // E is some arbitrary object used to  
    // pass information to actions
```

- Let [(p1, a1), (p2, a2), ..., (pn, an)] be the set of pattern/action pairs in a transformer
- if p1 matches st and returns captures c, then set st to a1(c, E)
- if p2 matches st and returns captures c, then set st to a2(c, E)
- ...
- if pn matches st and returns captures c, then set st to an(c, E)

Note: one can call `transformer.transform` on a subtree inside an action

# Sample Transformer for LET

---

See [src/main/java/atrai/interpreters/LET/LetInterpreter.java](#)

```
ANTLRTokenizer tokenizer = new ANTLRTokenizer(grammarName);
```

```
Transformer transformer = new Transformer(tokenizer);
```

```
transformer.addTransformer("(%LET @_ expr @_ + @_%)", (c, E) -> {  
    Location l = st.getLocationFromID((Integer) c[1]);  
    return i(transformer.transform(c[2], E), l) + i(transformer.transform(c[3], E), l);  
});
```

```
transformer.addTransformer("(%LET @_ expr ( @_ )%)", (c, E) -> {  
    Location l = st.getLocationFromID((Integer) c[1]);  
    return transformer.transform(c[2], E);  
});
```

```
transformer.addTransformer("(%LET @_ num @_%)", (c, E) -> {  
    Location l = st.getLocationFromID((Integer) c[1]);  
    return s2i(c[2], l);  
});
```

# Sample Transformer for LET

---

```
transformer.addTransformer("(%LET @_ iden @_ %)", (c, E) -> {  
    Location l = st.getLocationFromID((Integer) c[1]);  
    return e(E).get(s(transformer.transform(c[2], E), l));  
});
```

```
transformer.addTransformer("(%LET @_ expr let (%LET @_ iden @_ %) = @_ in @_ %)", (c, E) -> {  
    Location l = st.getLocationFromID((Integer) c[1]);  
    Environment Ep = Environment.extend(s(c[3], l), i(transformer.transform(c[4], E), l), e(E));  
    return transformer.transform(c[5], Ep);  
});
```

```
transformer.addTransformer("(%LET @_ expr if @_ then @_ else @_ %)", (c, E) -> {  
    Location l = st.getLocationFromID((Integer) c[1]);  
    if (b(transformer.transform(c[2], E), l)) {  
        return transformer.transform(c[3], E);  
    } else {  
        return transformer.transform(c[4], E);  
    }  
});
```

# DynamicTypeChecker.java

---

- See [src/main/java/atrai/interpreterscommon/DynamicTypeChecker.java](#) to see a dynamic type casting methods which we used in the LET interpreter.

*// Casts o to int and returns the int. Throws exception if casting fails.*

```
public static int i(Object o, Location location) {  
    if (o instanceof Integer) {  
        return (Integer) o;  
    } else {  
        throw new SemanticException("Dynamic type checking failed: expecting int instead of " +  
o, location);  
    }  
}
```

*// Casts o to Environment and return the Environment object. Throws exception if casting fails.*

```
public static Environment e(Object o) {  
    if (o == null) return null;  
    if (o instanceof Environment) {  
        return (Environment) o;  
    } else {  
        throw new RuntimeException("Internal error: expecting object of type Environment instead  
of " + o);  
    }  
}
```

# Debugging Transformer.transform

---

- If a transformer is not working as expected, we want to debug its behavior on a tree
- pass `-Ddebug1=true` to java, to print the (p,a) pairs that got successfully applied
- pass `-Ddebug2=true` to java, to print the (p,a) pairs that were not successfully applied

# Template

---

- Like a template string in regular expressions
  - remember s/pattern/template/ in regex
  - used in conjunction with a pattern
  - a template of an untyped tree that replaces the matched untyped tree
  - use to transform an untyped tree to another
- A template is again an untyped tree
  - with special tokens of the form \$n and \$\_n
  - \$n is replaced with the n<sup>th</sup> capture
  - \$\_n is replaced with the children of the n<sup>th</sup> capture

# Example of a Pattern/Template pair

---

Given grammar Ex:

An example pattern is (%Ex @\_ expr @\_ + @\_%)

An example template is (%Ex \$1 expr \$3 + \$2%)

The pattern matches the following Untyped Tree:

```
(%Ex 1 expr
  (%Ex 2 expr (%Ex 3 id x%)%)
  +
  (%Ex 4 expr (%Ex 5 num 10%)%)
%)
```

The transformed tree is

```
(%Ex 1 expr
  (%Ex 4 expr (%Ex 5 num 10%)%)
  +
  (%Ex 2 expr (%Ex 3 id x%)%)
%)
```



# Example of a Pattern/Template pair

---

Given grammar Ex:

An example pattern is (%Ex @\_ expr (%Ex @ num @\_%)%)

An example template is (%Ex \$1 num \$2%)

The pattern matches the following Untyped Tree:

(%Ex 2 expr (%Ex 3 num 10%)%)

The transformed tree is

(%Ex 2 num 10%)

# Template API

---

```
class Template extends Tree {  
    // create a template from a string and a lexer  
    static Template parse(String template, Lexer lexer);  
    // construct a tree by replacing $n with nth capture in captures  
    // return the resulting tree  
    Object replace(Object[] captures);  
}
```

Check [javadocs/index.html](http://javadocs/index.html) for further documentation

See [src/main/java/atra/core/Template.java](http://src/main/java/atra/core/Template.java) for source

See [src/test/java/atrai/core/TemplateTest.java](http://src/test/java/atrai/core/TemplateTest.java) for examples

# Pattern/Template in Atrai

---

```
// create a lexer used to tokenize strings in a Pattern
ANTLRTokenizer lexer = new ANTLRTokenizer("Ex");
// create a pattern
Pattern p = Pattern.parse("(% @_ expr (%Ex @ num @_%)%)", lexer);
// match the pattern against a Untyped tree st
// returns null if match fails
Object[] captures = p.match(st);

// create a template
Template r = Template.parse("(% $1 num $2%)", lexer);
// create the transformed tree
st = r.replace(st);
```

# An Example Usage of Pattern/Template API

---

```
public void test3() throws Exception {
    String pattern = "(% @_ @_ %)";
    String source = "(%hello (%world X%)%)";
    String template = "(% begin ` $1 $2 ` $3_ end %)";
    Lexer lexer = new SimpleStringTokenizer();
    UntypedTree s = UntypedTree.parse(source, lexer);
    Pattern p = Pattern.parse(pattern, lexer);
    Template t = Template.parse(template, lexer);
    Object t = t.replace(p.match(s));
    System.out.println(t);
    assertEquals("(%begin ` $1 (%world X%) ` $3_ end%)", t.toString());
}
```

` \$1 are not considered as holes because ` escapes \$.

See [src/test/java/atrait/core/TemplateTest.java](#) for more examples

# Usage of `$_n` in Template

---

```
public void test1() throws Exception {  
    String pattern = "@_";  
    String source = "(%hello world%)";  
    String template = "(% begin $_1 end %)";  
    Lexer lexer = new SimpleStringTokenizer();  
    UntypedTree s = UntypedTree.parse(source, lexer);  
    Pattern p = Pattern.parse(pattern, lexer);  
    Template t = Template.parse(template, lexer);  
    Object t = t.replace(p.match(s));  
    assertEquals("(%begin hello world end%", t.toString());  
}
```

`p.match(s)` returns `captures` where `captures[0] = captures[1] = (%hello world%)`

`$_1` gets replaced by the children of `(%hello world%)`