github.com

Tran Language Definition

11–13 minutes

Introduction:

Tran is an object-oriented language that has some significant differences from other languages that you may know of. The biggest differences are:

1. Tran methods can have multiple return types
2. Tran doesn't use curly braces `{ }` for blocks, instead it uses indentation
3. Tran while/for/do-while loops are much simpler than Java or C loops
4. Tran variables are private unless otherwise specified by an `accessor` or `mutator` keyword
5. Tran uses the keyword `shared` to represent static methods; it can be used for variables but it cannot be used for classes

Primitive Types

Data Types

- `number` (floating point)
- `boolean` (constant values: true, false)
- `string` (an arbitrarily large string of characters)
- `character` (a single number/letter/symbol)

Operators and Comparisons

Description

- `number` has the following operators: `+`, `−`, `∗`, `/`, `%`. The order of operations is: `parenthesis`, `∗`, `/`, `%`, `+`, `−` all done left to right. These operations also have left associativity.
- `boolean` has the following operators: `! (not)`, `&& (and)`, `|| (or)`. The order of operations is: `!` then && then `||`. The `!` has right associativity while the other two have left associativity.
- `character` has no operators.
- `string` only has + for the concatenation of characters or strings. It is left associative.
- Comparisons can only take place between the same data types.
- The comparison operators are: `== (equals)`, `!= (not equal)`, <, <=, >, >=. These are all done from left to right.

Variable Assignment

Description

In Tran, a variable/member cannot be assigned to the same line that it has been declared. To properly assign a variable or member, it must be done over two lines. Tran allows for multiple variables of the same type to be declared in the same line however, you CAN'T assign them in the same line as well.

Correct Examples

OR

OR

```
number k,e,n,x
k = 10.132023
e = 4.122024
n = 7.052024
x = 7.232021
```

Incorrect Example

```
number a = 10.132023, b = 4.122024, c = 7.052024
```

Blocks

Description

Blocks are one or more statements that are run one after another. Blocks of code are identified by an indentation level deeper than that of the enclosing owner. To mark the end of a block, “un-indent” one level. Indents on empty lines or ones with only comments are not counted.

Example

```
number y
y=10
loop y.times()
        number x
        x = 5
        console.print(In the loop block: + x)
console.print(Out of the block)
```

Classes

Description

In Tran, classes are defined using the `class` keyword followed by the class name. The class body can then contain fields, methods, constructors, and other member declarations. You can only have one class in a file.

Example

```
class Tran
        number x
        string y
        construct()
                x = 0
                y = ""
```

New

Description

In Tran, the `new` keyword creates an object of the specified class. It allocates the memory for the object and creates a reference to the object. The reference can then be used to access properties and methods. To learn more about the usage of `new` in Tran go here -> [Accessors and Mutators]

Example

```
Tran instanceOfTran
instanceOfTran = new Tran()
```

Constructors

Description

In Tran, a constructor is a special method used to initialize objects and is called when an instance of a class has been created. They are defined using the keyword `construct()`. Constructors can be overloaded, providing the ability to initialize these objects with different parameters.

Example

```
construct()
        x = 0
        y = ""
{Below is an example of an overloaded constructor}
construct(number n)
        x = n
        y = ""
```

Comments

Description

They start with `{` and end with `}`. Comments can span multiple lines. Comments can be nested.

Example

```
{This is a comment}
{This is {also} a comment}
{This
is a
comment that
spans multiple lines}
```

Statement Types

If Statements

Description

In Tran, an `if` statement works almost exactly like that in Java. The only difference lies in the syntax, in Java, the condition is surrounded by parentheses, whereas in Tran, it is not. Alongside these `if` statements, Tran also has `else` and `else if` statements that work like those in Java.

Example

```
number n
n = 200
if n > 100
	n--
else if n < 50
	n++
else
	console.print (Not in either case!)
```

Loops

Description

In Tran, the `loop` construct replaces traditional for, while, and do-while loops. The condition can either be an iterator or a boolean. When using an iterator, the `loop` will continue until the iterator reaches its end, effectively evaluating to false. For boolean conditions, the `loop` will run until the condition evaluates to false. Finally, loops can also be assigned to a variable. To learn more about the usage of the built-in `times()` for loops go here -> Functions Available

Examples

```
boolean keepGoing
keepGoing = true
loop keepGoing
       if n >= 65
       keepGoing = false
```

OR

```
number x
x = 10
loop x.times()
       console.print (WLR)
```

OR

```
number temp
number x
x = 18
       loop temp = x.times() {the loop is being
assigned to the variable 'temp'}
              console.print (temp)
```

Method Overview

Description

Tran methods are made of the following components:

- An Optional Access Modifier:

- `shared` which works like `static` in Java, allowing the user to create variables and methods that belong to the class itself rather than for individual objects. This allows for the following benefits:
- Shared Data: Allows variables to be shared across all instances of a class.
- Utility Methods: Useful for methods that perform common tasks, like mathematical operations, without needing an instance.
- Constants: Helps define constants that are shared among all instances of a class.
- `private` which restricts access to the method or variable, making it only accessible to the class it belongs to.
- A method that is defined by: `methodName()`
- Parameters (optional): Input values that the method can accept are specified after the colon in the method declaration. They allow methods to receive data from the caller and can be used within the method to perform tasks or to do calculations. If there are multiple parameters, they are separated with commas.
- Return Types (optional): When the variable is set in the method (e.g. t="some words"), that value is returned by the method. There can be multiple return types.
- Body: Contains all the code within one indentation level of the method. This can include variable declarations, loops, if-else statements, and any operations, calculations, or logic you need to implement.

Example

```
shared methodName() : number n, string t {The right
side of the colon are return values}
        n = x.clone()
        t = “some words” {These two variables are
return values in Tran and will be returned by this
particular method}
```

Method Calls

Description

In Tran, a method call involves executing a predefined block of code by its name and passing in any required arguments. When a method is invoked, it can return multiple types or no return type at all (similar to void in Java). Variables declared within a method can be assigned the output of a method call, allowing for an infinite number of variables to be assigned as long as their types match. Additionally, interfaces can be passed as parameters to a method.

This is an example of a method call with no return type, notice the absence of any arguments

```
printNumbers()
        number temp
        loop temp = x.times()
                console.print (temp)
main()
        printNumbers()
```

This is an example of a method call with one return type

```
square() : number s
        s = x * x
```

```
main()
       number t
       t = 3.07
       t = square()
```

This is an example of a method call with multiple return types

```
allMyData() : number n, string t
       t = y.clone()
       n = x
main()
       number num
       string s
       num = 963
       s = more
       num, s = allMyData()
```

This is an example of a method call where the return is assigned to numerous variables

```

getNumbers() : number a, number b, number c
       a = 1
       b = 10
       c = 100

main()
       number w
       number l
       number r
       w,l,r = getNumbers()
```

Built-in Objects

Methods Available

- `times()` is available on number objects. It returns an iterator that generates a sequence from 1 to the specified number. This is useful for creating loops that run a specific number of times.

Example of `times()`

```
number n
n=10
loop n.times()
       console.print(This is iteration number + n)
{This will run ten times}
```

- `clone()` is available on all objects, it creates a copy of the object made previously. This can be useful when you want to preserve any changes to the original object, allowing you to work with a modified version while keeping the initial data intact.

Example of `clone()`

```
number n
n = 42
string s
s = n.clone()
console.print (s)  {Output: 42}
```

- `getDate()` is provided by the clock object. It returns the current date and time as a datetime object.

Example of `getDate()`

```
datetime now = clock.getDate()
console.print(now)
```

Accessor and Mutators

Description

In Tran, accessors and mutators are used to manage the values of private fields within a class, providing controlled ways to read and modify these values. Below is more information for each and how they are used in Tran.

Accessors

Overview

In Tran, the `accessor` is designed to provide controlled access to typically private variables. This allows users to interact with these variables by defining an access parameter. The `accessor` is directly associated with the variable declared in the previous line and within the indented block it has been declared in. The colon indicates the start of the block that defines the rest of the `accessor`. `value` is the variable used to represent the data that is being interacted with, in this case where it is retrieved.

Example

```
string y
    accessor: value = y.clone()
```

Example 2

```
number x
```

```
       accessor: value = x.clone()
              loop x.times()
                     value = value + 1
```

Mutators

Overview

In Tran, the `mutator` allows controlled modification of typically private variables. This feature enables users to specify how a variable can be updated, ensuring that changes are properly saved. The `mutator` is directly associated with the variable declared in the previous line and within the indented block it has been declared in. The colon indicates the start of the block that defines the rest of the `mutator`. `value` is the variable used to represent the data that is being interacted with, in this case where it is assigned.

Example

```
string y
   mutator: y = value
```

Example 2

```
number x
       mutator: x = value
              loop x.times()
                     value = value + 1
```

Example of Using Both

```
Tran instanceOfTran
```

```
instanceOfTran = new Tran()
number x
string y
       accessor: value = y.clone()
       mutator: y = value
{By creating an accessor and mutator, we can access
values such as these:}
       instanceOfTran.x = 5
       console.print (instanceOfTran.y)
```

Interfaces

Description

In Tran, an `interface` is a set of methods and members that a class must implement. Unlike some other languages, however, this language does not support inheritance. Classes can support as many interfaces as the user would like to implement. Interfaces outline a class's structure without dictating the specifics of how the methods need to be implemented.

Example

```
interface someName
    square() : number s

class TranExample implements someName
    start()
        number x
        x = 10
        number y
        y = square(x)
        console.print(y)

    square(number x) : number s {The method we
defined in the interface is used here!}
        s = x*x
```