# Tran Language Definition – Spring 2025

## 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, like Python
3. Tran members are private
4. Tran uses the keyword **shared** to represent static methods; it can be used for variables, but it cannot be used for classes
5. Tran methods are public unless specified with the **private** keyword.
6. There is one type of loop (**loop**) instead of for, while, do-while.

## Included Types

* number (floating point) – this is a primitive type like Java
  + The math operations work like you would expect from Java (+ - \* / %)
* string (an arbitrarily large string of characters) – this is a class, like Java
* character (a single number/letter/symbol) – this is a primitive type, like Java

boolean is included but is a little different than you might expect. It is a class instead of a primitive type.

## Classes

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 Example1

number x

string y

construct() {called when someone uses “new” }

x = 0

y = ""

## Blocks

Blocks are one or more statements that are run consecutively. Blocks of code are identified by indentation one level deeper (further right) than that of the enclosing owner. To mark the end of a block, “dedent” one level. Indents on empty lines or ones with only comments are not counted.

### Example

class Example2

number x

string y

construct() {called when someone uses “new” }

x = 0

y = ""

doSomeWork()

x=10

loop x.times()

console.print(“In The Block”)

console.print(“Out of the block”)

## New

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.

### Example

Student instanceOfStudent

instanceOfStudent = new Student ()

## Constructors

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

class Example3

number x

string y

construct() {called when someone uses “new” }

x = 0

y = ""

construct(number n) {An example of an overloaded constructor}

x = n

y = ""

## Comments

Comments 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

## Variable Assignment

A variable or member can be assigned on the same line that it has been declared, or after declaration. Tran allows for multiple variables of the same type to be declared in the same line as well, supporting initialization at declaration in that as well.

### Example

string s

s = "agc"

number x

x = 4656

number k,e,n,x

k = 10.132023

e = 4.122024

n = 7.052024

x = 7.232021

number d = 42

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

## If Statements

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. Tran also has else and else if statements that work like those in Java.

**Example**

number n

n = 200

if n > 100

n = n -1

else if n < 50

n=n+1

else

console.print (“Not in either case!”)

## Loops

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, the loop value can also be assigned to a variable; this allows you to use the value you are looping over inside the loop.

### Examples

boolean keepGoing = boolean.true

loop keepGoing

if n >= 65

keepGoing = boolean.false

number x = 10

loop x.times()

console.print (“Hello”)

number temp

number x = 18

loop temp = x.times() {the loop is being assigned to the variable 'temp'}

console.print (temp)

## Method Overview

Methods are made of the following components:

* (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.
* Name – a descriptive name for the method
* Parameters (optional): Input values that the method can accept are specified inside parenthesis 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 Values (optional): When the return variable is set in the method (e.g. t="some words"), that value is returned by the method. There can be multiple return types. Return types are specified after the colon in the method header.
  + Note – there is no “return”, nor is one needed. You simply set the return values in the method and when the statements are all executed, Tran will return to the calling method.
* 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.

### Examples

class Example4

number a

{doesn’t require an instance of Example. To use it: Example.helloWorld() }

shared helloWorld() { no parameters, no return value, public}

console.print(“Hello World”)

add(number a, number b) : number sum {parameters, return values, public}

sum = a + b

private setA() {can only be called inside the class}

a = 42

## Method Calls

A method call involves executing a predefined block of code by its name and passing in any required arguments. When a method is called, 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

class Example5

printNumbers()

number temp

loop temp = x.times()

console.print (temp)

shared main()

printNumbers()

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

class Example6  
 shared square(number x) : number s  
 s = x \* x  
 shared start()  
 number t  
 t = 3.07  
 t = Example6.square(t)

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

class Example7

shared allMyData(number x) : number n, string t

t = "hello"

n = x

shared start()

number num

string s

num, s = Example7.allMyData(45)

## 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}

* console is a class with a shared method

print(string printMe)

* console.print outputs to the console (like Java’s system.out.println)
* boolean is an included class:

class boolean

shared boolean true

accessor: { makes this publicly readable}

value = true

shared boolean false

accessor: { makes this publicly readable}

value = false

shared not(boolean value) : boolean result

if value == true

result = false

else

result = true

shared and (boolean a, boolean b) : boolean result

if a == false

result = false

else

if b == false

result = false

else

result = true

shared or (boolean a, boolean b) : boolean result

if a == true

result = true

else

if b == true

result = true

else

result = false

## **NOTE** – You are not implementing accessors and mutators for your assignment

## 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

shared start()

number x = 10

number y = square(x)

console.print(y)

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

s = x\*x

class UseTranExample

someMethod()

someName t = new TranExample() {t is an instance of someName}

t.square(20)

### Operators and Comparisons

1. number has the following operators: +, -, \*, /, %. The order of operations is:  
   1) parenthesis  
   2) \*, /, %  
   3) +, -  
   all done left to right. These operations also have left associativity.
2. character has no operators.
3. string only has + for the concatenation of characters or strings. It is left associative.
4. Comparisons can only take place between the same data types.
5. The comparison operators are: == (equals), != (not equal), <, <=, >, >=. These are all done from left to right.