Seaflow Programming Language Proposal

Rohan Arora, Junyang Jin, Ho Sanlok Lee, Sarah Seidman
ra3091, jj3132, hl3436, ss5311

Overview

Modern applications handle many asynchronous events, but it is difficult to model such applications using programming languages such as Java and JavaScript. One popular solution among the developers is to use ReactiveX implementations in their respective languages to architect event-driven reactive models. However, since Java and JavaScript are not designed for reactive programming, it leads to complex implementations where multiple programming styles are mixed-used. Seaflow is a language designed to address this issue by supporting some of the core principles of ReactiveX and reactive programming natively.

Our goals include:

1. All data types are immutable, with the exception being observables, no pointers
2. The creation of an observable should be simple
3. Natively support core principles in the ReactiveX specification

Language Details

Basic Syntax

Seaflow is statically and explicitly typed; variable types must be declared at compile time, and different types cannot be assigned to each other without casting.

```
int add(int a, int b) {
    int c = a + b;
    return c;
}

int a = 0;
double b = add(a + 5); // error
```

Comments

```
// single-line comment

/*
 * multi-line comment
 */
```

Data Types and Operations

Seaflow supports following primitive types: integer, long, short, character, double, float

Primitive types can be compared with ==, !=, <=, >=, &&, || and manipulated with *, /, +, -, <<, >>, |, &. They are all signed.

Data Type	Description	Examples

`int`	An integer type, 4 bytes	`int x = 3;`
		`0 || x;  // True`
		`x * 4;  // 12`
		`x > 4;  // False`

`long`	A long integer, 8 bytes	`long a = 4;`
		`long b = 3;`
		`0 && a;  // True`
		`a * b;  // 12`
		`b > a;  // False`

`short`	A short integer, 2 bytes	`short a = 4;`
		`short b = 3;`
		`0 && a;  // True`
		`a * b;  // 12`
		`b > a;  // False`

`char`	An integer, 1 byte	`'a' == 'a';  // True`
		`'a' == 'b';  // False`

`double`	A floating point type, 8 bytes	`double a = 3.0;`
		`a == 3;  // True`
		`double b = a * 4; // 12.0, converted`
		`3.0 > 4.0;  // False`

`float`	A floating point type, 4 bytes	`float a = 3.0;`
		`a == 3;  // True`
		`float b = a * 4; // 12.0, converted`
		`a > 4.0;  // False`

Structs and arrays may be composed of these primitive types. Missing fields or values will be initialized to 0.

```
/* Structs */
struct foo {
    int field1;
    char field2;
    char[] name;
};

struct foo bar = { 16, ‘a’, “example” };

/* Arrays */
int[] arr = { 1, 2, 3, 4, 5 };

/* Arrays support length, indexing, concatenation and slicing */

int[] myarr = {1, 2, 3};
int[] myarr2 = {4, 5, 6};

int myarrLength = myarr.length;

int[] all = myarr + myarr2; // {1,2,3,4,5,6}

int x = myarr[0]; // 1
int[] slice = all[0:2]; // {1,2}

char[] str = “a string”;
```

Reserved Keywords

The following are reserved words:

`break, char, continue, do, double, else, float, for, if, int, long, print, return, short, sizeof, struct, typedef, void, while, ($)`

typedef

Seaflow supports typedefs.

```
typedef int size_t;
struct my_struct_type my_struct_variable;
typedef struct my_struct_type my_short_type_t;
```

Control Flow, Loops

Seaflow does not support if “statement”. “if” in Seaflow is an expression and must be evaluated to a value. Parentheses are always required for conditions, but braces are not necessary for single-line values.

```
char grade = if (score > 92) ‘A’ else if (score > 85) ‘B’ else ‘F’;

if (score > 92) {       // not supported
    grade = ‘A’;
}
```

The way Seaflow supports loops is still under discussion.

```
int $i = 0

void helloWorld(int i) {
     print("Hello World!");
}

$i.subscribe(helloWorld, $i < 10, $i + 1); // for loop
$i.subscribe(helloWorld, $i == 10); // infinite while loop
```

Core Language Features

Immutability

All objects are immutable after they are created. Moreover, reassigning to an existing name is also not allowed, so that developers are naturally forced to write *pure* codes.

```
int a = 10;
```

```
a = 5;   // error

int arr = [1, 2, 3];
arr.add(4);       // not supported
arr[2] = 4;       // error
arr.remove();     // not supported

user.username = "Joe";  // error
```

Higher Order Functions

Seaflow supports higher order functions. Functions must be typed.

```
int function(int x, (int)->int func) {
    return func(x);
}

function(10, (x)->{ x + 10 });
```

Observables

Observable in Seaflow is an implementation of a particular hot observable in ReactiveX specification called BehaviorSubject. We can declare an observable instance of any base type by using a $ sign.

```
int $a;  // initializing an int-type observable instance, a
int $b = 5  // initializing an observable with initial value

$b = 3;  // re-assigning is possible with observable types
```

A function with a single input type can subscribe to the observable by calling `.subscribe`:

- `.subscribe((T)->void func)`
    - `.subscribe` takes a function of type `(T)->void` as the input. This function is invoked when the observable emits a value.

```
void observer(int num) {
    print(num);
}

$a.subscribe(observer);
```

In addition to `.subscribe`, all observables of type `T` support following functions:

- `.map((T)->X func)`
    - Takes a function of type `(T)->X` as the input, where X could be any type. `.map` function returns another observable with type X. The function `func` is called for each upstream value and the returned value will be passed to the downstream.

```
int $c = $b.map((x)->{x + 10});

// or

$c = $b + 10;
```

- `.combine($T obs, (T, T)->X func)`
    - Takes another observable of type `T` and a function of type `(T, T)->X` func. `.combine` function returns a new observable with type `X` and the returned observable is subscribed to the observable and the obs.

```
int $d = $b.combine($c, (x, y)->{ x + y });

// or

$d = $b + $c;
```

- `.complete()`
    - Removes all subscriptions that the observable currently has.

```
$d.complete();
```

Memory

Seaflow is not garbage collected. Primitive types are placed on the stack, Arrays, Structs and Observables are placed on the heap. Developers can manually free heap objects using `free`.

```
free($user);
```

Next Steps

Currently, Seaflow is designed to be a single-threaded language. Our next goal is to allow multi-threaded programming by supporting “.observeOn” in ReactiveX specification. With “.observeOn” programmers can specify the thread that executes downstream.

```
$user.map(...)           // runs on calling thread
    .filter(...)         // runs on calling thread
    .observeOn(Thread.IO)
    .filter(...)         // runs on IO thread
    .map(...)            // runs on IO thread
    .observeOn(Thread.main)
    .subscribe(...)      // runs on Main thread
```