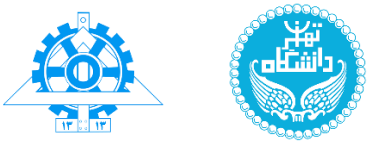

Sophia

Documentation

Amir PourMohammadAli
Mobina Shahbandeh

فهرست مطالب

۱. مقدمه

زبان Sophia یک زبان شی‌گراست و برخی از ویژگی‌های زبان‌های شی‌گرا نظیر ارث‌بری را داراست.

در این زبان، یک کلاس اصلی وجود دارد که در آن یک متد constructor پیاده‌سازی شده‌است. برنامه‌هایی که در این زبان نوشته می‌شوند، در هنگام اجرا دستورات درون این متد را اجرا می‌کنند.

۲. ساختار کلی

در این زبان، کد برنامه درون یک فایل با پسوند sop. قرار دارد. این فایل شامل:

- یک یا چند کلاس
- یک کلاس اصلی (Main) شامل یک متد constructor
- هر کلاس شامل تعدادی متغیر و متد

۲ـ۱. قواعد کلی نحو

زبان Sophia به بزرگ و کوچک بودن حروف حساس است. در این زبان، وجود کاراکتر های tab و space تاثیری در خروجی برنامه ندارند. جزئیات مربوط به scope و خطوط برنامه در ادامه توضیح داده خواهد شد.

۲ـ۲. کامنت‌ها

در این زبان کامنت‌ها تک خطی هستند و تمامی کاراکترهای بعد از // تا انتهای خط کامنت به حساب می‌آیند.

۲ـ۳. قواعد نامگذاری کلاس‌ها، متدها و متغیرها

اسامی انتخابی برای نام‌گذاری باید از قواعد زیر پیروی کنند:

- تنها از کاراکترهای a..z، A..Z ، _ و ارقام تشکیل شده باشند (محدودیتی روی تعداد کاراکترهای یک اسم در زبان Sophia وجود ندارد).
- با رقم شروع نشوند.
- معادل کلیدواژه‌ها نباشند. در جدول زیر تمامی کلیدواژه‌های زبان Sophia آمده است:

class	extends	this	def	func	return
if	else	for	foreach	continue	break
false	true	int	boolean	string	void
list	in	null	new	print	

- نام هر کلاس یکتاست.
- نام هر متد در هر کلاس یکتاست. به عبارتی method overloading در این زبان وجود ندارد.
- نام هر متغیر در هر scope یکتاست ولی در scope های درونی تر از نام های متغیر بیرونی میتوان استفاده کرد. در نتیجه در طول آن scope متغیر درونی هنگام استفاده ارجحیت دارد.
- امکان تعریف دو متد و متغیر با نام یکسان در یک کلاس وجود ندارد.

۳. کلاس‌ها و متدها

در تعریف هر کلاس، در ابتدا نام آن و نام کلاسی که از آن ارث بری می‌کند(در صورت وجود) مشخص می‌گردد. سپس scope این کلاس توسط { } مشخص می‌شود.

```
1 class A extends B {
2     // class body
3 }
```

متغیرها و متدهای هر کلاس می‌توانند با هر ترتیبی در scope آن کلاس تعریف شوند. مثال زیر نمونه‌ای از تعریف یک کلاس است:

```
class A {
  x: int;
  y: bool;
  def A(p:int, q: bool) {
    this.x = p;
    this.y = q;
  }
  def void foo() {
    print("foo");
  }
}
```

Methods

کلاس Main، باید حتما یک constructor بدون آرگومان داشته باشد. کلاس Main فرزند هیچ کلاس دیگری نیست. کلاس‌ها می‌توانند یک یا چند متغیر و متد(تابع) داشته باشند. در هر متد نیز تعریف متغیرها در ابتدای آن و قبل از همه‌ی دستورات انجام می‌شود. همچنین در این زبان، همه متدها بجز constructor باید مقدار بازگشتی داشته باشند و در صورتی که مقدار بازگشتی متدها از نوع void نباشد، باید یک دستور return در متد وجود داشته باشد. آرگومان‌های متدها به صورت pass by value به متدها داده می‌شوند.

به عنوان مثال، تعریف یک متد constructor به صورت زیر انجام می‌شود:

```
def A(a: int, b:string) {
    c: int;
    c = a * 5 % 2;
    this.a = c;
    this.b = b;
}
```

نمونه ای از تعریف متدهای دیگر:

```
def int foo(n: int) {
    print("foo");
    return n + 1;
}
```

برای فراخوانی متدها از () استفاده می‌کنیم. به مثال زیر توجه کنید:

```
x: int;
x = this.foo(2);
```

برای ساختن یک نمونه[1] جدید از یک کلاس، به صورت زیر از کلیدواژه new برای آن استفاده می‌کنیم:

```
a: A;
a = new A(2, "Sophia");
```

۳ـ۱. ارث بری

هر کلاسی می‌تواند از حداکثر یک کلاس دیگر ارث‌بری کند(با استفاده از کلیدواژه extends) و از فیلدها و متدهای آن استفاده کند. قابلیت overriding برای متدهای یک کلاس وجود ندارد. همچنین overloading برای متغیرهای کلاس[2] در این زبان وجود ندارد.

[1] instance

[2] برد. امکان تعریف متغیر همنام در یک کلاس و کلاسی که از آن به ارث می

۳ـ۲. کلیدواژه this

کلیدواژه this به کلاسی که در آن قرار داریم اشاره می‌کند و با استفاده از این کلیدواژه می‌توان به فیلدها و متدهای آن دسترسی پیدا کرد.

۴. متغیرها

تعریف متغیرها در زبان Sophia به صورت زیر انجام می‌شود:

```
identifier: type;
```

در این زبان نمی‌توان چندین متغیر را در یک خط تعریف کرد و یا در هنگام تعریف یک متغیر، آن را مقداردهی کرد.

در صورتی که به متغیری مقداری نسبت داده نشود، مقدار آن برابر با مقدار پیش‌فرض تایپ خود در نظر گرفته می‌شود. مقادیر پیش‌فرض تایپ‌های مختلف در جدول زیر آمده‌است:

int	0
bool	false
string	""

در صورتی که متغیر از جنس یک کلاس یا اشاره‌گر تابع باشد، مقدار اولیه آن null و در صورتی که لیست باشد، مقدار اولیه هرکدام از اعضای آن مطابق با مقدار اولیه تایپ آن عضو می‌شود.

۴ـ۱. متغیرهای محلی

تمامی متغیرهای محلی فقط در ابتدای متدها تعریف می‌شوند و در ابتدای هیچ scope جدید دیگری قابلیت تعریف متغیر محلی وجود ندارد.

یک متغیر در صورتی می‌تواند به جای دیگری استفاده شود (مثلا در آرگومان‌های توابع) که نوع آن زیرنوع[1] متغیر دیگر باشد.

۵. انواع داده

۵ـ۱. تایپ‌های پایه

در زبان Sophia، سه تایپ پایه[2] int، string و bool وجود دارند. در متغیرهای از نوع تایپ‌های پایه، خود مقادیر ذخیره می‌شود و نه اشاره‌گری به خانه‌ای از حافظه. علاوه بر آن‌ها، هر متغیر می‌تواند از جنس یکی از کلاس‌هایی باشد که در برنامه تعریف شده‌است. همچنین تایپ void نشانگر مقدار بازگشتی توابع بدون خروجی است. استفاده از این تایپ به عنوان عملوند و تعریف متغیر از این تایپ غیرمجاز است. تایپ‌های دیگر در ادامه آمده است.

۵ـ۲. اشاره‌گر تابع[3]

یک متغیر از نوع اشاره‌گر تابع، به یک تابع(متد) در برنامه اشاره می‌کند. برای مثال یک متغیر از جنس `<int,string->int>` به یک تابع با ورودی‌های به ترتیب از جنس int و string و خروجی از نوع int اشاره می‌کند. برای توصیف تابعی بدون ورودی و خروجی از `<void->void>` استفاده می‌کنیم. دقت کنید که توصیفاتی از قبیل `<void,int->string>` نامعتبر هستند. برای فراخوانی تابعی که اشاره‌گر به آن اشاره می‌کند، از () استفاده می‌کنیم. به مثال زیر توجه کنید:

[1] subtype
[2] primitive
[3] function pointer

```
x: int;
fptr: func<int->int>;
fptr = this.foo;
x = fptr();
```

۵ـ۳. لیست

لیست مجموعه‌ی مرتبی با اندازه‌ی مشخص از انواع داده‌های مختلف است. در طی اجرای برنامه اندازه‌ی یک لیست ثابت باقی می‌ماند. برای مثال، `[1, "Sophia", [2, 3]]` نمایشگر یک لیست سه تایی است که عنصر آخر آن خود یک لیست دوتایی دیگر است. در حالت کلی لیست را به صورت لیستی از نوع داده‌های درون آن تعریف می‌کنیم. به عنوان مثال:

```
myList: list(int, func<int->int>, bool);
myList = [2, this.foo, true];
```

در تعریف لیست می‌توان به بعضی یا تمام اعضای آن لیست شناسه نسبت داد:

```
myList: list(number: int, fptr: func<int->int>, isEven: bool);
myList.number = 2;
myList.fptr = this.foo;
myList[2] = true;
```

در صورتی که تمامی اعضای یک لیست از یک نوع باشند، می‌توان آن را به صورت زیر تعریف کرد. در تعریف لیست‌ها به صورت زیر، طول لیست یک عدد ثابت است و استفاده از متغیرها در آن مجاز نیست. طول لیست نمی‌تواند صفر یا عددی منفی باشد.

```
myList2: list(2 # string);
myList2 = ["foo", "bar"];
```

برای دسترسی به عناصر لیست می‌توان از شناسه آنها(در صورت وجود) و یا اندیس آنها(که از ۰ شروع می‌شود) استفاده کرد. اندیس‌ها از جنس int هستند. بررسی مجاز بودن مقدار عددی اندیس و خارج از محدوده نبودن آن، به صورت runtime است. هنگام دسترسی به اعضای لیست، در صورتی که اعضای لیست نوع یکسانی نداشته باشند (لیست به صورت `type`) `#` `list(length` تعریف نشده باشد) اندیس‌ها باید عدد ثابت باشند.

نوع یک لیست زیرنوعی از نوع لیست دیگر است، اگر تعداد اعضای آن با لیست دیگر برابر و نوع اعضای آن زیرنوعی از نوع اعضای متناظر در لیست دیگر باشد. به مثال زیر توجه کنید که در آن کلاس A از کلاس B ارث‌بری کرده است:

```
myList1: list(radius:int, center: list(x:int, y:int), widget:B);
myList2: list(border:int, list(int, int), A);
myList3: list(int, list(int, int), int, A);
myList4: list(bool, list(int, int), string);
```

در این مثال، نوع myList2 زیرنوعی از myList1 است اما این موضوع برای myList3 و myList4 برقرار نیست.

در صورتی که یک لیست(لیست اول) زیرنوعی از لیست دیگر(لیست دوم) باشد، می‌توان آن را به جای لیست دوم استفاده کرد. در مثال زیر، در صورتی که یک تابع آرگومانی از نوع myList1 داشته باشد، می‌توان یک متغیر از جنس myList2 را به آن داد. اما در نهایت اسامی اعضای لیست همان اسامی تایپ نهایی (در اینجا myList1) خواهد بود. در این مثال هم کلاس A از کلاس B ارث‌بری کرده است.

```sophia
myList1: list(radius:int, center: list(x:int, y:int), widget:B);
myList2: list(border:int, list(int, int), A);
myList1 = myList2;
print(myList1.radius); // valid reference
print(myList1.border); // invalid reference
```

اگر اعضای متناظر دو لیست با یکدیگر برابر باشند، آن دو لیست مساوی هستند.

۶. عملگرها

عملگرها در زبان Sophia به چهار دسته‌ی عملگرهای حسابی، مقایسه ای، منطقی و تخصیص تقسیم می‌شوند که شرح آنها در ادامه آمده‌است. توجه داشته باشید که عباراتی مانند i++ گزاره نیستند.

۶ـ۱. عملگرهای حسابی

این دسته از عملگرها تنها روی اعداد عمل می‌کنند. لیست این عملگرها در جدول زیر آمده است. در مثال های استفاده شده A را برابر با ۲۰ و B را برابر با ۱۰ درنظر بگیرید.

عملگر	شرکت‌پذیری	توضیح	مثال
+	چپ	جمع	A + B = 30
-	چپ	تفریق	B - A = -10
*	چپ	ضرب	A * B = 200
/	چپ	تقسیم	A / B = 2 B / A = 0
%	چپ	باقی‌مانده	A % B = 10
-	راست	منفی تک‌عملوندی پیشوندی	A = -20

-- و ++	راست	پیشوندی	--A
-- و ++	چپ	پسوندی	A++

۶ـ۲. عملگرهای مقایسه‌ای

این عملگرها وظیفه ی مقایسه را دارند؛ پس نتیجه ی آنها باید مقدار true یا false باشد. یعنی خروجی آنها یک bool است.

توجه داشته باشید که عملوندهای عملگرهای < و > تنها از جنس اعداد صحیح هستند. دو عملگر == و =! روی تمامی تایپ‌ها تعریف شده‌اند و باید تایپ عملوندهای آنها یکسان باشد. در صورت لیست بودن، باید تعداد اعضای دو لیست و نوع اعضای متناظر آنها برابر باشد. در غیر این صورت باید خطای کامپایل گرفته شود. توجه داشته باشید که هر متغیر می‌تواند با استفاده از عملگرهای == و =! با مقدار اولیه تایپ خود نیز مقایسه شود.

لیست عملگرهای مقایسه ای در جدول زیر آمده است. مقادیر A و B را همانند قبل درنظر بگیرید.

عملگر	شرکت‌پذیری	توضیح	مثال
==	چپ	تساوی	`(A == B) = false`
=!	چپ	عدم تساوی	`(A != B) = true`
>	چپ	کوچکتر	`(A < B) = false`
<	چپ	بزرگتر	`(A > B) = true`

۶ـ۳. عملگرهای منطقی

در این زبان، عملگرهای منطقی تنها روی تایپ bool قابل اعمال است. این عملگرها در جدول زیر آمده است. A را برابر true و B را برابر false در نظر بگیرید.

عملگر	شرکت‌پذیری	توضیح	مثال
&&	چپ	عطف منطقی	(A && B) = false
||	چپ	فصل منطقی	(A || B) = true
!	راست	نقیض منطقی	(!A) = false

۶ـ۴. عملگر تخصیص

این عملگر که به صورت = نمایش داده می‌شود وظیفه ی تخصیص را برعهده دارد. یعنی مقدار عملوند سمت راست را به عملوند سمت چپ اختصاص می‌دهد. برای لیست مقدار تک تک عناصر لیست سمت راست را به عناصر لیست سمت چپ تخصیص می‌دهد.

دقت داشته باشید که عملوند سمت چپ باید از نوع lvalue باشد. مفهوم lvalue و rvalue در زبان Sophia مشابه زبان C است. عبارات lvalue عباراتی هستند که به یک مکان در حافظه اشاره می‌کنند. در مقابل عبارات rvalue به مکان خاصی در حافظه اشاره نمی‌کنند و صرفا یک عبارت دارای مقدار هستند. به عنوان مثال یک متغیر یا یک دسترسی به یکی از عناصر لیست یک عبارت lvalue است اما عبارت 1 + 5 یک عبارت rvalue محسوب می‌شود. عبارات rvalue تنها در سمت راست عملگر تخصیص قرار می‌گیرند.

۶ـ۵. اولویت عملگرها

اولویت	دسته	عملگرها	شرکت‌پذیری
۱	پرانتز	()	چپ به راست
۲	دسترسی به اعضاء	.	چپ به راست

۳	دسترسی به عناصر لیست	[]	چپ به راست
۴	تک عملوندی پسوندی	++ --	چپ به راست
۵	تک عملوندی پیشوندی	- ! ++ --	راست به چپ
۶	ضرب و تقسیم	* / %	چپ به راست
۷	جمع و تفریق	+ -	چپ به راست
۸	رابطه‌ای	< >	چپ به راست
۹	مقایسه‌ی تساوی	!= ==	چپ به راست
۱۰	عطف منطقی	&&	چپ به راست
۱۱	فصل منطقی	||	چپ به راست
۱۲	تخصیص	=	راست به چپ
۱۳	کاما (ورودی متدها)	,	چپ به راست

۷. ساختار تصمیم‌گیری

در زبان Sophia تنها ساختار تصمیم‌گیری، if..else است. ساختار نحوی[1] آن مشابه زیر است:

[1] syntax

```
if (x[2] > 0) {
    print("positive");
} else if (x[2] < 0) {
    print("negative");
} else {
    print("zero");
}
```

این ساختار بدون else نیز می‌تواند به کار رود.

۸. ساختار تکرار

۸ـ۱. حلقه‌ی for

ساختار نحوی آن به این صورت است که ۳ عبارت به عنوان مقداردهی، شرط و بروزرسانی می‌گیرد که عبارت مقداردهی و بروزرسانی آن یک گزاره است و عبارت شرط هر عبارتی از تایپ bool می‌تواند باشد. هرکدام از عبارات مقداردهی، شرط و بروزرسانی for می‌توانند خالی باشند. دقت شود که تمامی متغیرها در ابتدای متدها تعریف می‌شوند و متغیری را در ۳ عبارت بالا یا در داخل اسکوپ for تعریف نمی‌کنیم.

```
for(initializationStatement;conditionExpression;updateStatement) {
    // statements inside the body of loop
}
```

۸ـ۲. حلقه‌ی foreach

در ساختار نحوی این حلقه از کلیدواژه in استفاده شده‌است. سمت چپ این کلیدواژه یک متغیر که قبلاً تعریف شده و سمت راست آن یک عبارت است. این حلقه برای پیمایش اعضای لیست‌هایی که تمامی اعضای آنها از یک جنس باشد تعبیه شده‌است.

```
foreach (identifier in list) {
    // statements inside the body of loop
}
```

۸ـ۳. کلیدواژه break

در هر کدام از حلقه‌های تعریف‌شده در زبان Sophia می‌توان با استفاده از کلیدواژه‌ی break از حلقه خارج شد و دستورات بعد از حلقه را اجرا کرد. بدیهی است که در صورت استفاده از چند حلقه‌ی تودرتو، دستور break روی درونی‌ترین حلقه عمل می‌کند و اجرای حلقه‌های بیرونی ادامه پیدا می‌کند.

۸ـ۴. کلیدواژه continue

در هر کدام از حلقه‌های تعریف‌شده در زبان Sophia می‌توان با استفاده از کلیدواژه‌ی continue تکرار کنونی حلقه را متوقف و تکرار بعدی را آغاز کرد. بدیهی است که در صورت استفاده از چند حلقه‌ی تودرتو، دستور continue روی درونی‌ترین حلقه عمل می‌کند.

۹. scopeها

۹ـ۱. scope های موجود در زبان

به طور کلی در زبان Sophia موارد زیر در scope جدیدی قرار دارند:

- خطوط کد داخل یک کلاس
- پارامتر ها و خطوط کد داخل یک متد
- داخل گزاره‌های تصمیم‌گیری و تکرار
- داخل scope های جدیدی که با {} مشخص می‌شوند

۹ـ۲. قوانین scope ها

- تعریف کلاس‌ها در بیرونی‌ترین scope است.
- خطوط خالی از کد اجرایی هیچ تاثیری در خروجی و اجرای برنامه ندارد.
- کدهای داخل هر متد در scope آن متد هستند.
- متغیرهایی که داخل یک scope تعریف می‌شوند در scopeهای بیرون آن دسترس‌پذیر نیستند و صرفاً در scopeهای درون آن قابل دسترسی هستند.
- امکان تعریف متغیر با نام یکسان در یک scope وجود ندارد امّا در scopeهای درونی آن امکان تعریف مجدد وجود دارد و تا زمان خروج از scope درونی، نزدیکترین تعریف به آن استفاده می‌شود.

۹ـ۳. قوانین خطوط برنامه

قوانین خطوط این زبان مشابه زبان Java است؛ تمامی دستورات، در انتهای خود یک کاراکتر ; دارند.

۱۰. توابع پیش‌فرض

در زبان Sophia تنها یک تابع پیش‌فرض وجود دارد و آن هم print است. این تابع به صورت ضمنی تعریف شده‌است و می‌تواند یک مقدار int، bool یا string دریافت کند و مقدار آن را در کنسول چاپ کند.

```
print("Hello World!");
```

۱۱. نمونه کد

نمونه‌هایی از کدهای نوشته شده در زبان Sophia در ادامه آمده است.

```
class Person {
  name: string;
  age: int;
  def Person(name: string, age: int) {
    this.name = name;
    this.age = age;
  }
  def void showName() {
    print(this.name);
    print("\n");
  }
}

class Employee extends Person {
  salary: int;
  def Employee(name: string, age: int, salary: int) {
    this.name = name;
    this.age = age;
    this.salary = salary;
  }
  def void showSalary() {
    print("Salary is:");
    print(this.salary);
    print("\n");
  }
}

class Main {
  def Main() {
    e: Employee;
    p: Person;
    e = new Employee("Hannah", 21, 3000);
    e.showName();
    e.showSalary();
    p = new Person("Sophia", 30);
    p.showName();
  }
}
```

```
class Cart {
  orders: list(4 # list(name: string, price: int, quantity: int));
  def void addToCart(product: list(string, int, int), idx: int) {
    this.orders[idx] = product;
  }
  def int getSum() {
    total: int;
    current: int;
    order: list(name: string, price: int, quantity: int);
    total = 0;
    foreach (order in this.orders) {
      if (order.quantity == 0)
        break;
      current = order.price * order.quantity + order.quantity *
                100 % 1000 - order.price / 100;
      total = total + current;
    }
    return total;
  }
}

class Main{
  def Main() {
    cart: Cart;
    i: int;
    total: int;
    order: list(string, int, int);
    product: list(string, int);
    productCatalog: list(4 # list(string, int));
    productCatalog = [["Doughnut", 5000], ["Croissant", 4000],
                      ["Cookies", 2000],
                      ["Chocolate Cake", 8000]];
    cart = new Cart();
    for (i = 0; i < 4; i = i + 1) {
      product = productCatalog[i];
      order = [product[0], product[1], i + 1];
      cart.addToCart(order, i);
    }
    total = cart.getSum();
    print(total);
    print("\n");
  }
}
```