سند زبان برنامه‌نویسی

SimpleLOOP

(Simple Language for Object-Oriented Programming)

فهرست

1. مقدمه

زبان ابداعی SimpleLOOP یک زبان شی‌گراست و برخی از ویژگی‌های زبان شی‌گرا نظیر ارث‌بری را داراست. در این زبان یک کلاس اصلی Main وجود دارد که در آن یک متد initialize پیاده‌سازی می‌شود. برنامه‌هایی که در این زبان نوشته می‌شوند، هنگام اجرا دستورات درون این متد را اجرا می‌کنند.

2. ساختار کلی

در این زبان، کد برنامه درون یک فایل با پسوند sl. قرار دارد. این فایل از قسمت‌های زیر تشکیل شده‌است:

- تعریف متغیرهای سراسری
- تعریف کلاس‌ها؛ هر کلاس شامل تعدادی فیلد و متد
- یک کلاس اصلی Main

یک نمونه کد در این زبان به صورت زیر است:

```
class Student {
   private int id
   private Course[10] courses
   private int course_num

   public initialize(int id) {
       self.course_num = 0
       self.id = id
   }

   public bool coursePassed(int id) {
       Course c
       self.courses.each do |c|
           if c.getId() == id
               if c.getGrade() > 10
                   return true
               else
                   return false
       return false
   }
```

```
  public void addCourse(int id, int grade) {
       Course newCourse
       newCourse = Course.new(id, grade)
       self.courses[self.course_num] = newCourse
       self.course_num ++
   }
}

class Course {
   private int grade
   private int id

   public initialize(int id, int grade) {
       self.id = id
       self.grade = grade
   }

   public int getGrade()
       return self.grade

   public int getId()
       return self.id
}

class Main {
   public initialize() {
       Student s
       s = Student.new(810198433)
       s.addCourse(810110, 15)
       s.addCourse(810114, 9)
       s.addCourse(810134, 14)
       print(s.coursePassed(810114))
   }
}
```

1-2. قواعد کلی نحو

زبان SimpleLOOP به بزرگ و کوچک بودن حروف حساس است. نام کلاس‌ها حتما با حرف بزرگ شروع می‌شود. بقیه نام‌ها باید با حرف کوچک آغاز شوند. در این زبان، وجود کاراکترهای Tab و Space تاثیری در خروجی برنامه ندارد. جزئیات مربوط به Scopeها و خطوط برنامه در ادامه به طور مفصل توضیح داده خواهد شد.

2-2. کامنت‌ها

در این زبان، کامنت‌ها یک یا چند خطی هستند. کامنت‌ها بدین صورت هستند که در یک خط، begin= شروع آن را مشخص می‌کند و در خط(ها)ی پس از آن کامنت قرار می‌گیرد و در خط انتهایی با end= پایان کامنت مشخص می‌شود. کامنت‌های تک خطی را می‌توان با علامت # آغاز کرد و انتهای خط، پایان کامنت را مشخص می‌کند. دقت کنید که امکان تعریف گزاره‌های زبان و کامنت در یک خط وجود ندارد. بنابراین کامنت باید در خط جدید نوشته شود.

```
# Spring 1401

=begin
Welcome to Compiler course
We wish you a good and successful semester
=end
```

2-3. قواعد نام‌گذاری کلاس‌ها، متدها و متغیرها

اسامی انتخابی برای نام‌گذاری کلاس‌ها، متدها و متغیرها باید از قواعد زیر پیروی کنند:

- تنها از کاراکترهای a..z،A..Z ، _ و ارقام تشکیل شده باشند.
- با رقم شروع نشوند.
- نام کلاس‌ها حتما با حرف بزرگ و بقیه نام‌ها با حرف کوچک شروع شوند.
- معادل کلیدواژه‌ها نباشند. در جدول زیر تمام کلیدواژه‌های این زبان آمده‌است:

class	private	public	int	bool
fptr	Set	self	initialize	true
false	return	print	each	do
new	if	else	elsif	void

null	delete	include	add	merge

- نام هر کلاس یکتاست.
- نام هر متد در هر کلاس یکتاست. به عبارت دیگر method overloading در این زبان وجود ندارد.
- نام هر متغیر در یک Scope یکتاست.
- امکان تعریف دو متغیر و متد با نام یکسان در یک کلاس وجود ندارد.

3. کلاس‌ها و متدها

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

```
class Class1 < Class2 {
   # class scope
}
```

متغیرها و متدهای هر کلاس می توانند با هر ترتیبی در scope آن کلاس تعریف شوند. متغیرها و متدها می توانند به صورت public یا private تعریف شوند؛ در صورتی که private باشند، فقط درون کلاس قابل دسترسی هستند. متد initialize در واقع همان constructor برای هر کلاس است. مثال زیر نمونه‌ای از تعریف یک کلاس است:

```
class Point {
   public int x, y
   public initialize(int x, int y) {
       self.x = x
```

```
        self.y = y
    }

    public int findSquaredDistance(Point p)
        return (self.x - p.x) * (self.x - p.x) + (self.y - p.y) * \\
         (self.y - p.y)
}
```

کلاس Main باید حتما یک constructor بدون آرگومان داشته باشد. کلاس Main فرزند هیچ کلاس دیگری نیست. کلاس‌ها می توانند یک یا چند متغیر و متد داشته باشند. در هر متد، تعریف متغیرها در ابتدای آن و قبل از همه‌ی دستورات انجام می‌شود. همچنین در این زبان، همه متدها به جز initialize 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.

در هنگام تعریف، آرگومان‌های متدها می توانند مقدار پیش‌فرض داشته باشند و در هنگام فراخوانی، مقدار دادن به این آرگومان‌ها اختیاری خواهد بود؛ این آرگومان‌ها باید پس از آرگومان‌هایی که مقدار پیش‌فرض ندارند تعریف شوند.

مثالی از نحوه تعریف متدها و فراخوانی آن‌ها در ادامه آمده است.

```
public int sum(int a, int b=0)
    return a + b


# different ways of method call
self.sum(1)
# returns 1
self.sum(1, 2)
# returns 3
```

برای ساختن یک نمونه‌ی جدید از یک کلاس از کلیدواژه new استفاده می کنیم.

```
class Sample {
    private int a, b
    private bool c
```

```
    public initialize(int a, int b, bool c) {
        self.a = a
        self.b = b
        self.c = c
    }
}


# create new instance of class Sample
Sample s
s = Sample.new(0, 1, true)
```

هر گزاره در این زبان در یک خط جداگانه نوشته می‌شود. در صورتی که یک گزاره بخواهد در چند خط ادامه داشته باشد می‌توان از // استفاده کرد.

یک بلاک با علامت } شروع شده و انتهای آن با علامت { مشخص می‌شود. در صورتی که در یک Scope چند گزاره داشته باشیم، باید درون بلاک قرار بگیرند؛ در غیر این صورت تعریف بلاک اختیاری است.

```
# using block in this example is optional
if true
    if true
        print(true)
    else
        print(false)


# using block in this example is necessary for second if
if true
    if true {
        print(true)
        print(1)
    }
    else
        print(false)
```

1-3. ارث‌بری

هر کلاسی می‌تواند از حداکثر یک کلاس دیگر ارث‌بری کند (با استفاده از نماد >) و از فیلدها و متدهای آن استفاده کند. قابلیت overriding برای متدهای یک کلاس وجود ندارد. هم چنین overloading برای متغیرهای کلاس در این زبان وجود ندارد.

2-3. کلیدواژه self

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

4. انواع داده

در زبان SimpleLOOP، سه نوع پایه bool، int و متغیر از نوع کلاس (class object) و سه نوع دیگر، آرایه، اشاره‌گر به تابع و مجموعه مرتب که توضیحات آن‌ها در ادامه آمده‌است، وجود دارند. در متغیرهای از نوع تایپ‌های پایه، خود مقادیر ذخیره می‌شوند و نه اشاره‌گر به خانه‌ای از حافظه. همچنین تایپ void نشانگر مقدار بازگشتی توابع بدون خروجی است. استفاده از این تایپ به عنوان عملوند مجاز نیست. تعریف متغیر از این تایپ غیرمجاز است.

1-4. آرایه

آرایه دارای تعدادی عنصر است که اندازه آن هنگام تعریف مشخص می‌شود. اندازه آرایه یک عدد ثابت بزرگتر از صفر است. در این زبان، اعضای آرایه باید حتما از یک نوع باشند. نوع عناصر آرایه می‌تواند یکی از چهار نوع class object، bool، int یا آرایه باشد. مقدار اولیه اعضای آرایه برابر با مقدار پیش فرض برای آن نوع است؛ امکان تعریف آرایه با مقادیر اولیه دلخواه وجود ندارد.

```
=begin
array of int of size 10
default value for all elements is 0
=end
int[10] arr
# 2D array - rows = 4, cols = 5
int[4][5] arr
```

2-4. اشاره‌گر به تابع

یک متغیر از نوع اشاره‌گر به تابع، به یک تابع (متد) از برنامه اشاره می کند و می توان تابع را به صورت غیرمستقیم توسط آن فراخوانی کرد. به طور مثال یک متغیر از نوع <int -> bool> به تابع با پارامتر ورودی از نوع int و مقدار بازگشتی از نوع bool اشاره می‌کند. برای توصیف تابعی بدون ورودی و خروجی از <void -> void> استفاده میکنیم. نحوه تعریف و استفاده از آن‌ها در زیر آمده است:

```
void printNumber(int number)
   print(number)


# variable fp is a function pointer
fptr <int -> void> fp
fp = printNumber
fp(3)
# printNumber is called and output = 3
```

3-4. مجموعه مرتب[1]

مجموعه مرتب نوعی ظرف[2] است که عناصر در آن به ترتیب صعودی چیده شده‌اند و عضو تکراری ندارد. در زبان SimpleLOOP اعضای مجموعه‌ی مرتب فقط از نوع int می توانند باشند. تعریف یک مجموعه به صورت زیر انجام می‌شود:

```
Set <int> mySet
# create empty set
mySet = Set.new()
# create set with initial values
mySet = Set.new((1, 2))
# mySet = {1, 2}
```

توسط تابع add می توان یک عضو را به مجموعه اضافه کرد؛ برای اضافه کردن همزمان چندین عضو از تابع merge استفاده می شود.

```
Set <int> s;
s = Set.new() # s = {}
s.add(2)
# s = {2}
s.merge(Set.new(1, 3))
# s = {1, 2, 3}
```

دو تابع دیگری که بر روی مجموعه‌ها تعریف می شوند، توابع include و delete هستند. هر دوی این توابع یک عدد را به عنوان ورودی دریافت می‌کنند. خروجی تابع include یک bool است که مشخص می کند عدد ورودی در مجموعه قرار دارد یا خیر. تابع delete عضو مشخص شده را از مجموعه حذف می‌کند.

[1] ordered set
[2] container

```
Set <int> s;
s = Set.new((1, 2, 3));
s.delete(1)
print(s.include(1))
# output = false
```

5. متغیرها

متغیرها می توانند به صورت محلی یا سراسری تعریف شوند. متغیرهای سراسری در ابتدای برنامه و پیش از تعریف کلاس‌ها، تعریف می شوند.

متغیرهای محلی درون متدها فقط در ابتدای آن‌ها تعریف شده و در ابتدای scopeهای درونی‌تر آن‌ها امکان تعریف متغیر جدید وجود ندارد.

در این زبان چندین متغیر را می‌توان در یک گزاره تعریف کرد ولی نمی‌توان به آن‌ها مقدار اولیه داد.

تعریف متغیرها به صورت زیر انجام می‌گیرد:

```
# type identifier
int a
int b, c;
```

در صورتی که به متغیری مقداری نسبت داده نشود، مقدار آن برابر با مقدار پیش فرض نوع خود در نظر گرفته می‌شود. مقادیر پیش فرض نوع‌های مختلف در جدول زیر مشخص شده‌است.

bool	false
int	0
fptr	null

در صورتی که متغیر از نوع یک کلاس یا مجموعه باشد، مقدار اولیه آن null است و در صورتی که متغیر از نوع آرایه باشد، مقدار اولیه هر کدام از اعضا برابر با مقدار پیش فرض نوع خود است.

6. عملگرها

عملگرها در این زبان به پنج دسته‌ی عملگرهای حسابی، مقایسه‌ای، منطقی، عملگر تخصیص و عملگر شرطی سه تایی تقسیم می‌شوند.

6-1. عملگرهای حسابی

این دسته از عملگرها تنها روی اعداد عمل می‌کنند، لیست این عملگرها در جدول زیر آمده است. در مثال‌های استفاده شده A برابر 20 و B را برابر 10 در نظر بگیرید:

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

2-6. عملگرهای مقایسه‌ای

این عملگرها وظیفه‌ی مقایسه را دارند، پس نتیجه‌ی آن‌ها باید مقدار صحیح یا غلط (true و false) باشد. بنابراین خروجی این عملگرها از نوع bool است.

توجه داشته باشید که عملوند عملگرهای < و > تنها از نوع عدد صحیح هستند. همچنین برای عملگر == نیز باید نوع عملوندها یکسان و حتما از انواع primitive باشند؛ در غیر اینصورت باید خطای کامپایل گرفته شود.

لیست عملگرهای مقایسه‌ای در جدول زیر آمده‌است. در مثال‌های استفاده شده مقدار A را برابر 20 و مقدار B را برابر 10 بگیرید:

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

3-6. عملگرهای منطقی

در زبان SimpleLOOP عملگرهای منطقی تنها روی نوع داده‌ی bool قابل استفاده است.این عملگرها در جدول زیر لیست شده‌اند. در مثال‌های استفاده شده 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

4-6. عملگر تخصیص

این عملگر که به صورت = نمایش داده می‌شود، وظیفه‌ی تخصیص را برعهده دارد. یعنی مقدار عملوند سمت راست را به عملوند سمت چپ اختصاص میدهد. این عملگر فقط برای تخصیص انواع primitive میباشد.

دقت داشته باشید که عملوند سمت چپ باید از نوع Lvalue باشد. عبارات Lvalue عباراتی هستند که به یک مکان در حافظه اشاره میکنند. 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.

5-6. عملگر شرطی سه تایی

این عملگر دارای سه عملوند است که تایپ عملوند اول bool است و تایپ دو عملوند بعدی باید یکسان باشد. در صورت true بودن عملوند اول، مقدار عملوند دوم و در غیر این صورت مقدار عملوند سوم برگردانده می‌شود.

```
int result
int op1, op2
bool condition
condition = true
result = condition ? op1 : op2
# result = op1
```

7-5. اولویت عملگرها

اولویت عملگرها طبق جدول زیر است:

اولویت	دسته	عملگرها	شرکت‌پذیری
1	پرانتز	( )	چپ به راست
2	دسترسی به اعضای کلاس	.	چپ به راست
3	دسترسی به عناصر آرایه	[ ]	چپ به راست
5	تک عملوندی پسوندی	-- ++	چپ به راست
6	تک عملوندی پیشوندی	- !	راست به چپ
7	ضرب و تقسیم	* /	چپ به راست
8	جمع و تفریق	+ -	چپ به راست
9	مقایسه‌ای بزرگتر و کوچکتر	< >	چپ به راست
10	تساوی	==	چپ به راست
11	عطف منطقی	&&	چپ به راست
12	فصل منطقی	||	چپ به راست

13	شرطی سه تایی	? :	راست به چپ
14	تخصیص	=	راست به چپ
15	کاما	,	چپ به راست

7. ساختار تصمیم‌گیری

در زبان SimpleLOOP تنها ساختار تصمیم‌گیری، if...elsif...else است. هر if می‌تواند به همراه صفر تا چند elsif و صفر یا یک else استفاده شود.

```
int maximum(int a, int b, int c) {
   int largest
   if (a > b || a == b) && (a > c || a == c)
       largest = a
   elsif (b > a || b == a) && (b > c || b == c)
       largest = b
   else
       largest = c
   return largest
}
```

8. ساختار تکرار

در این زبان از کیلدواژه each برای حلقه استفاده می شود. درواقع each، یک iterator است که تمامی عناصر آرایه را برمی‌گرداند. قبل از این کلیدواژه می توان یک متغیر از نوع آرایه، مجموعه یا یک بازه از اعداد را به صورت (begin..end) تعریف کرد.

```
(1..4).each do |i|
   print(i)

=begin
output =
```

```
1
2
3
=end
```

```
Set <int> s
int var
s = Set.new((4, 7, 8, 3, 1))
s.each do |var| {
   print(var)
   if var > 7
       s.delete(var)
}

=begin
output =
1
3
4
7
8

s = {1, 3, 4, 7}
=end
```

9. Scopeها

9-1. Scopeهای موجود در زبان

به طور کلی در زبان SimpleLOOP موارد زیر در Scope جدیدی قرار دارند:

1. خطوط کل داخل یک کلاس
2. پارامترها و خطوط کد داخل یک متد
3. داخل گزاره‌های تصمیم‌گیری و تکرار
4. داخل scopeهای جدیدی که با {} مشخص شده اند.

2-9. قوانین scopeها

نکات زیر در مورد Scopeها وجود دارد:

- از متغیرهای سراسری در همه scopeها می‌توان استفاده کرد.
- 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 دسترس پذیر نیستند و صرفا در scopeهای درونی آن قابل دسترسی هستند.
- امکان تعریف متغیر با نام یکسان در یک scope وجود ندارد.
- فیلدهای private یک کلاس در scope خارج آن کلاس، قابل دسترسی نیستند.

10. توابع پیشفرض

1-10. تابع print

این تابع یک مقدار int یا bool دریافت میکند و آن را در خروجی چاپ میکند. مثال :

```
print(2)
```

11. نمونه کدهایی در این زبان

1.

```
class Main {
   public initialize() {
       Factorial factorial
       fptr<void -> int> fp
       factorial = Factorial.new(5)
       fp = factorial.calculateFactorial
       print(fp())
```

```
    }
}

class Factorial {
   private int number;
   public initialize(int number)
       self.number = number
   public int claculateFactorial() {
       int fact = 1
       int i
       (1..self.number).each do |i|
           fact = fact * i
       return fact
   }
}
```

2.

```
class Person {
   private int id, age

   public initialize(int id, int age) {
       self.id = id
       self.age = age
   }
   public void showId()
       print(self.id)
}

class Employee < Person {
   private int salary
   private bool privileged
   public initialize(int id, int age, bool privileged, int salary) {
       self.id = id
       self.age = age
       self.privileged = privileged
       self.salary = salary
   }

   public void showSalary() {
       if ! self.privileged
           print(self.salary)
       else
           print(0)
   }
}
```

```
class Main {
   public initialize() {
       Employee e
       e = Employee.new(10, 37, false, 300000)
       e.showSalary()
       e.showId()
   }
}
```