سند زبان برنامه‌نویسی

TOORLA

۱. مقدمه

زبان Toorla (مخفف Tiny Object Oriented Readable Language) یک زبان شی‌گرا مشابه زبان Java است و برخی از ویژگی‌های زبان‌های شی‌گرا نظیر ارث‌بری را داراست.

در این زبان، کد برنامه درون یک فایل با پسوند **trl.** قرار دارد. این فایل شامل یک یا چند کلاس است و هر کلاس شامل تعدادی فیلد و متد است.

۲. ساختار کلی

یک برنامه به زبان Toorla از قسمت های زیر تشکیل شده است:

- تعریف کلاس :
  - تعریف فیلد
  - تعریف متد

یک نمونه از کد در این زبان به صورت زیر است (منظور از entry class کلاسی است که باید حتما یک متد main داشته باشد که public است و اجرای یک برنامه در این زبان از متد main این کلاس آغاز میشود ، در ادامه راجع به این کلاس بیشتر توضیح داده خواهد شد.)

```
entry class MainClass:
    private field a int;
    function main() returns int:
        a = 1;
        while( a > 2 ) begin
            print( a + 1 );
            a = a + 1;
        end
        return 0;
    end
end
```

۱-۲. قواعد کلی نحو

زبان Toorla به بزرگ و کوچک بودن حروف حساس است. در این زبان، وجود کاراکترهای Space ، Tab و Newline ، تاثیری در خروجی برنامه ندارند. جزئیات مربوط به Scope ها و خطوط برنامه در ادامه بیشتر توضیح داده خواهند شد.

۲-۲. کامنت‌ها

در زبان Toorla دو نوع کامنت می توان داشت :

- کامنت تک خطی :

تمامی حروف بعد از // تا انتهای خط جزو کامنت تک خطی می باشند.

- کامنت چند خطی :

تمامی حروف بعد از */ تا زمانی که /* دیده شود جزو کامنت چند خطی می باشند .

یک نمونه از استفاده از کامنت ها در تکه کد زیر آمده است :

```
entry class MyClass:
    function main() returns int:
        //single-line comment
        print( "viva Toorla" );
        /*
            multi-line comment
            multi-line comment
        */
        return 0;
    end
end
```

لازم به ذکر است که در زبان Toorla کامنت های چند خطی ساختار یافته و تو در تو نداریم . به مثال زیر دقت کنید :

```
/* open 1
    /*

    close2 which is end of multiline comment*/

*/ // close1 which is not a close for first open right now
```

۳-۲. 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، ـ و ارقام تشکیل شده باشند.
- با رقم شروع نشوند.
- معادل کلیدواژه‌ها نباشند. در جدول زیر تمامی کلیدواژه‌های زبان Toorla آمده‌است:

bool	string	int	class	function
if	print	private	field	self
false	true	while	else	new
return	elif	returns	break	continue
entry	begin	end	public	var
inherits				

۳. تعریف کلاس

هر کلاسی می‌تواند از حداکثر یک کلاس دیگر ارث‌بری کند (با استفاده از کلید واژه ی inherits) و از فیلدها و متدهای پدرش استفاده کند. این متدها و فیلد ها می توانند public و یا private[1] باشند.کلاس‌ها می‌توانند یک یا چند فیلد و متد داشته باشند. نام هر کلاس نیز در یک برنامه یکتاست. یک مثال از تعریف یک کلاس در تکه کد زیر آمده است :

```
class MyClass inherits MySecondClass:
    public field publicVar int;
    private field privateVar int;
    public function publicMethod( a: int ) returns int:
        print( "this method is public" );
        return 0;
    end
    private function privateMethod( b: int ) returns int:
        print( "this method is private" );
        return 0;
    end
end

entry class MySecondClass:
    public function main() returns int:
        print( "this method is main which is public" );
        return 0;
    end
end
```

۳-۱. گونه[2] پیش فرض Any

همه ی کلاس های داخل یک برنامه از زبان Toorla زیرگونه ی این گونه می باشند .

۳-۲. کلاس entry

این کلاس مانند تمام کلاس های دیگر است ، با این تفاوت که این کلاس یا یکی از پدر هایش باید حداقل یک متد main داشته باشد که public است و مقدار بازگشتی آن از گونه ی int باشد و هیچ پارامتر ورودی نیز نباید داشته باشد. اجرای یک برنامه در زبان Toorla از متد main این کلاس آغاز می شود . توجه داشته باشید که در این زبان فقط یک کلاس entry باید در برنامه موجود باشد ، در غیر این صورت باید خطای زمان کامپایل به کاربر داده شود. یک مثال از کلاس entry به مانند صفحه ی بعد است:

---

[1] در این زبان private به مانند protected در C++ می باشد ، یعنی همان کلاس و زیر کلاس های یک کلاس به اینگونه متد ها و فیلد ها دسترسی دارند و سایر کلاس ها نمیتوانند به آنها دسترسی پیدا کنند .

[2] Type

```
class YourClass:
    private field myVar1 int;
end

entry class MyClass inherits YourClass :
    public field myVar int;

    public function main() returns int:
        print( "hello my beautiful language" );
        return 0;
    end

    private function myMethod() returns int:
        print( "hello by beautiful language" );
        return 0;
    end
end
```

۴. تعریف متد

همانطور که گفته شد ، یک کلاس میتواند چند متد داشته باشد که این متد ها می توانند public و یا private باشند. متد های کلاس در زبان Toorla باید حتما مقدار بازگشتی داشته باشند و گونه ی بازگشتی آن نیز باید ذکر شود ، ضمن آنکه نام هر متد در یک کلاس یکتاست (یعنی نمیتوان در یک کلاس دو متد با یک نام به صورت صریح داشت). تعریف یک متد به صورت زیر انجام می‌شود (لازم به ذکر است که اگر access modifier برای یک متد ذکر نشود ، آن متد به صورت پیش فرض public است):

```
access_modifier function method( a: type , b: type, c: type , d: type ) returns type:
    // body
end
```

۵. فراخوانی متد

یک مثال از فراخوانی متد ها در تکه کد زیر آمده است :

```
entry class MyMain:
    function main() returns int:
        return new MyMain().func( 1 , new MyMain() , "hello" );
    end
    function func( first: int , second: MyMain , third: string ) returns int:
        print( "hello" );
        return 0;
    end
end
```

در زبان Toorla در همه ی کلاس ها می توان به متدهای public شی های از کلاس های دیگر را می توان فراخوانی کرد ، اما متد های private یک شی را فقط در متد های همان شی می توان فراخوانی کرد. به مثال زیر دقت کنید:

```
entry class MainClass:
    public function main() returns int:
        print( "main" );
        print( new MainClass().main() ); //this is legal , calling public method of an object
        print( new MainClass2().privateMethod() ); // this is illegal, you cannot access private method of any object in this lang
        print( new MainClass().privateMethod() ); // this is illegal, you cannot access private method of any object in this lang
        return 0;
    end
    private function privateMethod() returns int:
        print( "private method" );
        return 0;
    end
end

class MainClass2:
    private function privateMethod() returns int:
        print( "private method" );
        return 0;
    end
end
```

توجه داشته باشید که فراخوانی متد ها یک Expression است (تفاوت Statement با Expression در آن است که Statement یک خط قابل اجرای برنامه است ولی Expression یک خط قابل اجرا در برنامه ی سطح بالا نیست با اینکه مقدار بازگشتی دارد) ، در فراخوانی متد ها ، ارسال پارامتر ها by value انجام می شود ، یعنی این که به جای آدرس یک پارامتر ، مقدار آن برای متد ارسال می شود.

۶. کلیدواژه self

کلیدواژه self به کلاسی که در آن قرار داریم اشاره می‌کند که به وسیله ی آن میتوان به متد یا فیلد یک کلاس دسترسی پیدا کرد ، ضمن آنکه در بدنه یک متد از یک کلاس می توان یک متد یا فیلد از همان کلاس را بدون آوردن کلید واژه ی self فراخوانی کرد . به مثال زیر دقت کنید:

```
print(self.sayHello()); // ok
print(sayHello()); /* this print and the print in previous line have the same
functionality */
print(self.myVar);
print(myVar); /* this line and previous line have same functionality
, assume that myVar is not defined in local scopes */
```

۷. انواع داده

در زبان Toorla ، سه گونه پایه string ، bool و int وجود دارند. علاوه بر آن‌ها، هر متغیر می‌تواند از جنس یکی از کلاس‌هایی باشد که در برنامه تعریف شده‌است.در این زبان، یک نوع آرایه نیز تعریف شده است. این آرایه، یک بعدی و میتواند از هر گونه ای باشد. گونه آن به صورت type[ ] می‌باشد. به طور خلاصه گونه های موجود در این زبان در جدول زیر آمده است:

int
string
bool
type[ ]
Class

۸. متغیرها

در زبان Toorla میتوان دو نوع متغیر داشت :

۸-۱. متغیر های محلی :

تعریف متغیرهای محلی (شامل متغیر های داخل متد ها و بلاک ها و پارامتر های متد ها) به صورت زیر انجام میشود (غیر از پارامتر های متد ها که نحوه ی تعریف آنها در بخش ۴ توضیح داده شده است) :

```
var variable1 = expr , variable2 = expr2 , ... ;
```

گونه ی متغیر محلی که تعریف می شود ، گونه ی مقدار اولیه ی همان متغیر است ، توجه داشته باشید یک متغیر محلی حتما باید در هنگام تعریف مقدار دهی شود ، در غیر این صورت خطای زمان کامپایل خواهیم داشت . متغیر های محلی در هر کجای یک متد یا یک بلاک میتوانند تعریف شوند ، ضمن آنکه همان طور که مثال بالا مشاهده می شود ، چند متغیر محلی می توانند با هم در یک خط تعریف شوند.

۸-۲. متغیر های اشیا (فیلد ها) :

فیلد هایی که در کلاس ها تعریف می‌شوند نوع دیگری از متغیر ها هستند که هر شی از یک کلاس یک کپی از آن برای خود دارد .
تعریف فیلد ها در زبان Toorla به صورت زیر انجام می شود (توجه داشته باشید که میتوان تعریف چند فیلد با گونه و access modifier یکسان را در یک خط انجام داد ، ضمن آنکه اگر access modifier در این تعریف ذکر نشود ، آن فیلدها به صورت پیش فرض private محسوب می شوند) :

```
access_modifier field field1 , field2 , ... type;
```

قوانین دسترسی به فیلد های یک شی از یک کلاس نیز همانند فراخوانی متدها می باشد . ضمن آنکه در زبان Toorla نمی توان فیلد ها را در زمان تعریف مقدار دهی اولیه نمود و اینکه می توان تعریف فیلد ها را در هر خط از یک کلاس انجام داد. نحوه ی دسترسی به یک فیلد از یک شی از یک کلاس همانند زبان C++ می باشد .

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

گونه	مقدار پیش فرض
int	0
bool	false
string	""

در صورتی که متغیر از جنس یک کلاس یا آرایه باشد، مقدار اولیه ندارد و در صورت استفاده، باید خطای مناسب به کاربر داده‌شود.

در این زبان، گونه های string ، int و bool از نوع primitive هستند (خود مقادیر در آن‌ها ذخیره می‌شوند نه آدرس به خانه‌ای از حافظه) و گونه های دیگر، از نوع non-primitive هستند و در آن‌ها آدرس به خانه‌ای از حافظه وجود دارد.
پس از تعریف متغیری از جنس آرایه و یا کلاس، باید با استفاده از کلیدواژه new باید آن را instantiate کرد که یک نمونه از آن در تکه کد زیر آمده است :

```
variable = new int[10];
variable = new ClassName();
```

لازم به ذکر است که اندازه‌ی یک آرایه نمی‌تواند صفر یا عددی منفی باشد. تعریف متغیرها میتواند در هر کجای یک Scope (غیر از بیرونی ترین Scope که Scope کلاس هاست) انجام شود .

۹. عملگرها

عملگرها در زبان Toorla به چهار دسته‌ی عملگرهای حسابی، مقایسه‌ای، منطقی، عملگر تخصیص تقسیم می‌شوند.

۹-۱.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 تک عملوندی	-A=-20
%	چپ	باقیمانده	A % B = 0

۹-۲. عملگرهای مقایسه‌ای

این عملگرها وظیفه‌ی مقایسه را دارند، پس نتیجه‌ی آن‌ها باید مقدار صحیح یا غلط (true, false) باشد. با این حساب خروجی این عملگرها یک Boolean است. توجه داشته باشید که عملوند عملگرهای < و > تنها از جنس عدد صحیح هستند. همچنین برای عملگرهای == و <> نیز باید گونه ی عملوندها یکسان باشند و در صورت آرایه بودن، اندازه‌ی آن‌ها نیز برابر باشد؛ در غیر اینصورت باید خطای زمان اجرا گرفته شود.

لیست عملگرهای مقایسه‌ای در جدول صفحه ی بعد آمده است. 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	عدم تساوی	(A < > B) = true
<	چپ	کوچکتر	(A < B) = false
>	چپ	بزرگتر	(A > B) = true

۳-۹. عملگرهای منطقی

در زبان Toorla عملیات منطقی تنها روی نوع داده‌ی Boolean قابل اعمال 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

توجه داشته باشید که محاسبه ی عبارات منطقی در زبان Toorla به صورت short-circuit انجام می شود ، به این مفهوم که اگر یک جزء از عبارت نتیجه ی کل عبارت را معلوم کند ، دیگر باقیمانده ی عبارت محاسبه نمی شود ، مثلا در مثال بالا در A || B ، عبارت A به تنهایی نتیجه ی عبارت در برگیرنده اش را مشخص میکند و دیگر نیازی به محاسبه ی عبارت B نیست .

۹-۴. عملگر تخصیص

این عملگر که به صورت = نمایش داده می‌شود وظیفه‌ی تخصیص را بر عهده دارد. عملگر تخصیص مقدار عملوند سمت راست را به عملوند سمت چپ اختصاص می‌دهد (برای آرایه‌ها مقدار تک‌تک عناصر عملوند سمت راست به عناصر متناظر سمت چپ تخصیص می‌یابد). مقدار خروجی این عملگر برابر با مقدار تخصیص داده شده به عملوند سمت چپ آن است. توجه داشته باشید که تخصیص فقط میتواند Statement باشد.

لازم به ذکر است که عملوند سمت چپ باید حتماً از نوع lvalue باشد. مفهوم lvalue و rvalue در زبان Toorla مشابه زبان Java 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 صرفاً یک عبارت دارای مقدار هستند. به عنوان مثال یک متغیر یک عبارت lvalue است اما عبارت 10+30 یک عبارت rvalue محسوب می‌شود. در زبان Toorla عبارات rvalue تنها می‌توانند سمت راست عملگر تخصیص قرار بگیرند.

۹-۵. عملگر های افزایش و کاهش

این عملگر ها روی مقادیر lvalue از گونه عدد صحیح عمل می کنند ، وظیفه ی این دو عملگر ، افزایش یا کاهش عملوند شان به اندازه ی ۱ واحد می باشد ، اما بعد از اجرای تمام این عملگر ها مقدار عملوند داده شده ، کاهش یا افزایش یافته است . توجه داشته باشید که این عبارات فقط می توانند Statement باشند ، یعنی اینکه میتوانند در یک خط جدای برنامه ظاهر شوند و قابل اجرا هستند ولی مقدار بازگشتی ندارند . این عملگر ها در جدول زیر آمده اند ، فرض کنید A یک lvalue Expression است که مقدار اولیه اش ۴ است :

توضیح	مثال
افزایش پسوندی	A++ (A becomes 5 after this operation)
کاهش پسوندی	A-- (A becomes 3 after this operation)

۶-۹. اولویت عملگرها

اولویت عملگرها طبق جدول زیر است (اولویت ها از بالا به پایین کاهش می یابد) :

اولویت	دسته	عملگرها	شرکت‌پذیری
۱	پرانتز	()	چپ
۲	دسترسی به عناصر آرایه دسترسی به فیلد فراخوانی متد یک شی	. []	چپ
۳	تک عملوندی افزایش یا کاهش پیشوندی	- !	راست
۴	ضرب ، تقسیم و باقیمانده	/ * %	چپ
۵	جمع و تفریق	+-	چپ
۶	رابطه‌ای	< >	چپ
۷	مقایسه‌ی تساوی	== < >	چپ
۸	عطف منطقی	&&	چپ
۹	فصل منطقی	||	چپ
۱۰	تخصیص	=	راست
۱۱	کاما(ورودی متدها)	,	چپ به راست

۱۰. ساختار تصمیم‌گیری

در زبان Toorla تنها ساختار تصمیم‌گیری، ساختار شرطی است که شامل یک if ، چند elif بعد از آن و در نهایت یک else است . یک نمونه از استفاده آن به صورت زیر است :

```
if(a == 2)
    b = 2;
elif( a > 3 )
    b = 3;
elif( a == 3 )
    b = 6;
else
    b = 4;
```

همچنین ساختار if می‌تواند بدون else و یا elif استفاده گردد.

۱۱. ساختار تکرار

تنها ساختار تکرار در این زبان while می‌باشد، که یک expression با گونه Bool را می‌گیرد و تا زمانی که مقدار آن برابر true باشد، حلقه را تکرار می‌کند.

۱۱-۱. دستور break

این دستور فقط در ساختار حلقه قابل استفاده است . با رسیدن اجرای برنامه به این دستور ، برنامه از ساختار while خارج میشود و به اجرای Statement بعدی در برنامه می پردازد.

۱۱-۲. دستور continue

این دستور نیز همانند دستور break فقط در ساختار تکرار قابل استفاده است . با رسیدن اجرای برنامه به این دستور ، کنترل برنامه به اجرای دوباره حلقه تغییر میکند .

مثال صفحه ی بعد نحوه‌ی استفاده از این ساختار را نشان می‌دهد:

```
while(a <> 0) begin
       a = a - 1;
      if( a > 2 ) begin
          a = a + 2;
          break;
      end
      elif( a == 2 )
          continue;
      else
          a = a + 1;
end
```

۱۲. بلاک

بلاک دنباله ای از Statement ها می باشد که ابتدا و انتهای آن با کلمات کلیدی begin و end مشخص می شود ، یعنی به صورت زیر :

```
begin
   // statements
end
```

۱۳. قوانین Scope ها

۱۳-۱. Scope های موجود در زبان

به طور کلی در زبان Toorla موارد زیر در اسکوپ جدیدی قرار دارند:

۱- خطوط کد داخل یک کلاس.

۲- پارامترها و خطوط کد داخل یک متد.

۳- خطوط داخل یک بلاک.

۴- خطوط داخل حلقه و ساختار شرطی.

۱۳-۲. قوانین Scope ها

نکات زیر در مورد Scope ها وجود دارد:

- تعریف کلاس‌ها در بیرونی‌ترین Scope است.
- نام هر متغیر در یک Scope یکتاست اما می‌توان در Scopeهای درونی‌تر از نام متغیرهای بیرونی استفاده کرد و یا متغیری با همان نام در Scope های درونی تعریف کرد . یک Scope ، نزدیک ترین تعریف متغیر ها به خودش را می بیند یعنی به طور مثال در کد زیر ، a در خط مشخص شده با کامنت یک متغیر از گونه bool می باشد :

```
var a = 2;
begin
   var a = true;
   var b = a; // for this scope , a is of type bool
end
```

- متغیرهایی که داخل یک Scope تعریف می‌شوند در Scopeهای بیرون آن دسترس‌پذیر نیستند و صرفاً در Scope های درون آن قابل دسترسی هستند ، اما اگر در یک Scope به متغیری دسترسی پیدا میکنیم که محلی است حتما باید قبل از استفاده ، آن متغیر تعریف شده باشد، در غیر این صورت ، خطا خواهیم داشت . برای مثال در کد زیر یک خطا داریم:

```
entry class MyClass:
    function main() returns int:
        begin
            c = b + 1; // this line has illegal use of b , because b is defined after entering this scope
        end
        var b = 2;
        return 0;
    end
end
```

- فیلد های یک کلاس در کلاس های فرزند ، نمی توانند دوباره به عنوان فیلد کلاس فرزند تعریف شوند ، یعنی در کد زیر خطا خواهیم داشت :

۱۳-۳. قوانین خطوط برنامه

قوانین خطوط زبان Toorla مشابه زبان Java می‌باشد. تنها نکته قابل‌توجه این است که تمامی دستورات، در انتهای خود یک کاراکتر ; دارند.

۱۴. توابع و فیلدهای پیش‌فرض

در زبان Toorla تنها یک تابع پیش‌فرض وجود دارد و آن، تابع print است.

۱۴-۱. تابع print

این تابع به صورت ضمنی تعریف شده است و می‌تواند یک آرایه از int (با هر طولی) و یا یک مقدار int یا string دریافت کند و آن را در کنسول چاپ کند. نمونه‌ای از این دستور به صورت زیر است (دقت داشته باشید که این دستور بعد از چاپ ورودی داده شده یک newline نیز چاپ میکند):

```
print("viva Toorla!");
```

۱۴-۲. فیلد length

این فیلد تنها برای آرایه‌ها تعریف می‌شود و طول یک آرایه را بازمی‌گرداند. به عنوان مثال:

```
arr = new int[666];
print(arr.length); // the output is 666
```

۱۵. دستور return

از این دستور برای بازگشت از متد استفاده می شود که میتواند در هر کجای متد استفاده شود و باید همراه با یک عبارت بازگشتی باشد و باید حداقل یک دستور return با عبارت بازگشتی در بدنه ی غیر شرطی یا شرطی (در صورتی که حتما else داشته باشیم و تمام حالت های مختلف دستور شرطی حداقل دارای یک return باشد) یا غیر حلقه ای متد داشته باشیم . نحوه ی استفاده از این دستور به شکل زیر می باشد :

```
return expression; //return statement
```

۱۶. پیوست[3]

۱۶.۱. Overriding و Overloading

در زبان Toorla ، قابلیت overriding برای متدهای یک کلاس وجود دارد اما overloading وجود ندارد ، overriding به معنای آن است که یک کلاس یکی از متد های پدرش را با همان پارامتر ها ، همان گونه بازگشتی (یا زیرگونه) و همان access modifier (یا access modifier ای با دسترسی بیشتر) دوباره تعریف کند ولی کارکرد آن متد را عوض کند ، در این حالت پیاده سازی متد فرزند با متد پدر جایگزین می شود ، اما در overloading تنها اسم متد کلاس پدر و فرزند یکسان است ولی پارامتر های کلاس فرزند نسبت به پدر باید متفاوت باشد (شرط لازم برای overloading) و گونه ی بازگشتی و یا access modifier کلاس فرزند نسبت به کلاس پدر میتواند متفاوت باشد و یا نباشد (شروط اختیاری) ، در این حالت کلاس فرزند ، متد پدر را نیز ارث بری میکند (در واقع در این حالت دو متد مختلف خواهیم داشت) ، یک مثال از overloading و overriding در تکه کد زیر آمده است :

```
class MyClass inherits Main:
    public function firstMethod( c: int ) returns int: // this method overrides firstMethod of Main
        print( "a" );
        return 0;
    end
end


class MySecondClass inherits MyClass:
    private function firstMethod( a: int , b: string ) returns int :// this method overloads firstMethod of MyClass
        print( "b" );
        return 0;
    end
end

entry class Main:
    private function firstMethod( a: int ) returns int:
        print( "b" );
        return 0;
    end
    function main() returns int:
        print( "a" );
        return 0;
    end
end
```

[3] تمامی موارد مطرح شده در بخش پیوست جزو مواردی هستند که نمره ی اضافه خواهند داشت ، اگر علاقه مند به پیاده سازی این موارد بودید باید به صورت جداگانه این موارد را پیاده سازی کرده و تحویل دهید ، ولی در همه ی فاز های پیاده سازی این زبان که شما انجام میدهید ، فرض بر آن است که موارد گفته شده را پیاده سازی نکرده اید .