فاز اول

مقدمه

هدف از این پروژه طراحی کامپایلر زبان مینی جاوا می باشد . طراحی این کامپایلر به صورت فاز به فاز پیش خواهد رفت بنابراین فاز های بعدی ادامه همین قسمت خواهند بود . سند زبان مینی جاوا اینجا در اختیار شما قرار گرفته است. در این فاز از شما انتظار می رود پس از مطالعه سند این زبان و آشنایی با قواعد آن، برای یک ورودی که قطعه کدی به زبان مینی جاوا است خروجی مورد نظر که توضیحات آن در ادامه است را تولید نمایید. فاز یکم پروژه صرفا جهت آشنایی شما با قواعد زبان Minijava و ابزار ANTLR فراگیری چگونگی خروجی گرفتن از توابع طراحی شده است و بسیار ساده می باشد.

توضیحات

ابتدا گرامر و فایل های مورد نیاز پروژه را از همین ریپازیتوری clone کنید سپس با توجه به این ویدئو به راه اندازی اولیه پروژه بپردازید. در این ویدئو چگونگی عملکرد گرامر ها و طرز کار با listener ها نیز توضیح داده شده است. (توجه فرمایید ویدئو مربوط به پروژه ترم های گذشته می باشد)

با توجه به ویدئو شما باید پس از ایمپورت کردن یک قطعه کد مینی جاوا، با استفاده از Listener ها یک خروجی تولید نمایید. این خروجی نمایگر اجزای مختلف قطعه کد ورودی و جزئیات آن است .

شکل کلی خروجی مورد نظر به صورت زیر است. مواردی که داخل [ ] قرار ندارند نشان دهنده اجزای مختلف یک برنامه در حالت کلی می باشد (کلاس، اینترفیس، متغیر و ...) و باید عینا در خروجی نوشته شوند. موارد داخل [ ] وابسته به قطعه کد ورودی می باشد و در واقع توضیحی برای هر جزء هستند(نام کلاس ها، نام اینترفیس ها، نام متغیرها، نوع متغیر ها و.....) که باید توسط شما با توجه به قطعه کد ورودی تکمیل شوند. کد های خروجی شما تست خواهند شد بنابراین حتما مطابق فرمت داده شده خروجی را تعیین کنید، در غیر این صورت بخش زیادی از نمره را از دست خواهید داد.

```
program start{
    [program body]
}

import class: [class name]

main class: [class name]{
}

class: [class name]/ class parents: ([parent name], )*{
    [class body]
}

interface: [interface name]{
    [interface body]
}

interface method: [method name]/ return type=[return type]/ access modifier=
[access modifier]
(parameters list= [  ([[parameter type] [parameter name]], )+])?

field: [field name]/ type=[type]/ access modifier=[access modifier]

local var: [var name]/ type=[type]
```

```
class method: [method name]/ return type=[return type]/ access modifier=[access
modifier]{
    (parameters list= [  ([[parameter type] [parameter name]], )+])?
    [method body]
}

nested statement{
}
```

در ادامه یک نمونه ورودی و خروجی برای درک بهتر آورده شده است.

Input:

```
import Nothing;
class Classes {
    public static void main(String[] a) {
        Base b;
        Derived d;
        b = new Base();
        d = new Derived();
        b = d;
        System.out.println(b.set(1));
        System.out.println(b.set(3));
    }
}

class Base implements Face{
    int data;
    public int set(int x) {
        data = x;
        return data;
    }
    public int get() {
        return data;
    }
    private int test(){
        int a ;
        int b ;
        {
            int t;
            a = 0;
            b = 5;
        }
        return a + b;
    }

}

class Derived extends Base {
```

```
    public int set(int x) {
        data = x * 2;
        return data;
    }
}

interface Face{
    final int[] a = {1,2};
    int getFace();
}
```

Output:

```
program start{
    import class: Nothing
    main class: Classes{
        local var: b/ type=Base
        local var: d/ type=Derived
    }
    class: Base/ class parents: Face, {
        field: data/ type=int/ access modifier=public
        class method: set/ return type=int/ access modifier=public{
            parameters list= [int x, ]
        }
        class method: get/ return type=int/ access modifier=public{
        }
        class method: test/ return type=int/ access modifier=private{
            local var: a/ type=int
            local var: b/ type=int
            nested statement{
                local var: t/ type=int
            }
        }
    }
    class: Derived/ class parents: Base, {
        class method: set/ return type=int/ access modifier=public{
            parameters list= [int x, ]
        }
    }
    interface: Face{
        field: a/ type=int[]/ access modifier=public
        interface method: getFace/ return type=int/ access modifier=public
    }
}
```

توجه داشته باشید از شما خواسته شده است همانند مثال بالا دندانه گذاری (Indentation) بلاک های کد را در خروجی برآورده سازید. به این معنی که خطوط خروجی می بایستد با توجه جایگاهشان در ساختار کد با فاصله مناسب از ابتدای خط چاپ شوند. هر indent level چهار عدد space می باشد.

موفق باشید