گزارش پروژهٔ پایانی کامپایلر

پوریا عظیمی

سلیم ملکوتی

ما در این پروژه از سازندهٔ پارسر jison[1] استفاده کردیم. این سازندهٔ پارسر، توانایی ساخت پارسرهای LL(1)، LR(0)، LR(1)، SLR(1) و LALR(1) را دارد.

syntax گرامر ورودی این سازندهٔ پارسر، تا حد بسیار زیادی مشابه با Bison است، اما پارسر به‌طور کامل در جاوااسکریپت ساخته شده است و action ـهای قواعد هم کد جاوااسکریپت هستند. مزیت مهم jison آن است که خروجی (پارسر) هم به‌زبان جاوااسکریپت خواهد بود. بنابراین هم می‌توان آن را تحت خط فرمان اجرا کرد (به‌صورت ماژولی در Node.js) و هم می‌توان آن را در یک صفحهٔ وب قرار داد تا متن ورودی را به‌صورت آنلاین یا آفلاین در براوزر parse کند.

زبان CoffeeScript[2] که دیالکتی از جاواسکریپت است و در حال حاضر یکی از پراستفاده‌ترین زبان‌های اسکریپتی در وب (هم در براوزر و هم بعنوان زبان مورد استفاده در نوشتن برنامه برای Node.js[3]) به‌شمار می‌رود هم در ابتدا با کمک پارسری که توسط jison ساخته شده بود اجرا می‌شد[4].

---

۱. http://zaach.github.com/jison/

۲. http://coffeescript.org

۳. http://nodejs.org

۴. http://javascriptjabber.com/017-jsj-coffeescript-with-jeremy-ashkenas/

روند انجام‌کار:

روند کار به‌این صورت است که در ابتدا دو نوع کلاس (Node و Leaf) تعریف کرده‌ایم (در فایل node.coffee).

هر Node شامل یک rule (قانونی که این ند از آن استخراج شده)، متغیری برای نگهداری فرزندان (که Node یا Leaf) هستند، متغیری برای نگهداری scope (متغیرها، توابع و کلاس‌های تعریف‌شده در scope این Node)، متغیری برای نگهداری type این نود (در مواردی که نود می‌تواند تایپ داشته باشد)، اشاره‌گری به نود والد و تعدادی متغیر و تابع کمکی دیگر می‌باشد.

Leaf (که برای نگهداری ثوابت عددی، رشته‌ها، علائم گرامر مثل پرانتز و ; و { و ... به‌کار می‌رود و از کلاس Node مشتق شده) شامل یک value نیز می‌باشد.

در ابتدای پردازش، یک Node با نام RootNode می‌سازیم. هنگامی که قاعده‌ای پذیرفته شد، به‌تعداد مناسب Node و Leaf می‌سازیم، صفات آن‌ها را مشخص می‌کنیم و آن‌ها را بعنوان فرزندان آن قاعده به درخت اضافه می‌کنیم. همچنین، هر قاعده توابع منحصربه‌فردِ validate و resolveType هم دارد که هر قاعده موظف است آن را خود پیاده‌سازی کند. مثلاً validate در مورد قاعدهٔ + بررسی می‌کند که دو طرف علامت جمع باید حتماً از نوع int باشند. ممکن است هنوز نوع expression ـی که در اطراف علامت + است مشخص نشده باشد؛ در این مواقع، تابع resolveType آن فرزند را صدا می‌زنیم تا به‌صورت recursive با توجه به ساختار این expression و نوع فرزندان آن، نوع نهایی و قطعی عبارت را پیدا کند.

در مثال جمع، ما دو Node و یک Leaf می‌سازیم. ممکن است هرکدام از این Node ـها در حقیقت یک Leaf باشند (مثلاً ۲ در ۲+۳ یک Leaf است)،یا می‌توانند یک expression بسیار طولانی‌تر (مثلاً this.func(1,2,3).length) باشند.

اگر فایل ورودی مطابق گرامر نباشد (مثلاً تابعی return نداشته باشد)، پارسر به‌طور خودکار پیغامی مشابه پیغام زیر می‌دهد:

```
...public int func() {    }    public int Co
----------------------^
Expecting '{', 'IF', 'WHILE', 'SYSOUT', 'ID', 'INT', 'BOOLEAN', 'RETURN', got
'}'
```

در این پیغام کاراکتر (یا زیرعبارت) مورد انتظار، و عبارتی که دریافت شده نوشته شده‌اند.

اگر فایل ورودی خطای نحوی نداشته باشد، از این مرحله بدون خطا عبور می‌کند و rootNode بدون خطا ساخته می‌شود. حال می‌توانیم تابع validate را روی این نود صدا بزنیم تا به‌صورت بازگشتی این قاعده و تمام قواعد فرزندش را از نظر نحوی بررسی کند. هر نود می‌تواند تابع validate مخصوص‌به‌خود داشته باشد و logic مورد نظر خود را در آن پیاده‌سازی کند. مثلاً در موردی که بعنوان ایندکس یک آرایه عبارتی غیر از int بدهیم خطای زیر تولید می‌شود:

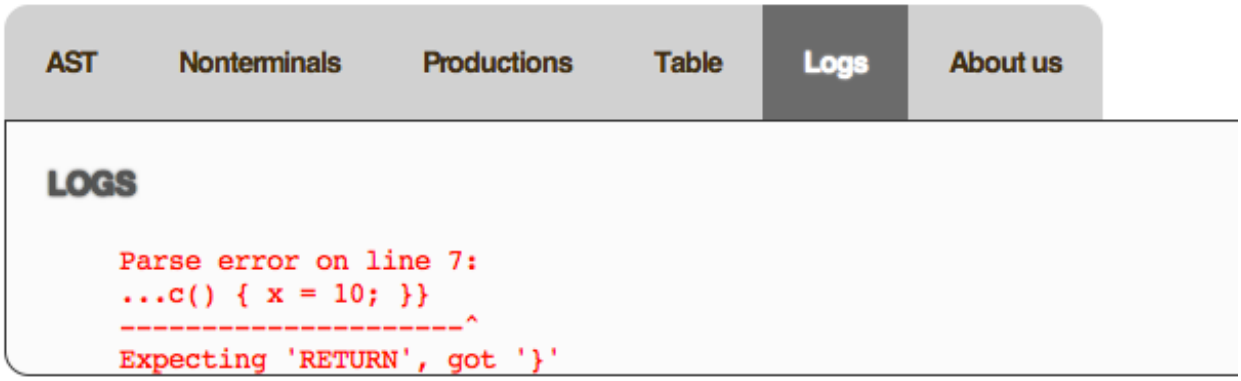
```
x [ true ] ...
    ^~-- type should be 'int'
```

رابط کاربری و نحوهٔ استفاده:

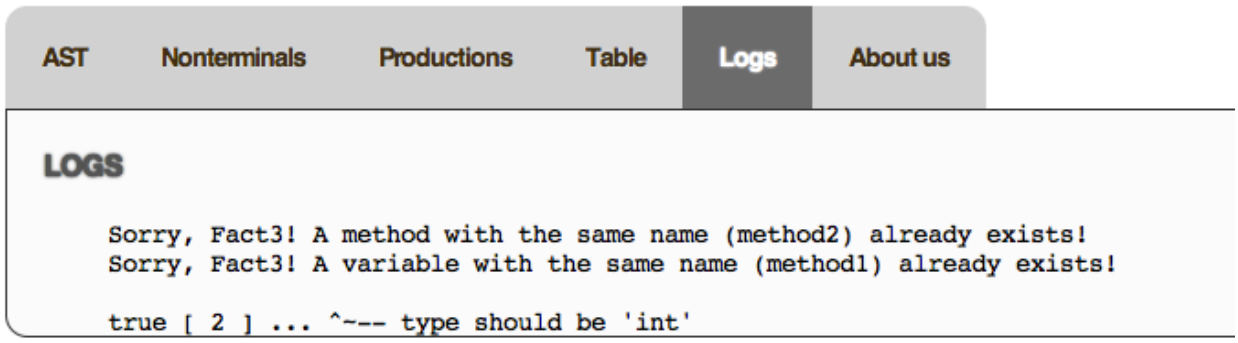
به‌دلایلی قسمتهایی از این پروژه در براوزرهای مختلف درست کار نمی‌کند. ما با کروم ۲۰ و سافاری ۶ قسمت‌های مختلفش را تست کرده‌ایم. در فایرفاکس هم نسبتاً بدون مشکل است، فقط tooltip ـی که تایپ، نام، اسکوپ و ... را نشان می‌دهد درگوشهٔ پایین–سمت چپ پنجرهٔ براوزر ظاهر می‌شود.

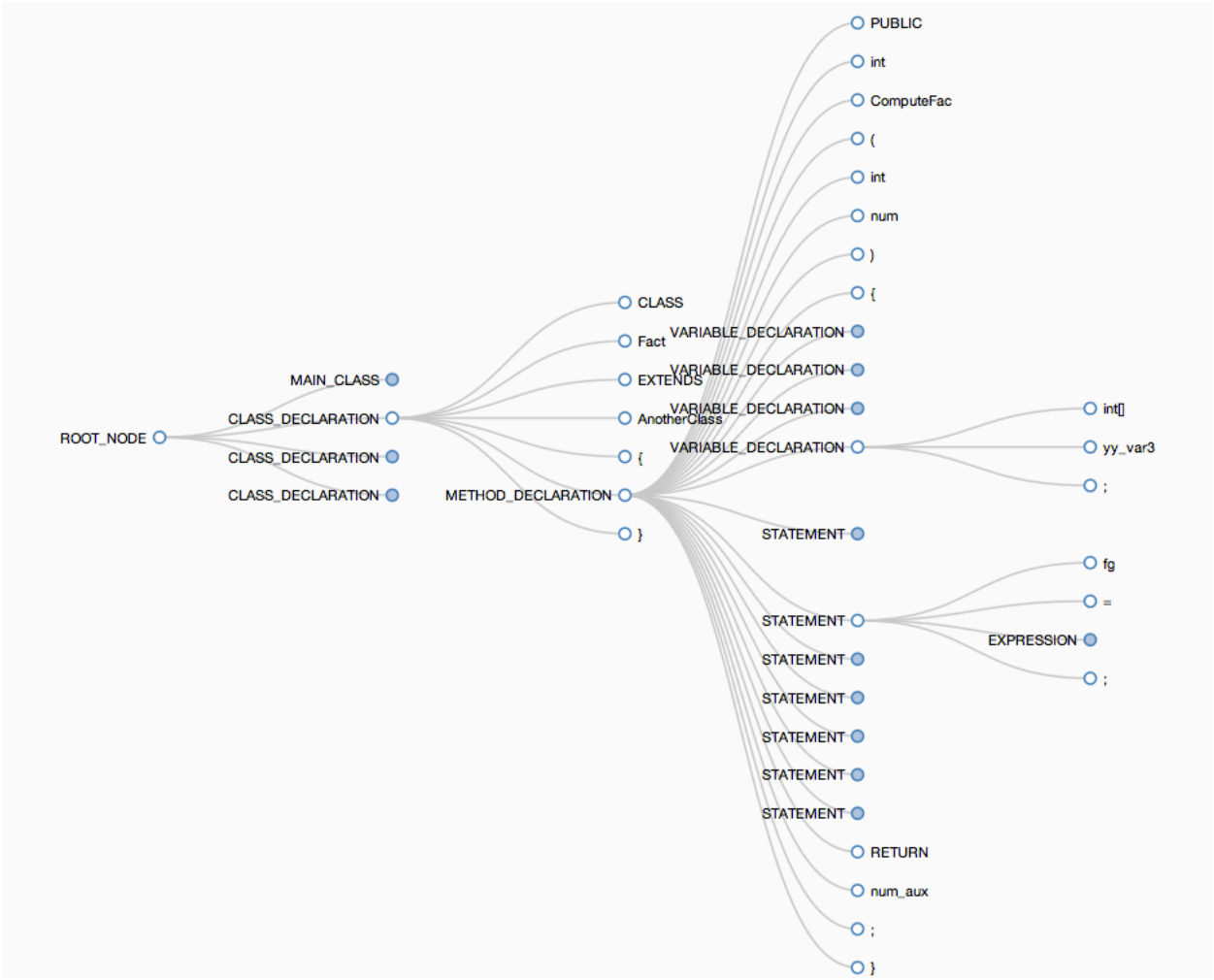
برای اجرای پارسر، فایل public/index.html را در یک براوزر مدرن بازکنید.

در ابتدا باید فایل ورودی خود را در قسمت بالای صفحه واردکنید و دکمهٔ Parse it رو بزنید. ما ۱۸ فایل تست آماده کرده‌ایم که نتایج خروجی تمام آن‌ها در ادامهٔ این گزارش آورده شده‌اند. اگر فایل اشکال نحوی داشته باشد (مثل تست شمارهٔ ۹) فقط در قسمت Logs ارور چاپ می‌کند (مانند شکل زیر).

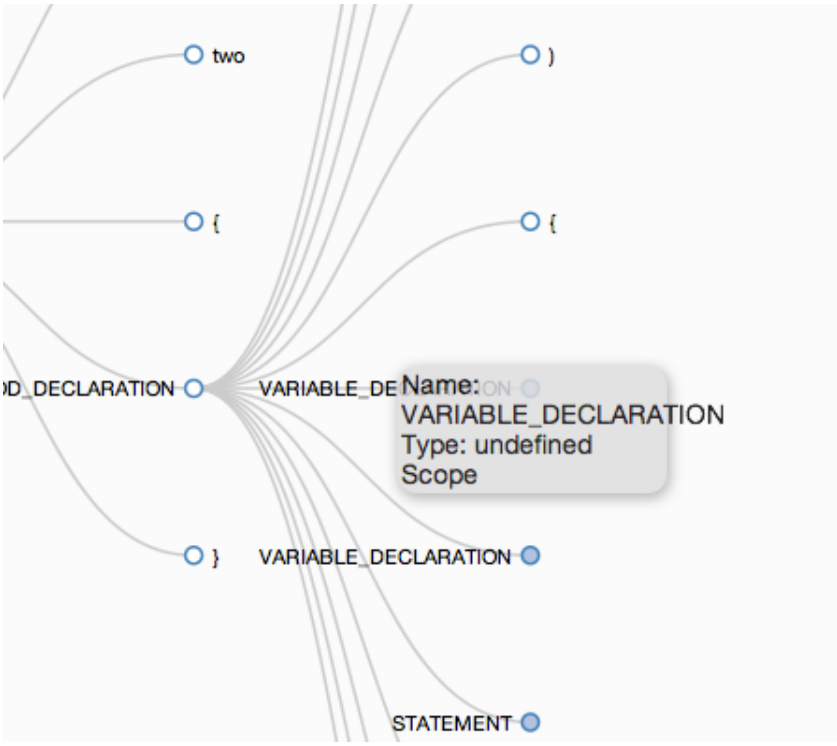


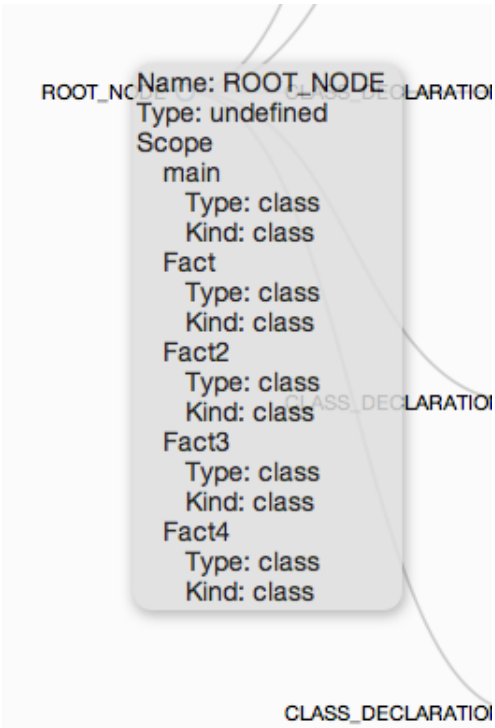
در غیر این صورت، درخت پارس ساخته می‌شود و در قسمت AST نمایش داده می‌شود (مانند شکل دوم در زیر). سپس تابع validate برای نود root صدا زده می‌شود و برنامه از نظر تایپ هم مورد بررسی قرار می‌گیرد. اگر خطای معنایی داشته باشد، در قسمت Logs چاپ می‌شود (مانند شکل اول در زیر).





با کلیک روی گره‌ها می‌توانید فرزندانشان را هم ببینید. اگر ماوس را روی یک گره نگه دارید، متغیرها و متدها و کلاس‌هایی که در scope ـش تعریف شده اند، نام، تایپ (اگر تایپ داشته باشد) و صفات دیگر گره نمایش داده می‌شوند.


two
)
{
{
D_DECLARATION
VARIABLE_DE
Name:
VARIABLE_DECLARATION
Type: undefined
Scope
}
VARIABLE_DECLARATION
STATEMENT


ROOT_NO
Name: ROOT_NODE
Type: undefined
Scope
main
Type: class
Kind: class
Fact
Type: class
Kind: class
Fact2
Type: class
Kind: class
Fact3
Type: class
Kind: class
Fact4
Type: class
Kind: class
LARATIO
LARATIO
CLASS_DECLARATIO

در قسمت Nonterminal ـها تمام قواعد تولید آورده شده اند:

AST | Nonterminals | Productions | Table | About us

NONTERMINALS

```
start
     nullable: No
     firsts: .
     follows: $end

        Productions: 1


initializer
     nullable: No
     firsts: .
     follows: CLASS

        Productions: 2


goal
     nullable: No
     firsts: CLASS
     follows: $end

        Productions: 3


statement_list
     nullable: Yes
     firsts: {,IF,WHILE,SYSOUT,ID
     follows: },RETURN

        Productions: 4 5


statement
     nullable: No
     firsts: {,IF,WHILE,SYSOUT,ID
     follows: {,IF,WHILE,SYSOUT,ID,ELSE,RETURN,}

        Productions: 6 7 8 9 10 11
```

در قسمت Productions هم اطلاعات آورده شده است:

AST | Nonterminals | Productions | Table | About us

PRODUCTIONS

```
$accept -> start $end
start -> initializer goal
initializer -> .
goal -> main_class class_decl_list EOF
statement_list -> statement statement_list
statement_list ->
statement -> { statement_list }
statement -> IF ( expression ) statement ELSE statement
statement -> WHILE ( expression ) statement
statement -> SYSOUT ( expression ) ;
statement -> ID = expression ;
statement -> ID [ expression ] = expression ;
expression_list -> expression expression_comma_list
expression_list ->
expression_comma_list -> , expression expression_comma_list
expression_comma_list ->
expression -> expression & expression
expression -> expression < expression
expression -> expression + expression
expression -> expression - expression
expression -> expression * expression
expression -> expression [ expression ]
expression -> expression . LENGTH
expression -> expression . ID ( expression_list )
expression -> INTEGER_LITERAL
expression -> TRUE
expression -> FALSE
expression -> ID
expression -> THIS
expression -> NEW INT [ expression ]
expression -> NEW ID ( )
expression -> ! expression
expression -> ( expression )
type -> INT [ ]
type -> BOOLEAN
type -> INT
type_id -> ID ID
type_id -> type ID
type_id_list -> type_id type_id_comma_list
type_id_list ->
type_id_comma_list -> , type_id type_id_comma_list
type_id_comma_list ->
```

در قسمت Table، جدول پارسر آورده شده. روی هر خانهٔ جدول که کلیک کنید، اطلاعات مربوط به آن خانه (قاعدهٔ تولید، قاعدهٔ shift و قاعدهٔ reduce) نشان داده می‌شود:

31										
32			s40							
33										
34										
35										
36			s41							
37 type_id -> ID ID . #lookaheads= ; ( ) ,			r36 - Reduce by 36) type_id -> ID ID	r36			r36			r36 - Reduce by 36) type_id
38 type_id -> type ID . #lookaheads= ; ( ) ,			r37 - Reduce by 37) type_id -> type ID	r37			r37			r37 - Reduce by 37) type_id

↓states	$end	&
0 $accept -> .start $end #lookaheads= $end start -> .initializer goal #lookaheads= $end initializer -> .. #lookaheads= CLASS		
1 $accept -> start .$end #lookaheads= $end	a	
2		
3		
4	r1 - Reduce by 1) start -> initializer goal	
5		

پروژه از خط فرمان هم قابل اجراست؛ کافی‌ست Node.js و CoffeeScript را نصب کنید و در شاخهٔ اصلی، make را بزنید.

test01.java

```
class one {
  public static void main(String[] args) {
  }
}
class two {
}
class three extends two {



}
```

Output:

test02.java

```
class one {
  public static void main(String[] args) {
    if (args.length < 2)
      System.out.println(1 + 2 * 3);
    else
      {
        args[0] = args[1];
      }
  }
}
class two {
}
class three extends two {

}
```

Output:

test03.java

```
class one {
  public static void main(String[] args) {
    if (args.length < 2)
      System.out.println(1 < 2 * 3);
    else
      {
        args[0] = args[1];
      }
  }
}
class two {
}
class three extends two {
}
```

Output:

```
System.out.println((1) < ((2) * (3)));
                   ^~-- type should be 'int'
```

test04.java

```
class one {
  public static void main(String[] args) {
    if (args.length < 2)
      System.out.println(1);
    else
      args[0] = args[1];
  }
}
class two {
  int a;
  int[] b;
  char c;
  character a;
}
```

Output:

```
Sorry, two! A variable with the same name (a) already exists!
```

test05.java

```
class one {
  public static void main(String[] args) {
  }
}
class two {
  int x;
  int y;
  int[] z;

  public int[] fun() {
    y = z.length;
    return y;
  }

  public character fun2(int[] x) {
    return x[true];
  }
}
```

Output:

```
x [ true ] ...
    ^~-- type should be 'int'
```

test06.java

```
class one {
  public static void main(String[] args) {
  }
}
class two {
  int x;
  int y;
  int[] z;

  public int[] fun() {
    y = z.length;
    return y;
  }

  public character fun2(int[] x) {
    return x[1 + 2<2 + 3];
  }
}
```

Output:

```
x [ ((1) + (2)) < ((2) + (3)) ] ...
     ^~-- type should be 'int'
```

test07.java

```
class one {
  public static void main(String[] args) {
  }
}
class two {
  public int[] func() {
    y = z.length;
    return y;
  }

  public character func() {
    return 0;
  }
}
```

Output:

```
Sorry, two! A method with the same name (func) already exists!
```

test08.java

```
class one {
  public static void main(String[] args) {
  }
}
class two {
  int func;

  public character func() {
    return 0;
  }
}
```

Output:

```
Sorry, two! A variable with the same name (func) already exists!
```

test09.java

```
class one {
  public static void main(String[] args) {
  }
}
class two {
  public character func() {
    x = 10;
  }
}
```

Output:

```
Error: Parse error on line 7:
...c() {    x = 10;  }}
---------------------^
Expecting 'RETURN', got '}'
```

test10.java

```
class one {
  public static void main(String[] args) {
  }
}
class two {
  public character func() {
    int x;
    int y;
    x = y < true;
    return x;
  }
}
```

Output:

```
y < true
        ^~-- type should be 'int'
```

test11.java

```
class one {
  public static void main(String[] args) {
  }
}
class two {
  public character func() {
    boolean x;
    int y;
    x = true && y;
    return x;
  }
}
```

Output:

```
true && y
         ^~-- type should be 'boolean'
```

test12.java

```
class one {
  public static void main(String[] args) {
  }
}
class two {
  public character func() {
    boolean x;
    int y;
    x = x - y;
    return x;
  }
}
```

Output:

```
x - y
          ^~-- type should be 'int'
```

test13.java

```
class one {
  public static void main(String[] args) {
  }
}
class two {
  public character func() {
    boolean x;
    int y;
    x = !(true);
    x = !y;
    return x;
  }
}
```

Output:

```
! y
  ^~-- type should be 'boolean'
```

test14.java

```
class one {
  public static void main(String[] args) {
  }
}
class two {
  public int func() {
    System.out.println(this.func());
    System.out.println(this.func2());
    return 0;
  }

  public int[] func2() {
    int[] res;
    return res;
  }
}
```

Output:

```
System.out.println((this).func2());
                   ^~-- type should be 'int'
```

test15.java

```
class one {
  public static void main(String[] args) {
  }
}
class two {
  public int func() {
    System.out.println(this.func());
    System.out.println(this.func2().length);
    return 0;
  }

  public int[] func2() {
    int[] res;
    return res;
  }
}
```

Output:

test16.java

```
class one {
  public static void main(String[] args) {
  }
}
class two {
  public int func() {
    return 0;
  }
}
class three {
  public int func() {
    return 0;
  }
}
```

Output:

test17.java:

```
class one {
  public static void main(String[] args) {
  }
}
class two {
  public int func() {
    return 0;
  }
}
class three extends two {
  public int func() {
    return 0;
  }
}
```

Output:

```
Sorry, three! Your parent class (two) already has a method named
func!
```

test18.java

```
class Factorial {
    public static void main(String[] args) {
        System.out.println(1);
        if (1<2 && true)
            alpha = b;
        else {
            System.out.println(1+2);
        }
    }
}

class Fact extends AnotherClass {
    public int func() {
        return 0;
    }

    public int ComputeFac(int num) {
        int num_aux;
        booleana yy_var;
        boolean yy_var2;
        int[] yy_var3;

        ff = new int[10];
        fg = true[2];
        fg[1] = 2-3;

        if (num < 1)
            num_aux = 1;
        else {
            num_aux1 = num * (this.ComputeFac(num-1));
            num_aux2 = num * (this.ComputeFac(num-1));
        }
        num_aux3 = num * (this.ComputeFac(num-1));
        while (this.length < 0)
            if (num_aux < num)
```

```
                num_aux4 = num * (this.ComputeFac(num-1));
            else
                y = 1;

        System.out.println(1+2+3*4);

        return num_aux;
    }
}

class Fact2 extends Fact {
    public int ComputeFac(int num, booool numssss, boolean bool)
{
        int x;
        return 0;
    }
}


class Fact3 extends Fact {
    bool x;
    boolean y;
    int method1;
    public int method1() {
        return 0;
    }
    public int[] method2() {
        return 0;
    }
    public some_type method3() {
        int zzz;
        zzz = this.method1();
        return 0;
    }
    public bool method2() {
        return 1;
    }
}
```

```
class Fact4 extends Fact3 {
    public int method1() {
        return 0;
    }
}
```

Output:

```
Sorry, Fact3! A method with the same name (method2) already exists!
Sorry, Fact3! A variable with the same name (method1) already exists!

true [ 2 ] ...
     ^~-- type should be 'int'
```

دقت کنید که ابتدا به‌خاطر تعریف شدن دوبارهٔ method1 ایراد می‌گیرد، و سپس به‌خاطر این‌که متغیری با نام method2 وجود داشته. اما در مورد کلاس Fact4 خطایی نمی‌دهد؛ چون به‌خاطر خطای هم‌نام بودن تابع با متغیر، در Fact3 متدی با نام method1 ساخته نشده بود و با این‌که Fact4 از Fact3، extend شده خطایی چاپ نمی‌شود.