به نام خدا

افزایش تست‌پذیری برنامه با اعمال ریفکتورینگ خودکار روی کد پروژه

فهرست مطالب:

# مقدمه

# شرح تئوری کار

# شرح کد پروژه

## تولید کلاس دیگرام

## محاسبه پیچیدگی و تست‌پذیری کد

## بازسازی الگو کارخانه(factory pattern)

## بازسازی الگو تزریق(injection pattern)

# شرح آزمایش‌ها

# نتیجه گیری

# کارهای آینده

# مقدمه:

هدف از این پروژه ارائه روشی برای تایین میزان آزمون پذیری برنامه ها می باشد. برای این منظور می باید بتوان با تحلیل خود برنامه میزان وابستگی بین کلاس ها را مشخص کرد. مسلما با افزایش میزان وابستگی آزمون واحد (unit test) کلاس ها پیچیده تر می گردد. در واقع این وابستگی موجب می شود که ردیابی خطا در داخل برنامه پیچیده شود چرا که با مشاهده خطا در یک متد در داخل یک کلاس به سادگی مشخص نمی شود که ایا علت خطا در آن متد است یا در متدهایی که به آن ها وابستگی وجود دارد. می توان با استفاده از الگوهایی مانند الگوی تزریق(injection pattern) و الگو کارخانه (factory pattern) و به کار گیری راهکار های بازسازی(refactoring code) این گونه اشکالات یا در اصطلاح بوی بد کد(code smell) را از میان برداشت. در این پروژه با استفاده از یک مولد کامپایلر و برای برنامه های جاوا کد پایتون برای مراحل زیر ایجاد می گردد :

1. ایجاد ابزاری برای استخراج مدل ارتباطی کلاس ها از متن پروژه های جاوا
2. ایجاد ابزاری برای تحلیل مدل وابستگی کلاس ها و محاسبه میزان آزمون پذیری کد بر اساس میزان وابستگی کلاس ها به یک دیگر
3. ایجاد ابزاری برای اعمال الگوی تزریق بر کلاس های جاوا و در واقع تولید کلاس رابط(interface) برای برقراری ارتباط با هر کلاس
4. پیاده سازی الگوی کارخانه با بررسی چگونگی ارتباط بین کلاس ها

با استفاده از ابزارهای فوق سعی می کنیم که فرایند TDD را در قالب فرایند جدید پیشنهادی خود به نام TsDD (testability driven development ) بهبود بخشید.

# شرح تئوری کار:

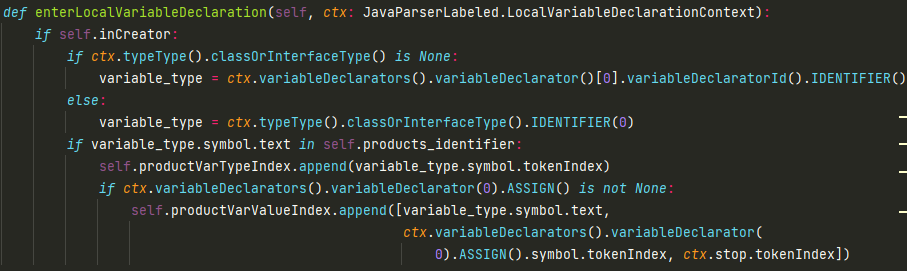
شرح کد پروژه:

برای خودکار انجام دادن این بازنمایی از 3 بار پیمایش DFS درخت تجزیه استفاده شده است. برای بدست آوردن درخت تجزیه و پیمایش آن به صورت DFS از کتابخانه ANTLR استفاده شده است.

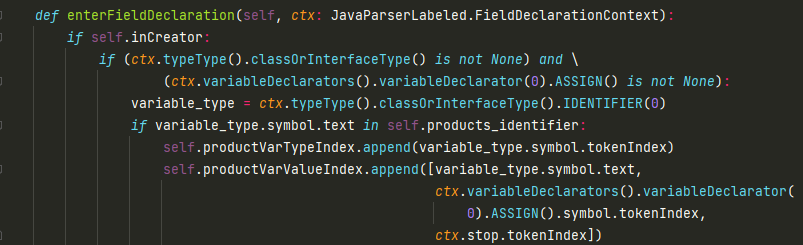
کلاس FixCreatorListener:

در این کلاس Listener مربوط به بازسازی کلاس creator نوشته شده است که در زیر به موارد مهم آن می‌پردازیم:

* در کد زیر تمام متغیرهای محلی که نوع آن‌ها برابر یکی از productها باشد را در productVarValueIndex ذخیره می‌کند.



* کد زیر نیز مانند قسمت بالا عمل می‌کند فقط تنها فرق آن این است که متغیرهای کلاس را بررسی می‌کند.



* در کد زیر در کلاس سازنده(creator) از نوع interface ساخته شده به جای نوع(type) کلاس‌های محصول استفاده می‌شود.



* برای فهم بهتر نقش کد بالا در این برنامه به مثال زیر توجه کنید:

+import simulator.Interface34;

public class SuiteGUI extends JFrame {

private Function[] availableFunctions = new Function[7];

- private Linear linear = new Linear();

+ private Interface34 linear = new Linear();

- private Random random1 = new Random();

+ private Interface34 random1 = new Random();

- private Random random2 = new Random();

+ private Interface34 random2 = new Random();

- private Discrete discrete = new Discrete();

+ private Interface34 discrete = new Discrete();

- private Raise raise = new Raise();

+ private Interface34 raise = new Raise();

- private MetDataFunction metData = new MetDataFunction();

+ private Interface34 metData = new MetDataFunction();

- private Raise price = new Raise();

+ private Interface34 price = new Raise();

* در مثال بالا Linear, Random, Discrete, Raise, MetDataFunction کلاس‌های محصول(product) هستند که همگی به Interface34 تبدیل شده‌اند.
* در مثال بالا Interface34 در ابتدا کد هم به صورت خودکار Import شده است.

کلاس FixProductListener: