به نام خدا

افزایش تست‌پذیری برنامه با اعمال ریفکتورینگ خودکار روی کد پروژه

فهرست مطالب:

1. مقدمه
2. شرح پروژه
   2.1. تولید کلاس دیاگرام
   2.2. محاسبه پیچیدگی و تست‌پذیری کد
   2.3. بازسازی الگو کارخانه(factory pattern)
   2.4. بازسازی الگو تزریق(injection pattern)
3. شرح آزمایش‌ها
4. نتیجه گیری
5. کارهای آینده

1. مقدمه:

هدف از این پروژه ارائه روشی برای تایین میزان آزمون پذیری برنامه‌ها می‌باشد. برای این منظور 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 می‌گردد. در واقع این وابستگی موجب می‌شود که ردیابی خطا در داخل برنامه پیچیده شود چرا که با مشاهده خطا در یک متد در داخل یک کلاس به سادگی مشخص نمی‌شود که آیا علت خطا در آن متد است یا در متدهایی که به آن ها وابستگی وجود دارد. می‌توان با استفاده از الگوهایی مانند الگوی تزریق(injection pattern) و الگو کارخانه (factory pattern) و به کارگیری راهکارهای 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 برنامه‌های جاوا کد پایتون برای مراحل زیر ایجاد می‌گردد :

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

3.1: تولید کلاس دیاگرام استخراج مدل ارتباطی کلاس‌ها از متن پروژه های جاوا:

برای درک بهتر چگونگی ساخت

شرح تئوری کار:
الگوی کارخانه(factory pattern):

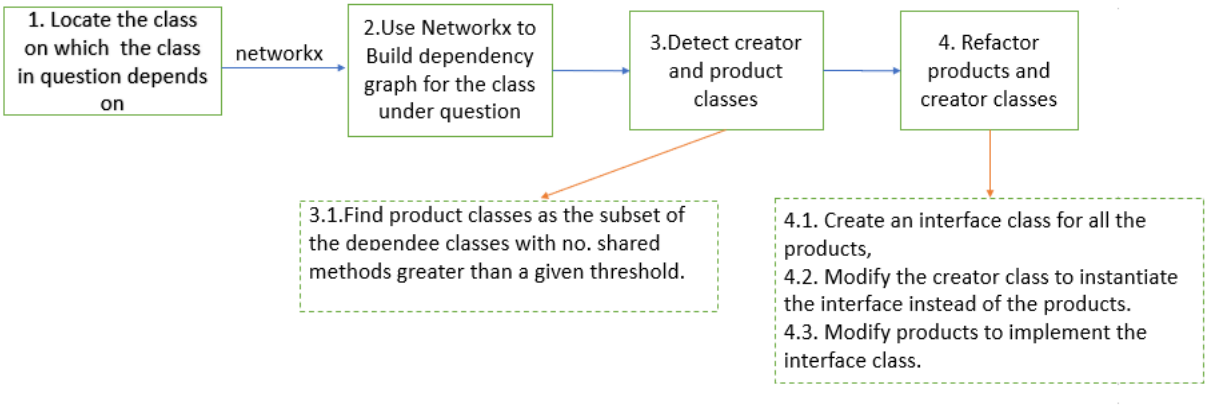
- روش کارخانه یک الگوی طراحی سازنده است که یک interface برای کلاس‌های محصول(products) ایجاد می‌کند و در کلاس سازنده(creator) اشیا محصول از نوع interface ساخته می‌شوند.
- برای فهم بیشتر این الگو به مثال زیر توجه کنید:
  - فرض کنید شما یک برنامه برای خواندن فایل‌های مختلف تصویری نوشته‌اید که در این برنامه فایل‌های تصویری مختلف مثل gif و jpeg خوانده شده و decode می‌شوند.
  - بعد از مدتی برنامه شما معروف می‌شود و نیاز است که انواع مختلف تصویر و فیلم مانند png را نیز بتواند بخواند و decode کند.
  - در این حالت شاید نیاز باشد مقدار زیادی از کد را تغییر دهید تا بتوان این نوع فایل تصویری را نیز پوشش دهد.
  - پس همیشه نیاز دارید کد برنامه خود را برای اضافه کردن یک نوع جدید تصویر تغییر دهید.
- راهکار:
  - الگو کارخانه به شما پیشنهاد می‌کند که به جای فراخانی مستقیم شی(object) مورد نظر از یک کارخانه برای این کار استفاده کنید و این شی کارخانه مسئول تشخیص نوع تصویر مورد نظر است و شی مورد نظر را تولید می‌کند.

مزیت‌ها و عیب‌های این الگو:

- ✓ شما از اتصال محکم بین سازنده و محصولات بتنی اجتناب می‌کنید.
- ✓ اصل مسئولیت واحد(Single Responsibility Principle): می‌توانید کد ایجاد محصول را به یک مکان در برنامه منتقل کنید تا پشتیبانی از کد آسان‌تر شود.
- ✓ اصل باز/بسته(Open/Close Principle): شما می‌توانید انواع جدیدی از محصولات را بدون شکستن کد مشتری موجود به برنامه معرفی کنید.
- ✘ ممکن است کد پیچیده‌تر شود زیرا برای پیاده‌سازی الگو باید زیر کلاس‌های جدید زیادی معرفی کنید.

برای اعمال الگو کارخانه(factory pattern) روی پروژه مدیریت فایل‌های تصویری باید مراحل زیر انجام شوند:

شرح کد پروژه:



شکل 1: نمای کلی مراحل انجام شده برای الگوی کارخانه

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.

```
def enterLocalVariableDeclaration(self, ctx: JavaParserLabeled.LocalVariableDeclarationContext):
    if self.inCreator:
        if ctx.typeType().classOrInterfaceType() is None:
            variable_type = ctx.variableDeclarators().variableDeclarator()[0].variableDeclaratorId().IDENTIFIER()
        else:
            variable_type = ctx.typeType().classOrInterfaceType().IDENTIFIER(0)
        if variable_type.symbol.text in self.products_identifier:
            self.productVarTypeIndex.append(variable_type.symbol.tokenIndex)
            if ctx.variableDeclarators().variableDeclarator(0).ASSIGN() is not None:
                self.productVarValueIndex.append([variable_type.symbol.text,
                                                  ctx.variableDeclarators().variableDeclarator(
                                                      0).ASSIGN().symbol.tokenIndex, ctx.stop.tokenIndex])
```

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

```
def enterFieldDeclaration(self, ctx: JavaParserLabeled.FieldDeclarationContext):
    if self.inCreator:
        if (ctx.typeType().classOrInterfaceType() is not None) and \
                (ctx.variableDeclarators().variableDeclarator(0).ASSIGN() is not None):
            variable_type = ctx.typeType().classOrInterfaceType().IDENTIFIER(0)
            if variable_type.symbol.text in self.products_identifier:
                self.productVarTypeIndex.append(variable_type.symbol.tokenIndex)
                self.productVarValueIndex.append([variable_type.symbol.text,
                                                  ctx.variableDeclarators().variableDeclarator(
                                                      0).ASSIGN().symbol.tokenIndex,
                                                  ctx.stop.tokenIndex])
```

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

```
def exitCompilationUnit(self, ctx: JavaParserLabeled.CompilationUnitContext):
    self.token_stream_rewriter.insertAfter(program_name=self.token_stream_rewriter.DEFAULT_PROGRAM_NAME,
                                           index=self.packageIndex,
                                           text='\n' + self.interface_import_text + '\n')

    for item in self.productVarTypeIndex:
        self.token_stream_rewriter.replace(program_name=self.token_stream_rewriter.DEFAULT_PROGRAM_NAME,
                                           from_idx=item,
                                           to_idx=item,
                                           text=self.interfaceName)

    new_product_method = "\n"
    for item in self.productConstructorMethod:
        new_product_method += item
    self.token_stream_rewriter.insertAfter(program_name=self.token_stream_rewriter.DEFAULT_PROGRAM_NAME,
                                           index=self.creator_start_index,
                                           text=new_product_method)
```

- برای فهم بهتر نقش این کلاس 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:

- در کد زیر سعی می‌شود مکانی که قرار است عبارت "implement interface_name" اضافه شود یافته را به لیست productsClassIndex اضافه کند.

```
def enterClassDeclaration(self, ctx: JavaParserLabeled.ClassDeclarationContext):
    if ctx.IDENTIFIER().getText() in self.products_identifier:
        self.inProducts = True
        if ctx.typeType().classOrInterfaceType().IDENTIFIER() is not None:
            self.productsClassIndex.append(ctx.typeType().classOrInterfaceType().IDENTIFIER()[0].symbol.tokenIndex)
        else:
            self.productsClassIndex.append(ctx.IDENTIFIER().symbol.tokenIndex)
        self.currentClass = ctx.IDENTIFIER().symbol.text
```

- در این قسمت عبارت import برای interface ایجاد شده به ابتدا فایل اضافه می‌شود و همچنین عبارت "implement interface_name" نیز به کلاس محصول(product) اضافه می‌شود.

```
def exitCompilationUnit(self, ctx: JavaParserLabeled.CompilationUnitContext):
    self.token_stream_rewriter.insertAfter(program_name=self.token_stream_rewriter.DEFAULT_PROGRAM_NAME,
                                           index=self.packageIndex,
                                           text='\n' + self.interface_import_text + '\n')
    for item in self.productsClassIndex:
        self.token_stream_rewriter.insertAfter(program_name=self.token_stream_rewriter.DEFAULT_PROGRAM_NAME,
                                               index=item,
                                               text=" implements " + self.interfaceName)
```

- برای فهم بهتر نقش این کلاس به مثال زیر توجه کنید:

```
package simulator.SA.gui;
+import simulator.SA.Interface26;
+

 import jade.gui.GuiEvent;

 import java.awt.BorderLayout;
 import java.awt.Dimension;

-public class DemandCurveParameterFrame extends JInternalFrame {
```

```
+public class DemandCurveParameterFrame extends JInternalFrame implements Interface26 {

        private static final long serialVersionUID = 1208279781892420981L;

        private Logger log = Logger.getLogger(DemandCurveParameterFrame.class);
        private boolean flagFunctionSelected = false;
```

- در مثال بالا ابتدا عبارت import interface ساخته شده(Interface26) اضافه می‌شود و سپس عبارت implements Interface26 نیز به کلاس DemandCurveParameterFrame اضافه می‌شود.

کلاس ProductCreatorDetectorListener:

- در این قسمت نام و نوع خروجی بدست می‌آید و لیست پارامترهای ورودی با یک لیست خالی مقدار دهی اولیه می‌شود.

```
def enterMethodDeclaration(self, ctx: JavaParserLabeled.MethodDeclarationContext):
    self.current_method_info = {}
    if self.current_class == self.class_name:
        if self.current_class_body_public is not None:
            if self.current_class_body_public.getText() == ctx.getText():
                self.current_method_info['name'] = ctx.IDENTIFIER().getText()
                self.current_method_info['return_type'] = ctx.typeTypeOrVoid().getText()
                self.current_method_info['formal_parameters'] = []
                self.methods[ctx.IDENTIFIER().getText()] = {}
```

- در این قسمت پارامترهای ورودی method مورد نظر پیدا شده و به لیست متغیرهای ورودی method اضافه می‌شود.

```
def enterFormalParameter(self, ctx: JavaParserLabeled.FormalParameterContext):
    if 'formal_parameters' in self.current_method_info.keys():
        formal_parameter_info = list()
        formal_parameter_info.append(ctx.typeType().getText())
        formal_parameter_info.append(ctx.variableDeclaratorId().getText())
        self.current_method_info['formal_parameters'].append(formal_parameter_info)
```

تعریف حساسیت(sensitivity):

- میزان حساسیت الگوریتم تشخیص الگوی کارخانه(factory pattern) گفته می‌شود که طبق فرمول زیر بدست می‌آید:
No_common_methods: تعداد methodهایی که بین کلاس‌هایی که کلاس سازنده(creator) به آن‌ها وابسته است.
No_methods_class1: تعداد methodهای کلاس class1

Sensitivity = no_common_methods / max(no_methods_class1, no_methods_class2, . . .)

کلاس Factory:

- در این method کلاس‌های محصول و methodهایی از آن‌ها که مشترک هستند بر اساس پارامتر sensitivity مشخص می‌شوند.

```python
def __find_products(self, parent_class, method_class_dic, sensitivity):
    result = {'factory': int(parent_class), 'products': {'classes': [], 'methods': []}}

    for c1 in method_class_dic.keys():
        class_list = []
        method_list = method_class_dic[c1]

        len_c1_methods = len(method_class_dic[c1])
        for c2 in method_class_dic.keys():
            len_c2_methods = len(method_class_dic[c2])
            method_list_help = self.__compare_similarity_of_two_list(method_list, method_class_dic[c2])
            if max(len_c1_methods, len_c2_methods) == 0:
                continue

            if len(method_list_help) / max(len_c1_methods, len_c2_methods) >= sensitivity:
                method_list = method_list_help.copy()
                class_list.append(c2)

        if len(class_list) > len(result['products']['classes']):
            result['products']['classes'] = class_list
            for m in method_list:
                if method_class_dic[class_list[0]][m] != {}:
                    result['products']['methods'].append(method_class_dic[class_list[0]][m])
    return result
```

سودو کدهای الگوی کارخانه:

factory.py\Factory:

```
def refactor(sensitivity, class_diagram):
        internal_nodes = get_internal_nodes(class_diagram)
        for node in internal_nodes:
                neighbors = node.neighbors
                if len(neighbors) > 1:
                        neighbors_methods_dict = {}
                        for child_node in neighbors:
```

```
                    listener =
ProductCreatorDetectorListener(child_node.class_name)
                    listener.walk()
                    neighbors_methods_dict[child_node] = listener.methods
                result = find_products(node, neighbors_methods_dict,
sensitivity)
                if len(result.products) > 1:
                    interface =
create_interface(result.products_common_methods)
                    fix_creator(node, interface, products)
                    for product in products:
                        fix_product(product, interface)
```

```
def find_products(node, neighbors_methods_dict, sensitivity):
    factory_info = {}
    factory_info.creator = node
    candidate_product_classes = neighbor_methods_dict.get_classes()
    for class1 in candidate_product_classes:
        products = []
        method_list = class1.methods

        no_class1_methods = len(class1.methods)
        for class2 in candidate_product_classes:
            no_class2_methods = len(class2.methods)
            common_methods = get_similarity_of_two_list(method_list,
class2.methods)

            if len(common_methods) / max(no_class1_methods,
no_class2_methods) >= sensitivity:
                method_list = common_methods.copy()
                class_list.append(class2)
    factory_info.products = products
```

```
        factory_info.products_common_methods = method_list
        return factory_info
```

```
def fix_product(product, interface):
        product.add_import_statement(interface.package, interface.name)
        product.add_implement_statement(product.name, interface.name)
```

```
def fix_creator(creator, interface, products):
        creator.add_import_statement(interface.package)
        for product in products:
                creator.replace_type(product, interface.name)
```

```
def add_import_statement(package, name):
        package = enterPackageDeclaration()
        current_token = package.stop.token
        import = enterImportDeclaration()
        while import is not None:
                current_token = import.stop.token
                import = enterImportDeclartion()
        import_text = 'import ' + package + '.' + name + ';'
        token_stream_rewirter.insertAfter(current_token, import_text)
```

```
def add_implement_statement(class_name, interface_name):
        if enterClassDeclaration().name == class_name:
                class = enterClassDeclaration()
        while (enterClassDeclaration() is not None):
                if enterClassDeclaration().name == class_name:
                        class = enterClassDeclaration()
                        break
        implement_statement_token = None
        implement_statement_text = 'implement ' + interface + ';'
        if class.extend is not None:
                implement_statement_token = class.extend.token
```

```
        if class.implement is not None:
                implement_statement_token = class.implement.token
                implement_statement_text = ‘ ’ + interface + ‘;’
        token_stream_rewirter.insertAfter(implement_statement_token,
implement_statement_text)
```

```
def replace_type(product, interface_name):
        dependee = enterClassOrInterfaceType()
        while dependee is not None:
                if dependee == product.type:
                        token_stream_rewirter.replace(dependee.start.token,
                                                      dependee.start.token + 1,
                                                      interface_name)
```