Exercises 2

Giả sử architect quyết định rằng việc tổ chức viết code phải tuân theo quy định chặt chẽ về tổ chức và cấu trúc. Các nguyên lý được hướng dẫn như sau:

1. Các package trong dự án phải theo mẫu: com.companyname.* (*:tên bất kỳ)
2. Các class phải có tên là một danh từ hoặc cụm danh ngữ và phải bắt đầu bằng chữ hoa.
3. Mỗi lớp phải có một comment mô tả cho lớp. Trong comment đó phải có ngày tạo (created-date) và author.
4. Các fields trong các class phải là danh từ hoặc cụm danh ngữ và phải bắt đầu bằng một chữ thường.
5. Tất cả các hằng số phải là chữ viết hoa và phải nằm trong một interface.
6. Tên method phải bắt đầu bằng một động từ và phải là chữ thường
7. Mỗi method phải có một ghi chú mô tả cho công việc của method trừ phương thức default constructor, accessors/mutators, hashCode, equals, toString.

Hãy viết một tool cho phép kiểm tra toàn bộ dự án và xuất ra report cho các trường hợp không thỏa mãn.

Hướng dẫn

Sử dụng thư viện java parser (https://javaparser.org/) để parse các file source-code sau đó quét tất cả các trường hợp và tạo báo cáo.

```
<dependency>
    <groupId>com.github.javaparser</groupId>
    <artifactId>javaparser-core</artifactId>
    <version>3.25.8</version>
</dependency>
```

```
//     https://mvnrepository.com/artifact/com.github.javaparser/javaparser-core
implementation 'com.github.javaparser:javaparser-core:3.25.8'
```

Hướng dẫn cho việt viết code có thể tìm thấy trong sách tại trang https://javaparser.org/

Common Code

Duyệt qua thư mục và con của nó (sử dụng đệ quy) rồi lọc các file có phần mở rộng được chỉ định

```
package vn.edu.iuh.fit.examples;
import java.io.File;

public class DirExplorer {
	// interface for file handler
	public interface FileHandler {
		void handle(int level, String path, File file);
	}

	// interface for file filter
	public interface Filter {
		boolean interested(int level, String path, File file);
	}
```

```
	private FileHandler fileHandler;
	private Filter filter;

	public DirExplorer(Filter filter, FileHandler fileHandler) {
		this.filter = filter;
		this.fileHandler = fileHandler;
	}

	public void explore(File root) {
		explore(0, "", root);
	}

	private void explore(int level, String path, File file) {
		if (file.isDirectory()) {
			for (File child : file.listFiles()) {
				explore(level + 1, path + "/" + child.getName(), child);
			}
		} else {
			if (filter.interested(level, path, file)) {
				fileHandler.handle(level, path, file);
			}
		}
	}
}
```

Để thử cho lớp này, có thể dùng code kiểu cũ

```
	void test_old_style() {
		File projectDir = new File("T:\\ThucHanh\\json-demo");
		DirExplorer.Filter filter = new Filter() {
			@Override
			public boolean interested(int level, String path, File file) {
				return path.endsWith(".java");
			}
		};
		DirExplorer.FileHandler handler = new FileHandler() {
			@Override
			public void handle(int level, String path, File file) {
				System.out.println(path);
			}
		};
		DirExplorer di = new DirExplorer(filter, handler);
		di.explore(projectDir);
	}
```

Hoặc dùng lambda expression (recommendation)

```
public static void main(String[] args) {
	File projectDir = new File("T:\\ThucHanh\\json-demo");
	new DirExplorer((level, path, file) -> path.endsWith(".java"), (level, path,
file) -> {
			System.out.println(path);
	}).explore(projectDir);
	}
```

Đối với mỗi file Java, trước tiên chúng ta xây dựng Cây cú pháp trừu tượng (Abstract Syntax Tree -AST) cho mỗi file Java và sau đó điều hướng nó. Có hai chiến lược chính để làm như vậy:

1. Sử dụng visitor: đây là chiến lược phù hợp khi chúng ta muốn vận hành trên các loại nút AST cụ thể
2. Sử dụng trình vòng lặp đệ quy: điều này cho phép xử lý tất cả các loại nút

Các Visitor có thể được viết các lớp mở rộng có trong JavaParser, code của nó như sau

```
package vn.edu.iuh.fit.examples;
import com.github.javaparser.ast.Node;
public class NodeIterator {
    public interface NodeHandler {
        boolean handle(Node node);
    }
    private NodeHandler nodeHandler;
    public NodeIterator(NodeHandler nodeHandler) {
        this.nodeHandler = nodeHandler;
    }
    public void explore(Node node) {
        if (nodeHandler.handle(node)) {
            for (Node child : node.getChildNodes()) {
                explore(child);
            }
        }
    }
}
```

Duyệt qua các file java và in ra các lớp bên trong file

```
package vn.edu.iuh.fit.examples;
import java.io.File;
import com.github.javaparser.StaticJavaParser;
import com.github.javaparser.ast.body.ClassOrInterfaceDeclaration;
import com.github.javaparser.ast.visitor.VoidVisitorAdapter;
import com.google.common.base.Strings;
public class ListClassesExample {
    public static void listClasses(File projectDir) {
        new DirExplorer((level, path, file) -> path.endsWith(".java"), (level, path,
file) -> {
            System.out.println(path);
            System.out.println(Strings.repeat("=", path.length()));
            try {
                new VoidVisitorAdapter<Object>() {
                    @Override
                    public void visit(ClassOrInterfaceDeclaration n, Object arg) {
                        super.visit(n, arg);
                        System.out.println(" * " + n.getName());
                    }
                }.visit(StaticJavaParser.parse(file), null);
                System.out.println(); // empty line
            } catch (Exception e) {
                new RuntimeException(e);
            }
        }).explore(projectDir);
    }
    public static void main(String[] args) {
        File projectDir = new File("T:\\ThucHanh\\json-demo");
        listClasses(projectDir);
    }
}
```

Nguyên tắc chung

1. Dùng DirExplorer để duyệt qua các files.
2. Trong mỗi file java được duyệt qua, dùng visitor để "ghé thăm" các thành phần trong file.
3. Override hàm visit với các tham số tùy đối tượng bạn muốn truy cập

Ví dụ

```
package vn.edu.iuh.fit.examples;

import java.io.File;
import com.github.javaparser.StaticJavaParser;
import com.github.javaparser.ast.PackageDeclaration;
import com.github.javaparser.ast.body.FieldDeclaration;
import com.github.javaparser.ast.body.MethodDeclaration;
import com.github.javaparser.ast.visitor.VoidVisitorAdapter;
import com.google.common.base.Strings;

public class CommonOperations {
	public static void listMethodCalls(File projectDir) {
		new DirExplorer((level, path, file) -> path.endsWith(".java"), (level,
path, file) -> {
			System.out.println(path);
			System.out.println(Strings.repeat("=", path.length()));
			try {
				new VoidVisitorAdapter<Object>() {

					@Override
					public void visit(PackageDeclaration n, Object arg)
{
						super.visit(n, arg);
						System.out.println(n.getNameAsString());
					}

					@Override
					public void visit(FieldDeclaration n, Object arg) {
						super.visit(n, arg);
						System.out.println(" [L " + n.getBegin() + "]
" + n);
					}

					@Override
					public void visit(MethodDeclaration n, Object arg) {
						super.visit(n, arg);
						System.out.println(" [L " + n.getBegin() + "]
" + n.getDeclarationAsString());
					}

				}.visit(StaticJavaParser.parse(file), null);
			} catch (Exception e) {
				new RuntimeException(e);
			}
		}).explore(projectDir);
	}

	public static void main(String[] args) {
		File projectDir = new File("T:\\ThucHanh\\json-demo\\src\\main\\java");
		listMethodCalls(projectDir);
	}
}
```