OOPT Stage 2050

<Construct>

Software Modeling & Analysis

소프트웨어 모델링 및 분석

보고서 Version. 3

Team. T1

201111388 조연호

201211374 이창오

201211379 장종훈

201314196 양동혁

Stage 2050. Construct

Activity 2151. Implement Class & Methods Definitions

Type	Class
Name	Controller
Purpose	전반적인 프로그램의 수행을 제어하는 클래스
Overview (Class)	폴더 선택, 파일 분석, 일치율 계산, 화면 출력 등을 제어한다.
Cross Reference	Select Folder, Start, Analyze File, Change Center, Calculate File, Display Sync
Input (Method)	-
Output (Method)	-
Abstract Operation (Method)	-
Exceptional Courses of Events	N/A

Type	Method
Name	mkFileInstance()
Purpose	소스 코드 파일들의 객체를 생성하는 메소드
Overview (Class)	소스 코드 파일들의 객체를 생성하여 이름과 내용을 초기화한다.
Cross Reference	N/A
Input (Method)	N/A
Output (Method)	void
Abstract Operation (Method)	소스코드 파일들의 객체를 생성한다. 각각의 객체에 파일 입출력으로 내용을 저장한다.
Exceptional Courses of Events	1. [numOfFile == 0] : ".c 확장자를 가진 파일이 없는 폴더입니다." 메시지 출력 2. [numOfFile > 100] : "파일이 100개 이상인 폴더입니다." 메시지 출력 3. [numOfFile == -1] : "0Byte 파일이 있는 폴더입니다." 메시지 출력

Type	Method
Name	displayMain()
Purpose	시작 화면 출력하는 메소드
Overview (Class)	시작 화면을 출력한다.
Cross Reference	Select Folder
Input (Method)	N/A
Output (Method)	void
Abstract Operation (Method)	메인 화면 및 폴더 선택 버튼을 출력한다. 폴더 선택 완료 시 시작하기 버튼을 출력한다.
Exceptional Courses of Events	N/A

Type	Method
Name	selectFolder()
Purpose	사용자가 폴더를 선택할 수 있도록 하는 메소드
Overview (Class)	사용자가 폴더를 선택하는 기능을 제공하여 선택된 경로를 출력한다.
Cross Reference	showFolderPath()
Input (Method)	N/A
Output (Method)	void
Abstract Operation (Method)	폴더 선택 기능을 제공한다. 선택된 경로를 *showFolderPath()*를 호출하여 출력한다.
Exceptional Courses of Events	N/A

Type	Method
Name	showFolderPath()
Purpose	선택된 폴더 경로를 출력하는 메소드
Overview (Class)	선택된 폴더 경로를 메인 화면에 출력한다.
Cross Reference	Select Folder
Input (Method)	N/A
Output (Method)	void
Abstract Operation (Method)	선택된 폴더 경로인 *chooser.getSelectedFile()*를 출력한다.
Exceptional Courses of Events	N/A

Type	Method
Name	start()
Purpose	시작하기 버튼에 대한 동작을 실시하는 메소드
Overview (Class)	시작하기 버튼을 누르면 지정된 폴더의 파일 객체를 생성
Cross Reference	mkFileInstance()
Input (Method)	N/A
Output (Method)	void
Abstract Operation (Method)	*folderPath*에 *chooser.getSelectedFile() + "~~₩₩~~"* 를 대입한다. *mkFileInstance()*를 통해 파일 객체를 생성한다.
Exceptional Courses of Events	FileNotFoundException, IOException
	소스 코드를 읽지 못하는 경우 "폴더 경로가 올바르지 않습니다." 오류 메시지를 출력하고, 결과 화면을 출력하지 않는다.

Type	Method
Name	displayResult()
Purpose	결과 화면 출력하는 메소드
Overview (Class)	결과 화면 출력한다.
Cross Reference	N/A
Input (Method)	N/A
Output (Method)	void
Abstract Operation (Method)	결과 화면과 소스 코드 파일 리스트, 클라우드를 출력한다. 마우스 클릭과 마우스 오버에 대한 동작을 실시한다.
Exceptional Courses of Events	N/A

Type	Method
Name	showCloud()
Purpose	클라우드 출력하는 메소드
Overview (Class)	클라우드 출력한다.
Cross Reference	Display Result
Input (Method)	N/A
Output (Method)	void
Abstract Operation (Method)	클라우드 출력한다.
Exceptional Courses of Events	N/A

Type	Method
Name	displaySync()
Purpose	소스 코드 파일의 일치율을 출력하는 메소드
Overview (Class)	소스 코드 파일의 일치율을 출력한다.
Cross Reference	Display Result
Input (Method)	N/A
Output (Method)	void
Abstract Operation (Method)	소스 코드 파일의 일치율을 출력한다.
Exceptional Courses of Events	N/A

Type	Method
Name	changeCenter()
Purpose	비교 중심이 되는 파일을 변경하는 메소드
Overview (Class)	사용자가 선택한 파일을 중심으로 결과 화면을 다시 출력한다.
Cross Reference	Calculate File, Display Sync
Input (Method)	N/A
Output (Method)	void
Abstract Operation (Method)	사용자가 클라우드 또는 오른쪽 리스트에서 파일을 클릭하면 그 파일을 중심으로 다른 파일과의 일치율을 계산하여 결과화면을 출력한다.
Exceptional Courses of Events	N/A

Type	Method
Name	showDialog()
Purpose	폴더 선택 창을 출력하는 메소드
Overview (Class)	폴더 선택 창을 출력한다.
Cross Reference	Select Folder
Input (Method)	N/A
Output (Method)	void
Abstract Operation (Method)	메인 화면에서 사용자가 폴더 선택 버튼을 누르면 폴더 선택 창을 띄운다.
Exceptional Courses of Events	N/A

Type	Method
Name	getFolderPath()
Purpose	String 변수 *folderPath*를 반환하는 메소드
Overview (Class)	String 변수 *folderPath*를 반환한다.
Cross Reference	N/A
Input (Method)	N/A
Output (Method)	String *folderPath*
Abstract Operation (Method)	String 변수 *folderPath*를 반환한다.
Exceptional Courses of Events	N/A

Type	Class
Name	Analyze
Purpose	생성된 파일 객체들 안에 저장된 코드를 분석하는 클래스
Overview (Class)	ArrayList *source*에 저장된 코드들을 수정을 거쳐 분석한 뒤 값을 저장한다.
Cross Reference	Analyze File, Change Annotation, Analyze Annotation, Analyze Line, Delete Annotation, Divide Code Line, Delete Printf, Find Type, Analyze Function, Analyze Variable, Analyze Preprocessor
Input (Method)	-
Output (Method)	-
Abstract Operation (Method)	-
Exceptional Courses of Events	N/A

Type	Method
Name	analyzeFile()
Purpose	전반적인 분석 기능을 수행하고 값을 저장하는 메소드
Overview (Class)	전반적인 분석 기능을 수행하고 값을 저장한다.
Cross Reference	Change Annotation, Analyze Annotation, Analyze Line, Delete Annotation, Divide Code Line, Delete Printf, Find Type, Analyze Function, Analyze Variable, Analyze Preprocessor
Input (Method)	Files *file*
Output (Method)	void
Abstract Operation (Method)	각각의 분석 및 코드 수정 기능을 맡고 있는 메소드들을 호출한다. 각각의 메소드를 통해 분석된 값을 *file* 객체에 저장한다.
Exceptional Courses of Events	N/A

Type	Method
Name	changeAnnotation()
Purpose	주석 분석 전에 '/**/'으로 주석 처리된 줄을 '//'으로 변경하는 메소드
Overview (Class)	'/**/'으로 주석 처리된 줄을 '//'으로 변경한다.
Cross Reference	Analyze File
Input (Method)	N/A
Output (Method)	void
Abstract Operation (Method)	ArrayList *source*에서 '/**/'으로 주석 처리된 줄을 '//'으로 변경한다.
Exceptional Courses of Events	N/A

Type	Method
Name	analyzeAnnotation()
Purpose	주석 처리된 줄의 개수를 계산하는 메소드
Overview (Class)	주석 처리된 줄의 개수를 계산한다.
Cross Reference	Analyze File
Input (Method)	N/A
Output (Method)	void
Abstract Operation (Method)	주석 처리된 줄의 개수를 계산하여 *numOfAnnotation*에 저장한다.
Exceptional Courses of Events	N/A

Type	Method
Name	analyzeLine()
Purpose	코드의 라인 수를 계산하는 메소드
Overview (Class)	코드의 라인 수를 계산한다.
Cross Reference	Analyze File
Input (Method)	N/A
Output (Method)	void
Abstract Operation (Method)	코드의 라인 수를 계산하여 *numOfLine*에 저장한다.
Exceptional Courses of Events	N/A

Type	Method
Name	deleteAnnotation()
Purpose	분석 기능에 불필요한 주석 처리된 부분을 삭제하는 메소드
Overview (Class)	'//'으로 주석 처리된 부분을 삭제한다.
Cross Reference	Analyze File
Input (Method)	N/A
Output (Method)	void
Abstract Operation (Method)	ArrayList *source*에서 '//'으로 주석 처리된 부분을 삭제한다.
Exceptional Courses of Events	N/A

Type	Method
Name	divideCodeLine()
Purpose	한 줄 안에 코드가 여러 개인 경우 여러 줄로 분할하는 메소드
Overview (Class)	한 줄 안에 코드가 여러 개인 경우 여러 줄로 분할한다.
Cross Reference	Analyze File
Input (Method)	N/A
Output (Method)	void
Abstract Operation (Method)	ArrayList *source*를 ';'를 기준으로 여러 개로 분할하여 추가한다.
Exceptional Courses of Events	N/A

Type	Method
Name	deletePrintf()
Purpose	분석 기능에 불필요한 printf문의 괄호 안의 내용들을 삭제하는 메소드
Overview (Class)	printf문의 괄호 안의 내용들을 삭제한다.
Cross Reference	Analyze File
Input (Method)	N/A
Output (Method)	void
Abstract Operation (Method)	ArrayList *source*에서 printf문의 괄호 안의 내용들을 삭제한다.
Exceptional Courses of Events	N/A

Type	Method
Name	findType()
Purpose	소스 코드 내 새로 선언된 타입을 찾는 메소드
Overview (Class)	구조체 또는 typedef로 새로 선언된 타입을 찾는다.
Cross Reference	Analyze File
Input (Method)	N/A
Output (Method)	void
Abstract Operation (Method)	ArrayList *source*에서 'typedef' 혹은 'struct' 단어 발견 시 선언 된 식별자 혹은 'struct'+태그명 부분을 *Analyze:listType*에 저장한다.
Exceptional Courses of Events	N/A

Type	Method
Name	analyzeFunction()
Purpose	코드의 함수 수, 이름을 분석하는 메소드
Overview (Class)	코드의 함수 수, 이름을 분석한다.
Cross Reference	Analyze File
Input (Method)	N/A
Output (Method)	void
Abstract Operation (Method)	코드의 함수 수, 이름을 분석하여 *numOfFunction*과 *listFunction*에 저장한다.
Exceptional Courses of Events	N/A

Type	Method
Name	analyzeVariable()
Purpose	코드의 변수 수, 이름을 분석하는 메소드
Overview (Class)	코드의 변수 수, 이름을 분석한다.
Cross Reference	Analyze File
Input (Method)	N/A
Output (Method)	void
Abstract Operation (Method)	코드의 변수 수, 이름을 분석하여 *numOfVariable*, *listVariable*에 저장한다.
Exceptional Courses of Events	N/A

Type	Method
Name	analyzePreprocessor()
Purpose	코드의 전처리기 수, 이름을 분석하는 메소드
Overview (Class)	코드의 전처리기 수, 이름을 분석한다.
Cross Reference	Analyze File
Input (Method)	N/A
Output (Method)	void
Abstract Operation (Method)	코드의 전처리기 수, 이름을 분석하여 *numOfPreprocessor*, *listPreprocessor*에 저장한다.
Exceptional Courses of Events	N/A

Type	Class
Name	Calculate
Purpose	분석된 값을 비교하여 일치율을 계산하는 클래스
Overview (Class)	분석된 값을 비교하여 일치율을 계산한다.
Cross Reference	Calculate File, Calculate Line Sync-Rate, Calculate Function Sync-Rate, Calculate Variable Sync-Rate, Calculate Preprocessor Sync-Rate, Calculate Annotation Sync-Rate
Input (Method)	-
Output (Method)	-
Abstract Operation (Method)	-
Exceptional Courses of Events	N/A

Type	Method
Name	calFile()
Purpose	분석된 값을 바탕으로 전반적인 비교 기능을 수행하는 메소드
Overview (Class)	분석된 값을 바탕으로 전반적인 비교 기능을 수행한다.
Cross Reference	Calculate Line Sync-Rate, Calculate Function Sync-Rate, Calculate Variable Sync-Rate, Calculate Annotation Sync-Rate
Input (Method)	N/A
Output (Method)	void
Abstract Operation (Method)	Analyze 클래스에서 분석되어 저장된 *Files*의 객체의 값들을 바탕으로 비교를 하는 메소드들을 호출한다.
Exceptional Courses of Events	N/A

Type	Method
Name	calLine()
Purpose	두 파일 간 라인 수를 비교, 일치율을 계산하는 메소드
Overview (Class)	두 파일 간 라인 수를 비교, 일치율을 계산한다.
Cross Reference	Calculate Sync-Rate
Input (Method)	N/A
Output (Method)	void
Abstract Operation (Method)	*Files* 객체의 *numOfLine*을 비교하여 일치율을 계산한다.
Exceptional Courses of Events	N/A

Type	Method
Name	calFunction
Purpose	두 파일 간 함수를 비교, 일치율을 계산하는 메소드
Overview (Class)	두 파일 간 함수를 비교, 일치율을 계산한다.
Cross Reference	Calculate Sync-Rate
Input (Method)	N/A
Output (Method)	void
Abstract Operation (Method)	*Files* 객체의 *numOfFunction*과 *listFunction*을 비교하여 일치율을 계산한다.
Exceptional Courses of Events	N/A

Type	Method
Name	calVariable()
Purpose	두 파일 간 변수를 비교, 일치율을 계산하는 메소드
Overview (Class)	두 파일 간 변수를 비교, 일치율을 계산한다.
Cross Reference	Calculate Sync-Rate
Input (Method)	N/A
Output (Method)	void
Abstract Operation (Method)	*Files* 객체의 *numOfVariable*과 *listVariable*을 비교하여 일치율을 계산한다.
Exceptional Courses of Events	N/A

Type	Method
Name	calPreprocesor()
Purpose	두 파일 간 전처리기를 비교, 일치율을 계산하는 메소드
Overview (Class)	두 파일 간 전처리기를 비교, 일치율을 계산한다.
Cross Reference	Calculate Sync-Rate
Input (Method)	N/A
Output (Method)	void
Abstract Operation (Method)	*Files* 객체의 *numOfPreprocessor*, *listPreprocessor*을 비교하여 일치율을 계산한다.
Exceptional Courses of Events	N/A

Type	Method
Name	calAnnotation()
Purpose	두 파일 간 주석 수를 비교, 일치율을 계산하는 메소드
Overview (Class)	두 파일 간 주석 수를 비교, 일치율을 계산한다.
Cross Reference	Calculate Sync-Rate
Input (Method)	N/A
Output (Method)	void
Abstract Operation (Method)	*Files* 객체의 *numOfAnnotation*을 비교하여 일치율을 계산한다.
Exceptional Courses of Events	N/A

Type	Method
Name	setNumCenter()
Purpose	선택된 기준 파일의 번호를 저장하는 메소드
Overview (Class)	선택된 기준 파일의 번호를 저장한다.
Cross Reference	Change Center
Input (Method)	int *pointerNum*
Output (Method)	void
Abstract Operation (Method)	매개변수 *pointerNum*을 *Calculate.centerNum*에 저장
Exceptional Courses of Events	N/A

Type	Method
Name	getTotalSync()
Purpose	총 일치율을 반환하는 메소드
Overview (Class)	총 일치율을 반환한다.
Cross Reference	Change Center
Input (Method)	int *pointerNum*
Output (Method)	void
Abstract Operation (Method)	*SyncRate[pointerNum][5]* (총 일치율)을 반환한다.
Exceptional Courses of Events	N/A

Type	Method
Name	getLineSync()
Purpose	라인 수 일치율을 반환하는 메소드
Overview (Class)	라인 수 일치율을 반환한다.
Cross Reference	Change Center
Input (Method)	int *pointerNum*
Output (Method)	void
Abstract Operation (Method)	*SyncRate[pointerNum][0]* (라인 수 일치율)을 반환한다.
Exceptional Courses of Events	N/A

Type	Method
Name	getFunctionSync()
Purpose	함수 일치율을 반환하는 메소드
Overview (Class)	함수 일치율을 반환한다.
Cross Reference	Change Center
Input (Method)	int *pointerNum*
Output (Method)	void
Abstract Operation (Method)	*SyncRate[pointerNum][1]* (함수 일치율)을 반환한다.
Exceptional Courses of Events	N/A

Type	Method
Name	getVariableSync()
Purpose	변수 일치율을 반환하는 메소드
Overview (Class)	변수 일치율을 반환한다.
Cross Reference	Change Center
Input (Method)	int *pointerNum*
Output (Method)	void
Abstract Operation (Method)	*SyncRate[pointerNum][2]* (변수 일치율)을 반환한다.
Exceptional Courses of Events	N/A

Type	Method
Name	getPreprocessorSync()
Purpose	전처리기 일치율을 반환하는 메소드
Overview (Class)	전처리기 일치율을 반환한다.
Cross Reference	Change Center
Input (Method)	int *pointerNum*
Output (Method)	void
Abstract Operation (Method)	*SyncRate[pointerNum][3]* (전처리기 일치율)을 반환한다.
Exceptional Courses of Events	N/A

Type	Method
Name	getAnnotationSync()
Purpose	주석 수 일치율을 반환하는 메소드
Overview (Class)	주석 수 일치율을 반환한다.
Cross Reference	Change Center
Input (Method)	int *pointerNum*
Output (Method)	void
Abstract Operation (Method)	*SyncRate[pointerNum][4]* (주석 수 일치율)을 반환한다.
Exceptional Courses of Events	N/A

Type	Class
Name	Files
Purpose	각각의 파일들의 소스 코드 및 분석된 값을 저장하는 클래스
Overview (Class)	소스코드 및 분석된 값을 저장하고 반환한다.
Cross Reference	Calculate File
Input (Method)	-
Output (Method)	-
Abstract Operation (Method)	-
Exceptional Courses of Events	N/A

Type	Method
Name	getName()
Purpose	파일 이름을 반환하는 메소드
Overview (Class)	파일 이름을 반환한다.
Cross Reference	Change Center
Input (Method)	N/A
Output (Method)	String *name*
Abstract Operation (Method)	String *name* (파일 이름)을 반환한다.
Exceptional Courses of Events	N/A

Type	Method
Name	getNumOfLine()
Purpose	라인 수를 반환하는 메소드
Overview (Class)	라인 수를 반환한다.
Cross Reference	Change Center
Input (Method)	N/A
Output (Method)	int *numOfLine*
Abstract Operation (Method)	int *numOfLine* (라인 수)를 반환한다.
Exceptional Courses of Events	N/A

Type	Method
Name	getNumOfFunction()
Purpose	함수 수를 반환하는 메소드
Overview (Class)	함수 수를 반환한다.
Cross Reference	Change Center
Input (Method)	N/A
Output (Method)	int *numOfFunction*
Abstract Operation (Method)	int *numOfFunction* (함수 수)를 반환한다.
Exceptional Courses of Events	N/A

Type	Method
Name	getNumOfVariable()
Purpose	변수 수를 반환하는 메소드
Overview (Class)	변수 수를 반환한다.
Cross Reference	Change Center
Input (Method)	N/A
Output (Method)	int *numOfVariable*
Abstract Operation (Method)	int *numOfVariable* (변수 수)를 반환한다.
Exceptional Courses of Events	N/A

Type	Method
Name	getNumOfPreprocessor()
Purpose	전처리기 수를 반환하는 메소드
Overview (Class)	전처리기 수를 반환한다.
Cross Reference	Change Center
Input (Method)	N/A
Output (Method)	int *numOfPreprocessor*
Abstract Operation (Method)	int *numOfPreprocessor* (전처리기 수)를 반환한다.
Exceptional Courses of Events	N/A

Type	Method
Name	getNumOfAnnotation()
Purpose	주석 수를 반환하는 메소드
Overview (Class)	주석 수를 반환한다.
Cross Reference	Change Center
Input (Method)	N/A
Output (Method)	int *numOfAnnotation*
Abstract Operation (Method)	int *numOfAnnotation* (주석 수)를 반환한다.
Exceptional Courses of Events	N/A

Type	Method
Name	getListFunction()
Purpose	함수 이름이 저장된 ArrayList를 반환하는 메소드
Overview (Class)	함수 이름이 저장된 ArrayList를 반환한다.
Cross Reference	Change Center
Input (Method)	N/A
Output (Method)	ArrayList *listFunction*
Abstract Operation (Method)	ArrayList *listFunction* (함수 이름이 저장된 ArrayList)를 반환한다.
Exceptional Courses of Events	N/A

Type	Method
Name	getListVariable()
Purpose	변수 이름이 저장된 ArrayList를 반환하는 메소드
Overview (Class)	변수 이름이 저장된 ArrayList를 반환한다.
Cross Reference	Change Center
Input (Method)	N/A
Output (Method)	ArrayList *listVariable*
Abstract Operation (Method)	ArrayList *listVariable* (변수 이름이 저장된 ArrayList)를 반환한다.
Exceptional Courses of Events	N/A

Type	Method
Name	getListPreprocessor()
Purpose	전처리기 이름이 저장된 ArrayList를 반환하는 메소드
Overview (Class)	전처리기 이름이 저장된 ArrayList를 반환한다.
Cross Reference	Change Center
Input (Method)	N/A
Output (Method)	ArrayList *listPreprocessor*
Abstract Operation (Method)	ArrayList *listPreprocessor* (전처리기 이름이 저장된 ArrayList)를 반환한다.
Exceptional Courses of Events	N/A

Type	Method
Name	setName()
Purpose	파일 이름을 저장하는 메소드
Overview (Class)	파일 이름을 저장한다.
Cross Reference	Change Center
Input (Method)	String *tempName*
Output (Method)	void
Abstract Operation (Method)	파일 이름을 *name*에 저장한다.
Exceptional Courses of Events	N/A

Type	Method
Name	setNumOfLine()
Purpose	라인 수를 저장하는 메소드
Overview (Class)	라인 수를 저장한다.
Cross Reference	Change Center
Input (Method)	int *tempNumOfLine*
Output (Method)	void
Abstract Operation (Method)	라인 수를 *numOfLine*에 저장한다.
Exceptional Courses of Events	N/A

Type	Method
Name	setNumOfFunction()
Purpose	함수 수를 저장하는 메소드
Overview (Class)	함수 수를 저장한다.
Cross Reference	Change Center
Input (Method)	int *tempNumOfFunction*
Output (Method)	void
Abstract Operation (Method)	함수 수를 *numOfFunction*에 저장한다.
Exceptional Courses of Events	N/A

Type	Method
Name	setNumOfVariable()
Purpose	변수 수를 저장하는 메소드
Overview (Class)	변수 수를 저장한다.
Cross Reference	Change Center
Input (Method)	int *tempNumOfVariable*
Output (Method)	void
Abstract Operation (Method)	변수 수를 *numOfVariable*에 저장
Exceptional Courses of Events	N/A

Type	Method
Name	setNumOfPreprocessor()
Purpose	전처리기 수를 저장하는 메소드
Overview (Class)	전처리기 수를 저장한다.
Cross Reference	Change Center
Input (Method)	int *tempNumOfPreprocessor*
Output (Method)	void
Abstract Operation (Method)	전처리기 수를 *numOfPreprocessor*에 저장한다.
Exceptional Courses of Events	N/A

Type	Method
Name	setNumOfAnnotation()
Purpose	주석 수를 저장하는 메소드
Overview (Class)	주석 수를 저장한다.
Cross Reference	Change Center
Input (Method)	int *tempNumOfAnnotation*
Output (Method)	void
Abstract Operation (Method)	주석 수를 *numOfAnnotation*에 저장한다.
Exceptional Courses of Events	N/A

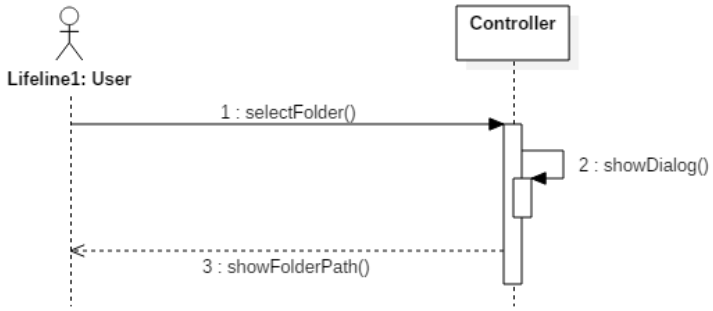
Type	Method
Name	setListFunction()
Purpose	함수 이름을 저장하는 메소드
Overview (Class)	함수 이름을 저장한다.
Cross Reference	Change Center
Input (Method)	ArrayList *tempFunctionName*
Output (Method)	void
Abstract Operation (Method)	함수 이름을 ArrayList *listFunction*에 저장한다.
Exceptional Courses of Events	N/A

Type	Method
Name	setListVariable()
Purpose	변수 이름을 저장하는 메소드
Overview (Class)	변수 이름을 저장한다.
Cross Reference	Change Center
Input (Method)	ArrayList *tempVariableName*
Output (Method)	void
Abstract Operation (Method)	변수 이름을 ArrayList *listVariable*에 저장한다.
Exceptional Courses of Events	N/A

Type	Method
Name	setListPreprocessor()
Purpose	전처리기 이름을 저장하는 메소드
Overview (Class)	전처리기 이름을 저장한다.
Cross Reference	Change Center
Input (Method)	ArrayList *tempPreprocessorName*
Output (Method)	void
Abstract Operation (Method)	전처리기 이름을 *listPreprocessor*에 저장한다.
Exceptional Courses of Events	N/A

Activity 2152. Implement Windows

1. Select Folder

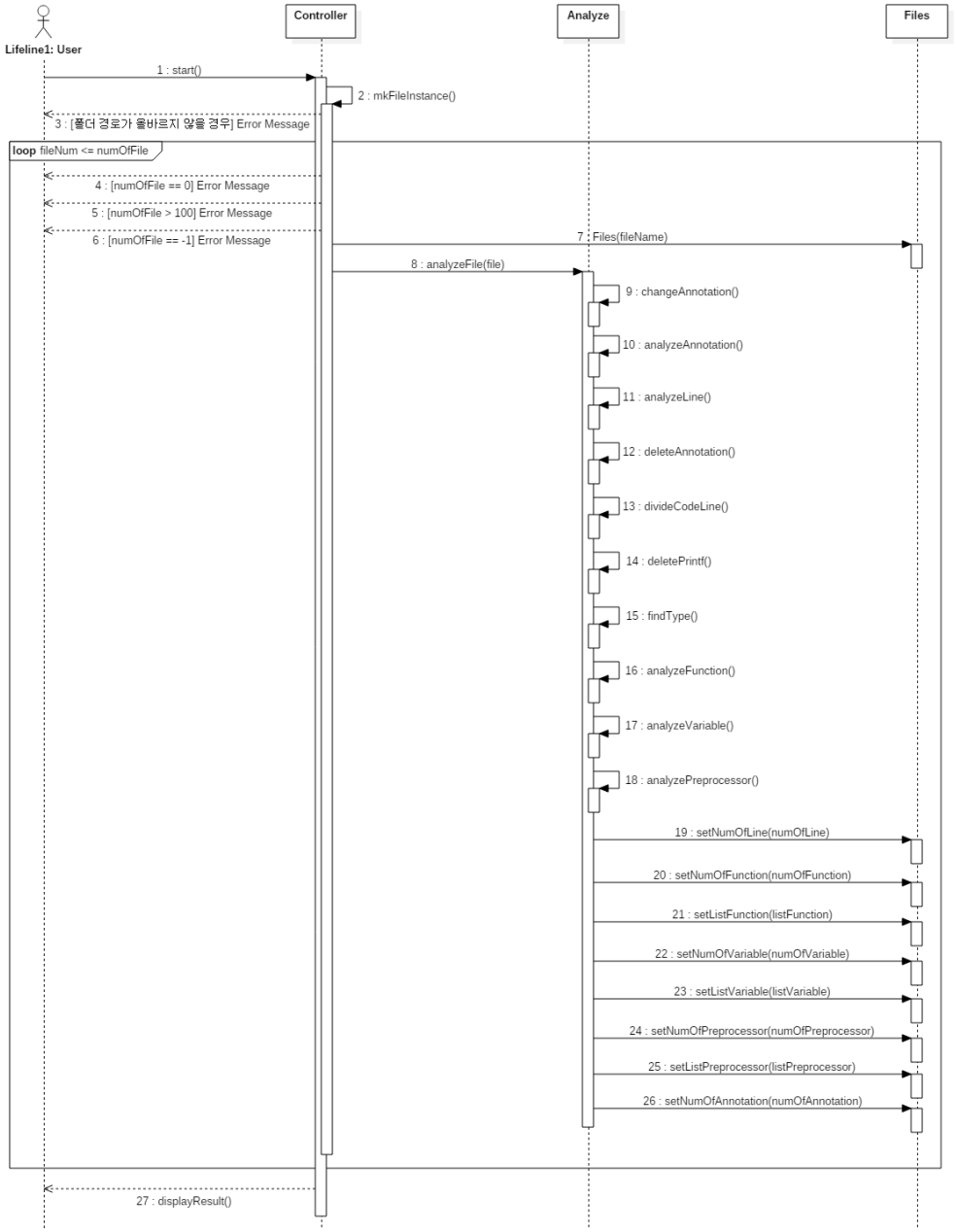


Name	selectFolder
Responsibilities	폴더 선택 버튼을 클릭한다.
Type	GUI
Cross Reference	Select Folder
Notes	폴더를 지정하는 창을 보여주는 *showDialog()* 메소드를 호출한다.
Pre-Conditions	N/A
Post-Conditions	*showDialog()*가 실행된다.

Name	showDialog()
Responsibilities	폴더 경로를 지정한다.
Type	GUI
Cross Reference	Select Folder
Notes	폴더를 지정하는 창이 출력된다.
Pre-Conditions	폴더 선택 버튼을 누른다.
Post-Conditions	메인 화면에 지정된 폴더 경로를 출력한다.

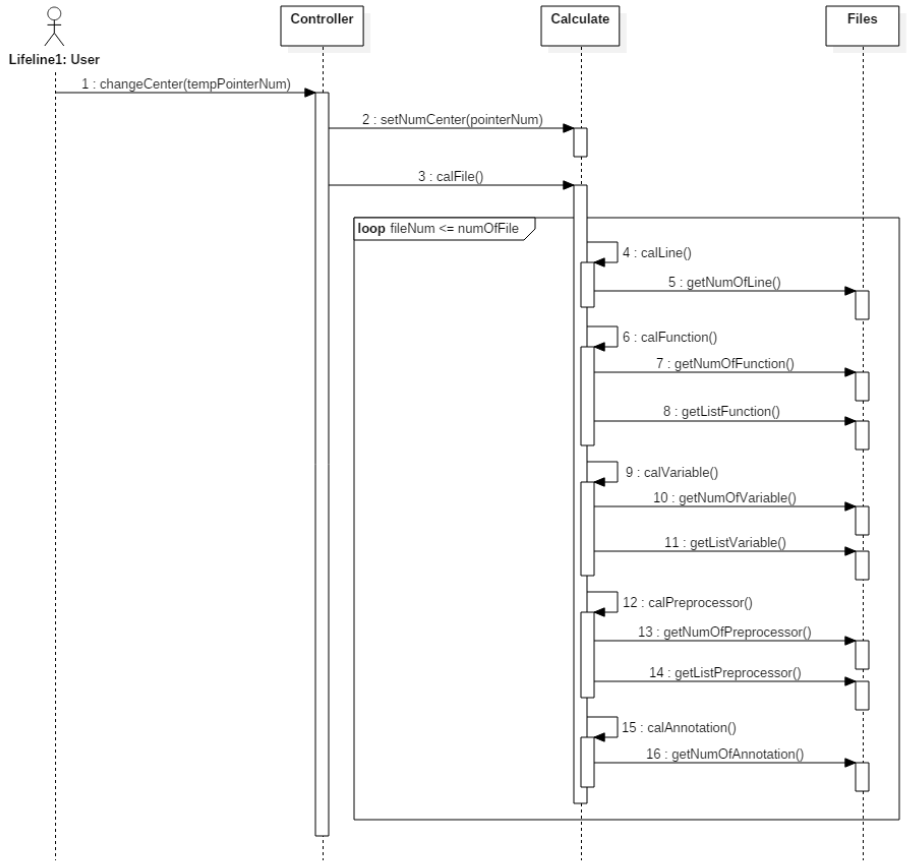
Name	showFolderPath()
Responsibilities	사용자가 폴더를 지정한 후 메인 화면으로 돌아온다.
Type	GUI
Cross Reference	Select Folder
Notes	메인 화면에 지정된 폴더 경로가 출력된다.
Pre-Conditions	폴더 선택을 완료한다.
Post-Conditions	메인 화면에 지정된 폴더 경로가 출력된다.

2. Start



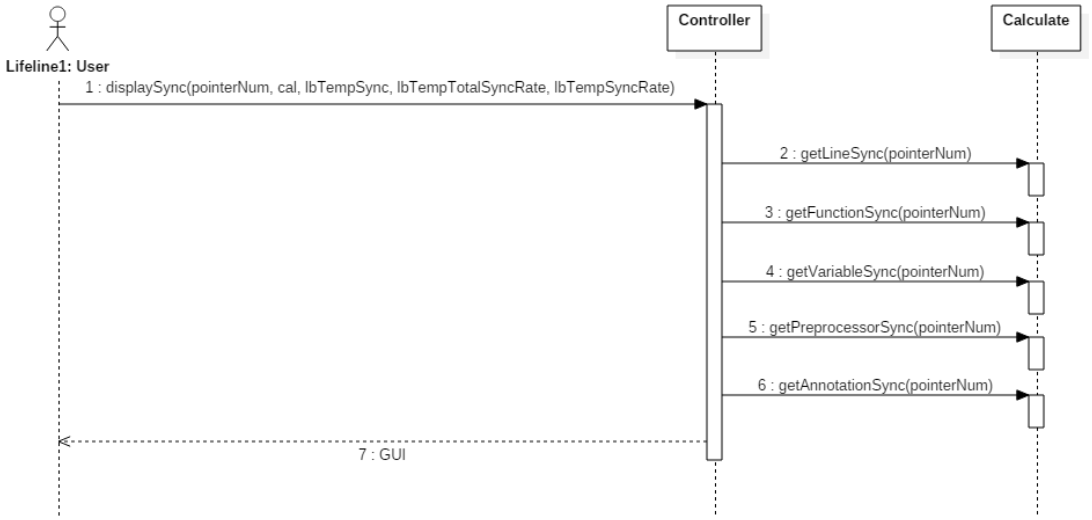
Name	displayResult()
Responsibilities	시작하기 버튼을 누른다.
Type	GUI
Cross Reference	Start
Notes	분석 결과 창을 출력한다.
Pre-Conditions	폴더 경로 지정과 파일 분석을 완료한다.
Post-Conditions	분석 결과 창을 출력한다.

3. Change Center



Name	changeCenter()
Responsibilities	결과 화면의 클라우드 또는 오른쪽 리스트에서 파일을 클릭한다.
Type	GUI
Cross Reference	Change Center
Notes	클릭한 파일을 기준으로 비교 결과 화면을 새로 출력한다.
Pre-Conditions	파일 분석이 끝나고 결과 화면이 나타난다.
Post-Conditions	클릭한 파일을 기준으로 비교 결과 화면을 새로 출력한다.

4. Display Sync



Name	displaySync()
Responsibilities	클라우드 내 소스 코드 파일 이름 위에 마우스를 오버한다.
Type	GUI
Cross Reference	Display Sync
Notes	기준 파일과 마우스 오버된 파일의 상세 일치율 정보를 출력한다.
Pre-Conditions	기준 파일이 선택되어 비교 결과가 나타난다.
Post-Conditions	기준 파일과 마우스 오버된 파일의 상세 일치율 정보를 출력한다.

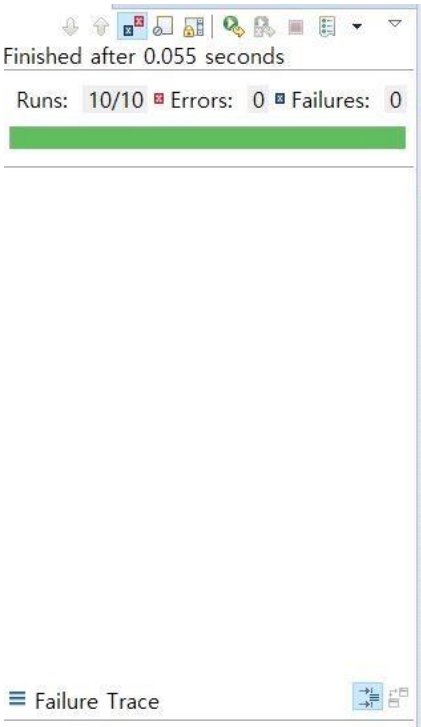
Activity 2153. Implement Reports

OOPT Stage 1000, 2030, 2040, 2050 보고서를 통하여 작성 완료

Activity 2154. Implement DB Schema

해당 사항 없음

Activity 2155. Write Unit Test Code




```java
import static org.junit.Assert.*;

public class TestAnalyze {
    ArrayList<String> aa = new ArrayList();
    Analyze a;

    @Test
    public void testAnalyzeLine() {
        a = new Analyze(aa);
        aa.add("//Hello world");
        aa.add("#include <stdio.h>");
        aa.add("#define Max 10");
        aa.add("typedef struct{");
        aa.add("int a, double b");
        aa.add("}myst;");
        aa.add("/*함수원형정의*/");
        aa.add("void f(void);");
        aa.add("int main(void)");
        aa.add("{");
        aa.add("f();");
        aa.add("f();");
        aa.add("f();");
        aa.add("return 0;");
        aa.add("}");
        aa.add("void f(void)");
        aa.add("{");
        aa.add("static int x = 0;");
        aa.add("int y = 0;");
        aa.add("x++;");
        aa.add("y++;");
        aa.add("printf('x = %d, y = %d\n', x, y);");
        aa.add("}");

        assertEquals(23, a.analyzeLine());
    }
```

```
@Test
public void testAnalyzeFunction() {
    a = new Analyze(aa);

    aa.add("void f(void);");
    aa.add("char g( );");
    aa.add("long l( );");
    aa.add("double d( );");

    aa.add("int h(void);");


    aa.add("int main(void)");
    aa.add("{");
    aa.add("f();");
    aa.add("g(2);");
    aa.add("h();");
    aa.add("return 0;");
    aa.add("}");

    aa.add("void f(void)");
    aa.add("{");
    aa.add("printf('1번 함수입니다.');");
    aa.add("}");

    aa.add("char g( )");
    aa.add("{");
    aa.add("printf('2번 (%d 번) 함수입니다.', g);");
    aa.add("return 'dd' ;}");

    aa.add("long l()");
    aa.add("{");
    aa.add("printf('3 번 함수입니다.');");
    aa.add("return 0 ;}");

    aa.add("double d()");
    aa.add("{");
    aa.add("printf('4 번 함수입니다.');");
    aa.add("return 0 ;}");

    assertEquals(5, a.analyzeFunction());
}
assertEquals("int main(void)", a.analyzeFunction().get(0)); //사용된 함수 이름 test1
assertEquals("void f(void)", a.analyzeFunction().get(1)); //사용된 함수 이름 test2
```

```
@Test
public void testAnalyzeVariable() {
    a = new Analyze(aa);

    aa.add("typedef struct{");
    aa.add("int a, double b");
    aa.add("}myst;");

    aa.add("boolean home = false");
    aa.add("char sentence;");
    aa.add("short z;");
    aa.add("long u;");
    aa.add("short k;");
    aa.add("short z;");
    aa.add("float f;");
    aa.add("static int x = 0;");
    aa.add("int y = 0;");

    /* 단순한 '=' 은 포함이 안된다. */
    aa.add("=");

    assertEquals(10, a.analyzeVariable());
}
```

```
assertEquals("int a", a.analyzeVariable().get(0));  // 사용된 변수 이름 test1
assertEquals("double b", a.analyzeVariable().get(1)); // 사용된 변수 이름 test2
```

```
@Test
public void testAnalyzePreprocessor() {
    a = new Analyze(aa);
    aa.add("//Hello world");
    aa.add("#include <stdio.h>");
    aa.add("#define Max 10");
    aa.add("#include <stdlib.h>");
    aa.add("printf('#');");
    aa.add("//#include <stdlib.h>");
    assertEquals(3, a.analyzePreprocessor());
}
```

```
assertEquals("#include <stdio.h>", a.analyzePreprocessor().get(0)); // 사용한 전처리기 1
assertEquals("#define Max 10", a.analyzePreprocessor().get(1));// 사용한 전처리기 2
```

```
@Test
public void testAnalyzeAnnotation() {
    a = new Analyze(aa);
    aa.add("//Hello world");
    aa.add("//처음만드는 프로그램 입니다.");
    aa.add("/*가위 바위 보 게임 - 이 줄이 포함이 안된다.");
    aa.add("컴퓨터와의 게임입니다. - 이건 포함이 된다. */");
    aa.add("// 이것도 포함이 될까 ?*/");


    assertEquals(4, a.analyzeAnnotation());
}
```

```
@Test
public void testDivideCodeLine() {
    a = new Analyze(aa);
    /*for 문이 아니고, 다른 ; 로 나뉜 코드는 여러줄로 바뀐다.*/
    aa.add("f();f();f();");

    /* for 문은 나뉘지 않는다. */
    aa.add("for(i=0;i<10;i++)");

    a.divideCodeLine();
    assertEquals(4, a.analyzeLine());
}
```

```
@Test
public void testChangeAnnotation() {
    a = new Analyze(aa);
    aa.add("//Hello world");
    aa.add("//처음만드는 프로그램 입니다.");
    aa.add("/*가위 바위 보 게임");
    aa.add("컴퓨터와의 게임입니다. */");
    aa.add("// 이것도 포함이 될까 ? */");

    a.changeAnnotation();

    assertEquals(5, a.analyzeAnnotation());
}
```

```
@Test
public void testDeletePrintf() {
    a = new Analyze(aa);
    aa.add("printf('asdasd');");
    aa.add("//include <stdio.h>");
    aa.add("/* ddddd ");
    aa.add("ddddd */");

    a.deletePrintf();
    assertEquals("printf();", a.source.get(0));
}

@Test
public void testDeleteAnnotation() {
    a = new Analyze(aa);
    aa.add("//Hello world");
    aa.add("//처음만드는 프로그램 입니다.");
    aa.add("/*가위 바위 보 게임");
    aa.add("컴퓨터와의 게임입니다. */");
    aa.add("// 이것도 포함이 될까 ? */");

    a.deleteAnnotation();

    assertEquals(0, a.analyzeAnnotation());
}
@Test
public void testFindType() {
    a = new Analyze(aa);
    aa.add("typedef struct");
    aa.add("{");
    aa.add("int a, double b");
    aa.add("}myst;");

    aa.add("typedef int jungsu;");

    aa.add("struct");
    aa.add("{");
    aa.add("int x, double y");
    aa.add("}point;");

    assertEquals("myst", a.findType().get(8));
    assertEquals("struct", a.findType().get(9));

}
```

```
@Test
public void testCalLine() {

    f1.setNumOfLine(100);
    f2.setNumOfLine(90);

    files.add(f1);
    files.add(f2);

    c = new Calculate(files);

    assertEquals(90, c.calLine());
}
```

```
@Test   /*비교파일이 기존 파일보다 라인수가 많을 때 */
public void testCalLine() {
    f1.setNumOfLine(90);
    f2.setNumOfLine(100);

    files.add(f1);
    files.add(f2);

    c = new Calculate(files);

    assertEquals(90, c.calLine());
}
```

```
@Test   /*라인수가 아예 같은때 */
public void testCalLine() {

    f1.setNumOfLine(100);
    f2.setNumOfLine(100);

    files.add(f1);
    files.add(f2);

    c = new Calculate(files);

    assertEquals(100, c.calLine());
}
```

```
@Test   /*라인이 둘 다 0 일때 */
public void testCalLine2() {
    f1.setNumOfLine(0);
    f2.setNumOfLine(0);

    files.add(f1);
    files.add(f2);

    c = new Calculate(files);

    assertEquals(0, c.calLine());
}
```

```
 @Test
 public void testCalFunction() {
     a1.add("main(void)");
     a1.add("void f()");
     a1.add("void a()");

     b1.add("main(void)");
     b1.add("void f()");

     f1.setListFunction(a1);
     f2.setListFunction(b1);

     f1.setNumOfFunction(3);
     f2.setNumOfFunction(2);

     files.add(f1);
     files.add(f2);

     c = new Calculate(files);

     assertEquals(66, c.calFunction());
 }
```

```
@Test
public void testCalFunction() {

    a1.add("main(void)");
    a1.add("int c();");

    b1.add("main(void)");
    b1.add("int k();");


    f1.setListFunction(a1);
    f2.setListFunction(b1);

    f1.setNumOfFunction(2);
    f2.setNumOfFunction(2);

    files.add(f1);
    files.add(f2);

    c = new Calculate(files);

    assertEquals(60, c.calFunction());
}
```

```
@Test
public void testCalFunction() {

    f1.setListFunction(a1);
    f2.setListFunction(b1);

    files.add(f1);
    files.add(f2);

    c = new Calculate(files);

    assertEquals(0, c.calFunction());
}

@Test  // 비교파일의 함수가 더 많을때
public void testCalFunction2() {
    a1.add("int c();");
    a1.add("int d();");

    b1.add("int a();");
    b1.add("int d();");
    b1.add("int e();");


    f1.setListFunction(a1);
    f2.setListFunction(b1);

    f1.setNumOfFunction(1);
    f2.setNumOfFunction(3);

    files.add(f1);
    files.add(f2);

    c = new Calculate(files);

    assertEquals(33, c.calFunction());
}
```

```
@Test
public void testCalVariable() {

    a1.add("int a");
    a1.add("int b");
    a1.add("int c");

    b1.add("int a");
    b1.add("int b");

    f1.setListVariable(a1);
    f2.setListVariable(b1);

    f1.setNumOfVariable(3);
    f2.setNumOfVariable(2);

    files.add(f1);
    files.add(f2);

    c = new Calculate(files);

    assertEquals(66, c.calVariable());
}
```

```
@Test
public void testCalVariable() {
    a1.add("int k");
    a1.add("int j");
    a1.add("int d");

    b1.add("int a");
    b1.add("int b");

    f1.setListVariable(a1);
    f2.setListVariable(b1);

    f1.setNumOfVariable(3);
    f2.setNumOfVariable(2);

    files.add(f1);
    files.add(f2);

    c = new Calculate(files);

    assertEquals(26, c.calVariable());
}
```

```
@Test
public void testCalVariable() {


    f1.setListVariable(a1);
    f2.setListVariable(b1);


    files.add(f1);
    files.add(f2);

    c = new Calculate(files);

    assertEquals(0, c.calVariable());

}

@Test
public void testCalVariable2() {
    a1.add("int k");
    a1.add("int j");

    b1.add("int d");
    b1.add("int a");
    b1.add("int b");

    f1.setListVariable(a1);
    f2.setListVariable(b1);

    f1.setNumOfVariable(2);
    f2.setNumOfVariable(3);

    files.add(f1);
    files.add(f2);

    c = new Calculate(files);

    assertEquals(26, c.calVariable());

}
```

```java
@Test
public void testCalPreprocessor() {

    a1.add("#include<stdio.h>");
    a1.add("#define MAX 10");

    b1.add("#include<stdio.h>");

    f1.setListPreprocessor(a1);
    f2.setListPreprocessor(b1);

    f1.setNumOfPreprocessor(2);
    f2.setNumOfPreprocessor(1);

    files.add(f1);
    files.add(f2);

    c = new Calculate(files);

    assertEquals(50, c.calPreprocessor());
}
```

```java
@Test
public void testCalPreprocessor() {

    f1.setListPreprocessor(a1);
    f2.setListPreprocessor(b1);

    f1.setNumOfPreprocessor(0);
    f2.setNumOfPreprocessor(0);

    files.add(f1);
    files.add(f2);

    c = new Calculate(files);

    assertEquals(0, c.calPreprocessor());
}
```

```
@Test
public void testCalPreprocessor() {
    a1.add("#include <Turboc.h>");

    b1.add("#define MAX 10");
    b1.add("#define MINI 5");

    f1.setListPreprocessor(a1);
    f2.setListPreprocessor(b1);

    f1.setNumOfPreprocessor(1);
    f2.setNumOfPreprocessor(2);

    files.add(f1);
    files.add(f2);

    c = new Calculate(files);

    assertEquals(20, c.calPreprocessor());
}
```

```
@Test
public void testCalAnnotation() {

    f1.setNumOfAnnotation(10);
    f2.setNumOfAnnotation(5);

    files.add(f1);
    files.add(f2);

    c = new Calculate(files);

    assertEquals(50, c.calAnnotation());
}
```

```java
@Test
public void testCalAnnotation() {

    f1.setNumOfAnnotation(0);
    f2.setNumOfAnnotation(0);

    files.add(f1);
    files.add(f2);

    c = new Calculate(files);

    assertEquals(0, c.calAnnotation());
}
```

```java
@Test
public void testCalAnnotation() {

    f1.setNumOfAnnotation(1);
    f2.setNumOfAnnotation(5);

    files.add(f1);
    files.add(f2);

    c = new Calculate(files);

    assertEquals(20, c.calAnnotation());
}
```