

# GitHub를 통해 개발자간 협업하기

엄민석 / 오세유 YB 2월

# 목차

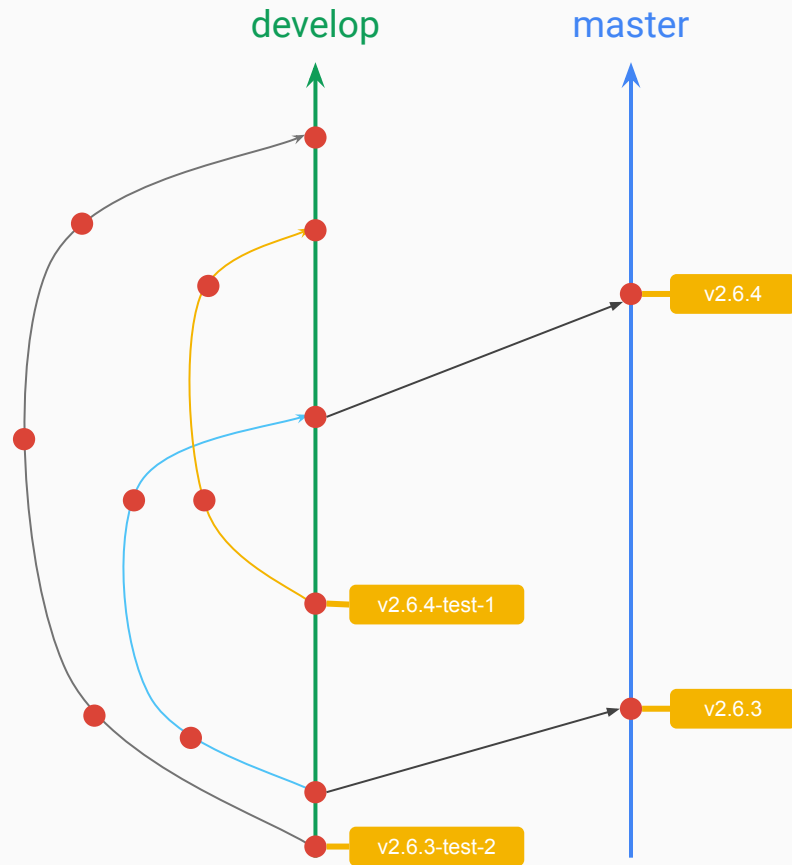
- git branch 전략
- Pull Request
- Continuous Integration
- 코드 스타일 검사
- 테스트 커버리지 측정
- 소스코드 문서화

# git branch 전략


## Git Flow

- master
- develop
- feature
- release
- hotfix

# git branch 전략



# Pull Request


 Pull requests 0

내가 작업한 브랜치를 중앙 브랜치(**develop**)에 합쳐 달라

# Pull Request

## Comparing changes

Choose two branches to see what's changed or to start



base: **develop** ▼

 ← 

compare: **bugfix/77** ▼

중앙 브랜치

개인  
작업 브랜치

Reviewers



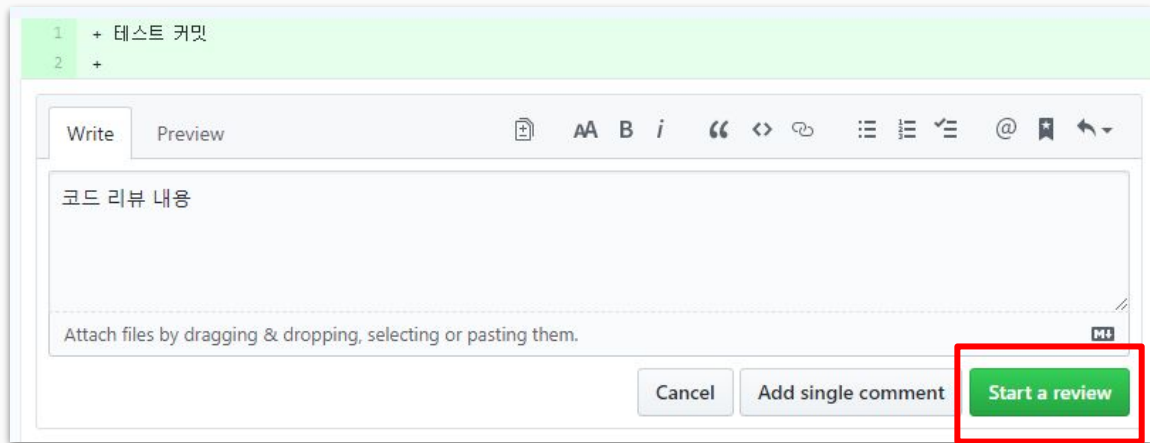
No reviews

Assignees



No one—assign yourself

# Pull Request



The screenshot shows a web interface for a Pull Request. At the top, there's a green header with two items: '1 + 테스트 커밋' and '2 +'. Below this is a comment box with a 'Write' tab selected and a 'Preview' tab. The comment box contains the text '코드 리뷰 내용'. Below the text area is a dashed line with the text 'Attach files by dragging & dropping, selecting or pasting them.' and a small icon. At the bottom of the comment box are three buttons: 'Cancel', 'Add single comment', and 'Start a review'. The 'Start a review' button is highlighted with a red rectangle.

Files changed 탭에서 특정 줄에 코멘트를 달 수 있음

# Pull Request

Write

Preview

AA B i

Leave a comment

Attach files by dragging & dropping, selecting or pasting them.

☒ **Comment**  
Submit general feedback without explicit approval.

☐ **Approve**  
Submit feedback and approve merging these changes.

☐ **Request changes**  
Submit feedback that must be addressed before merging.

Submit review

1 pending comment

## Comment

요구사항을 만족하나, 수정했으면 하는 것이 있을 때

## Approve

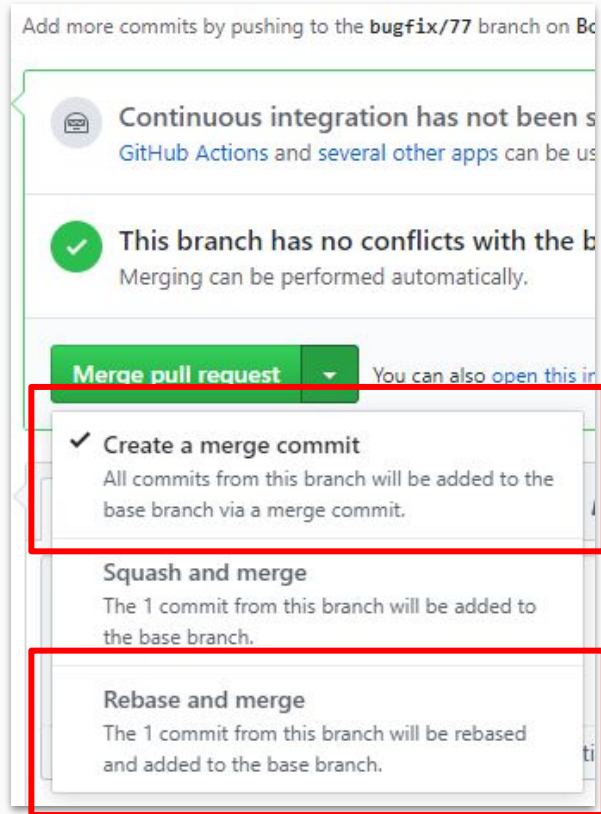
합쳐도 좋음

## Request changes

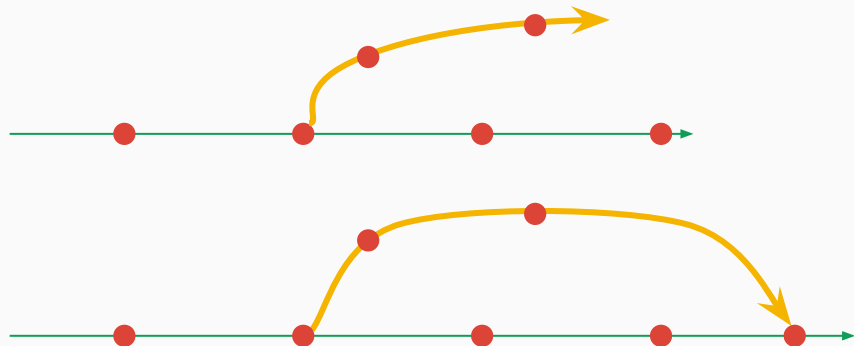
심각한 결함이 있거나 요구사항을 만족하지 않아  
반드시 수정해야 할 때



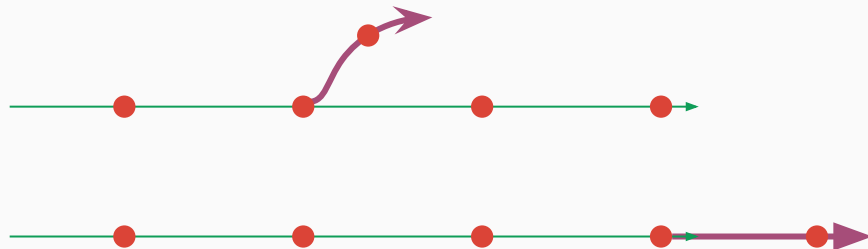
# Pull Request



## Create a merge commit 방식



## Rebase and merge 방식



# Continuous Integration

- 지속적인 통합
- 합쳐나가는 과정에 문제가 있는지 검사



# Jenkins

## 구축 방법

### Dockerfile 스크립트 작성

<https://github.com/jenkinsci/docker/blob/master/Dockerfile> 참조

적절한 위치에 Android SDK를 다운로드 하는  
스크립트 추가

도커 이미지 build & 컨테이너 띄우기

초기 세팅 및 GitHub 연동은 구글링

84 lines (66 sloc) | 3.53 KB

```
1 FROM openjdk:8-jdk-stretch
2
3 # Install git lfs on Debian stretch per https://github.com/git-lfs/git-lfs/wiki
4 # Avoid JENKINS-59569 - git LFS 2.7.1 fails clone with reference repository
5 RUN apt-get update && apt-get upgrade -y && apt-get install -y git curl && c
6
7 ARG user=jenkins
8 ARG group=jenkins
9 ARG uid=1000
10 ARG gid=1000
11 ARG http_port=8080
12 ARG agent_port=50000
13 ARG JENKINS_HOME=/var/jenkins_home
14 ARG REF=/usr/share/jenkins/ref
15
16 ENV JENKINS_HOME $JENKINS_HOME
17 ENV JENKINS_SLAVE_AGENT_PORT ${agent_port}
18 ENV REF $REF
19
20 # Jenkins is run with user `jenkins`, uid = 1000
21 # If you bind mount a volume from the host or a data container,
22 # ensure you use the same uid
23 RUN mkdir -p $JENKINS_HOME \
24     && chown ${uid}:${gid} $JENKINS_HOME \
25     && groupadd -g ${gid} ${group} \
26     && useradd -d "$JENKINS_HOME" -u ${uid} -g ${gid} -m -s /bin/bash ${user}
27
28 # Jenkins home directory is a volume, so configuration and build history
29 # can be persisted and survive image upgrades
30 VOLUME $JENKINS_HOME
```

# Continuous Integration


프로젝트 루트 디렉터리 아래에 **Jenkinsfile** 파일  
생성 후 우측과 같이 작성

git push를 하거나 Pull Request를 올릴 때마다  
젠킨스가 **Jenkinsfile**을 읽어 명령을 실행함

**Jenkinsfile**은 그루비 스크립트

```
pipeline {  
    agent any  
  
    stages {  
        stage('빌드') {  
            agent any  
            steps {  
                sh './gradlew clean assembleDebug'  
            }  
        }  
        stage('테스트') {  
            agent any  
            steps {  
                sh './gradlew testDebugUnitTest'  
            }  
        }  
    }  
}
```

# Continuous Integration


 **Jenkins**


search


admin | log out


Jenkins


ENABLE AUTO REFRESH


 New Item


 People


 Build History

 Manage Jenkins

 My Views

 Credentials

 Lockable Resources

 New View


**Build Queue**

No builds in the queue.











**Build Executor Status**

1 Idle




2 Idle

 add description

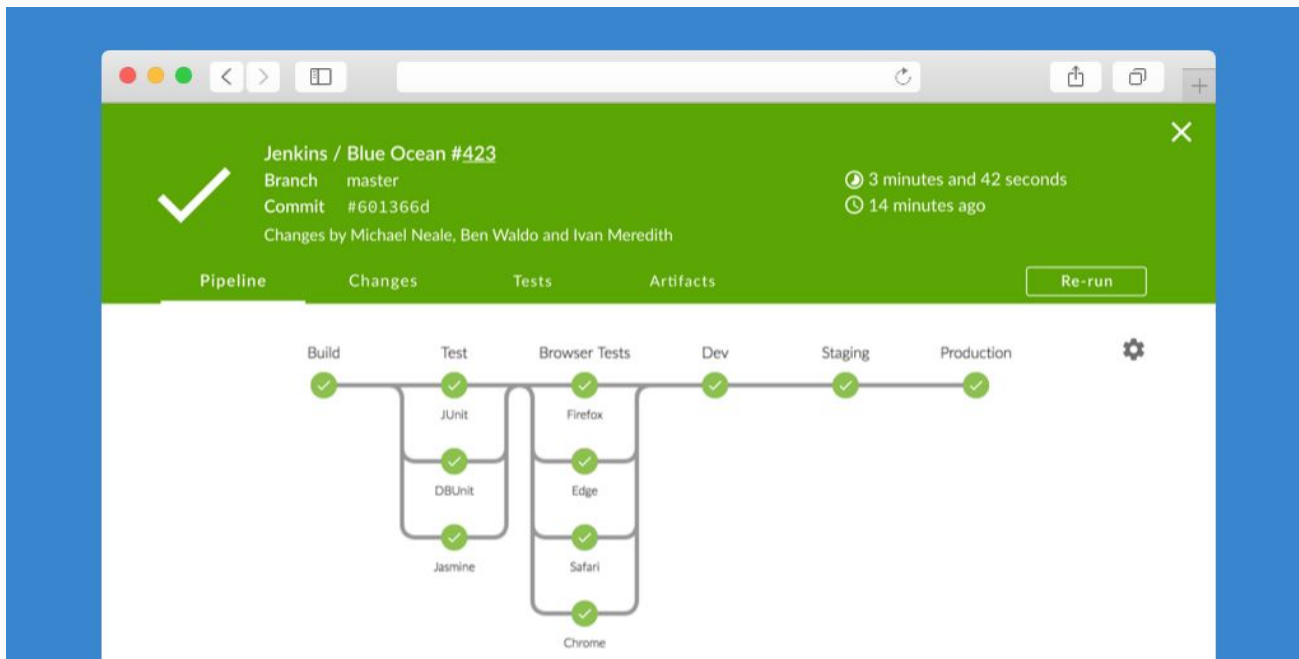
All +

S	W	Name ↓	Last Success	Last Failure	Last Duration	
		<a href="#">Freestyle project</a>	29 sec - <a href="#">#1</a>	N/A	41 ms	
		<a href="#">Github Org project</a>	N/A	20 sec - <a href="#">log</a>	2.9 sec	
		<a href="#">Multibranch</a>	N/A	N/A	N/A	
		<a href="#">Other</a>	N/A	N/A	N/A	

Icon: [S](#) [M](#) [L](#)

[Legend](#)  [RSS for all](#)  [RSS for failures](#)  [RSS for just latest builds](#)

## Blue Ocean 플러그인



# 코드 스타일 검사

- 코드 리뷰 때 마다 코딩 컨벤션을 지적해주는 것은 번거롭고 쉽지 않다.
- 자동화할 수 있을까?



# 코드 스타일 검사

## detekt - <https://github.com/arturbosch/detekt>

코틀린 코드 스타일 검사 도구

gradle 플러그인으로 제공

README.md

### detekt

chat on slack visit website Download 1.5.1 Gradle v1.5.1

build passing build passing codecov 82% codefactor A license scan passing awesome kotlin

Meet *detekt*, a static code analysis tool for the *Kotlin* programming language. It operates on the abstract syntax tree provided by the Kotlin compiler.

```
arg@quarto:0115 ~ % ./gradlew :detekt:cli:buildLibs:detekt:cli:1.0.0-M41:jar -p ./home/artur/Projects/detekt -D:home/artur/Projects/detekt/default-detekt-config.yml -D:detekt:cli:parallel
Analyzing 107 Kotlin files: .....
Ruleset: code-smell
Ruleset: comments
Ruleset: complexity
  TooManyFunctions - 20/28 - /Findings.kt - Line/Column=(1,1) - Path=detekt-api/src/main/kotlin/io/github/arturbosch/detekt/api/Findings.kt
  MethodCallDepth - 4/2 - /Procedural - Line/Column=(12,2) - Path=detekt-api/src/main/kotlin/io/github/arturbosch/detekt/api/Findings.kt
  ComplexityCondition - 5/2 - /Procedural - Line/Column=(21,7) - Path=detekt-api/src/main/kotlin/io/github/arturbosch/detekt/api/Findings.kt
  ComplexityCondition - 7/3 - /Procedural - Line/Column=(32,7) - Path=detekt-api/src/main/kotlin/io/github/arturbosch/detekt/api/Findings.kt
  MethodCallDepth - 4/2 - /Procedural - Line/Column=(22,2) - Path=detekt-api/src/main/kotlin/io/github/arturbosch/detekt/api/Findings.kt
  MethodCallDepth - 4/2 - /Procedural - Line/Column=(44,2) - Path=detekt-api/src/main/kotlin/io/github/arturbosch/detekt/api/Findings.kt
  TooManyFunctions - 12/28 - /Procedural - Line/Column=(1,1) - Path=detekt-api/src/main/kotlin/io/github/arturbosch/detekt/api/Findings.kt
Ruleset: empty
  EmptyFunctionBlock - /visit - Line/Column=(94,45) - Path=detekt-api/src/main/kotlin/io/github/arturbosch/detekt/api/Male.kt
  EmptyFunctionBlock - /visit - Line/Column=(102,44) - Path=detekt-api/src/main/kotlin/io/github/arturbosch/detekt/api/Male.kt
  EmptyFunctionBlock - /visit - Line/Column=(12,20) - Path=detekt-api/src/main/kotlin/io/github/arturbosch/detekt/api/ProcessListeners.kt
  EmptyFunctionBlock - /process - Line/Column=(12,38) - Path=detekt-api/src/main/kotlin/io/github/arturbosch/detekt/api/ProcessListeners.kt
  EmptyFunctionBlock - /visit - Line/Column=(4,55) - Path=detekt-api/src/main/kotlin/io/github/arturbosch/detekt/api/ProcessListeners.kt
Ruleset: exceptions
  CatchAndSwallowException - (e) - Line/Column=(32,11) - Path=detekt-api/src/main/kotlin/io/github/arturbosch/detekt/api/Junk.kt
Ruleset: potential-bugs
Ruleset: style
Complexity Report:
  1077 logical lines of code (lloc)
  263 McCabe complexity (mcc)
  311 mv per 1000 lloc
  7 code smells per 1000 lloc
detekt run within 1025 ms
Exception in thread "main" io.github.arturbosch.detekt.cli.SmallThresholdBuildFailure: Build failure threshold of 10 reached with 13 weighted smells!
at io.github.arturbosch.detekt.cli.SmallThreshold.check(SmallThreshold.kt:31)
at io.github.arturbosch.detekt.cli.Runner.executeRunner(Runner.kt:23)
at io.github.arturbosch.detekt.cli.MainCompanion.main(Main.kt:57)
at io.github.arturbosch.detekt.cli.Main.main(Main.kt:1)
```

### Features

- Code smell analysis for your Kotlin projects
- Complexity reports based on lines of code, cyclomatic complexity and amount of code smells
- Highly configurable rule sets
- Suppression of findings with Kotlin's `@Suppress` and Java's `@SuppressWarnings` annotations
- Specification of quality gates which will break your build
- Code Smell baseline and whitelisting for legacy projects
- **Gradle plugin** for code analysis via Gradle builds
- Gradle tasks to use local `IntelliJ` distribution for formatting and inspecting Kotlin code

# 코드 스타일 검사

build.gradle 파일에 아래 코드 추가

```
plugins {  
    id("io.gitlab.arturbosch.detekt").version("1.5.1")  
}
```

gradle sync 후 아래와 같은 명령 사용 가능

```
./gradlew detekt
```

위 명령을 **Jenkinsfile**에 추가하여 CI 톨과 연동하거나 **git pre-commit hook**에 등록하여 잘못된 스타일의 코드가 커밋되는 것을 원천적으로 방어할 수 있음

# 테스트 커버리지 측정






















- 테스트 커버리지: 테스트 코드가 실제 코드를 얼마나 커버해주는가?

## JaCoCo - <https://github.com/jacoco/jacoco>

자바 코드 커버리지 검사 도구  
(코틀린 코드도 됨)

gradle 플러그인으로 제공

### JaCoCo

Element	Missed Instructions	Cov.	Missed Branches
 <a href="#">org.jacoco.examples</a>		58%	
 <a href="#">org.jacoco.core</a>		97%	
 <a href="#">org.jacoco.agent.rt</a>		77%	
 <a href="#">jacoco-maven-plugin</a>		90%	
 <a href="#">org.jacoco.cli</a>		97%	
 <a href="#">org.jacoco.report</a>		99%	
 <a href="#">org.jacoco.ant</a>		98%	
 <a href="#">org.jacoco.agent</a>		86%	
Total	1,355 of 27,352	95%	143 of 2,125

# 테스트 커버리지 측정

build.gradle 파일에 아래 코드 추가

```
plugins {  
    id 'jacoco'  
}  
  
jacoco {  
    toolVersion = "0.8.5"  
    reportsDir = file("$buildDir/customJacocoReportDir")  
}
```

gradle sync 후 아래와 같은 명령 사용 가능

```
./gradlew jacocoTestReport
```

위 명령을 실행하면 프로젝트/build/reports/jacoco/ 경로에 **HTML** 문서가 생성됨

Jenkinsfile에 추가하여 CI 툴과 연동 가능

# 소스코드 문서화

- 내가 개발한 공통 모듈을 다른 개발자에게 쉽게 설명하기

## Dokka - <https://github.com/Kotlin/dokka>

코틀린 소스코드 문서 생성 도구  
(자바 코드도 됨)

KDoc 문법으로 작성된 주석을 기반으로  
문서를 생성해 줌

gradle 플러그인으로 제공

README.md

dokka   Download 0.10.1

Dokka is a documentation engine for Kotlin, performing the same function as javadoc for Java. Just like Kotlin itself, Dokka fully supports mixed-language Java/Kotlin projects. It understands standard Javadoc comments in Java files and [KDoc comments](#) in Kotlin files, and can generate documentation in multiple formats including standard Javadoc, HTML and Markdown.

### Using dokka

#### Using the Gradle plugin

```
buildscript {
    repositories {
        jcenter()
    }
    dependencies {
        classpath "org.jetbrains.dokka:dokka-gradle-plugin:${dokka_version}"
    }
}
repositories {
    jcenter() // or maven { url 'https://dl.bintray.com/kotlin/dokka' }
}

apply plugin: 'org.jetbrains.dokka'
```

or using the plugins block:

```
plugins {
    id 'org.jetbrains.dokka' version '0.10.1'
}
repositories {
    jcenter() // or maven { url 'https://dl.bintray.com/kotlin/dokka' }
```

## KDoc 문법 (예시)

```
/**
 * [타임스탬프][ts]로 부터 Day Code를 가져온다.
 *
 * @param ts Unix 타임스탬프
 * @return `yyyyMMdd` 형식의 문자열
 */
fun getDayCodeFromTS(ts: Long): String {
    return getDateTimeFromTS(ts).toString(YYYYMMdd)
}
```

HTML 기반의 Javadoc과 달리 Markdown 기반임

자세한 설명은 <https://kotlinlang.org/docs/reference/kotlin-doc.html> 참조



# 소스코드 문서화

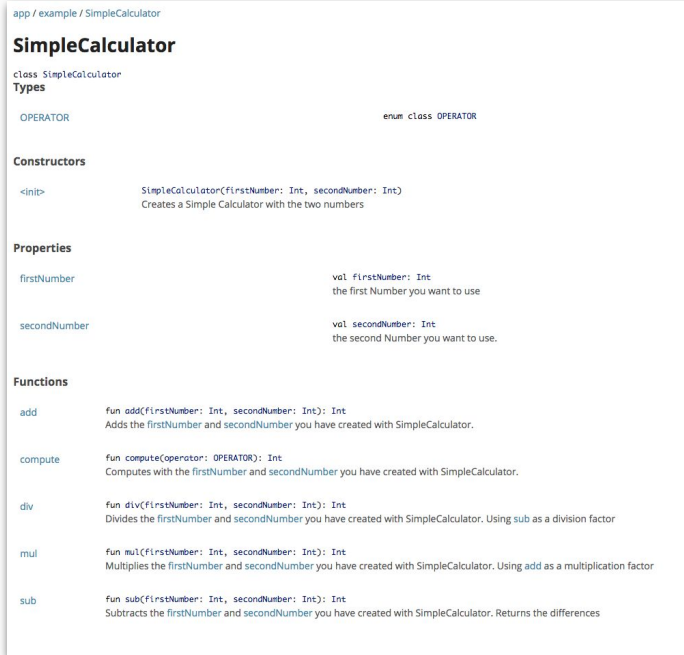
build.gradle 파일에 아래 코드 추가

```
plugins {  
    id 'org.jetbrains.dokka' version '0.10.1'  
}
```

```
dokka {  
    outputFormat = 'html'  
    outputDirectory = "$buildDir/dokka"  
}
```

gradle sync 후 아래와 같은 명령 사용 가능

```
./gradlew dokka
```



위 명령을 실행하면 프로젝트/build/dokka/ 경로에 HTML 문서가 생성됨

Jenkinsfile에 추가하여 CI 툴과 연동 가능

**End**