

Code Quality CI/CD Demo

Part 1: Code Quality

Качество кода

- Качество кода играет решающую роль в разработке ПО, поскольку это напрямую влияет на общий успех и долговечность программы.
- Написание высококачественного кода — не только вопрос личных предпочтений или стиля, но и профессиональная ответственность разработчиков.
- Качество кода влияет на различные аспекты разработки ПО:
 - Readability (удобочитаемость),
 - Maintainability (поддержка приложения),
 - Scalability (масштабируемость),
 - Collaboration (совместную работу в команде).

The Sun Code Style Guidelines

- Также известны как Oracle Code Style Guidelines
- Набор соглашений/конвенций/рекомендаций и лучших практик для написания Java-кода
- Цель этих рекомендаций — предоставить стандартизированный подход к написанию Java-кода, гарантирующий согласованность и удобочитаемость между проектами и разработчиками.
- Преимущества использования лучших практик:
 - **Readability**
 - Консистентное форматирование кода, отступы и конвенции о наименовании облегчают чтение и понимание кода как автору, так и другим разработчикам, которым нужно работать с кодом.
 - **Maintainability**
 - Согласованный стиль написания кода облегчает поиск и изменение определенных разделов кода, снижая вероятность ошибок.
 - **Collaboration**
 - Когда вся команда придерживается одних и тех же соглашений, это способствует созданию сплоченной и совместной разработки. Разработчики могут более эффективно понимать код друг друга и работать с ним, что облегчает совместную и командную работу.

Java Code Style Guidelines

- [Oracle Java Code Conventions](#) is a must :)
- [Google Java Style Guide](#)

4 - Indentation

Four spaces should be used as the unit of indentation. The exact construction of the indentation (spaces vs. tabs) is unspecified. Tabs must be set exactly every 8 spaces (not 4).

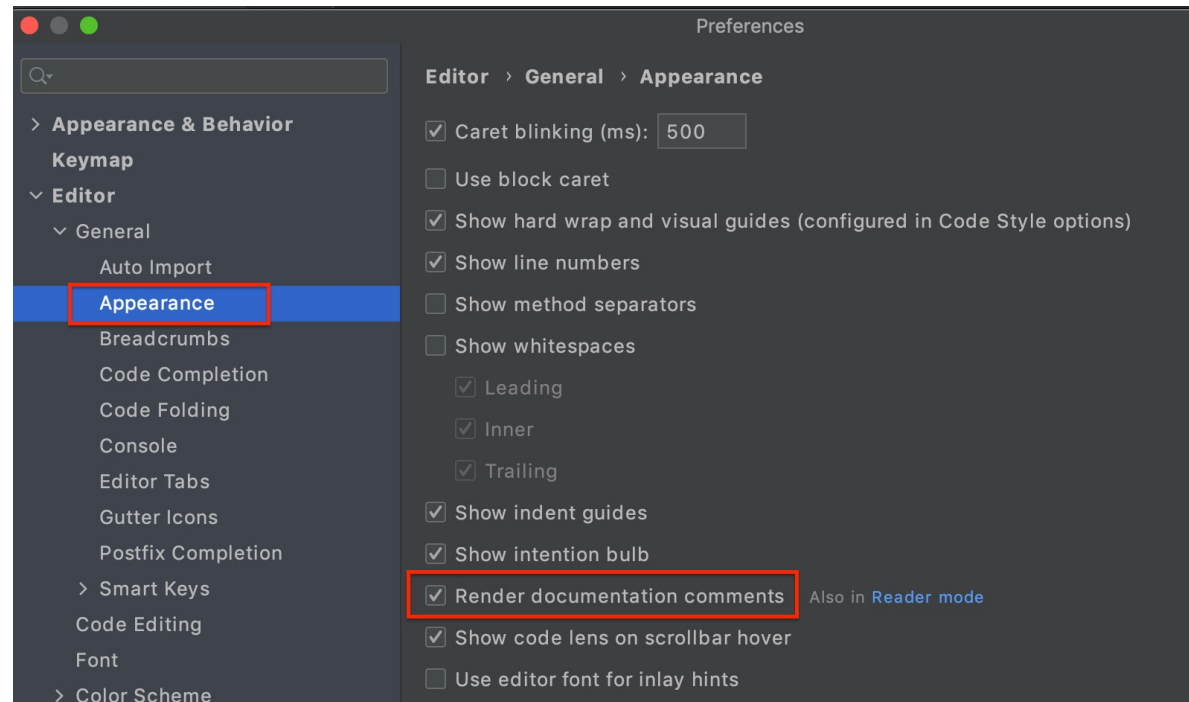
4.1 **Line Length**

Avoid lines longer than 80 characters, since they're not handled well by many terminals and tools.

Note: Examples for use in documentation should have a shorter line length—generally no more than 70 characters.

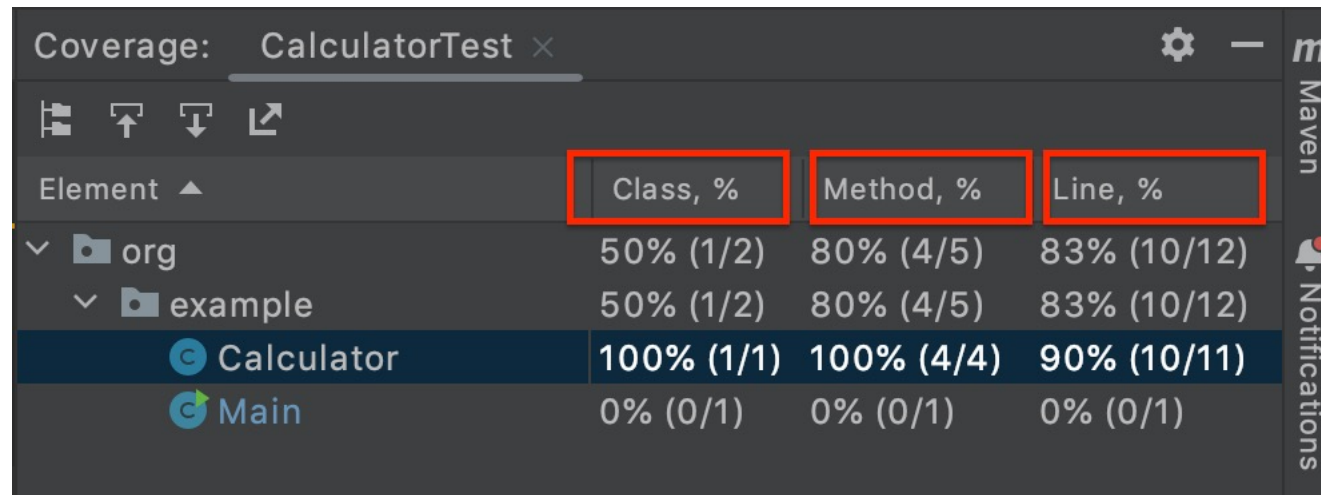
JavaDoc

- Начинаются с `/** ... */`
- [Javadocs in IntelliJ](#) tutorial
- Oracle: [How to Write Doc Comments for the Javadoc Tool](#)



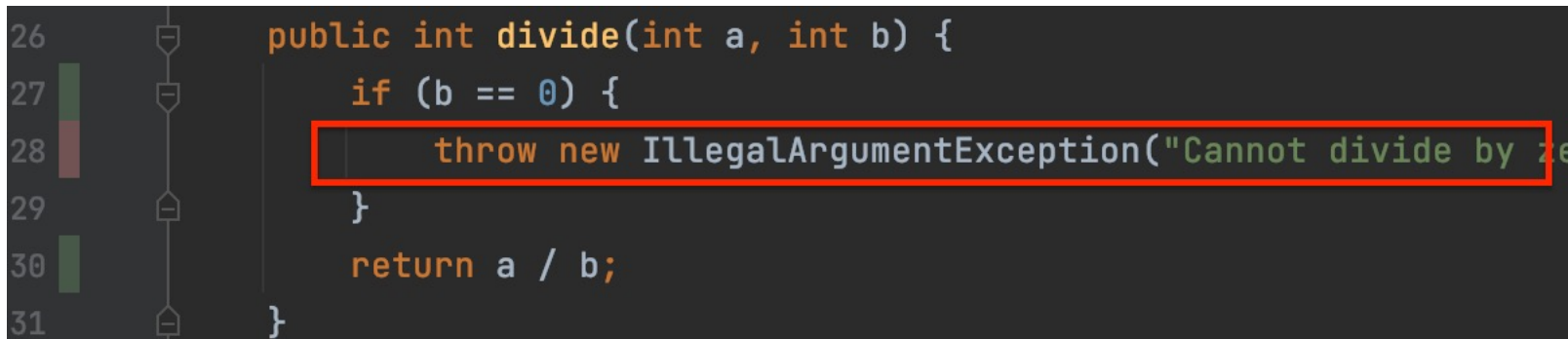
Test Coverage

- IntelliJ: [Run with Coverage](#) tutorial



The screenshot shows the IntelliJ Coverage window for the CalculatorTest. The window has a title bar 'Coverage: CalculatorTest' with a close button. Below the title bar are icons for coverage reports. The main table shows coverage data for the 'org' and 'example' packages, and the 'Calculator' and 'Main' classes. The 'Calculator' class is highlighted. The columns are 'Element', 'Class, %', 'Method, %', and 'Line, %'. The 'Calculator' class has 100% class coverage (1/1), 100% method coverage (4/4), and 90% line coverage (10/11). The 'Main' class has 0% coverage for all metrics.

Element ▲	Class, %	Method, %	Line, %
org	50% (1/2)	80% (4/5)	83% (10/12)
example	50% (1/2)	80% (4/5)	83% (10/12)
Calculator	100% (1/1)	100% (4/4)	90% (10/11)
Main	0% (0/1)	0% (0/1)	0% (0/1)



The screenshot shows the IntelliJ code editor with the 'divide' method. The method signature is 'public int divide(int a, int b)'. The code inside the method is: 'if (b == 0) { throw new IllegalArgumentException("Cannot divide by zero"); } return a / b;'. The line 'throw new IllegalArgumentException("Cannot divide by zero");' is highlighted with a red box.

```
26 public int divide(int a, int b) {  
27     if (b == 0) {  
28         throw new IllegalArgumentException("Cannot divide by zero");  
29     }  
30     return a / b;  
31 }
```

Checkstyle

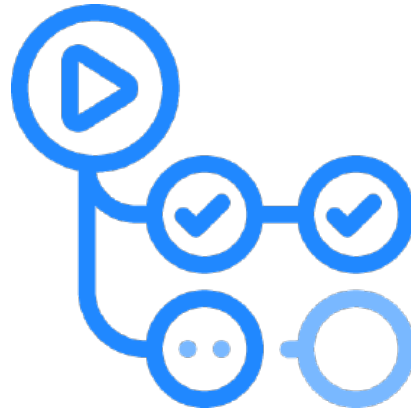
- [Maven Checkstyle Plugin](#)
 - Конфигурация через файл **checkstyle.xml**
- [SonarQube Checkstyle Plugin](#)
- IntelliJ hotkey combinations:
 - [Top 15 IntelliJ IDEA Shortcuts](#)
 - Отформатировать текущий файл: $\text{⌘}L$ or Ctrl+Alt+L
 - [IntelliJ IDEA keyboard shortcuts](#)

Part 2: CI/CD Demo

Workflow & Stack



1: Java Application with Maven Build



2: CI/CD with GitHub Actions



**3: Push Docker Image to
DockerHub Private Repository**

Создаём Java-приложение

1. Создаем Java-приложение со сборкой Maven / Gradle
 - В данном примере используется Maven
2. Создаем Dockerfile в корне проекта (по умолчанию)
 - **Внимание:** название должно быть «Dockerfile» (не «dockerfile»)
 - Пример Docker-файла:

```
# Use a base Java image  
FROM openjdk:19  
  
# Set the working directory inside the container  
WORKDIR /your-project  
  
# Copy the compiled Java application to the container  
COPY src ./src  
  
# Set the entry point for the container  
ENTRYPOINT ["java", "/your/full/path/to/Main.java"]
```

Настраиваем CI/CD с GitHub Actions

- Пушим приложение в GitHub
- Находясь в репозитории, переходим во вкладку «Actions»
 - Click on «New workflow»
 - Search for «Java with Maven», click on «Configure»
 - Данный шаблон позволит собрать проект на виртуальной машине
 - Используем этот шаблон у себя в проекте, называем как «build.yml» (например)
 - Commit & push
 - В корне проекта появится папка / .github/workflows
 - Также можно отдельно настроить workflow для тестов

Push to DockerHub Private Repository

- Чтобы запустить docker image:
 1. Заводим аккаунт на <https://hub.docker.com/>
 - Разрешен только 1 приватный репозиторий, публичные репозитории без лимита
 2. В файле сборки проекта «build.yml» добавляем шаг публикации в DockerHub
 - Обращаемся за помощью к [Docker Build & Push Action](#)
 - Используем его в своем файле конфигурации workflow
 - См. секцию «Examples»
 - Добавляем в своем GitHub-репозитории секреты во вкладке «Settings»
 - В файл сборки добавляем логин/пароль от аккаунта в DockerHub: DOCKER_USERNAME и DOCKER_PASSWORD
 - Можно добавить тег для Docker Image, см. флаги
 - Commit & Push
 - Profit

Publish Docker Image to DockerHub

```
build.yml
```


```
...
```

```
- name: Build and Push Docker Image
  uses: mr-smithers-excellent/docker-build-push@v6
  with:
    image: your-login/repo-name
    tags: latest
    registry: docker.io
    username: ${ secrets.DOCKER_USERNAME }
    password: ${ secrets.DOCKER_PASSWORD }
```

GitHub Secrets: How To



```
username: ${ secrets.DOCKER_USERNAME }  
password: ${ secrets.DOCKER_PASSWORD }
```

The screenshot displays the GitHub repository settings page for a specific repository. The top navigation bar includes links for Code, Issues, Pull requests, Actions, Projects, Wiki, Security, and Insights. The 'Settings' tab is highlighted. The left sidebar contains a list of settings categories: General, Access (Collaborators, Moderation options), Code and automation (Branches, Tags, Rules, Actions, Webhooks, Environments, Codespaces, Pages), and Security (Code security and analysis, Deploy keys, Secrets and variables). The 'Secrets and variables' option is selected under the 'Code and automation' section. The main content area is titled 'Actions secrets and variables' and includes a 'New repository secret' button. Below this, there are sections for 'Environment secrets' (currently empty) and 'Repository secrets' which lists two secrets: 'DOCKER_PASSWORD' and 'DOCKER_USERNAME', both updated 5 hours ago.



ExploreRepositoriesOrganizationsHelp



Upgrade


albina494>Repositories>demo-app>General


Using 1 of 1 private repositories. [Get more](#)

GeneralTagsBuildsCollaboratorsWebhooksSettings

 a -app

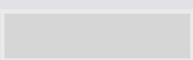
Description

This repository does not have a description 

 Last pushed: a few seconds ago



Docker commands

To push a new tag to this repository,

```
docker push a-app:tagname
```

Tags


This repository contains 1 tag(s).

Tag	OS	Type	Pulled	Pushed
 master-0d9c4a1		Image	---	2 hours ago

[See all](#)[Go to Advanced Image Management](#)

Automated Builds

Manually pushing images to Hub? Connect your account to GitHub or Bitbucket to automatically build and tag new images whenever your code is updated, so you can focus your time on creating.

Available with Pro, Team and Business subscriptions. [Read more about automated builds](#) .

Upgrade

Орг. вопросы

- Финальная защита проектов 20 июня, 2 пары с 18:10, онлайн

Командный проект (2)

- **Обязательно:**

- Стандартная система сборки (Maven или Gradle)
- Юнит-тесты — обязательно со 2-го демо-дня, желательно с 1-го
- Исполняемый артефакт (Docker, GraalVM native-image, jlink image) — обязательно с 3-го демо-дня. До этого, **можно** исполняемый JAR-файл + запускать руками/скриптом (java -jar ...)
- Сборка и деплой в системе непрерывной интеграции (GitHub Actions) — к 4-му демо-дню

- **Можно:**

- Библиотеки, напр. Google Guava
 - Можно даже библиотеку, которую сделает соседняя команда, но преподаватель должен об этом знать заранее
- Паттерны, абстракции (без фанатизма :-))

- **Нельзя:**

- Тривиальная «обёртка» над готовой внешней библиотекой, программой, веб-сервисом...
- Любая форма плагиата, в т.ч. креативно переработать студенческие проекты прошлого года