Домашнее задание: тестирование PasswordPolicy по спецификации

Цель задания

Научиться проектировать тесты на основе спецификации, находить с их помощью расхождения между требованиями и реализацией и закрывать найденные дефекты регрессионными тестами.

Ваш результат

В результате выполнения задания вы:

- создадите тест-сьют для класса PasswordPolicy;
- проверите правила из спецификации, включая граничные значения;
- найдёте расхождения между спецификацией и реализацией и оформите их как баг-репорты;
- исправите найденные дефекты и добавите регрессионные тесты;
- сформируете отчёты Allure или Jacoco и подтвердите необходимый уровень покрытия.

Материалы для выполнения

Для работы используйте:

1. Проект code/demo-junit — тот же проект, который использовался на семинаре.
2. Класс ru.mts.itmo.aqa.junit5.PasswordPolicy.
3. Спецификацию поведения PasswordPolicy, приведённую в задании.
4. Пример AllureDemoTest в том же пакете — для разметки Allure.

Спецификация из задания является единственным источником требований к поведению класса. Если фактическое поведение реализации расходится со спецификацией, это считается дефектом.

Спецификация

Пароль считается валидным, когда список нарушений пуст. Правила применяются независимо друг от друга: в результат попадают все нарушения сразу, а не только первое найденное.

Условие	**Результат**
пароль равен null	выбрасывается IllegalArgumentException (это исключение, а не нарушение)
длина меньше 8	TOO_SHORT
длина больше 64	TOO_LONG. Длина ровно 64 — ещё допустима
нет ни одной цифры	NO_DIGIT
нет ни одной заглавной латинской буквы	NO_UPPER
нет ни одной строчной латинской буквы	NO_LOWER
нет ни одного символа из !@#$%^&*()-_=+	NO_SPECIAL
есть пробельный символ (пробел, табуляция и подобные)	HAS_WHITESPACE
три и более одинаковых символа подряд	REPEATED_RUN. «aaa» — нарушение, «aa» — нет
пароль содержит логин, без учёта регистра	CONTAINS_LOGIN. Правило применяется только если логин передан и не пустой
пароль совпадает с одним из простых: password, qwerty, 123456, admin, welcome, без учёта регистра	BLACKLISTED

Что дано

Класс ru.mts.itmo.aqa.junit5.PasswordPolicy в code/demo-junit/src/main/java/. Тестов на него в проекте нет — в отчёте Jacoco он показывает 0 %. Публичный интерфейс такой:

```
PasswordPolicy policy = new PasswordPolicy();

// вариант без логина
PasswordPolicy.Result r1 = policy.check("Aa1!bcde");

// вариант с логином пользователя
PasswordPolicy.Result r2 = policy.check("Aa1!bcde", "ivan");

r2.isValid();              // true, если нарушений нет
r2.violations();  // List<Violation> — что именно нарушено
```

Violation — перечисление: TOO_SHORT, TOO_LONG, NO_DIGIT, NO_UPPER, NO_LOWER, NO_SPECIAL, HAS_WHITESPACE, REPEATED_RUN, CONTAINS_LOGIN, BLACKLISTED.

Два свойства, важных для проектирования тестов. Первое: правила проверяются независимо, в результат попадают все нарушения сразу. Второе: порядок нарушений в списке спецификацией не определён,

опираться на него в тестах нельзя — используйте проверки без учёта порядка.

Подготовка

1. Откройте проект code/demo-junit.
2. Убедитесь, что для работы используется JDK 21.
3. Найдите класс ru.mts.itmo.aqa.junit5.PasswordPolicy.
4. До изучения его реализации ознакомьтесь со спецификацией из задания и спроектируйте проверки на её основе.

Задача

Вам дан класс, который проверяет пароль по набору правил, и текстовая спецификация этих правил.

Ваша задача — написать для класса тест-сьют, опираясь только на спецификацию, найти с помощью тестов места, где реализация не соответствует требованиям, оформить найденные расхождения как баг-репорты, исправить их и закрыть регрессионными тестами.

Тесты необходимо проектировать **по спецификации, а не по текущей реализации класса**.

Что нужно сделать

1. Спроектируйте проверки

По каждому правилу из спецификации определите классы эквивалентности и граничные значения.

Для каждого правила предусмотрите случаи, когда оно должно срабатывать и когда не должно срабатывать.

Особое внимание уделите:

- границам длины пароля — 7/8 и 64/65;
- длине серии одинаковых символов — 2/3;
- правилам, работающим без учёта регистра;
- трём состояниям логина для CONTAINS_LOGIN: непустой логин, пустой логин, отсутствие логина.

Тестовую модель можно вести в любом удобном виде. Отдельным файлом сдавать её не нужно, но необходимо быть готовым показать её на разборе задания.

2. Напишите тест-сьют

Создайте файл:

code/demo-junit/src/test/java/ru/mts/itmo/aqa/junit5/PasswordPolicyTest.java

В тест-сьюте:

- сгруппируйте проверки по правилам через @Nested и @DisplayName;
- добавьте человекочитаемый @DisplayName для каждого теста;
- проверьте каждое правило на срабатывание и несрабатывание;
- проверьте границы 7/8, 64/65 и 2/3 через @ParameterizedTest;
- используйте хотя бы один @MethodSource;
- добавьте тест на накопление нескольких нарушений одновременно;
- добавьте отдельный тест на IllegalArgumentException для null;
- используйте AssertJ;
- хотя бы один тест реализуйте с SoftAssertions.

Для проверки длины используйте валидный пароль нужной длины, чтобы другие правила не мешали проверяемому условию.

Покрытие ветвей класса PasswordPolicy должно быть не ниже 90 % — критерий проверяется машинно, по отчёту mvn clean test → target/site/jacoco/index.html. Планка достижима: у эталонного решения 96 %. Вложенный класс Result в критерий не входит, Jacoco считает его отдельной строкой отчёта. При этом покрытие — не цель, а следствие: тест без ассерта поднимает процент и не засчитывается.

Агент Jacoco по умолчанию дописывает данные к предыдущему прогону, поэтому без clean процент покрытия будет завышен и вы будете смотреть на неправду. Для замера покрытия всегда mvn clean test.

3. Найдите расхождения и оформите баг-репорты

Если тест, написанный по спецификации, падает, определите, является ли причиной расхождение реализации с требованиями.

На каждый найденный дефект создайте отдельный баг-репорт в файле:

PasswordPolicyTest-баги.md

Для каждого дефекта укажите:

- заголовок;
- окружение: проект, класс и версию JDK;
- шаги воспроизведения;

- фактический результат;
- ожидаемый результат с цитатой из спецификации;
- Severity и Priority с кратким обоснованием;
- при необходимости — код падающего теста.

Один баг-репорт должен описывать один дефект.

4. Исправьте найденные дефекты и добавьте регрессионные тесты

Исправьте найденные расхождения в своей копии PasswordPolicy.

Для каждого дефекта соблюдайте последовательность:

сначала падающий тест → затем исправление реализации.

После исправления тест должен стать зелёным.

В комментарии к каждому регрессионному тесту укажите, какой дефект он фиксирует.

Порядок «сначала тест, потом фикс» нужно подтвердить при сдаче. Подойдёт любой из двух способов: ссылка на ветку, где видно два коммита подряд — сначала красный тест, затем исправление реализации; либо скриншот красного прогона теста на исходной реализации. Без такого подтверждения критерий «Регресс» считается невыполненным.

5. Оформите отчёт Allure

Настройте разметку тестов так, чтобы структура отчёта соответствовала структуре спецификации.

Используйте:

@Epic
@Feature
@Story
@Step
@Attachment

@Story используется для отдельных правил.

Повторяющееся действие вынесите в метод с @Step.

Метод с @Attachment должен возвращать строку или массив байтов.

Для примера используйте AllureDemoTest из того же пакета.

Важные ограничения

Тесты должны быть написаны по спецификации, а не по фактической реализации PasswordPolicy.

Не изменяйте ожидаемое поведение теста только потому, что реализация работает иначе.

Не проверяйте порядок нарушений через containsExactly: спецификация не задаёт порядок элементов в списке.

Не используйте пустые вызовы без ассертов только для повышения процента покрытия.

Не переписывайте PasswordPolicy целиком — исправления должны быть минимальными.

Критерии проверки

Задание оценивается максимум в 12 баллов по критериям ниже.

Критерий	**Макс.**	**Полный балл**	**Половина балла**	**0 баллов**
Покрытие	2	Покрытие ветвей PasswordPolicy 90 % и выше, тесты содержат проверки	Тесты содержат проверки, но покрытие ветвей ниже 90 %	Тесты отсутствуют или не содержат проверок
Границы	2	Проверены все пары граничных значений: 7/8, 64/65 и 2/3	Проверена только часть граничных значений	Граничные значения не проверены
Полнота	2	Каждое правило проверено и на срабатывание, и на несрабатывание	Часть правил проверена только в одном сценарии или проверены не все правила	Системной проверки правил нет
Найденные дефекты	2	Найдено больше одного расхождения, на каждое — воспроизводимый баг-репорт с цитатой из спецификации	Расхождения найдены, но требования к баг-репортам выполнены не полностью	Расхождения не найдены или баг-репортов нет

Регресс	2	На каждый дефект есть регрессионный тест, и подтверждён порядок «тест до фикса»	Регрессионные тесты есть не для всех дефектов либо порядок не подтверждён	Регрессионных тестов нет
Читаемость	1	Используются @Nested и @DisplayName, структура тестов читается как спецификация	—	Требования к структуре и читаемости не выполнены
Allure	1	Дерево отчёта соответствует структуре правил, есть @Step и @Attachment	—	Требования к отчёту Allure не выполнены
Итого	12			

Что и как сдавать

Отправьте:

1. PasswordPolicyTest.java — тест-сьют.
2. PasswordPolicyTest-баги.md — баг-репорты на найденные расхождения.
3. Исправленный PasswordPolicy.java и регрессионные тесты на каждый дефект.
4. Два скриншота:
    - дерево отчёта Allure;
    - строка PasswordPolicy из отчёта Jacoco с видимым процентом покрытия ветвей.
5. Подтверждение порядка «тест до фикса»: ссылка на ветку с двумя коммитами либо скриншот красного прогона.

Формат сдачи: ссылка на ветку в вашем репозитории (предпочтительно) или архив.

Дедлайн: **28 сентября**.

Команды

Прогон только тест-класса:

mvn test -Dtest=PasswordPolicyTest

Полный прогон с пересчётом покрытия:

mvn clean test

Отчёт Jacoco:

target/site/jacoco/index.html

Отчёт Allure:

allure serve target/allure-results

Все команды запускаются из каталога:

code/demo-junit