Вопросы к курсу

«Создание модульных тестов с помощью JUnit»

Оглавление

[2. Основные компоненты тестового класса 2](#__RefHeading__4030_1143559875)

[3. Группировка тестов 6](#__RefHeading__4032_1143559875)

[4. Утверждения и предположения 12](#__RefHeading__5238_465614980)

[1) Каким будет результат прогона следующего теста: 12](#__RefHeading__5240_465614980)

[1) Каким будет результат прогона следующего теста: 13](#__RefHeading__5242_465614980)

[3) Каким будет результат прогона следующего теста: 13](#__RefHeading__5244_465614980)

[4) Каким будет результат прогона следующего теста: 14](#__RefHeading__5246_465614980)

[5) Каким будет результат прогона следующего теста: 14](#__RefHeading__5569_465614980)

[5) Каким будет результат прогона следующего теста: 15](#__RefHeading__5571_465614980)

[6) Каким будет результат прогона следующего теста: 16](#__RefHeading__5573_465614980)

[7) Каким будет результат прогона следующего теста: 16](#__RefHeading__5575_465614980)

# 2. Основные компоненты тестового класса

Вопросы:

1) Является ли следующий класс тестовым:

**public** **class** Question {

**public** **void** testSum() {

Assert.*assertEquals*(1 + 1, 2);

}

}

A1: Является

A2: Не является

Объяснение: A2, так как метод testSum не является тестовым. Он не аннотирован аннотацией @Test. И так как в классе нет тестовых методов, он не является тестовым классом

2) Будет ли метод testSum2 выполнен успешно:

**public** **class** QuestionTest {

@Test

**public** **void** testSum1() {

Assert.*assertEquals*(1+1, 2);

}

@Test

@Ignore

**public** **void** testSum2() {

}

@Test

**public** **void** testSum3() {

Assert.*assertEquals*(1, 2);

}

@Test

**public** **void** testSum4() {

Assert.*assertEquals*(1, ((Integer) **null**).intValue());

}

}

+A1: Нет

A2: Да

Объяснение: тестовый метод помечен как пропущенный @Ignored

3) Будет ли выполнен успешно данный тестовый метод:

**public** **class** QuestionTest {

**private** Calculator calc = **new** Calculator();

@Test

**public** **void** testSum1() {

Assert.*assertEquals*(2, calc.divide(10, 5));

}

}

**class** Calculator {

**int** divide(**int** a, **int** b) {

**return** a/b;

}

}

+A1: да

A2: нет, в результате выполения, метод помечен как failure

A3: нет, в результате выполения, метод помечен как error

4) Будет ли выполнен успешно данный тестовый метод:

**public** **class** QuestionTest {

**private** Calculator calc = **new** Calculator();

@Test

**public** **void** testSum1() {

Assert.*assertEquals*(2, calc.divide(10, 3));

}

}

**class** Calculator {

**int** divide(**int** a, **int** b) {

**return** a/b;

}

}

A1: да

+A2: нет, в результате выполения, метод помечен как failure

A3: нет, в результате выполения, метод помечен как error

5) Будет ли выполнен успешно данный тестовый метод:

**public** **class** QuestionTest {

**private** Calculator calc = **new** Calculator();

@Test

**public** **void** testSum1() {

Assert.*assertEquals*(2, calc.divide(10, 0));

}

}

**class** Calculator {

**int** divide(**int** a, **int** b) {

**return** a/b;

}

}

A1: да

A2: нет, в результате выполения, метод помечен как failure

+A3: нет, в результате выполения, метод помечен как error

6) Будут ли выполнены успешно тестовые методы в следующем тестовом классе:

**public** **class** QuestionTest {

**private** Counter counter = **new** Counter();

@Test

**public** **void** testIncrement1() {

Assert.*assertEquals*(1, counter.increment());

}

@Test

**public** **void** testIncrement2() {

Assert.*assertEquals*(1, counter.increment());

}

}

**class** Counter {

**private** **int** i = 0;

**int** increment() {

**return** ++i;

}

**void** reset() {

i = 0;

}

}

+A1: Да

А2: Нет, откажет тестовый метод testIncrement1

А3: Нет, откажет тестовый метод testIncrement2

A4: Нет, откажет один из тестовых методов (порядок не определен)

Объяснение: Для каждого тестового случая создается новый экземпляр класса QuestionTest и соответственно экземпляр класса Counter

7) Будут ли выполнены успешно тестовые методы в следующем тестовом классе:

**public** **class** QuestionTest {

**private** static Counter counter = **new** Counter();

@Test

**public** **void** testIncrement1() {

Assert.*assertEquals*(1, counter.increment());

}

@Test

**public** **void** testIncrement2() {

Assert.*assertEquals*(1, counter.increment());

}

}

**class** Counter {

**private** **int** i = 0;

**int** increment() {

**return** ++i;

}

**void** reset() {

i = 0;

}

}

A1: Да

А2: Нет, откажет тестовый метод testIncrement1

А3: Нет, откажет тестовый метод testIncrement2

+A4: Нет, откажет один из тестовых методов (порядок не определен)

8) Что нужно сделать, чтобы вызов тестовых методов в тестовом классе завершился успешно:

**public** **class** QuestionTest {

**private** static Counter counter = **new** Counter();

@Test

**public** **void** testIncrement1() {

Assert.*assertEquals*(1, counter.increment());

}

@Test

**public** **void** testIncrement2() {

Assert.*assertEquals*(1, counter.increment());

}

}

**class** Counter {

**private** **int** i = 0;

**int** increment() {

**return** ++i;

}

**void** reset() {

i = 0;

}

}

A1: Ничего, тест будет выполнен успешно без изменений

А2: Поменять методы местами (в исходном файле метод testIncrement2 должен быть объявлен перед методом testIncrement

А3: Добавить метод:

@BeforeClass

**public** **static** **void** before() {

*counter*.reset();

}

+A4: Добавить метод:

@Before

**public** **void** before() {

*counter*.reset();

}

9) Будет ли выполняться тестовый метод в следующем классе:

**public** **class** QuestionTest {

@Test

**public** **static** **void** testMethod() {

Assert.*assertEquals*(1 + 1, 3);

}

}

A1: Да

+А2: Нет, потому что тестовый метод статический

А3: Нет, будет считаться отказавшим, потому что 1+1 != 3

Еще на этапе анализа тестового класса JUnit посчитает его некорретным, т. к. тестовый метод не должен быть статическим

10) Будет ли выполняться тестовый метод в следующем классе:

**public** **class** QuestionTest {

@Test

**void** method() {

Assert.*assertEquals*(1 + 1, 3);

}

}

A1: Да

+А2: Нет, потому что тестовый метод имеет модификатор видимости уровня пакета

А3: Нет, потому что тестовый метод не имеет префикс test

А4: Нет, будет считаться отказавшим, потому что 1+1 != 3

Еще на этапе анализа тестового класса JUnit посчитает его некорретным, т. к. тестовый метод не должен быть статическим

# 3. Группировка тестов

Вопросы:

1) Является ли следующий класс корректно описанным классом набора тестов:

@RunWith(Suites.**class**)

@SuiteClasses( {

CalculatorDivideTest.**class**,

CalculatorSqrtTest.**class**

})

**public** **class** BasicTests {

}

+А1: Да

А2: нет, он должен быть объявлен как интерфейс

A3: нет, он должен быть аннотирован @RunWith(JUnit4.**class**)

2) Является ли следующий класс корректно описанным классом набора тестов:

@RunWith(Suite.**class**)

@SuiteClasses( {

CalculatorDivideTest.**class**,

CalculatorSqrtTest.**class**

})

**public** **class** BasicTests {

**public** BasicTests() {

System.*out*.println("Doing one-time work here");

}

}

+A1:Да

A2: нет, в классе не должно быть объявленного конструктора

A3: нет, он должен быть аннотирован @RunWith(JUnit4.**class**)

Объявленный конструктор никак не влияет на работу класса как класса набора тестов, поскольку этот класс служит просто хранилищем для аннотаций

3) Является ли следующий класс корректно описанным классом набора тестов:

@RunWith(Suite.**class**)

@SuiteClasses( {

CalculatorDivideTest.**class**,

CalculatorSqrtTest.**class**

})

**public** **class** BasicTests {

**public** BasicTests(String s) {

System.*out*.println("Doing one-time work here");

}

}

+A1:Да

A2: нет, в классе должен присутствовать конструктор с параметрами

A3: нет, он должен быть аннотирован @RunWith(JUnit4.**class**)

Объявленный конструктор никак не влияет на работу класса как класса набора тестов, поскольку этот класс служит просто хранилищем для аннотаций

4) Что будет напечатано в консоли в результате запуска набора тестов BasicTests:

**public** **class** Test1 {

@Test

**public** **void** testMethod() {

}

}

@RunWith(Categories.**class**)

@SuiteClasses( {

Test1.**class**

})

**public** **class** BasicTests {

**public** BasicTests() {

System.*out*.println("In constructor");

}

**public** **void** before() {

System.*out*.println("In before method");

}

}

+A1:Ничего

A2: In constructor

A3: In constructor

In before method

А4: In before method

In constructor

Объявленный конструктор и методы никак не влияют на работу класса как класса набора тестов, поскольку этот класс служит просто хранилищем для аннотаций

5) Какие тестовые методы будут запущены при запуске набора тестов SecurityTests:

**public** **interface** UserTests {}

**public** **interface** RoleTests {}

@Category({UserTests.**class**})

**public** **class** UserManagerTest {

@Test

**public** **void** testAddUser() {

}

**public** **void** testRemoveUser() {

}

}

**public** **class** RoleManagerTest {

@Category({UserTests.**class**, RoleTests.**class**})

@Test

**public** **void** testAddRoleToUser() {

}

@Category({RoleTests.**class**})

@Test

**public** **void** testAddRole() {

}

@Test

**public** **void** testRemoveRole() {

}

}

@RunWith(Suite.**class**)

@IncludeCategory(UserTests.**class**)

@SuiteClasses( {

UserManagerTest.**class**,

RoleManagerTest.**class**

})

**public** **class** SecurityTests {

}

A1: testAddUser, testRemoveUser, testAddUserToRole, testAddRole, testRemoveRole

+A2: testAddUser, testAddUserToRole, testAddRole, testRemoveRole

A3: testAddUser, testRemoveUser

А4: testAddUser

А5: нет правильного ответа

Набор тестов объявлен с помощью аннотации @RunWith(Suite.**class**), значит категории учитываться не будут. Но при этом testRemoveUser метод не аннотирован аннотацией @Test и потому не является тестовым методом.

6) Какие тестовые методы будут запущены при запуске набора тестов SecurityTests:

**public** **interface** UserTests {}

**public** **interface** RoleTests {}

@Category({UserTests.**class**})

**public** **class** UserManagerTest {

@Test

**public** **void** testAddUser() {

}

**public** **void** testRemoveUser() {

}

}

**public** **class** RoleManagerTest {

@Category({UserTests.**class**, RoleTests.**class**})

@Test

**public** **void** testAddRoleToUser() {

}

@Category({RoleTests.**class**})

@Test

**public** **void** testAddRole() {

}

@Test

**public** **void** testRemoveRole() {

}

}

@RunWith(Categories.**class**)

@IncludeCategory(UserTests.**class**)

@SuiteClasses( {

UserManagerTest.**class**,

RoleManagerTest.**class**

})

**public** **class** SecurityTests {

}

A1: testAddUser, testRemoveUser, testAddUserToRole, testAddRole, testRemoveRole

A2: testAddUser, testRemoveUser

+A3: testAddUser, testAddUserToRole

А4: testAddUser

А5: нет правильного ответа

Набор тестов объявлен с помощью аннотации @RunWith(Categories.**class**), значит категории учитываться будут. Но при этом testRemoveUser метод не аннотирован аннотацией @Test и потому не является тестовым методом.

7) Какие тестовые методы будут запущены при запуске набора тестов SecurityTests:

**public** **interface** UserTests {}

**public** **interface** RoleTests {}

@Category({UserTests.**class**})

**public** **class** UserManagerTest {

@Test

**public** **void** testAddUser() {

}

**public** **void** testRemoveUser() {

}

}

**public** **class** RoleManagerTest {

@Category({UserTests.**class**, RoleTests.**class**})

@Test

**public** **void** testAddRoleToUser() {

}

@Category({RoleTests.**class**})

@Test

**public** **void** testAddRole() {

}

@Test

**public** **void** testRemoveRole() {

}

}

@RunWith(Categories.**class**)

@IncludeCategory(UserTests.**class**)

@ExcludeCategory(RoleTests.**class**)

@SuiteClasses( {

UserManagerTest.**class**,

RoleManagerTest.**class**

})

**public** **class** SecurityTests {

}

A1: testAddUser, testRemoveUser, testAddUserToRole, testAddRole, testRemoveRole

A2: testAddUser, testRemoveUser

A3: testAddUser, testAddUserToRole

+А4: testAddUser

А5: нет правильного ответа

Набор тестов объявлен с помощью аннотации @RunWith(Categories.**class**), значит категории учитываться будут. Но при этом testRemoveUser метод не аннотирован аннотацией @Test и потому не является тестовым методом, а testAddUserToRole явно исключен с помощью аннотации @ExcludeCategory

8) Какие тестовые методы будут запущены при запуске набора тестов SecurityTests:

**public** **interface** UserTests {}

**public** **interface** RoleTests {}

@Category({UserTests.**class**})

**public** **class** UserManagerTest {

@Test

**public** **void** testAddUser() {

}

**public** **void** testRemoveUser() {

}

}

**public** **class** RoleManagerTest {

@Category({UserTests.**class**, RoleTests.**class**})

@Test

**public** **void** testAddRoleToUser() {

}

@Category({RoleTests.**class**})

@Test

**public** **void** testAddRole() {

}

@Test

**public** **void** testRemoveRole() {

}

}

@RunWith(Categories.**class**)

@IncludeCategory({UserTests.**class,** RoleTests.**class}**)

@SuiteClasses( {

UserManagerTest.**class**,

RoleManagerTest.**class**

})

**public** **class** SecurityTests {

}

A1: testAddUser, testRemoveUser, testAddUserToRole, testAddRole, testRemoveRole

A2: testAddUser, testRemoveUser

A3: testAddUser, testAddUserToRole

А4: testAddUser

+А5: нет правильного ответа

данный код не скомпилируется, т. к. контракт @IncludeCategory аннотации не позволяет указать несколько категорий

9) Какие тестовые методы будут запущены при запуске набора тестов SecurityTests:

**public** **interface** UserTests {}

**public** **interface** RoleTests {}

@Category({UserTests.**class**})

**public** **class** UserManagerTest {

@Test

**public** **void** testAddUser() {

}

**public** **void** testRemoveUser() {

}

}

**public** **class** RoleManagerTest {

@Category({UserTests.**class**, RoleTests.**class**})

@Test

**public** **void** testAddRoleToUser() {

}

@Category({RoleTests.**class**})

@Test

**public** **void** testAddRole() {

}

@Test

**public** **void** testRemoveRole() {

}

}

@RunWith(Categories.**class**)

@IncludeCategory(RoleTests.**class**)

@SuiteClasses( {

UserManagerTest.**class**,

RoleManagerTest.**class**

})

**public** **class** SecurityTests {

}

A1: testAddUser, testRemoveUser, testAddUserToRole, testAddRole, testRemoveRole

A2: testAddUserToRole, testAddRole, testRemoveRole

+A3: testAddUserToRole, testAddRole

А4: testAddUserToRole

А5: нет правильного ответа

Набор тестов объявлен с помощью аннотации @RunWith(Categories.**class**), значит категории учитываться будут. Но при этом testRemoveRole метод не принадлежит какой-либо категории.

10) Какие тестовые методы будут запущены при запуске набора тестов SecurityTests:

**public** **interface** UserTests {}

**public** **interface** RoleTests {}

@Category({UserTests.**class**})

**public** **class** UserManagerTest {

@Test

**public** **void** testAddUser() {

}

**public** **void** testRemoveUser() {

}

}

**public** **class** RoleManagerTest {

@Category({UserTests.**class**, RoleTests.**class**})

@Test

**public** **void** testAddRoleToUser() {

}

@Category({RoleTests.**class**})

@Test

**public** **void** testAddRole() {

}

@Test

**public** **void** testRemoveRole() {

}

}

@IncludeCategory(UserTests.**class**)

@ExcludeCategory(RoleTests.**class**)

@SuiteClasses( {

UserManagerTest.**class**,

RoleManagerTest.**class**

})

**public** **class** SecurityTests {

}

A1: testAddUser, testRemoveUser, testAddUserToRole, testAddRole, testRemoveRole

A2: testAddUser, testRemoveUser

A3: testAddUser, testAddUserToRole

А4: testAddUser

+А5: нет правильного ответа

Набор тестов не помечен аннотацией @RunWith(Categories.**class**) и потому не считается классом набора тестов

# **4. Утверждения и предположения**

# 1) Каким будет результат прогона следующего теста:

1. **public** **class** Question1 {
2. @Test
3. **public** **void** testMethod() {
4. Assert.*assertEquals*("Операция сложения работает неверно", 8, **new** Calculator().sum(3, 5));
5. }
7. **class** Calculator {
8. **int** sum(**int** a, **int** b) {
9. **return** a + b;
10. }
11. }
12. }
13. +A1: Выполнится успешно
14. A2: Тест откажет
15. A3: Возникнет ошибка компиляции

# 1) Каким будет результат прогона следующего теста:

1. **public** **class** Question1 {
2. @Test
3. **public** **void** testMethod() {
4. Assert.*assertEquals*("Операция сложения работает неверно", 8, **new** Calculator().sum(3, 5));
5. }
7. **class** Calculator {
8. **int** sum(**int** a, **int** b) {
9. **return** a + b;
10. }
11. }
12. }
13. +A1: Выполнится успешно
14. A2: Тест откажет
15. A3: Возникнет ошибка компиляции
16. 2) Каким будет результат прогона следующего теста:
17. **public** **class** Question1 {
18. @Test
19. **public** **void** testMethod() {
20. Assert.*assertEquals*("Операция сложения работает неверно", 8, **new** Calculator().sum(3, 5));
21. }
23. **class** Calculator {
24. **int** sum(**int** a, **int** b) {
25. **return** a + a;
26. }
27. }
28. }
29. A1: Выполнится успешно
30. +A2: Тест откажет
31. A3: Возникнет ошибка компиляции
32. Тест откажет т. к. реализация калькулятора неверна — вместо сложения a + b он делает сложение a + a.

# 3) Каким будет результат прогона следующего теста:

1. **public** **class** Question1 {
2. @Test
3. **public** **void** testMethod() {
4. Assert.*assertEquals*(8, **new** Calculator().sum(3, 5), "Операция сложения работает неверно");
5. }
7. **class** Calculator {
8. **int** sum(**int** a, **int** b) {
9. **return** a + b;
10. }
11. }
12. }
13. A1: Выполнится успешно
14. A2: Тест откажет
15. +A3: Возникнет ошибка компиляции
16. Контракт методов assertEquals выглядит так: assertEquals(String message, int expected, int actual)

# 4) Каким будет результат прогона следующего теста:

1. **public** **class** Question2 {
2. @Test
3. **public** **void** testName() {
4. Person p1 = **new** Person("Alex");
5. Person p2 = **new** Person("Alex");
6. Assert.*assertEquals*(p1, p2);
7. }

10. **public** **class** Person {
11. **private** String name;
13. **public** Person(String name) {
14. **this**.name = name;
15. }
17. **public** String getName() {
18. **return** name;
19. }
20. }
21. }
22. A1: Выполнится успешно
23. +A2: Тест откажет
24. A3: Возникнет ошибка компиляции
25. У класса Person не реализован метод equals. assertEquals в таком случае будет использовать метод из класса Object который сравнивает по идентичности. Поскольку два экземпляра не идентичны, тест откажет

# 5) Каким будет результат прогона следующего теста:

1. **public** **class** Question2 {
2. @Test
3. **public** **void** testName() {
4. Person p1 = **new** Person("Alex");
5. Person p2 = **new** Person("Alex");
6. Assert.*assertNotSame*(p1, p2);
7. }

10. **public** **class** Person {
11. **private** String name;
13. **public** Person(String name) {
14. **this**.name = name;
15. }
17. **public** String getName() {
18. **return** name;
19. }
20. }
21. }
22. +A1: Выполнится успешно
23. A2: Тест откажет
24. A3: Возникнет ошибка компиляции
25. Поскольку создано два экземпляра класса Person, то сравнение по идентичности скажет что это два разных объекта. А поскольку assertNotSame – это утверждение вида «Утверждаю что два объекта не идентичны», то оно будет истинным и тест выполнится успешно.

# 5) Каким будет результат прогона следующего теста:

1. **public** **class** Question2 {
2. @Test
3. **public** **void** testName() {
4. Person p1 = **new** Person("Alex");
5. Person p2 = **new** Person("Alex");
6. Assert.*assertNotSame*(p1, p2);
7. }

10. **public** **class** Person {
11. **private** String name;
13. **public** Person(String name) {
14. **this**.name = name;
15. }
17. **public** String getName() {
18. **return** name;
19. }
20. }
21. }
22. +A1: Выполнится успешно
23. A2: Тест откажет
24. A3: Возникнет ошибка компиляции
25. Поскольку создано два экземпляра класса Person, то сравнение по идентичности скажет что это два разных объекта. А поскольку assertNotSame – это утверждение вида «Утверждаю что два объекта не идентичны», то оно будет истинным и тест выполнится успешно.

# 6) Каким будет результат прогона следующего теста:

1. **public** **class** Question {
2. @Test
3. **public** **void** testName() {
4. Person p1 = **new** Person("Alex", "Smith");
5. Person p2 = **new** Person("Alex", "Smith");
6. Assert.*assertEquals*(p1, p2);
7. }
9. **public** **class** Person {
10. **private** String name;
11. **private** String lastName;
13. **public** Person(String name, String lastName) {
14. **this**.name = name;
15. **this**.lastName = lastName;
16. }
18. **public** String getName() {
19. **return** name;
20. }
21. **public** String getLastName() {
22. **return** lastName;
23. }
25. **public** **boolean** equals(Person p) {
26. **return** name.equals(p.name);
27. }
28. }
29. }
30. A1: Выполнится успешно
31. +A2: Тест откажет
32. A3: Возникнет ошибка компиляции
33. Несмотря на то что создано два объекта с одинаковыми именем и фамилией, тест откажет. Это произойдет потому что нарушен контракт метода equals – объявленный в классе Object метод equals принимает параметр класса Object, поэтому метод equals объявленный в классе Person не переопределяет (override) а перегружает (overload) этот метод. Assert.assertEquals при этом будет использовать непереопределенную версию equals из класса Object, которая сравнивает идентичность объектов. А эти два объекта не идентичны, поэтому тест откажет.

# 7) Каким будет результат прогона следующего теста:

1. **public** **class** Question {
2. @Test
3. **public** **void** testName() {
4. Person p1 = **new** Person("Alex", "Smith");
5. Person p2 = **new** Person("Alex", "Smith");
6. Assert.*assertEquals*(p1, p2);
7. }
9. **public** **class** Person {
10. **private** String name;
11. **private** String lastName;
13. **public** Person(String name, String lastName) {
14. **this**.name = name;
15. **this**.lastName = lastName;
16. }
18. **public** String getName() {
19. **return** name;
20. }

**public** String getLastName() {

**return** lastName;

}

2. **public** **boolean** equals(Object p) {
3. **return** name.equals(p.name);
4. }
5. }
6. }
7. A1: Выполнится успешно
8. A2: Тест откажет
9. +A3: Возникнет ошибка компиляции
10. В методе equals необходимо сначала привести параметр p к классу Person и только после этого можно получать доступ к полю name.

8) Каким будет результат прогона следующего теста:

1. **public** **class** Question {
2. @Test
3. **public** **void** testName() {
4. Person p1 = **new** Person("Alex", "Smith");
5. Person p2 = **new** Person("Alex", "Smith");
6. Assert.*assertEquals*(p1, p2);
7. }
9. **public** **class** Person {
10. **private** String name;
11. **private** String lastName;
13. **public** Person(String name, String lastName) {
14. **this**.name = name;
15. **this**.lastName = lastName;
16. }
18. **public** String getName() {
19. **return** name;
20. }

**public** String getLastName() {

**return** lastName;

}

2. **public** **boolean** equals(Object p) {
3. **return** name.equals(p.name);
4. }
5. }
6. }
7. +A1: Выполнится успешно
8. A2: Тест откажет
9. A3: Возникнет ошибка компиляции
10. Несмотря на то, что создаются два объекта с одним именем, но разной фамилией, метод equals при сравнении учитывает только имя. А Assert.assertEquals для сравнения использует именно метод equals.

9) Каким будет результат прогона следующего теста:

1. **public** **class** Question {
2. @Test
3. **public** **void** testMethod() {
4. String message = (String) **null**;
5. Assert.*assertNotNull*(message);
6. Assume.*assumeNotNull*(message);
7. }
8. }
9. A1: Выполнится успешно
10. +A2: Тест откажет
11. A3: Возникнет ошибка компиляции
12. A4: Тест будет пропущен (из-за использования assumeNotNull)
13. Несмотря на то, что в тесте используется assumeNotNull, который при передаче null значения в качестве параметра помечает тестовый метод как пропущенный, выполнение просто не дойдет до вызова этого метода. Тест откажет при проверке утверждения assertNotNull(message) которое будет ложным.

10) Каким будет результат прогона следующего теста:

1. **public** **class** Question {
2. @Test
3. **public** **void** testMethod() {
4. String message = (String) **null**;
   * + 1. Assume.*assumeNotNull*(message);
5. Assert.*assertNotNull*(message);
6. }
7. }
8. A1: Выполнится успешно
9. A2: Тест откажет
10. A3: Возникнет ошибка компиляции
11. +A4: Тест будет пропущен (из-за использования assumeNotNull)
12. В тесте используется assumeNotNull, который при передаче null значения в качестве параметра помечает тестовый метод как пропущенный. Дальше выполнение не пойдет и метод assertNotNull просто не будет вызван.