Consent

We invite you to join our study which investigates the understandability of automatically generated unit tests. This survey compares the understandability of automatically generated test cases and manually written test cases. The study aims to collect data from participants about their perception of test cases to determine whether they find one type of test case more understandable than the other. The survey will take you approximately 30 minutes to complete.

Procedure: You will be asked to read and evaluate a set of test cases. The test cases will be presented in two rounds: first, compare the understandability of the test cases of three groups: two groups of the tests are generated automatically, and one group is written manually. In the second round, you will be asked to rate the understandability of different parts of the test case on a scale from 1 to 5.

Risks and Benefits: We believe that participating in this study poses no risk to you. By participating in this study, you are supporting to research on the understandability of test cases and the potential to improve software development practices.

Confidentiality: Your participation in this study is confidential. Your data will be stored securely and only accessible to the research team. Data will be reported in aggregate form and individual responses will not be identified.

Voluntary Participation: Participation in this study is voluntary. You have the right to withdraw at any time without penalty or loss of benefits to which you are otherwise entitled.

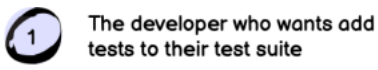
Consent: By participating in this study, you acknowledge that you have read and understood the information provided. You agree to participate voluntarily and understand that you may withdraw at any time without penalty or loss of benefits to which you are otherwise entitled.

Thank you for your participation in our study!

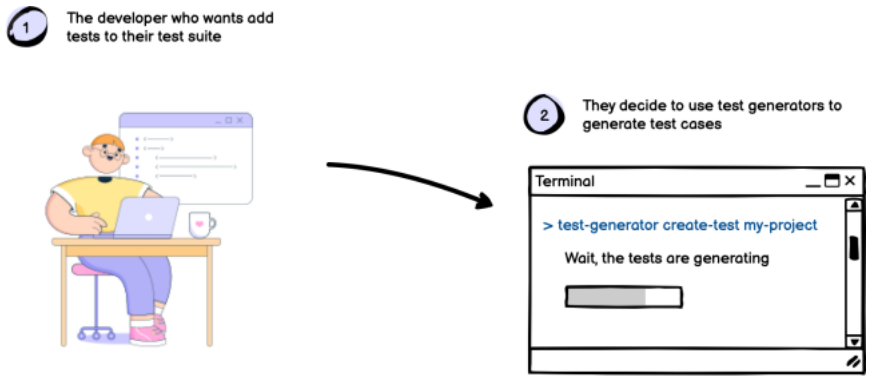
○ I consent to participate in this survey

First round - Intro

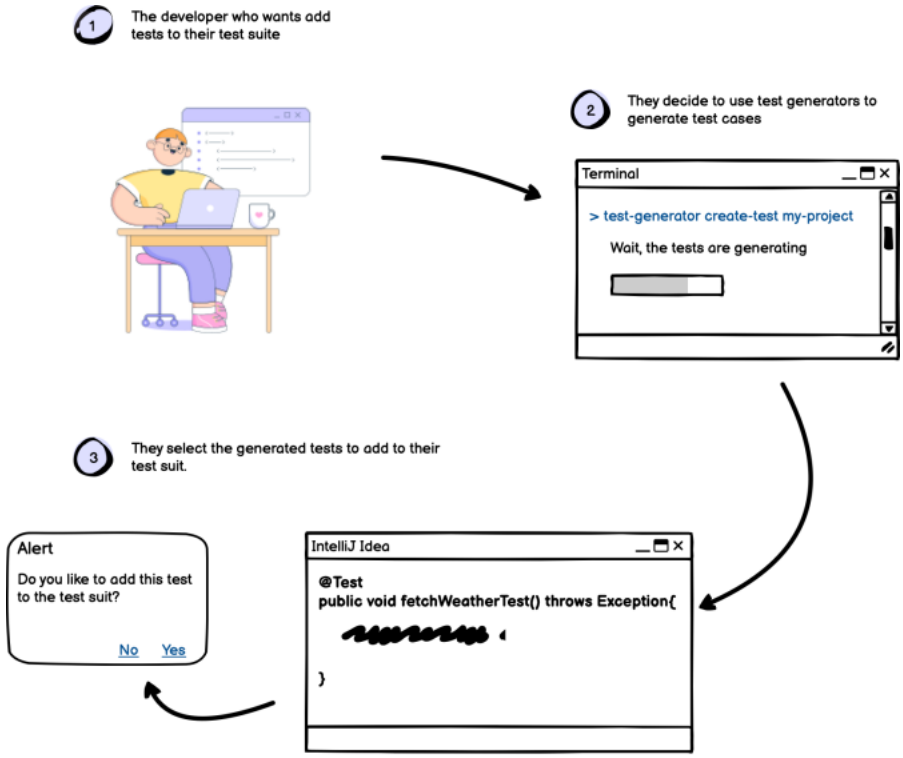
1. Suppose you are a developer who is working on a system that only has a few tests for each component, so you decide to improve your test suite.



2. You first attempted to write the tests manually, but you realized it would take a long time to do it. Then you decided to generate tests with test generators.



3. You want to select among tests generated by those test generators to add the test suite. You think that understanding the logic behind the tests is important because you want to make sure the main logic of the system is being tested and the generated tests are not redundant with existing tests.



4. To facilitate the lives of developers, we would greatly appreciate your assistance in identifying the test cases that require less time or effort to understand. So, please choose the test which is more understandable among the tests that are generated by the test generators, and/or manually written tests.

Compare w/ EvoSuite 1 [petclinic]

Which test do you think is more understandable? (find the class under test *here*)

```
@Test
public void testGetTypeReturningPetTypeWhereIsNewIsFalse()  throws Throwable
   Pet pet0 = new Pet();
   PetType petType0 = new PetType();
   Integer integer0 = new Integer(1);
   petType0.setId(integer0);
   pet0.setType(petType0);
   PetType petType1 = pet0.getType();
   assertEquals(1, (int)petType1.getId());
}
```

```
@BeforeEach
public void setUp() throws Exception {
   subject = new Pet();

   PetType petType = new PetType();
   petType.setId(1);
   petType.setName("cat");

   subject.setType(petType);
   subject.setBirthDate(LocalDate.parse("2010-09-07"));
   subject.setId(1);
   subject.setName("Leo");
}

@Test
public void getTypeTest() throws Exception {
   PetType getType = subject.getType();

   PetType petType = new PetType();
   petType.setId(1);
   petType.setName("cat");

   assertThat(getType, is(petType));
}
```

What makes this test more understandable? (Highlight text)

```
@BeforeEach
public void setUp() throws Exception {
  subject = new Pet();

  PetType petType = new PetType();
  petType.setId(1);
  petType.setName("cat");

  subject.setType(petType);
  subject.setId(1);
  subject.setName("Leo");
}

@Test
public void getTypeTest() throws Exception {
  PetType getType = subject.getType();

  PetType petType = new PetType();
  petType.setId(1);
  petType.setName("cat");
```

```
  assertThat(getType, is(petType));
}}
```

What makes this test more understandable? (Highlight text)

```
@Test
public void testGetTypeReturningPetTypeWhereIsNewIsFalse()  throws Throwable
  Pet pet0 = new Pet();
  PetType petType0 = new PetType();
  Integer integer0 = new Integer(1);
  petType0.setId(integer0);
  pet0.setType(petType0);
  PetType petType1 = pet0.getType();
  assertEquals(1, (int)petType1.getId());
}}
```

Can you explain why you think the selected test is more understandable?

Compare w/ EvoSuite 2 [spring-testing]

Which test do you think is more understandable? (find the class under test *here*)

```
@BeforeEach
public void setUp() throws Exception {
    subject = new Person("james", "carter");
}
@Test
public void getFirstNameTest() throws Exception{
    String getFirstName = subject.getFirstName();

    assertThat(getFirstName, is("james"));
}
```

```
@Test(timeout = 4000)
public void testGetFirstNameReturningNonEmptyString()  throws Throwable  {
    Person person0 = new Person("Kt(tyZA._U*Ce4A", "OM&}X$I!~QDSaIxi");
    person0.getFirstName();
    assertEquals("Person{id='0', firstName='Kt(tyZA._U*Ce4A', lastName='OM&}
}
```

What makes this test more understandable? (Highlight text)

```
@BeforeEach
public void setUp() throws Exception {
subject = new Person("james", "carter");
}
@Test
public void getFirstNameTest() throws Exception{
String getFirstName = subject.getFirstName();
```

```
assertThat(getFirstName, is("james"));
}}
```

What makes this test more understandable? (Highlight text)

```
@Test(timeout = 4000)
public void testGetFirstNameReturningNonEmptyString() throws Throwable {
Person person0 = new Person("Kt(tyZA._U*Ce4A", "OM&}X$I!~QDSaIxi");
person0.getFirstName();
assertEquals("Person{id='0', firstName='Kt(tyZA._U*Ce4A',
lastName='OM&}X$I!~QDSaIxi'}", person0.toString());
}}
```

Can you explain why you think the selected test is more understandable?

Compare w/ EvoSuite 3 [spring-testing]

Which test do you think is more understandable? (find the class under test *here*)

```
@BeforeEach
public void setUp() throws Exception {
    subject = new Person("james", "carter");
}
@Test
public void getLastNameTest() throws Exception{
    String getLastName = subject.getLastName();

    assertThat(getLastName, is("carter"));
}
```

```
@Test(timeout = 4000)
public void testGetLastNameReturningEmptyString()  throws Throwable  {
    Person person0 = new Person("", "");
    String string0 = person0.getLastName();
    assertEquals("", string0);
}
```

What makes this test more understandable? (Highlight text)

```
@BeforeEach
public void setUp() throws Exception {
   subject = new Person("james", "carter");
}
@Test
public void getLastNameTest() throws Exception{
   String getLastName = subject.getLastName();

   assertThat(getLastName, is("carter"));
}}
```

What makes this test more understandable? (Highlight text)

```
@Test(timeout = 4000)
public void testGetLastNameReturningEmptyString()  throws Throwable  {
   Person person0 = new Person("", "");
   String string0 = person0.getLastName();
   assertEquals("", string0);
}}
```

Can you explain why you think the selected test is more understandable?

Compare w/ EvoSuite 4 [spring-testing]

Which test do you think is more understandable? (find the class under test *here*)

```
@Test
public void testGetSummary() throws Throwable {
    WeatherResponse weatherResponse0 = new WeatherResponse("Jy(+XI9N", "Jy(+
    String string0 = weatherResponse0.getSummary();
    assertEquals("Jy(+XI9N: Jy(+XI9N", string0);
}
```

```
@Test
public void getSummaryTest() throws Exception{
    subject = new WeatherResponse("Clear", "clear sky");
    String getSummary = subject.getSummary();
    assertThat(getSummary, is("Clear: clear sky"));
}
```

What makes this test more understandable? (Highlight text)

```
@Test
public void testGetSummary() throws Throwable {
   WeatherResponse weatherResponse0 = new WeatherResponse("Jy(+XI9N",
"Jy(+XI9N");
   String string0 = weatherResponse0.getSummary();
   assertEquals("Jy(+XI9N: Jy(+XI9N", string0);
}}
```

What makes this test more understandable? (Highlight text)

```
@Test
public void getSummaryTest() throws Exception{
   subject = new WeatherResponse("Clear", "clear sky");
   String getSummary = subject.getSummary();
```

```
    assertThat(getSummary, is("Clear: clear sky"));
}}
```

Can you explain why you think the selected test is more understandable?

Compare w/ EvoSuite 5 [petclinic]

Which test do you think is more understandable? (find the class under test *here*)

```
@Test
public void testGetVisits()  throws Throwable  {
   Pet pet0 = new Pet();
   Collection<Visit> collection0 = pet0.getVisits();
   assertNotNull(collection0);
}
```

```
@Test
public void getVisitsTest() throws Exception{
    PersistentSet getVisits = subject.getVisits();

    Visit visit = new Visit();
    visit.setDescription("rabies shot");
    visit.setId(2);
    PersistentSet<Visit> visits = new PersistentSet<>();
    visits.add(visit);

    assertThat(getVisits, is(visits));
}
```

What makes this test more understandable? (Highlight text)

```
@Test
public void testGetVisits() throws Throwable {
Pet pet0 = new Pet();
Collection<Visit> collection0 = pet0.getVisits();
assertNotNull(collection0);assertNotNull(collection0);
```

What makes this test more understandable? (Highlight text)

```
@Test
public void getVisitsTest() throws Exception{
   PersistentSet getVisits = subject.getVisits();

   Visit visit = new Visit();
   visit.setDescription("rabies shot");
   visit.setId(2);
   PersistentSet<Visit> visits = new PersistentSet<>();
   visits.add(visit);

   assertThat(getVisits, is(visits));is(visits));
```

Can you explain why you think the selected test is more understandable?

Compare w/ EvoSuite 6 [petclinic]

Which test do you think is more understandable? (find the class under test *here*)

```
@BeforeEach
public void setUp() throws Exception {
    subject = new Pet();
    subject.setBirthDate(LocalDate.parse("2002-08-06"));
    subject.setId("2");
    subject.setName("Basil");
    PetType petType = new PetType();
    petType.setId(2);
    petType.setName("dog");
    subject.setType(petType);
}

@Test
public void getBirthDateTest() throws Exception{
    LocalDate getBirthDate = subject.getBirthDate();

    assertThat(getBirthDate, is(LocalDate.parse("2002-08-06")));
}
```

○

```
@Test
public void testGetBirthDateReturningNull()  throws Throwable  {
    Pet pet0 = new Pet();
    LocalDate localDate0 = pet0.getBirthDate();
    assertNull(localDate0);
}
```

○

What makes this test more understandable? (Highlight text)

```
@BeforeEach
public void setUp() throws Exception {
   subject = new Pet();
   subject.setBirthDate(LocalDate.parse("2002-08-06"));
   subject.setId("2");
   subject.setName("Basil");
   PetType petType = new PetType();
   petType.setId(2);
   petType.setName("dog");
   subject.setType(petType);
}

@Test
public void getBirthDateTest() throws Exception{
   LocalDate getBirthDate = subject.getBirthDate();

   assertThat(getBirthDate, is(LocalDate.parse("2002-08-
06")));is(LocalDate.parse("2002-08-06")));
```

What makes this test more understandable? (Highlight text)

```
@Test
public void testGetBirthDateReturningNull()  throws Throwable  {
   Pet pet0 = new Pet();
```

```
LocalDate localDate0 = pet0.getBirthDate();
assertNull(localDate0);assertNull(localDate0);
```

Can you explain why you think the selected test is more understandable?

Compare w/ EvoSuite 7 [petclinic]

Which test do you think is more understandable? (find the class under test *here*)

```
@Test
public void getPetsTest() throws Exception
    PersistentBag getPets = subject.getPets();

    PetType petType = new PetType();
    petType.setId(1);
    petType.setName("cat");
    Pet pet = new Pet();
    pet.addVisit(visit_1);
    pet.setId(1);
    pet.setType(petType_1);
    pet.setBirthDate(LocalDate.parse("2000-09-07"));
    pet.setName("Leo");
    PersistentBag<Pet> pets = new PersistentBag<>();
    pets.add(pet);

    assertThat(getPets, is(pets));
}
```

```
@Test
public void testGetPetsReturningListWhereIsEmptyIsTrueAndListWhereSizeIsZer
    Owner owner0 = new Owner();
    List<Pet> list0 = owner0.getPets();
    assertTrue(list0.isEmpty());
}
```

What makes this test more understandable? (Highlight text)

```
@Test
public void getPetsTest() throws Exception
    PersistentBag getPets = subject.getPets();

    PetType petType = new PetType();
    petType.setId(1);
    petType.setName("cat");
    Pet pet = new Pet();
    pet.addVisit(visit_1);
    pet.setId(1);
    pet.setType(petType_1);
    pet.setBirthDate(LocalDate.parse("2000-09-07"));
    pet.setName("Leo");
    PersistentBag<Pet> pets = new PersistentBag<>();
    pets.add(pet);

    assertThat(getPets, is(pets));assertThat(getPets, is(pets));
```

What makes this test more understandable? (Highlight text)

```
@Test
public void
testGetPetsReturningListWhereIsEmptyIsTrueAndListWhereSizeIsZero()  throws
Throwable  {
   Owner owner0 = new Owner();
   List<Pet> list0 = owner0.getPets();
   assertTrue(list0.isEmpty());assertTrue(list0.isEmpty());
```

Can you explain why you think the selected test is more understandable?

Compare w/ EvoSuite 8 [petclinic]

Which test do you think is more understandable? (find the class under test *here*)

```
@Test
public void getTelephoneWhereTest() throws Exception{
    subject.setCity("Sun Prairie");
    subject.setId("2");
    subject.setAddress("638 Cardinal Ave.");
    subject.setTelephone("6085551749");


    String getTelephone = subject.getTelephone();

    assertThat(getTelephone, is("6085551749"));
}
```

```
@Test
public void testGetTelephoneReturningNonEmptyString()  throws Throwable  {
    Owner owner0 = new Owner();
    owner0.setTelephone("RUNTIME_OR_ERROR");
    String string0 = owner0.getTelephone();
    assertEquals("RUNTIME_OR_ERROR", string0);
}
```

What makes this test more understandable? (Highlight text)

```
@Test
public void getTelephoneWhereTest() throws Exception{
   subject.setCity("Sun Prairie");
   subject.setId("2");
   subject.setAddress("638 Cardinal Ave.");
   subject.setTelephone("6085551749");

   String getTelephone = subject.getTelephone();

   assertThat(getTelephone, is("6085551749"));is("6085551749"));
```

What makes this test more understandable? (Highlight text)

```
@Test
public void testGetTelephoneReturningNonEmptyString()  throws Throwable  {
    Owner owner0 = new Owner();
    owner0.setTelephone("RUNTIME_OR_ERROR");
    String string0 = owner0.getTelephone();
    assertEquals("RUNTIME_OR_ERROR", string0);string0);
```

Can you explain why you think the selected test is more understandable?

Compare w/ EvoSuite 9 [petclinic]

Which test do you think is more understandable? (find the class under test *here*)

```
@Test
public void getAddressWhereTest() throws Exception{
    subject.setCity("Sun Prairie");
    subject.setId("2");
    subject.setAddress("638 Cardinal Ave.");
    subject.setTelephone("6085551749");

    String getAddress = subject.getAddress();

    assertThat(getAddress, is("638 Cardinal Ave."));
}
```

```
@Test
public void testGetAddressReturningNonEmptyString()  throws Throwable  {
    Owner owner0 = new Owner();
    owner0.setAddress("4Qu\"zgps_#c_UTxU2");
    String string0 = owner0.getAddress();
    assertEquals("4Qu\"zgps_#c_UTxU2", string0);
}
```

What makes this test more understandable? (Highlight text)

```
@Test
public void getAddressWhereTest() throws Exception{
    subject.setCity("Sun Prairie");
    subject.setId("2");
    subject.setAddress("638 Cardinal Ave.");
    subject.setTelephone("6085551749");

    String getAddress = subject.getAddress();

    assertThat(getAddress, is("638 Cardinal Ave."));Ave."));
```

What makes this test more understandable? (Highlight text)

```
@Test
public void testGetAddressReturningNonEmptyString()  throws Throwable  {
    Owner owner0 = new Owner();
    owner0.setAddress("4Qu\"zgps_#c_UTxU2");
```

```
String string0 = owner0.getAddress();
assertEquals("4Qu\"zgps_#c_UTxU2", string0);string0);
```

Can you explain why you think the selected test is more understandable?

Compare w/ EvoSuite 10 [petclinic]

Which test do you think is more understandable? (find the class under test *here*)

```
@Test
public void getCityWhereTest() throws Exception{
     subject.setCity("Sun Prairie");
     subject.setId("2");
     subject.setAddress("638 Cardinal Ave.");
     subject.setTelephone("6085551749");

     String getCity = subject.getCity();

     assertThat(getCity, is("Sun Prairie"));
}
```

○

```
@Test
public void testGetCityReturningNonEmptyString()  throws Throwable  {
    Owner owner0 = new Owner();
    owner0.setCity(".^Rkx");
    String string0 = owner0.getCity();
    assertEquals(".^Rkx", string0);
}
```

○

What makes this test more understandable? (Highlight text)

```
@Test
public void getCityWhereTest() throws Exception{
    subject.setCity("Sun Prairie");
    subject.setId("2");
    subject.setAddress("638 Cardinal Ave.");
    subject.setTelephone("6085551749");

    String getCity = subject.getCity();

    assertThat(getCity, is("Sun Prairie"));Prairie"));
```

What makes this test more understandable? (Highlight text)

```
@Test
public void testGetCityReturningNonEmptyString() throws Throwable {
Owner owner0 = new Owner();
owner0.setCity(".^Rkx");
String string0 = owner0.getCity();
assertEquals(".^Rkx", string0);string0);
```

Can you explain why you think the selected test is more understandable?

Compare w/ EvoSuite 11 [Insurance]

Which test do you think is more understandable? (find the class under test *here*)

```
@BeforeEach
public void setUp() throws Exception {
    ArrayList<ChoiceDto> choices = new ArrayList<>();
    choices.add(new ChoiceDto("ZB", "Crop"));
    choices.add(new ChoiceDto("KW", "Vegetable"));

    subject = new ChoiceQuestionDto(1, "Cultivation type", "TYP", choices);
}

@Test
public void getChoicesTest() throws Exception{
    ArrayList getChoices = subject.getChoices();

    ArrayList<ChoiceDto> choices = new ArrayList<>();
    choices.add(new ChoiceDto("ZB", "Crop"));
    choices.add(new ChoiceDto("KW", "Vegetable"));

    assertThat(getChoices, is(choices));
}
```

```
@Test
@Timeout(value = 4000)
public void testGetChoicesReturningListWhereIsEmptyIsTrueAndListWhereSizeIs
   Vector<ChoiceDto> vector0 = new Vector<ChoiceDto>();
   ChoiceQuestionDto choiceQuestionDto0 = new ChoiceQuestionDto("Zp3{5Qe}mt
   List<ChoiceDto> list0 = choiceQuestionDto0.getChoices();
   assertTrue(list0.isEmpty());
}
```

What makes this test more understandable? (Highlight text)

```
@BeforeEach
public void setUp() throws Exception {
   ArrayList<ChoiceDto> choices = new ArrayList<>();
   choices.add(new ChoiceDto("ZB", "Crop"));
   choices.add(new ChoiceDto("KW", "Vegetable"));

   subject = new ChoiceQuestionDto(1, "Cultivation type", "TYP", choices);
}

@Test
public void getChoicesTest() throws Exception{
   ArrayList getChoices = subject.getChoices();

   ArrayList<ChoiceDto> choices = new ArrayList<>();
   choices.add(new ChoiceDto("ZB", "Crop"));
   choices.add(new ChoiceDto("KW", "Vegetable"));

   assertThat(getChoices, is(choices));is(choices));
```

What makes this test more understandable? (Highlight text)

```
@Test
@Timeout(value = 4000)
public void
testGetChoicesReturningListWhereIsEmptyIsTrueAndListWhereSizeIsZero()
throws Throwable  {
  Vector<ChoiceDto> vector0 = new Vector<ChoiceDto>();
  ChoiceQuestionDto choiceQuestionDto0 = new
ChoiceQuestionDto("Zp3{5Qe}mtJ()\"o+", 0, "en&=#b/", vector0);
  List<ChoiceDto> list0 = choiceQuestionDto0.getChoices();
  assertTrue(list0.isEmpty());assertTrue(list0.isEmpty());
```

Can you explain why you think the selected test is more understandable?

Compare w/ EvoSuite 12 [Insurance]

Which test do you think is more understandable? (find the class under test *here*)

```
@BeforeEach
public void setUp() throws Exception {
    subject = new ChoiceDto("Crop", "ZB");
}

@Test
public void getCodeTest() throws Exception{
    String getCode = subject.getCode();

    assertThat(getCode, is("ZB"));
}
```

```
@Test
@Timeout(value = 4000 , unit = TimeUnit.MILLISECONDS)
public void testGetCodeReturningNonEmptyString()  throws Throwable  {
   ChoiceDto choiceDto0 = new ChoiceDto("A)21 h}gQ_h P6I>V", "A)21 h}gQ_h P6
   String string0 = choiceDto0.getCode();
   assertEquals("A)21 h}gQ_h P6I>V", string0);
}
```

What makes this test more understandable? (Highlight text)

```
@BeforeEach
public void setUp() throws Exception {
   subject = new ChoiceDto("Crop", "ZB");
}

@Test
public void getCodeTest() throws Exception{
   String getCode = subject.getCode();

   assertThat(getCode, is("ZB"));is("ZB"));
```

What makes this test more understandable? (Highlight text)

```
@Test
@Timeout(value = 4000 , unit = TimeUnit.MILLISECONDS)
public void testGetCodeReturningNonEmptyString()  throws Throwable  {
  ChoiceDto choiceDto0 = new ChoiceDto("A)21 h}gQ_h P6I>V", "A)21 h}gQ_h
P6I>V");
  String string0 = choiceDto0.getCode();
  assertEquals("A)21 h}gQ_h P6I>V", string0);string0);
```

Can you explain why you think the selected test is more understandable?

Compare w/ EvoSuite 13 [Insurance]

Which test do you think is more understandable? (find the class under test *here*)

```
@BeforeEach
public void setUp() throws Exception {
    subject = new ChoiceDto("Crop", "ZB");
}

@Test
public void getLabelTest() throws Exception{
    String getLabel = subject.getLabel();

    assertThat(getLabel, is("Crop"));
}
```

○

```
@Test
@Timeout(value = 4000 , unit = TimeUnit.MILLISECONDS)
public void testGetLabelReturningEmptyString()  throws Throwable  {
    ChoiceDto choiceDto0 = new ChoiceDto("", "");
    String string0 = choiceDto0.getLabel();
    assertEquals("", string0);
}
```

○

What makes this test more understandable? (Highlight text)

```
@BeforeEach
public void setUp() throws Exception {
  subject = new ChoiceDto("Crop", "ZB");
}

@Test
public void getLabelTest() throws Exception{
  String getLabel = subject.getLabel();

  assertThat(getLabel, is("Crop"));is("Crop"));
```

What makes this test more understandable? (Highlight text)

```
@Test
@Timeout(value = 4000 , unit = TimeUnit.MILLISECONDS)
public void testGetLabelReturningEmptyString()  throws Throwable  {
  ChoiceDto choiceDto0 = new ChoiceDto("", "");
```

```
String string0 = choiceDto0.getLabel();
assertEquals("", string0);string0);
```

Can you explain why you think the selected test is more understandable?

Compare w/ EvoSuite 14 [Insurance]

Which test do you think is more understandable? (find the class under test *here*)

```
@BeforeEach
public void setUp() throws Exception {
    subject = new NumericQuestionDto("Number of adults", "NUM_OF_ADULTS");
}

@Test
public void getTextTest() throws Exception{
    String getText = subject.getText();

    assertThat(getText, is("Number of adults"));
}
```

```
@Test
@Timeout(value = 4000 , unit = TimeUnit.MILLISECONDS)
public void testCreatesNumericQuestionDtoTaking3Arguments()  throws Throwabl
    NumericQuestionDto numericQuestionDto0 = new NumericQuestionDto("%>%", (
    assertEquals("%>%", numericQuestionDto0.getText());
}
```

What makes this test more understandable? (Highlight text)

```
@BeforeEach
public void setUp() throws Exception {
   MockitoAnnotations.openMocks(this);
   subject = new NumericQuestionDto("Number of adults",
"NUM_OF_ADULTS");
}

@Test
public void getTextTest() throws Exception{
   String getText = subject.getText();

   assertThat(getText, is("Number of adults"));adults"));
```

What makes this test more understandable? (Highlight text)

```
@Test
@Timeout(value = 4000 , unit = TimeUnit.MILLISECONDS)
public void testCreatesNumericQuestionDtoTaking3Arguments()  throws
Throwable  {
   NumericQuestionDto numericQuestionDto0 = new
NumericQuestionDto("%>%", 0, "%>%");
   assertEquals("%>%",
numericQuestionDto0.getText());numericQuestionDto0.getText());
```

Can you explain why you think the selected test is more understandable?

Compare w/ EvoSuite 15 [Insurance]

Which test do you think is more understandable? (find the class under test *here*)

```
@BeforeEach
public void setUp() throws Exception {
    subject = new CalculatePriceCommand();
    subject.setPolicyFrom(LocalDate.parse("2023-05-17"));
    subject.setProductCode("FAI");
}

@Test
public void getPolicyFromTest() throws Exception{
    LocalDate getPolicyFrom = subject.getPolicyFrom();

    assertThat(getPolicyFrom, is(LocalDate.parse("2023-05-17")));
}<
/code>
```

```
@Test
  @Timeout(value = 4000 , unit = TimeUnit.MILLISECONDS)
  public void testPolicyFrom()  throws Throwable  {
      CalculatePriceCommand.CalculatePriceCommandBuilder calculatePriceComma
      Clock clock0 = MockClock.systemUTC();
      LocalDate localDate0 = MockLocalDate.now(clock0);
      CalculatePriceCommand.CalculatePriceCommandBuilder calculatePriceComma
      CalculatePriceCommand calculatePriceCommand0 = calculatePriceCommand_(
      LocalDate localDate1 = calculatePriceCommand0.getPolicyFrom();
      assertSame(localDate1, localDate0);
  }
```

What makes this test more understandable? (Highlight text)

```
@BeforeEach
public void setUp() throws Exception {
   MockitoAnnotations.openMocks(this);
   subject = new CalculatePriceCommand();
   subject.setPolicyFrom(LocalDate.parse("2023-05-17"));
}

@Test
public void getPolicyFromTest() throws Exception{
   LocalDate getPolicyFrom = subject.getPolicyFrom();

   assertThat(getPolicyFrom, is(LocalDate.parse("2023-05-
17")));is(LocalDate.parse("2023-05-17")));
```

What makes this test more understandable? (Highlight text)

```
@Test
  @Timeout(value = 4000 , unit = TimeUnit.MILLISECONDS)
  public void testPolicyFrom()  throws Throwable  {
     CalculatePriceCommand.CalculatePriceCommandBuilder
```

```
calculatePriceCommand_CalculatePriceCommandBuilder0 =
CalculatePriceCommand.builder();
    Clock clock0 = MockClock.systemUTC();
    LocalDate localDate0 = MockLocalDate.now(clock0);
    CalculatePriceCommand.CalculatePriceCommandBuilder
calculatePriceCommand_CalculatePriceCommandBuilder1 =
calculatePriceCommand_CalculatePriceCommandBuilder0.policyFrom(localDate0
    CalculatePriceCommand calculatePriceCommand0 =
calculatePriceCommand_CalculatePriceCommandBuilder1.build();
    LocalDate localDate1 = calculatePriceCommand0.getPolicyFrom();
    assertSame(localDate1, localDate0);localDate0);
```

Can you explain why you think the selected test is more understandable?

Compare w/ EvoSuite 16 [Insurance]

Which test do you think is more understandable? (find the class under test *here*)

```
@BeforeEach
public void setUp() throws Exception {
    subject = new CalculatePriceCommand();
    subject.setPolicyFrom(LocalDate.parse("2023-05-17"));
    subject.setProductCode("FAI");
}

@Test
public void getProductCodeTest() throws Exception{
    String getProductCode = subject.getProductCode();

    assertThat(getProductCode, is("FAI"));
}
```

```
@Test
@Timeout(value = 4000 , unit = TimeUnit.MILLISECONDS)
public void testSetProductCode()  throws Throwable  {
    CalculatePriceCommand calculatePriceCommand0 = new CalculatePriceCommand
    calculatePriceCommand0.setProductCode("io.micronaut.data.runtime.interce
    assertEquals("io.micronaut.data.runtime.intercept.$DefaultFindOneInterce
  }
```

What makes this test more understandable? (Highlight text)

```
@BeforeEach
public void setUp() throws Exception {
    subject = new CalculatePriceCommand();
    subject.setPolicyFrom(LocalDate.parse("2023-05-17"));
    subject.setProductCode("FAI");
}

@Test
public void getProductCodeTest() throws Exception{
    String getProductCode = subject.getProductCode();
```

```
    assertThat(getProductCode, is("FAI"));is("FAI"));
```

What makes this test more understandable? (Highlight text)

```
@Test
@Timeout(value = 4000 , unit = TimeUnit.MILLISECONDS)
public void testSetProductCode()  throws Throwable  {
    CalculatePriceCommand calculatePriceCommand0 = new
CalculatePriceCommand();

calculatePriceCommand0.setProductCode("io.micronaut.data.runtime.intercept.$I

assertEquals("io.micronaut.data.runtime.intercept.$DefaultFindOneInterceptor$Intı
calculatePriceCommand0.getProductCode());calculatePriceCommand0.getProduc
```

Can you explain why you think the selected test is more understandable?

Compare w/ EvoSuite 17 [Insurance]

Which test do you think is more understandable? (find the class under test *here*)

```
@Test
public void getAnswersTest() throws Exception{
    ArrayList getAnswers = subject.getAnswers();

    QuestionAnswer questionAnswer = new QuestionAnswer();
    questionAnswer.setAnswer("KW");
    questionAnswer.setQuestionCode("TYP");
    ArrayList<QuestionAnswer> questionAnswers = new ArrayList<>();
    questionAnswers.add(questionAnswer);

    assertThat(getAnswers, is(questionAnswers));
}
```

```
@Test
@Timeout(value = 4000 , unit = TimeUnit.MILLISECONDS)
public void testAnswers()  throws Throwable  {
    CalculatePriceCommand.CalculatePriceCommandBuilder calculatePriceCommanc
    CalculatePriceCommand.CalculatePriceCommandBuilder calculatePriceCommanc
    assertSame(calculatePriceCommand_CalculatePriceCommandBuilder0, calculat
}
```

What makes this test more understandable? (Highlight text)

```
@Test
public void getAnswersTest() throws Exception{
    ArrayList getAnswers = subject.getAnswers();

    QuestionAnswer questionAnswer = new QuestionAnswer();
    questionAnswer.setAnswer("KW");
    questionAnswer.setQuestionCode("TYP");
    ArrayList<QuestionAnswer> questionAnswers = new ArrayList<>();
```

```
    questionAnswers.add(questionAnswer);

    assertThat(getAnswers, is(questionAnswers));is(questionAnswers));
```

What makes this test more understandable? (Highlight text)

```
@Test
@Timeout(value = 4000 , unit = TimeUnit.MILLISECONDS)
public void testAnswers()  throws Throwable  {
    CalculatePriceCommand.CalculatePriceCommandBuilder
calculatePriceCommand_CalculatePriceCommandBuilder0 =
CalculatePriceCommand.builder();
    CalculatePriceCommand.CalculatePriceCommandBuilder
calculatePriceCommand_CalculatePriceCommandBuilder1 =
calculatePriceCommand_CalculatePriceCommandBuilder0.answers((List<Questio
null);
    assertSame(calculatePriceCommand_CalculatePriceCommandBuilder0,
calculatePriceCommand_CalculatePriceCommandBuilder1);calculatePriceComma
```

Can you explain why you think the selected test is more understandable?

Compare w/ EvoSuite 18 [Insurance]

Which test do you think is more understandable? (find the class under test *here*)

```
@BeforeEach
public void setUp() throws Exception {
    HashMap<String, BigDecimal> stringMappedBigDecimal = new HashMap<>();
    stringMappedBigDecimal.put("C3", new BigDecimal("90.00"));
    stringMappedBigDecimal.put("C4", new BigDecimal("120.00"));
    stringMappedBigDecimal.put("C1", new BigDecimal("30.00"));
    stringMappedBigDecimal.put("C2", new BigDecimal("60.00"));

    subject = new CalculatePriceResult(new BigDecimal("300.00"), stringMappe
}

@Test
public void getTotalPriceTest() throws Exception{
    BigDecimal getTotalPrice = subject.getTotalPrice();

    assertThat(getTotalPrice, is(new BigDecimal("300.00")));
}
```

```
@Test
@Timeout(value = 4000 , unit = TimeUnit.MILLISECONDS)
public void testEmpty()  throws Throwable  {
    CalculatePriceResult calculatePriceResult0 = CalculatePriceResult.empty(
    BigInteger bigInteger0 = BigInteger.TEN;
    BigDecimal bigDecimal0 = new BigDecimal(bigInteger0);
    calculatePriceResult0.setTotalPrice(bigDecimal0);
    BigDecimal bigDecimal1 = calculatePriceResult0.getTotalPrice();
    assertEquals((short)10, bigDecimal1.shortValue());
}
```

What makes this test more understandable? (Highlight text)

```
@BeforeEach
public void setUp() throws Exception {
    HashMap<String, BigDecimal> stringMappedBigDecimal = new HashMap<>
();
    stringMappedBigDecimal.put("C3", new BigDecimal("90.00"));
    stringMappedBigDecimal.put("C4", new BigDecimal("120.00"));
    stringMappedBigDecimal.put("C1", new BigDecimal("30.00"));
    stringMappedBigDecimal.put("C2", new BigDecimal("60.00"));

    subject = new CalculatePriceResult(new BigDecimal("300.00"),
stringMappedBigDecimal);
}

@Test
public void getTotalPriceTest() throws Exception{
    BigDecimal getTotalPrice = subject.getTotalPrice();

    assertThat(getTotalPrice, is(new
BigDecimal("300.00")));BigDecimal("300.00")));
```

What makes this test more understandable? (Highlight text)

```
@Test
@Timeout(value = 4000 , unit = TimeUnit.MILLISECONDS)
public void testEmpty()  throws Throwable  {
    CalculatePriceResult calculatePriceResult0 = CalculatePriceResult.empty();
    BigInteger bigInteger0 = BigInteger.TEN;
    BigDecimal bigDecimal0 = new BigDecimal(bigInteger0);
    calculatePriceResult0.setTotalPrice(bigDecimal0);
    BigDecimal bigDecimal1 = calculatePriceResult0.getTotalPrice();
    assertEquals((short)10, bigDecimal1.shortValue());bigDecimal1.shortValue());
```

Can you explain why you think the selected test is more understandable?

Compare w/ EvoSuite 19 [Insurance]

Which test do you think is more understandable? (find the class under test *here*)

```
@Test
public void getCoversPricesTest() throws Exception{
    HashMap getCoversPrices = subject.getCoversPrices();

    HashMap<String, BigDecimal> stringMappedBigDecimal = new HashMap<>();
    stringMappedBigDecimal.put("C3", new BigDecimal("90.00"));
    stringMappedBigDecimal.put("C4", new BigDecimal("120.00"));
    stringMappedBigDecimal.put("C1", new BigDecimal("30.00"));
    stringMappedBigDecimal.put("C2", new BigDecimal("60.00"));

    assertThat(getCoversPrices, is(stringMappedBigDecimal));
}
```

```
@Test
public void testSetCoversPrices()  throws Throwable  {
    CalculatePriceResult calculatePriceResult0 = new CalculatePriceResult();
    HashMap<String, BigDecimal> hashMap0 = new HashMap<String, BigDecimal>();
    calculatePriceResult0.setCoversPrices(hashMap0);
    Map<String, BigDecimal> map0 = calculatePriceResult0.getCoversPrices();
    assertTrue(map0.isEmpty());
}
```

What makes this test more understandable? (Highlight text)

```
@Test
public void getCoversPricesTest() throws Exception{
   HashMap getCoversPrices = subject.getCoversPrices();

   HashMap<String, BigDecimal> stringMappedBigDecimal = new HashMap<>
();
   stringMappedBigDecimal.put("C3", new BigDecimal("90.00"));
   stringMappedBigDecimal.put("C4", new BigDecimal("120.00"));
   stringMappedBigDecimal.put("C1", new BigDecimal("30.00"));
   stringMappedBigDecimal.put("C2", new BigDecimal("60.00"));

   assertThat(getCoversPrices,
is(stringMappedBigDecimal));is(stringMappedBigDecimal));
```

What makes this test more understandable? (Highlight text)

```
@Test
public void testSetCoversPrices()  throws Throwable  {
   CalculatePriceResult calculatePriceResult0 = new CalculatePriceResult();
   HashMap<String, BigDecimal> hashMap0 = new HashMap<String,
BigDecimal>();
   calculatePriceResult0.setCoversPrices(hashMap0);
   Map<String, BigDecimal> map0 = calculatePriceResult0.getCoversPrices();
   assertTrue(map0.isEmpty());assertTrue(map0.isEmpty());
```

Can you explain why you think the selected test is more understandable?

Compare w/ EvoSuite 20 [Alfio]

Which test do you think is more understandable? (find the class under test *here*)

```
@BeforeEach
public void setUp() throws Exception {
    subject = new Language("en", "English");
}

@Test
public void shouldTestGetLocale(){
    String getLocale = subject.getLocale();

    assertThat(getLocale, is("en"));
}
```

```
@Test
public void testGetLocaleReturningNonEmptyString()  throws Throwable  {
    Language language0 = new Language("alfio.controller.api.v2.model.Languag
    String string0 = language0.getLocale();
    assertEquals("alfio.controller.api.v2.model.Language", string0);
}
```

What makes this test more understandable? (Highlight text)

```
@BeforeEach
public void setUp() throws Exception {
   subject = new Language("en", "English");
}

@Test
public void shouldTestGetLocale(){
   String getLocale = subject.getLocale();

   assertThat(getLocale, is("en"));is("en"));
```

What makes this test more understandable? (Highlight text)

```
@Test
public void testGetLocaleReturningNonEmptyString()  throws Throwable  {
   Language language0 = new
Language("alfio.controller.api.v2.model.Language", (String) null);
   String string0 = language0.getLocale();
   assertEquals("alfio.controller.api.v2.model.Language", string0);string0);
```

Can you explain why you think the selected test is more understandable?

Compare w/ EvoSuite 21 [Alfio]

Which test do you think is more understandable? (find the class under test *here*)

```
@BeforeEach
public void setUp() throws Exception {
    subject = new Language("en", "English");
}

@Test
public void shouldTestGetDisplayLanguage(){
    String getDisplayLanguage = subject.getDisplayLanguage();

    assertThat(getDisplayLanguage, is("English"));
}
```

```
@Test
public void testGetDisplayLanguageReturningEmptyString()  throws Throwable
    Language language0 = new Language("", "");
    String string0 = language0.getDisplayLanguage();
    assertEquals("", string0);
}
```

What makes this test more understandable? (Highlight text)

```
@BeforeEach
public void setUp() throws Exception {
   subject = new Language("en", "English");
}

@Test
public void shouldTestGetDisplayLanguage(){
   String getDisplayLanguage = subject.getDisplayLanguage();

   assertThat(getDisplayLanguage, is("English"));is("English"));
```

What makes this test more understandable? (Highlight text)

```
@Test
public void testGetDisplayLanguageReturningEmptyString()  throws Throwable  {
   Language language0 = new Language("", "");
   String string0 = language0.getDisplayLanguage();
   assertEquals("", string0);string0);
```

Can you explain why you think the selected test is more understandable?

Compare W/ Manually-written 1 [spring-testing]

Which test do you think is more understandable? (find the class under test *here*)

```
@Test
public void weatherTest() throws Exception {
 given(weatherClient.fetchWeather()).willReturn(
   Optional.of(new WeatherResponse("Clouds", "few clouds")
 ));

 String weather = subject.weather();

 assertThat(weather, is("Clouds: few clouds"));
}
```

```
@Test
public void shouldReturnWeatherClientResult() throws Exception {
    WeatherResponse weatherResponse = new WeatherResponse("raining", "a ligh
    given(weatherClient.fetchWeather()).willReturn(Optional.of(weatherRespor

    var weather = subject.weather();

    assertThat(weather, is("raining: a light drizzle"));
}
```

What makes this test more understandable? (Highlight text)

```
@Test
public void weatherTest() throws Exception {
     given(weatherClient.fetchWeather()).willReturn(
       Optional.of(new WeatherResponse("Clouds", "few clouds")
     ));
     String weather = subject.weather();
     assertThat(weather, is("Clouds: few clouds"));is("Clouds: few clouds"));
```

What makes this test more understandable? (Highlight text)

```
@Test
public void shouldReturnWeatherClientResult() throws Exception {
   WeatherResponse weatherResponse = new
WeatherResponse("raining", "a light drizzle");
   given(weatherClient.fetchWeather()).willReturn(Optional.of(weatherResponse));

   var weather = subject.weather();

   assertThat(weather, is("raining: a light drizzle"));is("raining: a light drizzle"));
```

Can you explain why you think the selected test is more understandable?

Compare W/ Manually-written 2 [spring-testing]

Which test do you think is more understandable? (find the class under test here)

```
@Test
public void helloWhereCarterTest() throws Exception{
   Person carter = new Person("james", "carter");
   given(personRepository.findByLastName("carter")).willReturn(Optional.of(

String hello = subject.hello("carter");

   assertThat(hello, is("Hello james carter!"));
}
```

```
@Test
public void shouldReturnWeatherClientResult() throws Exception {
    WeatherResponse weatherResponse = new WeatherResponse("raining", "a ligh
    given(weatherClient.fetchWeather()).willReturn(Optional.of(weatherRespor

    var weather = subject.weather();

    assertThat(weather, is("raining: a light drizzle"));
}
```

What makes this test more understandable? (Highlight text)

```
@Test
public void helloWhereCarterTest() throws Exception{
   Person carter = new Person("james", "carter");

given(personRepository.findByLastName("carter")).willReturn(Optional.of(carter));

   String hello = subject.hello("carter");

   assertThat(hello, is("Hello james carter!"));carter!"));
```

What makes this test more understandable? (Highlight text)

```
@Test
public void shouldReturnFullNameOfAPerson() throws Exception {
   Person peter = new Person("Peter", "Pan");
   given(personRepository.findByLastName("Pan")).willReturn(Optional.of(peter));

   var greeting = subject.hello("Pan");

   assertThat(greeting, is("Hello Peter Pan!"));Pan!"));
```

Can you explain why you think the selected test is more understandable?

Compare W/ Manually-written 3 [spring-testing]

Which test do you think is more understandable? (find the class under test here)

```
@Test
public void helloWhereAmirTest() throws Exception{
    given(personRepository.findByLastName("Amir")).willReturn(Optional.empty

    String hello = subject.hello("Amir");

    assertThat(hello, is("Who is this 'Amir' you're talking about?"));
}
```

```
@Test
public void shouldTellIfPersonIsUnknown() throws Exception {
    given(personRepository.findByLastName(anyString())).willReturn(Optional.

    var greeting = subject.hello("Pan");

    assertThat(greeting, is("Who is this 'Pan' you're talking about?"));
}
```

What makes this test more understandable? (Highlight text)

```
@Test
public void helloWhereAmirTest() throws Exception{
   given(personRepository.findByLastName("Amir")).willReturn(Optional.empty());

   String hello = subject.hello("Amir");

   assertThat(hello, is("Who is this 'Amir' you're talking about?"));about?"));
```

What makes this test more understandable? (Highlight text)

```
@Test
public void shouldTellIfPersonIsUnknown() throws Exception {

given(personRepository.findByLastName(anyString())).willReturn(Optional.empty());

   var greeting = subject.hello("Pan");

   assertThat(greeting, is("Who is this 'Pan' you're talking about?"));about?"));
```

Can you explain why you think the selected test is more understandable?

Compare W/ Manually-written 4 [spring-testing]

Which test do you think is more understandable? (find the class under test here)

```
@Test
public void fetchWeatherTest() throws Exception{
    given(restTemplate.getForObject("https://api.openweathermap.org/data/2.5
           WeatherResponse.class)).willReturn(new WeatherResponse("Clear", "

    Optional fetchWeather = subject.fetchWeather();

    assertThat(fetchWeather, is(Optional.of(new WeatherResponse("Clear", "cl
}
```

```
@Test
public void shouldCallWeatherService() throws Exception {
    var expectedResponse = new WeatherResponse("raining", "a light drizzle");
    given(restTemplate.getForObject("http://localhost:8089/data/2.5/weather?
            WeatherResponse.class)).willReturn(expectedResponse);

    var actualResponse = subject.fetchWeather();

    assertThat(actualResponse, is(Optional.of(expectedResponse)));
}
```

What makes this test more understandable? (Highlight text)

```
@Test
public void fetchWeatherTest() throws Exception{

given(restTemplate.getForObject("https://api.openweathermap.org/data/2.5/weath
q=Hamburg,de&appid,
        WeatherResponse.class)).willReturn(new WeatherResponse("Clear",
"clear sky"));

   Optional fetchWeather = subject.fetchWeather();

   assertThat(fetchWeather, is(Optional.of(new WeatherResponse("Clear", "clear
sky")))));sky"))));
```

What makes this test more understandable? (Highlight text)

```
@Test
public void shouldCallWeatherService() throws Exception {
   var expectedResponse = new WeatherResponse("raining", "a light drizzle");
   given(restTemplate.getForObject("http://localhost:8089/data/2.5/weather?
q=Hamburg,de,
        WeatherResponse.class)).willReturn(expectedResponse);

   var actualResponse = subject.fetchWeather();

   assertThat(actualResponse,
is(Optional.of(expectedResponse)));is(Optional.of(expectedResponse)));
```

Can you explain why you think the selected test is more understandable?

Compare W/ Manually-written 5 [petclinic]

Which test do you think is more understandable? (find the class under test here)

```
@Test
void testPrint() {
    PetType petType = new PetType();
    petType.setName("Hamster");
    String petTypeName = this.petTypeFormatter.print(petType, Locale.ENGLISH
    assertThat(petTypeName).isEqualTo("Hamster");
}
```

```
@Test
public void printTest() throws Exception{
    PetType petType = new PetType();
    petType.setId(2);
    petType.setName("dog");

    String print = subject.print(petType, Locale.ENGLISH);

    assertThat(print, is("dog"));
}
```

What makes this test more understandable? (Highlight text)

```
@Test
public void printTest() throws Exception{
    PetType petType = new PetType();
    petType.setId(2);
    petType.setName("dog");

    String print = subject.print(petType, Locale.ENGLISH);

    assertThat(print, is("dog"));is("dog"));
```

What makes this test more understandable? (Highlight text)

```
@Test
void testPrint() {
    PetType petType = new PetType();
    petType.setName("Hamster");
    String petTypeName = this.petTypeFormatter.print(petType, Locale.ENGLISH);

assertThat(petTypeName).isEqualTo("Hamster");assertThat(petTypeName).isEqual
```

Can you explain why you think the selected test is more understandable?

Compare W/ Manually-written 7 [petclinic]

Which test do you think is more understandable? (find the class under test here)

```
private List<PetType> makePetTypes() {
    List<PetType> petTypes = new ArrayList<>();
    petTypes.add(new PetType() {
        { setName("Dog"); }
    });
    petTypes.add(new PetType() {
        { setName("Bird"); }
    });
    return petTypes;
}

@Test
void shouldParse() throws ParseException {
    given(this.pets.findPetTypes()).willReturn(makePetTypes());
    PetType petType = petTypeFormatter.parse("Bird", Locale.ENGLISH);
    assertThat(petType.getName()).isEqualTo("Bird");
}
```

```
@Test
public void parseWhereCatTest() throws Exception {
    PetType petType = new PetType();
    petType.setId(1);
    petType.setName("cat");
    PetType petType_1 = new PetType();
    petType_1.setId(2);
    petType_1.setName("dog");
    ArrayList<PetType> petTypes = new ArrayList<>();
    petTypes.add(petType);
    petTypes.add(petType_1);

    given(owners.findPetTypes()).willReturn(petTypes);

    PetType parse = subject.parse("cat", Locale.ENGLISH);

    PetType petType_2 = new PetType();
    petType_2.setId(1);
    petType_2.setName("cat");

    assertThat(parse, is(petType_2));
}
```

What makes this test more understandable? (Highlight text)

```
private List<PetType> makePetTypes() {
   List<PetType> petTypes = new ArrayList<>();
   petTypes.add(new PetType() {
      { setName("Dog"); }
   });
   petTypes.add(new PetType() {
      { setName("Bird"); }
   });
   return petTypes;
}

@Test
```

```
void shouldParse() throws ParseException {
   given(this.pets.findPetTypes()).willReturn(makePetTypes());
   PetType petType = petTypeFormatter.parse("Bird", Locale.ENGLISH);

assertThat(petType.getName()).isEqualTo("Bird");assertThat(petType.getName()).isE
```

What makes this test more understandable? (Highlight text)

```
@Test
public void parseWhereCatTest() throws Exception {
   PetType petType = new PetType();
   petType.setId(1);
   petType.setName("cat");
   PetType petType_1 = new PetType();
   petType_1.setId(2);
   petType_1.setName("dog");
   ArrayList<PetType> petTypes = new ArrayList<>();
   petTypes.add(petType);
   petTypes.add(petType_1);

   given(owners.findPetTypes()).willReturn(petTypes);

   PetType parse = subject.parse("cat", Locale.ENGLISH);

   PetType petType_2 = new PetType();
   petType_2.setId(1);
   petType_2.setName("cat");

   assertThat(parse, is(petType_2));is(petType_2));
```

Can you explain why you think the selected test is more understandable?

Compare W/ Manually-written 8 [petclinic]

Which test do you think is more understandable? (find the class under test here)

```
@Test
public void findAllOwnersTest() throws Exception{
    given(ownerRepository.findAll()).willReturn(getOwners());

    ArrayList findAllOwners = subject.findAllOwners();

    Owner owner_1 = new Owner();
    owner_1.setLastName("Franklin");
    owner_1.setFirstName("George");
    owner_1.setId(1);

    Owner owner_2 = new Owner();
    owner_2.setLastName("Davis");
    owner_2.setFirstName("Betty");
    owner_2.setId(2);

    ArrayList<Owner> owners_1 = new ArrayList<>();
    owners.add(owner);
    owners.add(owner_2);

    assertThat(findAllOwners, is(owners));
}
```

```
@Test
void shouldFindAllOwners(){
    Collection<Owner> owners = this.clinicService.findAllOwners();
    Owner owner1 = EntityUtils.getById(owners, Owner.class, 1);
    assertThat(owner1.getFirstName()).isEqualTo("George");
    Owner owner3 = EntityUtils.getById(owners, Owner.class, 3);
    assertThat(owner3.getFirstName()).isEqualTo("Eduardo");
}
```

What makes this test more understandable? (Highlight text)

```
@Test
void shouldFindAllOwners(){
   Collection<Owner> owners = this.clinicService.findAllOwners();
   Owner owner1 = EntityUtils.getById(owners, Owner.class, 1);
   assertThat(owner1.getFirstName()).isEqualTo("George");
   Owner owner3 = EntityUtils.getById(owners, Owner.class, 3);

assertThat(owner3.getFirstName()).isEqualTo("Eduardo");assertThat(owner3.getFir
```

What makes this test more understandable? (Highlight text)

```
@Test
public void findAllOwnersTest() throws Exception{
   given(ownerRepository.findAll()).willReturn(getOwners());

   ArrayList findAllOwners = subject.findAllOwners();

   Owner owner_1 = new Owner();
   owner_1.setLastName("Franklin");
   owner_1.setFirstName("George");
   owner_1.setId(1);

   Owner owner_2 = new Owner();
   owner_2.setLastName("Davis");
   owner_2.setFirstName("Betty");
   owner_2.setId(2);
```

```
ArrayList<Owner> owners_1 = new ArrayList<>();
owners.add(owner);
owners.add(owner_2);

assertThat(findAllOwners, is(owners));is(owners));
```

Can you explain why you think the selected test is more understandable?

Compare W/ Manually-written 9 [Alfio]

Which test do you think is more understandable? (find the class under test here)

```
@BeforeEach
public void setUp() throws Exception {
    subject = new ConfigurationLevels.SystemLevel();
}

@Test
public void shouldTestGetPathLevel(){
    ConfigurationPathLevel getPathLevel = subject.getPathLevel();

    assertThat(getPathLevel, is(ConfigurationPathLevel.SYSTEM));
}
```

```
@Test
void system() {
    ConfigurationLevel system = ConfigurationLevel.system();
    assertTrue(system instanceof ConfigurationLevels.SystemLevel);
    assertEquals(ConfigurationPathLevel.SYSTEM, system.getPathLevel());
}
```

What makes this test more understandable? (Highlight text)

```
@Test
void system() {
    ConfigurationLevel system = ConfigurationLevel.system();
    assertTrue(system instanceof ConfigurationLevels.SystemLevel);
    assertEquals(ConfigurationPathLevel.SYSTEM,
system.getPathLevel());system.getPathLevel());
```

What makes this test more understandable? (Highlight text)

```
@BeforeEach
public void setUp() throws Exception {
    subject = new ConfigurationLevels.SystemLevel();
}

@Test
public void shouldTestGetPathLevel(){
    ConfigurationPathLevel getPathLevel = subject.getPathLevel();
```

assertThat(getPathLevel,
is(ConfigurationPathLevel.SYSTEM));is(ConfigurationPathLevel.SYSTEM));

Can you explain why you think the selected test is more understandable?

Compare W/ Manually-written 10 [Alfio]

Which test do you think is more understandable? (find the class under test here)

```
@Test
public void shouldTestGetStatus(){
    subject = new TicketReservationStatusAndValidation(PENDING, true);
    TicketReservation.TicketReservationStatus getStatus = subject.getStatus(

    assertThat(getStatus, is(TicketReservation.TicketReservationStatus.PENDI
}
```

```
@Test
void confirmAndLockTickets() {
    when(ticketReservationRepository.findOptionalStatusAndValidationById(eq(
    when(configurationManager.getFor(eq(SEND_TICKETS_AUTOMATICALLY), any()))
        new MaybeConfiguration(SEND_TICKETS_AUTOMATICALLY)
    );
    when(configurationManager.getFor(eq(BANKING_KEY), any())).thenReturn(BAN
    when(ticketRepository.forbidReassignment(any())).thenReturn(1);
    mockBillingDocument();
    testPaidReservation(true, true);
}
```

What makes this test more understandable? (Highlight text)

@Test
public void shouldTestGetStatus(){
subject = new TicketReservationStatusAndValidation(PENDING, true);
TicketReservation.TicketReservationStatus getStatus = subject.getStatus();

assertThat(getStatus,
is(TicketReservation.TicketReservationStatus.PENDING));is(TicketReservation.Tick

What makes this test more understandable? (Highlight text)

@Test
void confirmAndLockTickets() {

when(ticketReservationRepository.findOptionalStatusAndValidationById(eq(RESEF
TicketReservationStatusAndValidation(PENDING, true)));
when(configurationManager.getFor(eq(SEND_TICKETS_AUTOMATICALLY),
any())).thenReturn(
new MaybeConfiguration(SEND_TICKETS_AUTOMATICALLY)
);
when(configurationManager.getFor(eq(BANKING_KEY),
any())).thenReturn(BANKING_INFO);

```
when(ticketRepository.forbidReassignment(any())).thenReturn(1);
mockBillingDocument();
testPaidReservation(true, true);true);
```

Can you explain why you think the selected test is more understandable?

Ranking Question

Could you please rank the tests from 1 to 3 based on their level of understandability, with 1 being the most understandable, and 3 being thee least understandable?
(find the class under test here)

```
@Test
public void testAddVisitReturningOwnerWhereIsNewIsFalse() throws Throwa
  Owner owner0 = new Owner();
  Pet pet0 = new Pet();
  owner0.addPet(pet0);
  Integer integer0 = new Integer(448);
  owner0.setId(integer0);
  pet0.setId(integer0);
  Visit visit0 = new Visit();
  Owner owner1 = owner0.addVisit(integer0, visit0);
  assertNull(owner1.getAddress());
}
```

```
@Test
public void findOwnerTest() throws Exception{
    given(owners.findById(2)).willReturn(getOwner());

    Owner findOwner = subject.findOwner(2);

    PetType petType = new PetType();
    petType.setName("dog");
    Pet pet = new Pet();
    pet.setType(petType);
    pet.setBirthDate(LocalDate.parse("2002-08-06"));
    pet.setName("Basil");

    Owner owner = new Owner();
    owner.addPet(pet);
    owner.setAddress("638 Cardinal Ave.");
    owner.setCity("Sun Prairie");
    owner.setLastName("Davis");
    owner.setFirstName("Betty");
    owner.setTelephone("6085551749");
    owner.setId(2);

    assertThat(findOwner, is(owner));
}
```

```
@Test
@Transactional
void shouldInsertOwner() {
    Page<Owner> owners = this.owners.findByLastName("Schultz", pageable
    int found = (int) owners.getTotalElements();

    Owner owner = new Owner();
    owner.setFirstName("Sam");
    owner.setLastName("Schultz");
    owner.setAddress("4, Evans Street");
    owner.setCity("Wollongong");
    owner.setTelephone("4444444444");
    this.owners.save(owner);
```

```
    assertThat(owner.getId().longValue()).isNotEqualTo(0);

    owners = this.owners.findByLastName("Schultz", pageable);
    assertThat(owners.getTotalElements()).isEqualTo(found + 1);
}
```

Ranking 2 [petclinic]

Could you please rank the tests from 1 to 3 based on their level of understandability, with 1 being the most understandable, and 3 being thee least understandable?
(find the class under test here)

```
@Test
public void testGetLastNameReturningNonEmptyString()  throws Throwable
   Person person0 = new Person();
   person0.setLastName("Lw*`onX`MIV%");
   String string0 = person0.getLastName();
   assertEquals("Lw*`onX`MIV%", string0);
}
```

```
void shouldUpdateOwner() {
    Owner owner = this.owners.findById(1);
    String oldLastName = owner.getLastName();
    String newLastName = oldLastName + "X";

    owner.setLastName(newLastName);
    this.owners.save(owner);

    owner = this.owners.findById(1);
    assertThat(owner.getLastName()).isEqualTo(newLastName);
}
```

```
@Test
public void getLastNameTest(){
    subject = new Person();

    subject.setId(1);
    subject.setFirstName("George");
    subject.setLastName("Franklin");

    String getLastName = subject.getLastName();

    assertThat(getLastName, is("Franklin"));
}
```

Ranking 3 [petclinic]

Could you please rank the tests from 1 to 3 based on their level of understandability, with 1 being the most understandable, and 3 being thee least understandable?
(find the class under test here)

```
@Test
@Transactional
void shouldInsertPetIntoDatabaseAndGenerateId() {
    Owner owner6 = this.owners.findById(6);
    int found = owner6.getPets().size();

    Pet pet = new Pet();
    pet.setName("bowser");
    Collection<PetType> types = this.owners.findPetTypes();
```

```
    pet.setType(EntityUtils.getById(types, PetType.class, 2));
    pet.setBirthDate(LocalDate.now());
    owner6.addPet(pet);
    assertThat(owner6.getPets().size()).isEqualTo(found + 1);

    this.owners.save(owner6);

    owner6 = this.owners.findById(6);
    assertThat(owner6.getPets().size()).isEqualTo(found + 1);
    pet = owner6.getPet("bowser");
    assertThat(pet.getId()).isNotNull();
}
```

```
@Test
public void getPetsTest() throws Exception
    PersistentBag getPets = subject.getPets();

    PetType petType = new PetType();
    petType.setId(1);
    petType.setName("cat");
    Pet pet = new Pet();
    pet.addVisit(visit);
    pet.setId(1);
    pet.setType(petType);
    pet.setBirthDate(LocalDate.parse("2000-09-07"));
    pet.setName("Leo");
    PersistentBag<Pet> pets = new PersistentBag<>();
    pets.add(pet);

    assertThat(getPets, is(pets));
}
```

```
@Test
public void testGetPetTaking2ArgumentsReturningPetWhereIsNewIsFalse()
   Owner owner0 = new Owner();
   Pet pet0 = new Pet();
   owner0.addPet(pet0);
   Integer integer0 = new Integer(1);
   pet0.setId(integer0);
   pet0.setName("*");
   Pet pet1 = owner0.getPet("*", true);
   assertEquals(1, (int)pet1.getId());
}
```

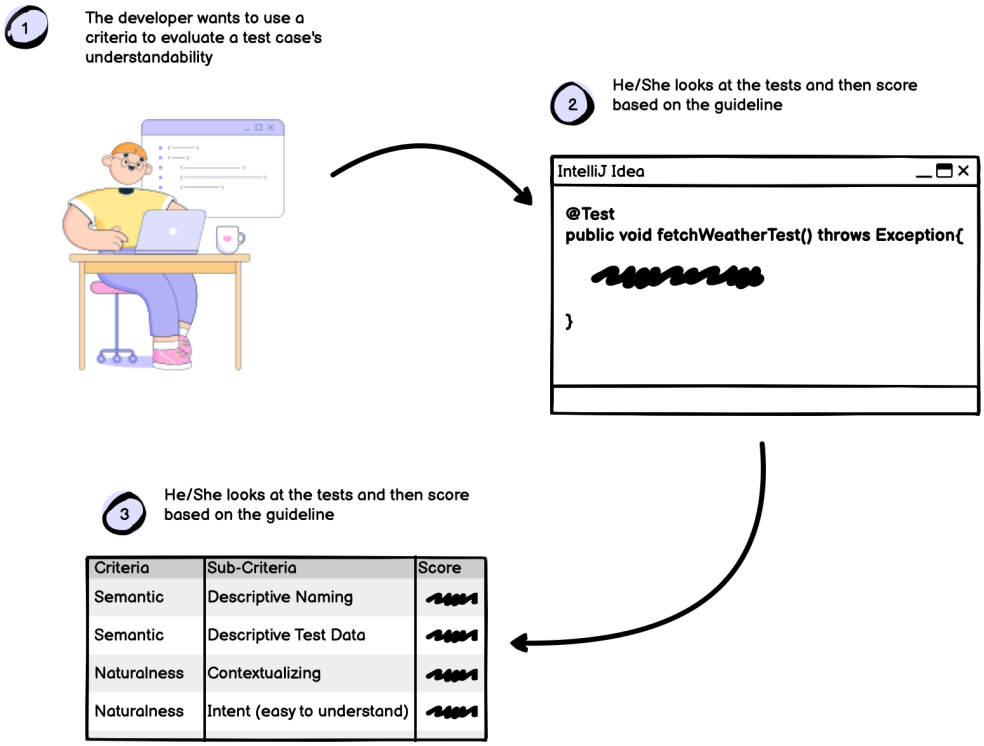
Second Round - Intro

Thanks for your so far responses!

Now, we want to understand which parts of these tests are understandable and which parts are not. So we propose criteria to define understandability, and we ask you to rate the tests based on these criteria.

The criteria:

- Semantic:
  - Descriptive naming: The names of variables and methods that describe their function.
  - Descriptive test data: Test data clearly (input, output, mock data) illustrates the test scenario.
- Naturalness:
  - Meaningful in the context: The test scenario is meaningful in the domain of the system.
  - Intent (easy to understand): The behavior of the test is easy to understand.



Criteria-Based-MTC-1 [spring-testing]

According to the criteria, score the test as follows:

```
@Test
public void weatherTest() throws Exception{
            given(weatherClient.fetchWeather()).willReturn(Optional.of(new WeatherResponse("C]


            String weather = subject.weather();


            assertThat(weather, is("Clear: clear sky"));
}
```

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
The identifiers are descriptive	○	○	○	○	○
The test data is descriptive	○	○	○	○	○
The test makes sense in the domain	○	○	○	○	○
The intent of the test is easy to understand	○	○	○	○	○

Would it be possible for you to explain which parts are easy to understand and which parts are difficult to understand?

Criteria-Based-MTC-2 [petclinic]

According to the criteria, score the test as follows:

```
@Test
public void findOwnerTest() throws Exception{
    given(owners.findById(2)).willReturn(getOwner());
```

```
    Owner findOwner = subject.findOwner(2);

    PetType petType = new PetType();
    petType.setName("dog");
    Pet pet = new Pet();
    pet.setType(petType);
    pet.setBirthDate(LocalDate.parse("2002-08-06"));
    pet.setName("Basil");

    Owner owner = new Owner();
    owner.addPet(pet);
    owner.setAddress("638 Cardinal Ave.");
    owner.setCity("Sun Prairie");
    owner.setLastName("Davis");
    owner.setFirstName("Betty");
    owner.setTelephone("6085551749");
    owner.setId(2);

    assertThat(findOwner, is(owner));
}
```

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
The identifiers are descriptive	○	○	○	○	○
The test data is descriptive	○	○	○	○	○
The test makes sense in the domain	○	○	○	○	○
The intent of the test is easy to understand	○	○	○	○	○

Would it be possible for you to explain which parts are easy to understand and which parts are difficult to understand?

Criteria-Based-MTC-3 [petclinic]

According to the criteria, score the test as follows:

```
    @Test
    public void parseWhereDogTest() throws Exception {
                PetType PetType = new PetType();
                PetType.setId(1);
                PetType.setName("cat");
                PetType PetType_1 = new PetType();
                PetType_1.setId(2);
                PetType_1.setName("dog");
                ArrayList<p> PetTypes = new ArrayList();
                PetTypes.add(PetType);
                PetTypes.add(PetType_1);

                given(owners.findPetTypes()).willReturn(PetTypes);

                PetType parse = subject.parse("dog", Locale.ENGLISH);

                PetType PetType_2 = new PetType();
```

```
        PetType_2.setId(2);
        PetType_2.setName("dog");

        assertThat(parse, is(PetType_2));
    }
```

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
The identifiers are descriptive	○	○	○	○	○
The test data is descriptive	○	○	○	○	○
The test makes sense in the domain	○	○	○	○	○
The intent of the test is easy to understand	○	○	○	○	○

Would it be possible for you to explain which parts are easy to understand and which parts are difficult to understand?

Criteria-Based-MTC-4 [petclinic]

According to the criteria, score the test as follows:

```
    @BeforeEach
    public void setUp() throws Exception {
        MockitoAnnotations.openMocks(this);
        subject = new Specialty();
        subject.setId(5);
    }
     @Test
     public void getNameTest() throws Exception {
         subject.setName("radiology");

         String getName = subject.getName();
         assertThat(getName, is("radiology"));
}
```

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
The identifiers are descriptive	○	○	○	○	○
The test data is descriptive	○	○	○	○	○
The test makes sense in the domain	○	○	○	○	○
The intent of the test is easy to understand	○	○	○	○	○

Would it be possible for you to explain which parts are easy to understand and which parts are difficult to understand?

Criteria-Based-MTC-5 [Alfio]

According to the criteria, score the test as follows:

```
@BeforeEach
public void setUp() throws Exception {
    MockitoAnnotations.openMocks(this);
    String BASE_URL = "https://127.0.0.1:8080/admin";
    subject = new ConfigurationKeyValuePathLevel(ConfigurationKeys.BANK_TRANSFER_ENABLED.getVa
}

@Test
public void getConfigurationKeyTest(){
    ConfigurationKeys getConfigurationKey = subject.getConfigurationKey();

    assertThat(getConfigurationKey, is(ConfigurationKeys.BANK_TRANSFER_ENABLED));
}
```

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
The identifiers are descriptive	○	○	○	○	○
The test data is descriptive	○	○	○	○	○
The test makes sense in the domain	○	○	○	○	○
The intent of the test is easy to understand	○	○	○	○	○

Would it be possible for you to explain which parts are easy to understand and which parts are difficult to understand?

Criteria-Based-MTC-6 [Alfio]

According to the criteria, score the test as follows:

```
@Test
public void shouldTestGetEmail(){
    subject = new Organization.OrganizationContact("ACM", "info@acm.org");

    String getEmail = subject.getEmail();

    assertThat(getEmail, is("info@acm.org"));
}
```

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
The identifiers are descriptive	○	○	○	○	○
The test data is descriptive	○	○	○	○	○
The test makes sense in the domain	○	○	○	○	○
The intent of the test is easy to understand	○	○	○	○	○

Would it be possible for you to explain which parts are easy to understand and which parts are difficult to understand?

Criteria-Based-MTC-7 [Alfio]

According to the criteria, score the test as follows:

```
@Test
public void getDescriptionTest(){
    subject = new EventDescription(2, "en", DESCRIPTION, "AST 2023 - Conference");

    String getDescription = subject.getDescription();

    assertThat(getDescription, is("AST 2023 - Conference"));
}
```

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
The identifiers are descriptive	○	○	○	○	○
The test data is descriptive	○	○	○	○	○
The test makes sense in the domain	○	○	○	○	○
The intent of the test is easy to understand	○	○	○	○	○

Would it be possible for you to explain which parts are easy to understand and which parts are difficult to understand?

Criteria-Based-MTC-8 [Insurance]

According to the criteria, score the test as follows:

```
@BeforeEach
public void setUp() throws Exception {
    MockitoAnnotations.openMocks(this);
    HashMap<String, Integer> stringMappedInteger = new HashMap<>();
    stringMappedInteger.put("NUM_OF_CLAIM", 5);
    stringMappedInteger.put("FLOOD", 1);
    stringMappedInteger.put("AREA", 5000);
    stringMappedInteger.put("TYP", 0);

    ArrayList<String> strings = new ArrayList<>();
    strings.add("C1");
    strings.add("C2");
    strings.add("C3");
    strings.add("C4");

    subject = new Calculation(stringMappedInteger, "FAI", strings, LocalDate.parse("2023-05-31
}

@Test
public void getTotalPremiumTest() throws Exception{
    BigDecimal getTotalPremium = subject.getTotalPremium();
```

```
        assertThat(getTotalPremium, is(new BigDecimal("300.00")));
    }
```

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
The identifiers are descriptive	○	○	○	○	○
The test data is descriptive	○	○	○	○	○
The test makes sense in the domain	○	○	○	○	○
The intent of the test is easy to understand	○	○	○	○	○

Would it be possible for you to explain which parts are easy to understand and which parts are difficult to understand?

Criteria-Based-MTC-9 [Insurance]

According to the criteria, score the test as follows:

```
    @BeforeEach
    public void setUp() throws Exception {
        MockitoAnnotations.openMocks(this);
        subject = new CalculatePriceCommand();
        subject.setPolicyFrom(LocalDate.parse("2023-05-17"));
        subject.setProductCode("FAI");
    }

    @Test
    public void getPolicyToTest() throws Exception{
        LocalDate getPolicyTo = subject.getPolicyTo();

        assertThat(getPolicyTo, is(LocalDate.parse("2023-05-31")));
    }
```

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
The identifiers are descriptive	○	○	○	○	○
The test data is descriptive	○	○	○	○	○
The test makes sense in the domain	○	○	○	○	○
The intent of the test is easy to understand	○	○	○	○	○

Would it be possible for you to explain which parts are easy to understand and which parts are difficult to understand?

Criteria-Based-M 1 [Spring-Testing]

According to the criteria, score the test as follows:

```
@Test
public void shouldSaveAndFetchPerson() throws Exception {
    var peter = new Person("Peter", "Pan");
    subject.save(peter);

    var maybePeter = subject.findByLastName("Pan");

    assertThat(maybePeter, is(Optional.of(peter)));
}
```

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
The identifiers are descriptive	○	○	○	○	○
The test data is descriptive	○	○	○	○	○
The test makes sense in the domain	○	○	○	○	○
The intent of the test is easy to understand	○	○	○	○	○

Would it be possible for you to explain which parts are easy to understand and which parts are difficult to understand?

Criteria-Based-M 2 [Petclinic]

According to the criteria, score the test as follows:

```
@Test
@Transactional
        void shouldUpdatePetName() throws Exception {
        Owner owner6 = this.owners.findById(6);
        Pet pet7 = owner6.getPet(7);
        String oldName = pet7.getName();

        String newName = oldName + "X";
        pet7.setName(newName);
        this.owners.save(owner6);

        owner6 = this.owners.findById(6);
        pet7 = owner6.getPet(7);
        assertThat(pet7.getName()).isEqualTo(newName);
}
```

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
The identifiers are descriptive	○	○	○	○	○
The test data is descriptive	○	○	○	○	○
The test makes sense in the domain	○	○	○	○	○

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
The intent of the test is easy to understand	○	○	○	○	○

Would it be possible for you to explain which parts are easy to understand and which parts are difficult to understand?

Criteria-Based-M 3 [Petclinic]

According to the criteria, score the test as follows:

```
@Test
void shouldFindSingleOwnerWithPet() {
        Owner owner = this.owners.findById(1).get();
        assertThat(owner.getLastName()).startsWith("Franklin
        assertThat(owner.getPets()).hasSize(1);
        assertThat(owner.getPets().get(0).getType()).isNotNu
        assertThat(owner.getPets().get(0).getType().getName(
}
```

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
The identifiers are descriptive	○	○	○	○	○
The test data is descriptive	○	○	○	○	○
The test makes sense in the domain	○	○	○	○	○
The intent of the test is easy to understand	○	○	○	○	○

Would it be possible for you to explain which parts are easy to understand and which parts are difficult to understand?

Criteria-Based-M 4 [Petclinic]

According to the criteria, score the test as follows:

```
@Test
void shouldFindAllPetTypes() {
        Collection<p> petTypes = this.owners.findPetTypes();

        PetType petType = EntityUtils.getById(petTypes, PetT
        assertThat(petType1.getName()).isEqualTo("cat");
}
```

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
The identifiers are descriptive	○	○	○	○	○
The test data is descriptive	○	○	○	○	○
The test makes sense in the domain	○	○	○	○	○
The intent of the test is easy to understand	○	○	○	○	○

Would it be possible for you to explain which parts are easy to understand and which parts are difficult to understand?

Criteria-Based-M 5 [Petclinic]

According to the criteria, score the test as follows:

```
@Test
void shouldFindVets() {
        Collection vets = this.vets.findAll();

        Vet vet = EntityUtils.getById(vets, Vet.class, 3);
        assertThat(vet.getLastName()).isEqualTo("Douglas");
        assertThat(vet.getSpecialties().get(1).getName()).is
}
```

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
The identifiers are descriptive	○	○	○	○	○
The test data is descriptive	○	○	○	○	○
The test makes sense in the domain	○	○	○	○	○
The intent of the test is easy to understand	○	○	○	○	○

Would it be possible for you to explain which parts are easy to understand and which parts are difficult to understand?

Criteria-Based-M 6 [Petclinic]

According to the criteria, score the test as follows:

```
@Test
@Transactional
void shouldAddNewVisitForPet() {
        Owner owner6 = this.owners.findById(6).get();
        Pet pet7 = owner6.getPet(7);
        int found = pet7.getVisits().size();
```

```
        Visit visit = new Visit();
        visit.setDescription("test");

        owner6.addVisit(pet7.getId(), visit);
        this.owners.save(owner6);

        owner6 = this.owners.findById(6).get();

        assertThat(pet7.getVisits()) //
                .hasSize(found + 1) //
                .allMatch(value -> value.getId() !=
    }
```

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
The identifiers are descriptive	○	○	○	○	○
The test data is descriptive	○	○	○	○	○
The test makes sense in the domain	○	○	○	○	○
The intent of the test is easy to understand	○	○	○	○	○

Would it be possible for you to explain which parts are easy to understand and which parts are difficult to understand?

Criteria-Based-M 7 [Petclinic]

According to the criteria, score the test as follows:

```
@Test
void shouldFindVisitsByPetId() throws Exception {
    Owner owner = this.owners.findById(6).get();
    Pet pet = owner.getPet(7);
    Collection<Visit> visits = pet.getVisits();

    assertThat(visits) //
            .hasSize(2) //
            .element(0).extracting(Visit::getDate).isNotNull();
}
```

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
The identifiers are descriptive	○	○	○	○	○
The test data is descriptive	○	○	○	○	○
The test makes sense in the domain	○	○	○	○	○
The intent of the test is easy to understand	○	○	○	○	○

Would it be possible for you to explain which parts are easy to understand and which parts are difficult to understand?

Criteria-Based-M 8 [Alfio]

According to the criteria, score the test as follows:

```
@Test
void totalPriceDoesNotIncludeVatIfSplitPayment() {
    PriceContainerImpl vs = new PriceContainerImpl(1100, "CHF",
    assertEquals(new BigDecimal("10.00"), vs.getFinalPrice());
    assertEquals(new BigDecimal("1.00"), vs.getVAT());

    vs = new PriceContainerImpl(1000, "CHF", new BigDecimal("10.
    assertEquals(new BigDecimal("10.00"), vs.getFinalPrice());
    assertEquals(new BigDecimal("1.00"), vs.getVAT());
}
```

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
The identifiers are descriptive	○	○	○	○	○
The test data is descriptive	○	○	○	○	○
The test makes sense in the domain	○	○	○	○	○
The intent of the test is easy to understand	○	○	○	○	○

Would it be possible for you to explain which parts are easy to understand and which parts are difficult to understand?

Criteria-Based-M 9 [Alfio]

According to the criteria, score the test as follows:

```
@Test
void totalPriceZeroIfVatStatusIsNull() {
    PriceContainerImpl vs = new PriceContainerImpl(1000, "CHF",
    assertEquals(BigDecimal.ZERO, vs.getFinalPrice());
}
```

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
The identifiers are descriptive	○	○	○	○	○
The test data is descriptive	○	○	○	○	○

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
The test makes sense in the domain	○	○	○	○	○
The intent of the test is easy to understand	○	○	○	○	○

Would it be possible for you to explain which parts are easy to understand and which parts are difficult to understand?

Criteria-Based-M 10 [Alfio]

According to the criteria, score the test as follows:

```
@Test
void testDoNotProduceNegativePrices() {
    var discount = mock(PromoCodeDiscount.class);
    when(discount.getCodeType()).thenReturn(PromoCodeDiscount.Co
    when(discount.getDiscountType()).thenReturn(PromoCodeDiscoun
    when(discount.getFixedAmount()).thenReturn(true);
    when(discount.getDiscountAmount()).thenReturn(3100);

    PriceContainerImpl vs = new PriceContainerImpl(1000, "CHF",
    assertEquals(new BigDecimal("0.00"), vs.getFinalPrice());
}
```

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
The identifiers are descriptive	○	○	○	○	○
The test data is descriptive	○	○	○	○	○
The test makes sense in the domain	○	○	○	○	○
The intent of the test is easy to understand	○	○	○	○	○

Would it be possible for you to explain which parts are easy to understand and which parts are difficult to understand?

Criteria-Based-M 11 [Alfio]

According to the criteria, score the test as follows:

```
@ParameterizedTest
@ValueSource(strings = {"select", "radio"})
void getValueDescriptionForSingleOptionField(String type) {
    when(description.getRestrictedValuesDescription()).thenRetur
    var field = new TicketFieldConfigurationDescriptionAndValue(
```

```
        assertEquals("simple value", field.getValueDescription());
    }
```

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
The identifiers are descriptive	○	○	○	○	○
The test data is descriptive	○	○	○	○	○
The test makes sense in the domain	○	○	○	○	○
The intent of the test is easy to understand	○	○	○	○	○

Would it be possible for you to explain which parts are easy to understand and which parts are difficult to understand?

Criteria-Based-M 12 [Alfio]

According to the criteria, score the test as follows:

```
@Test
    void confirmPaidReservation() {
        when(ticketReservationRepository.findOptionalStatusAndValida
        when(configurationManager.getFor(eq(ENABLE_TICKET_TRANSFER),
            new MaybeConfiguration(ENABLE_TICKET_TRANSFER)
        );
        when(configurationManager.getFor(eq(SEND_TICKETS_AUTOMATICAL
            new MaybeConfiguration(SEND_TICKETS_AUTOMATICALLY)
        );
        when(configurationManager.getFor(eq(BANKING_KEY), any())).th
        mockBillingDocument();
        testPaidReservation(true, true);
        verify(notificationManager).sendTicketByEmail(any(), any(),
    }
```

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
The identifiers are descriptive	○	○	○	○	○
The test data is descriptive	○	○	○	○	○
The test makes sense in the domain	○	○	○	○	○
The intent of the test is easy to understand	○	○	○	○	○

Would it be possible for you to explain which parts are easy to understand and which parts are difficult to understand?

Criteria-Based-E 1 [spring-testing]

According to the criteria, score the test as follows:

```
@Test(timeout = 4000)
  public void testCreatesPersonTakingNoArguments0()  throws Throwabl
      Person person0 = new Person();
      Person person1 = new Person();
      boolean boolean0 = person0.equals(person1);
      assertTrue(boolean0);
  }
```

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
The identifiers are descriptive	○	○	○	○	○
The test data is descriptive	○	○	○	○	○
The test makes sense in the domain	○	○	○	○	○
The intent of the test is easy to understand	○	○	○	○	○

Would it be possible for you to explain which parts are easy to understand and which parts are difficult to understand?

Criteria-Based-E 2 [spring-testing]

According to the criteria, score the test as follows:

```
@Test(timeout = 4000)
  public void testCreatesPersonTaking2Arguments1()  throws Throwable
      Person person0 = new Person("Kt(tyZA._U*Ce4A", "OM&}X$I!~QDSaI
      Person person1 = new Person("XS~*j;xHDF/,r", "OM&}X$I!~QDSaIxi
      boolean boolean0 = person0.equals(person1);
      assertFalse(boolean0);
      assertEquals("OM&}X$I!~QDSaIxi", person1.getLastName());
  }
```

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
The identifiers are descriptive	○	○	○	○	○
The test data is descriptive	○	○	○	○	○
The test makes sense in the domain	○	○	○	○	○

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
The intent of the test is easy to understand	○	○	○	○	○

Would it be possible for you to explain which parts are easy to understand and which parts are difficult to understand?

Criteria-Based-E 3 [petclinic]

According to the criteria, score the test as follows:

```
@Test
@Timeout(value = 4000 , unit = TimeUnit.MILLISECONDS)
public void testCreatesVetsAndCallsGetVetList()  throws Throwable
    Vets vets0 = new Vets();
    List<Vet> list0 = vets0.getVetList();
    assertTrue(list0.isEmpty());

    List<Vet> list1 = vets0.getVetList();
    assertEquals(0, list1.size());
}
```

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
The identifiers are descriptive	○	○	○	○	○
The test data is descriptive	○	○	○	○	○
The test makes sense in the domain	○	○	○	○	○
The intent of the test is easy to understand	○	○	○	○	○

Would it be possible for you to explain which parts are easy to understand and which parts are difficult to understand?

Criteria-Based-E 4 [petclinic]

According to the criteria, score the test as follows:

```
@Test
@Timeout(value = 4000 , unit = TimeUnit.MILLISECONDS)
public void testCreatesVetAndCallsSetSpecialtiesInternal()  throws
    Vet vet0 = new Vet();
    LinkedHashSet linkedHashSet0 = new LinkedHashSet();
    Specialty specialty0 = new Specialty();
    linkedHashSet0.add(specialty0);
    vet0.setSpecialtiesInternal(linkedHashSet0);
    Set set0 = vet0.getSpecialtiesInternal();
    assertEquals(1, set0.size());
}
```

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
The identifiers are descriptive	○	○	○	○	○
The test data is descriptive	○	○	○	○	○
The test makes sense in the domain	○	○	○	○	○
The intent of the test is easy to understand	○	○	○	○	○

Would it be possible for you to explain which parts are easy to understand and which parts are difficult to understand?

Criteria-Based-E 5 [petclinic]

According to the criteria, score the test as follows:

```
@Test
public void testGetPetTakingInteger()  throws Throwable  {
    Owner owner0 = new Owner();
    Pet pet0 = new Pet();
    owner0.addPet(pet0);
    Integer integer0 = new Integer(1);
    Pet pet1 = owner0.getPet(integer0);
    assertNull(pet1);
}
```

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
The identifiers are descriptive	○	○	○	○	○
The test data is descriptive	○	○	○	○	○
The test makes sense in the domain	○	○	○	○	○
The intent of the test is easy to understand	○	○	○	○	○

Would it be possible for you to explain which parts are easy to understand and which parts are difficult to understand?

Criteria-Based-E 6 [Alfio]

According to the criteria, score the test as follows:

```
@Test(timeout = 4000)
public void testEqualsReturningFalse()  throws Throwable  {
    ContentLanguage contentLanguage0 = ContentLanguage.TURKISH;
    ContentLanguage contentLanguage1 = ContentLanguage.BULGARIAN;
    boolean boolean0 = contentLanguage0.equals(contentLanguage1);
    assertFalse(boolean0);
    assertEquals(2048, contentLanguage1.getValue());
}
```

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
The identifiers are descriptive	○	○	○	○	○
The test data is descriptive	○	○	○	○	○
The test makes sense in the domain	○	○	○	○	○
The intent of the test is easy to understand	○	○	○	○	○

Would it be possible for you to explain which parts are easy to understand and which parts are difficult to understand?

Criteria-Based-E 7 [Alfio]

According to the criteria, score the test as follows:

```
@Test(timeout = 4000)
public void testIsHasFirstAndLastNameReturningTrue()  throws Throw
    CustomerName customerName0 = new CustomerName("fullName must n
    boolean boolean0 = customerName0.isHasFirstAndLastName();
    assertEquals("%lF>TAak~ ", customerName0.getFullName());
    assertEquals("%lF>TAak~ null", customerName0.toString());
    assertTrue(boolean0);
}
```

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
The identifiers are descriptive	○	○	○	○	○
The test data is descriptive	○	○	○	○	○
The test makes sense in the domain	○	○	○	○	○
The intent of the test is easy to understand	○	○	○	○	○

Would it be possible for you to explain which parts are easy to understand and which parts are difficult to understand?

Criteria-Based-E 8 [Alfio]

According to the criteria, score the test as follows:

```
@Test
public void testEqualsAndEqualsReturningFalse0()  throws Throwable
    SummaryRow.SummaryType summaryRow_SummaryType0 = SummaryRow.Su
    SummaryRow summaryRow0 = new SummaryRow((String) null, (String
    SummaryRow summaryRow1 = new SummaryRow((String) null, (String
    boolean boolean0 = summaryRow1.equals(summaryRow0);
    assertEquals("SummaryRow(name=null, price=null, priceBeforeVat
```

```
        assertFalse(boolean0);
    }
```

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
The identifiers are descriptive	○	○	○	○	○
The test data is descriptive	○	○	○	○	○
The test makes sense in the domain	○	○	○	○	○
The intent of the test is easy to understand	○	○	○	○	○

Would it be possible for you to explain which parts are easy to understand and which parts are difficult to understand?

Criteria-Based-E 9 [Alfio]

According to the criteria, score the test as follows:

```
@Test
public void testCompareToReturningPositive()  throws Throwable  {
    Integer integer0 = new Integer((-904));
    SerializablePair<em> serializablePair0 = SerializablePair.of(i
    Pair<em> pair0 = ImmutablePair.left(integer0);
    MutablePair<em> mutablePair0 = MutablePair.of((Map.Entry<em>)
    int int0 = serializablePair0.compareTo(mutablePair0);
    assertEquals(1, int0);
}
```

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
The identifiers are descriptive	○	○	○	○	○
The test data is descriptive	○	○	○	○	○
The test makes sense in the domain	○	○	○	○	○
The intent of the test is easy to understand	○	○	○	○	○

Would it be possible for you to explain which parts are easy to understand and which parts are difficult to understand?

Criteria-Based-E 10 [Alfio]

According to the criteria, score the test as follows:

```
@Test(timeout = 4000)
public void testCreatesItalianEInvoicingTaking5ArgumentsAndCallsGe
    TicketReservationInvoicingAdditionalInfo.ItalianEInvoicing.Ref
```

```
    TicketReservationInvoicingAdditionalInfo.ItalianEInvoicing tic
    String string0 = ticketReservationInvoicingAdditionalInfo_Ital
    assertEquals("p87XT28WJRgN", ticketReservationInvoicingAdditio
    assertFalse(ticketReservationInvoicingAdditionalInfo_ItalianEI
    assertEquals("p87XT28WJRgN", ticketReservationInvoicingAdditio
    assertNull(string0);
    assertEquals("<*6`s3Yn]n}g", ticketReservationInvoicingAdditio
}
```

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
The identifiers are descriptive	○	○	○	○	○
The test data is descriptive	○	○	○	○	○
The test makes sense in the domain	○	○	○	○	○
The intent of the test is easy to understand	○	○	○	○	○

Would it be possible for you to explain which parts are easy to understand and which parts are difficult to understand?

Criteria-Based-E 11 [Insurance]

According to the criteria, score the test as follows:

```
@Test
@Timeout(value = 4000 , unit = TimeUnit.MILLISECONDS)
public void testCreatesPercentMarkupRuleTaking3Arguments()  throws
    Tariff tariff0 = new Tariff("");
    BigDecimal bigDecimal0 = BigDecimal.TEN;
    PercentMarkupRule percentMarkupRule0 = new PercentMarkupRule(t
    List list0 = List.of("");
    BasicObjectRouteMatch basicObjectRouteMatch0 = new BasicObject
    Map map0 = basicObjectRouteMatch0.getVariableValues();
    Calculation calculation0 = new Calculation("", (LocalDate) nul
    Calculation calculation1 = percentMarkupRule0.apply(calculatio
    assertSame(calculation1, calculation0);
}
```

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
The identifiers are descriptive	○	○	○	○	○
The test data is descriptive	○	○	○	○	○
The test makes sense in the domain	○	○	○	○	○
The intent of the test is easy to understand	○	○	○	○	○

Would it be possible for you to explain which parts are easy to understand and which parts are difficult to understand?

Criteria-Based-E 12 [Insurance]

According to the criteria, score the test as follows:

```
@Test
@Timeout(value = 4000 , unit = TimeUnit.MILLISECONDS)
public void testGetCodeReturningNonEmptyString()  throws Throwable
    Question question0 = new Question("tV&T$PzB[5", 1922, "L{S");
    String string0 = question0.getCode();
    assertEquals("L{S", question0.getText());
    assertEquals("tV&T$PzB[5", string0);
    assertEquals(1922, question0.getIndex());
}
```

	Strongly agree	Somewhat agree	Neither agree nor disagree	Somewhat disagree	Strongly disagree
The identifiers are descriptive	○	○	○	○	○
The test data is descriptive	○	○	○	○	○
The test makes sense in the domain	○	○	○	○	○
The intent of the test is easy to understand	○	○	○	○	○

Would it be possible for you to explain which parts are easy to understand and which parts are difficult to understand?

Powered by Qualtrics