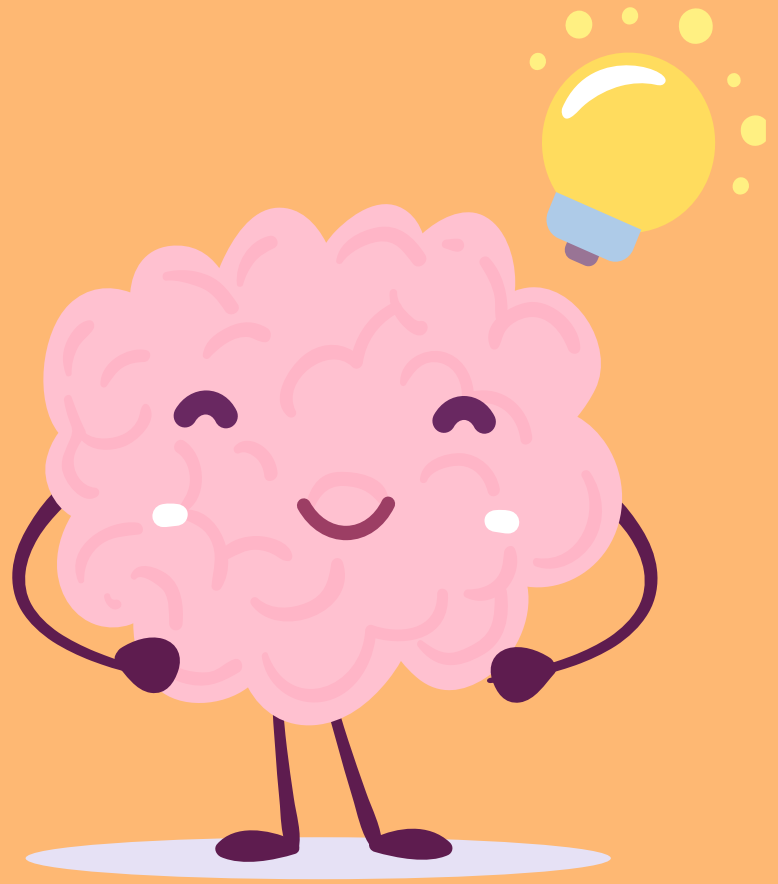
https://TED.DEV/about

REFACTORING TESTS

TED M. YOUNG
technical coach
live coder
board game designer

Ask Questions As You Need



I may defer the answer if I'm going to cover it later!

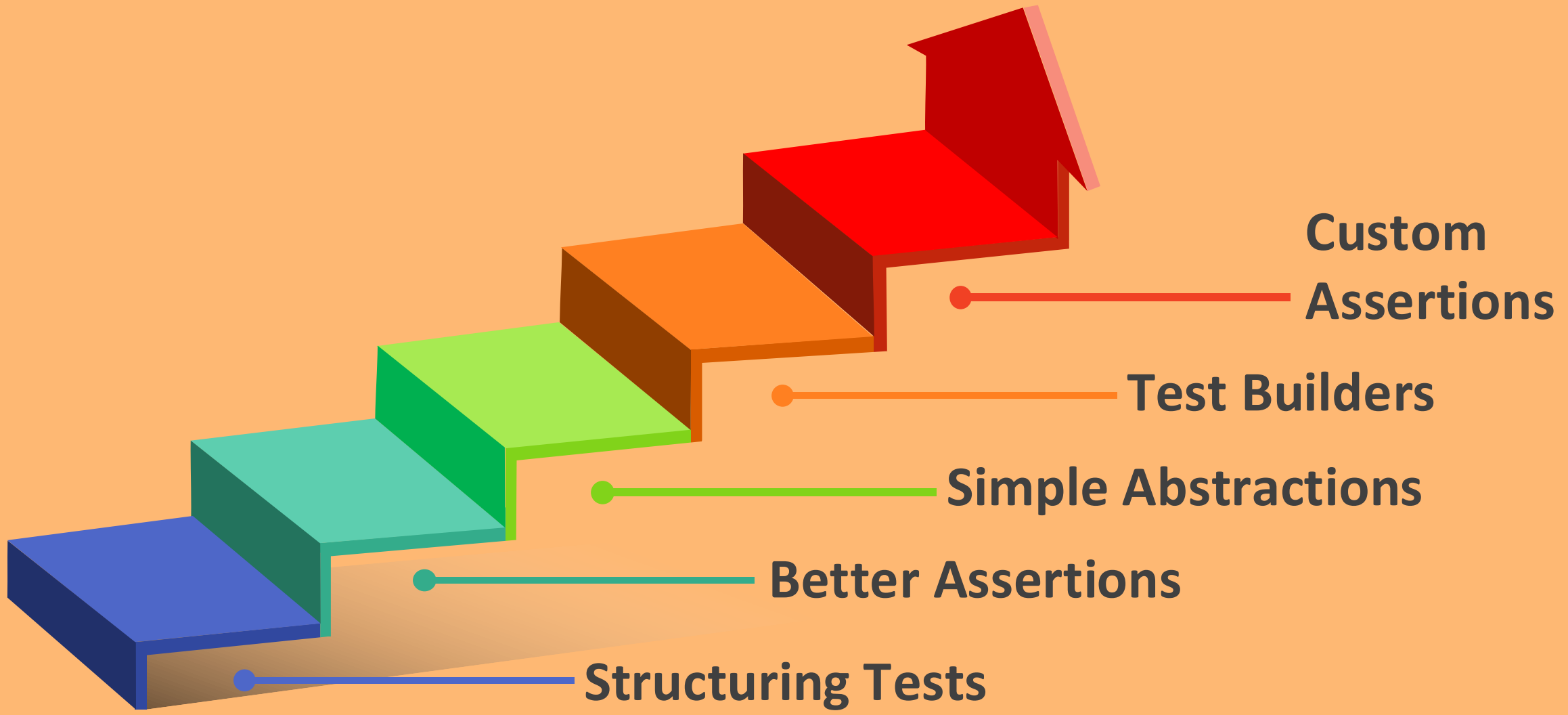
WARNING: Java Ahead

Hi, I'm Ted and
I'm a Java Developer

JUnit

AssertJ

Ladder of Refactoring Tests



What makes a...

Good Test Suite?

Good Test Suites…

Safety Net for System **Behavior**

Flexible to support refactoring

Living Documentation (good naming with ubiquitous language)

Comprehensive (not just code coverage!)

Helps with discipline (guardrails to good developer behavior)

Test Principles

FIRST Principles (of Tests)

Brett Schuchert, Tim Ottinger

F - fast

I - isolated (independent)

R - repeatable

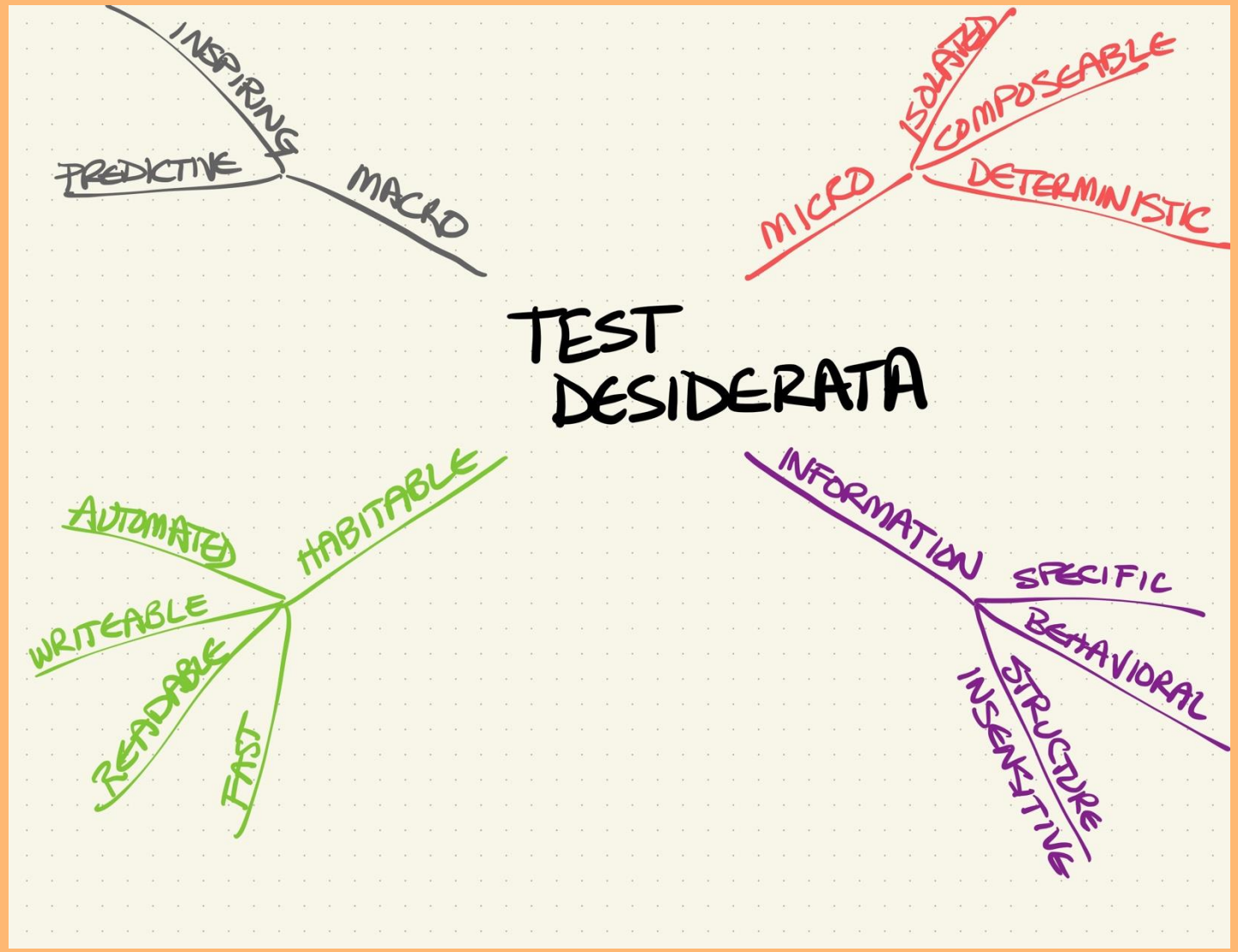
S - self-validating

T - timely

https://agileinaflash.blogspot.com/2009/02/first.html

Test Desiderata

Kent Beck



- ✓ Isolated
- ✓ Composable
- ✓ Deterministic
- ✓ Fast
- ✓ Writable
- ✓ Readable
- ✓ Behavioral
- ✓ Structure-insensitive
- ✓ Automated
- ✓ Specific
- ✓ Predictive
- ✓ Inspiring

Test Structures

- Arrange, Act, Assert[1] (3A or AAA)
- Given, When, Then (BDD-terminology)[2]
- Assemble, Activate, Assert[3]
- Setup, Execute, Verify[, Teardown][4]

[1] Bill Wake, 2001 (https://xp123.com/3a-arrange-act-assert/)

[2] Daniel Terhorst-North, 2006 (https://dannorth.net/blog/introducing-bdd/)

[3] C2 Wiki (https://wiki.c2.com/?AssembleActivateAssert)

[4] Gerard Meszaros, *xUnit Test Patterns* (book)

The *SEA* Test Structure

Setup - prepare object(s) for execution

Execute - invoke a Command (trigger action)

(sometimes: Create an Object]

Assert - what do we Expect to be Observed?

Basic SEA Test

```
@Test
void whenCompletingEnsembleThenEnsembleIsCompleted() {
    Ensemble ensemble = new Ensemble("completed", ZonedDateTime.now());

    ensemble.complete();

    assertThat(ensemble.isCompleted())
            .isTrue();
}
```

Another Basic SEA Test

```
@Test
void fromWhatShouldItDoWhenDiscardCardReturnsHowWillYouKnowItDidIt() {
    HexTile startingHexTile = HexTile.WHAT_SHOULD_IT_DO;

    HexTile nextHexTile = startingHexTile.cardDiscarded();

    assertThat(nextHexTile)
            .isEqualByComparingTo(HexTile.HOW_WILL_YOU_KNOW_IT_DID_IT);
}
```

SEA Tests With No "Setup"

```java
@Test
void newEnsembleIsNotCanceled() {
    Ensemble ensemble = new Ensemble("ensemble", ZonedDateTime.now());

    assertThat(ensemble.isCanceled())
            .isFalse();
}

@Test
void newEnsembleIsNotCompleted() {
    Ensemble ensemble = new Ensemble("not completed", ZonedDateTime.now());

    assertThat(ensemble.isCompleted())
            .isFalse();
}
```

SEA Test With Embedded Execute

```java
@Test
void cancelCompletedEnsembleThrowsException() {
    Ensemble ensemble = new Ensemble("completed", ZonedDateTime.now());
    ensemble.complete();

    assertThatThrownBy(ensemble::cancel)
            .isInstanceOf(EnsembleCompleted.class);
}
```

Test Smells

Like *Code Smells*, but for Test Code

smell: Hidden Setup

```java
@BeforeEach
public void setUp() {
    bagel1 = new Bagel(EVERYTHING, HERB_GARLIC_CREAM_CHEESE, true);
    cookie1 = new Cookie(CHOCOLATE_CHIP, true);
    donut1 = new Donut(GLAZED);
    americano = new Americano(DrinkTemperature.HOT);
    itemList = List.of(bagel1, cookie1, donut1, americano);
    coffeeShopOrder = new CoffeeShopOrder("Emilie", itemList);
}
```

```java
@Test
public void generateReceiptForFoodItemsTest() {
    String expectedReceipt = """
            Bagel: EVERYTHING $2.5
            Cookie: CHOCOLATE_CHIP $1.25
            Donut: GLAZED $1.75
            Total: $5.5""";

    assertEquals(
            expectedReceipt,
            coffeeShopOrder.generateReceiptForFoodItems());
}
```

fix: Inline Setup

```java
@Test
public void generateReceiptForFoodItemsTest() {
    Bagel bagel = new Bagel(EVERYTHING, HERB_GARLIC_CREAM_CHEESE, true);
    Cookie cookie = new Cookie(CHOCOLATE_CHIP, true);
    Donut donut = new Donut(GLAZED);
    Americano americano = new Americano(DrinkTemperature.HOT);
    List<Item> itemList = List.of(bagel, cookie, donut, americano);
    CoffeeShopOrder coffeeShopOrder = new CoffeeShopOrder("Emilie", itemList);

    String expectedReceipt = """
            Bagel: EVERYTHING $2.5
            Cookie: CHOCOLATE_CHIP $1.25
            Donut: GLAZED $1.75
            Total: $5.5""";

    assertEquals(
            expectedReceipt,
            coffeeShopOrder.generateReceiptForFoodItems());
}
```

smell: Unclear/Undefined Setup Details

```
@Test
void whenCancelingScheduledEnsembleThenEnsembleIsCanceled() {
    Ensemble ensemble = new Ensemble("To be Canceled", ZonedDateTime.now());

    ensemble.cancel();

    assertThat(ensemble.isCanceled())
            .isTrue();
}
```

fix: Add Useful Details

```
@Test
void whenCompletingEnsembleThenEnsembleIsCompleted() {
    ZonedDateTime ensembleStartDateTime = ZonedDateTime.now();
    Ensemble ensemble = new Ensemble("completed", ensembleStartDateTime);

    ensemble.complete();

    assertThat(ensemble.isCompleted())
            .isTrue();
}
```

smell: Unclear Assertion Message

```
@Test
void whenCancelingScheduledEnsembleThenEnsembleIsCanceled() {
    Ensemble ensemble = new Ensemble("To be Canceled", ZonedDateTime.now());

    ensemble.cancel();

    assertThat(ensemble.isCanceled())
            .isTrue();
}
```

```
org.opentest4j.AssertionFailedError:
Expecting value to be true but was false
Expected :true
Actual   :false
```

fix: Clarify Assertion Message

```
@Test
void whenCompletingEnsembleThenEnsembleIsCompleted() {
    Ensemble ensemble = new Ensemble("completed", ZonedDateTime.now());

    ensemble.complete();

    assertThat(ensemble.isCompleted())
            .as("Ensemble should have been completed, but was not.")
            .isTrue();
}
```

```
org.opentest4j.AssertionFailedError: [Ensemble should have been completed, but was not.]
Expecting value to be true but was false
Expected :true
Actual   :false
```

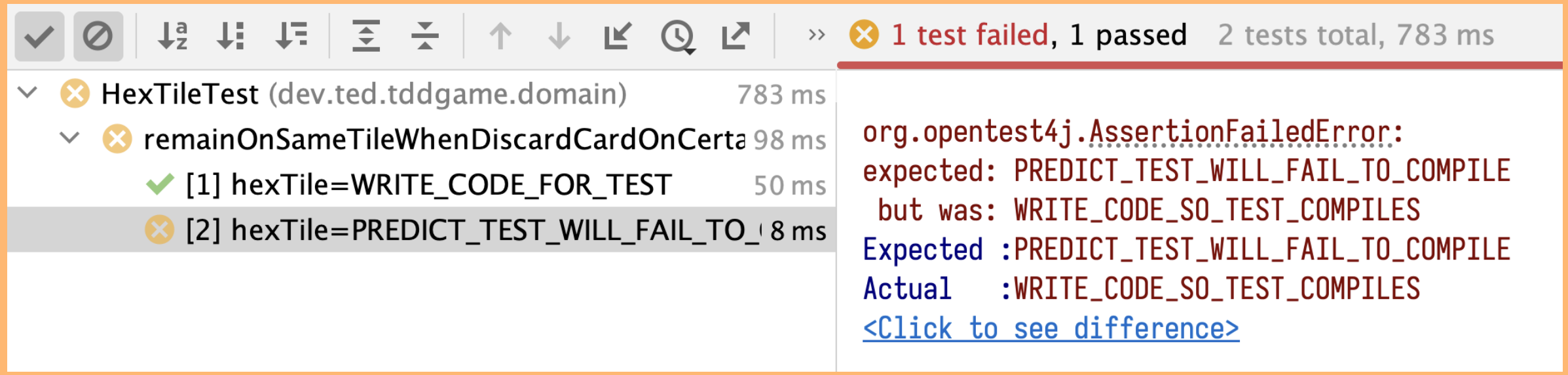
smell: Very Similar Tests

```
@Test
void remainOnSameTileWhenDiscardCardOnWriteCodeForTest() {
    HexTile startingHexTile = HexTile.WRITE_CODE_FOR_TEST;

    HexTile nextTile = startingHexTile.cardDiscarded();

    assertThat(nextTile)
            .isEqualByComparingTo(startingHexTile);
}

@Test
void remainOnSameTileWhenDiscardCardOnPredictTestWillFailToCompile() {
    HexTile startingHexTile = HexTile.PREDICT_TEST_WILL_FAIL_TO_COMPILE;

    HexTile nextTile = startingHexTile.cardDiscarded();

    assertThat(nextTile)
            .isEqualByComparingTo(startingHexTile);
}
```

fix: Parameterize Differences

```
@ParameterizedTest
@EnumSource(names = {
        "WRITE_CODE_FOR_TEST",
        "PREDICT_TEST_WILL_FAIL_TO_COMPILE"
})
void remainOnSameTileWhenDiscardCard(HexTile startingHexTile) {
    HexTile nextTile = startingHexTile.cardDiscarded();

    assertThat(nextTile)
            .isEqualByComparingTo(startingHexTile);
}
```

smell: Split Attention



fix: Add Description!

```
org.opentest4j.AssertionFailedError: [From tile PREDICT_TEST_WILL_FAIL_TO_COMPILE,
 discard should result in remaining on that tile.]
expected: PREDICT_TEST_WILL_FAIL_TO_COMPILE
 but was: WRITE_CODE_SO_TEST_COMPILES
Expected :PREDICT_TEST_WILL_FAIL_TO_COMPILE
Actual   :WRITE_CODE_SO_TEST_COMPILES
<Click to see difference>
```

I'm lazy: I often skip this step, but then regret it.

LLMs also seems lazy, because they fail to do this as well!

smell: Verbose Example

```
@Test
public void testPerson() throws Exception {
    Person rick = new Person("Rick", LocalDate.now().minusYears(70));
    Person morty = new Person("Morty", LocalDate.now().minusYears(14));
    Person beth = new Person("Beth", LocalDate.now().minusYears(35));
    Person jerry = new Person("Jerry", LocalDate.now().minusYears(35));
    String actual = JStachio.render(
            new HelloWorld("Hello alien", List.of(rick, morty, beth, jerry)));
    String expected = """
            Hello alien Rick! You are 70 years old!
            Hello alien Morty! You are 14 years old!
            Hello alien Beth! You are 35 years old!
            Hello alien Jerry! You are 35 years old!
            That is all for now!
            """;
    assertEquals(expected, actual);
}
```

fix: Eliminate Unnecessary Stuff

```
@Test
public void testPerson() throws Exception {
    Person rick = new Person("Rick", LocalDate.now().minusYears(70));
    Person morty = new Person("Morty", LocalDate.now().minusYears(14));
    String actual = JStachio.render(
            new HelloWorld("Hello alien", List.of(rick, morty)));
    String expected = """
            Hello alien Rick! You are 70 years old!
            Hello alien Morty! You are 14 years old!
            That is all for now!
            """;
    assertEquals(expected, actual);
}
```

Refactoring Setup

```
@Test
public void handWithOneAceAndOtherCardValuedLessThan10ThenAceIsValuedAt11() {
    List<Card> cards = new ArrayList<>(List.of(
            new Card(Suit.CLUBS, Rank.ACE),
            new Card(Suit.CLUBS, Rank.NINE)
    ));
    Hand hand = new Hand(cards);

    assertThat(hand.value())
            .isEqualTo(11 + 9);
}
```

✍ HandValueAceBeforeTest

Evident Data

Evident Data seems to be an exception to the rule that you don't want magic numbers in your code.

Kent Beck

TDD By Example (Ch. 25)

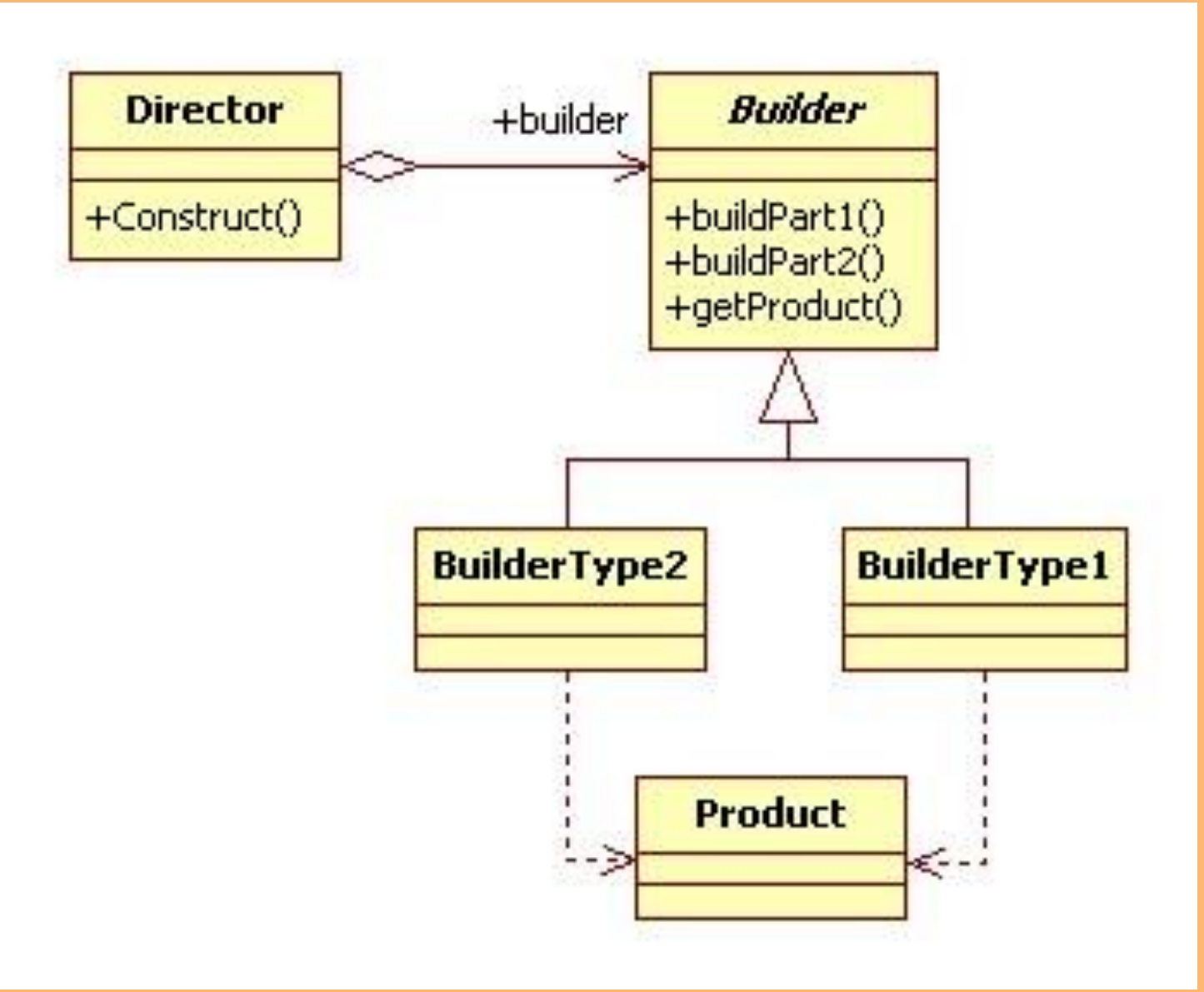
Blackjack Hand Value Calculation

```java
@Test
public void handWithOneAceAndOtherCardsValuedAt10ThenAceIsValuedAt11() {
    Hand hand = createHand(Rank.ACE, Rank.TEN);

    assertThat(hand.value())
            .isEqualTo(21);
}
```

```java
@Test
public void handWithOneAceAndOtherCardsValuedAt10ThenAceIsValuedAt11() {
    Hand hand = createHand(Rank.ACE, Rank.TEN);

    assertThat(hand.value())
            .isEqualTo(11 + 10);
}
```

Test Data Builders

"Gang of Four" Builder Pattern



(Joshua) Bloch Builder Pattern

Essentially the fluent

(or *chained*) API

```
public class NutritionFacts {
    private final int servingSize;
    private final int servings;
    private final int calories;
    // etc.
    private NutritionFacts(Builder builder) {
        servingSize = builder.servingSize;
        servings = builder.servings;
        calories = builder.calories;
        // etc.
    }
```

```
public static class Builder {
    private final int servingSize;
    private final int servings;
    private int calories = 0; // etc.

    // ensure required properties are set
    public Builder(int servingSize, int servings) {
        this.servingSize = servingSize;
        this.servings = servings;
    }

    public Builder calories(int val) {
        calories = val;
        return this;
    }
    // other config methods
    public NutritionFacts build() {
        return new NutritionFacts(this);
    }
}
```

(Joshua) Bloch Builder Pattern

```java
public static class Builder {
    private final int servingSize;
    private final int servings;
    private int calories = 0; // etc.

    // ensure required properties are set
    public Builder(int servingSize, int servings) {
        this.servingSize = servingSize;
        this.servings = servings;
    }

    public Builder calories(int val) {
        calories = val;
        return this;
    }
    // other config methods
    public NutritionFacts build() {
        return new NutritionFacts(this);
    }
}
```

```java
NutritionFacts cocaCola =
        new NutritionFacts.Builder(240, 8)
                .calories(100)
                .sodium(35)
                .carbohydrate(27)
                .build();
```

Customizer

```
Stream<ConcertEvent> concertEventStream =
        MakeEvents.with()
                .concertScheduled(
                        concertId,
                        concert -> concert
                                .ticketPrice(35)
                                .ticketsSold(6))
                .stream();
```

```
Stream<ConcertEvent> concertScheduledStream =
        MakeEvents.with()
                .concertScheduled(
                        firstConcertId,
                        c -> c.showDateTime(firstShowDateTime))
                .concertScheduled(
                        secondConcertId,
                        c -> c.showDateTime(secondShowDateTime))
                .stream();
```

```
Stream<ConcertEvent> concertScheduledStream =
        MakeEvents.with()
                .concertScheduled(
                        concertId,
                        c -> c.showDateTime(showDateTime)
                                .ticketSalesStopped())
                .stream();
```

```
public MakeEvents concertScheduled(
        ConcertId concertId,
        Function<ConcertCustomizer, ConcertCustomizer> concertCustomizer) {
    ConcertCustomizer customizer = concertCustomizer.apply(new ConcertCustomizer());
    ConcertScheduled concertScheduled =
            createConcertScheduled(concertId,
                                   customizer.ticketPrice,
                                   customizer.artistName,
                                   customizer.showDateTime);
    events.add(concertScheduled);
    customizer.ticketsSoldQuantity
            .stream() Stream<Integer>
            .map(qty -> new TicketsSold(
                    concertId,
                    eventSequenceIterator.next(),
                    qty,
                    qty * customizer.ticketPrice)) Stream<TicketsSold>
            .forEach(events::add);
    if (customizer.isTicketSalesStopped()) {
        events.add(new TicketSalesStopped(
                concertId, eventSequenceIterator.next()));
    }
    if (customizer.rescheduledShowDateTime != null) {
        reschedule(concertId, customizer.rescheduledShowDateTime,
                   customizer.rescheduledShowDateTime.minusHours(1).toLocalTime());
    }
    return this;
}
```

Assertions are Better with AssertJ...

Exception Asserts

```java
@Test
void listWithDuplicateNumbersThrowsException() {
    List<Integer> cardOrderIndexes = List.of(1, 1);

    // code first, then exception and message
    assertThatThrownBy(() -> new OrderedDeck(cardOrderIndexes))
            .isExactlyInstanceOf(IllegalArgumentException.class)
            .hasMessage("Found duplicate card indexes");

    // exception first, then code, then message
    assertThatExceptionOfType(IllegalArgumentException.class)
            .isThrownBy(() -> new OrderedDeck(cardOrderIndexes))
            .withMessage("Found duplicate card indexes");

    // dedicated to specific exceptions (sugary!)
    assertThatIllegalArgumentException()
            .isThrownBy(() -> new OrderedDeck(cardOrderIndexes))
            .withMessage("Found duplicate card indexes");
}
```

Blackjack: Face Up/Down Cards

Blackjack Game Standard Assert

```
@Test void playerStandsDealerCardsFaceUp() {
    Deck deck = StubDeckBuilder.playerCountOf(1)
                        .addPlayerWithRanks(Rank.TEN, Rank.JACK)
                        .buildWithDealerDoesNotDrawCards();
    Game game = GameFactory.createOnePlayerGamePlaceBetsInitialDeal(deck);

    game.playerStands();

    assertThat(game.dealerHand().cards())
        .allMatch(card -> !card.isFaceDown());
}
```

With Predicate

```
@Test void playerStandsDealerCardsFaceUp() {
    Deck deck = StubDeckBuilder.playerCountOf(1)
                        .addPlayerWithRanks(Rank.TEN, Rank.JACK)
                        .buildWithDealerDoesNotDrawCards();
    Game game = GameFactory.createOnePlayerGamePlaceBetsInitialDeal(deck);

    game.playerStands();

    Predicate<Card> faceUpCardPredicate = card -> !card.isFaceDown();
    assertThat(game.dealerHand().cards())
        .allMatch(faceUpCardPredicate);
}
```

Encapsulate Assertion

```
@Test void playerStandsDealerAllCardsFaceUp_predicate() {
   Deck deck = StubDeckBuilder.playerCountOf(1)
                       .addPlayerWithRanks(Rank.TEN, Rank.JACK)
                       .buildWithDealerDoesNotDrawCards();
   Game game = GameFactory.createOnePlayerGamePlaceBetsInitialDeal(deck);

   game.playerStands();

   assertAllDealerCardsFaceUp(game);
}
```

```
static void assertAllDealerCardsFaceUp(Game game) {
   assertThat(game.dealerHand().cards())
            .allMatch(card -> !card.isFaceDown());
}
```

Blackjack Game Custom Asserts

```
@Test void playerStandsDealerCardsFaceUp() {
    Deck deck = StubDeckBuilder.playerCountOf(1)
                        .addPlayerWithRanks(Rank.TEN, Rank.JACK)
                        .buildWithDealerDoesNotDrawCards();
    Game game = GameFactory.createOnePlayerGamePlaceBetsInitialDeal(deck);

    game.playerStands();

    assertThat(game)
        .dealerHand()
        .allCardsFaceUp();
}
```

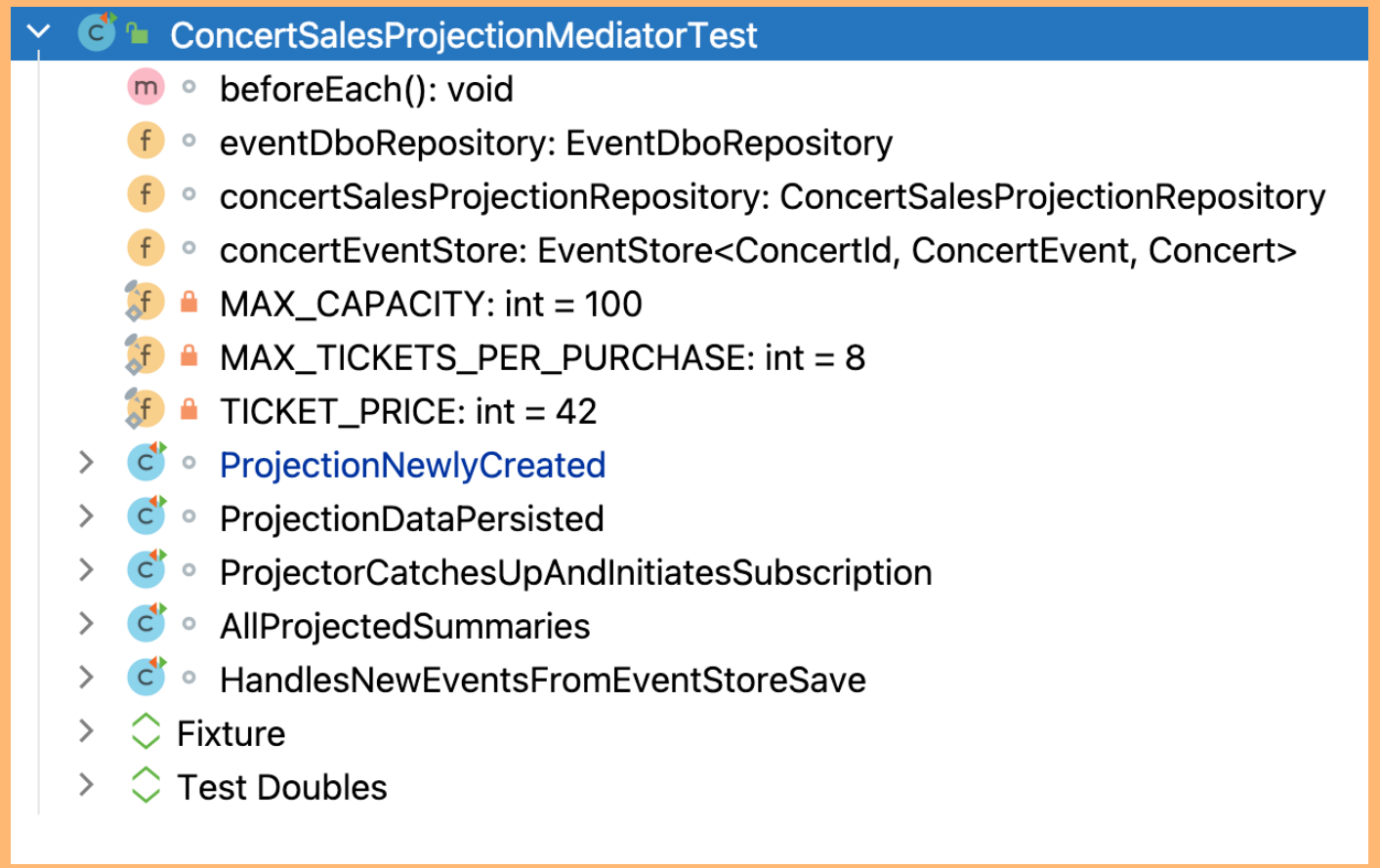
Refactoring to Fixture Record

```
private static Fixture createForPurchaseTicketsWithCapacityOf(int originalConcertAvailableTicketCount) {
    Customer customer = CustomerFactory.newlyRegistered();
    Concert concertBefore = ConcertFactory.createWithCapacity(originalConcertAvailableTicketCount);
    var concertStore = InMemoryEventStore.forConcerts();
    concertStore.save(concertBefore);
    ConcertId concertId = concertBefore.getId();
    var customerStore = InMemoryEventStore.forCustomers();
    customerStore.save(customer);
    return new Fixture(customer, concertStore, concertId, customerStore);
}

private record Fixture(
        Customer customer,
        EventStore<ConcertId, ConcertEvent, Concert> concertStore,
        ConcertId concertId,
        EventStore<CustomerId, CustomerEvent, Customer> customerStore
) {}

private static Fixture createForPurchaseTickets() {
    return createForPurchaseTicketsWithCapacityOf(42);
}
```

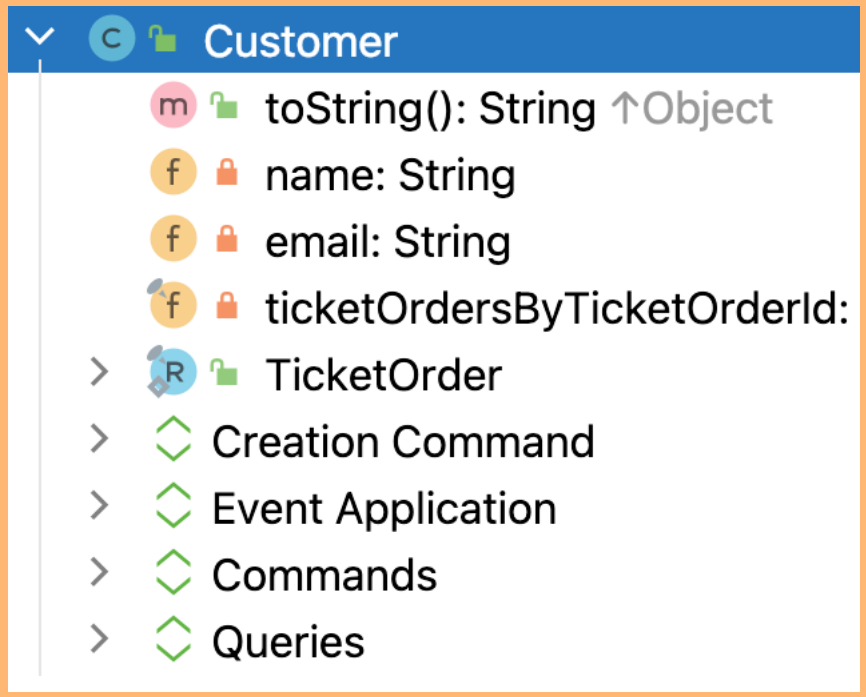
Fixture Inside Production Code??

```
@Controller
class ScheduleConcertController {
...
    static Fixture createForTest() {
        var concertEventStore = InMemoryEventStore.forConcerts();
        var projectionCoordinator = new ProjectionCoordinator<>(
                new ScheduledConcertsProjector(),
                new MemoryScheduledConcertsProjectionPersistence(),
                concertEventStore);
        Commands commands = new Commands(
                CommandExecutorFactory.create(concertEventStore),
                projectionCoordinator);
        ScheduleConcertController scheduleConcertController =
                new ScheduleConcertController(commands.createScheduleCommand());
        return new Fixture(concertEventStore, scheduleConcertController);
    }

    record Fixture(
            EventStore<ConcertId, ConcertEvent, Concert> concertEventStore,
            ScheduleConcertController controller) {}
```

Organizing Tests



...and code!



Tests are Specific, Code is General

Duplication, literals (evident data)

- Tests 
- Code 

Abstractions

- Tests 
- Code 

What Questions Do You Have??

Tests are code, too, and need the same amount of attention for refactoring

https://TED.DEV/about

https://TED.DEV/talks

THANK YOU

You've Been a Great Audience