LUXOFT TRAINING

Разработка через тестирование
Тесты на поведение

Test Driven Development

Ivan Dyachenko <IDyachenko@luxoft.com>

Module 4

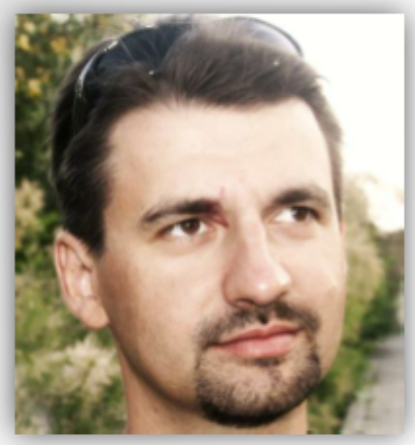
Содержание

LUXOFT Training Center

LUXOFT Training Center

Тесты на поведение супротив тестов на состояние

Пример

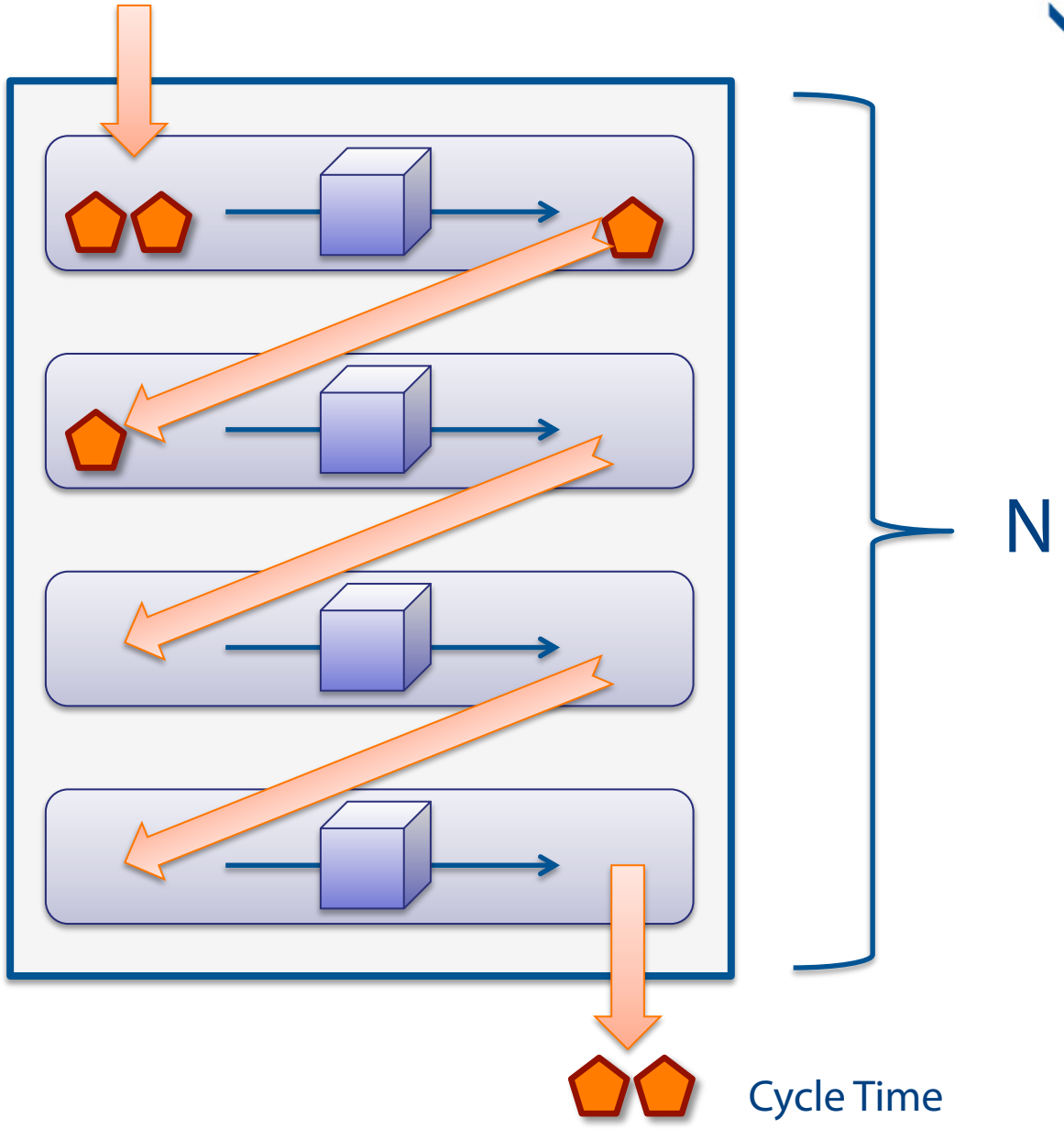


Андрей Бибичев

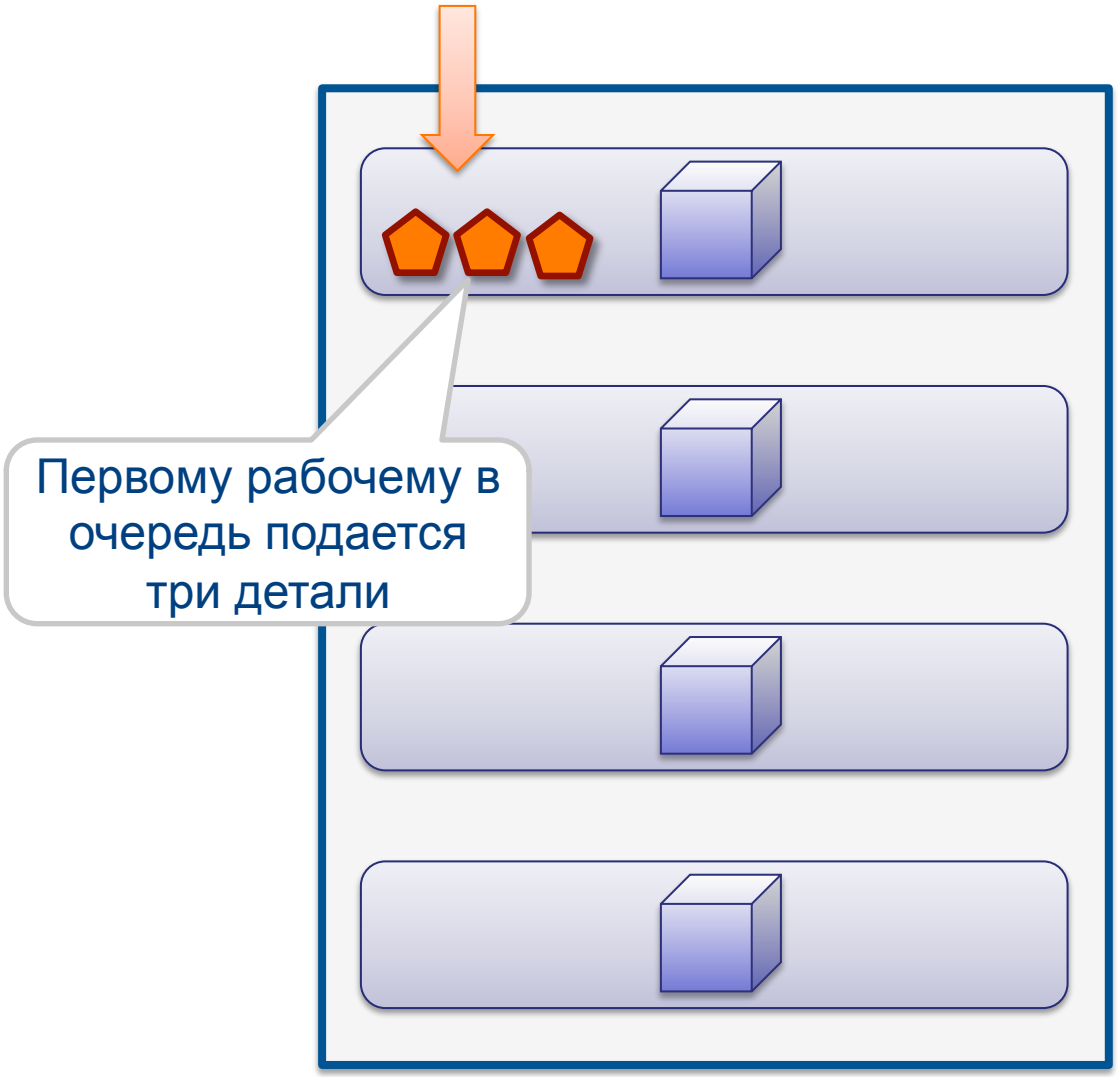
Conveyor - сравнение тестов на поведение и тестов на состояние

http://www.slideshare.net/bibigine

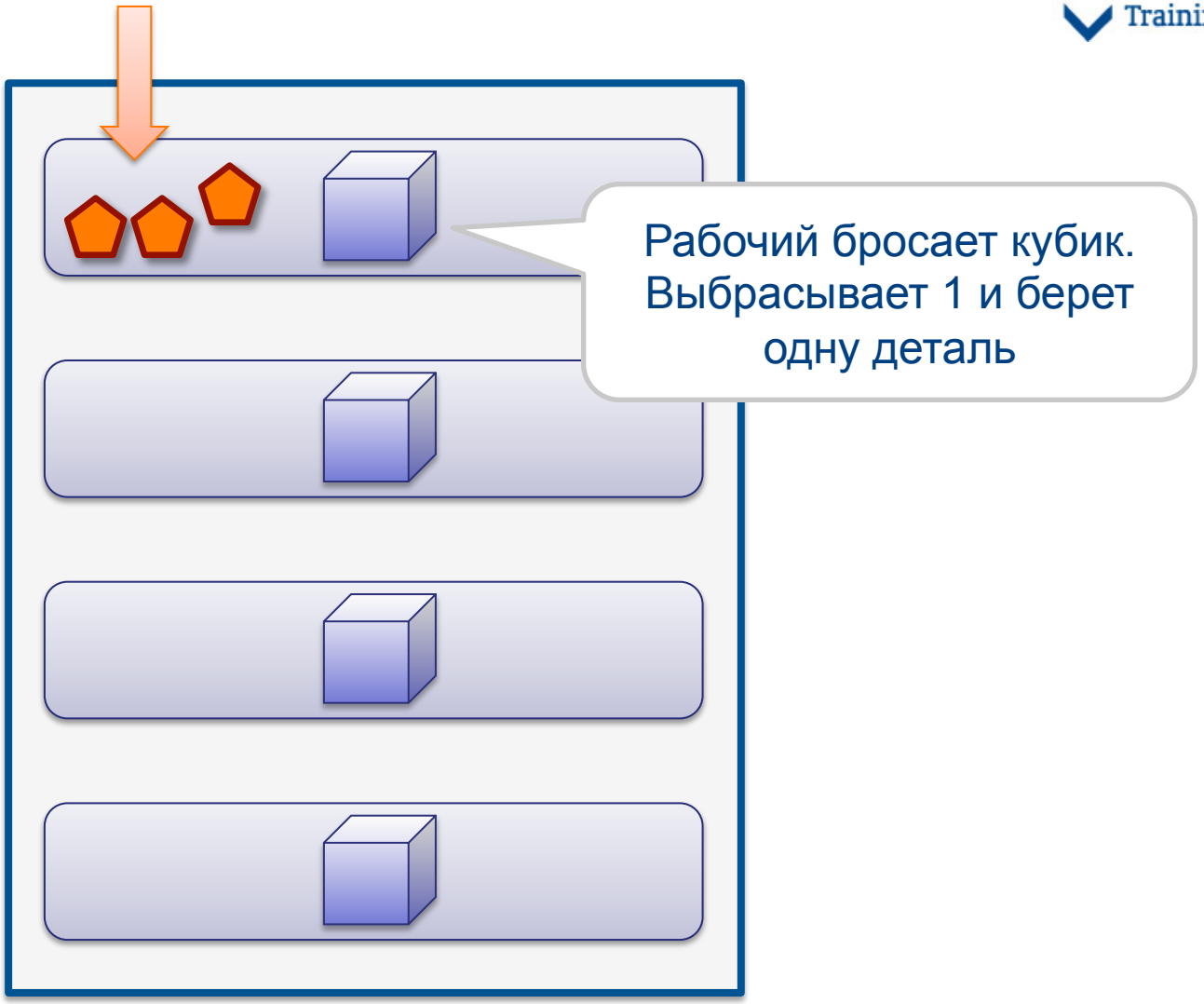
Конвейер



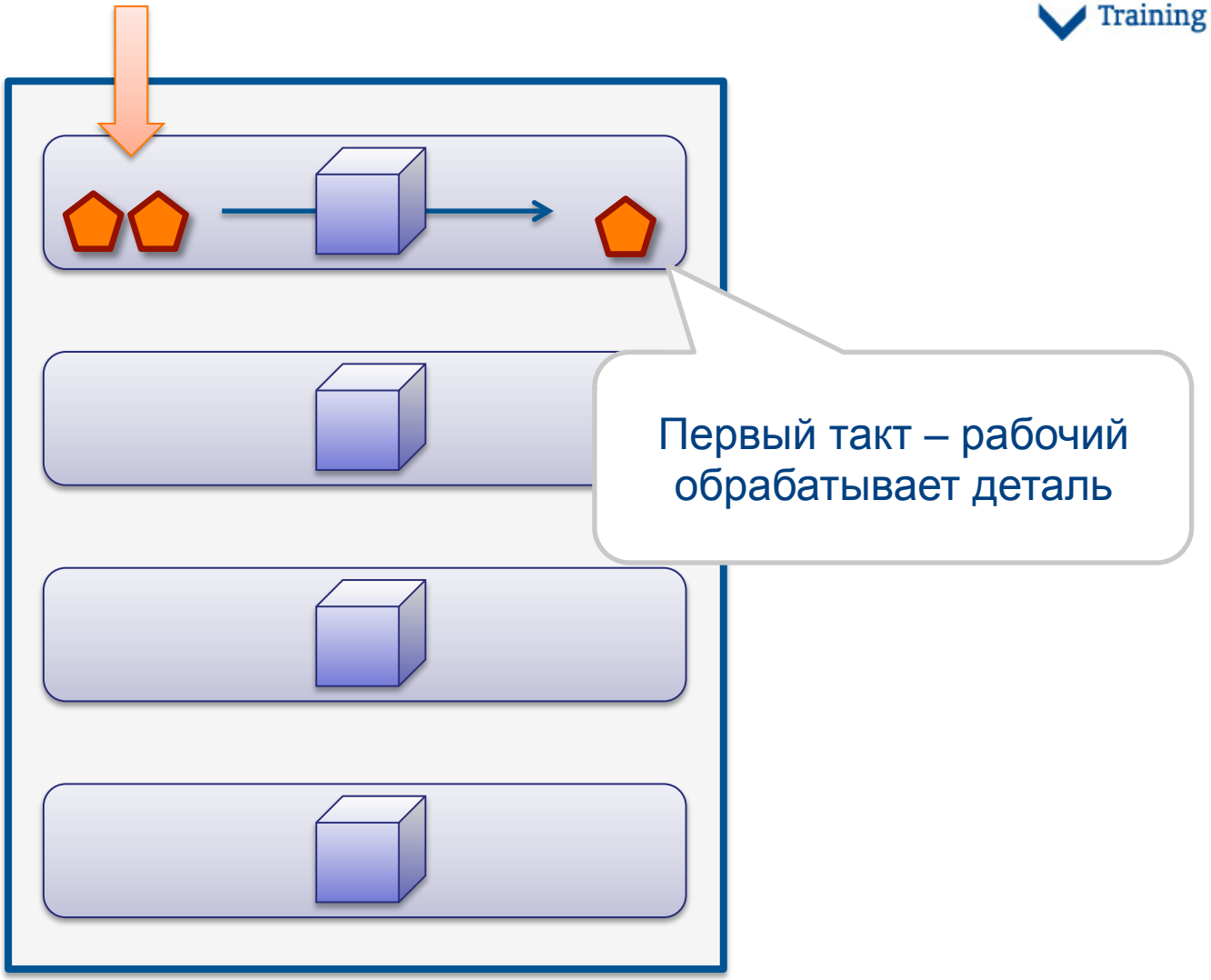
Контрольный пример



Контрольный пример

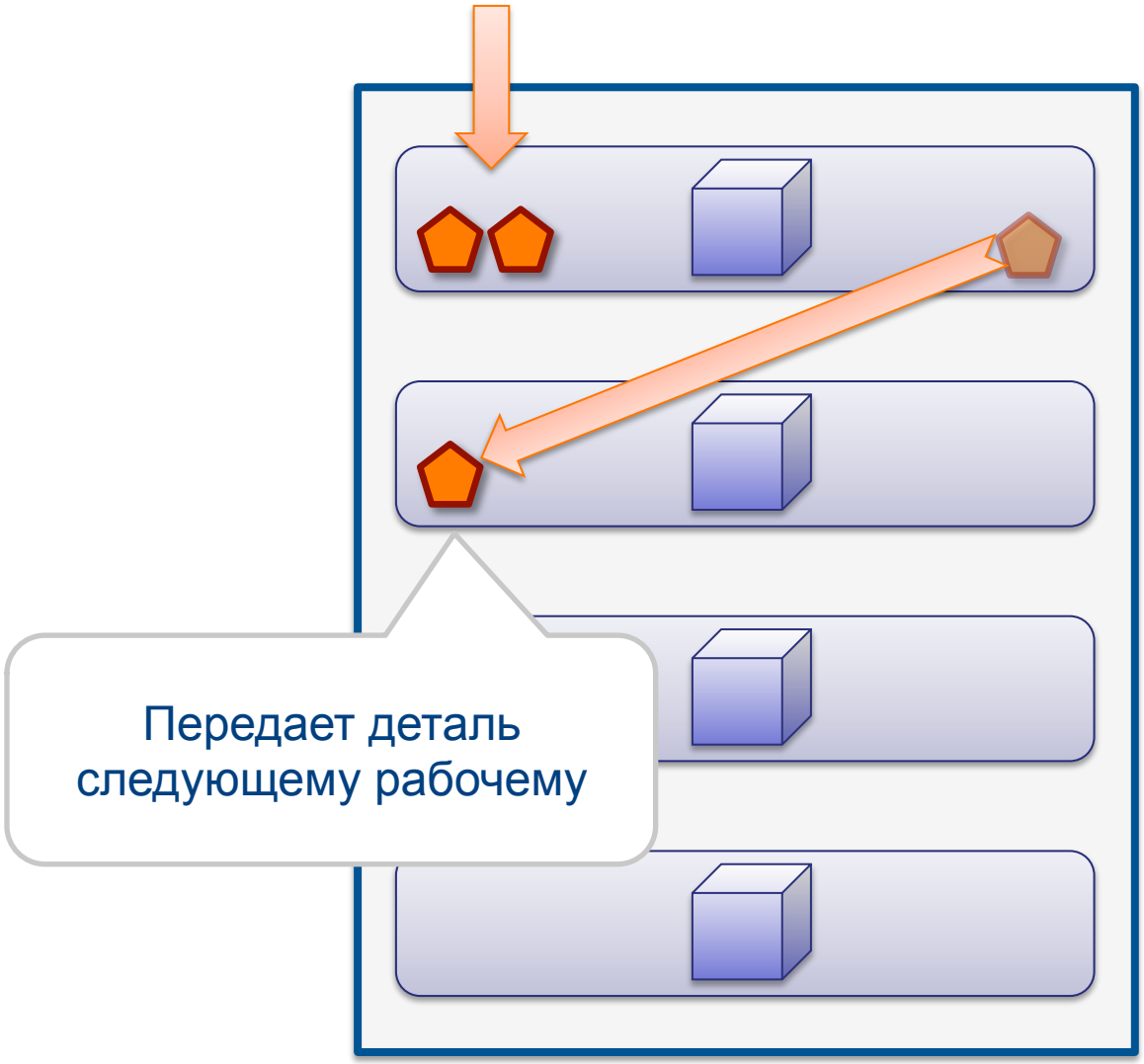


Контрольный пример

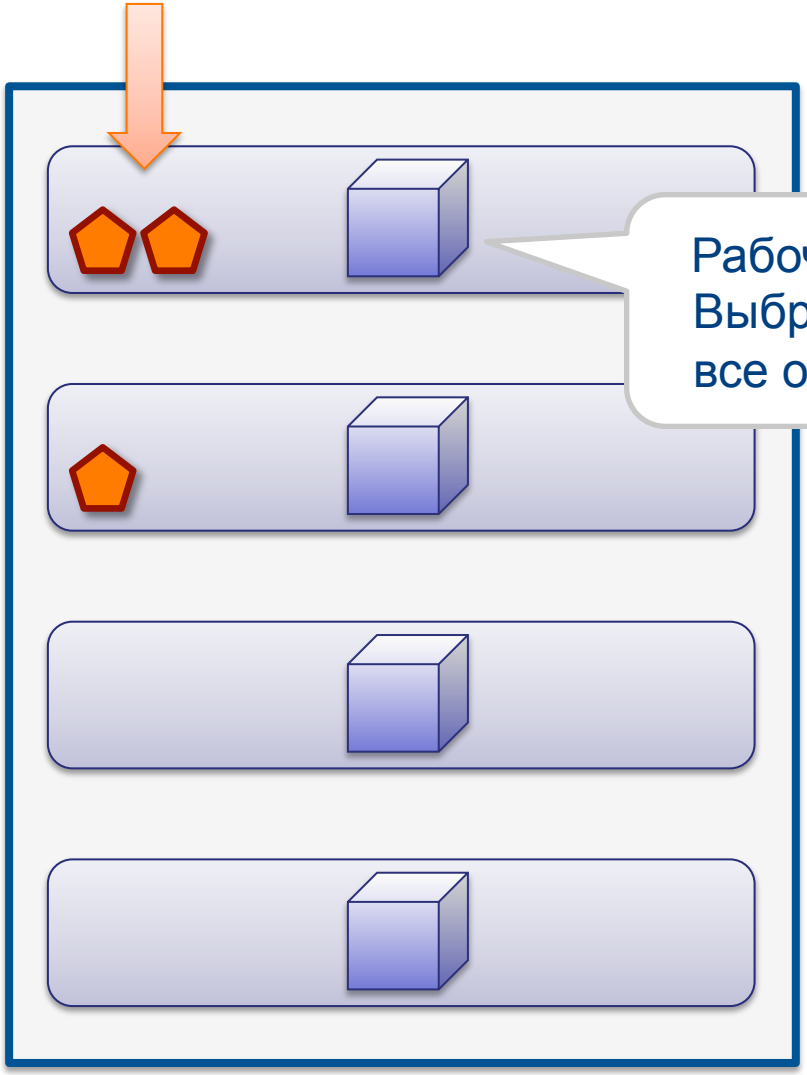


Контрольный пример

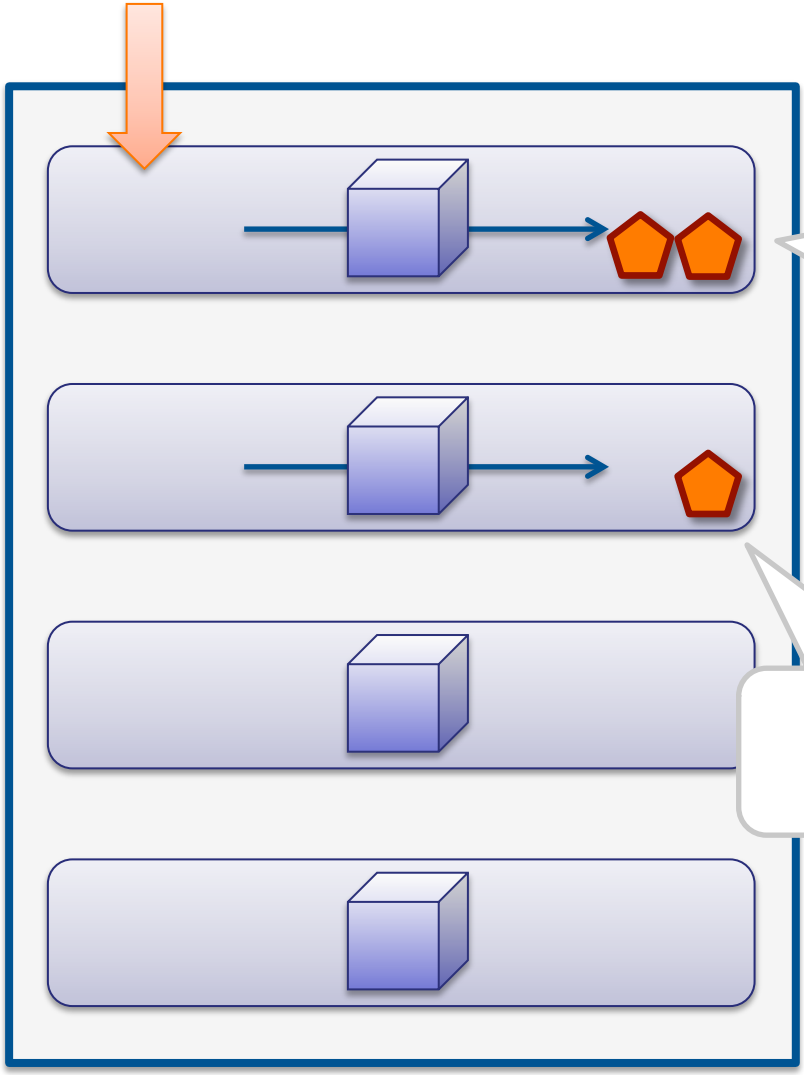
LUXOFT Training Center



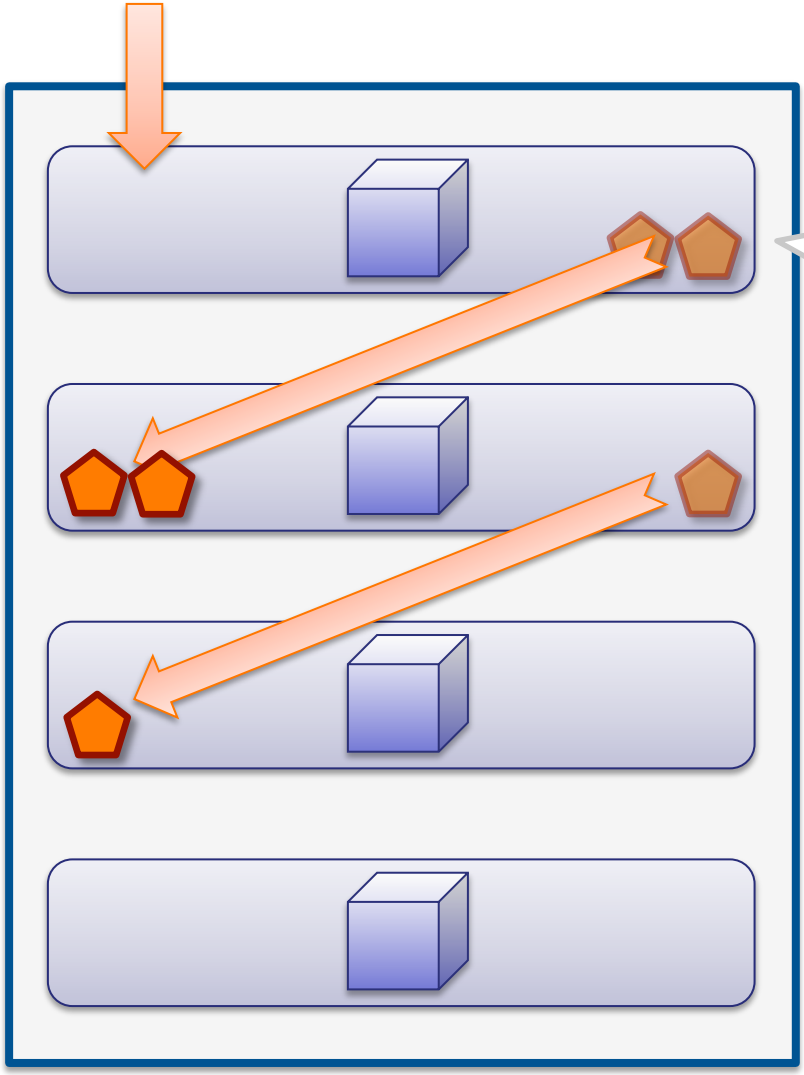
Конвейер



Конвейер

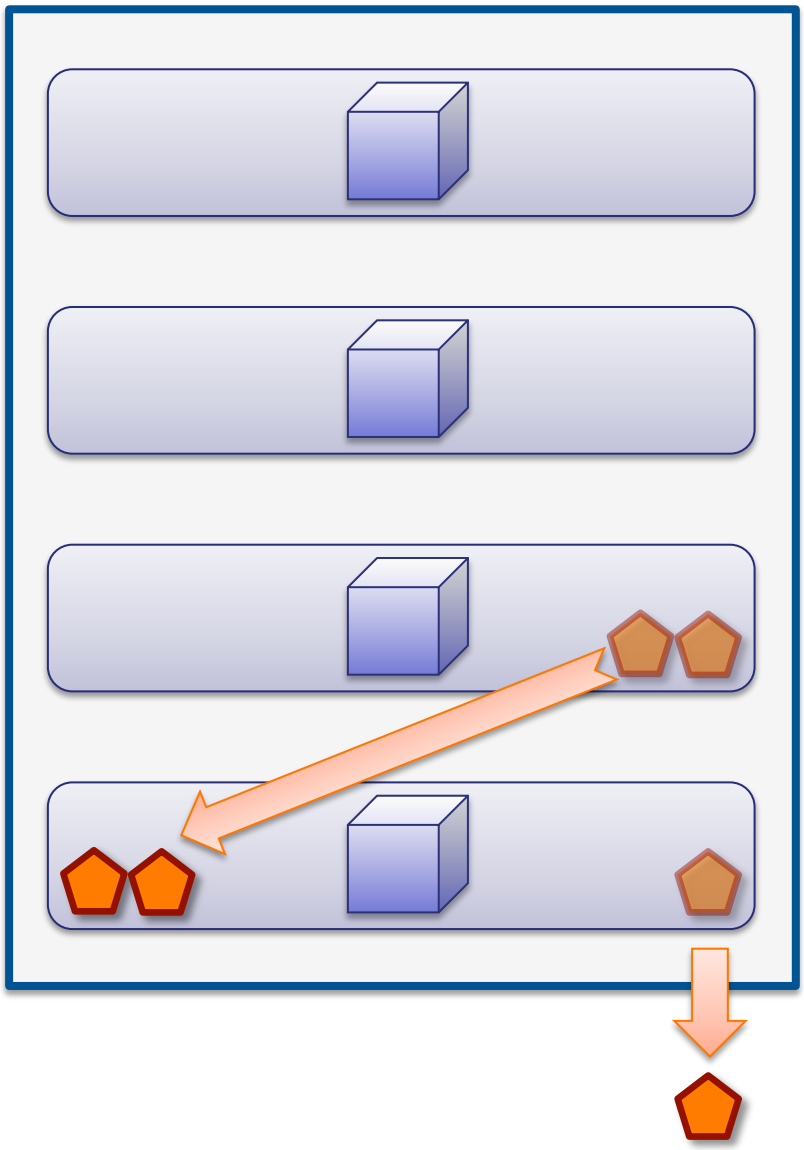


Конвейер

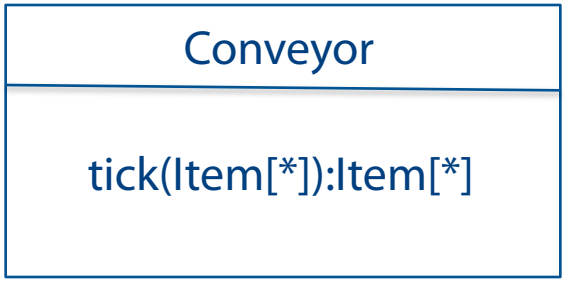


Передают детали дальше

Конвейер



Надо написать

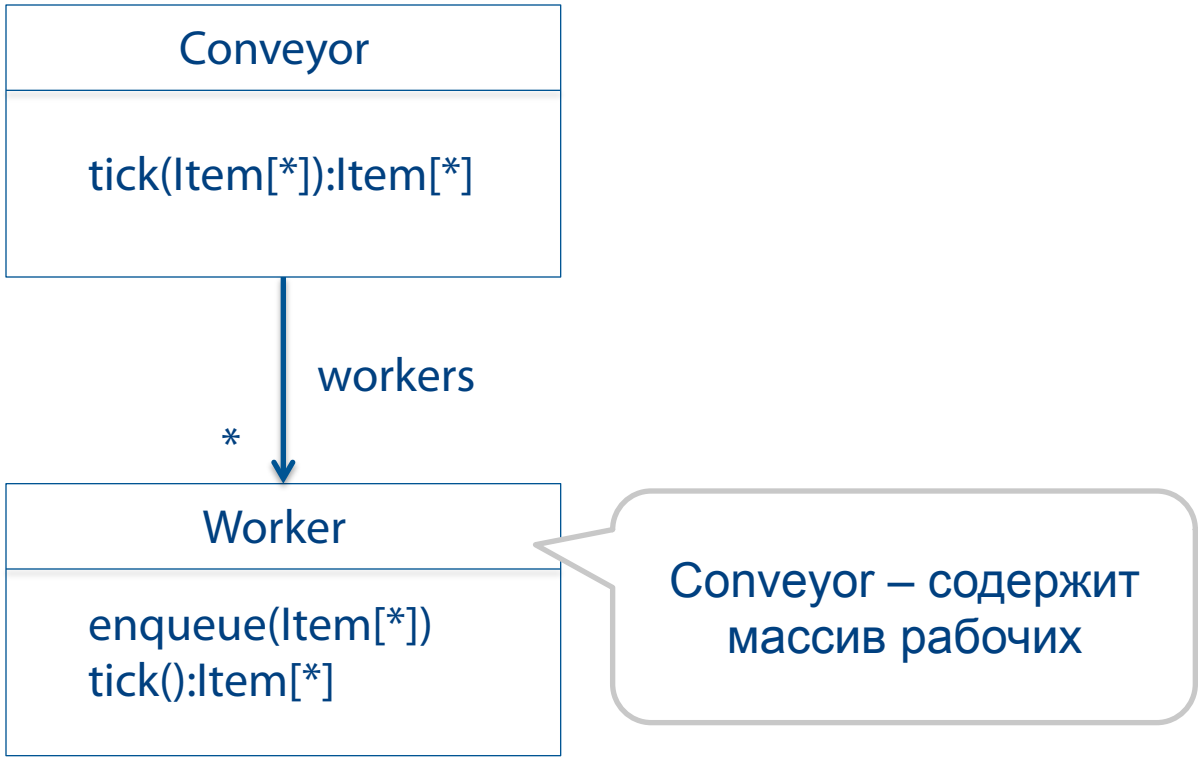


Написать класс Conveyor, который имеет один метод

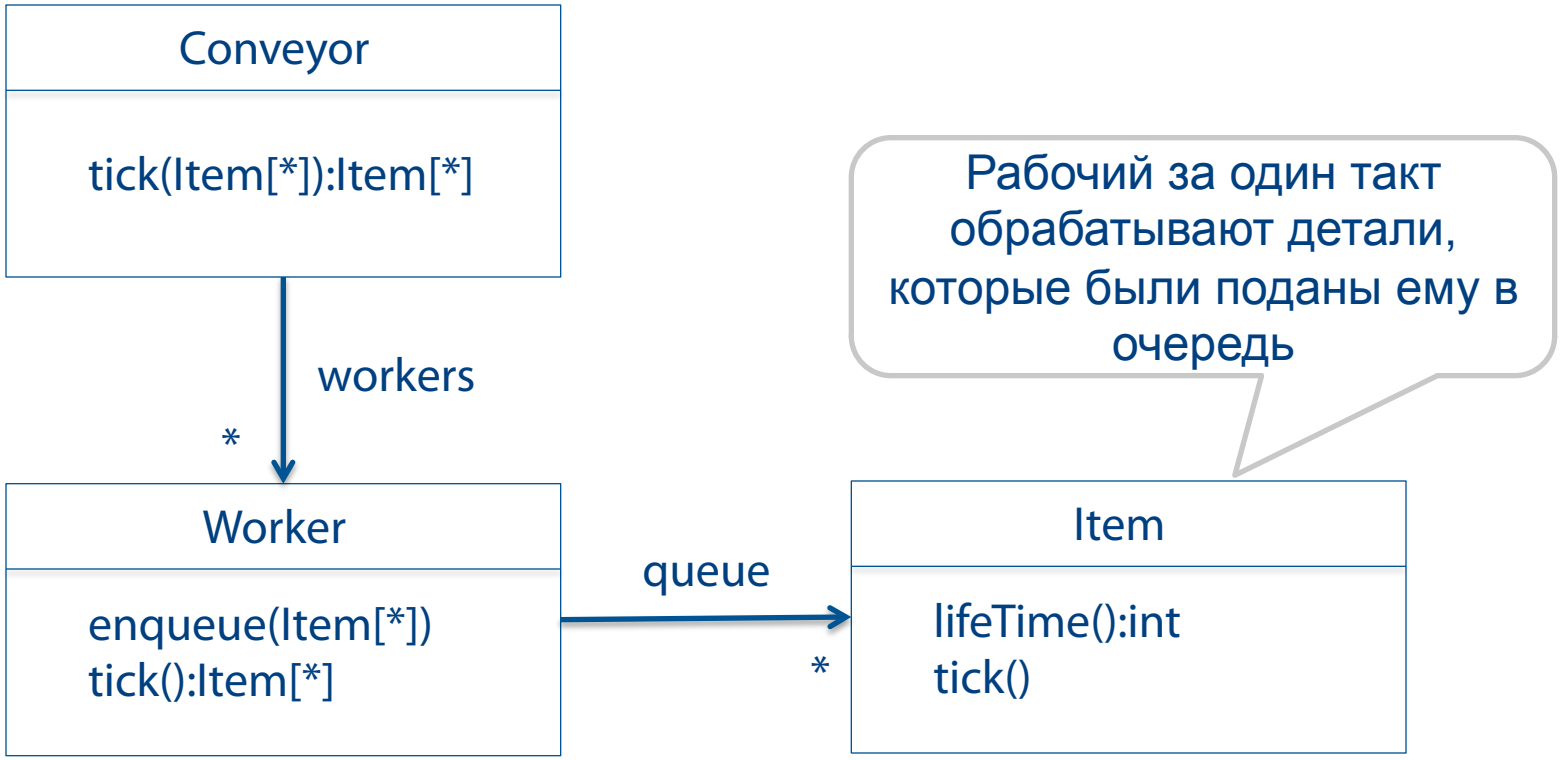
- tick(Item[*]):Item[*] – вызывается каждый такт. Параметром передается детали на входную очередь первого рабочего.
- Возвращает массив обработанных деталей - выход конвейера за соответствующий цикл

A quick design session

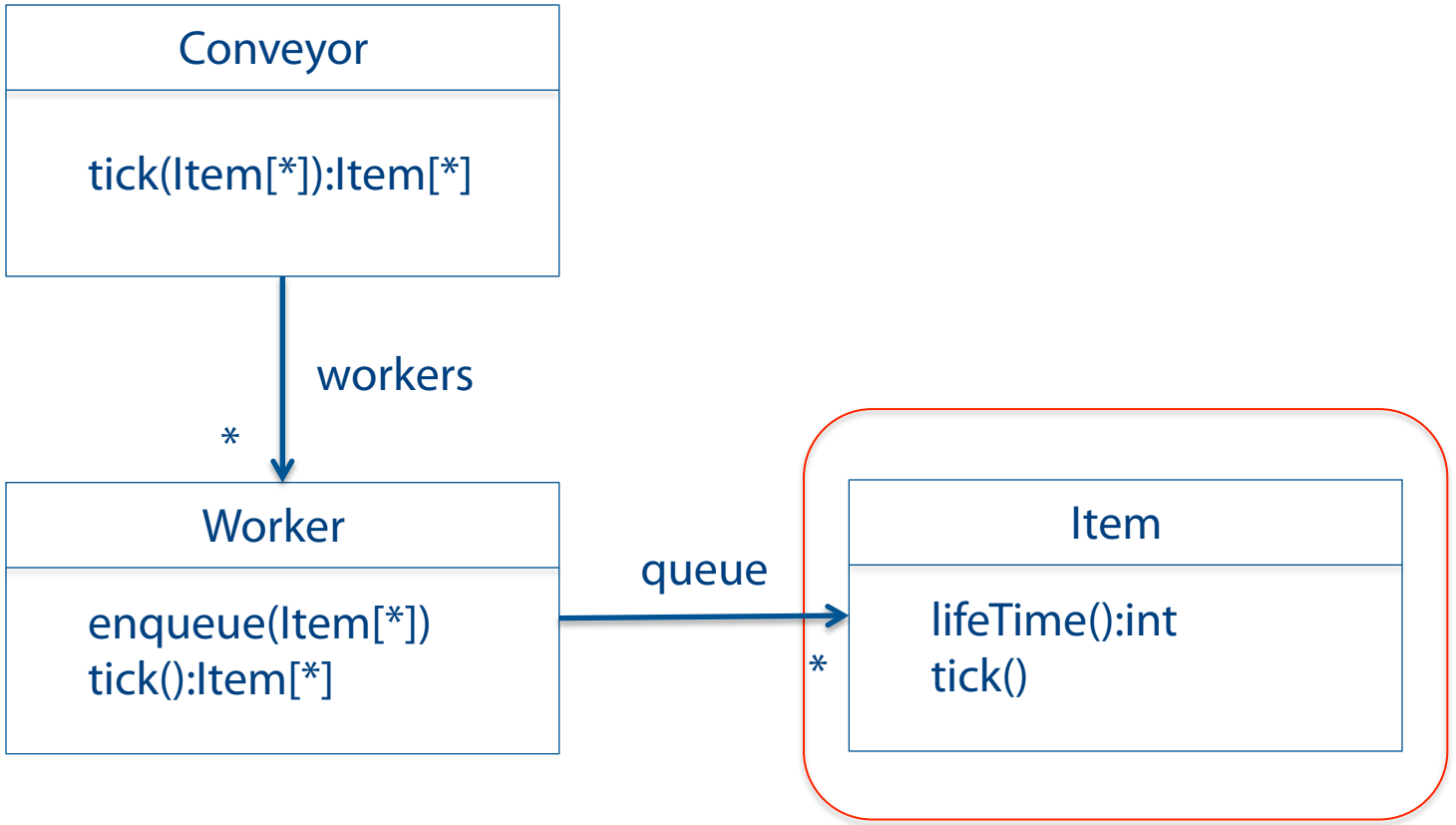
A quick design session



A quick design session



A quick design session



Item

LUXOFT Training Center

У вновь созданной детали

время жизни должно равняться нулю

- дано: новая деталь
- когда: запрашиваем у нее время жизни
- тогда: получаем в результате 0

Пишем тест

LUXOFT Training Center

```
package conveyor;

import org.junit.Test;
import static org.fest.assertions.Assertions.assertThat;

public class ItemTest {
    @Test
    public void shouldHaveZeroLifeTimeAfterCreation() {
        // given
        final Item item = new Item();
        // when
        int lifeTime = item.lifeTime();
        // then
        assertThat(lifeTime).isZero();
    }
}
```

FEST

Fixtures for Easy Software Testing

LUXOFT Training Center

```
int removed = employees.removeFired();
assertThat(removed).isZero();

List<Employee> newEmployees = employees.hired(TODAY);
assertThat(newEmployees).hasSize(6)
                        .contains(frodo, sam);

assertThat(yoda).isInstanceOf(Jedi.class)
                .isEqualTo(foundJedi)
                .isNotEqualTo(foundSith);
```

Fixtures for Easy Sofware Testing

FEST

Это «вырожденный» тест на состояние

Пишем минимум кода

LUXOFT Training Center

```
public class Item {
    public int lifeTime() {
        return 0;
    }
}
```

LUXOFT Training Center

Дано:

арбуз + гиря 1 кг = гиря 6 кг
арбуз = ?

Решение:

x + 1 = 6
x = 6 – 1 = 5

Ответ: 5 кг

Шаблон теста

```
Test...
{
    // Arrange
    ...
    // Action
    ...
    // Assetion
    ...
}
```

```
Should...
{
    // Given
    ...
    // When
    ...
    // Then
    ...
}
```

Критерий хорошо оформленного теста

LUXOFT Training Center

- Содержательное название
- Короткое тело (max = 20-30 строк)
- По шаблону AAA или GIVEN-WHEN-THEN
- Без циклов
- Без ветвлении (if-ов и case-ов)
- Должен легко читаться (literate programming)

Следующий тест

Оповещение о том, что прошел такт конвейера должно увеличивать значение времени жизни на один

- дано: новая деталь
- когда: оповещаем ее о такте конвейера
- тогда: время жизни становится 1

Пишем тест

LUXOFT Training Center

```
@Test
public void shouldIncrementLifeTimeDuringTick() {
    // given
    final Item item = new Item();
    // when
    item.tick();
    // then
    assertThat(item.lifeTime()).isEqualTo(1);
}
```

LUXOFT Training Center

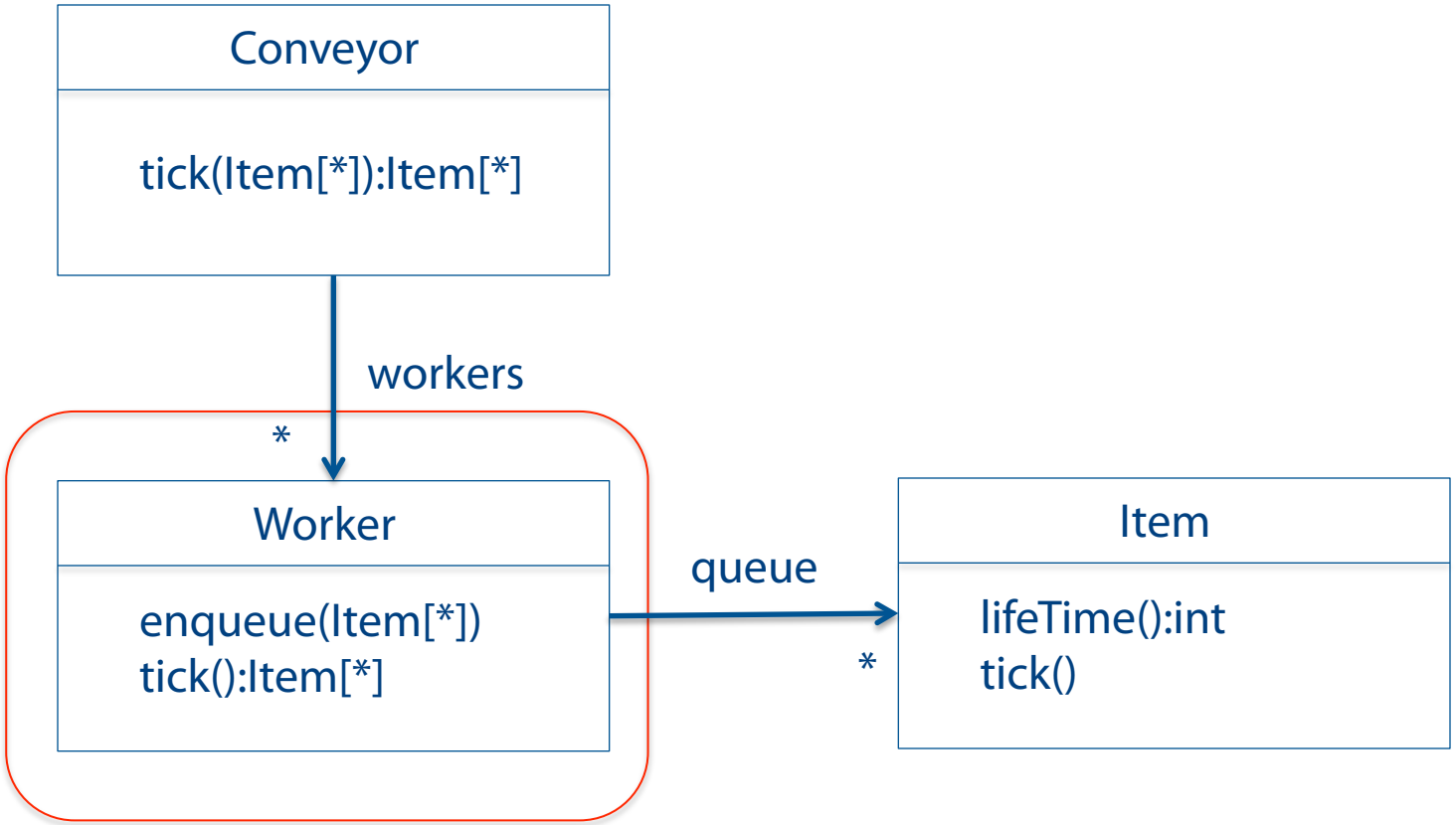
Это примитивный пример
теста на состояние

Пишем код

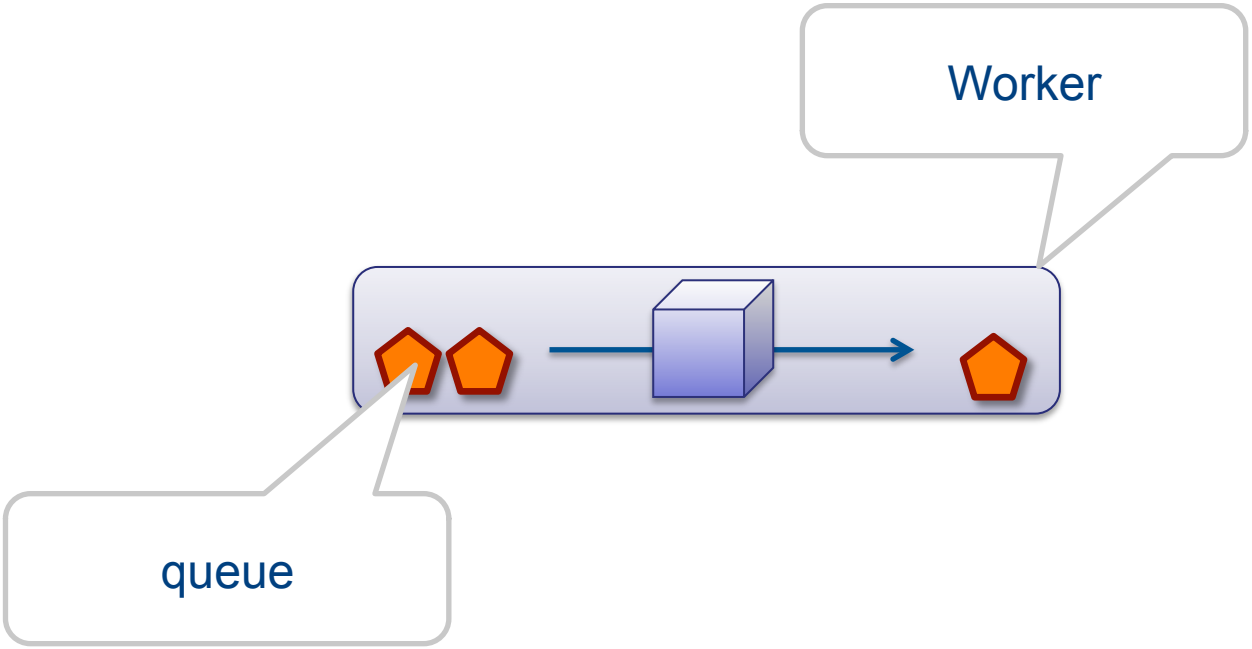
LUXOFT Training Center

```
public class Item {
    private int lifeTime;

    public int lifeTime() {
        return lifeTime;
    }

    public void tick() {
        lifeTime++;
    }
}
```

Переходим к Worker



Worker



Worker

LUXOFT Training Center

- Рабочий ничего не обрабатывает, если нет деталей
- У вновь созданного рабочего входная очередь деталеий пуста

Пишем тест

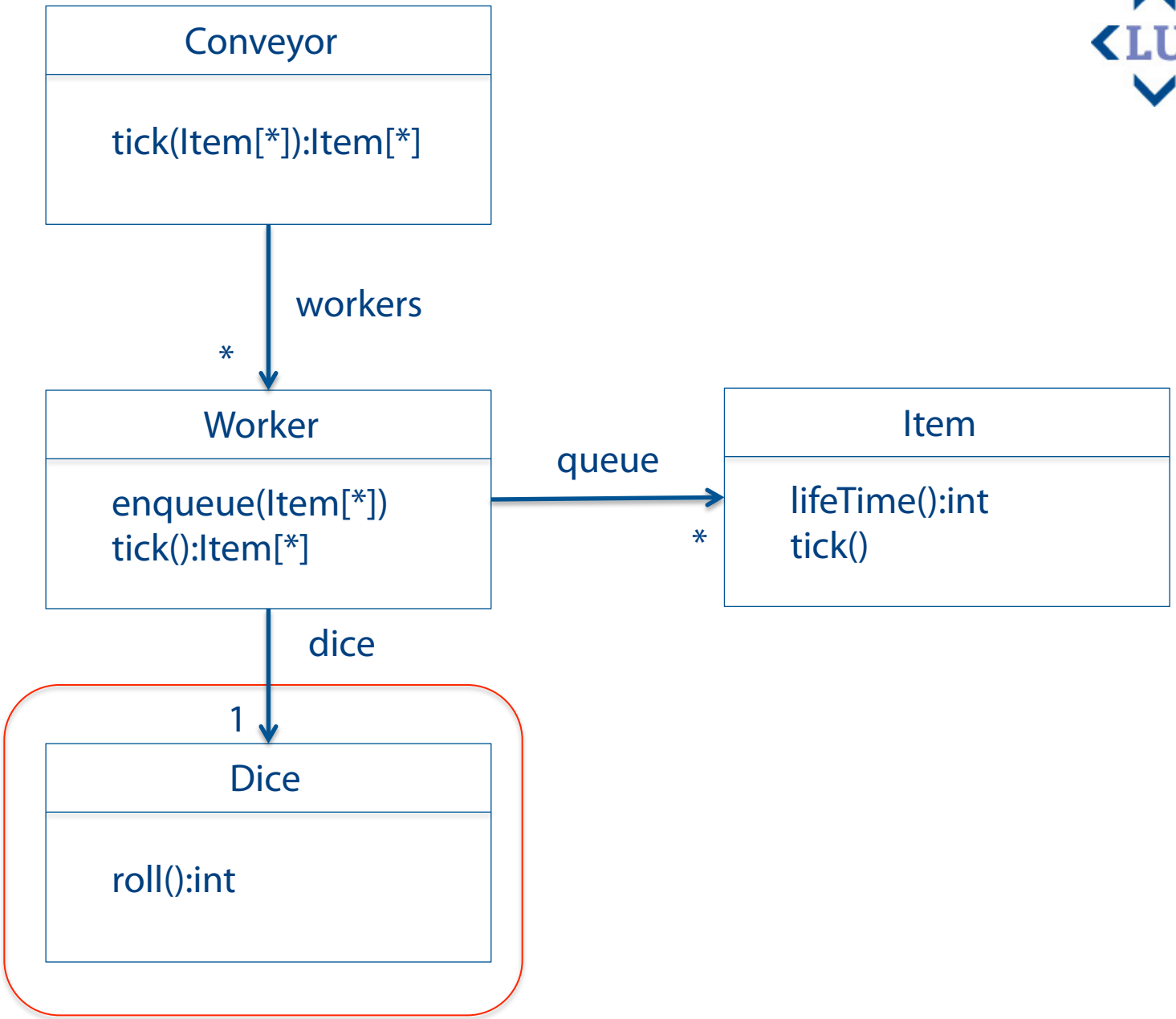
LUXOFT Training Center

```
public class WorkerTest {
    @Test
    public void shouldReturnNothingIfNothingToDo() {
        // given
        final Worker worker = new Worker();
        // when
        final List<Item> output = worker.tick();
        // then
        assertThat(output).isEmpty();
    }
}
```

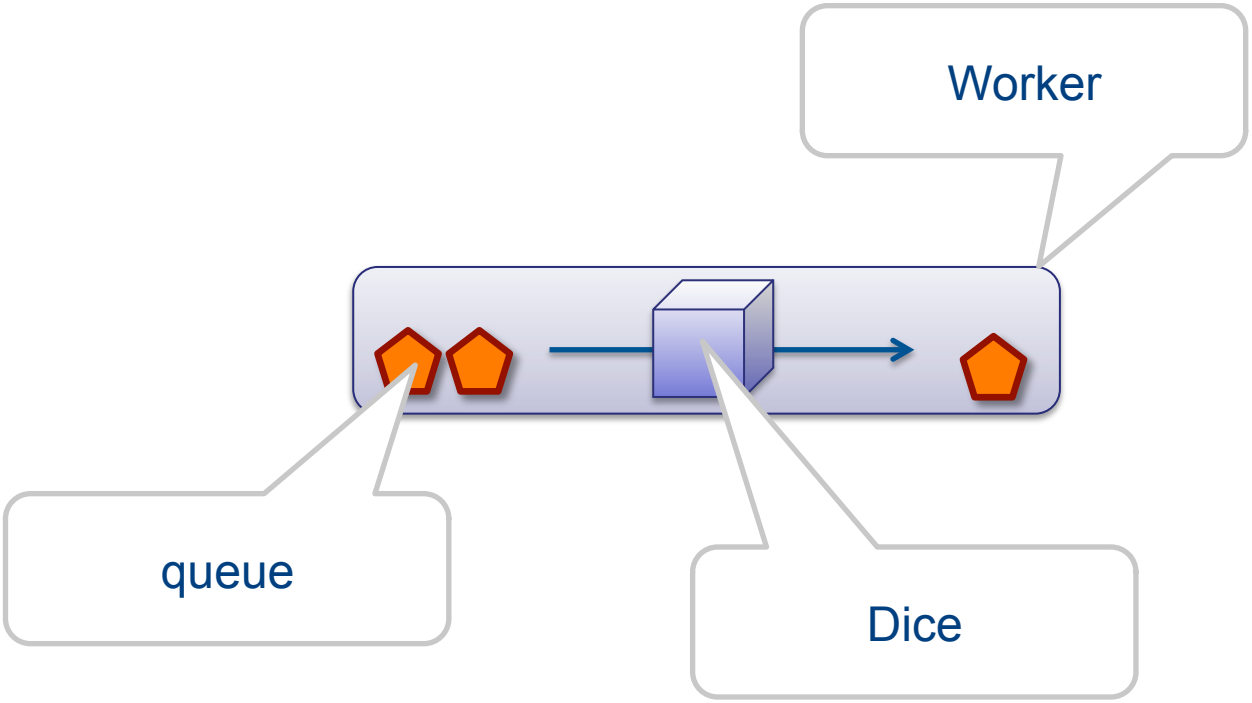
Worker

Если во время обработки на кубике выпало значение большее количества деталеий в очереди,

то рабочиий обрабатывает все детали в очереди (и больше ничего)



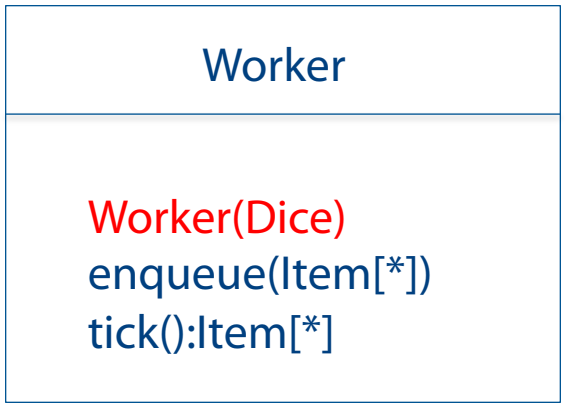
Worker



Dependency Injection (DI)

LUXOFT Training Center

Dependency Injection (DI) через конструктор

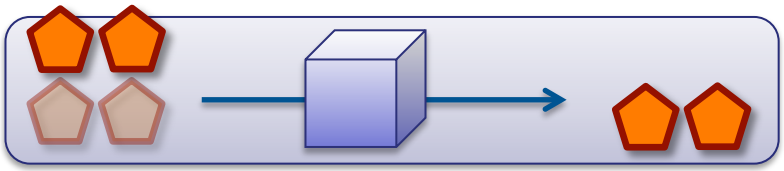


Пишем тест

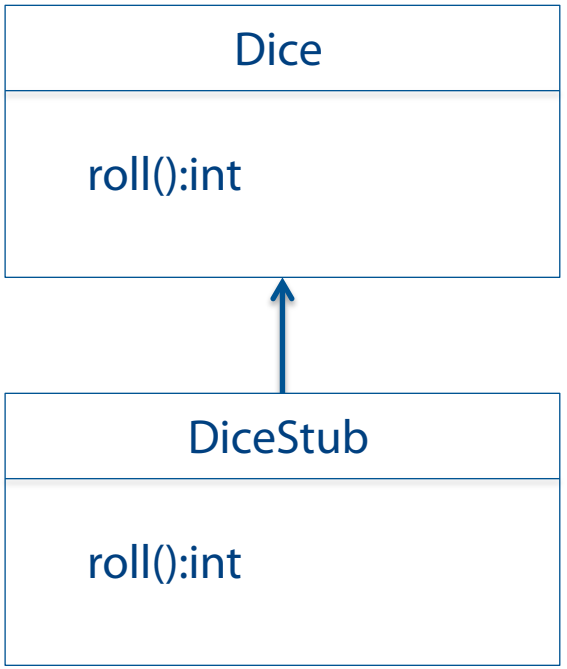
LUXOFT Training Center

```
@Test
public void shouldProcessNotGreaterThanItemsInQueue() {
    // given
    final Dice dice = createDiceStub(4);
    final Worker worker = new Worker(dice);
    final List<Item> items = Arrays.asList(
                                    new Item(),
                                    new Item());
    worker.enqueue(items);
    // when
    final List<Item> output = worker.tick();
    // then
    assertThat(output).isEqualTo(items);
}
```

Stub



enqueue tick



Mock

LUXOFT Training Center

```
private Dice createDiceStub(int rollValue) {
    final Dice dice = mock(Dice.class);
    when(dice.roll()).thenReturn(rollValue);
    return dice;
}
```

LUXOFT Training Center

Хотя мы и воспользовались mock-объектом,
это всё равно, по большому счету, тест на состояние

Worker

Если во время обработки на кубике выпало значение N, меньше количества деталей в очереди,
то обрабатывается только первые N деталей из очереди

Worker

LUXOFT Training Center

```
@Test
public void shouldProcessNotGreaterThanRolledValue() {
    // given
    final int rollValue = 3;
    final Dice dice = createDiceStub(rollValue);
    final Worker worker = new Worker(dice);
    final List<Item> items = Arrays.asList(
                    new Item(), new Item(), new Item(), new Item());
    worker.enqueue(items);
    // when
    final List<Item> output = worker.tick();
    // then
    assertThat(output).isEqualTo(items.subList(0, rollValue));
}
```

Еще аналогичные тесты:

LUXOFT Training Center

- Проверяем, что enqueue() **добавляет** в очередь
- Проверяем, что tick() **удаляет** из очереди обработанные детали

Тупой, но важный тест:

Во время Worker.tick() кубик бросается ровно один раз!

Кубик бросается один раз

LUXOFT Training Center

```
@Test
public void shouldRollDiceOnlyOnceDuringTick() {
    // given
    final Dice dice = createDiceStub(3);
    final Worker worker = new Worker(dice);
    final List<Item> items = Arrays.asList(
                                    new Item(), new Item(),
                                    new Item(), new Item());
    worker.enqueue(items);
    // when
    worker.tick();
    // then
    verify(dice, times(1)).roll();
}
```

Тест на поведение

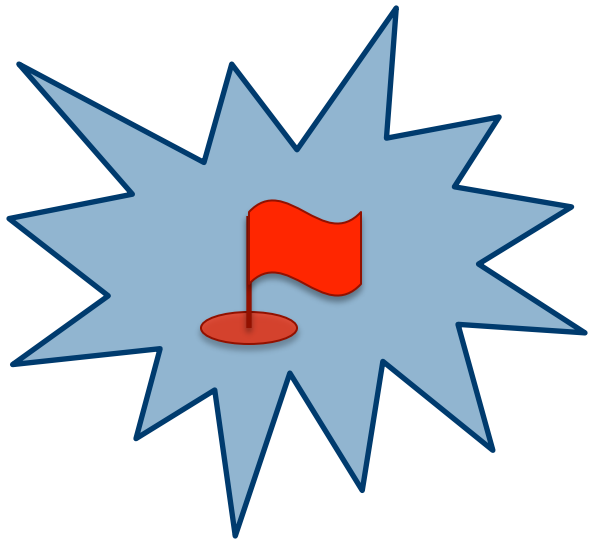
LUXOFT Training Center

Это примитивный пример **теста на поведение**:
мы проверили как взаимодействует наш объект с другим объектом.

Тесты

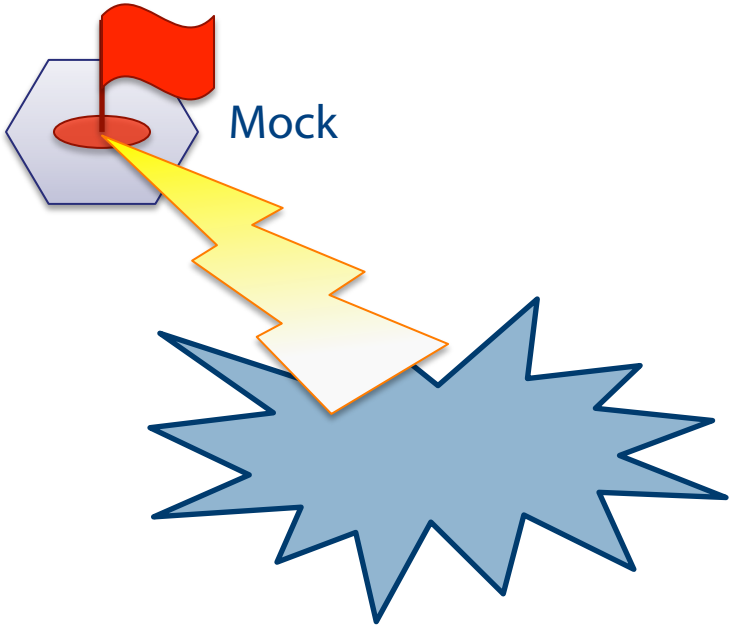
LUXOFT Training Center

На состояние



Object

На поведение



Object

Закрепим материал:

LUXOFT Training Center

- Проверим, что во время Worker.tick() вызывается Item.tick() для всех деталей, находящихся в очереди на начало tick()-а
- Это можно проверить, не прибегая к mock-ам – через значение lifeTime(), но тогда мы тестируем два класса сразу, а не один в изоляции

Закрепим материал:

LUXOFT Training Center

```
@Test
public void shouldCallTickForAllItemsInQueue() {
    // given
    final Dice dice = createDiceStub(2);
    final Worker worker = new Worker(dice);
    final Item firstItem = mock(Item.class);
    final Item secondItem = mock(Item.class);
    final List<Item> items = Arrays.asList(firstItem, secondItem);
    worker.enqueue(items);
    // when
    worker.tick();
    // then
    verify(firstItem, times(1)).tick();
    verify(secondItem, times(1)).tick();
}
```

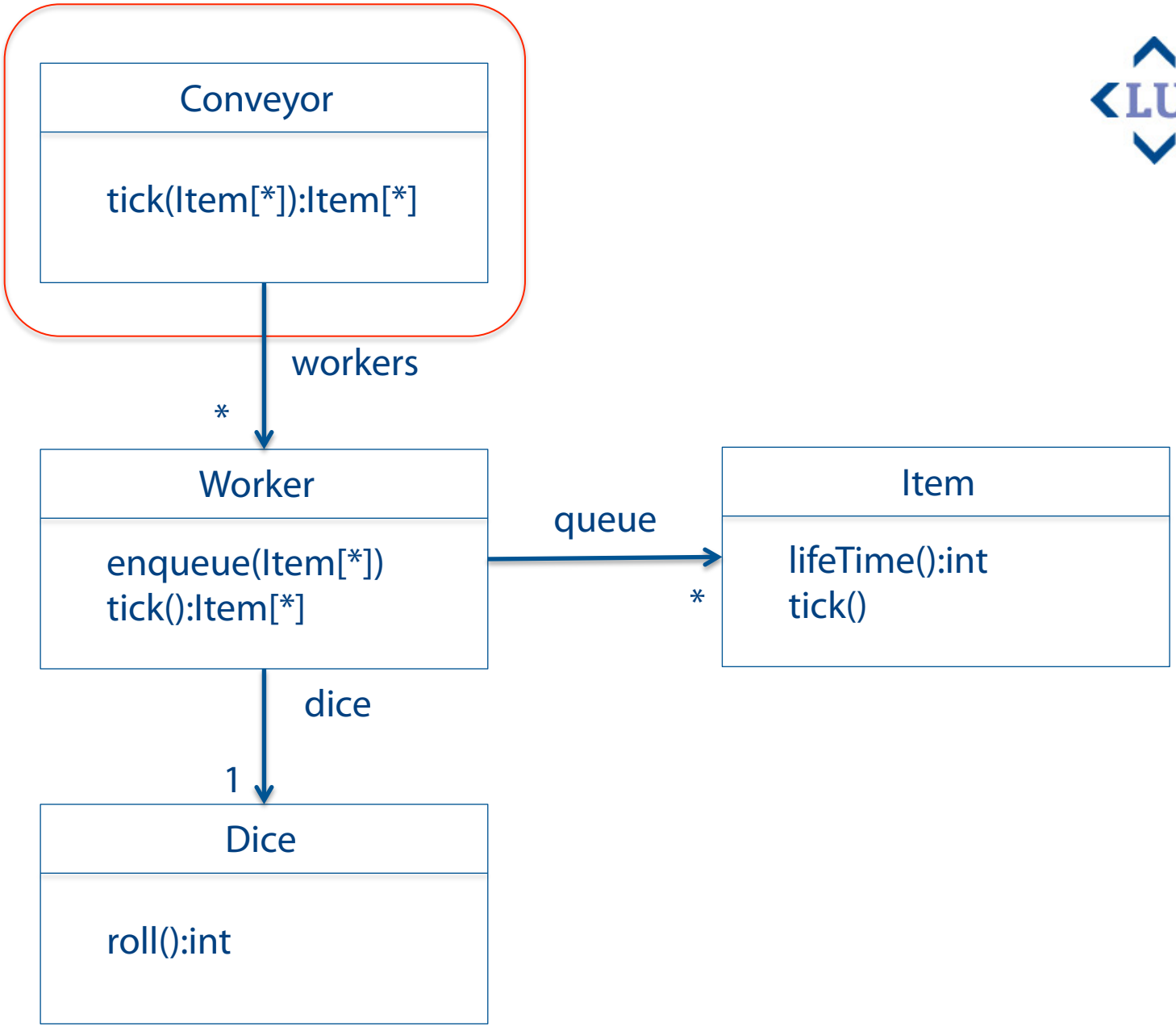
Совет «по случаю»

LUXOFT Training Center

- Избегайте имен переменных item1, item2 и т.п.
- Точно запутаетесь и опечатаетесь
- Лучше говорящие имена
- Или на худоий конец: firstItem, secondItem и т.п.

LUXOFT Training Center

Переходим к самому интересному



LUXOFT Training Center

LUXOFT Training Center

Как тестировать?

Тесты на состояние

LUXOFT Training Center

Как вариант

- Можно придумать несколько тестовых сценариев (разрисовать на бумажке)

Проблемы

- Но как они помогут написать реализацию?
- Какие тесты написать первыми, а какие потом?
- Как быть уверенным, что протестированы все случаи и нюансы? (полнота покрытия)
- Как эти тесты будут соотноситься со спецификацией? (test == executable specification)

Напомним спецификацию

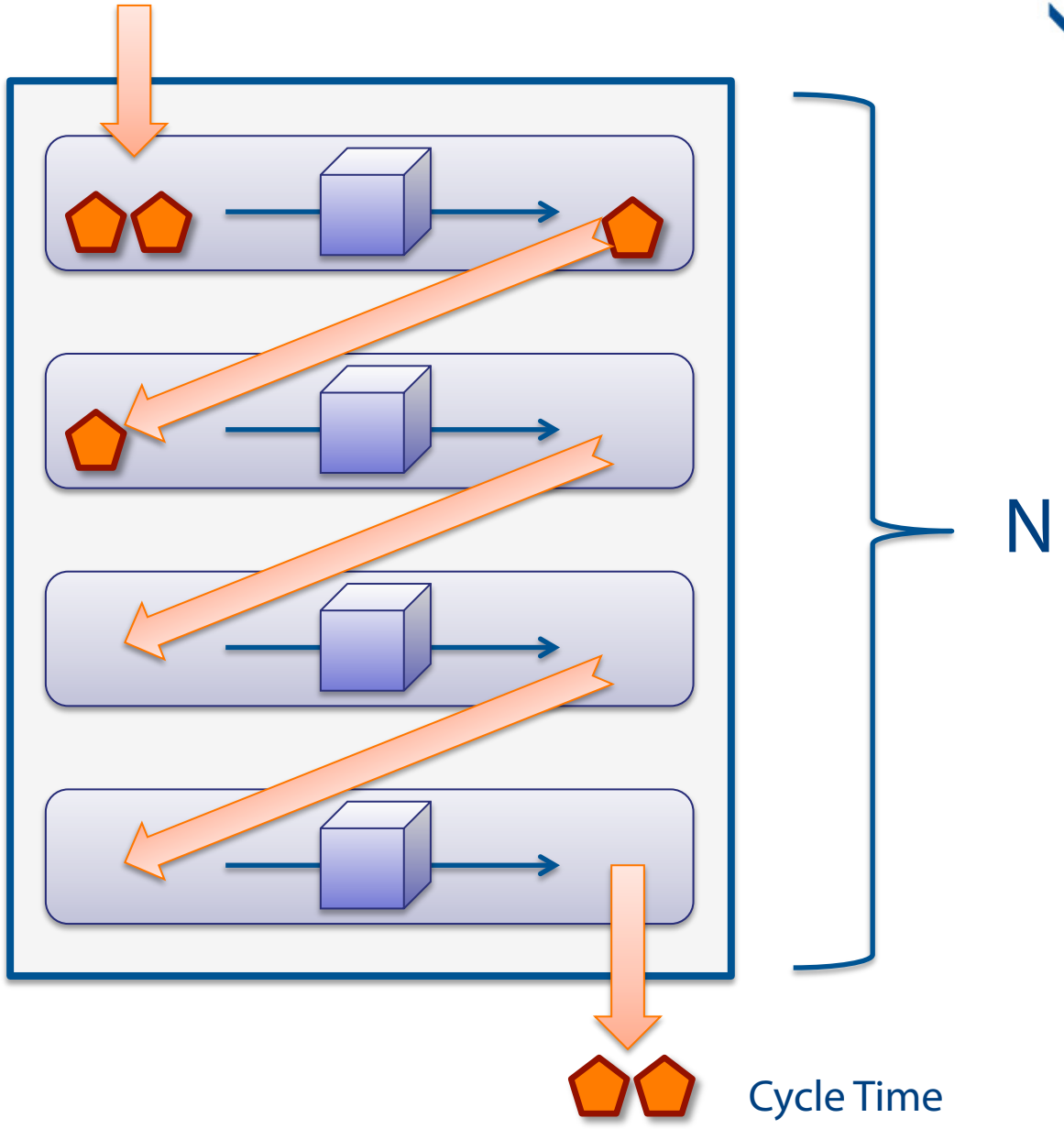
LUXOFT Training Center

1. То, что подается на вход конвейера, сражу же оказывается в очереди первого рабочего, т.е. до начала обработки им деталей

2. То, что обработал последний рабочий, является выходом конвейера за соответствующий цикл

3. Для всех остальных рабочих их результат работы попадает в очередь к следующему рабочему, но уже после того, как тот произвел обработку

Конвейер



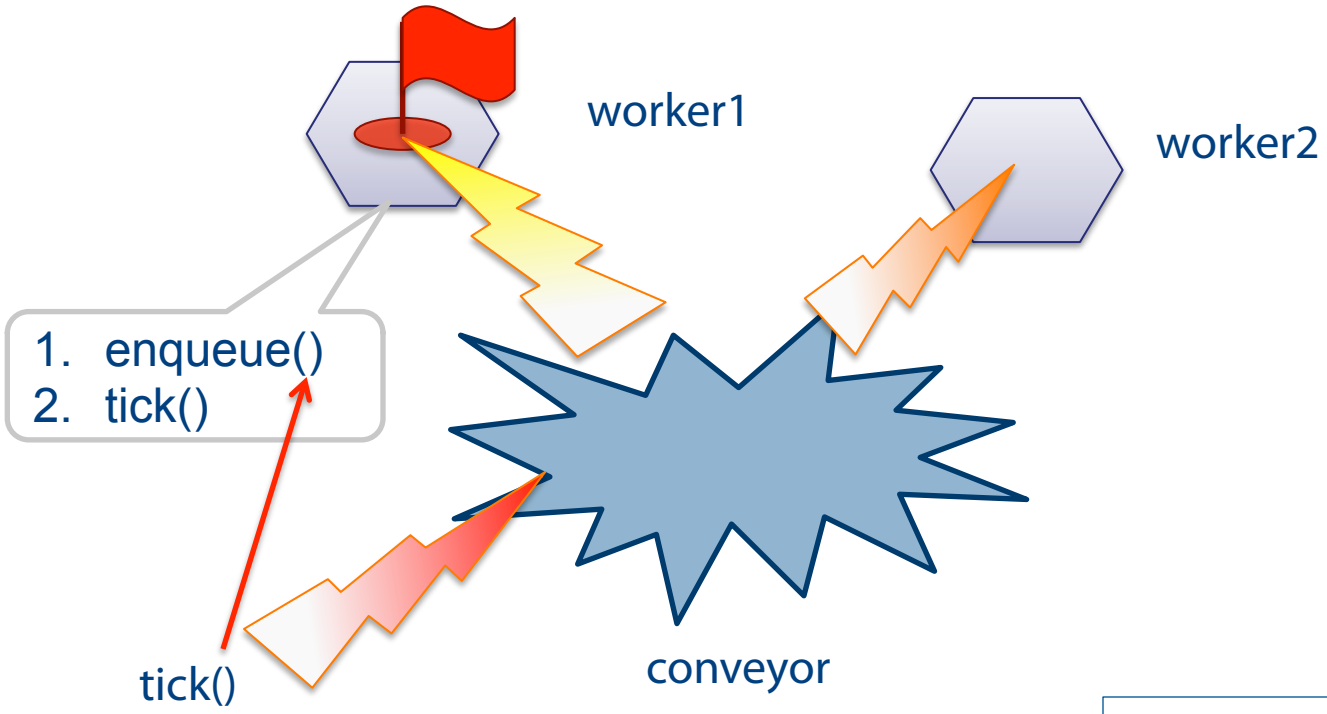
LUXOFT Training Center

Тесты на поведение позволяют протестировать эту спецификацию один в один

Спецификация

LUXOFT Training Center

1. То, что подается на вход конвейера, сражу же оказывается в очереди первого рабочего, т.е. до начала обработки им деталей
2. То, что обработал последний рабочий, является выходом конвейера за соответствующий цикл
3. Для всех остальных рабочих их результат работы попадает в очередь к следующему рабочему, но уже после того, как тот произвел обработку



Conveyor
Conveyor(Worker[*]) tick(Item[*]):Item[*]

Conveyor

LUXOFT Training Center

```
@Test
public void shouldEnqueueInputToFirstWorkerBeforeProcessing() {
    // given
    final List<Item> someInput = createItems(3);

    final Worker firstWorker = mock(Worker.class);
    final Worker secondWorker = mock(Worker.class);

    final List<Worker> workers = Arrays.asList(
                                        firstWorker, secondWorker);
    final Conveyor conveyor = new Conveyor(workers);

    // when
    conveyor.tick(someInput);

    // then
    final InOrder order = inOrder(firstWorker);
    order.verify(firstWorker, times(1)).enqueue(someInput);
    order.verify(firstWorker, times(1)).tick();
}
```

Спецификация

LUXOFT Training Center

1. То, что подается на вход конвейера, сражу же оказывается в очереди первого рабочего, т.е. до начала обработки им деталей
2. То, что обработал последний рабочий, является выходом конвейера за соответствующий цикл
3. Для всех остальных рабочих их результат работы попадает в очередь к следующему рабочему, но уже после того, как тот произвел обработку

Conveyor

LUXOFT Training Center

```
@Test
public void shouldReturnOutputOfLastWorker() {
    // given
    final List<Item> someOutput = createItems(2);

    final Worker firstWorker = mock(Worker.class);
    final Worker secondWorker = mock(Worker.class);
    when(secondWorker.tick()).thenReturn(someOutput);

    final List<Worker> workers = Arrays.asList(firstWorker, secondWorke
    final Conveyor conveyor = new Conveyor(workers);
    final List<Item> someInput = createItems(1);

    // when
    final List<Item> output = conveyor.tick(someInput);

    // then
    assertThat(output).isEqualTo(someOutput);
}
```

LUXOFT Training Center

Это можно было проверить, создав реальных Worker-ов,
«накормив» заранее второго нужными Item-ами,
но такие тесты уже больше похожи на интеграционные

LUXOFT Training Center

Mock-и очень удобны, чтобы имитировать любое необходимое состояние стороннего объекта, не связываясь с длинной цепочкой вызовов, необходимой для приведения реального объекта в это состояние

Mocks

LUXOFT Training Center

Fake

Mock

Stub, Dummy

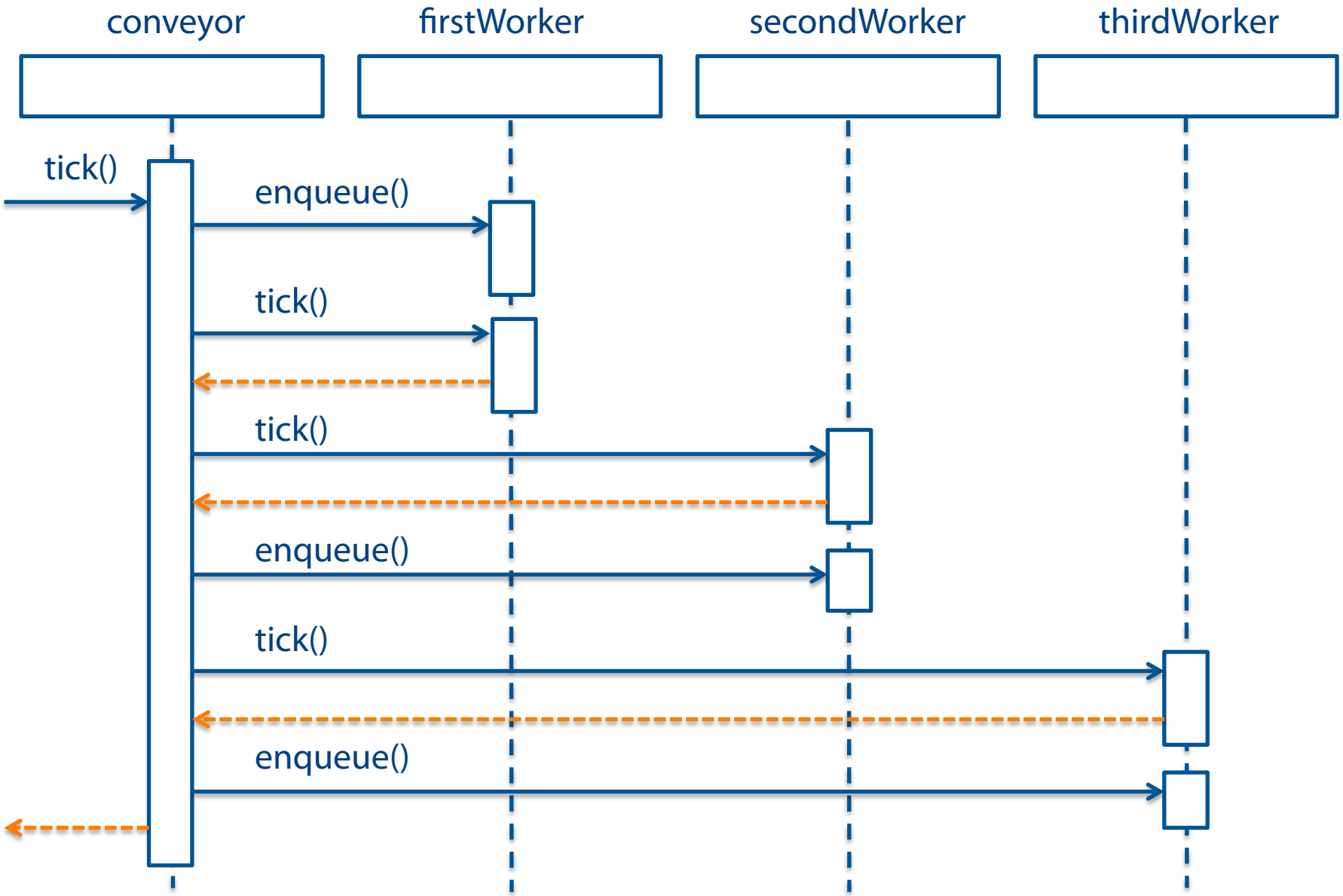
Спецификация

LUXOFT Training Center

1. То, что подается на вход конвейера, сражу же оказывается в очереди первого рабочего, т.е. до начала обработки им деталей

2. То, что обработал последний рабочий, является выходом конвейера за соответствующий цикл

3. Для всех остальных рабочих их результат работы попадает в очередь к следующему рабочему, но уже после того, как тот произвел обработку



```java
@Test
public void shouldEnqueueOutputOfPreviousWorkerToTheNextAfterProcessing() {
    // given
    final List<Item> outputOfFirstWorker = createItems(2);
    final List<Item> outputOfSecondWorker = createItems(3);

    final Worker firstWorker = mock(Worker.class);
    when(firstWorker.tick()).thenReturn(outputOfFirstWorker);

    final Worker secondWorker = mock(Worker.class);
    when(secondWorker.tick()).thenReturn(outputOfSecondWorker);

    final Worker thirdWorker = mock(Worker.class);

    final List<Worker> workers = Arrays.asList(
                                    firstWorker, secondWorker, thirdWorker);
    final Conveyor conveyor = new Conveyor(workers);

    final List<Item> someInput = Arrays.asList(new Item());

    // when
    conveyor.tick(someInput);

    // then
    InOrder secondWorkerOrder = inOrder(secondWorker);
    secondWorkerOrder.verify(secondWorker).tick();
    secondWorkerOrder.verify(secondWorker).enqueue(outputOfFirstWorker);

    InOrder thirdWorkerOrder = inOrder(thirdWorker);
    thirdWorkerOrder.verify(thirdWorker).tick();
    thirdWorkerOrder.verify(thirdWorker).enqueue(outputOfSecondWorker);
}
```

LUXOFT Training Center

Тест на поведение – это проверка, что код соответствует задуманной диаграмме последовательности

Реализация

LUXOFT Training Center

```
public List<Item> tick(final List<Item> input) {
    if (workers.isEmpty())
        return input;

    final Worker firstWorker = workers.get(0);
    firstWorker.enqueue(input);
    List<Item> output = firstWorker.tick();

    for (int i = 1; i < workers.size(); i++) {
        final Worker worker = workers.get(i);
        final List<Item> tmp = worker.tick();
        worker.enqueue(output);
        output = tmp;
    }

    return output;
}
```

Полный код

LUXOFT Training Center

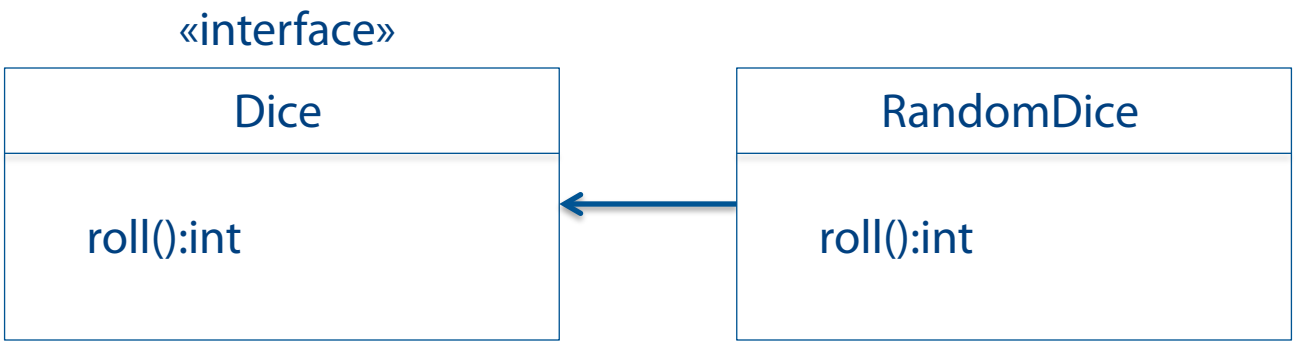
git clone https://github.com/ivan-dyachenko/Trainings.git

WARNING

LUXOFT Training Center

В коде не выделены интерфейсы для Item, Dice и Worker только в целях «упрощения» примера.

Выделение интерфейсов предпочтительнее перекрытия самих классов с функциональностью.



Плюсы тестов на поведение

LUXOFT Training Center

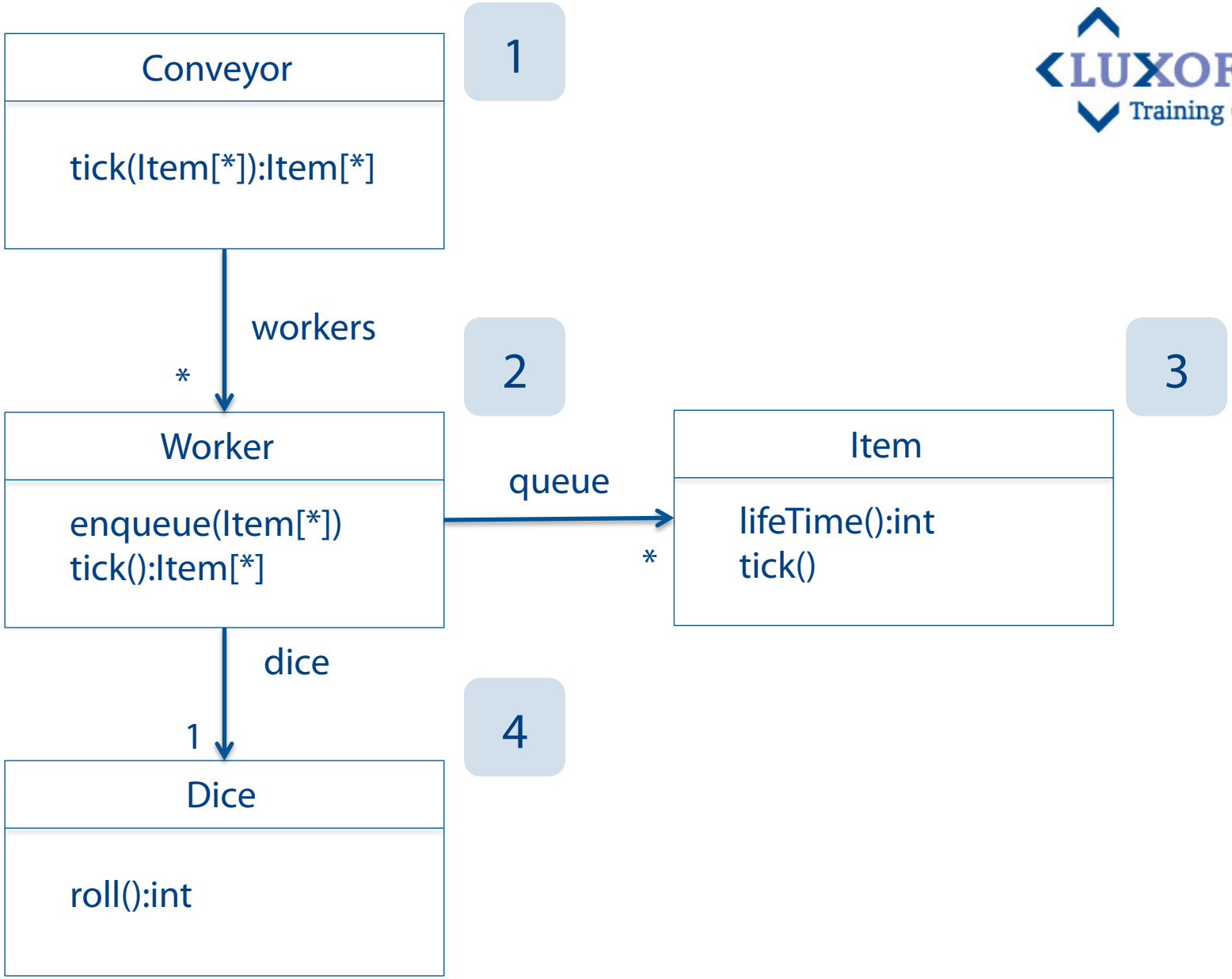
1. Просто писать (когда привыкнешь)
   - не нужно долго и мучительно приводить окружение в нужное состояние

2. Являются истинными unit-тестами
   - проверяют функционал класса в изоляции от всех остальных

3. Хорошо отражают спецификации и дают уверенность в хорошем покрытии кода
   - executable specification

4. Принуждают к модульному дизайну
   - SRP, LSP, DIP, ISP

5. Позволяют разрабатывать функционал сверху-вниз от сценариев использования
   - а не снизу вверх от данных



Минусы тестов на поведение

LUXOFT Training Center

1. Чтобы ими овладеть, требуется ментальный сдвиг
2. Проверяют, что код работает так, как вы ожидаете, но это не значит, что он работает правильно
   - этот недостаток легко снимается небольшим количеством интеграционных тестов
3. Не весь функционал можно так протестировать
4. Тесты хрупкие
   - изменение в реализации ломает тесты
5. Требуют выделения интерфейсов или виртуальности методов
   - Обычно не проблема. А если проблема, то используйте «быстрые mock-и» на C++ templates

Mock Hell

Чтобы его избежать, очень важно соблюдать ISP
Широко используемыми могут быть только стабильные интерфейсы

LUXOFT Training Center

~~Mockist vs Classicist~~

Mockist + Classicist

Вопросы ?

Разработка через тестирование

IDyachenko@luxoft.com

git clone git://github.com/ivan-dyachenko/Trainings.git

https://github.com/ivan-dyachenko/Trainings