GET FAMILIAR

10 MIN

- Start the launchscript
  - eclipse: TanteEmmas.launch
  - intellij: tante_emmas.xml -> .idea/runConfigurations
  - other:

```
java -cp [classpath] net.amygdalum.tanteemmas.server.Server
```

- Browse to http://localhost:8080
- Navigate through the application
- Explore the source code

AND NOW SOME HINTS:

- Stop the previous server (Vert.x) before starting a new server (you will possibly not recognize that you run on an old server session)
- Save all tests you want to keep. Each recorder session will replace existing files
- Delete all tests before starting a new server. Otherwise you cannot distinguish old recorded and new recorded files
- Read the javadocs for the used annotations. They will give you further hints on how to accomplish succesful recordings
- Do not rely on few tests, about 100 tests will produce an acceptable coverage

INSTALL TESTRECORDER

10 MIN

- Put testrecorder-0.3.12-jar-with-dependencies.jar in your workspace
- Put `AgentConfig.java` (located in src/hints) in the package `net.amygdalum.tanteemmas.testrecorder` in your src/main/java folder
- Try to understand the configuration in `AgentConfig.java`

- Modify the launch script by adding

```
-javaagent:testrecorder-0.3.12-jar-with-dependencies.jar=net.amygdalum.tanteemmas.testrecorder.AgentConfig
```

- eclipse: run configurations -> TanteEmmas -> Arguments -> Vm Arguments
- intellij: run -> edit configurations -> tante-emmas -> Vm Options
- shell:

```
java
  -cp [classpath]
  -javaagent:testrecorder-0.3.12-jar-with-dependencies.jar=net.amygdalum.tanteemmas.testrecorder.AgentConfig
  net.amygdalum.tanteemmas.server.Server
```

- Start the launch script
  - if the message `loading AgentConfig` appears => everything ok
  - else if the message `loading default config` appears => agent has been loaded but configuration not found (maybe AgentConfig is not in the correct package or the run script references a different config)
  - else => agent has not been loaded => adjust the path to the testrecorder.jar
- Stop the application

RECORD SIMPLE METHODS

15 MIN

- Annotate `PriceCalculator.computeFairPrice` with `@Recorded`
- Start the launch script, the messages should be:

```
loading AgentConfig`
recording snapshots of ...
```

- Browse to http://localhost:8080
- Navigate through the application
- Watch the console for generated tests
- Stop the server and examine the generated tests in the workspace target folder

- Try out different customer names such as:
  - Michaela Mustermann
  - Otto Normalverbraucher
  - Armer Schlucker
  - Reicher Schnösel
  - Reicher Pinkel
  - Gerd Grosskunde
- Analyze the generated tests for the customers and see how testrecorder analysis tries to generate readable tests

- If you are finished
    - You may even add a customer in `CustomerRepo` to see how testrecorder generates tests for it
    - Feel free to add some new properties and explore how flexible testrecorder will adjust code generation

GLOBAL DEPENDENCIES

10 MIN

- Annotate `PriceCalculator.applyUnfairCharges` with `@Recorded`
- Start the launch script, the messages should be:

```
loading AgentConfig
recording snapshots of ...
```

- Browse to http://localhost:8080
- Navigate through the application
- Watch the console for generated tests
- Stop the server and examine the generated tests in the workspace target folder

- Unfortunately - the tests fail
- But the productive code did work properly
- Analyze the test and the source code, what did we miss to record?

- The missing part is global state stored in static variables
- There are two ways to give testrecorder a hint which global state should be recorded
  - Annotate the static variable with `@Global`
  - Or adjust the `AgentConfig`s method `getGlobals`
- So find the static variable and tag it as global

- Start the launch script and browse to http://localhost:8080
- While navigating through the application watch the generated tests
- The tests should work

INPUT DEPENDENCIES

15 MIN

- Annotate `PriceCalculator.computePrice` with `@Recorded`
- Start the launch script, the messages should be:

```
loading AgentConfig
recording snapshots of ...
```

- Browse to http://localhost:8080
- Navigate through the application
- Use fast motion and generate at least 100 tests
- Examine the tests

- Unfortunately - some tests will fail
- Analyze the test and the source code, what did we miss to record?

- The missing part is input
- As you learned testrecorder stores the state before and after execution of the recorded method
- Sometimes a method does not only depend on state in the JVM but state originating from external systems, e.g.
  - time
  - random
  - reading files (that could have been edited)
  - reading from web services
  - ...

- Since input sources are external to the JVM the only option to include input is to mock it
- There are two ways to give testrecorder a hint which methods provide input and have to be mocked
  - Annotate the input methods with `@Input`
  - Or adjust the `AgentConfig`s method `getInputs`
  - Note that each argument and each result of a tagged method is recorded and replayed later on
- So find the input method and tag it

- Start the launch script and browse to http://localhost:8080
- While navigating through the application watch the generated tests
- The tests should work, if not - maybe there is more than one input

OUTPUT DEPENDENCIES

15 MIN

- Annotate `PriceCalculator.order` with `@Recorded`
- Start the launch script, the messages should be:

```
loading AgentConfig
recording snapshots of ...
```

- Browse to http://localhost:8080
- Navigate through the application and order some products
- Examine the generated tests

- The tests are green
- Change the timestamp argument in some of your tests

- Astonishingly the tests stay green
- You see: This green state is treacherous
- Analyze what the recorded method does and analyze which aspects of the method were not recorded

- The missing part is output
- As you learned testrecorder captures the state before and after execution of the recorded method
- Sometimes a method writes state to external systems, e.g.
  - writing files (that could have been edited)
  - requesting user input
  - notifying web services
  - ...

- Since output sinks are external to the JVM the only option to include output is to mock it and to verify the outputs
- There are two ways to give testrecorder a hint which methods consume output and have to be mocked
  - Annotate the output methods with `@Output`
  - Or adjust the `AgentConfig`s method `getOutputs`
  - Note that each argument of a tagged method is recorded and verified later on
- So find the output method and tag it

- Start the launch script and browse to http://localhost:8080
- While navigating through the application watch the generated tests
- The tests should work
- Now again change the timestamp argument of the tested method
- The tests should turn red

FIX BUG

15 MIN

- maybe you have already found some nasty parts of the code
    - the customer/tester feels that a single rainy day increases the average price of all products for all time after
    - a tester mentioned that some event seems to modify the properties of all products
- find the variable that represents the product
- find changes to the product that correspond to the customers/testers experiences

- now we fix this bug test-driven
    - correct the tests that expect a behavior that is not expected by the customer/tester
    - these tests should turn red
    - then fix the code
    - these tests should turn green

- for sure - execute the other tests in your generated suite
  - if you worked truely diligent (or your test suite was quite small) all tests are green
  - otherwise assume that you did work rather effective than diligent - but try to explain why other test do fail now