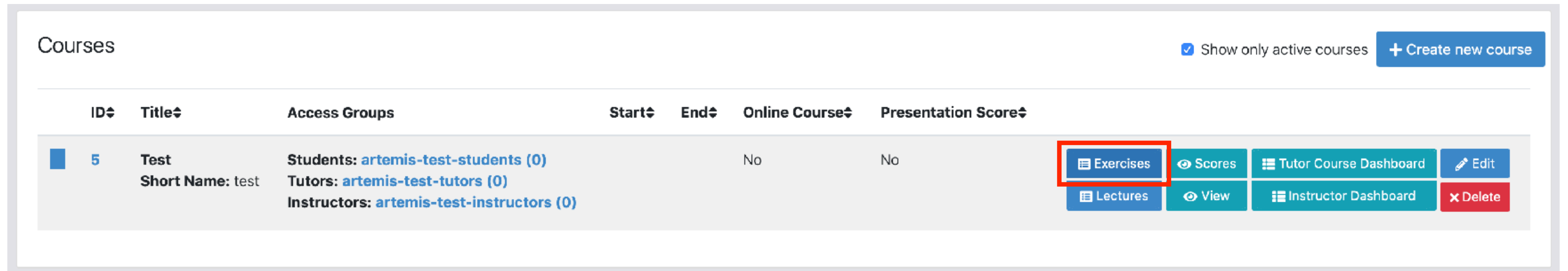


Artemis

Tutorial: Create Programming Exercises

1. Open Course Management

- <https://artemis.ase.in.tum.de/#/course-management>
- Navigate into **Exercises** of your preferred course



The screenshot displays the 'Courses' management interface. At the top right, there is a checkbox for 'Show only active courses' and a '+ Create new course' button. Below this is a table with columns for 'ID', 'Title', 'Access Groups', 'Start', 'End', 'Online Course', and 'Presentation Score'. A single course is listed with ID '5' and title 'Test'. The 'Access Groups' column shows 'Students: artemis-test-students (0)', 'Tutors: artemis-test-tutors (0)', and 'Instructors: artemis-test-instructors (0)'. The 'Online Course' and 'Presentation Score' columns both contain 'No'. To the right of the course row is a grid of action buttons: 'Exercises' (highlighted with a red box), 'Scores', 'Tutor Course Dashboard', 'Edit', 'Lectures', 'View', 'Instructor Dashboard', and 'Delete'.

ID	Title	Access Groups	Start	End	Online Course	Presentation Score	
5	Test Short Name: test	Students: artemis-test-students (0) Tutors: artemis-test-tutors (0) Instructors: artemis-test-instructors (0)			No	No	Exercises Scores Tutor Course Dashboard Edit Lectures View Instructor Dashboard Delete

2. Generate programming exercise

- Click on **Generate new programming exercise**

Test - 0 Exercises

Programming Exercises ▼

No Programming Exercises

[+ Generate new Programming Exercise](#) [+ Import new Programming Exercise](#)

2. Generate programming exercise

- Fill out all required values and click on **Generate**

Generate new Programming Exercise

Title [?]
Adapter Patter

Short Name [?]
adapter

Preview [?]

Repositories

- testadapter-exercise [?]
- testadapter-solution [?]
- testadapter-tests [?]

Build Plans

- TESTADAPTER-BASE [?]
- TESTADAPTER-SOLUTION [?]

Categories [?]
Enter a new category

Difficulty

No Level **Easy** Medium Hard

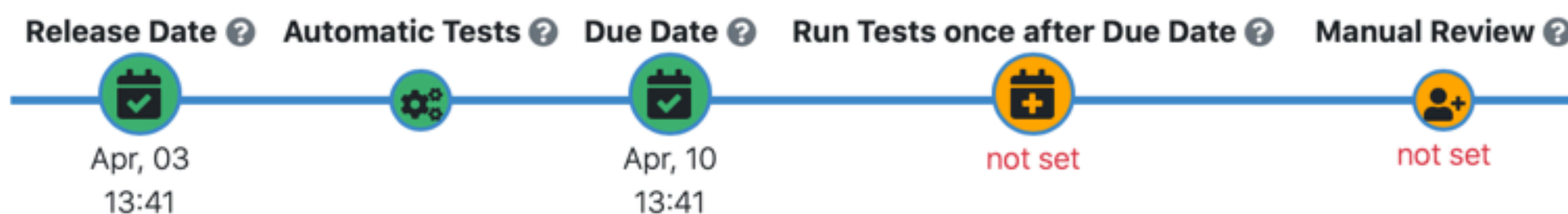
Mode [?]

Individual **Team**

Programming Language
Java

Package Name
de.tum.in.ase

Timeline of the whole programming exercise [?]



Max Score

10

Problem Statement

Edit Preview

B *I* U “ ” </> Style ▾ Color

Formula [task] Task Insert Test Case ▾ Add task specific hint ▾

```
1 - # Sorting with the Strategy Pattern
2
3 In this exercise, we want to implement sorting algorithms and choose them based on runtime specific v
4
5 - ### Part 1: Sorting
6
7 First, we need to implement two sorting algorithms, in this case `MergeSort` and `BubbleSort`.
8
9 **You have the following tasks:**
10
11 1. [task][Implement Bubble Sort](testBubbleSort)
12 Implement the method `performSort(List<Date>)` in the class `BubbleSort`. Make sure to follow the Bub
13
14 2. [task][Implement Merge Sort](testMergeSort)
15 Implement the method `performSort(List<Date>)` in the class `MergeSort`. Make sure to follow the Merg
16
17 - ### Part 2: Strategy Pattern
18
19 We want the application to apply different algorithms for sorting a `List` of `Date` objects.
20 Use the strategy pattern to select the right sorting algorithm at runtime.
21
22 **You have the following tasks:**
23
```

- Sequential Test Runs [?]
- Allow Online Editor
- Publish Build Plan

Cancel **Generate**

Result: Programming Exercise

1 Programming Exercises + Generate new Programming Exercise + Import new Programming Exercise

ID ↕	Title ↕	Short Name ↕	Release Date ↕	Due Date ↕	Max Score ↕	Repositories	Build Plans	Publish Build Plan ↕	Allow Online Editor ↕	
5	Adapter Patter	adapter	Apr 3, 2020, 1:41:45 PM	Apr 10, 2020, 1:41:44 PM	10	Template Solution Test	Template Solution	false	true	Scores Edit in editor View Reset Participations Manage Test Cases Edit Delete

- 3 repositories
 - **Template:** template code, can be empty, all students receive this code at the beginning of the exercises
 - **Test:** contains all test cases, e.g. based on JUnit, hidden for students
 - **Solution:** solution code, typically hidden for students, can be made available after the exercise
- 2 build plans
 - **Template:** also called BASE, basic configuration for the test + template repository, used to create student build plans
 - **Solution:** also called SOLUTION, configuration for the test + solution repository, used to manage test cases and to verify the exercise configuration

Result: Programming Exercise

Programming Exercise 5

[* Add External Submission](#) [Manage Hints](#) [Download Repos](#) [Cleanup](#)

Title

Adapter Patter

Short Name

adapter

Mode

INDIVIDUAL

Release Date

Apr 3, 2020, 1:41:45 PM

Due Date

Apr 10, 2020, 1:41:44 PM

Run Tests once after Due Date

Max Score

10

Presentation Score enabled

No

Template Repository Url

<https://artemistest2gitlab.ase.in.tum.de/TESTADAPTER/testadapter-exercise.git>

Solution Repository Url (optional)

<https://artemistest2gitlab.ase.in.tum.de/TESTADAPTER/testadapter-solution.git>

Test Repository Url (optional)

<https://artemistest2gitlab.ase.in.tum.de/TESTADAPTER/testadapter-tests.git>

Template Build Plan Id

TESTADAPTER-BASE

Solution Build Plan Id (optional)

TESTADAPTER-SOLUTION

Sequential Test Runs

false

Publish Build Plan

false

Allow Online Editor

true

Programming Language

Java

Package Name

de.tum.in.ase

LTI

LTI Configuration

Template Result

❌ **Score 0%**, 0 of 13 passed (5 minutes ago) [🔄](#)

Solution Result

✅ **Score 100%**, 13 of 13 passed (5 minutes ago) [🔄](#)

Problem Statement



Sorting with the Strategy Pattern

In this exercise, we want to implement sorting algorithms and choose them based on runtime specific variables.

Part 1: Sorting

First, we need to implement two sorting algorithms, in this case **MergeSort** and **BubbleSort**.

You have the following tasks:

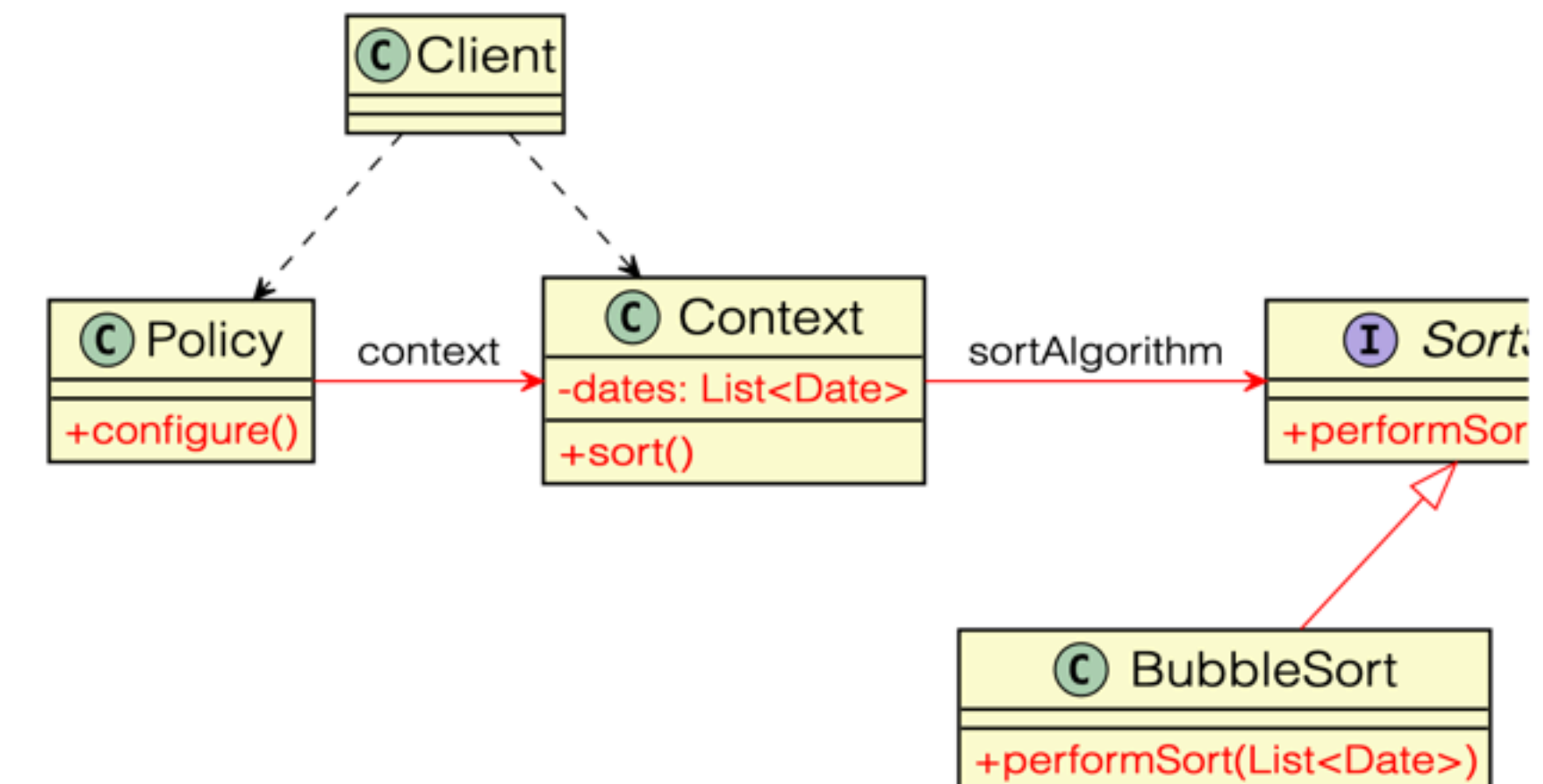
- ❌ **Implement Bubble Sort** 0 of 1 tests passing
 - Implement the method `performSort(List<Date>)` in the class **BubbleSort**. Make sure to follow the Bubble Sort algorithm exactly.
- ❌ **Implement Merge Sort** 0 of 1 tests passing
 - Implement the method `performSort(List<Date>)` in the class **MergeSort**. Make sure to follow the Merge Sort algorithm exactly.

Part 2: Strategy Pattern

We want the application to apply different algorithms for sorting a **List** of **Date** objects. Use the strategy pattern to select the right sorting algorithm at runtime.

You have the following tasks:

- ❌ **SortStrategy Interface** 0 of 2 tests passing
 - Create a **SortStrategy** interface and adjust the sorting algorithms so that they implement this interface.



Part 3: Optional Challenges

(These are not tested)

- Create a new class **QuickSort** that implements **SortStrategy** and implement the Quick Sort algorithm.
- Make the method `performSort(List<Dates>)` generic, so that other objects can also be sorted by the same method. Hint: Have a look at Java Generics and the interface **Comparable**.
- Think about a useful decision in **Policy** when to use the new **QuickSort** algorithm.

Grading Instructions

[← Back](#) [Edit](#) [Combine Template Commits](#) [Update Structure Test Oracle](#)

3. Update exercise code in repositories

- **Alternative 1:** Clone the 3 repositories and adapt the code on your local computer in your preferred development environment (e.g. Eclipse)
 - To execute tests, copy the template (or solution) code into a folder **assignment** in the test repository and execute the tests (e.g. using maven clean test)
 - Commit and push your changes
- **Alternative 2:** Open Edit in Editor in Artemis (in the browser) and adapt the code in online code editor
 - You can change between the different repos and submit the code when needed
- **Alternative 3:** Use IntelliJ with the Orion plugin and change the code directly in IntelliJ

Edit in Editor

The screenshot displays a coding editor interface with the following components:

- Top Bar:** Username 'ztre' with a green checkmark, a 'TEMPLATE' dropdown, a '+ Create Assignment Repository' button, a progress indicator '0%, 0 of 13 passed', and 'Save' and 'Submit' buttons.
- File browser (left):** Shows 'No items found'.
- Main Editor (center):** Contains the text 'Select a file to get started!'.
- Instructions Panel (right):**
 - Has 'Edit' and 'Preview' tabs, with 'Save' button.
 - Includes a rich text toolbar with icons for Bold (B), Italic (I), Underline (U), Quote, Code, Link, Image, List, and Style/Color options.
 - Contains a 'Formula' button, a '[task] Task' button, and two dropdown menus: 'Insert Test Case' and 'Add task specific hint'.
 - Displays a code snippet:

```
1 # Sorting with the Strategy Pattern
2
3 In this exercise, we want to implement sorting algorithms and choose them based
4
5 ### Part 1: Sorting
6
7 First, we need to implement two sorting algorithms, in this case `MergeSort` and
8
9 **You have the following tasks:**
10
11 1. [task][Implement Bubble Sort](testBubbleSort)
12 Implement the method `performSort(List<Date>)` in the class `BubbleSort`. Make
13
14 2. [task][Implement Merge Sort](testMergeSort)
15 Implement the method `performSort(List<Date>)` in the class `MergeSort`. Make
16
```
 - Bottom status bar shows 'Saved.', 'Test cases ok.', and 'Hints ok.' indicators.

3. Update exercise code in repositories

- Check the results of the template and the solution build plan
- They should not have the status **build failed**
- In case of a **build failed** result, some configuration is wrong, please check the build errors on the corresponding build plan.
- **Hints:**
 - Test cases should only reference code, that is available in the template repository. In case this is **not** possible, please try out the option **Sequential Test Runs**

4. Optional: Adapt the build plans

- The build plans are preconfigured and typically do not need to be adapted
- However, if you have additional build steps or different configurations, you can adapt the BASE and SOLUTION build plan as needed
- When students start the programming exercise, the current version of the BASE build plan will be copied. All changes in the configuration will be considered

5. Adapt the interactive problem statement

1 Programming Exercises

[+ Generate new Programming Exercise](#) [+ Import new Programming Exercise](#)

ID ↕	Title ↕	Short Name ↕	Release Date ↕	Due Date ↕	Max Score ↕	Repositories	Build Plans	Publish Build Plan ↕	Allow Online Editor ↕	
5	Adapter Patter	adapter	Apr 3, 2020, 1:41:45 PM	Apr 10, 2020, 1:41:44 PM	10	Template Solution Test	Template Solution	false	true	Scores Edit in editor View Reset Participations Manage Test Cases Edit Delete

- Click the Edit button of the programming exercise or navigate into Edit in Editor and adapt the interactive problem statement.
- The initial example shows how to integrate tasks, link tests and integrate interactive UML diagrams

6. Manage test cases

Manage Test Cases

Show inactive test cases

No unsaved changes

Not released

Save test cases

Reset weights

Trigger all

Id	Test Name	Weight	After Due Date ?	Is Active
45	testAttributes[Context]	1	<input type="checkbox"/>	true
42	testAttributes[Policy]	1	<input type="checkbox"/>	true
47	testBubbleSort	1	<input type="checkbox"/>	true
50	testClass[BubbleSort]	1	<input type="checkbox"/>	true
43	testClass[MergeSort]	1	<input type="checkbox"/>	true
49	testClass[SortStrategy]	1	<input type="checkbox"/>	true

7. Verify the exercise configuration

- Open the **View** page of the programming exercise

Template Result

⊗ **Score 0%, 0 of 13 passed** (18 minutes ago) 

Solution Result

✓ **Score 100%, 13 of 13 passed** (18 minutes ago) 

- The template result should have a score of **0%** with **0 of X passed**
- The solution result should have a score of **100%** with **X of X passed**
- Click on **Edit**
 - Below the problem statement, you should see **Test cases** ok and **Hints** ok

```
18
19 We want the application to apply different algorithms for sorting a `List` of `Date` objects.
20 Use the strategy pattern to select the right sorting algorithm at runtime.
21
22 **You have the following tasks:**
23
```

✓ Saved. | ✓ Test cases ok. | ✓ Hints ok.