schwarz IT

Web development in Kotlin

Kotlin workshop by Schwarz IT

https://github.com/SIT-Kotlin-Workshop

IntelliJ IDEA cheat sheet

Function	Windows	MacOS
Search everywhere	Double Shift	Double Shift
Show documentation	F1	F1
Auto-completion	Control + Space	Command + Space
Intention actions	Alt + Enter	Option + Enter
Comment out/in	Control + /	Command + /

Kotlin cheat sheet

Function	Code examples

Variable declarations	`var mutableString: String = "Mutable"`
	`val immutableString: String = "Immutable"`
	`val inferredString = "Type inferred"`
	`var nullableString : String? = "Nullable" // Can be null`

Nullability	`val safeNavigation = object?.property`
	
	`val elvis = nullableValue ?: "Alternative"`

Lists	`val immutableList = listOf("Immutable", "list")`
	`val mutableList = mutableListOf("Mutable", "list")`
	
	`val firstEntry = list[0] / list.first() / list.firstOrNull()`

Data classes	`data class MyClass(`
	`    val primary: String = "Hello",`
	`) {`
	`    val secondary: String? = "World"`
	`}`

Functions	`fun functionExpression(name : String) =`
	`    "Hello, $name"`
	
	`fun functionBlock(name : String) : String {`
	`    return "Hello, ${name.uppercase()}"`
	`}`
	
	`fun <T, V> higherOrder(x : T, f : (T) -> V) : V = f(x)`

Basic syntax

Exercise: For-loop

Task:

Open the file `ExerciseForLoop.kt.`

Write a function
fun `printEvenNumbers(n: Int)`
that prints all even numbers smaller than or equal to n.

Exercise: If-else

Task:

Open the file `ExerciseIfElse.kt.`

Write a function
`fun printGrades(score: Int) {`
that prints the grade associated to a score according to the following table:

- A: 90-100
- B: 80-89
- C: 70-79
- D: 60-69
- F: Below 60

Exercise: When

Task:

Open the file `ExerciseWhen.kt.`

Write a function
`fun dayOfWeek(day: Int)`
that takes an integer `day` and prints name of the day of the week, else `Invalid day number.`

Exercise: While-loop

Task:

Open the file `ExerciseWhileLoop.kt.`

Write a function
`fun countDown(start: Int) {}`
that takes an integer `start` and prints a countdown from `start` to 0 using a `while` loop.

Immutable data

Exercise: Immutable variables

Objective: Understand the concept of immutable variable using `val`.

Task:

Open the file `ExerciseImmutableValues.kt.`

- Declare an immutable variable to store your favorite number.
- Try to reassign a new value to this variable and observe the compiler error.
- Try the same with a mutable variable declared using `var`.

Exercise: Data classes

Objective: Create and use immutable data classes

Task:

Open the file `ExerciseDataClasses.kt.`

- Define an immutable data class `Student` with properties `name`, `age` and `grade`.
- Create an instance of the student class.
- Attempt to modify one of the properties and observe the behavior.
- Use the `copy` method to create a new instance with a modified `grade`.

Exercise: Lists

Objective: Work with immutable collections

Task:

Open the file `ExerciseImmutableLists.kt.`

- Create an immutable list of integers.
- Try to add or remove elements from list and observe the behavior.
- Create a new list by adding an element to the original list without modifying it.

Nullability

Exercise: Handling null safely

Open the file `ExerciseNullability.kt.`

- Write code that iterates over the `friends` list and prints information about each friend in a separate line. (e.g. for the first friend, print "`Adam (friends since 2023-05-02)`".)
  When the `friendsSince` date is `null`, print `UNKNOWN` instead.
- Do the same for the `moreFriends` list below. Print `UNKNOWN FRIEND` when the friend itself is `null`.
  You can try the various methods of handling `null` that you have seen on the slides.

Higher order functions

Exercise: Lambdas

Open the file `ExerciseLambda.kt.`

- Write a higher order function `operate` that takes two integers and a function as parameters. The function parameter should take two integers and return an integer.
- Use the `operate` function with a suitable lambda expression to perform addition, subtraction and multiplication.

Exercise: Map

Open the file `ExerciseMap.kt.`

- Modify the `applyOperationToList` function so that it applies the given operation to all elements of the list.
- Replace the missing lambdas in the `main` function below so that each entry in the list `numbers` is doubled or squared, respectively.
- Use `filter` from the standard library to get a list of the even numbers in the list `numbers`. Print this list.

Testing

Exercise: Write a few tests

The file `Simple.kt` contains a function `testMe`.

- When does `testMe` return `true`?
- In `SimpleTest.kt`, write a few tests for this method and execute them.
- How can testing help when refactoring code (see e.g. `testMeRefactored`)?

Exercise: Test-driven development

`TestDrivenDevelopmentTest.kt` contains tests for a `mysteriousFunction`.

- In `TestDrivenDevelopment.kt`, implement that function so that all tests are successful.

Exercise: Mocking

- Add a test for `WorkingInfoService.getWorkingHoursToday` to `WithDependencyTest` that tests the weekday case using mocking.
- Write tests for `WorkingInfoService.getRemainingWorkingHoursToday` that check that the function has the expected behavior at various points in time.

Web development

Exercise: Kodee's little online shop

The file `AdvancedRoutes.kt` contains endpoints implementing a REST API
that is used by a rudimentary frontend available at http://localhost:8080/frontend/advanced.html.

- Familiarize yourself with the code. Which functionality does each of the endpoints implement?
  (Feel free to add OpenAPI documentation as shown for `get("/decoupled", …)` in `BasicRoutes.kt`.
- What do we need to change so that the dates in the frontend are displayed in a nicer way?
  (E.g. instead of "2024-07-22T03:00", we would like to see something like "Monday, July 22, 2024 3:00".)

Exercise: Deleting orders

Oh no! The "Delete" button next to each order in the frontend doesn't work!

- In `AdvancedRoutes.kt`, implement an endpoint `delete("/order/{identifier}")` that reads an identifier from the path and deletes the corresponding order.
- Respond with an appropriate error when the order does not exist.
  (Can this error actually be caused by using the frontend?)
- To `AdvancedRoutesTest.kt`, add test(s) for your endpoint.

Exercise: Rescheduling orders

The "Reschedule" functionality doesn't seem to work either.

- In `AdvancedRoutes.kt`, implement an endpoint `post("/order/{identifier}")` that reads an identifier from the URL and reads an `OrderReschedulingDTO` from the body of the request and reschedules the corresponding order as specified by the user.
- Once you have the basic functionality working, you can implement some improvements, for example:
  - Orders can only be rescheduled if they are not in delivery yet.
  - Kodee only delivers on Monday to Tuesday 9:00 to 17:00 or Friday and Saturday 9:00 to 12:00
- Don't forget to write tests for your endpoint!

ORM

Exercise: Kodee's inventory management system

The files `ArticleRoutes.kt` and `ArticlePriceRoutes.kt` containsendpoints implementing a REST API
that is used by a rudimentary frontend available at http://localhost:8080/frontend/orm.html.

- Familiarize yourself with the code.
- Which parts of the code do not follow the "layered architecture" that we presented?
- The classes `ArticlePriceDTO` and `ArticlePriceCreationDTO` seem to be identical. Should we remove one of them?

Exercise: Searching for articles

The inventory management system has a search box that allows you to search for articles.

- Try searching for the following values: “123456”, “123”, “Kodee”, “stick”. What is the problem?
- Change the backend code so that the search result contains all articles whose identifier or name contains the search string.

Exercise: Price that!

In `ArticlePriceRoutes.kt`, implement an endpoint `post("/{articleIdentifier}")` that reads an article identifier from the URL and reads an `ArticlePriceCreationDTO` from the body of the request and creates a corresponding price.

- Create repository & service methods as needed so that your code follows the layered architecture pattern.
- Write at least one test for your route.
- Once you have the basic functionality working, you can implement some improvements, for example:
    - Change the price start / end dates to be shown in a user-friendly format and allow the user to create prices by using this format.
    - When creating a new price, make sure that there are no overlapping prices.