

Step Fulfilment Document

Tuesday, 11 March 2025 14:59

By: Grace Shuuya 220109702

1. Create a type alias `Identifier`, an `IdentifierFactory` class, and a `uniqueIdentifier` function in the `jetbrains.kotlin.course.alias.util` package.

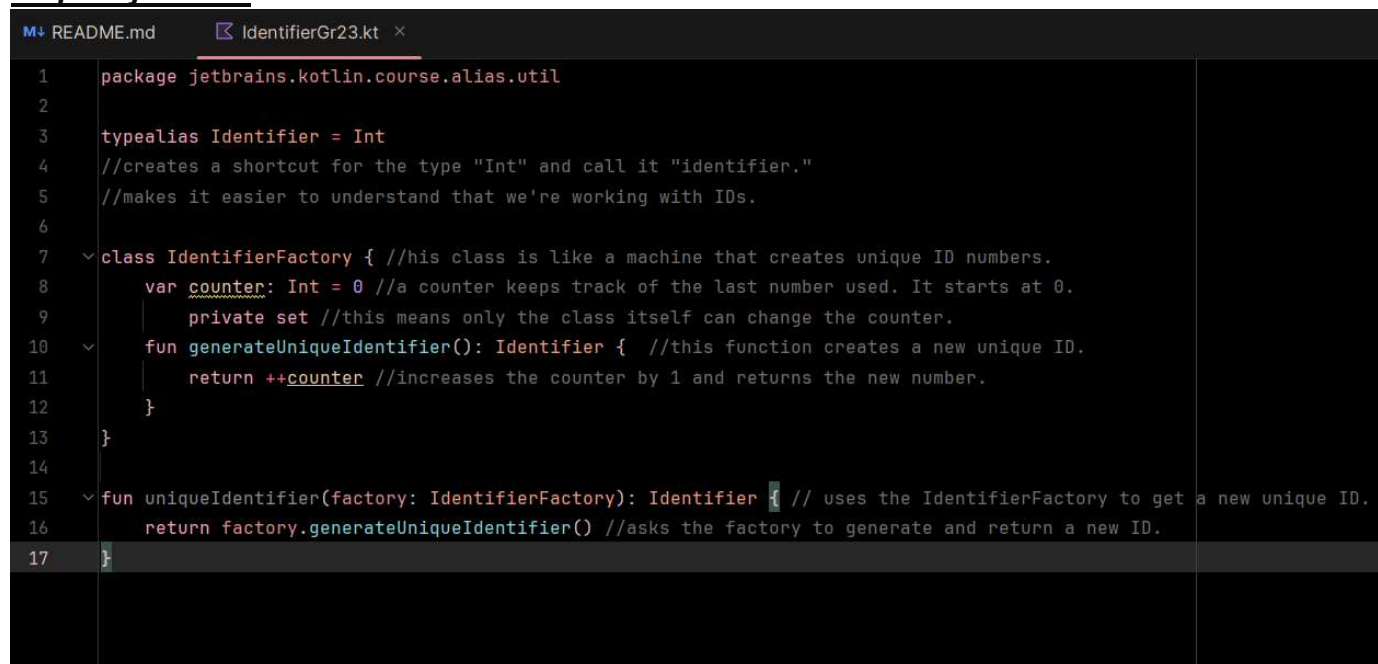
- The type alias `Identifier` needs to be an alias for the `Int` type. If you change the type in the future, e.g. create a new class, it will be changed automatically in all places.

- The `IdentifierFactory` class is a class for generating unique identifiers, e.g. identifiers for different game cards or teams. It should have a special `counter` – an `Int` property for storing the last unique number. By default, `counter` should be zero.

It should have a special `counter` – an `Int` property to store the last unique number. By default, `counter` should be zero.

- The `uniqueIdentifier` function returns a new unique identifier by incrementing the `counter` and returning it.

Step Fulfilment:



```
1 package jetbrains.kotlin.course.alias.util
2
3 typealias Identifier = Int
4 //creates a shortcut for the type "Int" and call it "identifier."
5 //makes it easier to understand that we're working with IDs.
6
7 class IdentifierFactory { //this class is like a machine that creates unique ID numbers.
8     var counter: Int = 0 //a counter keeps track of the last number used. It starts at 0.
9     private set //this means only the class itself can change the counter.
10    fun generateUniqueIdentifier(): Identifier { //this function creates a new unique ID.
11        return ++counter //increases the counter by 1 and returns the new number.
12    }
13 }
14
15 fun uniqueIdentifier(factory: IdentifierFactory): Identifier { // uses the IdentifierFactory to get a new unique ID.
16     return factory.generateUniqueIdentifier() //asks the factory to generate and return a new ID.
17 }
```

2. Create a data class `Team` in the `jetbrains.kotlin.course.alias.team` package to store the information about teams.

- It must have two properties in the primary constructor: `id` of `Identifier` type to identify each team and `points` of `Int` type to store the number of points in the game. For points, set the default value `0`.

- It must have an additional property `name`, which initializes automatically as `"Team#{id + 1}"` and will be shown in the leaderboard.

Step Fulfilment:

```
1 package jetbrains.kotlin.course.alias.team
2
3 import jetbrains.kotlin.course.alias.util.Identifier
4 //imports the Identifier type from the jetbrains.kotlin.course.alias.util package
5
6
7 data class Team( //this is a data class to store information about a team.
8     val id: Identifier, //unique ID for the team
9     var points: Int = 0, //the team's points, starting at 0 by default
10 ) {
11     //this property creates a team name like "Team#1", "Team#2", etc.
12     val name: String = "Team#${id + 1}"
13 }
```

3. The package `jetbrains.kotlin.course.alias.team` already has a regular class `TeamService`. It is responsible for the game logic for the teams. In this task, you need to implement several things to bring the game to life.

- Add a property `identifierFactory` with the type `IdentifierFactory` to generate identifiers for each team. Don't forget to add the default value for it by creating a new instance of the `IdentifierFactory` class.
- Add a companion object to the `TeamService` class and declare the `teamsStorage` variable to store all previous teams. The storage type should be `MutableMap`, which maps `Identifier` to `Team`. Don't forget to initialize it via an empty map.
- Implement the `generateTeamsForOneRound` method. The method must generate a list of teams and also store all of them in the `teamsStorage` map. To create new teams you need to use `identifierFactory` from the `TeamService` class to generate a new ID. We need to create this method to save game results for the leaderboard.

Step Fulfilment:

```
11 import jetbrains.kotlin.course.alias.util.Identifier
12 import jetbrains.kotlin.course.alias.util.IdentifierFactory
13 import org.springframework.stereotype.Service
14
15 @Service
16 class TeamService {
17     //a factory to generate unique IDs for teams
18     val identifierFactory: IdentifierFactory = IdentifierFactory()
19
20     //a companion object to store all teams in one place
21     companion object {
22         //a map to store teams, using their IDs as keys
23         val teamsStorage: MutableMap<Identifier, Team> = mutableMapOf()
24     }
25
26     //this method creates a list of teams for one round of the game.
27     fun generateTeamsForOneRound(teamsNumber: Int): List<Team> { //the line that was here before implementation
28         val teams = mutableListOf<Team>() //creates an empty list to store the new teams
29
30         //this is a loop to create the specified number of teams
31         for (i in 1..teamsNumber) {
32             //generates a unique ID for the team
33             val teamId = identifierFactory.generateUniqueIdentifier()
34             //creates a new team with the generated ID
35             val team = Team(id = teamId)
36             //adds the team to the list
37             teams.add(team)
38             //stores the team in the teamsStorage map for future reference
39             teamsStorage[teamId] = team
40         }
41
42         return teams //returns the list of created teams
43     }
44 }
```

4. Create two classes to work with the cards in the `jetbrains.kotlin.course.alias.card` package.

- A value class `Word` with one `String` `word` property to store a word.
- A data class `Card` to store information for each card. Each card must store an `id` with the `Identifier` type and a list of `words` (`List<Word>`). These properties don't have default values and must also be defined in the primary constructor.

Step Fulfilment:

```
1 package jetbrains.kotlin.course.alias.card
2
3 import jetbrains.kotlin.course.alias.util.Identifier
4
5 // this is a value class to store a single word. It is optimized
6 // for performance and avoids unnecessary object creation.
7
8 @JvmInline
9 value class Word(val word: String)
10
11 //data class to store information about a card.
12 data class Card(
13     val id: Identifier, //unique ID for the card
14     val words: List<Word> //list of words on the card
15 )
16
```

5. The package `jetbrains.kotlin.course.alias.card` already has the regular class `CardService`. You need to add several properties and implement several methods.

- Add a property `identifierFactory` with the type `IdentifierFactory` to generate identifiers for each card. Don't forget to add the default value for it by creating a new instance of the `IdentifierFactory` class.
- Add a property `cards` that stores a list of cards (`List<Card>`). Initialize it by calling the `generateCards` method.
- Add a companion object to the `CardService` class and declare the `WORDS_IN_CARD` const variable to store the number of words for the cards. You need to assign the value `4` to it. Also, declare `cardsAmount` here, which stores the possible number of cards: `words.size / WORDS_IN_CARD`. The project contains a predefined list of words called `words`.
- Add the `toWords` function to the `CardService` class, which is an extension function for `List<String>` and converts each element from this list into `Word`.
- Implement the `generateCards` function, which shuffles the `words` list, splits it into chunks with `WORDS_IN_CARD` words, takes `cardsAmount` chunks for `cardsAmount` cards, and finally creates a new `Card` for each chunk.
- Implement the `getCardByIndex` method, which accepts `index` (an integer number) and the `Card` at this index. If the card does not exist, throw an error.

Step Fulfilment:

```

12 package jetbrains.kotlin.course.alias.card
13 import jetbrains.kotlin.course.alias.util.IdentifierFactory
14 import org.springframework.stereotype.Service
15
16
17 @Service
18 class CardService {
19     //a factory to generate unique IDs for cards
20     private val identifierFactory: IdentifierFactory = IdentifierFactory()
21
22     //a list of cards, initialized by calling generateCards()
23     private val cards: List<Card> = generateCards()
24
25     // Companion object to store constants and shared properties
26     companion object {
27         //number of words per card
28         const val WORDS_IN_CARD: Int = 4
29
30         //predefined list of words (made up to fit 5 cards)
31         val words: List<String> = listOf(
32             "Apple", "Banana", "Orange", "Grape", "Salad", "Boerewors", "Braai", "Burger", "Matthew",
33             "Mark", "Luke", "John", "Android", "Apple", "Linux", "Windows", "Pink",
34             "Blue", "Red", "Purple"
35         )
36
37         //number of cards based on the number of words and WORDS_IN_CARD
38         val cardsAmount: Int = words.size / WORDS_IN_CARD
39     }
40
41     //extension function to convert List<String> to List<Word>
42     private fun List<String>.toWords(): List<Word> = this.map { Word(it) }
43

```

6. Implement several things in the already defined class `GameResultsService` in the `jetbrains.kotlin.course.alias.results` package.

- Add a type alias `GameResult` referring to `List<Team>` to the `jetbrains.kotlin.course.alias.results` package.
- Add a companion object to the `GameResultsService` and declare the `gameHistory` variable for storing the list of game results (`MutableList<GameResult>`). By default, it must be initialized via an empty list.
- Implement the `saveGameResults` method that adds the `result` to the `gameHistory`. Before adding the `result` you need to check two requirements and throw an error if they are broken: `result` must be not empty and all team IDs from the `result` must be in the `TeamService.teamsStorage`.
- Implement the `getAllGameResults` method that returns the reversed `gameHistory` list.

Step Fulfilment:

```
M4 README.md GameResultsService.kt ×
12 package jetbrains.kotlin.course.alias.results
13 import jetbrains.kotlin.course.alias.team.Team
14 import jetbrains.kotlin.course.alias.team.TeamService
15 import org.springframework.stereotype.Service
16
17 // Type alias for GameResult (a list of teams)
18 typealias GameResult = List<Team>
19
20 @Service
21 class GameResultsService {
22     // Companion object to store game history
23     companion object {
24         // Mutable list to store game results
25         val gameHistory: MutableList<GameResult> = mutableListOf()
26     }
27
28     // Function to save game results
29     fun saveGameResults(result: GameResult) {
30         // Check if the result is not empty
31         require(result.isNotEmpty()) { "Game result cannot be empty." }
32
33         // Check if all team IDs in the result are valid (exist in TeamService.teamsStorage)
34         val invalidTeams = result.filter { team ->
35             team.id !in TeamService.teamsStorage.keys
36         }
37         require(invalidTeams.isEmpty()) {
38             "Invalid teams found in the result: ${invalidTeams.joinToString { it.name }}"
39         }
40
41         // Add the result to the game history
42         gameHistory.add(result)
43     }
}
```

At the end of this part you will have the following application:
![The Alias game](./utils/src/main/resources/images/states/alias/state2.gif)

Step Fulfilment:

