Programação Orientada aos Objetos

- Trabalho 1 –
Universidade do Minho

TRABALHO REALIZADO POR:

GROUP 29

TEAM CONSISTS OF:

EDGAR CARVALHO FERREIRA
a99890

MIGUEL FERNANDES BARBOSA
a104451

OLEKSII TANTSURA
a102131

Data: 04/05/2024

Description of the Class Architecture Used

The class architecture designed for our application is based on the principles of object-oriented programming (OOP) to ensure modularity, reusability, and clarity. Each class is created to encapsulate specific functionalities and to interact with other classes through well-defined interfaces. Below you will see a detailed explanation:

Menu Class

Attributes: Includes Scanner for input, UserManager for managing user data, and lists of activities and users.

Purpose: Manages the user interface, facilitating interactions like registration, login, and access to various functionalities.

Decision: The Menu class works as a control center, managing user interactions and system responses. Its design uses dependency injection for the Scanner and UserManager for modularity enhancement.

UserManager Class

Attributes: Holds a Map of User objects.

Purpose: Responsible for user management operations such as add, remove, and search for users.

Decision: Centralizing user management tasks in this class reduces complexity in other parts of the system.

User Class

Attributes: User information such as ID, name, email, and lists of training sessions.

Purpose: Represents a user and encapsulates all related operations.

Decision: Encapsulating user data in a class simplifies management and operations on user data, ensuring consistency.

User Types (ProfessionalUser, AmateurUser, OccasionalUser)

Attributes: Inherits from User, with additional methods specific to the user type.

Purpose: Differentiate between various levels of user engagement.

Decision: Creating distinct classes for different user types allows the application to offer customized features and interfaces depending on the user's profile.

Activity Class Hierarchy

Base Class: Activity

Derived Classes: DistanceActivity, RepetitionActivity, WeightActivity, etc.

Attributes: Common attributes include name and duration. Specific activities add relevant attributes like distance or weight.

Purpose: Represents different types of physical activities.

Decision: Using a hierarchy allows for shared methods and attributes in the base class allowing us to be more specific in derived classes through inheritance.

Specific Activity Classes (e.g., BenchPress, Canoe)

Attributes: Each class includes specific attributes necessary for calculating activity specific metrics.

Purpose: Provide detailed implementations for different physical activities.

Decision: These classes enable precise tracking and calculations for various activities, which are essential for the application's functionality.

TrainSession Class

Attributes: Date and a list of activities.

Purpose: Manages individual training sessions, encapsulating all activities performed in a session.

Decision: This class allows the program to handle multiple activities within a session easily and to calculate related statistics such as total calories burned.

Menu Reusability (ListMenu)

Purpose: Used across different parts of the application to simplify navigation and ensure a consistent user experience.

Decision: The reusability of the Menu class enhances the system's maintainability and user experience by providing a consistent and familiar navigation across various user interactions.

Summary

The designed class architecture efficiently separates concerns among various system components, following OOP principles such as encapsulation, inheritance, and polymorphism. Each class has well-defined responsibilities, minimizing dependencies and making the system easier to maintain, test, and extend. This modular approach also supports scalability, as new functionalities or classes can be added with minimal impact on existing code.

Application Functionality

The application that was developed provides a robust user interface for managing fitness tracking activities, including user registration, session management, and reporting stats. Below we'll explain its functionalities, accompanied by references to the source code:

Main Menu Functionality

Implementation: displayMainMenu() (lines 35-57)

Function: Allows the user to navigate through options such as registering, logging in, viewing all users, accessing stats, changing the date, or exiting the application.

Visual Representation:

```
 *** Main Menu ***
1 - Register (Create new Account)
2 - Login (Use an Account already registered)
3 - View all Users registered
4 - Stats
5 - Change Date
6 - Exit and Save
0 - Exit the application
Option:
```

User Registration

Implementation: menuRegister() (lines 60-147)

Function: Handles new user registration. Users can enter their unique ID, name, address, email, heart rate, and weight. It also allows choosing the type of user: Professional, Amateur, or Occasional.

Visual Representation:

```
===== Register =====
Create a new account
Enter a new and unique User Id (Leave empty to return): uniqueID
Enter your Name: UniqueName
Enter your Address: UniqueAddress
Enter your Email: uniqueemail@gmail.com
Enter your Average Heart Rate (BPM): 60
Enter your Weight (Kg): 70

 *** Registration Menu ***
1 - Professional
2 - Amateur
3 - Occasional
4 - Return to Main Menu
0 - Exit the application
Option: 1
New Professional User created successfully!
```

User Login

Implementation: menuLogin() (lines 150-181)

Function: Allows users to log into their accounts using a unique ID. Validates user existence.

Visual Representation:

```
===== Login =====
Enter an account
Enter your id (Leave empty to return): uniqueID

You entered your account!
```

User Profile Management

Implementation: userMenu() (lines 184-282)

Function: After login, this menu allows the user to update personal information such as name, address, email, heart rate, and weight, or to manage training sessions and account deletion.

Visual Representation:

```
 *** User Menu ***
1 - Change my Name
2 - Change my Address
3 - Change my Email
4 - Change my Average Heart Rate
5 - Change my Weight
6 - My Train Sessions
7 - Delete Account
8 - Exit to Menu
0 - Exit the application
Option:
```

Training Session Management

Implementation: userActivitiesMenu() (lines 285-345)

Function: Provides options to view stats, list activities, create new training sessions, and delete existing sessions. Each training session can include various activities with specific attributes.

Visual Representation:

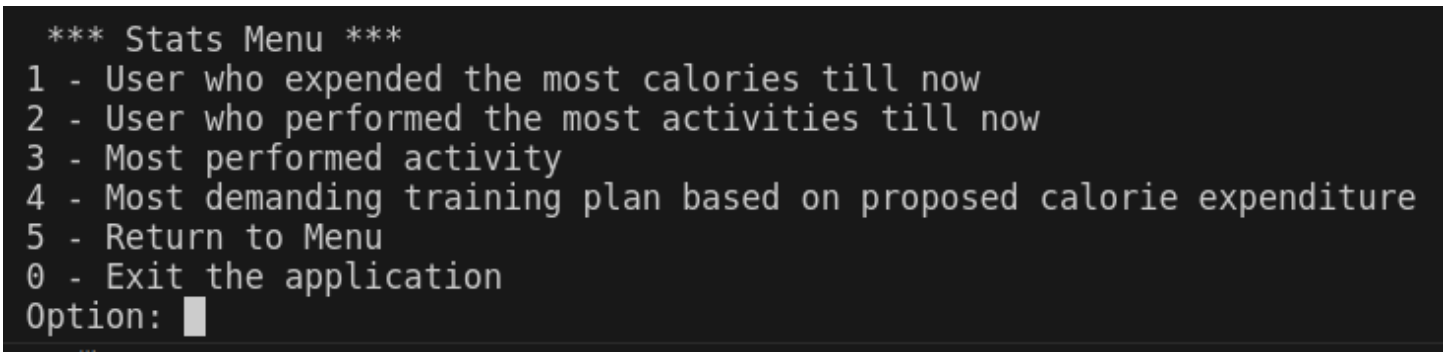
```
 *** My Train Sessions ***
1 - My User Training Stats
2 - My Activities
3 - Create new Train Session
4 - Delete a Train Session
5 - Exit to Menu
0 - Exit the application
Option:
```

Statistics Reporting

Implementation: displayStatsMenu() (lines 348-403)

Function: Offers statistical insights such as the user who expended the most calories, performed the most activities, and details about the most popular and demanding activities.

Visual Representation:

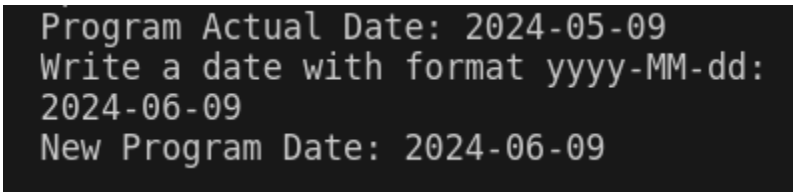


Date Management

Implementation: changeDate() (lines 600-612)

Function: Allows the user to change the current date of the system, impacting the reporting and scheduling of activities.

Visual Representation:



Data Persistence

Implementation: loadUserData() and exitAndSave() (lines 570-582, 584-593)

Function: Manages the loading and saving of user data to a file. Ensures that user data persists across sessions.

Summary

These functionalities collectively enable an efficient management of fitness activities and user data, providing a great interface for both casual and professional use. The application is built to be intuitive, allowing users to easily navigate through various functionalities and manage their fitness goals effectively.

Explanation of “final” Usage in Lambda Expressions

In our Java application, the final keyword is used in lambda expressions within the **menuRegister()** and **createTrainSession()** methods to ensure variables remain consistent when accessed by these expressions. This is because Java lambda expressions require variables referenced from their enclosing environment to be either final or effectively final.

menuRegister() Method

In menuRegister(), we use:

final User[] newUser = new User[1];

This array is used with lambda expressions that handle different user types during registration. Marking newUser as final ensures it cannot be reassigned within the lambda, maintaining thread safety and consistency across various lambda invocations.

createTrainSession() Method

Similarly, in createTrainSession():

final int[] nActivities = { 0 };

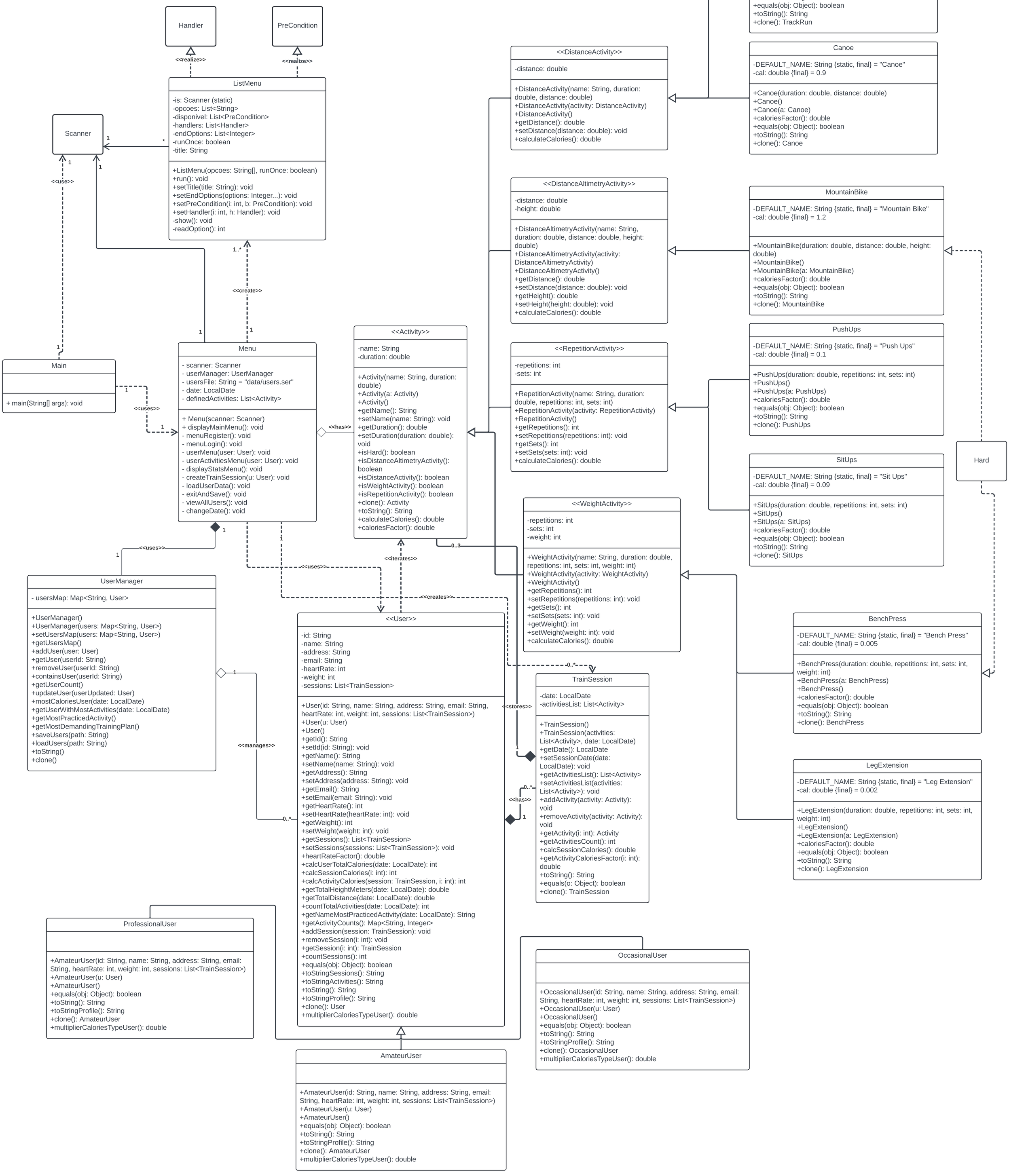
This array tracks the number of activities added. The final keyword prevents reassignment of the array reference, allowing only the content of the array to be modified, which is crucial since the array's state needs to persist correctly across successive lambda executions.

Final words

This project involved developing a fitness management app that simplifies tracking user activities and health metrics. Throughout the project, we applied key principles of object-oriented programming (OOP), such as encapsulation and modularity. Encapsulation helped us protect data and functionality within our classes, making our system more robust and our interfaces user-friendly.

We recognized the need to further enhance our code's modularity, which would allow for easier updates and expansions in the future. Additionally, we identified the importance of simplifying our user interface to improve overall user experience.

Overall, this project was a valuable exercise. It allowed us to put our theoretical knowledge into practice and taught us practical skills that will be beneficial for future software development tasks.


Handler
PreCondition
<<realize>>
<<realize>>
ListMenu
-is: Scanner (static)
-opcoes: List<String>
-disponivel: List<PreCondition>
-handlers: List<Handler>
-endOptions: List<Integer>
-runOnce: boolean
-title: String
+ListMenu(opcoes: String[], runOnce: boolean)
+run(): void
+setTitle(title: String): void
+setEndOptions(options: Integer...): void
+setPreCondition(i: int, b: PreCondition): void
+setHandler(i: int, h: Handler): void
-show(): void
-readOption(): int
Scanner
1
*
1
<<use>>
1
1
1..*
<<create>>
1
1
Main
+ main(String[] args): void
1
<<uses>>
1
Menu
- scanner: Scanner
- userManager: UserManager
- usersFile: String = "data/users.ser"
- date: LocalDate
- definedActivities: List<Activity>
+ Menu(scanner: Scanner)
+ displayMainMenu(): void
- menuRegister(): void
- menuLogin(): void
- userMenu(user: User): void
- userActivitiesMenu(user: User): void
- displayStatsMenu(): void
- createTrainSession(u: User): void
- loadUserData(): void
- exitAndSave(): void
- viewAllUsers(): void
- changeDate(): void
<<has>>
<<Activity>>
-name: String
-duration: double
+Activity(name: String, duration: double)
+Activity(a: Activity)
+Activity()
+getName(): String
+setName(name: String): void
+getDuration(): double
+setDuration(duration: double): void
+isHard(): boolean
+isDistanceAltimetryActivity(): boolean
+isDistanceActivity(): boolean
+isWeightActivity(): boolean
+isRepetitionActivity(): boolean
+clone(): Activity
+toString(): String
+calculateCalories(): double
+caloriesFactor(): double
0..3
1
1
<<uses>>
<<iterates>>
<<uses>>
<<creates>>
UserManager
- usersMap: Map<String, User>
+UserManager()
+UserManager(users: Map<String, User>)
+setUsersMap(users: Map<String, User>)
+getUsersMap()
+addUser(user: User)
+getUser(userId: String)
+removeUser(userId: String)
+containsUser(userId: String)
+getUserCount()
+updateUser(userUpdated: User)
+mostCaloriesUser(date: LocalDate)
+getUserWithMostActivities(date: LocalDate)
+getMostPracticedActivity()
+getMostDemandingTrainingPlan()
+saveUsers(path: String)
+loadUsers(path: String)
+toString()
+clone()
1
<<manages>>
0..*
<<User>>
-id: String
-name: String
-address: String
-email: String
-heartRate: int
-weight: int
-sessions: List<TrainSession>
+User(id: String, name: String, address: String, email: String, heartRate: int, weight: int, sessions: List<TrainSession>)
+User(u: User)
+User()
+getId(): String
+setId(id: String): void
+getName(): String
+setName(name: String): void
+getAddress(): String
+setAddress(address: String): void
+getEmail(): String
+setEmail(email: String): void
+getHeartRate(): int
+setHeartRate(heartRate: int): void
+getWeight(): int
+setWeight(weight: int): void
+getSessions(): List<TrainSession>
+setSessions(sessions: List<TrainSession>): void
+heartRateFactor(): double
+calcUserTotalCalories(date: LocalDate): int
+calcSessionCalories(i: int): int
+calcActivityCalories(session: TrainSession, i: int): int
+getTotalHeightMeters(date: LocalDate): double
+getTotalDistance(date: LocalDate): double
+countTotalActivities(date: LocalDate): int
+getNameMostPracticedActivity(date: LocalDate): String
+getActivityCounts(): Map<String, Integer>
+addSession(session: TrainSession): void
+removeSession(i: int): void
+getSession(i: int): TrainSession
+countSessions(): int
+equals(obj: Object): boolean
+toStringSessions(): String
+toStringActivities(): String
+toString(): String
+toStringProfile(): String
+clone(): User
+multiplierCaloriesTypeUser(): double
0..*
<<stores>>
1
TrainSession
-date: LocalDate
-activitiesList: List<Activity>
+TrainSession()
+TrainSession(activities: List<Activity>, date: LocalDate)
+getDate(): LocalDate
+setSessionDate(date: LocalDate): void
+getActivitiesList(): List<Activity>
+setActivitiesList(activities: List<Activity>): void
+addActivity(activity: Activity): void
+removeActivity(activity: Activity): void
+getActivity(i: int): Activity
+getActivitiesCount(): int
+calcSessionCalories(): double
+getActivityCaloriesFactor(i: int): double
+toString(): String
+equals(o: Object): boolean
+clone(): TrainSession
0..*
<<has>>
1
<<DistanceActivity>>
-distance: double
+DistanceActivity(name: String, duration: double, distance: double)
+DistanceActivity(activity: DistanceActivity)
+DistanceActivity()
+getDistance(): double
+setDistance(distance: double): void
+calculateCalories(): double
<<DistanceAltimetryActivity>>
-distance: double
-height: double
+DistanceAltimetryActivity(name: String, duration: double, distance: double, height: double)
+DistanceAltimetryActivity(activity: DistanceAltimetryActivity)
+DistanceAltimetryActivity()
+getDistance(): double
+setDistance(distance: double): void
+getHeight(): double
+setHeight(height: double): void
+calculateCalories(): double
<<RepetitionActivity>>
-repetitions: int
-sets: int
+RepetitionActivity(name: String, duration: double, repetitions: int, sets: int)
+RepetitionActivity(activity: RepetitionActivity)
+RepetitionActivity()
+getRepetitions(): int
+setRepetitions(repetitions: int): void
+getSets(): int
+setSets(sets: int): void
+calculateCalories(): double
<<WeightActivity>>
-repetitions: int
-sets: int
-weight: int
+WeightActivity(name: String, duration: double, repetitions: int, sets: int, weight: int)
+WeightActivity(activity: WeightActivity)
+WeightActivity()
+getRepetitions(): int
+setRepetitions(repetitions: int): void
+getSets(): int
+setSets(sets: int): void
+getWeight(): int
+setWeight(weight: int): void
+calculateCalories(): double
TrailRun
-DEFAULT_NAME: String {static, final} = "Trail Run"
-cal: double {final} = 1.3
+TrailRun(duration: double, distance: double, height: double)
+TrailRun()
+TrailRun(a: TrailRun)
+caloriesFactor(): double
+equals(obj: Object): boolean
+toString(): String
+clone(): TrailRun
TrackRun
-DEFAULT_NAME: String {static, final} = "Track Run"
-cal: double {final} = 1.0
+TrackRun(duration: double, distance: double)
+TrackRun()
+TrackRun(a: TrackRun)
+caloriesFactor(): double
+equals(obj: Object): boolean
+toString(): String
+clone(): TrackRun
Canoe
-DEFAULT_NAME: String {static, final} = "Canoe"
-cal: double {final} = 0.9
+Canoe(duration: double, distance: double)
+Canoe()
+Canoe(a: Canoe)
+caloriesFactor(): double
+equals(obj: Object): boolean
+toString(): String
+clone(): Canoe
MountainBike
-DEFAULT_NAME: String {static, final} = "Mountain Bike"
-cal: double {final} = 1.2
+MountainBike(duration: double, distance: double, height: double)
+MountainBike()
+MountainBike(a: MountainBike)
+caloriesFactor(): double
+equals(obj: Object): boolean
+toString(): String
+clone(): MountainBike
PushUps
-DEFAULT_NAME: String {static, final} = "Push Ups"
-cal: double {final} = 0.1
+PushUps(duration: double, repetitions: int, sets: int)
+PushUps()
+PushUps(a: PushUps)
+caloriesFactor(): double
+equals(obj: Object): boolean
+toString(): String
+clone(): PushUps
SitUps
-DEFAULT_NAME: String {static, final} = "Sit Ups"
-cal: double {final} = 0.09
+SitUps(duration: double, repetitions: int, sets: int)
+SitUps()
+SitUps(a: SitUps)
+caloriesFactor(): double
+equals(obj: Object): boolean
+toString(): String
+clone(): SitUps
Hard
BenchPress
-DEFAULT_NAME: String {static, final} = "Bench Press"
-cal: double {final} = 0.005
+BenchPress(duration: double, repetitions: int, sets: int, weight: int)
+BenchPress(a: BenchPress)
+BenchPress()
+caloriesFactor(): double
+equals(obj: Object): boolean
+toString(): String
+clone(): BenchPress
LegExtension
-DEFAULT_NAME: String {static, final} = "Leg Extension"
-cal: double {final} = 0.002
+LegExtension(duration: double, repetitions: int, sets: int, weight: int)
+LegExtension()
+LegExtension(a: LegExtension)
+caloriesFactor(): double
+equals(obj: Object): boolean
+toString(): String
+clone(): LegExtension
ProfessionalUser
+AmateurUser(id: String, name: String, address: String, email: String, heartRate: int, weight: int, sessions: List<TrainSession>)
+AmateurUser(u: User)
+AmateurUser()
+equals(obj: Object): boolean
+toString(): String
+toStringProfile(): String
+clone(): AmateurUser
+multiplierCaloriesTypeUser(): double
AmateurUser
+AmateurUser(id: String, name: String, address: String, email: String, heartRate: int, weight: int, sessions: List<TrainSession>)
+AmateurUser(u: User)
+AmateurUser()
+equals(obj: Object): boolean
+toString(): String
+toStringProfile(): String
+clone(): AmateurUser
+multiplierCaloriesTypeUser(): double
OccasionalUser
+OccasionalUser(id: String, name: String, address: String, email: String, heartRate: int, weight: int, sessions: List<TrainSession>)
+OccasionalUser(u: User)
+OccasionalUser()
+equals(obj: Object): boolean
+toString(): String
+toStringProfile(): String
+clone(): OccasionalUser
+multiplierCaloriesTypeUser(): double