# Java OOP Retake Exam – 19 December 2023



# Overview

There is always a huge thrill in climbing peaks. It's your turn to show what you can do. Go ahead and defeat the other climbers!

## Setup

* Upload **only the** **climbers** package in every task **except Unit Tests**.
* **Do not modify the provided interfaces or their packages.**
* Use **strong cohesion** and **loose coupling.**
* **Use inheritance and the provided interfaces whenever possible**.
  + This includes **constructors**, **method parameters,** and **return types.**
* **Do not** violate your **interface** **implementations** by adding **more public methods** in the concrete class than the interface has defined.
* Make sure you have **no public fields** anywhere.

## Task 1: Structure (50 points)

You are given **4** interfaces and must implement their functionalities in the **correct classes**.

There are **4** types of entities in the application: **Climber, Climbing, Mountain and Roster**. There are also **2** repositories: a **ClimberRepository** and a **MountainRepository**.

### Climber

Base**Climber** is a **base class** or any **type of Climber** and **should not be instantiated**.

#### Data

* **name** – **String**
  + If the value of the name is either **null** or **empty** (containing only whitespaces), throw a **NullPointerException** with the following message: **"Climber name cannot be null or empty."**
  + The values of the names are **unique.**
* **strength** – **double**
  + The strength of а Climber.
  + If the strength is a **negative** number, throw an **IllegalArgumentException** with the following message: **"Cannot create Climber with negative strength."**
* roster – Roster
  + A Roster field type.

#### Behaviour

##### void climb()

The **climb()** method decreases the climber's strength. Keep in mind that some Climber types can implement the method differently.

* The method **decreases** the climber's strength **depending on** Climber type:
* for Climber of type **WallClimber** by **30 units**.
* for Climber of type **RockClimber** by **60 units.**
* The strength value **should** **not** drop **below** **zero**.
* Set the value to be zero if the strength value drops below zero.

##### boolean canClimb()

The **canClimb()** method returns a **boolean**. Tell us if the strength is more than zero.

#### Constructor

A **BaseClimber** should take the following values upon initialization:

String name, double strength

#### Child Classes

There are several concrete types of **BaseClimber**:

**WallClimber**

Has **90 initial units of strength**.

The constructor should take the following values upon initialization:

Stri**ng** **name**

**RockClimber**

Has **120 initial units of strength**.

The constructor should take the following values upon initialization:

String name

### Roster

The RosterImpl class holds a **collection** of **peaks**. It should be **instantiated**.

#### Data

* **peaks** – a collection of **Strings**

#### Constructor

The constructor should not take any values upon initialization.

### Mountain

The **MountainImpl** class holds information about the **peaks** that can be conquered. It should be instantiated.

#### Data

* **name** – **String**
  + If the value of the **name** is either **null** or **empty** (containing only whitespaces), throw a **NullPointerException** with the following message: **"Invalid name!"**
* **peaksList** – a collection of Strings

#### Constructor

The constructor should take the following values upon initialization:

String name

### Climbing

The **ClimbingImpl** class holds the main action, which is the **conqueringPeaks** method.

#### Behaviour

##### void conqueringPeaks(Mountain mountain, Collection<Climber> climbers)

Here is how the **conqueringPeaks** method works:

* Climbers **cannot** climb peaks if their **strength** is **equal to or below** 0.
* They start climbing and **conquering peaks,** one by one.
* If they **conquer a peak**, they **add** it to their roster of conquered peaks and their **strength** is **decreased**.
* The conquered **peak** should then be **removed** from the **peaks list** of the current mountain.
* Climbers **cannot** **continue** climbing if their **strength** **drops** to 0.
  + If their strength drops to 0, the next climber starts climbing**.**

### ClimberRepository

The **ClimberRepository** class is a **repository** for climbers.

#### Data

* climbers – **a** **collection of climbers**

#### Behaviour

##### void add(Climber climber)

* Adds a climber to the collection.
* Every climber is unique in the collection.
  + It is guaranteed that there will not be a climber with the same name.

##### boolean remove(Climber climber)

* Removes a climber from the collection. Returns true if the deletion was successful.

##### Climber byName(String name)

* Returns a climber with that name.
* If the climber is not in the collection, return null.

##### Collection<Climber> getCollection()

* Returns an unmodifiable collection of climbers.

### MountainRepository

The **MountainRepository** class is a **repository** for mountains.

#### Data

* mountains **– a collection of mountains**

#### Behavior

##### void add(Mountain mountain)

* Adds a mountain to the collection.
* Every mountain is unique in the collection.
  + It is guaranteed that there will not be a mountain with the same name.

##### boolean remove(Mountain mountain)

* Removes a mountain from the collection. Returns true if the deletion was successful.

##### Mountain byName(String name)

* Returns a mountain with that name.
* If the mountain is not in the collection, return null.

##### Collection<Mountain> getCollection()

* Returns an unmodifiable collection of mountains.

## Task 2: Business Logic (150 points)

### The Controller Class

The business logic of the program should be concentrated around several **commands**. You are given interfaces that you must implement in the correct classes.

**Note: The** ControllerImpl **class SHOULD NOT handle exceptions! The tests are designed to expect exceptions, not messages!**

The interface is Controller. You must create a ControllerImplclass, which implements the interface and implements all its methods. The constructor of ControllerImpl does **not take** any **arguments**. It should be instantiated. The given methods should have the following logic:

### Commands

There are several commands, which control the business logic of the application. They are stated below.

#### AddClimber Command

##### Parameters

* **type – String**
* **climberName – String**

##### Functionality

Creates a **climber** with the given **name** of the given **type** and saves it in the repository. If the type is invalid, throw an **IllegalArgumentException** with the following message:

**"Climber type doesn't exist."**

Otherwise, the method should **return** the following message:

* **"Added {type}: {climberName}."**

#### AddMountain Command

##### Parameters

* **mountainName** - String
* peaks – String... (peaks)

##### Functionality

Create a **mountain** with the provided **peaks** and **name** and save it in the repository.

The method should **return** the following message:

* **"Added Mountain: {mountainName}."**

#### RemoveClimber Command

##### Parameters

* climberName – String

##### Functionality

Remove the climber from climbing by removing them from the repository. If a climber with that name doesn’t exist, **throw IllegalArgumentException** with the following message:

* **"Climber {climberName} doesn't exist."**

##### If a climbers is successfully removed, remove them from the repository and return the following message:

* **"Climber {climberName} was removed!"**

#### StartClimbing Command

##### Parameters

* **mountainName - String**

##### Functionality

When the start climbing command is called, the climbing at the given mountain begins.

* If you **don't have any** **climbers**, throw an **IllegalArgumentException** with the following message: **"You must have at least one climber to start climbing."**
* After climbing, you must **return the following message** with the **name of the climbed mountain** and the **count** of the **climber/s** that **was/were removed** from the climbing:

**"The climb took place at {mountainName}. {removedClimberCount} climber/s was/were removed from this climb."**

#### GetStatistics Command

##### Functionality

Returns the information about the climbers in the following format:

* If the climbers don't conquer any peak, print **"None"** in their place.

**"{count} mountain/s was/were climbed.**

**Climber's statistics:**

**Name: {climberName}**

**Strength: {climberStrength}**

**Conquered peaks: {peak1, peak2, peak3, …, peakn}**

**…**

**Name: {climberName}**

**Strength: {climberStrength}**

**Conquered peaks: {peak1, peak2, peak3, …, peakn}"**

### Input / Output

You are provided with one interface, which will help you with the correct execution process of your program. The interface is called **Engine** and its **implementational** class should read the input. When the program finishes, the class should print the **output** to the **console**.

#### Input

These are the input commands:

* **AddClimber** **{climberType} {climberName}**
* **AddMountain** **{mountainName} {**String... (peaks)}
* **RemoveClimber** **{climberName}**
* **StartClimbing {mountainName}**
* **GetStatistics**
* **Exit**

#### Output

Print the output from each command when issued. If an exception is thrown during any of the commands' execution, print the exception message.

#### Examples

|  |
| --- |
| **Input** |
| **AddClimber WallClimber Asen**  **AddMountain Himalaya Everest K2**  **StartClimbing Himalaya**  **RemoveClimber Mary**  **AddClimber RockClimber Georgi**  **AddMountain Rila Musala Irechek**  **StartClimbing Rila**  **GetStatistics**  **Exit** |
| **Output** |
| **Added WallClimber: Asen.**  **Added Mountain: Himalaya.**  **The climb took place at Himalaya. 0 climber/s was/were removed from this climb.**  **Climber Mary doesn't exist.**  **Added RockClimber: Georgi.**  **Added Mountain: Rila.**  **The climb took place at Rila. 1 climber/s was/were removed from this climb.**  **2 mountain/s was/were climbed.**  **Climber's statistics:**  **Name: Asen**  **Strength: 0**  **Conquered peaks: Everest, K2, Musala**  **Name: Georgi**  **Strength: 60**  **Conquered peaks: Irechek** |
| **Input** |
| **AddClimber WallClimber Zahari**  **AddClimber RockClimber Kiril**  **AddMountain Rhodopes GolyamPerelik GolyamPersenk Turlata**  **AddMountain Balkan Botev**  **StartClimbing Rhodopes**  **AddClimber RockClimber John**  **AddClimber RockClimber Silvy**  **RemoveClimber Silvy**  **StartClimbing Balkan**  **GetStatistics**  **Exit** |
| **Output** |
| **Added WallClimber: Zahari.**  **Added RockClimber: Kiril.**  **Added Mountain: Rhodopes.**  **Added Mountain: Balkan.**  **The climb took place at Rhodopes. 1 climber/s was/were removed from this climb.**  **Added RockClimber: John.**  **Added RockClimber: Silvy.**  **Climber Silvy was removed!**  **The climb took place at Balkan. 1 climber/s was/were removed from this climb.**  **2 mountain/s was/were climbed.**  **Climber's statistics:**  **Name: Zahari**  **Strength: 0**  **Conquered peaks: GolyamPerelik, GolyamPersenk, Turlata**  **Name: Kiril**  **Strength: 60**  **Conquered peaks: Botev**  **Name: John**  **Strength: 120**  **Conquered peaks: None** |

## Task 3: Unit Tests (100 points)

You will receive a skeleton with three classes inside – **Main**, **RockClimber,** and **Climbing**. **The Climbing** class will have some methods, fields, and constructors. Cover the whole class with the unit test to make sure that the class is working as intended. In Judge, you upload **.zip** to **stuntClimb (**with **ClimbingTests** inside**)** from the **skeleton**.