# **Java OOP Retake Exam**

**Athletics Competition**

**Overview**

*Athletics is a sport that requires speed, strength, and endurance. In this problem, you are going to simulate an athletics competition involving athletes of different types competing at various stadiums.*

## **Setup**

* Upload **only the** **athletics** package in every task **except Unit Tests**.
* **Do not modify the interfaces provided 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.
* You need to create the appropriate **getters and setters**.

## **Task 1: Structure (50 points)**

You are given **3** interfaces, and you must implement their functionalities in the **correct classes**.

There are **3** types of entities in the application: **SportsFacility, Athlete, Competition**. There is also 1 repository: a **SportsFacilityRepository**.

***Note: You must follow the provided structure, class names, method names exactly.***

A screenshot of a computer

AI-generated content may be incorrect.

## **Athlete**

**BaseAthlete** is a **base class** that implements the Athlete interface. It defines the common data and constructor for all types of athletes but should **not** be instantiated directly.

**Fields**

* **name** - **String**
  + If the value is null, empty, or contains only whitespaces, throw a IllegalArgumentException with the message: **"Athlete name cannot be null or empty."**
  + Names are unique **per facility.**
* **stamina** - **int**
  + Initial **stamina** depends on the **type** of athlete.
* **performance** - **int**
  + Tracks the performance points earned by the athlete. The initial value is **0** (zero).

**Constructor**

A **BaseAthlete** should take the following values upon initialization: **String name, int stamina**

**Abstract Methods:**

##### **public abstract void compete()**

**public abstract void train()**

These methods define the behavior of the athlete during a competition and training session. They are implemented **differently** by each concrete athlete type.

## **Child Classes**

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

**Runner**

* It has **80 initial units of stamina**.

The **constructor** should take the following values upon initialization: **String name**

**compete():**

* The method **increases** the Runner's performance by **1 point** and **decreases** its stamina value by 15.
* The stamina value **should** **not** drop **below** **zero**. If stamina drops below 0, set it to **0.**

**train()**

* The **train()** method **increases** stamina by **10**

**Jumper**

* It has **70 initial units of stamina**.

The **constructor** should take the following values upon initialization: **String name**

**compete()**

* The method **increases** the Jumper's performance by **1 point** and **decreases** its stamina value by 20.
* The stamina value **should** **not** drop **below** **zero**. If stamina drops below 0, set it to **0.**

**train()**

* The **train()** method **increases** stamina by **12**

**Thrower**

* It has **90 initial units of stamina**.

The **constructor** should take the following values upon initialization: **String name**

**compete()**

* The method **increases** the Thrower's performance by **1 point** and **decreases** its stamina value by 12.
* The stamina value **should** **not** drop **below** **zero**. If stamina drops below 0, set it to **0.**

**train()**

* The **train()** method **increases** stamina by **8**

**SportsFacility**

The Base**SportsFacility** class is a **base class** for any **type of SportsFacility** that implements the SportsFacility interface and **should not be instantiated.**

**Fields**

* **name** - **String**
  + If the value of the **name** is either **null** or **empty** (containing only whitespaces), throw a IllegalArgumentException with the following message: **"SportsFacility name cannot be null or empty."**
* **capacity** - **int**
  + Represents the maximum number of athletes that can be added to the facility.  
    If the value is **negative**, throw an IllegalArgumentException with the message:  
    **"Facility capacity cannot be a negative number."**
* **athletes – Collection<Athlete>**
  + Stores the athletes added to the facility.

**Constructor**

The constructor should take the following values upon initialization: **String name, int capacity**

**Behavior**

**protected abstract boolean isCompatible(Athlete athlete)**

* Implement **each** concrete **facility to define valid athlete types.**

**For example**: Track only allows Runner, JumpingPit only allows Jumper, ThrowingField only allows Thrower.

**void addAthlete(Athlete athlete)**

The BaseSportsFacility class implements a concrete addAthlete(Athlete athlete) method that performs the shared logic for all facility types:

* If the facility has reached its capacity throw IllegalArgumentException with the message: "**Not enough capacity."**
* If the athlete is not compatible (via the `**isCompatible()`** method) throw IllegalArgumentException: **"The facility is not appropriate for this type of athlete."**
* If **valid** athlete is **added** to the collection.

**Collection<Athlete> getAthletes()**

* Returns an unmodifiable collection of the athletes in the facility.

## **Child Classes**

There are several concrete types of SportsFacilityImpl, each suitable for a specific type of Athlete.

Each facility has a specific capacity and accepts only compatible athlete types.

**Track**

* **Accepts only**: Runner
* **Capacity**: 6 athletes

The constructor should take the following values upon initialization: **String name**

**JumpingPit**

* **Accepts only**: Jumper
* **Capacity**: 8 athletes

The constructor should take the following values upon initialization: **String name**

**ThrowingField**

* **Accepts only**: Thrower
* **Capacity**: 12 athletes

The constructor should take the following values upon initialization: **String name**

## **Competition**

The **CompetitionImpl** class holds the main action, which is the **executeCompetition** method.

**Behavior**

**List<Athlete> executeCompetition(SportFacility sportsFacility)**

This method simulates a competition between all athletes in the given facility. The competition follows these rules:

* Each athlete takes turns to **compete repeatedly**, one by one, until:
  + Their stamina reaches **0**, or
  + Their performance reaches the **target of 5**.
* The competition ends immediately **when any athlete reaches performance 5**.
* If at any point **no athlete is able to compete anymore** (because they are all exhausted or already reached performance 5), the competition stops.
* After the competition ends, the method:
  + Ranks all athletes based on:
    - First: **Performance**, in **descending** order.
    - Then: **Remaining stamina**, in **descending** order (to resolve ties).
  + Returns a list of the **top 3 performing athletes** (or fewer, if less than 3 competed).

## **SportsFacilityRepository**

The **SportsFacilityRepository** stores and manages all created sports facilities.

**Fields**

* **facilities - a collection of SportsFacility**

**Behavior**

**void add(SportsFacility facility)**

* Adds a new facility to the collection.

**boolean remove(SportsFacility facility)**

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

**SportsFacility byName(String name)**

* Return the facility with the given name, or null if not found.

**Collection< SportsFacility > getCollection()**

* Returns an unmodifiable collection of all facilities.

## **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 have a **ControllerImpl** class, which implements the interface and must implement 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.

### **AddSportsFacility Command**

**Parameters**

* **type - String**
* **name - String**

**Functionality**

Create a **facility of the given type (Track, JumpingPit, ThrowingField)** with the provided **type, name** and **capacity**. Add it to the repository.

* If a facility with the same name already exists in the repository, throw an **IllegalArgumentException** with message: **"This facility already exists in repository."**
* If the **SportsFacility** type is invalid, throw an **IllegalArgumentException** with message:

**"Invalid SpotrsFacility."**

If successfully addedthe method should **return** the following message:

* **"Successfully added {type}."**

### **AddAthlete Command**

**Parameters**

* **facilityName - String**
* **athleteType - String**
* **athleteName - String**

**Functionality**

Create an **Athlete** of the specified type (Runner, Jumper, Thrower) with the given **name** and save it to the collection of **Athletes** in the given **SportsFacility**. Athletes will always be added to **existing** facilities.

* If the **Athlete** type is invalid, throw an **IllegalArgumentException** with the following message: **"Invalid athlete."**
* If an **Athlete** with the same name already exists in the **SportsFacility**, throw an **IllegalArgumentException** with message:

**"Athlete with name {athleteName} already exist in this facility."**

* If the athlete **was added**, return:  
  **"Successfully added {athleteType} to {facilityName}."**

### **TrainAthlete Command**

**Parameters**

* **sportsFacilityName - String**
* **athleteName - String**

**Functionality**

Trains the specified athlete in the given facility.

If the **SportsFacility** where you are trying to train the athlete **does not** exist in the repository, throw a **NullPointerException** with the following message:

* **"{sportsFacilityName} does not exist in the repository."**

If the **Athlete** with the given name **does not** exist in the stadium, throw a **NullPointerException** with the following message:

* **"Athlete with name {athleteName} does not exist in {sportsFacilityName}."**

Otherwise, you should train the athlete and **return** the following message:

* **"Athlete {athleteName} trained successfully. Stamina: {stamina}"**

### **StartCompetition Command**

**Parameters**

* **sportsFacilityName - String**

**Functionality**

When the startCompetition command is called, the competition happens at the given facility.

The method retrieves the **top 3 athletes** and **returns** a message in the format:

**"Competition at {sportsFacilityName} completed.**

**Top 3 Athletes:**

**1 place: {name} - {performance}**

**2 place: {name} - {performance}**

**3 place: {name} - {performance}"**

* If there are **fewer than 3 athletes**, only list those available.

### **GetStatistics Command**

**Functionality**

Returns the information about athletes. Filter **only those that have performance > 0** in the following format:

**"Athletes competing at {sportsFacilityName1} {sportsFacilityType1}:**

**Name: {athleteName1}**

**Stamina left: {athleteStamina1}**

**Performance: {athletePerformance1}**

**...**

**Athletes competing at {sportsFacilityName-n} { sportsFacilityType-n}:**

**Name: {athleteName-n}**

**Stamina left: {athleteStamina-n}**

**Performance: {athletePerformance-n}"**

### **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:

* **AddSportsFacility {name} {type}**
* **AddAthlete {facilityName} {athleteType} {athleteName}**
* **TrainAthlete {facilityName} {athleteName}**
* **StartCompetition {facilityName}**
* **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** |
| **AddSportsFacility TrackOne Track**  **AddSportsFacility PitX JumpingPit**  **AddSportsFacility TrackOne Track**  **AddSportsFacility UnknownType Stadium**  **AddAthlete TrackOne Runner Alice**  **AddAthlete TrackOne Runner Bob**  **AddAthlete PitX Jumper Ivan**  **AddAthlete PitX Jumper Peter**  **AddAthlete PitX Thrower Martin**  **AddAthlete PitX Jumper Ivan**  **TrainAthlete TrackOne Alice**  **TrainAthlete TrackOne Alice**  **TrainAthlete TrackOne Bob**  **TrainAthlete PitX Peter**  **StartCompetition TrackOne**  **StartCompetition PitX**  **GetStatistics**  **Exit** |
| **Output** |
| **Successfully added Track.**  **Successfully added JumpingPit.**  **This facility already exists in repository.**  **Invalid SportsFacility.**  **Successfully added Runner to TrackOne.**  **Successfully added Runner to TrackOne.**  **Successfully added Jumper to PitX.**  **Successfully added Jumper to PitX.**  **The facility is not appropriate for this type of athlete.**  **Athlete with name Ivan already exist in this facility.**  **Athlete Alice trained successfully. Stamina: 90**  **Athlete Alice trained successfully. Stamina: 100**  **Athlete Bob trained successfully. Stamina: 90**  **Athlete Peter trained successfully. Stamina: 82**  **Competition at TrackOne completed.**  **Top 3 Athletes:**  **1 place: Alice - 5**  **2 place: Bob - 4**  **Competition at PitX completed.**  **Top 3 Athletes:**  **1 place: Peter - 5**  **2 place: Ivan - 4**  **Athletes competing at TrackOne Track:**  **Name: Alice**  **Stamina left: 25**  **Performance: 5**  **Name: Bob**  **Stamina left: 30**  **Performance: 4**  **Athletes competing at PitX JumpingPit:**  **Name: Ivan**  **Stamina left: 0**  **Performance: 4**  **Name: Peter**  **Stamina left: 0**  **Performance: 5** |
| **Input** |
| **AddSportsFacility FieldA ThrowingField**  **AddSportsFacility TrackB Track**  **AddAthlete FieldA Thrower Peter**  **AddAthlete FieldA Thrower Ivan**  **AddAthlete TrackB Runner Maria**  **AddAthlete TrackB Runner Anna**  **AddAthlete TrackB Runner Sofia**  **AddAthlete TrackB Runner Maria**  **TrainAthlete FieldA Peter**  **TrainAthlete FieldA Peter**  **TrainAthlete FieldA Peter**  **TrainAthlete FieldA Ivan**  **TrainAthlete TrackB Maria**  **TrainAthlete TrackB Maria**  **TrainAthlete TrackB Sofia**  **StartCompetition FieldA**  **StartCompetition TrackB**  **GetStatistics**  **Exit** |
| **Output** |
| **Successfully added ThrowingField.**  **Successfully added Track.**  **Successfully added Thrower to FieldA.**  **Successfully added Thrower to FieldA.**  **Successfully added Runner to TrackB.**  **Successfully added Runner to TrackB.**  **Successfully added Runner to TrackB.**  **Athlete with name Maria already exist in this facility.**  **Athlete Peter trained successfully. Stamina: 98**  **Athlete Peter trained successfully. Stamina: 106**  **Athlete Peter trained successfully. Stamina: 114**  **Athlete Ivan trained successfully. Stamina: 98**  **Athlete Maria trained successfully. Stamina: 90**  **Athlete Maria trained successfully. Stamina: 100**  **Athlete Sofia trained successfully. Stamina: 90**  **Competition at FieldA completed.**  **Top 3 Athletes:**  **1 place: Peter - 5**  **2 place: Ivan - 4**  **Competition at TrackB completed.**  **Top 3 Athletes:**  **1 place: Maria - 5**  **2 place: Anna - 4**  **3 place: Sofia - 4**  **Athletes competing at FieldA ThrowingField:**  **Name: Peter**  **Stamina left: 54**  **Performance: 5**  **Name: Ivan**  **Stamina left: 50**  **Performance: 4**  **Athletes competing at TrackB Track:**  **Name: Maria**  **Stamina left: 25**  **Performance: 5**  **Name: Anna**  **Stamina left: 20**  **Performance: 4**  **Name: Sofia**  **Stamina left: 30**  **Performance: 4** |

## **Task 3: Unit Tests (100 points)**

You will receive a skeleton with three classes inside – **Main**, **Stadium,** and **Athlete**. **Stadium** class will have some methods, fields, and constructor. Cover the whole class with the unit test using Junit 5 to make sure that the class is working as intended. In Judge, you upload **.zip** to **stadium package (**with **StadiumTests** inside**)** from the **test** package in the **skeleton**.