BioT Team Leader Exercise – Entity Cache

Guidelines

- Please read the requirements carefully first.
- You may implement your solution in Java/C#.

  Send your solution to exercise@biot-med.com
  - Email subject should be as follows: "<<Your Full Name>> - Exercise"
- If you should have any questions please send them to exercise@biot-med.com

Introduction

The following interface is given (in Java/C#):

```
Public interface Entity
{
        Public int getId();        // *** unique value ***
}
```

- An entity represents any object in a data model containing only properties.
- Entities are created, read, updated, and deleted (CRUD operations) by the users of the data model.
- Entities are stored in a **persistent document-based repository** e.g. relevant DB, file system, cloud storage etc.

Functional Requirements

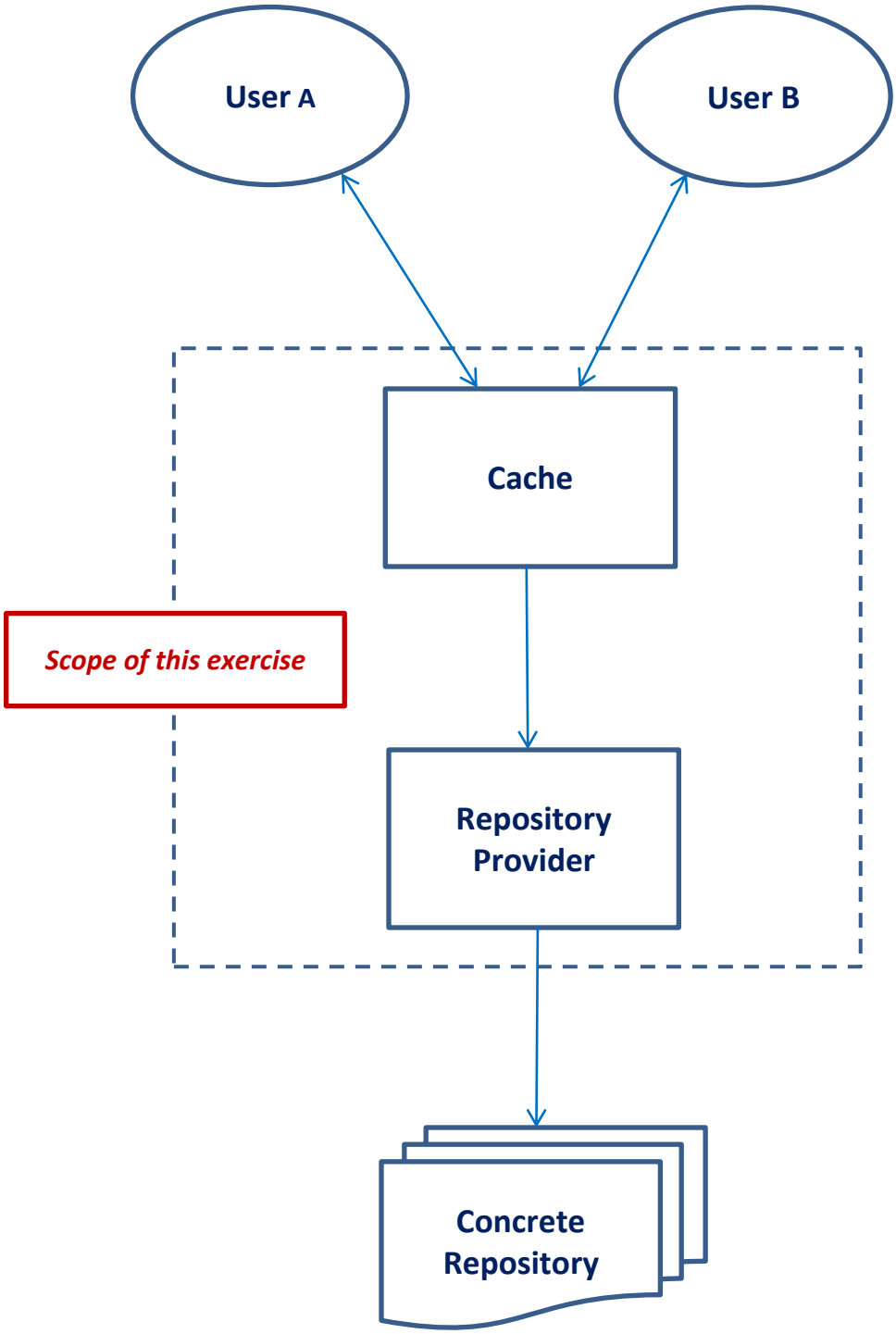
You are asked to implement a code library which provides its users with an **entity cache** module; the main purpose of the cache is to **optimize** access to entities e.g. read an updated entity from the cache in memory, and not directly from the repository.

1. The cache should be able to store **any type of entity**; but a single cache instance can contain only entities of the same type e.g. either persons OR vehicles.
2. The cache should support the following operations performed by the user:
   a. Add a new entity.
   b. Get a cached entity.
   c. Update a cached entity.

    d. Remove a cached entity.

3. The cache should be **initialized** by the user as follows:
    a. The user should specify a repository provider (see below), to which the cache should "attach" itself.
    b. The user should also specify the requested loading mode – eager/ lazy;
        i. In *eager loading* the cache should read all existing entities from the provider during initialization and populate this data in memory (note: this is the default mode, in case that you do not implement also lazy loading).
        ii. In *lazy loading* the cache should **not** read any entities from the provider until they are required in order to perform user operations "just-in-time".
4. The cache should use a **repository provider** which is responsible for accessing a specific type of repository (access layer), as follows:
    a. Each type of repository (e.g. document format) should have its own provider.
    b. The provider should support the following operations performed by the cache: add/ update/ remove entity.
    c. The provider might fail in executing any of these operations; and if so, then the cache should remain in a consistent state.
    d. The cache should **not** be aware of the document format used by the provider to save the entities' data into the repository (since any format may be used), and vice versa; the provider should **not** be aware of the data type (i.e. entity) used by the cache to save the entities' data as objects in memory. This means that each of these 2 components **cannot** use the data format which is used by the other component.
5. The cache should **notify** any external subscribers (e.g. users) whenever entities are added to/ updated in/ removed from it.
6. The cache should operate consistently in a multithreaded environment.

System Diagram



Testing Requirements

1. You are asked to provide a test program that covers the requirements for cache operations of add/ get/ update/ remove entity (#2 above).
2. These requirements should be verified **automatically** by your program; using a TDD unit-testing framework (e.g. Junit/ Visual Studio) is recommended
3. Please use an actual (not mock!) document-based repository provider for your tests.

Good luck! ☺