Technical Assessment

Instructions

Introduction

This document describes our Technical Assessment in terms of requirements and deliverables needed to be submitted. You should read it carefully before you begin working on the assessment.

The assessment consists of designing and implementing a sorting service.

The assessment is distributed with:

- instructions,
- use case describing the sorting service;
- test cases and;
- a simple design model that serves as the starting point for your own design.

Requirements

You are requested to produce a design model and an implementation of this model, respecting the following requirements.

It is not required to provide a fully functional application. An implementation of the sorting service is sufficient.

Functional Requirements

Functional requirements are described in the user story of the sorting service.

Note: Although the use case may seem very simple, pay special attention to the 'Special Requirements' section.

Other Requirements

The work involves a number of design choices that have to be made. In all such cases, the following principles should be applied.

Clarity and Maintainability

A clear design, such as will be readily understood by junior programmers, will be preferred to a complex one, even if the complex one is a little more efficient than the simple choice. However, poor algorithm design where standard solutions are well known will be penalized.

Documentation

The code itself should be as clear as possible, and "obvious" comments should be avoided. Awkward or complex code should have descriptive comments, and technical documentation must be used for each element of the class.

Correctness

The design used must correctly implement the specified requirements. **The sorting service must pass the provided test case.** Although this is not mandatory, you are free to provide the assessor with appropriate tests.

Use of Frameworks

External frameworks can be used to solve a part of the problem, provided that the framework and its dependencies are packaged within the submission.

Decoupling

Implementation choices must not impact the public interface of the sorting service. This applies to external frameworks as well, since these are implementation choices.

State of the Art

The design should use design patterns wherever appropriate. Do not invent your own design when well known and proven solution already exists. Best practices should be applied both for the design and for the coding of the solution.

Target Platform

Throughout this assessment, you must use Java, Ruby, JavaScript, PHP or .Net platform. You are not required to develop your code using any particular implementation of the platform, but the submission that you return *must have been tested and shown to work under a production (not development) version of the most used application servers*.

Duration

The assessment has been designed to require a maximum of 8 hours work.

Execution of Submissions

Your submission must run under a production (not development) version of chosen platform. You may develop using an IDE (Integrated Development Environment) but your final product may not have any residual dependency upon that.

When you submit your assignment, you must ensure that it is packaged in such a way that it is completely clear how the assessor should run it.

Deliverables

When you submit your assessment, you should provide the following parts:

- Document containing the design of your solution, including at least one UML class diagram and textual description of your design. The UML diagram must be in an image format: gif, jpeg or png formats are accepted. The document can be a MS-Word document, a plain ASCII document, or an HTML document.
- Full source code. You may use programming language artifact files to contain groups of elements of your submission as you deem appropriate. Dependencies such as external frameworks, if any, must be included as programming language library artifact files, such as JAR files at Java platform.
- A README.txt file that describes to the assessor the exact version of programming language you used, including the platform you worked on.

Marking

The marking of our Technical Assessment submission is done in two phases. First, the assessor runs the code, ensuring that it functions correctly through the specified operations.

Provided the essential behavioral requirements of the assignment have been correctly implemented, the assessor proceeds to investigate the design and implementation of your assignment. In this phase, the design is of greatest importance.

The grading process is closely controlled to ensure consistency and fairness, and it is performed according to criteria not disclosed publically.

In addition to the submission, you will be required to take an oral examination. This exam tests your understanding of your submission and asks you to justify a number of design choices embodied in that submission. For any design choice concerning topics not specifically described in the requirements, marks are awarded for a clear and consistent approach, rather than for any particular solution. **Design decisions should be *briefly but clearly* described in your comments.**

What to do if you have a question

You may find that you want to ask for further explanation of some part of these notes, perhaps to seek permission to solve a problem in a particular way. These notes deliberately leave some issues unspecified, and some problems unraised. Your ability to think through these issues, in the face of realistically imperfect specifications, and come to a tenable solution is something upon which you are being graded.

You should consider the options available and make a decision about how to address the problem yourself. This decision making process is part of the marking scheme, and as such it is crucially important that you provide documentation of your choice. Be sure to describe the options you considered, the perceived benefits and weaknesses of each, and why you chose the solution you did. You will not be marked so much on the choice that you made, but rather on the consistency of your

decision making process and your adherence to other aspects of these notes during that decision making process.

Use Case

ACTORS

- SSC - Sorting Service Client

FLOW OF EVENTS

Basic Flow

1. SSC sends a set of Books to be sorted by the Sorting Service.
2. The Sorting Service sorts the set of Books by title and author's name, both ascendently.
3. The Sorting Service returns the sorted set of Books to the SSC.

Preconditions

A non-null set to be sorted.

Post-conditions

The set of Books has been sorted successfully.

Related Use Cases

None.

Special Requirements

The Sorting Service considers one or more attributes to sort the Books. Also, for each attribute, a sort direction can be defined: ascending or descending. For example, in the point 2 of the flow, it considered the title and the author's name, both ascendently.
By this way, it is important to this solution, to provide means to prepare the Sorting Service with attributes and their directions.
The following design constraints apply:

- Without changes to the code and by means of a configuration file, the design must allow configure attributes and directions to the Sorting Service.

Test Cases

This document presents the test case to be passed by the implementation of the BooksSorter interface.

SCENARIOS

Rules	***Expected***	***Output Notes***
Title ascending	Books 3, 4, 1, 2	
Author ascending Title descending	Books 1, 4, 3, 2	
Edition descending Author descending Title ascending	Books 4, 1, 3, 2	
Null	-	Should throw an SortingServiceException
(empty set)	(empty set)	

DATA

The tests should consider as input the following set of Book instances:

	Title	***Author***	***Edition Year***
Book 1	Java How to Program	Deitel & Deitel	2007
Book 2	Patterns of Enterprise Application Architecture	Martin Fowler	2002
Book 3	Head First Design Patterns	Elisabeth Freeman	2004
Book 4	Internet & World Wide Web: How to Program	Deitel & Deitel	2007