**INTRODUCTION**

Publicis Sapient (PS) is on mission to help global clients revolutionize their digital landscape using technology as enabler. This requires us to build a strong technology team that can drive the transformations. The technology landscape is changing at a rapid pace but most fundamentals remain the same. So, we are looking for people with strong engineering fundamentals.

**INSTRUCTIONS**

1. Read instructions below carefully and plan your activities & time accordingly.
2. Over next **1.5 hours,** you will be showcasing your engineering fundamentals by coding for exercise mentioned below (**first 60 mins)** and discussing the solution & other technology topics (**next 30 mins)** with a PS technical expert.
3. Here are few criteria’s that we are looking for:
   1. Ability to understand the problem statement and approach towards the solution.
   2. Making reasonable assumptions wherever required and presenting them during the discussion
   3. Design & Code correctness
   4. Code completeness
   5. Code clarity and quality
   6. Presentation of the solution
4. Review the Unit test(s) carefully as they will answer any questions you may have regarding the acceptance criteria. Tests are already written so you do not spend time writing tests.
5. Set aside a few minutes to think through how you would implement any acceptance criteria you were not able to implement for discussion with PS technical expert. Make notes if required on how to implement these acceptances.
6. A maven-based starter project is provided that has basic structure & code to help you start.

**Problem description**

An application that Publicis Sapient is working on has very specific needs around a lightweight cache. For various reasons, reuse any of the readily available caching solutions is not an option. Hence, we need to create a home-grown solution.

You have been tasked with creating this cache. Here are the acceptance criteria this cache solution. Your solution should cover all Must-Haves requirements. Extra weightage will be given to solutions that also cover Nice to have requirements.

**Must Haves:**

1. At the bare minimum the cache **MUST** be able to perform add(K key, V value) and get(Object) operations.
2. The cache has its own type safety rules.
   * 1. User can add key of any type and value as long as a key of that type is not already present.
     2. Once a unique key type is added any subsequent entries should allow values which are of the same type or its subclass/subtype.

***Example****: Let’s say we have a class hierarchy of Shape -> Rectangle -> Square*

*The user should be able to add objects of all these classes to the cache against the same Key object say ShapeKey but if he tries to add a string or any object not from the Shape hierarchy against the ShapeKey it should fail.*

*Once ShapeKey is removed from the cache then he should be able to add a String or any other object as value to ShapeKey.*

*There are unit test covering this scenario*

*(Eg: testSuperAndSubTypesTypes\_RemoveAndAdd).*

1. The caching solution should work in a single JVM multithreaded environment with no loss of functionality.

**Nice to Haves:**

1. Avoid using any of the readily available java Map implementations to build the cache or to store caching data. Usage of List, Set, Arrays or any other Collection implementations is fine.
2. Solution allows for items to expire from cache at a preconfigured interval and this should be configurable at a key type level. Once a key type expires the rule of type safety should get reset.

**Getting Started:**

A Maven based starter project file(s) have the unit tests and value objects which can be used as key and value for the cache. The main class CustomCache.java has TODO where you can put your code. Feel free to create additional classes both top level and nested for this implementation. To test the solution, you can run the Unit test using the IDE or run the maven package target.