# 70094 18

## System Integration

## Submitters

| | |
|---|---|
| **sf23** | Shihan Fu |
| **kac23** | Karen Chave |

```
Final Tests: Summary for sf23 of v5
-----------------------------------

  Public Tests:
    Compiles:                  1 / 1
    Tests Pass:                1 / 1
    Coverage and Style Checks: 1 / 1

Git Repo: git@gitlab.doc.ic.ac.uk:lab2324_spring/SE_Design_Ex8_sf23.git
Commit ID: e3d4b
```

Although the code works, and your cache does what it should, you have a lot of coupling between your classes and the third party forecaster
An adapter could have been used to encapsulate the third party library (-1), and an interface for weather services (-1) that avoids coupling with the same data types as the third party library. (-1). Also the Proxy could be using this interface so that client code can change between a cache weather service or a non-cached one easily (-1). It would have been nice to see some tests on the cache that mock the downstream source ensuring that interaction is correct (-1). In general naming could be imporved (-1).

I suggest working through the example solution to undertand better how to use proxies and adapters.

13/20

```java
package ic.doc;

import com.weather.Day;
import com.weather.Forecast;
import com.weather.Forecaster;
import com.weather.Region;
import org.jmock.Expectations;
import org.jmock.integration.junit4.JUnitRuleMockery;
import org.junit.Test;
import static org.junit.Assert.assertEquals;

public class ForecasterTest {
  @Test
  public void forecasterFunctionTest() {
    Forecaster forecaster = new Forecaster();
    Forecast forecast1 = forecaster.forecastFor(Region.LONDON, Day.MONDAY);
    Forecast forecast2 = forecaster.forecastFor(Region.LONDON, Day.MONDAY);
    assertEquals(forecast1.summary(), forecast2.summary());
    assertEquals(forecast1.temperature(), forecast2.temperature());
  }
}
```





Client is a very generic name.

```java
package ic.doc;

import com.weather.Day;
import com.weather.Forecast;
import com.weather.Forecaster;
import com.weather.Region;
import org.jmock.Expectations;
import org.jmock.integration.junit4.JUnitRuleMockery;
import org.junit.Before;
import org.junit.Rule;
import org.junit.Test;

import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.io.PrintStream;

import static org.hamcrest.CoreMatchers.containsString;
import static org.junit.Assert.assertTrue;
import static org.junit.Assert.assertThat;

public class ClientTest {
  final String fileName = "forecast_test_cache.txt";
  final ForecasterProxy forecasterProxy = new ForecasterProxy(100, 3600 * 1000, fileName);

  @Test
  public void triggerCleanOld() {
    ForecasterProxy forecasterProxyCleanOld =
        new ForecasterProxy(1, 3600 * 1, "forecast_cache_old.txt");
    forecasterProxyCleanOld.forecastFor(Region.LONDON, Day.MONDAY);
    assertTrue(forecasterProxyCleanOld.evictOldEntries());
  }
```

This test does not check if the class cleans old enties, the test does it itself.
Also, might want to use a mock clock for this. -1

```java
  @Test
  public void testForecasterInitialize() {
    ForecasterProxy forecasterProxyInitial = new ForecasterProxy(100, 3600 * 1000);

    assertTrue(forecasterProxyInitial.loadCacheFromFile());
  }

  @Test
  public void evictOldEntriesTest() {

    ForecasterProxy forecasterProxyOld =
        new ForecasterProxy(1, 3600 * 1000, "forecast_cache_old.txt");
  }
```

What is this method testing?

```java
  @Test
  public void testForecasterProxyResult() throws FileNotFoundException {

    ByteArrayOutputStream outstream = replaceSystemOutStreamForTesting();
    forecasterProxy.forecastFor(Region.LONDON, Day.MONDAY);
    String actualOutput = outstream.toString();
    assertThat(actualOutput, containsString("Requesting"));

    outstream = replaceSystemOutStreamForTesting();
    forecasterProxy.forecastFor(Region.LONDON, Day.MONDAY);
    actualOutput = outstream.toString();
    assertThat(actualOutput, containsString("cache"));

    //    deleteFile();
    clearFile();
  }
```

```java

  public void clearFile() throws FileNotFoundException {
    PrintWriter pw = new PrintWriter(fileName);
    pw.close();
  }

  public void deleteFile() {
    // Create a File object representing the file to be deleted
    File file = new File(fileName);
    // Check if the file exists
    if (file.exists()) {
      // return if the file is deleted
      file.delete();
    }
  }

  public static ByteArrayOutputStream replaceSystemOutStreamForTesting() {
    ByteArrayOutputStream outstream = new ByteArrayOutputStream();
    System.setOut(new PrintStream(outstream));
    return outstream;
  }
}
```





```java
package ic.doc;

import com.weather.Forecast;
import org.junit.Test;

import static org.junit.Assert.assertEquals;

// Unit Test
public class CacheEntryTest {

  @Test
  public void cacheEntryTest() {
    long currentTime = System.currentTimeMillis();
    CacheEntry cacheEntry = new CacheEntry(new Forecast("summary", 0),
currentTime);
    assertEquals(currentTime, cacheEntry.getTimeStamp());
  }
}
```

You are coupling this class (and others) with com.weather.Forecast which is third party. If one day you decide to use a different third party forecaster you will have to change many classes.

Write test methods with names that state what you are testing.

```java
package ic.doc;

import com.weather.Day;
import com.weather.Forecast;
import com.weather.Forecaster;
import com.weather.Region;

import java.io.BufferedWriter;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;

public class ForecasterProxy extends Forecaster {
  private Forecaster forecaster;
  private Map<Region, CacheEntry> cache;
  private File cacheFile;
  private int maxCacheSize;
  private long cacheExpirationTime;

  public ForecasterProxy(int maxCacheSize, long cacheExpirationTime) {
    this(maxCacheSize, cacheExpirationTime, "forecast_cache.txt");
  }

  public ForecasterProxy(int maxCacheSize, long cacheExpirationTime, String fileName) {
    this.forecaster = new Forecaster();
    this.cache = new HashMap<>();
    this.maxCacheSize = maxCacheSize;
    this.cacheExpirationTime = cacheExpirationTime;
    this.cacheFile = new File(fileName);

    // Load cache from file if it exists
    if (cacheFile.exists()) {
      loadCacheFromFile();
    }
  }

  boolean loadCacheFromFile() {
    try (BufferedReader reader = new BufferedReader(new FileReader(cacheFile))) {
      String line;
      while ((line = reader.readLine()) != null) {
        String[] parts = line.split(":");
        Region region = Region.valueOf(parts[0]);
        long timestamp = Long.parseLong(parts[1]);
        String summary = parts[2];
        int temperature = Integer.parseInt(parts[3]);
        Forecast forecast = new Forecast(summary, temperature);
        cache.put(region, new CacheEntry(forecast, timestamp));
      }
    } catch (IOException e) {
      e.printStackTrace();
      return false;
    }
    return true;
  }

  private void saveCacheToFile() {
    try (BufferedWriter writer = new BufferedWriter(new FileWriter(cacheFile))) {
      for (Map.Entry<Region, CacheEntry> entry : cache.entrySet()) {
        Forecast forecast = entry.getValue().forecastData;
        writer.write(
            entry.getKey().name()
```

The proxy class is coupled the com.weatherForecaster. This was an opportunity to use an adaptor

Should implement an interface.





```java
                + ":"
                + entry.getValue().timestamp
                + ":"
                + forecast.summary()
                + ":"
                + forecast.temperature());
        writer.newLine();
      }
    } catch (IOException e) {
      e.printStackTrace();
    }
  }

  @Override
  public Forecast forecastFor(Region region, Day day) {
    long currentTime = System.currentTimeMillis();

    // Check if location is in cache and if it's still valid
    if (cache.containsKey(region)) {
      CacheEntry entry = cache.get(region);
      if (currentTime - entry.timestamp < cacheExpirationTime) {
        System.out.println("getting from cache");
        return entry.forecastData;
      } else {
        // Remove expired entry from cache
        cache.remove(region);
        saveCacheToFile();
        System.out.println("getting from request");
      }
    }

    // Fetch data from real forecaster
    Forecast forecastData = forecaster.forecastFor(region, day);

    // Update cache
    cache.put(region, new CacheEntry(forecastData, currentTime));
    saveCacheToFile();
    // Check if cache size exceeds the limit, and evict old entries if necessary
    if (cache.size() > maxCacheSize) {
      evictOldEntries();
      saveCacheToFile();
    }

    return forecastData;
  }

  boolean evictOldEntries() {
    long currentTime = System.currentTimeMillis();
    int initialSize = cache.size();
    cache
        .entrySet()
        .removeIf(entry -> currentTime - entry.getValue().timestamp > cacheExpirationTime);


    return initialSize != cache.size();
  }
}
```

```java
package ic.doc;

import com.weather.Day;
import com.weather.Forecast;
import com.weather.Forecaster;
import com.weather.Region;

public class Client {
//  public static void main(String[] args) {
//    // Create a proxy with max cache size and expiration time
//    Forecaster forecaster = new ForecasterProxy(100, 3600 * 1000); // 1 hour in milliseconds
//
//    long startTime = System.currentTimeMillis();
//
//    Forecast londonForecast = forecaster.forecastFor(Region.LONDON, Day.MONDAY);
//    System.out.println("London outlook: " + londonForecast.summary());
//    System.out.println("London temperature: " + londonForecast.temperature());
//
//    long endTime = System.currentTimeMillis();
//    long elapsedTime = (endTime - startTime) / 1000;
//    System.out.println("Elapsed time for the first call: " + elapsedTime + " seconds");
//
//    startTime = System.currentTimeMillis();
//
//    londonForecast = forecaster.forecastFor(Region.LONDON, Day.MONDAY);
//    System.out.println("London outlook: " + londonForecast.summary());
//    System.out.println("London temperature: " + londonForecast.temperature());
//
//    endTime = System.currentTimeMillis();
//    elapsedTime = (endTime - startTime) / 1000;
//    System.out.println("Elapsed time for the second part: " + elapsedTime + " seconds");
//  }

}
```





```java
package ic.doc;

import com.weather.Forecast;

public class CacheEntry {
  Forecast forecastData;
  long timestamp;

  public CacheEntry(Forecast forecastData, long timestamp) {
    this.forecastData = forecastData;
    this.timestamp = timestamp;
  }

  public long getTimeStamp() {
    return this.timestamp;
  }
}
```

Not needed.

```
-------- Test Output --------
Running LabTS build... (Wed  6 Mar 17:18:58 UTC 2024)

Submission summary...
You made 5 commits
  - e6df316 feat: build client with cache with the proxy pattern [4 files changed, 110 insertions, 2 deletions]
  - b619304 feat: add a file to store local cache [2 files changed, 41 insertions, 1 deletion]
  - 2c229a8 feat: add triggerCleanOldTest [12 files changed, 255 insertions, 141 deletions]
  - b4def5f test: add Unit test for CacheEntry [4 files changed, 17 insertions, 9 deletions]
  - e3d4b4a fix: set triggerCleanOld’s cache file to default [1 file changed, 1 insertion, 1 deletion]

Preparing...

BUILD SUCCESSFUL in 471ms

Compiling...
BUILD SUCCESSFUL in 4s

Running tests...

ic.doc.CacheEntryTest > cacheEntryTest PASSED

ic.doc.ClientTest > triggerCleanOld PASSED

ic.doc.ClientTest > evictOldEntriesTest PASSED

ic.doc.ClientTest > testForecasterInitialize PASSED

ic.doc.ClientTest > testForecasterProxyResult PASSED

ic.doc.ForecasterTest > forecasterFunctionTest PASSED

BUILD SUCCESSFUL in 26s

Checking test coverage and code style...
BUILD SUCCESSFUL in 4s
Finished auto test. (Wed  6 Mar 17:19:41 UTC 2024)

-------- Test Errors --------

```