**Part 1**

一个ZIP存档文件(不超过一个)，包含您对第1部分问题的回答的所有源代码文件；

一个PDF文件(不超过一个)，其中包含程序设计的书面理由和显示每个应用程序输出示例的屏幕截图。

**Part 2**

一个PDF文件(不超过一个)，其中包含图表、描述、解释、讨论、程序列表(从代码中摘录，清楚地显示您为了完成任务做了什么)；

一个压缩存档文件(不超过一个)，包含您对Task 1-3的回答的所有源代码文件。

**Assignment**

**Part 1 – 算法和数据结构**

1. 从文本文档中提取单词是人工智能的许多基于文本的应用的第一步，例如，检测推特上的滥用推文。该任务要求您根据给定的词汇“Google-10000-English-no-sweers . txt”（ 这个词汇表包含了10，000个最常见的英语单词，按照在谷歌的万亿单词语料库中出现的频率排序）(您可以从学习中心下载这两个文件)从文档“an\_article.txt”中提取所有有效的单词。具体来说，如果“an\_article.txt”中的单词token与“Google-10000-English-no-sweers . txt”(不区分大小写)中的单词匹配，则保留该单词，否则将其丢弃。您应该使用数组列表在Java程序中存储数据，所以这个任务的返回是一个数组列表，包含来自“an\_article.txt”的所有有效单词。注意这两个文本文件比较大，你要考虑如何让你的程序高效。

[30 marks] (Functionality: 16, Design: 10, Ease of use: 2, Presentation: 2)

1. 执行合并排序(合并排序的伪代码在讲座幻灯片中提供)，以便按字母顺序对从上面获得的单词进行排序，即您的程序的输出将是按字母顺序排序的单词。对于合并排序算法，编写一个方法，例如mergeSort(…)，计算

•通过算法排序100个单词、200个单词、300个单词……等直到所有单词所需的时间。

•对元素排序时发生的移动/比较进行计数。

[20 marks] (Functionality: 12, Design: 4, Ease of use: 2, Presentation: 2)

1. 您应该为LinkedList的数据结构创建两种方法。您的数据结构应该有类名MyLinkedList。应该实现的两种方法是:

•在列表中的特定位置添加元素:

public void addAtPosition(int position，String item){…}

•从列表中删除一个元素，该元素位于该方法应该返回删除哪个节点的特定位置:

public Node deleteAtPosition(int position){…}

在这两种方法中，正确的处理错误(例如，如果位置不存在会发生什么)。

在学习中心将有一个MyLinkedList的框架代码，它包含:带有calculateSize(), addFirst(), addLast(), traverse(), findByPosition()方法并且您应该实现的addAtPosition(int position, String item), deleteAtPosition(int position)两个方法的签名的Node类和MyLinkedList类。

您可以在提供的框架代码中重用任何已实现的方法。不允许您更改所提供代码的实现方法的任何部分。

[10 marks] (Functionality: 10, Design: 0, Ease of use: 0, Presentation: 0)

**第 1 部分的评估标准**

***功能性***

• 程序在多大程度上执行问题要求的任务以及它的效率。

***设计***

• 代码设计得有多好，尤其是在易用性方面它可以保持或扩展。特别要考虑：

• 使用来自核心 API 的适当类型、程序控制结构和类别。

• 适当类别和方法的定义。

• 实现高效算法的创造力/考虑。

• 代码的"优秀"设计将仔细考虑使用最合适的数据结构，并针对高效和优雅的算法。代码的"坏"设计很少/不注意数据结构和算法效率。

***易用性***

• 输入/输出的格式化。

• 用户的互动。

• 代码如何处理无效的用户输入？某些数据的应用程序会崩溃吗？

***文档和演示文稿***

• 适当使用注释。

• 代码的可读性。

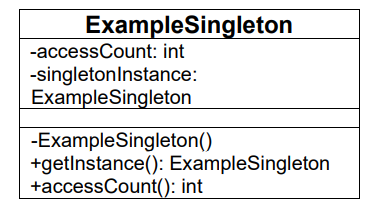
• 清晰和适当的截图。

**Part 2 - 设计和实施设计模式**

为了方便，本课程作业规范的末尾列出了以下每项任务的文件清单。

**Task 1**

从"学习中心"下载包含此任务的源代码文件的 zip 存档文件：**ExampleTest.java**。考虑下面的UML类图，描述了一种特殊的Singleton(单例)，其中accessCount()返回了Singleton通过getInstance()方法被访问的次数:



(i) 陈述Singleton设计模式的目的。

(ii) 写一个ExampleSingleton.java的实现:

a. ExampleSingleton()构造函数将打印出消息 “I, the ExampleSingleton, am being created”.

b. getInstance()方法应该打印出消息 “The sole instance of ExampleSingleton is being retrieved”.

c. 当在ExampleTest.java执行main()方法时，它将在标准输出流上生成以下输出:

I, the ExampleSingleton, am being created

The sole instance of ExampleSingleton is being retrieved

The ExampleSingleton has been accessed via the getInstance() method 1 time(s)

The sole instance of ExampleSingleton is being retrieved

The ExampleSingleton has been accessed via the getInstance() method 2 time(s)

(iii) 解释范例Singleton类是如何工作的，将关注点放在用于控制Singleton的创建和访问的特性上。

[10 marks]

**Task 2**

从学习中心下载包含此任务源代码文件的zip存档文件: CelsiusLogger.java, Test.java. 考虑下面的UML类图，描述一种特定的适配器，涉及两种基本类型的温度记录器:一种以华氏度为单位记录最近测量的温度；另一个是摄氏度:

图示

描述已自动生成

(i) 陈述适配器设计模式的目的

(ii) 写一个FahrenheitLogger的实现。getTemperature()和setTemperature()方法应该编写为没有任何副作用的获取和设置温度变量的值的访问器。

(iii) 写一个CelsiusToFahrenheitLoggerAdapter类的实现。创建适配器时，它应该创建它所引用的FahrenheitLogger。其方法应做到以下几点:

• setTemperature()应该调用FahrenheitLogger的setTemperature()方法，参数为 (aCelsiusTemperature \* 9/5 + 32)

• getTemperature()应该调用FahrenheitLogger的getTemperature()方法并返回 (((the FahreneitLogger’s temperature) – 32) \* 5/9)

(iv) 用你自己的话描述CelsiusToFahrenheitLoggerAdapter类的实现是如何工作的，关注点放在对实现这种设计模式的每个类的作用上。

[10 marks]

**Task 3**

从学习中心下载包含此任务的源代码文件的zip存档文件: : WeatherRecorder.java, WeatherObserver.java, WarningWatcher.java, TestWeather.java, 这个Java程序是一个不完整的实现，它使用观察者设计模式来输出特定天气预报事件发生时的警报。观察者正在监视 “Warning” 事件。一个监控英格兰的预警事件，另一个监控威尔士。

TestWeather类说明了WeatherRecorder/WeatherObserver的代码该如何使用。该程序应产生以下输出:

The WarningWatcher watching for Warnings for Wales

has noticed a new warning:

"Very Windy"

The WarningWatcher watching for Warnings for England

has noticed a new warning:

"Very Snowy"

(i) 陈述观察者设计模式的目的

(ii) 画一个UML类图，表示这个程序中产生的观察者设计模式，并解释每个类在实现这个设计模式中的作用。

(iii) 如所附代码注释所示，修改以下方法，以便获得上述输出:

• The WeatherRecorder attach() method

• the WeatherRecorder notifyObservers() method

• the WarningWatcher update() method.

(iv) 仔细解释你在方法中的微小改变如何有助于观察者设计模式的实现。[20 marks]

**第2部分的评估标准**

能够:

* 理解Java程序的目的并应用良好的objectoriented设计原则，以便对所提供的程序进行评论和/或修改(检查所提供的源代码文件)；
* 应用和实现与特定编程任务相关的设计模式；
* 反思设计模式在面向对象Java程序中的应用。

清晰的洞察力将获得更高的分数。

**任何代码的引用需要注意：**

* 转载的部分并不是一个问题的完整解决方案
* 您的解决方案可以利用核心Java API中的任何类。
* 源代码中明确引用代码的来源
* 对您的代码进行注释，以清楚地表明您理解复制的代码是如何工作的(即解释为什么选择了类型，为什么使用了其他语言特性，等等)。
* 您不得从任何其他网站下载代码。

**第2部分的程序列表**

**Task 1**

public class ExampleTest {

public static void main(String[] args) {

ExampleSingleton s = ExampleSingleton.getInstance();

System.out.println("The ExampleSingleton has been "

+ "accessed via the getInstance() method "

+ s.accessCount()

+ " time(s)");

s = ExampleSingleton.getInstance();

System.out.println("The ExampleSingleton has been "

+ "accessed via the getInstance() method "

+ s.accessCount()

+ " time(s)");

}

}

**Task 2**

public abstract class CelsiusLogger {

public abstract void setTemperature(double aCelsiusTemp);

public abstract double getTemperature();

}

public class Test {

public static void main(String[] theArguments) {

CelsiusLogger l= new CelsiusToFahrenheitLoggerAdapter();

l.setTemperature(22.0);

System.out.println("Current logged temperature: " + l.getTemperature() + " Celsius.");

}

}

**Task 3**

public class WeatherRecorder {

private ArrayList <WeatherObserver> observers = new ArrayList <WeatherObserver> ();

private String latestUpdateType; private String latestUpdateCountry;

private String latestUpdateText;

public void attach(WeatherObserver o) {

// Complete this method so it adds the observer to the observers list

}

public void setLatestNews(String theCountry, String theUpdateType, String theUpdateText) {

latestUpdateType=theUpdateType; latestUpdateCountry=theCountry;

latestUpdateText=theUpdateText; this.notifyObservers();

}

public String getUpdateType() { return latestUpdateType; }

public String getUpdateCountry() { return latestUpdateCountry; }

public String getUpdateText() { return latestUpdateText; }

private void notifyObservers() {

// Complete this method to go through each observer in turn,

// sending it a message to notify that an update has occurred

}

}

public abstract class WeatherObserver {public abstract void update(); }

public class WarningWatcher extends WeatherObserver {

private String countryWatched; private WeatherRecorder theRecorder;

public WarningWatcher(WeatherRecorder aWeatherRecorder, String countryToWatch) {

theRecorder = aWeatherRecorder; countryWatched = countryToWatch;

theRecorder.attach(this);

}

public void update() {

// Modify this so that it only prints out the update text

// if the update is a "Warning" for the country being watched

System.out.println("The WarningWatcher watching for Warnings for " +

countryWatched + "\nhas noticed a new warning:\n\"" +

theRecorder.getUpdateText() + "\"\n");

}

public class TestWeather {

public static void main(String[] args) {

WeatherRecorder wr = new WeatherRecorder();

WeatherObserver englandWatcher = new WarningWatcher(wr, "England");

WeatherObserver walesWatcher = new WarningWatcher(wr, "Wales");

wr.setLatestNews("England", "Current Temperature", "15");

wr.setLatestNews("Wales", "Warning", "Very Windy");

wr.setLatestNews("Wales", "Current Rainfall", "0");

wr.setLatestNews("England", "Warning", "Very Snowy");

}

}