405704 Programming 2/735320 Object-Oriented Applications
Assignment 3 – The Digital Service Provisioning Framework

Due date: Friday, October 16th 2015 at 5pm
This assignment is worth 10% of your final grade
This assignment is marked out of 100

Brief

Digital services are used throughout Information Technology. Digital Service Providers offer Internet connectivity, streaming movies and music and even computing infrastructure on a pay-as-you go basis. That is, the customer pays only for what they use. For this assignment, you will individually develop a Digital Service Provisioning Framework (DSPF) in Java. The DSPF comprises Java classes modelling digital service providers and their usage by customers. The Digital Service Provider classes implement functionality for a payment system interface and uses specialised exception classes extending the standard Java Exception classes.

Marking Scheme

To obtain full marks, you must design the DSPF using OOP design principles e.g. constructors, encapsulation and provide detailed Javadoc code commenting.

Criteria:	**Weight:**	**Grade A** **Grade Range:** **100 ≥ x ≥ 80%**	**Grade B** **Grade Range:** **80 > x ≥ 65%**	**Grade C** **Grade Range:** **65 > x ≥ 50%**	**Grade D** **Grade Range:** **50 > x ≥ 0%**
DigitalServiceProvider (sub)classes	**10%**	OOP paradigm consistently used for all service provider functionality. Interesting subclasses have been created and used correctly.	Inconsistent use of OOP paradigm but correct implementation of service providers. Reasonable subclasses created	Incorrect service provider functionality, unoriginal service providers or poor use of OOP paradigm	Absent service provider classes/functionality/ code does not compile
PaymentSystem Interface, Bill and ServiceUsage (sub)classes	**20%**	Correct specification and implementation of the interface. Appropriate Bill object generated.	Inconsistent use of OOP paradigm but correct implementation of interface. Appropriate Bill object generated.	Incorrect implementation of PaymentSystem, Bill or ServiceUsage classes	Absent classes, functionality or code does not compile
ServiceException classes	**15%**	OOP paradigm consistently used. Appropriate exception classes created and used correctly.	Inconsistent use of OOP paradigm but Appropriate use of exception classes in most places	Inconsistent use of OOP paradigm or poor use of exception classes.	Absent exception classes or code does not compile.
DigitalServiceSelectionApp	**25%**	The object-oriented program is unique, purposeful and provides an elegant demonstration of service provider selection. Program is interactive.	The object-oriented program is unique, purposeful. Reasonable demonstration of the service provider selection. Program is interactive.	Object-oriented program features an incomplete demonstration of the service provider selection Program is not interactive.	Absent functionality of service provider selection or code does not run after compiling
Code Quality: **-Whitespace** **-Naming** **-Reuse** **-Modularity** **-Encapsulation**	**15%**	Whitespace is comprehensively consistent. All naming is sensible and meaningful. Code reuse is present. Code is modular. Code is well encapsulated.	Whitespace is comprehensively consistent. Majority of naming is sensible. Code is modular. Code is encapsulated.	Whitespace is comprehensively consistent. Code has some modularity. Code has some encapsulation.	Whitespace is inconsistent and hence code is difficult to read.
Documentation Standards: **-Algorithms Commented** **-Javadoc**	**15%**	Entire codebase has comprehensive Javadoc commenting. Algorithms are well commented.	Majority of the codebase features Javadoc commenting. Majority of algorithms are commented.	Some Javadoc comments present. Some algorithms are commented.	No Javadoc comments present.

Methodology

Customer Accounts

Design a class modelling a Customer's service provider account. Your class stores the relevant details of the customer, such as their name and contact details which are supplied as input to a constructor. When a customer account is created, it is assigned a unique number (hint: use a class field to keep track of the number of accounts created). The customer's name is also modified and stored in uppercase letters. Customer accounts implement the **Comparable** interface and are ordered according the name of the Customer.

Digital Service Providers

Design an abstract **DigitalServiceProvider** class that maintains a record of the customer's usage of the service. You may consider using a **HashMap** to map customer accounts to their usage of the service. Your class will provide three methods to update and access the provider's records:

- `void subscribe(CustomerAccount,ServiceUsage)` adds the customer account and their initial service usage to the provider's records.
- `void unsubscribe(CustomerAccount)` removes the customer's account from the provider's records
- `ServiceUsage getUsage(CustomerAccount)` return the customer's service usage
- `void updateUsage(CustomerAccount,ServiceUsage)` updates the customer's service usage

These methods throw a **ServiceException** (or an appropriate subclass of) when an exceptional event occurs, such as the customer is not found in the records or is already subscribed.

Customer Transcripts

Select a suitable class to complete the method `String customerTranscript(boolean byName)` which outputs a string containing a sorted list of customer names and their service usage on a separate line. The last line prints the total number of customers. When **byName** is **true**, customers are sorted in ascending order according to their name. Otherwise, customers are sorted according to their usage of the service, in descending order (See the **ServiceUsage** class below).

Create the concrete subclass **CloudProvider** of **DigitalServiceProvider**.

Service Exception Classes

Design a **ServiceException** class extending Java's **Exception** class. An exception contains the customer account causing the exception, a text description of the error and the time the error occurred. This can be done by calling System.*currentTimeMillis()* in the constructor and storing the result in an instance variable. A nice representation of the time is obtained by invoking **new** Date(errorOccuranceTime).toString().

Create subclasses that are appropriate for exceptional events that occur when the service provider's records are updated. For example, **subscribe** may throw a **CustomerAlreadyExistsException** when a customer already existing in the service provider's records attempts to re-subscribe to the service.

Service Usage

Digital service provider's store details of each of their customer's usage of the service. Create an abstract **ServiceUsage** class which implements the **Comparable** interface.

Design a **CloudUsage** subclass of **ServiceUsage** containing the cloud customer's current usage information: *how many* virtual machines and virtual disk storage devices the customer is using.

Payment System

Digital service provider's bill their customers according to the amount of their service used. Create a **PaymentSystem** interface which contains a signature for a method to calculate a customer's **Bill** given the customer's account and their service usage.

The **DigitalServiceProvider** class implements the **PaymentSystem** interface.

Design a **Bill** class that stores the customer's account details, a description of the charges and the total amount owing.

CloudProvider Bill Payment

Implement the **PaymentSystem** interface's method in the **CloudProvider** class. Here's an example of a String output by the **Bill** class toString method:

```
Customer name: KENNETH JOHNSON. Account Number: 4
Contact details: WT703C WT BUILDING, AUT CITY CAMPUS
CLOUD USAGE:
1) $1.68 (12 Virtual Machines @ $0.14 each)
2) $3.57 (7 Disk Storage Devices @ $0.51 each)
TOTAL Amount owing: $5.25.
```

Extending Digital Service Providers

Extend the **DigitalServiceProvider** class with three more subclasses of Digital Services. You are required to complete

- Three subclasses of **DigitalServiceProvider**
- Three **ServiceUsage** subclasses, one for each new digital service provider
- An implementation of the **PaymentSystem** interface for each provider with their own unique bill calculation.

Program Interaction

Develop an application class called **DigitalServiceProviderSelectionApp** which has a main method that interacts with the core functionality of your classes. Your program should involve multiple Digital Services to select from, the ability for customer's to subscribe and unsubscribe. Your application should be capable of demonstrating all class functionality. You must catch and handle all exceptions arising from improper console input or exceptions thrown by the service providers.

Authenticity

Remember, it is unacceptable to hand in any code which has previously been submitted for assessment (for any paper, including Programming 2), and all work submitted must be unique and your own! We use automated methods to detect academic integrity breaches.

Submission Instructions

Submit the following documents as an archive **`.zip`** file of your documents on AUTOnline before the deadline:

- Your source code (**`.java`** files).
- Sample console output demonstrating your program in use (**`.txt`** file)

Zip structure and file naming requirements. Please ensure your submission matches the following:

```
🗐 lastname-firstname-studentid.zip
    🗎 *.java
    🗎 studentid-console-sample.txt
```

Replace the underlined text with your personal details.

Late submissions will receive a grade of 0

An extension will only be considered with a Special Consideration Form approved by the School Registrar. These forms are available at the School of Computer and Mathematical Science located in the WT Level 1 Foyer.

You will receive your marked assignment via AUTonline. Please look over your entire assignment to make sure that it has been marked correctly. If you have any concerns, you must raise them with the TA who marked your assignment. You have **one week** to raise any concerns regarding your mark. After that time, your mark cannot be changed.

Do not go to the TA because you do not like your mark. Only go if you feel something has been marked incorrectly.

Kenneth Johnson

Auckland University of Technology

kenneth.johnson@aut.ac.nz