CS-AD 209 - Software Engineering Professor Michael Paik
New York University Abu Dhabi February 11, 2014

Assignment 1

Due Date: This assignment is due at 23:59:59 on 25 Febuary, 2014.

Submission: Solutions should be uploaded as an attachment to the appropriate Assignment on `http://newclasses.nyu.edu`

Style: Code should be well-commented, logically packaged, and easy to read. Variable names should be self-documenting wherever possible. No line of source code should exceed 80 characters.

Indications: You will work in groups of **2-3**, and implement in either Java or Python. Java code that does not compile will not be accepted.

PLEASE read the entire document before beginning.

You have been tasked by the Abu Dhabi Driver's Licensing Department to build the core infrastructure that will handle license issuance for expatriates from the United States, United Kingdom, and Australia. This entails dealing with the arrival of customers, the workflow assigning customers to agents, the processing of documents, and the issuance of licenses.

This being Abu Dhabi, you may find that the Driver's Licensing Department (henceforth **DLD**) is woefully underprovisioned for the number of customers, or incredibly overprovisioned, and your design must be flexible enough to support arbitrary numbers of agents and customers.

The requirements are as follow:

- The DLD has three queues - one for licensing, one for eye testing, and one for driver's license translation into Arabic.
- Customers enter the DLD and take a number for one of the three queues; this number must be unique and range from A0000 to A9999 for the licensing queue, B0000 to B9999 for the eye testing queue, and C0000 to C9999 for the license translation queue.
- Agents process customers in FIFO order; that is, in order of increasing number through 9999, at which point the number wraps back around go 0000 (with the appropriate prefix).
- License processing entails:
  - Entering biograhical details.

  - Entering existing driver's license details.
  - Entering visa information.
  - Entering Emirates ID information.
  - Verifying eye test information.
  - Taking payment and marking the application as paid.

- Applications are not always successful - if a customer fails to bring his or her passport, driver's license, driver's license translation, Emirates ID, or eye test results, the application fails until such materials are provided. Applications also fail if information on the documents - date of birth, first name, last name, nationality, gender, is inconsistent, or if one or more documents is expired.

- If an application is missing license translation or eye test results, the customer is redirected to the appropriate queue(s).

- Customers may be on both the eye test and license translation queue at the same time.

- A customer whose application fails must take a new number.

- Assume that each service - licensing, eye testing, and translation, have a separate group of agents. You may assume that all translations and eye tests are successful.

- If a customer is on both queues and is called for one service while another is being processed (e.g. if he or she is called for eye testing while his or her license is being translated), the customer loses his or her place and must take a new number for the service missed.

- Once a license application is successfully processed, the application enters a queue for one or more license card printers, is printed, and issued to the relevant customer.

- The printers are dumb and have no internal print queues/spoolers - they will not read from an external queue when they are idle and cards to be printed must be directly pushed to them. It takes 300 seconds to print a card - if a new job is pushed to the printer before it completes, it will jam, and so external queue management is necessary. Printers do, however, support a call `isIdle()` or `is_idle()` that returns a boolean indicating whether the printer is idle, and does not interrupt a print job in progress.

Implementation details (since we are simulating)

- License processing takes a uniformly random integer number of seconds between 120 and 300, inclusive.

- Eye tests take a uniformly random integer number of seconds between 120 and 300, inclusive.

- License translations take a uniformly random integer number of seconds between 300 and 600, inclusive.

- Each queue should support peeking at how long the queue is - the difference between the last number called and the next number that will be issued.

You may neglect persistence for the sake of simplicity. That is, you may assume that the program is always running and no database or file storage is required to save entered data between runs of the program. Further, you may assume that once a customer's license is issued, the information associated with that customer can be discarded.

1. [**0 points**] Get your programming environment set up; I assume that since you all have prior programming experience that this will not be a difficult task.

2. [**10 points**] Create a workflow diagram (use whatever format helps you understand the process - flowcharts work well for this sort of task) of how a license application flows through the system, with optional branches into license translation and eye testing. One branch should terminate in permanent failure if any of the required documents not available at the DLD (driver's license, passport, Emirates ID) is missing, expired, or inconsistent with other documents.

3. [**30 points**] Create a complete module dependency diagram (MDD) of your design for this overall system, and provide a separate document detailing design choices (where you used what object-oriented strategy, design pattern, or concurrent approach and *why*)

4. [**30 points**] Create a complete finite state machine (FSM) of the system assuming one agent in each queue (that is, non-concurrent). Describe what conceptual complexities arise when you add multiple agents (i.e. concurrency) to the situation.

5. [**30 points**] Create a working implementation of your MDD and FSM in Java or Python. Print to console liberally to display the motion of customers/applications through the system.

   Testing Details. I will be using automated testing to sanity check your implementation. Your constructor/initializer should take no arguments, and you should provide a method `run()` in your top-level class that takes a `java.util.Vector` of `Object[]`s (Java) or `list` of `tuple`s (Python). Each `Object[]`/`tuple` will contain an arrival time in number of seconds after starting and an instance of `Customer`. See definition of `Customer` and other data encapsulating objects in the Appendices. Source is also on newclasses. `run()` should be synchronous (blocking) and return a `java.util.Vector` or `list` of `UAEDriversLicense` or `UAE_Drivers_License` instances containing `Customer` instances with successful applications (see source).

6. [**25 points**] **Extra Credit.** Use the Strategy design pattern to allow substitution of queuing algorithms without changing any other code. Provide at least two materially different such queueing strategies (e.g. entirely serial, one queue after another, v.s. parallel queueing making use of prediction of time left in the various queues by peeking.)

Appendix A: Java Source

```
// Java version - package and imports in each source file as appropriate
package assignment1;

import java.util.Date;

public abstract class AbstractPrimaryDocument {
    public String firstName;
    public String lastName;
    public String nationality;
    public char gender;
    public Date dateOfBirth;
    public Date expiryDate;

    public AbstractPrimaryDocument(String firstName, String lastName,
                                   String nationality, char gender,
                                   Date dateOfBirth, Date expiryDate) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.nationality = nationality;
        this.gender = gender;
        this.dateOfBirth = dateOfBirth;
        this.expiryDate = expiryDate;
    }
}

public class EmiratesId extends AbstractPrimaryDocument {
    public String idNumber;

    public EmiratesId(String firstName, String lastName, String nationality,
                      char gender, Date dateOfBirth, Date expiryDate,
                      String idNumber) {
        super(firstName,lastName,nationality,gender,dateOfBirth,expiryDate);
        this.idNumber = idNumber;
    }
}

public class DriversLicense extends AbstractPrimaryDocument {
    public DriversLicense(String firstName, String lastName, String nationality,
                          char gender, Date dateOfBirth, Date expiryDate) {
        super(firstName,lastName,nationality,gender,dateOfBirth,expiryDate);
    }
```

```
}

public class Passport extends AbstractPrimaryDocument {
    public Passport(String firstName, String lastName, String nationality,
                    char gender, Date dateOfBirth, Date expiryDate) {
        super(firstName,lastName,nationality,gender,dateOfBirth,expiryDate);
    }
}

public class EyeTest extends AbstractPrimaryDocument {
    public String emiratesIdNumber;

    public EyeTest(String firstName, String lastName, String nationality,
                   char gender, Date dateOfBirth, Date expiryDate,
                   String emiratesIdNumber) {
        // Java date libraries suck; this is imprecise
        super(firstName,lastName,nationality,gender,dateOfBirth,
              expiryDate != null ? expiryDate : new Date(
                System.currentTimeMillis()+2592000000L));
        this.emiratesIdNumber = emiratesIdNumber;
    }
}

public class DriversLicenseTranslation {
    public DriversLicense driversLicense;

    public DriversLicenseTranslation(DriversLicense driversLicense) {
        this.driversLicense = driversLicense;
    }
}

// All members public for simplicity - no setters/getters, null by default
public class Customer {
    public EmiratesId emiratesId = null;
    public DriversLicense driversLicense = null;
    public Passport passport = null;
    public EyeTest eyeTest = null;
    public DriversLicenseTranslation driversLicenseTranslation = null;

    public Customer(EmiratesId emiratesId, DriversLicense driversLicense,
                    Passport passport, EyeTest eyeTest,
                    DriversLicenseTranslation driversLicenseTranslation) {
```

```
        this.emiratesId = emiratesId;
        this.driversLicense = driversLicense;
        this.passport = passport;
        this.eyeTest = eyeTest;
        this.driversLicenseTranslation = driversLicenseTranslation;
    }
}

// For simplicity, this just has a single member representing the Customer
public class UAEDriversLicense {
    public Customer driver;

    public UAEDriversLicense(Customer driver) {
        this.driver = driver;
    }
}
```

Appendix B: Python Source

```
# in a file called assignment1.py
# Python 2.7 version
# dates are Python datetime.date instances

class AbstractPrimaryDocument(object):
    def __init__(self, first_name, last_name, nationality, gender,
                 date_of_birth, expiry_date):
        self.first_name = first_name
        self.last_name = last_name
        self.nationality = nationality
        self.gender = gender
        self.date_of_birth = date_of_birth
        self.expiry_date = expiry_date

class Emirates_ID(AbstractPrimaryDocument):
    def __init__(self, first_name, last_name, nationality, gender,
                 date_of_birth, expiry_date, id_number):
        super(Emirates_ID,self).__init__(first_name, last_name, nationality,
                                         gender, date_of_birth, expiry_date)
        self.id_number = id_number

class Drivers_License(AbstractPrimaryDocument):
    def __init__(self, first_name, last_name, nationality, gender,
                 date_of_birth, expiry_date):
        super(Drivers_License,self).__init__(first_name, last_name, nationality,
                                             gender, date_of_birth, expiry_date)

class Passport(AbstractPrimaryDocument):
    def __init__(self, first_name, last_name, nationality, gender,
                 date_of_birth, expiry_date):
        super(Passport,self).__init__(first_name, last_name, nationality,
                                      gender, date_of_birth, expiry_date)

from datetime import date, timedelta # for date math in EyeTest

class EyeTest(AbstractPrimaryDocument):
    def __init__(self, first_name, last_name, nationality, gender,
                 date_of_birth, expiry_date):
        # Eye Tests expire after 30 days
        if expiry_date is None:
            super(EyeTest,self).__init__(first_name, last_name, nationality,
```

```
                                        gender, date_of_birth,
                                        date.today() + timedelta(days=30))
        else:
            super(EyeTest,self).__init__(first_name, last_name, nationality,
                                        gender, date_of_birth, expiry_date)


class Drivers_License_Translation(object):
    def __init__(self,drivers_license):
        self.drivers_license = drivers_license

class Customer(object):
    def __init__(self, emirates_id, drivers_license, passport, eye_test,
                 drivers_license_translation):
        self.emirates_id = emirates_id
        self.drivers_license = drivers_license
        self.passport = passport
        self.eye_test = eye_test
        self.drivers_license_translation = drivers_license_translation

# For simplicity, this just has a single member representing the Customer
class UAE_Drivers_License(object):
    def __init__(self,driver):
        self.driver = driver
```