Assignment 1 —Application Engineering

1. Buy a camera from Amazon.com

Object: user, Amazon, Bank

1.1

Class: user

Data: account, keyword, price, address, order

Behavior: login,search, purchase, review order, cancel

```
loginAmazon () {
        Amazon.login (account);
}
searchAmazon() {
        Amazon.search (keyword);
}
purchaseCamera(){
        if(CreditCard.cardAvailable () )
              Amazon.match (this.Data);
        else
              Error;
}
review(){
      System.out.print (this.order);
}
cancel(){
      Amazon.orderEnd ();
}
```

1.2

Class: Amazon

Data: accountList, camerInfo, productList, status

Behavior:

```
login (userAccount) {
      List<String> accountList;
      foreach (String account in accountList){
          if (useraccount == account)
                status = True;
      }
          else
                status = False;
}
match ( productInfo) {
    List <string> productList;
    foreach (product in productList) {
        if (userKeyword in productInfo)
            system.out.print(results.product);
      else
            system.out.print(noResultsMatched);
}
orderComplete() {
    alert(User);
    User.order = null;
}
```

1.3

Class : CreditCard

Data: cardType, bankName, cardNumber, cardholder, address, securityCode, expirationDate

Behavior:

```
cardAvailable () {
      if (currentDate < expirationDate)
          return True;
      else
          return False;
}
```

2. Design a platform for buying tickets of local events

object: user, platform, creditCard

2.1

Class: user

Data: name, eventKeyword

Behavior: log in, search, order

```
loginPlatform () {
    platform.login (name);
}
search () {
    search.event (time, location)
}
order () {
    if (creditCard.cardAvailable() )
              platform.match (this.Data);
    else
              Error;
review () {
    system.out.print(this.order);
}
```

```
cancel () {
    platform.orderEnd()；
}
```

2.2
Class: platform
Data: status
Behavior:

```
login (userName) {
    list <String> nameList;
    foreach (String name in nameList) {
            if (username == name)
                status = True;
            else
                status = False;
    }
}
search () {
    List<String> eventList;
        foreach(String event in eventList) {
            if (userKeyword in eventInfo)
                system.out.print (“results.event.userKeyword”);
            else
                system.out.print (“NoResultsMatched”);
        }
}

orderComplete(){
    alert (user);
    user.order = null;
}
```

2.3
Class: creditCard
Data: cardType, bankName, cardNumber, cardholder, address, securityCode, expirationDate
Behavior:

```
cardAvailable () {
      if (currentDate < expirationDate)
          return True;
      else
          return False;
}
```

3. Design an app to book a doctor’s appointment using your medical insurance provider
object: patient, app, doctor, insurance company
3.1
Class: patient
Data: name, birthdate, insuranceInfo
Behavior: log in, search doctor, book an appointment, view order, cancel order

```
loginApp () {
    App.login (patientName);
}
search () {
    App.search (doctorName);
}
bookAppointment () {
    if (user. insuranceInfo == insurance.userInfo)
        App.match (this. Data);
    else
        Error;
}
review () {
    system.out.print (this. Appointment);
}
cancel () {
    App.appointmentEnd ();
}
```

3.2
Class: app
Data: status
Behavior: login, searchResults, matchInsuranceInfo

```
login (patient.name) {
    List <String> nameList;
    foreach (String name in nameList) {
                 if (patientName == name)
                     status = True;
                 else
                     status = False;
    }
}
search (user. doctorName) {
    List <String> nameList;
    foreach ( String name in nameList) {
                 if (doctorName == name)
                     system. out. print (“ doctorName, doctorTime”);
                 else
```

```
                system.out.print (“NoResultsMatched”);
    }
}
```

3.3
Class: doctor
Data: name, experience, clinic, appointment time,
Behavior: login, comfirmAppointment

```
loginApp () {
    App.login (doctorName);
}
comfirmAppointment () {
    system.out.print (patientAppointment);
    if (confirm)
        return True;
    else
        return False;
}
```

3.4
Class: insurance company
Data: patientName, patientBirthdate, patientInsuranceInfo, doctorInfo
Behavior: verifyInsuranceInfo, Copay

```
verify (patientName) {
    list<String> nameList;
    foreach (String name in nameList) {
        if ( patientName == name)
            return True;
        else
            return False;
}
Copay () {
    system.out.print (patientInsuranceInfo);
    return patientCopy;
}
```

4. Design a job searching platform
object: employee, platform, employer
4.1
Class: employee
Data: name, education, workExperience, expectedCareer, employeeResume
Behavior: login, search, applyJob

```
loginPlatform () {
    platform.login (name);
}
search () {
    platform.search (keyword);
}
applyJobs () {
    platform.fill (employee.name)
}
```

4.2
Class: platform
Data: status
Behavior: matchInfo

```
login (employee.name) {
    List <String> nameList;
    foreach (String name in nameList) {
                 if (employeeName == name)
                     status = True;
                 else
                     status = False;
    }
}
search (keyword) {
    List <String> jobList;
    foreach ( String job in jobList) {
                 if (keyword == job)
                     system. out. print (“ job, jobDescription ”);
                 else
                     system.out.print (“NoResultsMatched”);
    }
}
```

4.3
Class: employer
Data: name, positions, JobDescription, salary
Behavior: login, postJobs, selectEmployees, sendInvitations

```
loginPlatform() {
    platform.login (employerName);
}
postJobs() {
    employer. upload( jobs);
}
```

5. Order Pizza from Dominos

object: customer, Dominos, bank

5.1

Class: customer

Data: name, phone, address, creditCard, order

Behavior: login, search, order, reviewOrder, cancelOrder

```
loginDominos() {
    Dominos.login(name);
}
search() {
    Dominos.search(keyword);
}
order() {
    if( creditCard.cardAvailable())
        Dominos.match(this.Data);
    else
        Error;
}
cancel() {
    Dominos.orderEnd();
}
```

5.2

Class: Dominos

Data: status

Behavior: confirmOrder, deliverPizza,

```
login(customerName) {
    List<String> nameList;
    foreach (String name in nameList) {
          if (customerName == name)
              status = True;
          else
              status = False;
    }
}
search(keyword) {
    List<String> pizzaList;
    foreach (String pizza in pizzaList) {
        if (keyword == pizza)
            system.out.print("pizza");
        else
            system.out.print("NoResultsMatched");
        }
}
```

```
confirmOrder(customerAddress) {
    system.out.print (customerAddress);
    if (confirm)
          return True;
    else
          return False;
}
```

5.3

Class : CreditCard

Data: cardType, bankName, cardNumber, cardholder, address, securityCode, expirationDate

Behavior:

```
cardAvailable () {
      if (currentDate < expirationDate)
          return True;
      else
          return False;
}
```