EPHEC
Louvain-la-Neuve

Nom : .....

Prénom : .....

Classe : 2TL1 - 2TL2

Interrogation de janvier 2014

Langage avancé de programmation

Professeur : V. Van den Schrieck

Date : 6/1/2014 **Matériel autorisé** : -

Nombre de pages : 13 **Durée** : 120 minutes

Répondre aux questions directement sur ce document.

Acquis d'apprentissage évalués

- Utiliser le vocabulaire adéquat pour expliquer et discuter les concepts orientés-objets
- Analyser une situation en l'exprimant dans un formalisme orienté-objet (en l'occurrence, UML)
- Développer une solution logicielle dans le langage Java sur base de spécifications détaillées (textuelles ou UML)
- Utiliser une méthodologie, des outils et des techniques de collaboration afin de mener à bien un projet de développement java en équipe

Pondération

	Points pour la partie	Points de l'étudiant
UML	5	
POO	13	
Documentation et tests	6	
Programmation réseau	5	
Programmation concurrente	3	
Total	**32**	
Total	**20**	

Contexte

Le programme sur lequel l'examen se base représente un logiciel de réservation de vols. Un vol possède un nombre limité de places réservables par des clients. Un même client ne peut pas réserver plus d'une seule place.
Vous trouverez en fin du feuillet les classes qui composent ce programme. Commencez par les lire attentivement, avant de répondre aux questions. Les classes relatives aux exceptions ne sont pas fournies, mais vous pouvez considérer qu'elles sont disponibles.

1 UML (/5)

Question 1 [/5] : Dessinez le diagramme UML de l'application de réservation de vols. Dans ce diagramme doivent figurer aussi bien les classes dont le code vous est fourni à la fin de ce document, que les classes de l'API Java qui y sont explicitement mentionnées. Si vous avez besoin de notations particulières (ex : communication réseau), ajoutez une note indiquant la notation choisie.

Programmation orientée-objet (/13)

Question 2 [/2] : Utilisation d'objets

Complétez la méthode main de la classe Vol, afin de créer un vol Paris-Bruxelles de 2 places. Ajouter les passagers listés ci-dessous. Expliquez ce qu'il se passe lors de l'ajout du dernier passager. Ecrivez votre code de telle sorte qu'un message d'erreur adéquat s'affiche.

Liste des passagers avec leur numéro d'identité :

- Robert Baratheon (2837)
- Ned Stark (9021)
- Tyrion Lannister (9101)

```
public static void main(String [] args){




}
```

Question 3 [/7] : Héritage et structures de données

La compagnie aérienne souhaite développer un programme de fidélisation de ses clients. Pour cela, elle souhaiterait associer à chacun un nombre de points, initiallement égal à zéro. Lors de l'impression des informations sur un vol, le solde de points de chaque client doit également être affiché. Un passager gagne un point à chaque fois qu'il réserve un vol avec succès. Lorsqu'il atteint le quota de vingt points, le serveur de réservation envoie un message au client lui indiquant qu'il a gagné un vol, et réinitialise le crédit de points du passager. Le serveur de réservation doit donc avoir un accès efficace à l'ensemble des passagers ayant déjà voyagé.
Il vous est demandé d'expliquer comment vous modifieriez le programme existant pour ajouter cette fonctionnalité, en exploitant au maximum les possibilités de l'orienté-objet pour minimiser les changements. Pour cela, répondez à chacune des sous-questions ci-dessous. Le code de ces modifications n'est pas demandé, mais il peut être utile pour votre réflexion de l'ébaucher sur feuille de brouillon, au minimum sous forme de pseudo-code.

1. Expliquez le principe général de vos modifications et dessinez le diagramme UML de l'application modifiée (uniquement classes, attributs et relations, pas les noms de méthodes).

2. Si votre solution implique la création de nouvelles classes, donnez les signatures et documentez (brièvement) leurs méthodes

3. Pour chacune des classes existantes (Personne, Vol, ReservationServer et ReservationClient), indiquez s'il y a des modifications à effectuer, et pourquoi. S'il y en a, listez-les.

4. Justifiez en quoi votre solution est efficace, c'est-à-dire minimise les modifications demandées

5. Le serveur de réservation doit garder trace de tous les passagers ayant effectué une réservation dans une structure de données. Quelles sont les besoins qui vont guider le choix de cette structure de données? Une fois ces besoins définis, proposez une structure de données adéquate parmi l'ensemble des Collections proposées dans l'API Java.

Question 4 [/2] : Polymorphisme
Dans la classe ReservationClient, la méthode main() effectue deux réservations. La seconde réservation ne peut avoir lieu parce qu'il s'agit d'un doublon. A quel niveau le doublon est-il détecté? Donnez l'instruction correspondante. Quelle méthode (au sens java) cette instruction va-t-elle exploiter? Expliquez le principe qui se cache derrière ce mécanisme.

```
    public static void main(String[] args) {
        serverIP="127.0.0.1";
        port = 12345;
        try {
            System.out.println(
                reservation("Bxl-Paris", "Van den Schrieck", "Virginie", 12345));
            System.out.println(
                reservation("Bxl-Paris", "Van den Schrieck", "Virginie", 12345));
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

Question 5 [/2] : Exceptions
Toujours dans le cas de la gestion de cette deuxième réservation dans la méthode main de la classe ReservationClient, retracez le fil d'exécution de cet appel, en mentionnant toutes les méthodes qui sont parcourues et le traitement d'éventuelles exceptions.

Documentation et tests (/6)

Question 6 [/3] : Documentation

Documentez la méthode ajoutePassager(Personne p) de la classe Vol.

```
/**
  *
  *
  *
  *
  *
  *
  *
  *
  *
  *
  *
  *
  */
    public void ajoutePassager(Personne p)
                throws VolCompletException, DejaEnregistreException{
        if(listePassagers.contains(p)){
            throw new DejaEnregistreException(
                "Le passager "+ p + " est deja enregistre sur le vol");
        }
        if(this.estComplet()){
            throw new VolCompletException("Plus de place sur le vol" + this);
        }
        listePassagers.add(p);
    }
```

Question 7 [/3] : Tests

Expliquez en quoi consistent les tests Black-Box, et quelle est la différence avec les tests White-Box. Proposez ensuite un ensemble de tests Black-Box pour la méthode ajoutePassager de la classe Vol, sous forme d'un **tableau valeurs d'entrée (paramètres ou état de l'objet) - résultat attendu** (pas de code java). Basez-vous sur votre réponse à la question précédente.

Programmation réseau (/5)

Question 8 [/2] : Principes

L'application de réservation de vols est une application distribuée utilisant des sockets. Pourquoi a-t-on choisi cette solution plutôt que des DatagramSocket? Expliquez la différence entre les deux, et expliquez pour quoi le premier choix est préférable dans cette situation.

Question 9 [/3] : Fonctionnement

Expliquez le fonctionnement du client de réservation, en complétant les 5 points de commentaires.

```
/**
 * Cette methode effectue une reservation de vol sur le serveur de reservation
 * @param vol le nom du vol
 * @param nom le nom du passager
 * @param prenom le prenom du passger
 * @param numID le numero d'identite du passager
 * @return le resultat de la reservation
 * @throws IOException en cas d'erreur lors de la communication
 */
public static String reservation(String vol, String nom, String prenom, int numID)
                    throws IOException{
    //1.
    //
    Socket socket = new Socket(serverIP, port);
    //2.
    //
    BufferedReader in = new BufferedReader(
                            new InputStreamReader(
                                    socket.getInputStream()));
    PrintWriter out = new PrintWriter(
                        new BufferedWriter(
                                new OutputStreamWriter(
                                        socket.getOutputStream())),true);
    //3.
    //
    out.println(vol);
    out.println(nom);
    out.println(prenom);
    out.println(numID);
    //4.
    //
    String result = in.readLine();
    //5.
    //
    in.close();
    out.close();
    socket.close();
    return result;
}
```

Programmation concurrente (/3)

Question 10 [/3] : Parallélisation du serveur Web

En l'état, le serveur de réservation traite les demandes de réservation l'une après l'autre. On souhaite augmenter l'efficacité du serveur en parallélisant le traitement des requêtes de réservation.

1. Expliquez comment on peut simplement mettre en place une telle parallélisation.
2. La programmation concurrente peut poser certains problèmes d'intégrité des données si elle est appliquée sans précaution. Expliquez ici quelles données pourraient potentiellement poser problème. Que faire pour arranger ça? Indiquez précisément la modification à appliquer, et à quel(s) endroit(s) du programme.

Classes

Classe Personne

```java
package reservation_vols;

public class Personne {
    private String nom;
    private String prenom;
    private int numId;

    /**
     * @param id est un entier positif
     * Creation d'une nouvelle personne
     */
    public Personne(String nom, String prenom, int numId){
        this.nom = nom;
        this.prenom = prenom;
        this.numId = numId;
    }
    /**
     * @return une repr sentation textuelle de la personne
     */
    public String toString(){
        return this.nom + " " +this.prenom + " - " + this.numId;
    }
    /*
     * L'equivalence entre deux personnes s'effectue sur base
     * du numero d'identite
     * @see java.lang.Object#equals(java.lang.Object)
     */
    public boolean equals(Object o){
        if(this == o) return true;
        if ( !(o instanceof Personne) ) return false;
        return  ((Personne)o).numId== this.numId;
    }
}
```

Classe Vol

```java
package reservation_vols;
import java.util.ArrayList;
import java.util.List;
public class Vol {
    private String ligne;
    private List<Personne> listePassagers;
    private int numPlaces;
    /**
     * Constructeur avec arguments
     * @param ligne : le nom de la ligne aerienne deservie par le vol
     * @params numPlaces : le nombre de places disponibles sur le vol, >0.
     */
    public Vol(String ligne, int numPlaces){
        this.listePassagers = new ArrayList<Personne>();
        this.numPlaces = numPlaces;
        this.ligne = ligne;
    }
    /**
     * A completer - Question 6
     */
    public void ajoutePassager(Personne p)
                throws VolCompletException, DejaEnregistreException{
        if(listePassagers.contains(p)){
            throw new DejaEnregistreException(
                    "Le passager "+ p + " est deja enregistre sur le vol");
        }
        if(this.estComplet()){
            throw new VolCompletException("Plus de place sur le vol" + this);
        }
        listePassagers.add(p);
    }
    /**
     * @return une representation sous forme de chaine de caractere du vol,
     * avec l'etat de remplissage du vol et le nom des passagers enregistres
     */
    public String toString(){
        String result = "=== Vol "+ ligne + " ===\n";
        result += ("\ttotal : \t" + this.numPlaces + " places\n");
        result += ("\treservees : \t" + this.listePassagers.size() + " places\n");
        result += ("\tListe des passagers : \n");
        for(Personne p : listePassagers)
            result += ("- "+ p + "\n");
        return result;
    }
    /**
     * Deux vols sont equivalent si le nom de la ligne est identique
     */
    public boolean  equals(Object o){
        if(this == o) return true;
        if ( !(o instanceof Vol) ) return false;
        return ((Vol)o).ligne == this.ligne;
    }
    /**
     * @return true si le vol est complet et qu'il n'y a plus de reservation possible
     */
    public boolean estComplet() {
        return listePassagers.size()==this.numPlaces;
    }
}
```

Classe ReservationServer

```
package reservation_vols;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class ReservationServer {
    private static Map<String, Vol> vols;
    private static boolean volsComplets(){
        for(Vol v : vols.values()){
            if(!v.estComplet())
                return false; //un vol incomplet trouve
        }
        return true;
    }
    public static void main(String [] args) throws IOException{
        //Creation des vols reservables
        vols =new HashMap<String,Vol>();
        vols.put("Bxl-Paris", new Vol("Bxl-Paris", 4));
        vols.put("Bxl-Londres", new Vol("Bxl-Londres", 10));
        vols.put("Paris-NY", new Vol("Paris-NY", 15));
        ServerSocket s=null;
        Socket soc;
        s = new ServerSocket(12345);

        while(!volsComplets()){
            //Arrivee d'une requete :
            // Ouverture de la connexion et des flux de communication
            soc = s.accept();
            BufferedReader in = new BufferedReader(
                    new InputStreamReader(soc.getInputStream()));
            PrintWriter out = new PrintWriter(
                    new BufferedWriter(
                            new OutputStreamWriter(soc.getOutputStream())), true);
            //Traitement de la requete
            String vol = in.readLine();
            String nomPass = in.readLine();
            String prenomPass = in.readLine();
            int numIDPass = Integer.parseInt(in.readLine());
            //Tentative de reservation
            if(vols.containsKey(vol)){
                try {
                    vols.get(vol).ajoutePassager(
                                new Personne(nomPass, prenomPass, numIDPass));
                    out.println("Reservation effectuee avec succes");
                } catch (VolCompletException e) {
                    out.println("Erreur : Plus de place sur le vol "+vol);
                } catch (DejaEnregistreException e) {
                    out.println("Erreur : Vous etes deja enregistre sur ce vol");
                }
            }
            else {
                out.println("Erreur : Le vol demande n'existe pas\n");
            }

        }
        s.close();

    }

}
```

Classe ReservationClient

```java
package reservation_vols;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.Socket;

public class ReservationClient {
    static String serverIP;
    static int port;
    /**
     * Cette methode effectue une reservation de vol sur le serveur de reservation
     * @param vol le nom du vol
     * @param nom le nom du passager
     * @param prenom le prenom du passger
     * @param numID le numero d'identite du passager, >0
     * @return le resultat de la reservation
     * @throws IOException en cas d'erreur lors de la communication
     */
    public static String reservation(String vol, String nom, String prenom, int numID)
                                                throws IOException{
        //1.
        //
        Socket socket = new Socket(serverIP, port);
        //2.
        //
        BufferedReader in = new BufferedReader(
                                new InputStreamReader(
                                        socket.getInputStream()));
        PrintWriter out = new PrintWriter(
                                new BufferedWriter(
                                        new OutputStreamWriter(
                                                socket.getOutputStream())),true);
        //3.
        //
        out.println(vol);
        out.println(nom);
        out.println(prenom);
        out.println(numID);
        //4.
        //
        String result = in.readLine();
        //5.
        //
        in.close();
        out.close();
        socket.close();
        return result;

    }
    public static void main(String[] args) {
        serverIP="127.0.0.1";
        port = 12345;
        try {
            System.out.println(
                    reservation("Bxl-Paris", "Van den Schrieck", "Virginie", 12345));
            System.out.println(
                    reservation("Bxl-Paris", "Van den Schrieck", "Virginie", 12345));
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```