Examen de Programmation Java – ING1 EISTI 2009-2010

L'architecture (nom des répertoires) de votre projet Java est imposée de la manière suivante :

rendujava2 : répertoire de rendu

- **javasources** : sources java (**/*.java)
- **made** : partie générée automatiquement
  - **class** : bytecode (**/*.class)
  - **bin** : exécutable (*.jar)
- **build.xml**

Merci de nettoyer le répertoire de rendu en fin d'examen (laisser uniquement sources Java et fichier ant).

La javadoc de l'API 1.6 est disponible dans le répertoire **/usr/share/doc/sun-java6-jdk/html**

Pour accéder à la version la plus récente d'Eclipse, tapez eclipse-3.6

Rappel du barème des pénalités (à ramener au prorata de chaque exercice) :

- code non compilable : -10 (moitié de la note)
- warning (annotation @deprecated interdite) : -2 (un 10ème)
- norme de programmation Java non respectée : -5 (un quart)
- pas de commentaires : -5 (un quart) (**)
- non respect des consignes (noms de classe/package, fichiers déposés) : -2

NB : (**) le temps imparti étant relativement court, votre capacité à commenter votre code sera évaluée uniquement sur la classe Flight (exercice 2). Vous êtes dispensé (exceptionnellement) des commentaires habituels pour les autres classes.

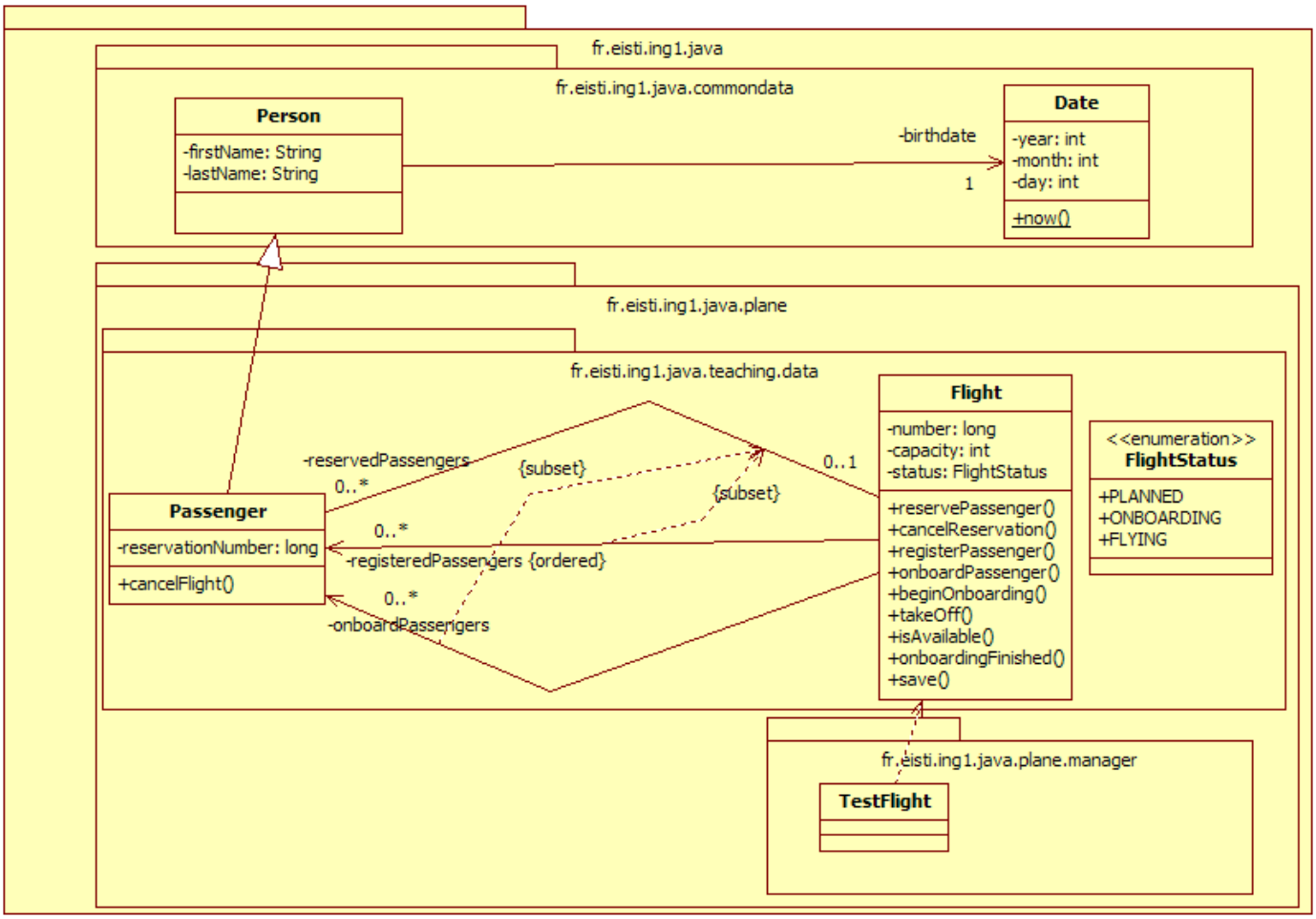
PJ : vous trouverez à la racine de votre compte exam-manager :

- le sujet en PDF
- le fichier source de la classe fr.eisti.ing1.java.util.Date à placer dans votre projet (toute modification du fichier est interdite) ;

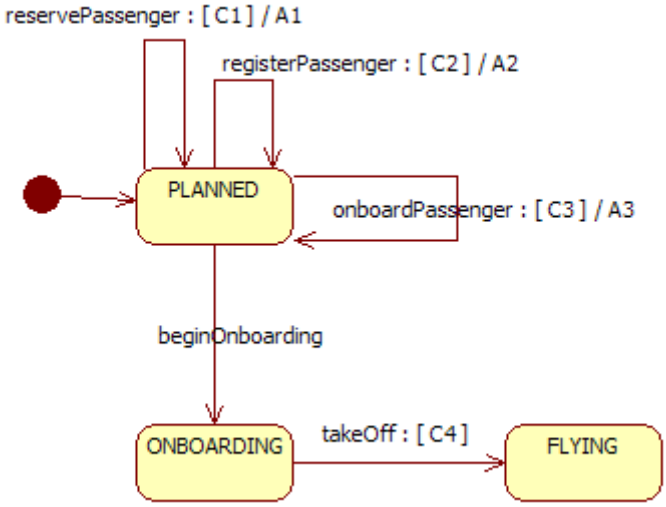
Introduction

Vous devez réaliser une application qui permet de suivre les passagers d'un vol depuis la réservation jusqu'au décollage. Les diagrammes UML suivant vous permettent d'avoir un aperçu global du système.

1. Diagramme de classe :
   - package fr.eisti.ing1.java.commondata
     - classe Date : pour manipuler simplement les dates usuelles ; **classe fournie** avec méthodes standard equals, compareTo et toString ; attention à ne pas confondre avec les classes de l'API Java java.util.Date et java.sql.Date ;
     - classe Person : une personne avec prénom, nom et date de naissance
   - package fr.eisti.ing1.java.plane.data
     - classe Passenger : spécialise la classe Person en ajoutant une numéro de réservation et associé à un vol de type Flight
     - classe Flight : représente un vol avec son numéro, sa capacité maximale et un ensemble de méthodes et de collections pour gérer les passagers sur les phases de réservation, d'enregistrement et d'embarquement ;
     - énumération FlightStatus : pour représenter les 3 phases d'un vol (cf diagramme d'état)
   - package fr.eisti.ing1.java.plane.manager
     - classe TestFlight : une petite application pour tester l'ensemble.



2. Diagramme d'état de la classe Flight



- Passenger reservePassenger(Person p) :
  C1 : reservedPassengers->size()<capacity
  A1 : crée un nouveau passager avec un nouveau numéro de réservation et associé au vol courant ; ajoute le nouveau passager à l'ensemble des réservations ; retourne le passager créé
- int registerPassenger(Passenger p) :
  C2 : p ∈ reservedPassengers et p ∉ registeredPassengers
  A2 : ajoute le passager à la liste des passagers enregistrés en lui assignant une place (n° dans la liste par

exemple) ; renvoie le numéro de place assigné

- void onboardPassenger(Passenger p) :
  C3 : p ∈ registeredPassengers
  A3 : ajoute le passager à l'ensemble des passagers embarqués
- void takeOff() :
  C4 : tous les passagers enregistrés ont embarqués

Exercice 1 – Classe Person

Ecrire la classe Person décrite ci-dessus avec :

- un constructeur complet et un deuxième sans paramètre de date de naissance (prend la date d'aujourd'hui comme date initiale) ;
- 3 accesseurs en lecture ;
- redéfinir les méthodes equals (sur les 3 attributs) et toString (modèle : prénom nom (date naissance)) ;

Exercice 2 – Classe Passenger

Ecrire la classe Passenger avec :

- un constructeur complet (4 propriétés + vol associé) avec un accès package private (~ en UML) ; préciser un commentaire l'intérêt de ce type d'accès ;
- un constructeur prenant en paramètre une personne, un numéro de réservation et un vol associé (toujours avec accès package private) ;
- des accesseurs en lecture pour les 5 caractéristiques ;
- une méthode cancelFlight pour enlever le vol associé ;
- redéfinir les méthodes equals (3 propriétés de Person et numéro de réservation) et toString (modèle : prénom nom (date naissance) #numéro réservation
- implémentant l'interface Comparable : définir la comparaison avec un passager sur les 4 mêmes propriétés qu'equals avec priorité au nom puis prénom puis date de naissance puis numéro de réservation.

Exercice 3 – Classe Flight

Ecrire la classe Flight avec tous ses commentaires ainsi que :

- conforme aux diagrammes UML ci-dessus ;
- un constructeur avec numéro et capacité en paramètres et initialisant les collections permettant de gérer les passagers ; le status initial est PLANNED ;
- deux accesseurs en lecture pour le numéro et la capacité ;
- une méthode toString avec le modèle : flight#800#cap100#res:85#reg:83#onb:70#PLANNED (les n° représentent respectivement le n° du vol, le nombre de réservations, d'enregistrements et de personnes à bord) ;
- les différentes méthodes de gestion des réservations décrites ci-dessus ainsi que les méthodes permettant de passer d'un état à l'autre ;
- gestion de la sécurité : cette application étant critique, toutes les méthodes n'étant pas appelée dans le bon status ou avec une condition non respectée devront déclencher une exception à définir et à ajouter au package fr.eisti.ing1.java.plane.data ;
- 3 méthodes permettant d'obtenir les listings des 3 collections en copie (toujours par soucis de sécurité) :
  - listingReservation(boolean sorted) : renvoie les passagers ayant réservé, triés suivant leur ordre naturel si le paramètre sorted est à vrai sinon tels ;
  - listingRegistered() : renvoie la liste des passagers enregistrés dans cet ordre (ordre des places assises) ;
  - listingOnboard(boolean sorted) : même chose que le 1er listing mais pour les passagers à bord

- une méthode save(String filename) pour sauver dans un fichier objet le vol ainsi constitué avec ses passagers.

NB : penser à utiliser les outils à votre disposition du framework des collections (tri, inversion de liste, etc…)

Exercice 4 – Application TestFlight

Ecrire l'application TestFlight qui :

- prend en paramètre en ligne de commande le nom d'un fichier de sauvegarde ;
- créé un tableau de 10 personnes ;
- un vol de 9 places ;
- affiche le vol ;
- essaie de réserver une place sur le vol pour les 10 personnes ;
- affiche le vol ;
- enregistre 8/9 passagers ;
- affiche le vol ;
- embarque 7/8 passagers ;
- affiche le vol ;
- essaie de décoller ;
- embarque le dernier passager enregistré et décolle ;
- affiche le vol et le sauve dans le fichier passé en paramètre.

Les problèmes soulevés exprès devront être immédiatement attrapés et signalés sur la console ; les autres (qui ne doivent donc pas arriver) seront captés en fin de programme avec affichage sur la console.

Exercice 5 – Gestion du projet Java et exécutable

Ecrire un fichier **build.xml** qui permet de :

- compiler l'ensemble de vos sources compilables (si vous avez une classe incomplète, l'exclure des fichiers à compiler) ;
- fabriquer un jar exécutable pour l'application TestFlight : **testFlight.jar**
- effacer le bytecode
- effacer tout ce qui est construit

NB : Votre fichier de configuration devra être en adéquation avec l'architecture de fichiers demandée en introduction.