Examen du cours de Programmation et Algorithmique II
1ère Session, Juin 2015
Partie 1

NOM : . . . . . PRENOM : . . . . . SECTION : . . . . . .

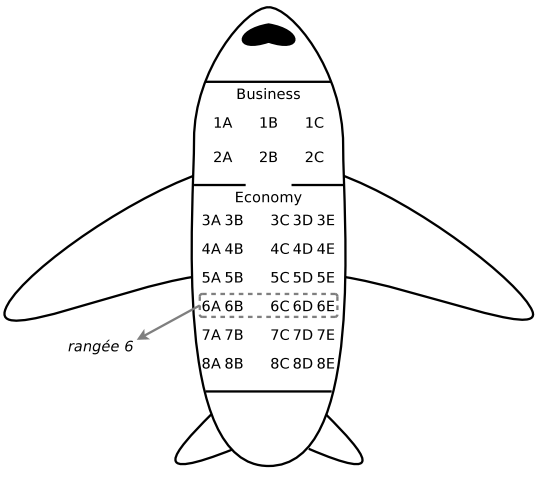
Consignes à lire impérativement !

L'examen est composé de **2 parties**. Chaque partie dure **2 heures**. Il vous est demandé de respecter les consignes suivantes.

- Commencez par écrire vos **nom**, **prénom** et **section** (math, info, ...) sur chaque feuille, y compris les feuilles de brouillon.
- Laissez vos calculatrice, téléphone portable et notes de cours dans votre sac. Leur usage n'est **pas autorisé**. Pensez à éteindre votre téléphone portable !
- Faites attention à la clarté et à l'organisation de vos réponses. Respectez les règles grammaticales et orthographiques.
- Utilisez pour vos réponses les **cadres** prévus à cet effet. Si davantage d'espace est nécessaire, utilisez le dos de la feuille ou une feuille supplémentaire et indiquez clairement où se situe le restant de la réponse.
- Vous devez terminer cette partie de l'examen avant de pouvoir sortir de la salle (pour aller à la toilette par exemple).
- Toutes les feuilles (énoncé et brouillon) doivent être remises en fin d'examen.
- Vérifiez que vous avez répondu à toutes les questions (il y a **2 questions** dans cette partie).

Question 1 – Réservation de sièges dans un avion (/7)

Cette question s'intéresse au système de réservation de sièges dans un avion. Ce paragraphe introduit d'abord l'organisation des sièges dans un avion. Les sièges sont organisés en **rangées**. Chaque rangée comprend un certain nombre de sièges. Chaque rangée est assignée à une **catégorie de voyageurs** : 1ère classe, *business*, *economy*, etc. Le nombre de rangées varie d'une catégorie à l'autre. Le nombre de sièges par rangée peut être différent d'une catégorie à l'autre mais il est identique au sein d'une catégorie. La figure ci-dessous illustre cette organisation. L'avion illustré ne comporte que les catégories *business* et *economy*. La catégorie *business* offre 2 rangées de 3 sièges, alors que la catégorie *economy* offre 6 rangées de 5 sièges, pour un total de 36 places.



Examen du cours de Programmation et Algorithmique II

1ère Session, Juin 2015

Partie 1

NOM : ..... PRENOM : ..... SECTION : .....

La modélisation en Java de la réservation des sièges repose sur trois classes. La classe `Passenger` modélise un passager. Elle définit simplement le nom de celui-ci. La classe `TravelClass` modélise une catégorie de voyageurs (par exemple *business*). Elle définit le nom de la catégorie. Elle comporte également l'instanciation de 3 catégories de passagers. La classe `Airplane` modélise les sièges, leur organisation et l'assignation des sièges aux passagers. Des extraits de ces classes sont donnés ci-dessous.

```
public class Passenger {
  public final name;
  ...
}
```

```
public class TravelClass {
  public static final TravelClass FIRST   = new TravelClass("first");
  public static final TravelClass BUSINESS= new TravelClass("business");
  public static final TravelClass ECONOMY = new TravelClass("economy");
  public final name;
  private TravelClass(...) {
    ...
  }
}
```

```
public class Airplane {

  /* Tous les sieges */
  public final Passenger [][] allSeats;

  /* Les sieges par categorie de voyageur */
  public final HashMap<TravelClass, Passenger[][]> travelClassSeats;

  /* Retourne le nombre de sieges libres */
  public int numFreeSeats();

  /* Verifie et convertit un identifiant de siege */
  private static int [] parseSeat(String seat) throws Exception;

  /* Determine s'il s'agit d'un siege cote fenetre */
  public boolean isWindowSeat(String seat) throws Exception;

  /* Affecte au passager un siege dans la categorie donnee */
  public boolean bookSeat(TravelClass tc, Passenger p) throws Exception;

  /* Affecte au passager un siege en donnant son identifiant */
  public boolean bookSeat(String seat, Passenger p) throws Exception;

  /* Retourne un iterateur pour les sieges */
  public Iterator<String> seatIterator();

}
```

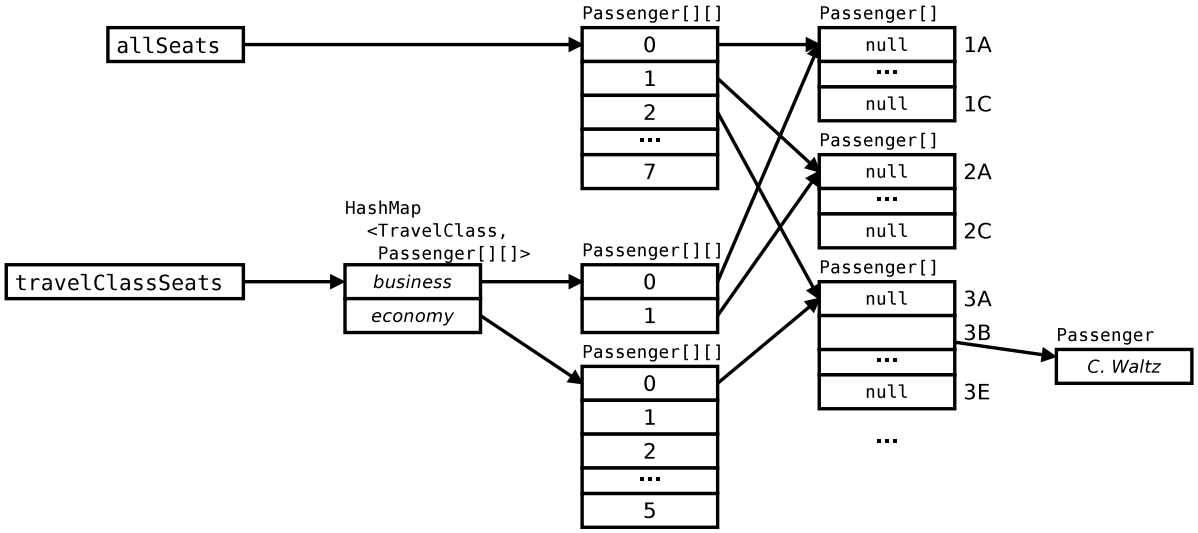
Examen du cours de Programmation et Algorithmique II

1ère Session, Juin 2015

Partie 1

NOM : ..... PRENOM : ..... SECTION : .....

Dans la classe `AirPlane`, l'ensemble des sièges est modélisé avec un tableau de tableaux de `Passenger`. Il s'agit de la variable d'instance `allSeats`. Le tableau externe représente l'ensemble des rangées de sièges. Chaque tableau interne modélise une rangée de sièges. Les tableaux internes peuvent avoir des longueurs différentes, en fonction du nombre de sièges par rangée. Ceci est illustré sur la figure ci-dessous, qui correspond à l'avion donné en exemple plus tôt. On observe dans l'exemple que `allSeats[0]` référence un tableau de 3 sièges alors que `allSeats[2]` référence un tableau de 5 sièges.



Afin de modéliser la répartition des sièges entre catégories de voyageurs, une table de hachage est utilisée. Il s'agit de la variable d'instance `travelClassSeats`. Les clés de cette table sont des catégories de voyageurs (instances de `TravelClass`). Les valeurs de la table sont les sièges de cette catégorie, i.e. des tableaux de tableaux de `Passenger`. Il est fait usage d'*aliasing* de sorte qu'un même siège puisse être retrouvé à partir de la variable `allSeats` ou de la variable `travelClassSeats`. On observe dans l'exemple de la figure ci-dessus que le siège `3B` peut être trouvé via `allSeat[2][1]` ou via `travelClassSeats.get(TravelClass.ECONOMY)[0][1]`.

Chaque siège est modélisé comme une référence vers une instance de `Passenger`. Si la référence est `null`, le siège n'est pas assigné. Sinon, il est assigné au passager représenté par l'instance de `Passenger`. Dans l'exemple, seul le siège `3B` est assigné (au passager C. WALTZ).

Adin de désigner un siège, un identifiant sous forme d'une chaîne de caractères est utilisé. Par exemple, `23F` ou `3B` sont des identifiants de sièges. Le format de cet identifiant est défini comme suit. Un identifiant est composé d'une suite non nulle de chiffres suivie d'une lettre. Les premiers caractères spécifient la rangée du siège tandis que le dernier caractère spécifie le siège de la rangée. On considère qu'une rangée ne peut contenir plus de 26 sièges. L'identifiant `23F` désigne le 6ème siège de la rangée 23 alors que `3B` désigne le 2ème siège de la rangée 3.

Examen du cours de Programmation et Algorithmique II
1ère Session, Juin 2015
Partie 1

NOM : ..... PRENOM : ..... SECTION : .....

Il vous est demandé de :

Q1a Implémenter la méthode `numFreeSeats` qui retourne le nombre de sièges libres de l'avion.

Q1b Implémenter la méthode privée `parseSeat(String)`. Cette méthode vérifie un numéro de siège spécifié sous forme d'une chaîne de caractères. Si la chaîne de caractères ne respecte pas le format, une exception est générée. Sinon, elle est convertie en un couple (numéro de rangée, numéro de siège) qui est retourné dans cet ordre dans un tableau de 2 entiers. Ces numéros permettent de retrouver un siège à partir du tableau `allSeats`. Attention, les numéros de rangée et siège commencent à 0.

Q1c Implémenter la méthode `isWindowSeat(String)` qui retourne `true` si le siège est situé côté fenêtre et `false` sinon.

Q1d Implémenter la méthode `bookSeat(TravelClass, Passenger)` qui assigne un siège libre d'une catégorie à un passager. Si la catégorie n'existe pas, une exception est générée. Si aucun siège n'est libre dans cette catégorie, la valeur `false` est retournée. Dans le cas contraire, la valeur `true` est retournée et le passager est assigné au siège.

Q1e Implémenter la méthode `bookSeat(String, Passenger)` qui assigne un siège spécifique à un passager. Si le siège n'existe pas, une exception est générée. Si le siège spécifié n'est pas libre, la valeur `false` est retournée. Dans le cas contraire, la valeur `true` est retournée et le passager est assigné au siège.

Q1f Implémenter la méthode `seatIterator()` qui retourne un itérateur `Iterator<String>` pour les sièges. L'itérateur retourne les identifiants des sièges, sous forme de chaîne de caractères (p.ex. `3B` ou `23F`). L'itérateur doit être implémenté sous la forme d'une classe anonyme.

Référence API java. La table ci-dessous contient la description de classes et méthodes de l'API Java qui peuvent être utiles à l'implémentation.

Table de hachage `HashMap<K,V>`	
`V get(K key)`	retourne la valeur associée à la clé `key` si elle existe. Retourne `null` si la clé n'existe pas.
`void put(K key, V value)`	associe la valeur `value` à la clé `key`. Si une valeur était déjà associée à la clé `key`, cette valeur est remplacée par la nouvelle.
`boolean containsKey(K key)`	retourne `true` si la clé `key` existe, `false` sinon.
`Set<K> keySet()`	retourne l'ensemble des clés. Un ensemble est itérable.

Itérateur `Iterator<E>`	
`boolean hasNext()`	retourne `true` s'il y a encore un élément à retourner, `false` sinon. Attention, un appel à `hasNext()` devrait laisser l'état de l'itérateur inchangé.
`E next()`	retourne l'élément suivant s'il existe et fait avancer l'itérateur. Le comportement de `next` est indéfini s'il n'y a pas d'élément suivant.

Examen du cours de Programmation et Algorithmique II
1ère Session, Juin 2015
Partie 1

NOM : . . . . . PRENOM : . . . . . SECTION : . . . . . .

Q1a **(méthode `numFreeSeats()`)**

Q1b **(méthode `parseSeat(String)`)**

Examen du cours de Programmation et Algorithmique II
1ère Session, Juin 2015
Partie 1

NOM : . . . . . PRENOM : . . . . . SECTION : . . . . . .

Q1c **(méthode `isWindowSeat(String)`)**

Q1d **(méthode `bookSeat(TravelClass, Passenger)`)**

Examen du cours de Programmation et Algorithmique II
1ère Session, Juin 2015
Partie 1

NOM : . . . . . PRENOM : . . . . . SECTION : . . . . . .

Q1e **(méthode `bookSeat(String, Passenger)`)**

Examen du cours de Programmation et Algorithmique II
1ère Session, Juin 2015
Partie 1

NOM : . . . . . PRENOM : . . . . . SECTION : . . . . . .

Q1f **(méthode `seatIterator()`)**

Examen du cours de Programmation et Algorithmique II
1ère Session, Juin 2015
Partie 1

NOM : ..... PRENOM : ..... SECTION : .....

Question 2 – Student smoke test (/3)

Cette question s'intéresse au *binding* de variables et de méthodes en mixant leur utilisation dans un enchevêtrement de classes. Vous disposez ci-dessous de 4 classes `A`, `B`, `C1` et `C2` qui contiennent un certain nombre de variables et de méthodes. Vous disposez également d'un court programme utilisant ces classes.

```
public class A {
  public int x= 10, y=11;
  public int getX() {
    return x;
  }
  public void setY(int y) {
    this.y= y;
  }
}

public class B extends A {
  public static int x= 255;
  public void plop(int _x) {
    x= _x;
    setY(x);
  }
}
```

```
public class C1 extends B {
  public void setY(int y) {
    this.y= y;
  }
}

public class C2 extends B {
  public byte x= (byte) super.x;
  public int y= 20;
  public int getX() {
    return x;
  }
  public void plop(int x) {
    x= x * 2;
    super.plop(x);
  }
  public void setY(int y) {
    super.y= y;
  }
}
```

Il vous est demandé : de fournir les 8 valeurs affichées par le programme suivant. Pour chacune des valeurs veuillez donner une brève explication ; par exemple quel(s) *binding*(s) sont réalisés. Une réponse sans justification sera considérée comme incorrecte.

```
A a= new C2();
B b= new B();
C1 c1= new C1();
C2 c2= (C2) a;
System.out.println(a.x);                                    /*  Q2a */
System.out.println(c1.x);                                   /*  Q2b */
System.out.println(a.getX());                               /*  Q2c */
System.out.println(c1.getX());                              /*  Q2d */
b.x+= 10;
c1.x+= 10;
System.out.println(b.x);                                    /*  Q2e */
c2.plop(200);
c1.plop(300);
System.out.println(a.y);                                    /*  Q2f */
System.out.println(c1.y);                                   /*  Q2g */
System.out.println(c2.y);                                   /*  Q2h */
```

Examen du cours de Programmation et Algorithmique II
1ère Session, Juin 2015
Partie 1

NOM : . . . . . PRENOM : . . . . . SECTION : . . . . . .

Q2a **(valeur et justification)**

. . . . . .

Q2b **(valeur et justification)**

. . . . . .

Q2c **(valeur et justification)**

. . . . . .

Q2d **(valeur et justification)**

. . . . . .

Q2e **(valeur et justification)**

. . . . . .

Q2f **(valeur et justification)**

. . . . . .

Q2g **(valeur et justification)**

. . . . . .

Q2h **(valeur et justification)**

. . . . . .

Examen du cours de Programmation et Algorithmique II
1ère Session, Juin 2015
Partie 2

NOM : ..... PRENOM : ..... SECTION : .....

Consignes à lire impérativement !

L'examen est composé de **2 parties**. Chaque partie dure **2 heures**. Il vous est demandé de respecter les consignes suivantes.

- Commencez par écrire vos **nom**, **prénom** et **section** (math, info, ...) sur chaque feuille, y compris les feuilles de brouillon.
- Laissez vos calculatrice, téléphone portable et notes de cours dans votre sac. Leur usage n'est **pas autorisé**. Pensez à éteindre votre téléphone portable !
- Faites attention à la clarté et à l'organisation de vos réponses. Respectez les règles grammaticales et orthographiques.
- Utilisez pour vos réponses les **cadres** prévus à cet effet. Si davantage d'espace est nécessaire, utilisez le dos de la feuille ou une feuille supplémentaire et indiquez clairement où se situe le restant de la réponse.
- Vous devez terminer cette partie de l'examen avant de pouvoir sortir de la salle (pour aller à la toilette par exemple).
- Toutes les feuilles (énoncé et brouillon) doivent être remises en fin d'examen.
- Vérifiez que vous avez répondu à toutes les questions (il y a **2 questions** dans cette partie).

Question 1 – Comptage de cellules (/4)

Soit un tableau de `boolean`s partitionné en deux. La partie de gauche ne contient que des cellules de valeur `false` tandis que celle de droite ne contient que des cellules de valeur `true`. Un exemple d'un tel tableau est donné ci-dessous. Dans ce tableau, la partie de gauche contient cinq cellules de valeur `false` tandis que la partie de droite contient deux cellules de valeur `true`.

false	false	false	false	false	true	true

L'objectif de cette question est de concevoir un algorithme qui compte le nombre de cellules qui valent `true`. Comme cet algorithme est destiné à être appliqué sur des tableaux de très grande taille (p.ex. $2 * 10^9$ éléments), il doit être conçu de manière à **minimiser le nombre d'itérations**.

Il vous est demandé : d'écrire une méthode `countTrue` qui compte le nombre de cellules de valeur `true` d'un tableau. La signature de la méthode est donnée ci-dessous. L'argument `tab` est le tableau dont il faut compter les cellules de valeur `true`. La méthode retourne le nombre de cellules de valeur `true`. Pour le tableau donné en exemple ci-dessus, le résultat de la méthode doit être 2.

```
int countTrue(boolean [] tab);
```

ATTENTION ! Un algorithme dont le nombre d'itérations varie linéairement avec la taille du tableau ne sera pas considéré comme suffisant pour réussir cette question !

Examen du cours de Programmation et Algorithmique II
1ère Session, Juin 2015
Partie 2

NOM : . . . . . PRENOM : . . . . . SECTION : . . . . . .

Q1 **(méthode `countTrue`)**

Examen du cours de Programmation et Algorithmique II
1ère Session, Juin 2015
Partie 2

NOM : ..... PRENOM : ..... SECTION : .....

Question 2 – Fraction irréductible (/6)

L'objectif de cette question est la conception d'un algorithme de simplification de fractions. Soit p et q deux nombres naturels non nuls. Simplifier la fraction $\frac{p}{q}$ consiste à déterminer p' et q' naturels premiers entre eux tels que $\frac{p}{q} = \frac{p'}{q'}$. On nomme $\frac{p'}{q'}$ fraction irréductible car il n'est pas possible de la simplifier, p' et q' n'ayant pas de diviseur commun.

Il existe plusieurs algorithmes pour simplifier une fraction. Un moyen classique consiste à déterminer le PGCD du numérateur et du dénominateur à l'aide de l'algorithme d'Euclide. Dans cet exercice, une autre approche est imposée. Le numérateur et le dénominateur sont d'abord décomposés en facteurs premiers. Les facteurs communs sont ensuite éliminés. La décomposition en facteurs premiers du nombre 45 est $3 \times 3 \times 5$.

Une fraction est modélisée par la classe `Fraction` dont un extrait est montré ci-dessous. La classe modélise essentiellement les numérateur et dénominateur. Il s'agit d'une classe dont les instances sont immuables.

```
public class Fraction {

  public final int p; /* numerateur */
  public final int q; /* denominateur */

  public Fraction(int p, int q) {
    this.p= p;
    this.q= q;
  }

  /* Calcule la fraction irreductible correspondante */
  public Fraction reduce();

  /* Decompose n en facteurs premiers. Ces derniers sont retournes par
     ordre croissant dans une instance d'ArrayList */
  private static ArrayList<Integer> factorize(int n);

}
```

Il vous est demandé : d'écrire un algorithme qui effectue au maximum la simplification d'une fraction. Le problème est découpé en deux étapes.

Q2a Premièrement, écrivez la méthode de classe privée `factorize` qui effectue la décomposition de n en facteurs premiers. Ces facteurs sont retournés en ordre croissant dans une instance d'`ArrayList`. La signature de cette méthode est donnée dans l'extrait de la classe `Fraction` ci-dessus.

Q2b Deuxièmement, écrivez une méthode `reduce` qui effectue la simplification d'une fraction. Le résultat est retourné sous la forme d'une nouvelle instance de `Fraction`. La signature de cette méthode est donnée dans l'extrait de la classe `Fraction` ci-dessus.

Examen du cours de Programmation et Algorithmique II
1ère Session, Juin 2015
Partie 2

NOM : . . . . . PRENOM : . . . . . SECTION : . . . . . .

Q2a **(méthode `factorize`)**

Examen du cours de Programmation et Algorithmique II
1ère Session, Juin 2015
Partie 2

NOM : . . . . . PRENOM : . . . . . SECTION : . . . . . .

Q2b **(méthode `reduce`)**