Initiation à la programmation Java
IP2 - Séance No 7

Yan Jurski

16 mars 2021

Révisions

Plan

- Partiel d'IP2 mars 2016
- Partiel d'IP2 mars 2015
- A noter : AUCUN DOCUMENT AUTORISES

Révisions - Partiel mars 2016

- Un patient est représenté par son nom, son prénom, un identifiant entier unique et un autre entier signalant la gravité de son cas (de 0 : urgence absolue, à 10 : pas grave)

Fichier InterfacePatient.java

```
int id(); // identifiant du patient ( entier unique )
String toString(); // une presentation avec nom, prenom et identifiant
    du patient
void setGravite( int gravite); // si le paramètre n'est pas entre 0 et
    10, le fixer à 10
int getGravite();
```

- Exercice 1 : (4,5 points) Écrire la classe Patient (attributs, constructeurs et méthodes). Il y aura deux constructeurs : un qui prend deux chaînes de caractères pour fixer le nom et le prénom ; l'autre avec un argument de plus pour fixer la gravité si elle est connue au départ

Révisions - Partiel mars 2016

Fichier Patient.java

```
public class Patient implements InterfacePatient{
 private final String nom;
 private final String prenom;
 public final int id; // non modifiables, mais pour id public est bien
 private int gravité;
 private static int nbPatient=0;
 public Patient (String n, String p){
  this(n,p,10); // tant qu'à faire ...
 }
 public Patient (String n,String p, int g){
  this.nom=n;
  this.prenom=p;
  this.gravité=g;
  this.id=Patient.nbPatient++; // garantit l'unicité
 }
}
```

Fichier Patient.java

```
public class Patient implements InterfacePatient{
 private String nom;
 private String prenom;
 private final int id;
 private int gravité;
 private static int nbPatient=0;
 public int id() { return id; }
 public String toString(); {
  return nom+" "+prenom+" de numéro "+ id + " a niveau :"+gravité;
 }
 public void setGravite( int gravité); {
  if (gravité <0 || gravité >10) this.gravité = 10;
  else this.gravité=gravité;
 }
 public int getGravite() { return gravité;}
}
```

Fichier Patient.java

```
public class Patient implements InterfacePatient{
 private String nom;
 private String prenom;
 private final int id;
 private int gravité;
 private static int nbPatient=0;
 public int id() { return id; }
 public String toString(); {
  return nom+" "+prenom+" de numéro "+ id + " a niveau :"+gravité;
 }
 public void setGravite( int gravité); {
  if (gravité <0 || gravité >10) this.gravité = 10;
  else this.gravité=gravité;
 }
 public int getGravite() { return gravité;}
}
```

Remarque : **4.5 points** était généreux. Faites BIEN ce que vous savez faire.

Révisions - Partiel mars 2016

- la salle d'attente (classe Urgence) est une liste chaînée
- chaque nouveau patient est ajouté en fonction de sa gravité : les éléments restent rangés dans l'ordre croissant des gravités.

Interface SalleAttente.java

```
int nbPatients();
int nbUrgencesAbsolues(); // ceux en urgence absolue (ie gravite =0)
void listePatients(); // affiche la liste des patients
void listeParGravite(int gravite); // affiche les patients concernés
void ajouterNouvPatient(Patient p); // en respectant l'ordre
boolean suppressionPatient(int id); // true si le patient existait
```

- Exercice 2 : (3 points) Donner les attributs et les constructeurs des classes Urgence et Cellule . Il s'agit de décider quels constructeurs écrire pour que ces deux classes soient utilisables et qu'il soit possible d'écrire toutes les méthodes données pour Urgence

Révisions - Partiel mars 2016

Fichier Urgence.java

```
public class Urgence{
 private Cellule first=null;
 public Urgence(){first=null;} // facultatif, constructeur par défaut
}
```

Fichier Cellule.java

```
public class Cellule{
 private Patient content;
 private Cellule next;
 public Cellule(Patient x){this(x,null);}
 public Cellule(Patient x, Cellule s){ content=x; next=s;}
 public Patient getPatient() {return content;}
 public Patient getNext() {return next;}
}
```

- Les 4 premières méthodes de l'interface se réduisent à un parcours
- Pour ajouter : les constructeurs `Cellule(...)`
- Pour suppression (en tête) : `getPatient` et `getNext();`

Ex. 3 (2 points) Écrire nbPatients nbUrgencesAbsolues

Fichier Urgence.java

```
public int nbPatients(){
 if (first==null) return 0; else return first.nbPatients();
}
public int nbUrgencesAbsolues(){
 if (first==null) return 0; else return first.nbUrgencesAbsolues();
}
```

Fichier Cellule.java

```
public int nbPatients(){ // une en version récursive
 if (next==null) return 1;
 else return 1+next.nbPatients();
}
public int nbUrgencesAbsolues(){ // l'autre en itératif
 int rep=0;
 for(Cellule aux=this; aux!=null; aux=aux.next)
  if ( aux.content.getGravite()==0 ) rep++;
 return rep; // remarquez aussi que si la liste est triée,
} // on aurait pu arrêter plus tôt
```

Ex 4 (2 points) Écrire listePatients et listeParGravite

Fichier Urgence.java

```
public void listePatients(){
 if (first==null) return; else first.listePatients();
}
public void listeParGravite(int g){
 if (first==null) return; else first.listeParGravite(g);
}
```

Fichier Cellule.java (encore deux parcours simples)

```
public void listePatients(){
 Cellule aux=this;
 while (aux !=null) { // avec un while pour changer
  System.out.println(aux.content.toString());
  aux=aux.next;
 }
}
public void listeParGravite(int g){ // en recursif
 if (content.getGravite()==g) System.out.println(content.toString());
 if (next!=null) next.listeParGravite(g);
}
```

Ex 5 (1,5 points) Écrire suppressionPatient

Fichier Urgence.java

```
public boolean suppressionPatient(int id){
 if (first==null) return false;
 if (first.getPatient().id==id) {
  first=first.getNext(); return true;
 }
 return first.suppressionPatient(id);
}
```

Fichier Cellule.java

```
// invariant : content n'est pas de numéro id
public boolean suppressionPatient(int id){
 if (next==null) return false;
 if (next.content.id==id) {
  next=next.next; return true;
 }
 return next.suppressionPatient(id);
}
```

Ex 6 (2 points) Écrire ajouterNouvPatient

ordre de gravité croissant, et en "fin de file"

Fichier Urgence.java

```
public void ajouterNouvPatient(Patient p){
 if ((first==null) || (first.getPatient().getGravite()>p.getGravite()))
     first=new Cellule(p,first);
 else first.ajouterNouvPatient(p);
}
```

Fichier Cellule.java

```
// invariant : le patient dans content doit précéder p
public void ajouterNouvPatient(Patient p){
 if ( (next==null) || (next.content.getGravite() > p.getGravite()))
     next=new Cellule(p,next);
 else next.ajouterNouvPatient(p);
}
```

Ex 7

- On gère à présent l'attribution des patients à une équipe médicale.
- Elle est constituée d'un nombre *nb_medecin* supposé constant de médecins (parce qu'il y aurait un nombre fixe de salles de soins).
- Chaque médecin est simplement repéré par un numéro entre 0 et *nb_medecin* − 1, c.a.d qu'on choisit ici de ne pas écrire de classe propre aux médecins, mais de manipuler ce numéro.
- L'attribution d'un patient à un médecin se fait à l'aide d'un tableau : la case i du tableau contient une référence vers le patient traité par le médecin d'identifiant i.
- Pour modéliser cela, on impose de procéder ainsi : on ajoute des attributs à la classe Urgence : le nombre de salles, et le tableau de correspondance entre médecin et patient.

Ex 7 (1,5 points) attributs supplémentaires d'Urgence

- L'équipe médicale est associée à une `Urgence` ⇒ non statique
- Le nombre correspond à la taille du tableau ⇒ pour cette `Urgence`
- nécessité de préciser la taille de l'équipe à la construction
- l'équipe est supposée de taille fixe ⇒ final

Fichier Urgence.java

```
public class Urgence{
 private Cellule first=null;
 private final Patient[] tab; // tab est final, pas son contenu
 public Urgence(int nbMedecin){ // constructeur modifié
  first=null;
  tab=new Patient [nbMedecin];
 }
}
```

- Discussion : conceptuellement il aurait peut être été meilleur de créer une classe **SalleDeSoin** , mais prenons l'exercice tel qu'il est posé.

Ex 8. (1,5 points) écrire String toString(int noMedecin)

Présentation du patient traité par un médecin - Justifiez la classe où elle est écrite

Ex 8. (1,5 points) écrire String toString(int noMedecin)

Présentation du patient traité par un médecin - Justifiez la classe où elle est écrite

- Dans Urgence, c'est là où se trouvent les informations

Fichier Urgence.java

```
public class Urgence{
 private Cellule first=null;
 private final Patient[] tab;
 public String toString(int noMedecin){
  if (tab[noMedecin]==null) return ""+noMedecin+" est libre";
  else return ""+ noMedecin + " s'occupe de " +
       tab[noMedecin].toString();
 }
}
```

Ex 9 (2 points) - PatientGuéri - PatientOrienté

Deux situations peuvent se présenter :

- Un patient bien soigné rentre chez lui.
  Son médecin récupère immédiatement le premier patient de la liste d'attente.
- Ou bien le patient n'est pas guéri, il est orienté vers un autre service :
  - la gravité de son cas est remise à jour,
  - il retourne dans la salle d'attente,
  - son médecin récupère immédiatement le premier patient de la liste d'attente.

Ecrire deux méthodes pour décrire ces situations

Ex 9 (2 points) - PatientGuéri - PatientOrienté

Fichier Urgence.java

```
public class Urgence{
 private Cellule first=null;
 private final Patient[] tab;
 public void patientGuéri(int noMed){
  if ( (noMed<0) || (noMed>=tab.length) ) return; // méthode robuste
  tab[noMed] = prendreSuivant(); // à écrire
 }
 private Patient prendreSuivant(){
  if (first==null) return null;
  Partient rep=first.getPatient();
  this.suppressionPatient(rep.id);
  return rep;
 }
}
```

Ex 9 (2 points) - PatientGuéri - PatientOrienté

Fichier Urgence.java

```
public class Urgence{
 private Cellule first=null;
 private final Patient[] tab;
 // aux urgences le médecin oriente son patient en fixant un nouveau
     niveau de gravité
 public void patientOrienté(int noMed, int niveauUrgence){
 // tests de robustesse
  if ( (noMed<0) || (noMed>=tab.length) || tab[noMed]==null) return;
  tab[noMed].setGravite(niveauUrgence);
  this.ajouterNouvPatient(tab[noMed]);
  tab[noMed] = prendreSuivant();
 }
}
```

Ex 9 (2 points) - PatientGuéri - PatientOrienté

Fichier Urgence.java

```
public class Urgence{
 private Cellule first=null;
 private final Patient[] tab;
 // aux urgences le médecin oriente son patient en fixant un nouveau
     niveau de gravité
 public void patientOrienté(int noMed, int niveauUrgence){
 // tests de robustesse
  if ( (noMed<0) || (noMed>=tab.length) || tab[noMed]==null) return;
  tab[noMed].setGravite(niveauUrgence);
  this.ajouterNouvPatient(tab[noMed]);
  tab[noMed] = prendreSuivant();
 }
}
```

- Rq : on aurait pu utiliser le même nom : `patientTraité(int noMed)` et `patientTraité(int noMed, int niveauUrgence)` au lieu de `patientGuéri` et `PatientOrienté` car la signature lève l'ambiguïté.

Partiel mars 2016

Reprenez rapidement l'ensemble du sujet et estimez votre note ...

Autre correction de partiel : 2014-2015

- On y modélise un carnet de contacts inspiré de celui d'un téléphone
- On y ajoute des opérations classiques,
- On l'organise en plusieurs listes.

Partiel 2014-2015

- Un contact correspond aux coordonnées d'une personne, c. à d : nom, prénom, numéro de téléphone.
  On les modélise comme des instances de la classe `Contact`
- L'interface de cette classe est donnée par :

Fichier InterfaceContact.java

```
/** renvoie le nombre de contacts crées depuis le début */
static int nbCreatedContacts();
/** renvoie le nom , le prenom et le telephone du contact */
String toString();
/** change le numero de telephone du contact si son format est correct
    c.à.d 10 chiffres dont le premier est un 0 sinon ne fait rien */
void setPhone(String phone);
```

`Contact` possède deux constructeurs. L'un fonction du nom et du prénom ; l'autre prenant en plus son numéro de téléphone

Partiel 2014-2015

Ex1-2 (1+4 points) Ecrire contact

Fichier Contact.java (travail en cours)

```
public class Contact{
 private final String name, firstname;
 private String tel; // permet les 0 au début, et "indéfini"
 public Contact(String n, String p){ this(n,p,null); }
 public Contact(String n, String p, String t){
  name=n; firstname=p;
  tel=goodFormat(t);
 }
 private static String goodFormat(String t){
  if ( (t==null) || (t.length()!=10) ) return null;
  for (int i=0 ; i<t.length() ; i++) {
   char c=t.charAt(i);
   if ( (c<'0') || (c>'9') ) return null; // respecte code ASCII
  }
  return t.charAt(0)=='0'?t:null; // serait plus lisible avec un if ...
 }
}
```

Partiel 2014-2015

Ex1-2 (1+4 points) Ecrire contact - choix statiques ?

Discussion Interface

```
public static int nbCreatedContacts(); // depuis l'origine
public String toString(); // renvoie une description du contact
public void setPhone(String phone); // seulement si format correct
```

Fichier Contact.java

```
public class Contact{
 private final String name, firstname;
 private String tel;
 private static int nbTotal=0;
 public Contact(String n, String p){ this(n,p,null); }
 public Contact(String n, String p, String t){
  name=n; firstname=p; tel=goodFormat(t); nbTotal++;
 }
 public static int nbCreatedContact(){ return nbTotal; }
 public String toString(){ return name+" "+firstname+" tel :"+tel; }
 public void setPhone(String nb){tel=goodFormat(nb);} // déjà écrite
 public String getPhone(){ return tel; } // utiles plus tard
 public String getName(){ return name; }
}
```

Partiel 2014-2015

Interface

```
int nbContacts(); // de ce carnet d'adresse
int nbFrancilienContacts(); // dont le numero commence par 01
void ls(); // affiche la liste des contacts
void lsByName(String name); // idem, mais ceux d'un nom précis
void addNewContact(Contact c); // peu importe la place
boolean rmContact(String phone); // efface le 1er contact dont le
    téléphone est donné
boolean rmByName(String nom); // supprime tous les contacts de ce nom
```

- Exercice 3 (3 points) attributs et constructeurs de `ContactCell` et `AddressBook` ?
- justifiez les choix des constructeurs et accesseurs pour pouvoir écrire les méthodes demandées

Partiel 2014-2015

Fichier AdressBook.java

```
public class AdressBook{
 private ContactCell first=null;
 public AdressBook() { first=null;} // facultatif car implicite
}
```

Fichier ContactCell.java

```
puclic class ContactCell{
 private Contact content;
 private ContactCell next;
 public ContactCell(Contact c,ContactCell n){ content=c; next=n;}
 public ContactCell(Contact c){ this(c,null);}
 public Contact getContact(){ return content; }
 public ContactCell getNext(){ return next; }
}
```

- Les constructeurs public de `ContactCell` pour `addNewContact`
- Les accesseurs `getContact()`, `getNext()` pour la suppression (éventuellement en tête)

Ex 4 (2 points) Écrire nbContacts et nbFrancilienContacts

Fichier AdressBook.java

```
public int nbContact(){ if (first==null) return 0; else return
    first.nbContact(); }
public int nbFrancilienContact(){ if (first==null) return 0; else
    return first.nbFrancilienContact(); }
```

Fichier ContactCell.java

```
public int nbContact(){ // une version récursive ...
 if (next==null) return 1; else return 1+next.nbContact();
}
public int nbFrancilienContact(){ // une version itérative ...
 ContactCell aux=this;
 int rep=0;
 while(aux!=null){ // pensez à écrire getPhone dans Contact
  String tel=aux.content.getPhone();
  if (tel != null && tel.charAt(1)==1) rep++;
  aux=aux.next;
 }
 return rep;
}
```

Ex 5 (2 points) Écrire ls et lsByName

Fichier AdressBook.java

```
public void ls(){
 if (first==null) return; else first.ls();
}
public void lsByName(String n){
 if (first != null) first.lsByName(n);
}
```

Fichier ContactCell.java

```
public void ls(){
 System.out.println(content.toString());
 if (next!=null) next.ls();
}
public void lsByName(String n){
 for (ContactCell aux=this; aux!=null; aux=aux.next)
 // pensez à écrire getName() dans Contact
  if (content.getName().equals(n)) // penser à equals...
   System.out.println(content.toString());
}
```

Ex 6 (1,5 points) Écrire addNewContact

Fichier AdressBook.java

```
public void addNewContact(Contact c){
    first=new ContactCell(c,first);
}
```

- Il n'est pas précisé où on ajoute, autant choisir le plus simple

Ex 7. (2,5 points) Écrire rmContact

(num tel est supposé unique)

Fichier AdressBook.java

```
public boolean rmContact(String tel){
 if (first==null) return false;
 if (first.getContact().getPhone().equals(tel)) {
  first=first.getNext(); // s'assurer que les getters sont écrits
  return true;
 }
 return first.rmContact(tel);
}
```

Fichier ContactCell.java

```
// invariant : contact n'a pas ce numéro de tel
public boolean rmContact(String tel){
 if (next==null) return false; // version récursive
 if (! (next.content.getPhone().equals(tel)) )
    return next.rmContact(tel);
 next=next.next;
 return true;
}
```

Ex 7. (2,5 points) Écrire rmByName

Fichier AdressBook.java

```
public boolean rmByName(String n){
 if (first==null) return false;
 if (first.getContact().getName().equals(n)) {
  first=first.getNext();
  this.rmByName(n); // partie récursive sur AdressBook !
                    // en effet le nouveau first peut être concernée
  return true; // il y a eu au moins une suppression
 }
 return first.rmByName(n); // ne pas confondre :
                           // cet appel est pour ContactCell
}
```

Fichier ContactCell.java

```
public boolean rmByName(String n){ // this.content n'a pas ce nom
 boolean found=false;
 ContactCell aux=this; // invariant : aux.content n'a pas ce nom
 while (aux.next != null) { // version itérative
  if ( !(next.content.getName().equals(n)) ) aux=aux.next;
  else { next=next.next; found =true; }
 } return found;
}
```

Ex 8. (2 points) Stockage interne en tableau de listes

- Pour accélérer l'accès aux données, on décide de représenter en interne, dans le téléphone, l'ensemble des contacts sous la forme d'un tableau de taille 26, contenant des listes chaînées.
- Chaque élément de ce tableau est une liste de contacts qui ont en commun la même initiale de nom
- Proposez un type pour ce tableau :

Ex 8. (2 points) Stockage interne en tableau de listes

- Pour accélérer l'accès aux données, on décide de représenter en interne, dans le téléphone, l'ensemble des contacts sous la forme d'un tableau de taille 26, contenant des listes chaînées.
- Chaque élément de ce tableau est une liste de contacts qui ont en commun la même initiale de nom
- Proposez un type pour ce tableau :
  `AdressBook[]`

Ex 8. (2 points) Stockage interne en tableau de listes

- Pour accélérer l'accès aux données, on décide de représenter en interne, dans le téléphone, l'ensemble des contacts sous la forme d'un tableau de taille 26, contenant des listes chaînées.
- Chaque élément de ce tableau est une liste de contacts qui ont en commun la même initiale de nom
- Proposez un type pour ce tableau : `AdressBook[]`
- Ecrivez une méthode qui affiche tout les contacts d'un tel élément

Ex 8. (2 points) Stockage interne en tableau de listes

- Pour accélérer l'accès aux données, on décide de représenter en interne, dans le téléphone, l'ensemble des contacts sous la forme d'un tableau de taille 26, contenant des listes chaînées.
- Chaque élément de ce tableau est une liste de contacts qui ont en commun la même initiale de nom
- Proposez un type pour ce tableau : `AdressBook[]`
- Ecrivez une méthode qui affiche tout les contacts d'un tel élément

```
public static void afficheAll(AdressBook[] tab){
 for (int i=0;i<tab.length;i++)
 if (tab[i]!=null) tab[i].ls();
}
```

Ex 9. (2 points) lsByName

- Ecrivez une méthode `lsByName` qui travaille sur ce tableau et affiche l'ensemble des contacts ayant ce nom

```
public static void lsByName(AdressBook[] tab, String n){
 int i=n.charAt(0)-'A'; // on suppose n bien formé !
 if (tab[i]!=null) tab[i].lsByName(n);
}
```

Seul document autorisé : une feuille A4 recto-verso manuscrite (non photocopiée). Aucune machine.
Le barème est indicatif. Une question peut toujours être traitée en utilisant les précédentes (traitées ou non).
Les morceaux de code Java devront être clairement présentés, indentés et commentés.
L'utilisation de collections de l'API Java (en particulier `LinkedList` et `ArrayList`) est interdite.

- Tous les attributs d'instance de toutes les classes introduites doivent être privés.
- Il est possible d'écrire des méthodes non spécifiquement demandées si c'est plus pratique, en spécifiant pour chaque méthode dans quelle classe elle se trouve.

Au fast-food "Le gland de chêne", les sandwichs sont conçus à la demande : le client commande auprès du caissier, celui-ci se tourne alors vers les préparateurs situés derrière lui en cuisine, leur crie le nom du sandwich et encaisse le client. En cas de grande affluence ce mode de fonctionnement à l'ancienne a des ratés. Le gérant décide donc d'informatiser la situation et c'est ce que nous allons modéliser dans cette épreuve.

Sandwichs

Il y a 20 types de sandwichs, chacun étant identifié par un entier entre 0 et 19. Pour des raisons de comptabilité, on veut connaître à la fin de la journée le nombre de sandwichs de chaque type commandés.

Un sandwich sera donc représenté par une instance de la classe **`Sandwich`**. Cette classe contient (au moins) un attribut privé entier **`type`** qui représente le type d'un sandwich et un attribut de type tableau d'entiers **`commandes`** permettant de stocker le nombre de sandwichs de chaque type déjà commandés.

L'interface de cette classe est donnée par (sans que soit précisé le caractère statique ou non de ces méthodes) :

```
/** renvoie le type du sandwich */
int type();

/** renvoie le nombre de sandwichs commandés du type donné en argument */
int nbSandwichs(int type);

/** affiche pour chaque type (entier) de sandwich le nombre d'unités commandées */
```

d'informatiser la situation et c'est ce que nous allons modéliser dans cette épreuve.

Sandwichs

Il y a 20 types de sandwichs, chacun étant identifié par un entier entre 0 et 19. Pour des raisons de comptabilité, on veut connaître à la fin de la journée le nombre de sandwichs de chaque type commandés.

Un sandwich sera donc représenté par une instance de la classe **`Sandwich`**. Cette classe contient (au moins) un attribut privé entier **`type`** qui représente le type d'un sandwich et un attribut de type tableau d'entiers **`commandes`** permettant de stocker le nombre de sandwichs de chaque type déjà commandés.

L'interface de cette classe est donnée par (sans que soit précisé le caractère statique ou non de ces méthodes) :

```
/** renvoie le type du sandwich */
int type();

/** renvoie le nombre de sandwichs commandés du type donné en argument */
int nbSandwichs(int type);

/** affiche pour chaque type (entier) de sandwich le nombre d'unités commandées */
void recapitulatifQuotidien();

/** remet à zéro le nombre de sandwichs commandés, pour tous les types */
void debutDeJournee();
```

Cette classe possède en outre un constructeur qui prend en argument le type de sandwich (sous forme d'un entier).

Exercice 1. (4,5 points) Écrire la classe **`Sandwich`** (attributs, constructeurs et méthodes). Penser à spécifier ce qui est statique.

Optimiser les commandes

Dans cette partie, on modélise la liste des commandes sous forme d'une liste doublement chaînée : la liste est représentée par la classe **`APreparer`**, les cellules sont représentées par la classe **`UneCommande`**.

Lorsqu'un nouveau sandwich est commandé, une cellule de valeur ce sandwich (donc de type **`Sandwich`**) est ajoutée en fin de liste. Dès qu'un préparateur est libre, il récupère et prépare le sandwich en tête de liste.

L'interface de la classe **`APreparer`** contient au moins les méthodes suivantes :

```
/** renvoie le nombre de sandwichs en attente de préparation (ie la longueur de
```

```
int nbSandwichs(int type);

/** affiche pour chaque type (entier) de sandwich le nombre d'unités commandées */
void recapitulatifQuotidien();

/** remet à zéro le nombre de sandwichs commandés, pour tous les types */
void debutDeJournee();
```

Cette classe possède en outre un constructeur qui prend en argument le type de sandwich (sous forme d'un entier).

Exercice 1. (4,5 points) Écrire la classe **`Sandwich`** (attributs, constructeurs et méthodes). Penser à spécifier ce qui est statique.

Optimiser les commandes

Dans cette partie, on modélise la liste des commandes sous forme d'une liste doublement chaînée : la liste est représentée par la classe **`APreparer`**, les cellules sont représentées par la classe **`UneCommande`**.

Lorsqu'un nouveau sandwich est commandé, une cellule de valeur ce sandwich (donc de type **`Sandwich`**) est ajoutée en fin de liste. Dès qu'un préparateur est libre, il récupère et prépare le sandwich en tête de liste.

L'interface de la classe **`APreparer`** contient au moins les méthodes suivantes :

```
/** renvoie le nombre de sandwichs en attente de préparation (ie la longueur de
la liste) */
int nbSandwichs();

/** renvoie le nombre de sandwichs en attente de préparation du type donné en
argument */
int nbSandwichs(int type);
```

```
/** renvoie le type du prochain sandwich à préparer (tête de liste) et le
supprime de la liste */
int prochainAPreparer();

/** ajoute en fin de liste un nouveau sandwich dont le type est donné en
paramètre */
void nouvelleCommande(int type);
```

Exercice 2. (3 points) Donner les attributs et les constructeurs des classes **`APreparer`** et **`UneCommande`**. Il s'agit de décider quels constructeurs écrire pour que ces deux classes soient utilisables et qu'il soit possible d'écrire toutes les méthodes données dans le cadre ci-dessus pour **`APreparer`**.

Exercice 3. (2,5 points) Écrire les deux méthodes **`nbSandwichs`**. A-t-on le droit de leur donner le même nom ? (Justifier.)

Exercice 4. (2 points) Écrire la méthode **`prochainAPreparer`**.

Exercice 5. (2 points) Écrire la méthode **`nouvelleCommande`**.

Optimiser les préparations

Le gérant du "gland de chêne" ayant remarqué que préparer plusieurs sandwichs d'un même type en même temps est plus rapide que les préparer les uns après les autres, nous allons adapter notre modélisation à cette remarque, en ajoutant dans l'interface de la classe **`APreparer`** les méthodes suivantes :

```
/** renvoie un tableau d'entiers, la valeur de la case i correspondant au nombre
de sandwichs de type i à préparer, puis vide la liste */
int[] enAttente();

/** met à jour le tableau des sandwichs à préparer: pour chaque type de sandwich
on ajoute à ce qui est dans la case du tableau le nombre de sandwichs apparaissant
dans la liste chaînée; puis vide la liste */
void miseAJourAttente(int[] att);
```

(Justifier.)

Exercice 4. (2 points) Écrire la méthode **`prochainAPreparer`**.

Exercice 5. (2 points) Écrire la méthode **`nouvelleCommande`**.

Optimiser les préparations

Le gérant du "gland de chêne" ayant remarqué que préparer plusieurs sandwichs d'un même type en même temps est plus rapide que les préparer les uns après les autres, nous allons adapter notre modélisation à cette remarque, en ajoutant dans l'interface de la classe **`APreparer`** les méthodes suivantes :

```
/** renvoie un tableau d'entiers, la valeur de la case i correspondant au nombre
de sandwichs de type i à préparer, puis vide la liste */
int[] enAttente();

/** met à jour le tableau des sandwichs à préparer: pour chaque type de sandwich
on ajoute à ce qui est dans la case du tableau le nombre de sandwichs apparaissant
dans la liste chaînée; puis vide la liste */
void miseAJourAttente(int[] att);

/** met à jour le tableau des sandwichs à préparer en tenant compte de la largeur
du plan de travail: pour chaque type de sandwich on ajoute à ce qui est dans la
case du tableau le nombre de sandwichs apparaissant dans la liste chaînée sans
dépasser max; et on supprime de la liste les sandwichs correspondant */
void miseAJourAttenteMax(int[] att, int max);
```

Exercice 6. (3 points) Implémenter les méthodes **`enAttente`** et **`miseAJourAttente`**, en respectant les contraintes suivantes :

- chacune de ces méthodes ne parcourt qu'une fois la liste chaînée,
- une de ces méthodes fait un appel à l'autre (au choix).

Exercice 7. (3 points) Le plan de travail pour préparer les sandwichs n'étant pas extensible à l'infini, on se propose de limiter les reports à un nombre maximal de sandwichs : écrivez la méthode **`miseAJourAttenteMax`** qui met à jour le nombre de sandwichs de chaque type à préparer dans le tableau **`att`**, sans pouvoir dépasser la valeur **`max`**, et enlève les sandwichs ainsi comptabilisés de la liste des sanwichs (en enlevant en priorité ceux qui apparaissent d'abord dans la liste).