SCIENCES SORBONNE UNIVERSITÉ

LU2IN002-2023oct
Éléments de programmation
par objets avec Java

Examen du 18 décembre 2023 – Durée : 2 heures

Seul document autorisé : **feuille A4 manuscrite, recto-verso.**
Pas de calculatrice ou téléphone. Barème indicatif sur 43.

Partie 1 Exercices (18 points)

Exercice 1 (4 points) (Question des Quizzes) *Redéfinition complexe*

Soit les 5 classes suivantes :

```
1  public class X {}
2  public class Y extends X {}
3  public class Z extends Y {}
4  public class Mere {
5      public void methode(X x) { System.out.println("Mere : methode(X)"); }
6      public void methode(Y y) { System.out.println("Mere : methode(Y)"); }
7  }
8  public class Fille extends Mere {
9      public void methode(Y y) { System.out.println("Fille : methode(Y)"); }
10     public void methodeDeFille() { System.out.println("methode de Fille"); }
11 }
```

et soit les définitions suivantes :

```
12 Mere mf=new Fille(); X x=new X(); Y y=new Y(); Z z=new Z(); X xy=new Y();
```

Pour chaque appel de méthode ci-dessous, quel est le résultat ?

```
13 mf.methode( y );
14 mf.methode( z );
15 mf.methode( xy );
16 mf.methodeDeFille();
```

Total : 4pts

```
13 mf.methode(y) // --- rend -> Fille : methode(Y)
14 mf.methode(z) // --- rend -> Fille : methode(Y)
15 mf.methode(xy) // --- rend -> Mere : methode(X)
```

Pour la dernière instruction, cela ne compile pas :

```
1 TestExo.java:19: error: cannot find symbol
2         mf.methodeDeFille();
3           ^
4   symbol:   method methodeDeFille()
5   location: variable mf of type Mere
```

Exercice 2 (4 points) *Ordinateur et TNT*

Soit les classes suivantes :

```
1  public class Appareil {
2      public String toString() { return "Appareil"; }
3  }
4  public class Television extends Appareil {
5      public String toString() { return "Television"; }
6  }
7  public class TeleTNT extends Television { }
8  public class AppareilInformatique extends Appareil { }
9  public class Ordinateur extends AppareilInformatique { }
10 public class Test {
11     public static void main(String [] args) {
12         Ordinateur ordi=new Ordinateur();System.out.println(ordi.toString());
13         TeleTNT tvTNT=new TeleTNT();System.out.println(tvTNT.toString());
```

```
14
15          Appareil a1=new Ordinateur();System.out.println(a1.toString());
16          Appareil a2=new TeleTNT();System.out.println(a2.toString());
17
18          AppareilInformatique ai1=new TeleTNT();
19          AppareilInformatique ai2=new Ordinateur();
20          AppareilInformatique ai3=new Appareil();
21
22          AppareilInformatique ai4=ordi;
23          AppareilInformatique ai5=(AppareilInformatique)ordi;
24          AppareilInformatique ai6=a1;
25          AppareilInformatique ai7=(AppareilInformatique)a1;
26
27          Appareil ap1=new TeleTNT();
28          Appareil ap2=new Ordinateur();
29          Television tv1=(Television)ap1;
30          Television tv2=(Television)ap2;
31      } }
```

Q2.1 (a) Qu'affichent les lignes 12 et 13 ? Expliquer comment est choisie la méthode `toString()` appelée. (b) Quelle est la différence avec l'affichage des lignes 15 et 16 ? Expliquer.

Total : 1.5pts
(a) ligne 12 affiche `Appareil`, ligne 13 affiche `Television`
Explication : s'il existe une méthode `toString()` dans la classe de l'objet crée, c'est cette méthode qui est appelée, sinon la méthode appelée est la méthode `toString()` de la classe mère, sinon on remonte dans la hiérarchie d'héritage jusqu'à ce qu'une méthode `toString()` soit trouvée.

(b) La ligne 15 affiche la même chose que la ligne 12 (resp. la ligne 16 affiche la même chose que la ligne 13), car les objets crées sont de même type, même si le handle n'est pas le même. Or la méthode `toString()` appelée est celle du type de l'objet réel (l'objet qui a été créé par le `new`), et ne dépend pas du type du handle.

Q2.2 Parmi les lignes 18 à 20, lesquelles sont correctes, lesquelles sont fausses. Expliquez chaque erreur.

Total : 1pt
Ligne 18 fausse : une télévision TNT n'est pas un appareil informatique
Ligne 19 correcte
Ligne 20 fausse : un appareil n'est pas un appareil informatique

Q2.3 Parmi les lignes 22 à 25, lesquelles ne compilent pas ? Expliquez chaque erreur.

Total : 0.5pts
1 erreur : Echec à la compilation de la ligne 24, car un appareil n'est pas un appareil informatique
Les lignes 22 et 23 compilent, car un ordinateur est un appareil informatique.
La ligne 25 compile grâce au cast, pas d'erreur à l'exécution car `a1` référence un objet de type réel `Ordinateur` (voir ligne 15) qui est un appareil informatique.

Q2.4 Parmi les lignes 29 et 30, lesquelles ne compilent pas ? lesquelles provoquent une erreur lors de l'exécution du programme ? Expliquez chaque erreur.

Total : 1pt
Les deux lignes compilent. La ligne 30 provoque une erreur à l'exécution car l'objet référencé par `ap2` est de type réel `Ordinateur` (voir ligne 28). Or un `Ordinateur` n'est pas par héritage une `Television`. La référence `tv2` de type `Television` ne peut donc pas être utilisée pour atteindre un objet de type réel `Ordinateur`.

Exercice 3 (3 points) *Répétition de rendez-vous*
Soit les classes `Date`, `RendezVous` et `Test` suivantes :

```
1 public class Date {
2     private int jour, mois, annee;
```

```
3       public Date(int jour, int mois, int annee) {
4           this.jour=jour; this.mois=mois; this.annee=annee;
5       }
6       public String toString() {return jour+"/"+mois+"/"+annee;}
7   }
8   public class RendezVous {
9       private Date d;
10      private String description;
11      public RendezVous(Date d, String desc) {
12          this.d=d; description=desc;
13      }
14      public String toString() { return d.toString()+" : "+description; }
15  }
16  public class Test {
17      public static void main(String [] args) {
18          RendezVous rv=new RendezVous(new Date(19,12,2014),"Cours Java");
19      }
20  }
```

Dans un agenda, un même rendez-vous peut revenir régulièrement. Par exemple, le cours de Java a lieu toutes les semaines le même jour. Pour gérer cela, on propose d'utiliser des copies de rendez-vous par clonage ou par constructeur par copie. Pour simplifier, on veut créer une copie d'un rendez-vous qui soit décalée d'un jour (sans gérer le problème de dépassement (ie. si `jour=31` alors `jour+1=32` est accepté)).

Donner toutes les méthodes et instructions à ajouter aux classes fournies (y compris dans le `main`) pour pouvoir créer une copie de l'objet référencé par `rv` (ligne 18) en décalant sa date d'un jour.

Total : 3pts

Solution 1 : constructeur par copie

```
1   // Dans la classe Date
2   public Date(Date d) {
3       this.jour=d.jour+1;  this.mois=d.mois; this.annee=d.annee;
4   }
5   // Dans la classe RendezVous
6   public RendezVous(RendezVous rv) {
7       this(new Date(rv.d),rv.description);
8   }
9   // Dans le main
10  RendezVous copie=new RendezVous(rv);
```

Solution 2 : clonage

```
1   // Dans la classe Date
2   public Date clone() {
3       return new Date(jour+1,mois,annee);
4   }
5   // Dans la classe RendezVous
6   public RendezVous clone() {
7       return new RendezVous(d.clone(),description);
8   }
9   // Dans le main
10  RendezVous copie=rv.clone();
```

Exercice 4 (7 points) *Pâtissiers et macarons*

On considère les trois classes correctes suivantes :

```
1   public class Macaron {
2       public static final int POIDS_GROS_MACARON=75;
3       private static int cptMacarons=0;
4       public final int numero;
5       private int poids; // en gramme
6       public Macaron(int poids) {
7           this.poids=poids;
8           cptMacarons++;
9           numero=cptMacarons;
```

```
10      }
11      public Macaron() { this(60); }
12      public static int getCptMacarons() {
13          return cptMacarons ;
14      }
15  }
16  public class Patissier {
17      public Macaron produire() {
18          return new Macaron();
19      } }
20  public class Test {
21      public static void main(String [] a){
22          Patissier p1=new Patissier();
23          Macaron o1=p1.produire();
24          Macaron [] boite=new Macaron[4];
25          boite[0]=p1.produire();
26          boite[1]=boite[0];
27          boite[0]=null;
28          boite[2]=o1;
29          boite[3]=p1.produire();
30          boite[3]=null;
31      } }
```

Q4.1 On veut compléter la classe `Patissier`. Pour chacun des 4 membres (attribut (A) ou méthode (M)) ci-dessous, indiquer en expliquant si le membre est un membre d'instance (I) ou un membre statique (S) de la classe `Patissier`. Par exemple, `nom` (le nom du pâtissier) est un Attribut d'Instance (`AI`).

(a) `cptMacarons` : le nombre de macarons produits par le pâtissier
(b) `codeDessert` : le code du dessert pour l'inventaire de la pâtisserie
(c) `getCptPatissiers()` : retourne le nombre de pâtissiers créés depuis le début du programme
(d) `dormir()` : l'action de dormir et se reposer

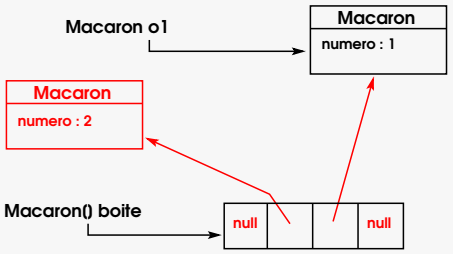
Total : 1pt

(a) `cptMacarons` AI car le nombre de macarons du pâtissier dépend de chaque pâtissier
(b) `codeDessert` AS car le code du dessert est commun à tous les pâtissiers
(c) `getCptPatissiers()` MS car le nombre de pâtissiers ne dépend pas du pâtissier en particulier
(d) `dormir()` MI car il faut un pâtissier pour dormir

Q4.2 (a) Combien d'instances de la classe `Macaron` ont été créées dans les lignes 26 à 34 ?
(b) Donner le diagramme mémoire après l'exécution de la ligne 34 en ne précisant sur le diagramme que le numéro du macaron.

Total : 2pts
Remarque : la correction a tenu compte du fait que pour (a) les lignes indiquées (26 à 34) pouvaient être remplacées par les lignes 22 à 30, et pour (b) la ligne demandée pouvait être remplacée par la ligne 30.

(a) 3 instances, créées aux lignes 23, 25 et 29
(b) Voir schéma. Le troisième objet (crée à la ligne 29 est supprimé par le ramasse-miette à cause de `boite[3]=null` et parce que plus aucune référence ne pointe vers lui.



Q4.3 A la fin de la méthode `main`, on ajoute l'instruction :

```
System.out.println(EXPRESSION);
```

où `EXPRESSION` est remplacée par chacune des expressions ci-dessous. Pour chacune des 10 expressions, écrire le numéro de la ligne, puis indiquer si l'instruction est correcte (OK) ou fausse (FAUX). Si elle est

fausse, justifier brièvement pourquoi (en un ou deux mots).

```
40 o1.POIDS_GROS_MACARON
41 o1.cptMacarons
42 o1.numero
43 o1.poids
44 o1.getCptMacarons()
```

```
50 Macaron.POIDS_GROS_MACARON
51 Macaron.cptMacarons
52 Macaron.numero
53 Macaron.poids
54 Macaron.getCptMacarons()
```

Total : 2pts

```
40 OK
41 FAUX car cptMacarons privé
42 OK
43 FAUX car  poids privé
44 OK
```

```
50 OK
51 FAUX car cptMacarons privé
52 FAUX car numero pas static
53 FAUX car poids privé et pas static
54 OK
```

Q4.4 On ajoute dans la classe `Macaron`, la méthode suivante :

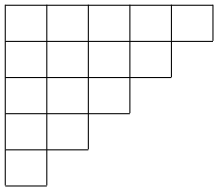
```
public static double mystere() {
        return 4*poids/5.0;
}
```

Cette méthode compile-t-elle ? Expliquer brièvement.

Total : 0.5pts

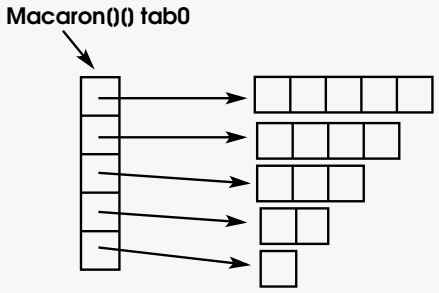
Non, cette méthode ne compile pas, car on ne peut pas utiliser une variable d'instance (ici, `poids`) dans une méthode statique. Remarque : le calcul effectué ne correspond à rien car cette question était seulement sur l'impossibilité d'utiliser des variables d'instance dans une méthode statique.

Q4.5 On veut créer une boite à macarons qui a la forme ci-contre. Donner la déclaration d'un tableau à deux dimensions permettant de stocker les macarons de cette boîte. Ensuite, initialiser toutes les cases avec un macaron. Remarque : utiliser obligatoirement des boucles.



Total : 1.5pts

```
Macaron [][] tabO=new Macaron[5][];
for(int i=0;i<tabO.length;i++) {
   tabO[i]=new Macaron[tabO.length-i];
   for(int j=0;j<tabO[i].length;j++)
        tabO[i][j]=new Macaron();
}
```



Partie 2 Problème (25 points)

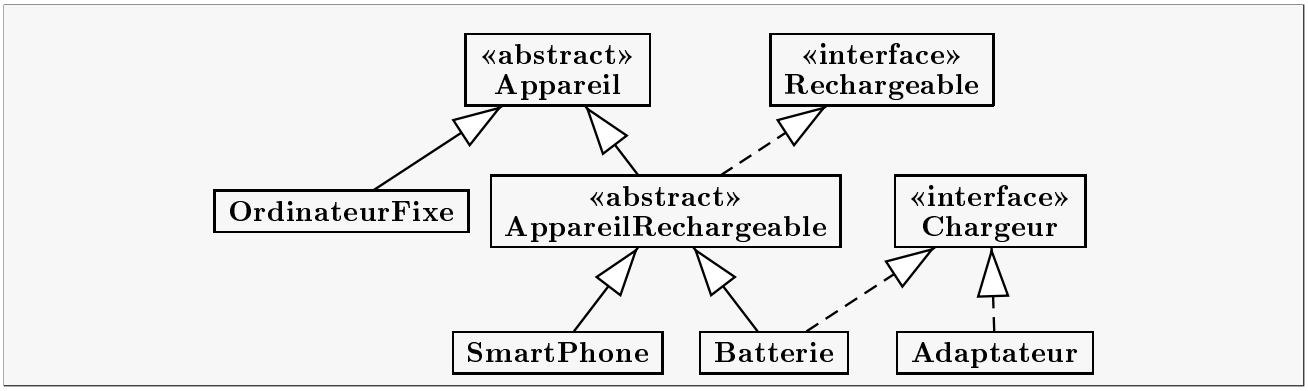
Remarques préliminaires : *la visibilité des variables et des méthodes à définir, ainsi que leurs aspects statique, ou final ne sont pas précisés dans le sujet, c'est à vous de décider de la meilleure solution à apporter. N'oubliez pas de définir les méthodes abstraites quand cela est possible. Ne pas écrire les méthodes `toString`, sauf si elles sont demandées.*

Attention : *certaines des idées de cet examen ont été simplifiées ou inventées pour les besoins de l'examen et ne correspondent pas à la réalité.*

On considère des appareils. Certains appareils (smartphones, batteries externes, ...) ont la propriété d'être rechargeable, et d'autres non (ordinateur fixe, ...). Divers dispositifs (adaptateurs secteur, batteries externes,...) peuvent servir pour charger ce qui est rechargeable. Les adaptateurs secteur ne sont pas des appareils.

Q5.1 (3 points) Donner le diagramme de classes UML contenant les classes `Adaptateur` (adaptateur

secteur), `Appareil`, `AppareilRechargeable`, `Batterie` (batterie externe), `OrdinateurFixe`, `SmartPhone` ainsi que les interfaces `Rechargeable` et `Chargeur`. Ne pas écrire les attributs ni les méthodes. Précisez la (ou les) classe(s) abstraite(s).



Interface `Rechargeable`. Soit l'interface `Rechargeable` suivante :

```
public interface Rechargeable {
    public double getCapacite(); // capacité actuelle en mAh
    public boolean recharger(double quantite); // quantité en mAh ; vrai si plein
}
```

La méthode `getCapacite` renvoie la capacité actuelle exprimée en milliampères-heures (mAh) du rechargeable (et non pas la capacité maximale). La méthode `recharger` recharge la capacité du rechargeable avec `quantite` mAh. Cette méthode renvoie le booléen vrai si le rechargeable est plein, ou faux sinon.

Classe `Appareil` et calcul du poids de l'appareil. Soit la classe `Appareil` suivante :

```
public abstract class Appareil {
    protected final String id ;
    public Appareil(String id) {
        this.id=id;
    }
    public abstract double getPoids(); // en grammes
}
```

Le poids d'un appareil est exprimé en grammes (g). Le poids d'un smartphone est obtenu en divisant sa capacité maximale par 40 et en ajoutant 50. Le poids d'une batterie externe est obtenu en divisant sa capacité maximale par 40.

Exemples : le smarphone d'identifiant SP1001 a actuellement une capacité de 2700 mAh sur une capacité maximale de 3000 mAh, il pèse 125 g. Une batterie externe d'une capacité maximale de 10 000 mAh a un poids de 250 g.

Q5.2 (4 points) Un appareil rechargeable est un appareil qui a la propriété d'être rechargeable. Il a une capacité qui varie entre 0 et sa capacité maximale. Écrire la classe `AppareilRechargeable` qui contient seulement deux attributs :

- `capaciteMax` : la capacité maximale (en mAh) que peut contenir l'appareil rechargeable. La capacité maximale ne doit pas pouvoir être modifiée et peut être connue des classes descendantes.
- `capacite` : la capacité actuelle (en mAh) de l'appareil initialisée à 0.

Elle contient aussi un unique constructeur `AppareilRechargeable(String id, double capaciteMax)` et une méthode `toString` qui renvoie par exemple : `"SP1001 2700/3000mAh 125g"`. Ne pas oublier de définir les méthodes abstraites à définir éventuellement dans cette classe. Quand on recharge un appareil, si sa capacité dépasse sa capacité maximale, alors sa capacité est remise à la valeur de la capacité maximale.

```
1 public abstract class AppareilRechargeable extends Appareil implements
      Rechargeable {
2     protected final double capaciteMax ; // en mAh
3     private double capacite ; // en mAh
4
```

```
5       public AppareilRechargeable(String id, double capaciteMax) {
6           super(id);
7           this.capaciteMax = capaciteMax;
8           this.capacite = 0;
9       }
10      public String toString() {
11          return id+" "+capacite+"/"+capaciteMax+"mAh "+getPoids()+"g";
12      }
13      public double getCapacite() {
14          return capacite ;
15      }
16      public boolean recharger(double quantite) {
17          capacite += quantite ;
18          if (capacite >= capaciteMax) {
19              capacite = capaciteMax;
20              return true;
21          }
22          return false;
23      }
24  }
```

Q5.3 (3 points) Si on utilise un appareil rechargeable, mais qu'il n'a pas assez de capacité actuellement, alors une exception doit être levée.
(a) Écrire la classe `VideException` qui contient un constructeur prenant en unique paramètre l'identifiant d'un appareil et qui a, par exemple, pour message d'erreur : `"SP1001 est vide"`.
(b) Écrire la méthode `utiliser` de la classe `AppareilRechargeable` qui prend en paramètre la quantité (exprimée en mAh) nécessaire pour une utilisation de l'appareil et qui lève l'exception `VideException` si la capacité actuelle de l'appareil est inférieure à la quantité nécessaire, sinon cette méthode diminue la capacité actuelle de la quantité nécessaire. Seules les classes descendantes de `AppareilRechargeable` doivent pouvoir appeler cette méthode.

(a)

```
public class VideException extends Exception {
    public VideException(String id) {
        super(id+" est vide");
    }
}
```

(b)

```
protected void utiliser(double quantite) throws VideException {
    if (capacite < quantite) {
        throw new VideException(id);
    }
    capacite = capacite - quantite ;
}
```

Q5.4 (4 points) On suppose que tous les smartphones ont un identifiant incrémenté automatiquement de la forme "SP" suivi d'un nombre supérieur à 1001 (on considère qu'il sera toujours inférieur à 9999). Exemples : `"SP1001"`, `"SP1002"`, `"SP1003"`, ... Écrire une classe `SmartPhone` contenant un seul attribut servant à générer l'identifiant du smartphone, un constructeur ayant pour seul paramètre la capacité maximale du smartphone. Une méthode `void telephoner()` qui, chaque fois qu'elle est appelée, utilise une quantité équivalente à 10% de la capacité maximale du téléphone. Si le smartphone n'a pas actuellement assez de capacité pour téléphoner alors le message de l'exception est affiché.

```
1   public class SmartPhone extends AppareilRechargeable {
2       private static int cpt = 1001;
3       public SmartPhone(double capaciteMax) {
4           super("SP"+cpt,capaciteMax);
5           cpt++;
6       }
7       public void telephoner() {
8           try {
9               utiliser(capaciteMax/10);
```

```
10          } catch(VideException e) {
11              System.out.println(e.getMessage());
12          }
13      }
14      public double getPoids() {
15          return capaciteMax/40+50;
16      }
17  }
```

Port USB. Un port USB a un type (USB-A, USB-C,...), une intensité en ampère (A) et une tension en volt (V). La puissance d'un port USB est obtenue en multipliant l'intensité par la tension du port. Elle est exprimée en watts (W). Exemple : pour un port de 2A et 5V, la puissance est de 10W. Dans cet examen, on suppose que la tension des ports est toujours de 5V.

Q5.5 (4 points) Pour les besoins de l'examen, on suppose qu'il existe seulement deux ports possibles. On s'inspire du patron de conception Singleton, pour garantir qu'il n'existe que deux instances de la classe `Port`. Écrire la classe `Port` avec 6 attributs :

- `type` : le type du port (`String`),
- `intensite` : l'intensité du port (`double`),
- `TENSION` : une constante (`int`) initialisée à 5,
- `USBA` qui référence une instance de `Port` correspondant au type "USB-A" et à l'intensité 1A,
- `USBC` qui référence une instance de `Port` correspondant au type "USB-C" et à l'intensité 2A,
- `tabPorts` : un tableau qui contient seulement les 2 instances de `Port`.

Cette classe contient également un unique constructeur et seulement 2 méthodes :

- le constructeur `Port(String type, double intensite)`,
- la méthode `getPuissance()`
- la méthode `Port getPortAlea()` qui renvoie aléatoirement un des ports du tableau `tabPorts`.

Vérifier que par vos choix de modificateurs (`static`, `final`, `public`/`private`), il ne peut pas exister 2 instances de la classe `Port` et que l'instance est accessible aux autres classes.

```
1  public class Port {
2      public final String type ;
3      public final double intensite ; // en A
4      public static final int TENSION = 5 ; // 5 volt
5      public static final Port USBA=new Port("USB-A",1) ;
6      public static final Port USBC=new Port("USB-C",2) ;
7      private static final Port [] tabPorts = {USBA,USBC};
8
9      private Port(String type, double intensite) { // private
10         this.type = type ;
11         this.intensite = intensite ;
12     }
13     public double getPuissance() {
14         return TENSION * intensite ;
15     }
16     public static Port getPortAlea() {
17         return tabPorts[(int)(Math.random()*tabPorts.length)];
18     }
19 }
```

Q5.6 (2 points) (a) Écrire l'interface `Chargeur` qui contient une seule méthode `charger` qui prend en paramètre un rechargeable, qui ne retourne rien et dont le but est de recharger (complètement ou en partie) le rechargeable en paramètre. **(b)** Un adaptateur secteur peut charger un rechargeable avec une quantité supposée infinie (utiliser la constante `Double.POSITIVE_INFINITY`) et contient un unique port initialisé avec la méthode `getPortAlea()` de la classe `Port`. Écrire une classe `Adaptateur`.

(a)

```
public interface Chargeur {
    public void charger(Rechargeable r);
}
```

(b)

```
public class Adaptateur implements Chargeur { // n'est pas un appareil
    private Port port ;
    public Adaptateur() {
        port = Port.getPortAlea();
    }
    public void charger(Rechargeable r) {
        r.recharger(Double.POSITIVE_INFINITY);
    }
}
```

Q5.7 (3 points) Une batterie externe est un appareil qui peut être rechargé, mais qui peut aussi servir pour charger un autre rechargeable. On suppose qu'elle a un port d'entrée de type USB-C et un port de sortie de type USB-A. Écrire une classe `Batterie` avec deux attributs, un constructeur et les méthodes nécessaires. On suppose que quand une batterie charge un rechargeable, elle réalise les étapes suivantes :

1. d'abord, elle calcule la quantité qu'elle peut transmettre par son port de sortie. Cette quantité est calculée ainsi : puissance du port de sortie multiplié par 25.
2. ensuite, elle diminue sa capacité de cette quantité et recharge le rechargeable de cette quantité ;
3. enfin, elle recommence l'étape 2. tant que le rechargeable n'est pas plein et que la batterie n'est pas vide.

```
public class Batterie extends AppareilRechargeable implements Chargeur {
    private Port entree ;
    private Port sortie ;

    public Batterie(String id, double capaciteMax) {
        super(id,capaciteMax);
        entree = Port.USBC;
        sortie = Port.USBA;
    }
    public double getPoids() {
        return capaciteMax / 40 ;
    }
    public void charger(Rechargeable r) {
        double quantite = sortie.getPuissance()*25;
        boolean estPlein = false;
        try {
            while (! estPlein) {
                utiliser(quantite);
                estPlein=r.recharger(quantite);
            }
        } catch (VideException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Q5.8 (2 points) Donner les instructions pour :

1. créer un smartphone, un adaptateur secteur et une batterie externe,
2. créer une liste au sens `ArrayList` pouvant contenir des objets rechargeables,
3. ajouter le smartphone et la batterie à la liste,
4. charger avec l'adaptateur tous les objets de la liste.

```
SmartPhone sp = new SmartPhone(3000);
Adaptateur adapt = new Adaptateur();
Batterie bat = new Batterie("BE",5000);
ArrayList<Rechargeable> alr=new ArrayList<Rechargeable>();
alr.add(sp); alr.add(bat);
for(Rechargeable r : alr) {
    adapt.charger(r);
}
```