Redéfinition, Paquetage, structure de donnée, relation d'implantation

Exercice 1 - Redéfinition

```
class A {
  protected int x() {
    return 3;
  }
}
class B extends A {
  public int x() {
    return 4;
  }
  public void printX() {
    System.out.println(x());
  }
}
class Test {
  public static void main(String[] args) {
    B b = new B();
    System.out.println(b.x());
  }
}
```

1 Quelle est la valeur affichée par `printX()` ?

2 Combien de méthodes `x()` sont accessibles dans `B` (i.e. à combien de méthodes `x` une méthode de `B` a t-elle accès ?) ? S'il y en a plusieurs, donner pour chacune un moyen d'accès ?

3 Mêmes questions, mais en se plaçant dans `Test`.

4 Donnez le nom du mécanisme utilisé dans `B` à propos de la méthode `x()`.

Exercice 2 - Surcharge, redéfinition, appel de méthode

Dans les exemples de classes suivants vous commencerez par dire où se trouvent les erreurs de compilation. Vous les expliquerez et retirerez les méthodes en provoquant. Vous indiquerez ensuite où sont les surcharges et les redéfinitions. Pour les appels de méthodes vous indiquerez également les méthodes appelées.

```
class A {
  public void a() {System.out.println("Aa"); }
  void b(B b) {System.out.println("Ab(B)");}
  void c() {System.out.println("Ac");}
  void c(A a) {System.out.println("Ac(A)"); }
  void c(B b) {System.out.println("Ac(B)"); }
  static void d() {System.out.println("static Ad");}
  void e() {System.out.println("Ae");}
  int f() {System.out.println("Af"); return 2;}
  void g() throws RuntimeException {System.out.println("Ag");  }
  void h() throws ArrayIndexOutOfBoundsException
{System.out.println("Ah");  }
  protected void i() {System.out.println("Ai");}
}
class B extends A{
  public void a() {System.out.println("Ba"); }
  protected void b(B b) {System.out.println("Bb(B)");}
```

```
      public void c(A a) {System.out.println("Bc(A)");}
      static void d() {System.out.println("static Bd");}
      static void d(A a) {System.out.println("Bd(A a)");}
      char f() {System.out.println("Bf"); return 'c';}
      void g() throws ArrayIndexOutOfBoundsException
{System.out.println("Bg");  }
      void h() throws Exception {System.out.println("Bh");}
      private void i() {System.out.println("Bi");}
    }
    class Min {
          public static void main(String[] args) {
            A aa = new A();
            A ab = new B();
            B bb = new B();
            aa.a();
            ab.a();
            aa.b(aa);
            ab.b(ab);
            bb.e();
            aa.d();
            bb.d();

            aa.c(aa);
            aa.c(ab);
            aa.c(bb);
            ab.c(aa);
            ab.c(ab);
            ab.c((B)ab);

            ab.c(bb);
            ab.c((A)bb);
            ((B)ab).c(aa);
            ((B)ab).c(ab);
            ((B)ab).c((B)ab);
            ((B)ab).c(bb);
            bb.c(ab);
            bb.c(bb);
            ((A)bb).c(aa);
            bb.c(ab);
            bb.c(bb);
          }
    }
```

Exercice 3 - Les liste chaînée

Le but de cet exercice est d'écrire une implanattion de liste de chainée en utilisant les paquetages.
Les sources (les .java) seront stockées dans un répertoire nommé `src`.
Pour la suite de l'exercice, l'ensemble des classes créées devra être créé dans le paquetage `fr.umlv.datas`.
Pour cela, créer un répertoire "fr" dans le répertoire "src", puis un répertoire "umlv" à l'intérieur du répertoire "fr" et enfin un répertoire "datas" à l'intérieur du répertoire "umlv".
Les fichiers sources (`.java`) du paquetage `fr.umlv.datas` seront donc stockées dans le répertoire `src/fr/umlv/datas`
Nous allons dans un premier temps, créer une liste chaînée de chaîne de caractères.

1 Créer une classe `fr.umlv.datas.Link` correspondant à un maillon de la liste chaînée. (donc un fichier `Link.java` dans le répertoire `src/fr/umlv/datas`
Pour compiler, placez-vous dans `src` et écrire `javac fr/umlv/datas/Link.java`.
Où est généré le `.class` correspondant ?
2 Quelle est la commande pour exécuter le `main` de la classe `fr.umlv.datas.Link` ?

3 Créer une classe `fr.umlv.datas.LinkedLink` qui maintient une référence sur le premier maillon de la liste.
Cette classe devra définir les méthodes :
1 `add(String text)` qui ajoute un élément avant le premier élément.
2 `size()` qui affiche le nombre d'éléments de la liste.
3 `toString()` qui affiche le contenu de la liste.

Pour tester ses classes, créer une classe `Main` sans paquetage (on dit dans le paquetage par défaut) et utiliser le mot-clé `import` pour utiliser les classes `fr.umlv.datas.Link` et `fr.umlv.datas.LinkedLink`.

Exercice 4 - liste chaînée (suite)

1 Jusqu'à présent les fichiers byte-code sont stockés au même endoit que les `.class`.
Nous allons demander au compilateur de générer ceux-ci dans un autre répertoire.
Pour cela, utilisez l'option `'-d'` du compilateur, donc avec le chemin courant pointant sur `src` :

```
javac -d ../classes fr/umlv/datas/*.java
```

Noter que l'arborescence des répertoires sous `classes` est généré automatiquement.

2 Quelle ligne de command doit-on maintenant utiliser pour exécuter le `main` de test :
Si on se place dans le répertoire `classes`
Si on se place dans le répertoire parent des répertoires `src` et `classes`.

3 A quoi sert la variable d'environnement `CLASSPATH` ?

1 Renommer la classe `Main` en `fr.umlv.datas.main.Main`.

2 Implanter `String get(int index)` qui renvoie la `index`ième chaîne de caractère.
Que doit-on faire si l'index est invalide ?
Pourquoi serait-il logique de changer l'implantation de `size` pour que la méthode s'exécute en temps constant ?
Ré-implanter `size`.

3 Changer la classe `fr.umlv.datas.LinkedLink` pour une implantation plus générique à base d'`Object`.
Que doit-on changer dans la classe `fr.umlv.datas.main.Main` ?

4 Implanter la méthode `boolean contains(Object o)` indiquant si un objet est où non contenu dans la liste chaînée.