Exercices de Programmation Orientée Objet en Java

1. Modularité (objet et classe)

1.1 Analyse de code - solution
1.2 Analyse de code - solution
1.3 Analyse de code - solution
1.4 Analyse de code - solution
1.5 Analyse de code - solution
1.6 Analyse de code - solution
1.7 Analyse de code - solution
1.8 Analyse de code - solution
1.9 Analyse de code - solution
1.10 Analyse de code - solution
1.11 Poupées russes - solution
1.12 Montres - solution
1.13 Recettes - solution

2. Héritage et polymorphisme

2.1 Analyse de code - solution
2.2 Analyse de code - solution
2.3 Analyse de code - solution
2.4 Analyse de code - solution
2.5 Analyse de code - solution
2.6 Analyse de code - solution
2.7 Analyse de code - solution
2.8 Analyse de code - solution
2.9 Analyse de code - solution
2.10 Sujets d'examen - solution
2.11 Élus - solution
2.12 Diagnostics et soins - solution

3. Abstraction

3.1 Analyse de code - solution
3.2 Analyse de code - solution
3.3 Analyse de code - solution
3.4 Analyse de code - solution
3.5 Analyse de code - solution
3.6 Tri - solution
3.7 Poupées russes - solution
3.8 Jeu vidéo - solution
3.9 Arbres - solution
3.10 Précipitations - solution
3.11 Bibliothèque - solution

4. Encapsulation

4.1 Analyse de code - solution
4.2 Analyse de code - solution
4.3 Analyse de code - solution
4.4 Analyse de code - solution
4.5 Analyse de code - solution
4.6 Analyse de code - solution
4.7 Analyse de code - solution
4.8 Analyse de code - solution
4.9 Analyse de code - solution

5. Exercices récapitulatifs

5.1 Graphes - solution
5.2 Les enseignants dans le supérieur - solution
5.3 Laboratoire de chimie - solution
5.4 Le Seigneur des Anneaux (1) - solution
5.5 Le Seigneur des Anneaux (2) - solution
5.6 Le Seigneur des Anneaux (3) - solution
5.7 La Guerre des Étoiles (1) - solution
5.8 La Guerre des Étoiles (2) - solution
5.9 La Guerre des Étoiles (3) - solution
5.10 Tableaux associatifs - solution
5.11 Modèle relationnel - solution
5.12 Formes géométriques - solution

Exercice 1.1 : ce code compile t-il et, si non, quelle(s) erreur(s) le compilateur va t-il indiquer?

```
class Toto{
  int toto = 0;
  Toto() {
    toto = toto + 1;
  }
  public static void main(String[] tutu) {
    Toto t1 = new Toto();
    Toto t2 = new Toto();
    System.out.println("Toto : " + toto);
  }
}
```

Exercice 1.2 : le code suivant compile t-il? Si non, indiquez les erreurs affichées par le compilateur et proposez des corrections. À quel affichage conduit l'exécution du programme (éventuellement corrigé)?

```
class Test {
  int i;
  Test(int i) {
    this.i = 12;
    i = 15;
  }
  void i() {
    Test i = new Test(3);
    System.out.println(i.i);
    i.i();
  }
  public static void main(String[] toto) {
    Test i = new Test(34);
    i.i();
  }
}
```

Exercice 1.3 : le code suivant compile t-il? Si non, indiquez les erreurs affichées par le compilateur et proposez des corrections. À quel affichage conduit l'exécution du programme (éventuellement corrigé)?

```
class Exo2 {
  Exo2 e;
  Exo2(Exo2 e) {
    this = e;
  }
  Exo2() {
    this.e = this;
```

retour au sommaire

```
  }
  public String toString() {
    if(this.e == null) return "NULL";
    else return "LLUN";
  }
  Exo2 m1() {
    System.out.println("Bonjour le monde");
    return this;
  }
  void m2(Exo2 e) {
    this.e = null;
    this.m1();
    e.e = this;
    System.out.println(this);
  }
  public static void main(String[] truc) {
    new Exo2();
    Exo2 e = new Exo2();
    e.m2(e);
    Exo2 f = new Exo2(e);
    e.m2(f);
  }
}
```

Exercice 1.4 : le code suivant compile t-il? Si non, indiquez les erreurs affichées par le compilateur et proposez des corrections. A quel affichage conduit l'exécution du programme (éventuellement corrigé)?

```
class Test {
  int i;
  Test(Test t) {
    if(t == null) this.i = 12;
    else{
      t.m();
      this.i = t.i;
    }
  }
  void m() {
    this.i++;
    System.out.println(this.i);
  }
  public static void main(String[] toto) {
    Test i = new Test(new Test(new Test(null)));
  }
}
```

retour au sommaire

Exercice 1.5 : le code suivant compile t-il? Si non, indiquez les erreurs affichées par le compilateur et proposez des corrections. À quel affichage conduit l'exécution du programme (éventuellement corrigé)?

```
class C1{
  C2 c;
  int i;
  C1(int i, C2 c){
    if(c == null) this.c = new C2(i+1,this);
    else this.c = c;
    this.i = i;
  }
  public static void main(String[] toto){
    C1 c = new C1(1,null);
    C2 d = new C2(c.i,c);
    C1 e = new C1(d.i,d);
    System.out.println(e.i + "," + e.c.i + "," + e.c.c.i + "," + e.c.c.c.i);
  }
}

class C2{
  C1 c;
  int i;
  C2(int i, C1 c){
    if(c == null) this.c = new C1(i+1,this);
    else this.c = c;
    this.i = i;
  }
}
```

Exercice 1.6 : le code suivant compile t-il? Si non, indiquez les erreurs affichées par le compilateur et proposez des corrections. À quel affichage conduit l'exécution du programme (éventuellement corrigé)?

```
class Test{
  int i;
  Test(){
    this.i = 1;
  }
  Test(int i){
    this();
    this.i = i;
  }
  void m(){
    this.i++;
    System.out.println(this.i);
  }
```

retour au sommaire

```
  protected void finalize(){
    System.out.println(this.i);
  }
  public static void main(String[] toto){
    Test i = new Test(2);
    i.m();
    i = new Test();
    System.gc();
  }
}
```

Exercice 1.7 : le code suivant compile t-il? Si non, indiquez les erreurs affichées par le compilateur et proposez des corrections. À quel affichage conduit l'exécution du programme (éventuellement corrigé)?

```
class A{
  int i;
  A a;
  A(int i){
    if(i<=0){
      this.i = i;
      this.a = new A(i-1);
    }
    else this.i = i;
  }
  void passeATonVoisin(){
    this.i++;
    if(this.a!=null) this.a.passeATonVoisin();
    else System.out.println(this.i);
  }
  public static void main(String[] t){
    A a = new A(10);
    a.passeATonVoisin();
  }
}
```

Exercice 1.8 : à quel affichage conduit l'exécution du programme suivant?

```
class Truc{
  String s;
  Truc(){
    this.s = "Bonjour";
```

retour au sommaire

```
  }
  Truc(String s){
    this.s = s;
  }
  public String toString(){
    return this.s;
  }
  public boolean equals(Object o){
    return o instanceof Truc && this.s.equals(((Truc) o).s);
  }
}

class Bidule{
  String s;
  Truc t;
  Bidule(Truc t){
    this.t = t;
    if(t!=null) this.s = t.s;
    else this.s = "Bonjour";
  }
  public String toString(){
    if(this.t == null) return this.s;
    else return this.t.s;
  }
  public static void main(String[] toto){
    Truc t1 = new Truc(), t2 = new Truc("Hello");
    Bidule b1 = new Bidule(t1), b2 = new Bidule(null);
    System.out.println(t1.toString());
    System.out.println(t2.toString());
    System.out.println(b1.toString());
    System.out.println(b2.toString());
    System.out.println(t1.equals(t2));
    System.out.println(t1.equals(b1));
  }
}
```

Exercice 1.9 : à quel affichage conduit l'exécution du programme suivant?

```
class A{
  int i = 0;
  A(int j){
    this.i = j;
  }
  void setI(int k){
    this.i=k;
  }
```

retour au sommaire

```
  void setI(A a){
    this.i = a.i;
  }
}

class B extends A{
  int i = 1;
  B(){
    super(2);
  }
  void setI(int l){
    this.i = l;
  }
  void setI(A a){
    super.i = a.i;
  }
  void setI(B b){
    this.i = b.i;
  }
  public static void main(String[] truc){
    A a = new A(5);
    B b = new B();
    System.out.println("a.i="+a.i+", b.i="+b.i+" ou "+((A) b).i);
    b.setI(3); b.setI(a);
    System.out.println("a.i="+a.i+", b.i="+b.i+" ou "+((A) b).i);
  }
}
```

Exercice 1.10 : Ce code compile t-il et, si non, quelle(s) erreur(s) le compilateur va t-il indiquer?

```
class Toto{
  int toto = 0;
  Toto() {
    toto = toto + 1;
  }
  public static void main(String[] tutu) {
    Toto t1 = new Toto();
    Toto t2 = new Toto();
    System.out.println("Toto : " + toto);
  }
}
```

Exercice 1.11 : on veut écrire un programme simulant des poupées russes de di#érentes tailles. Chaque poupée a une taille donnée, peut s'ouvrir ou se fermer, peut contenir une autre poupée et être contenue dans une autre poupée.

Écrivez une classe PoupeeRusse contenant les méthodes suivantes :

retour au sommaire

- un constructeur
- *void ouvrir()* : ouvre la poupée si elle n'est pas déjà ouverte et si elle ne se trouve pas à l'intérieur d'une autre poupée
- *void fermer()* : ferme la poupée si elle n'est pas déjà fermée et si elle ne se trouve pas à l'intérieur d'une autre poupée
- *void placerDans(PoupeeRusse p)* : place la poupée courante dans la poupée p si c'est possible. Il faut que la poupée courante soit fermée et ne soit pas déjà dans une autre poupée, que la poupée p soit ouverte et ne contienne aucune poupée, et que p soit d'une taille supérieure à la poupée courante.
- *void sortirDe(PoupeeRusse p)* : sort la poupée courante de la poupée p si elle est dans p et si p est ouverte.

Exercice 1.12 : on veut réaliser un programme qui représente des montres et les gens qui les portent. Une montre donne l'heure et les minutes. On peut initialiser une montre soit à partir d'un couple heure/ minute donné, soit en utilisant l'heure affichée par une autre montre. Il doit être possible de faire avancer l'heure d'une montre en ajoutant une minute (attention, les minutes sont limitées à 60 et les heures à 24).

a. Écrivez une classe qui représente les montres telles que décrites au dessus et une méthode principale qui crée une montre affichant 13h45 et une autre montre qui est un clône de la première.

Une personne a un nom et peut éventuellement porter une montre. On peut faire porter une montre donnée à une personne, si elle n'en a pas déjà une. On peut aussi lui enlever sa montre si elle en a une. Une personne peut demander l'heure à une autre, qui lui donne l'heure sous forme d'une chaine de caractères, en consultant sa montre si elle en a une (sinon elle peut retourner une chaine vide).

b. Écrivez une classe qui représente les personnes telles que décrites au dessus.

c. On veut faire en sorte que chaque montre ne soit portée que par une seule personne. Proposez des ajouts/modifications des deux classes précédentes pour garantir cela.

Exercice 1.13 : on veut réaliser un programme de gestion des recettes de cuisine, qui sera installé sur des appareils électroménagers pour leur permettre de faire la cuisine de façon autonome. Un programmeur a déjà écrit la classe Ingredient donnée ci-dessous :

```
class Ingredient{
  String nom_aliment, etat;
  int quantite;
  String unite;
  Ingredient(String n, String e, int q, String unite){
    this.nom_aliment = n;
    this.etat = e;
    this.quantite = q;
    this.unite = unite;
  }
}
```

retour au sommaire

NB: l'état d'un ingrédient peut être cuit, entier, cru, découpé, ou une combinaison de ces états (par exemple cuit et entier). L'unité peut être une unité de poids (gramme, kg, etc), de volume (litre, ml, cl) ou simplement une *cardinalité*.

a. Écrivez une classe Plat qui représente les plats, chaque plat ayant un nom et une liste d'ingrédients. On doit pouvoir créér un plat avec son nom. Il faut également avoir des accesseurs sur le nom du plat et les ingrédients, et pouvoir ajouter un ingrédient à un plat. Écrivez également une méthode main qui crée un plat appelé *choucroute* contenant comme ingrédients : 500g de choucroute cuite, 150g de lard cuit et entier et 2 saucisses entières et cuites

b. On veut pouvoir comparer les plats et donc leurs ingrédients. Ajoutez une méthode equals dans la classe Ingrédient qui renvoie true si deux ingrédients ont le même nom d'aliment et le même état (pas forcément la même quantité). Ajoutez une méthode equals dans la classe Plat, qui renvoie true si deux plats contiennent les mêmes ingrédients, au sens donné juste avant.

c. On veut faire la distinction entre les ingrédients qu'on peut cuire et ceux qu'on peut découper. Un ingrédient qu'on peut cuire doit avoir une méthode cuire() qui le fait passer dans l'état "cuit" et une température de cuisson. Un ingrédient qu'on peut découper doit avoir une méthode decouper() qui le fait passer dans l'état "découpé". Proposez du code objet pour représenter ces types d'ingrédients.

Exercice 2.1 : indiquez si ce code compile et, s'il ne compile pas, quelle(s) erreur(s) va afficher le compilateur.

```
interface I {
  public int getI();
}

abstract class C implements I {
  int i;
  C(int i) {
    this.i = i;
  }
}

class D extends C implements I {
  D(int x) {
    super(x);
  }
  public int getI() {
    return this.i;
  }
}
```

Exercice 2.2 : indiquez quel affichage va produire l'exécution de la classe Essai.

retour au sommaire

```
public class Toto {
  int x;
  Toto(int k) {
    x = k;
  }
  int ajoute(int a) {
    return x+a;
  }
  public void moi() {
    System.out.println(" x = "+ x);
  }
}

public class Titi extends Toto {
  int y;
  Titi(int k, int l) {
    super(k);
    y = l;
  }
  int ajoute(int a) {
    return x+2*a;
  }
}

public class Tutu extends Titi {
  int z;
  Tutu(int k, int l, int m) {
    super(k, l);
    z = m;
  }
  int ajoute(int a) {
    return x+3*a;
  }
  public void moi() {
    super.moi();
    System.out.println(" z = "+ z);
  }
}

public class Essai {
  public static void main (String coucou[]) {
    int a = 2;
    Toto p = new Toto(3);
    p.moi();
    System.out.println(" ajoute("+ a +") = "+ p.ajoute(a));
    Titi e1 = new Titi(3, 4);
```

retour au sommaire

```
    e1.moi();
    System.out.println(" ajoute("+ a +") = "+ e1.ajoute(a));
    e1 = new Tutu(3, 4, 5);
    e1.moi();
    System.out.println(" ajoute("+ a +") = "+ e1.ajoute(a));
}
```

Exercice 2.3 :

```
class Toto {
  int i,j;
  Toto(int i) {
    this.i = i;
    this.j = i;
  }
}

class Titi extends Toto {
  int k;
  Titi(int i) {
    super(i);
    this.i = i+super.i;
    this.k = this.j+2;
  }
  public static void main(String[] truc){
    …
    System.out.println(t.i);
    System.out.println(t.j);
    System.out.println(t.k);
  }
}
```

Si on remplace les … par chacune des quatre instructions ci-dessous, indiquer si le programme compile et si oui, à quel affichage il conduit.

a. Toto t = new Titi(1);
b. Titi t = new Titi(1);
c. Titi t = (Toto) new Titi(1);
d. Titi t = new Titi(1);

Exercice 2.4 :

```
class A{
  B b;
  void addB(B b){
    this.b = b;
```

retour au sommaire

```
  }
  int m(){
    if(this.b != null) return -1 + this.b.m(); else return 0;
  }
}

class B extends A{
  int i = 1;
  int m(){
    i = this.i+1;
    return this.i;
  }
}
```

Pour chaque programme ci-dessous, indiquez s'il compile et si oui, à quel affichage il conduit.

a.
```
A a = new A();
B b = new B();
a.addB(b);
System.out.println(a.m());
System.out.println(b.m());
```

b.
```
A a = new A();
a.addB(new B());
System.out.println(a.b.m());
System.out.println(((A) a.b).m());
```

c.
```
A x = new B();
A y = new A();
B z = new B();
x.addB(z);
System.out.println(x.m());
System.out.println(y.m());
```

Exercice 2.5 : le code suivant compile t-il? Si non, indiquez les erreurs affichées par le compilateur et proposez des corrections. À quel affichage conduit l'exécution du programme (éventuellement corrigé)?

```
class B{
  int i = 5;
  B(){ this.i = this.i-1; }
  B(int i){ this(); this.i = i; }
}

class C extends B{
  C(int i){ this.i = i; }
  public static void main(String[] arggggghhhh){
    B b = new B(2);
    C c = new C(1);
    System.out.println(b.i + "  " + c.i);
  }
}
```

Exercice 2.6 : à quel affichage conduit l'exécution du programme suivant?

```
class A{
```

retour au sommaire

```
  A a;
  A(){ this.a = this; }
  A(A a){ this.a = a;  }
  void m(){
    if(this == this.a) System.out.println("Ahah!");
    else System.out.println("Héhé!");
  }
}

class B extends A{
  B o;
  B(){ super(); this.a = this; this.o = (B) this; }
  void m(){ System.out.println("Ohoh!"); }
  public static void main(String[] toto){
    A u = new A();
    A i = new A(u);
    A b = new B();
    u.m();
    i.m();
    b.m();
    ((B) b).o.m();
  }
}
```

Exercice 2.7 : le code suivant compile t-il? Si non, indiquez les erreurs, si oui, indiquez à quel affichage conduit l'exécution du programme (les classes sont supposées être écrites dans des fichiers séparés).

```
class A {
  String f(B obj) { return ("A et B"); }
  String f(A obj) { return ("A et A"); }
}

class B extends A {
  String f(B obj) { return ("B et B"); }
  String f(A obj) { return ("B et A"); }
}

class Test {
  public static void main (String [] args) {
    A a1 = new A();
    A a2 = new B();
    B b = new B();
    System.out.println(a1.f(a1));
    System.out.println(a1.f(a2));
    System.out.println(a2.f(a1));
```

retour au sommaire

```
    System.out.println(a2.f(a2));
    System.out.println(a2.f(b));
    System.out.println(b.f(a2));
  }
}
```

Exercice 2.8 : le code suivant compile t-il? Si non, indiquez les erreurs affichées par le compilateur et proposez des corrections. A quel affichage conduit l'exécution du programme (éventuellement corrigé)?

```
class A extends Object{
  int i;
  A(int i){ this.i = i; }
}

class B extends A{
  A bidule;
  int i = 2;
  B(){ this.bidule = this; }
  B(A a){
    super(3);
    this.bidule = a;
  }
  public static void main(String[] toto){
    A a2 = new A(5);
    B b1 = new B(a2);
    B b2 = new B();
    System.out.println(b1.i);
    System.out.println(((A) b1).i);
    System.out.println(b1.bidule.i);
    System.out.println(b2.i);
    System.out.println(((A) b2).i);
    System.out.println(b2.bidule.i);
  }
}
```

Exercice 2.9 : le code suivant compile t-il? Si non, indiquez les erreurs affichées par le compilateur et proposez des corrections. A quel affichage conduit l'exécution du programme (éventuellement corrigé)?

```
class A{
  int i;
  A(){
    this.i = 0;
  }
  void m(){System.out.println("A");}
```

retour au sommaire

```
}

class B extends A{
  int i = 2;
  void m(){System.out.println("B");}
  public static void main(String[] toto){
    A a = new A();
    B b = new B();
    System.out.println(b.i);
    System.out.println(a.i);
    System.out.println(((A) b).i);
    b.m();
    a.m();
  }
}
```

Exercice 2.10 : on veut développer un logiciel pour générer des sujets d'examen. Un programmeur a écrit la classe suivante avant de démissionner (il a gagné au loto). Vous devez continuer le développement.

```
class Question{
  String enonce;
  int difficulte=50; // la difficulte varie de 0 à 100
  Question(String e){
    this.enonce = e;
  }
  void setDifficulte(int d){
    this.difficulte = d;
  }
  int getDifficulte(){
    return this.difficulte;
  }
  String getEnonce(){
    return this.enonce;
  }
}
```

On veut représenter les questions de QCM (Questionnaire à Choix Multiples). Chaque question à choix multiple a, en plus d'un énoncé et d'une difficulté, un ensemble de réponses possibles, chaque réponse ayant un énoncé et une valeur de vérité (une réponse peut être vraie ou fausse).

a. Proposez du code pour représenter les questions à choix multiples.

b. Un sujet d'examen est une liste de questions (de QCM ou autre) avec un barême. Chaque question du sujet a donc un nombre de points. Sans modifier les classes que vous avez écrites avant, écrivez du code objet pour représenter les sujets, avec en particulier une méthode qui permet d'obtenir

retour au sommaire

la difficulté moyenne du sujet. La difficulté moyenne est la somme des difficultés des questions du sujet, divisée par le nombre de questions.

Exercice 2.11 : on veut écrire un programme pour simuler le comportement des élus. Un programmeur a écrit la classe ci-dessous (avant de démissionner pour se présenter aux élections législatives).

```
class Personne{
  String nom, prenom;
  int compteBancaire; // montant de son compte en euros
  Personne(String nom, String prenom){
    this.nom = nom; this.prenom = prenom;
    this.compteBancaire = 0;
  }
  void addSous(int montant){
    this.compteBancaire = this.compteBancaire+montant;
  }
}
```

En réutilisant cette classe Personne, écrivez une classe qui représente les élus, qui sont des personnes avec un ensemble d'assistants (qui sont aussi des personnes bien sur). Un élu peut embaucher un assistant ou le licencier. Un élu peut aussi distribuer à ses assistants une dotation budgétaire : il répartit la somme qui lui est allouée de façon équitable entre ses assistants en ajoutant des sous sur les comptes bancaires des assistants.

a. Écrivez une classe Elu qui représente les élus.

Dans la suite, on veut représenter différents types d'élus, mais qu'on veut pouvoir manipuler comme des élus "normaux", c'est-à-dire qu'un objet de type Elu n'aura toujours que les trois méthodes définies dans la classe Elu et permettant d'embaucher un assistant, de licencier un assistant et de verser une dotation.

b. On veut représenter les élus mafieux, qui n'embauchent comme assistants que des membres de leur famille (c'est-à-dire des personnes qui ont le même nom qu'eux, pour simplifier). Mais ils répartissent la dotation budgétaire qui leur est allouée de façon équitable entre leurs assistants. Écrivez une classe qui représente les élus mafieux.

c. On veut représenter les élus économes, qui répartissent la dotation budgétaire qui leur est allouée de la façon suivante : chaque assistant reçoit au plus le SMIC (1480 euros brut) et s'il reste des sous, ils ne sont pas dépensés. Écrivez une classe qui représente les élus économes.

d. On veut représenter les élus escrocs, qui ont un compte en Suisse en plus de leur compte bancaire normal, qui n'embauchent pas forcément que les membres de leur famille, et qui répartissent la dotation budgétaire qui leur est allouée de la façon suivante : chaque assistant reçoit au plus le SMIC (1480 euros brut) et s'il reste des sous, ils sont versés sur le compte en Suisse de l'élu. Écrivez une classe qui représente les élus escrocs.

retour au sommaire

Exercice 2.12 : on veut développer un logiciel pour informatiser le parcours de soin dans un hopital. Un programmeur a écrit le code suivant, mais il a du arrêter le développement (il a été hospitalisé).

```
enum Maladie{
  // on suppose que toutes les maladies sont énumérées ici
  GRIPPE, PESTE, CHOLERA, ...;
}

import java.time.Instant;
class Diagnostic{
  String soignant;
  Instant date;
  Maladie maladie;
  int validite; // fiabilité du diagnostic, entre 0 et 100%
  Diagnostic(String s, Instant d, Maladie m, int v){
    this.soignant = s;
    this.date = d;
    this.maladie = m;
    this.valididte = v;
  }
}
```

Il faut représenter dans le logiciel les patients, avec leurs nom, prénom, date de naissance, sexe et les actes médicaux qu'ils ont subis. Ces actes peuvent être des diagnostics ou des soins. Un soin est réalisé par un personnel soignant, à une certaine date et produit une certaine amélioration de l'état du patient (en pourcentage). On peut représenter les dates avec la classe java.time.Instant. Il doit être possible de récupérer l'âge d'un patient et de lui ajouter un acte médical (si d est un objet de type Instant, le nombre d'années (au format long) qui sépare d de maintenant est d.until(Instant.now(),ChronoUnit.YEARS)).

a. Proposez du code objet pour représenter les patients et les actes (on a le droit de modifier la classe Diagnostic).

Parmi tous les soins possibles, il y a la prise de médicaments, avec une posologie (quantité entière) et une fréquence (nombre de prises du médicament par jour).

b. Proposez du code objet pour représenter les prises de médicaments. Ajoutez une méthode permettant de savoir combien un patient doit prendre de médicaments par jour.

Exercice 3.1 : le code suivant compile t-il? Si non, indiquez les erreurs (les classes sont supposées être écrites dans des fichiers séparés).

```
abstract class Machin{
  int i = 1;
  static int j = 3;
  void bidule();
```

retour au sommaire

```
  Machin(){
    i = 4;
    this.j = 5;
  }
}

class Bidule extends Machin{
  void bidule(Machin m){
    System.out.println("Chose");
  }
}
```

Exercice 3.2 : le code suivant compile t-il et si non pourquoi?

```
class Machin {
  int i = 1;
  static int j = 3;
  static { j = 2; }
  Machin() {
    i = 4;
    j = 5;
  }
}
```

Exercice 3.3 : le code suivant compile t-il? Si non, indiquez les erreurs.

```
class C{
  interface I{
    int i;
    public void m();
  }
  class D implements I{
    public int m(){
      System.out.println("Je suis un D");
    }
  }
  abstract class E implements I{
    public void m(){
      System.out.println("Je suis un E");
    }
  }
}
```

Exercice 3.4 : le code suivant compile t-il? Si non, indiquez les erreurs. On suppose que la classe CaRentrePasException existe.

```
class MorceauDeGruyere{
```

retour au sommaire

```
 private int poids;
 public MorceauDeGruyere(int p){this.poids = p;}
 private interface Trou{
  // cette méthode permet de remplir le trou avec du gruyère
  public void remplir(MorceauDeGruyere o) throws CaRentrePasException;
 }
 public class TrouCarre implements Trou{
  private int largeur, hauteur, profondeur;
  private TrancheDeGruyere mg;
  public TrouCarre(int l, int h, int p){
   this.largeur = l; this.hauteur = h; this.profondeur = p;
  }
  public void remplir(MorceauDeGruyere o) throws CaRentrePasException{
   if(o instanceof TrancheDeGruyere){
    TrancheDeGruyere tg = (TrancheDeGruyere) o;
    if(tg.getL() <= this.largeur && tg.getH() <= this.hauteur && tg.getP() <= this.profondeur)
                                                                        this.mg = tg;
    else throw new CaRentrePasException();
   }
   else throw new CaRentrePasException();
  }
 }
 class TrancheDeGruyere extends MorceauDeGruyere{
  private int largeur, hauteur, profondeur;
  public int getL(){return this.largeur;}
  public int getH(){return this.hauteur;}
  public int getP(){return this.profondeur;}
  public TrancheDeGruyere(int p, int l, int h, int pr){
   super(p);this.largeur = l; this.hauteur = h;
   this.profondeur = pr;
  }
 }
}
```

Exercice 3.5 : le code suivant compile t-il? Si non, indiquez les erreurs. Proposez éventuellement des corrections et indiquez quel sera l'affichage lors de l'exécution une fois ces corrections effectuées.

```
interface Titi{
 int m1();
}

interface Tutu extends Titi{
 int m1();
 int m2();
}
```

retour au sommaire

```
abstract class Toto extends Object implements Tutu{
  private int i;
  public Toto(int j){
    super();
    this.i = j;
  }
  public static void main(String[] toto){
    Tete t = new Toto();
    System.out.println(t.m1());
    System.out.println(t.m2());
  }
}

class Tete extends Toto{
  private int i;
  public Tete(){
    super();
    this.i = 2;
  }
  public int m1(){return this.i;}
  public int m2(){return super.i;}
}
```

Exercice 3.6 : on veut écrire une fonction de tri de liste d'éléments, qui puisse être utilisée pour n'importe quel type de liste et n'importe quel type d'élément mais avec les contraintes suivantes :

- les listes triables doivent avoir une méthode pour récupérer un élément par son indiceget(int i) et une méthode pour échanger deux éléments aux indices i et j swap(int i, int j).
- les éléments des listes triables doivent pouvoir être comparés deux à deux en utilisant une méthode int compare(... o) qui renvoie un entier négatif si l'objet sur lequel on appelle la méthode est plus petit que o, 0 si est égal et un entier positif s'il est plus grand.

a. Écrire des types de données objets (classe, classe abstraite, interface) permettant d'écrire une telle fonction de tri.

b. Réécrire la fonction ci-dessous qui tri des tableaux d'entiers entre deux indices a et b, de manière à ce qu'elle puisse trier n'importe quelle liste vérifiant les conditions posées.

```
public void quickSort(int[] tab, int a, int b) {
  int pivot;
  if (b > a) {
    pivot = a;
    for (int i = a+1; i <= b;i++) {
      if (tab[i] < tab[pivot]) {
        temp = tab[i];
        tab[i] = tab[pivot+1];
        tab[pivot+1] = tab[pivot];
```

retour au sommaire

```
      tab[pivot] = temp;
      pivot = pivot+1;
     }
    }
   quickSort(tab,a,pivot-1);
   quickSort(tab,pivot+1,b);
  }
}
```

Exercice 3.7 : on veut représenter des poupées russes dans un programme objet. Ces poupées peuvent s'emboiter les unes dans les autres. On a déjà écrit la classe suivante :

```
public class PoupeeRusse{
  private int taille;
  public PoupeeRusse(int t){
    this.taille = t;
  }
}
```

On veut distinguer deux types de poupées : celles qui sont creuses et dans lesquelles on peut mettre une autre poupée si elle est plus petite, et celles qui sont pleines et ne peuvent rien contenir.

a. Proposez du code permettant de représenter ces poupées et, pour les poupées creuses, d'ajouter une poupée dedans ou d'en retirer une poupée (on ne peut ajouter une poupée que dans une poupée creuse qui n'en contient pas encore et on ne peut retirer une poupée que d'une poupée creuse qui en contient une autre).

b. Ecrire une fonction main qui crée 3 poupées, de tailles différentes, la plus petite n'étant pas creuse, et qui les emboite les unes dans les autres. On ne tient pas compte de l'ordre d'emboitement (c'est-à-dire qu'on n'est pas obligé d'aller de la plus petite à la plus grande).

Exercice 3.8 : on veut réaliser un jeu vidéo dans lequel des personnages ou des animaux se déplacent de différentes façons. Une interface décrivant les méthodes de déplacement a déjà été écrite. Elle est utilisée pour les animaux et ne peut donc pas être modifiée (la classe Point possède deux attributs publics x et y de type int) :

```
public interface Localisable{
  public final int PAS = 10;
  public java.awt.Point getPosition();
  public void avance(); // fait avancer l'objet
}
```

De plus, une classe implémentant un comportement de déplacement aléatoire a déjà été écrite :

```
public class Exploration implements Localisable{
  private int x,y;
```

retour au sommaire

```
  public Exploration(int x, int y){
    this.x = x; this.y = y
  }
  public java.awt.Point getPosition(){ return new java.awt.Point(x,y); }
  public void avance(){
    double dx = Math.random();
    double dy = Math.random();
    this.x = (int) (this.x+dx*PAS/Math.sqrt(dx*dx+ dy*dy));
    this.y = (int) (this.y+dy*PAS/Math.sqrt(dx*dx+ dy*dy));
  }
}
```

Il faut maintenant écrire le code qui permet de gérer les personnages.

a. Proposez du code objet pour implémenter les comportements de déplacement non aléatoire, c'est-à-dire les déplacements qui se font selon une direction donnée.

b. Proposez du code objet pour représenter les personnages qui ont un nom et peuvent adopter différents comportements de déplacement. On peut supposer qu'à la création d'un personnage, il se déplace aléatoirement, mais il peut, au cours de l'exécution du jeu, changer de façon de se déplacer. Écrire une méthode principale qui crée un personnage nommé Toto, positionné au point (0,0), puis lui donne un mode de déplacement non aléatoire avec une direction de vecteur (6,8).

Exercice 3.9 : l'Office National des Forêts (ONF) veut réaliser un programme pour gérer les données récoltées par les ingénieurs des eaux et forêts concernant les arbres et les parcelles. Un début de code a été écrit par un programmeur qui ne connait pas la POO. Proposez un nouveau programme conforme aux principes de la POO et qui o#re les mêmes possibilités de représentation des données et les mêmes fonctionnalités.

```
public class ONF_Data {
  private int typeArbre; // 0:résineux, 1:feuillu
  private int hauteur, diametre;
  private int parcelle; // chaque parcelle est identifiée par un entier
  private int[] surfaces; // surfaces[i] est la surface de la parcelle numéro i
  public static void creerArbre(int type, int h, int d, int parcelle) {
    ...
  }
  // affecte une surface à une parcelle
  public static void setSurface(int parcelle, int surface) {
    ...
  }
  // retourne le nombre d'arbres sur une parcelle
  public static int nombreArbres(int parcelle) {
    ...
  }
}
```

retour au sommaire

Exercice 3.10 : On veut écrire un programme pour simuler les effets des précipitations au sol. On suppose que le terrain est découpé en zones, qui chacune possède un nom, une superficie (en km²) et un certains nombre de zones voisines. Chaque zone a aussi un niveau d'eau présente en surface (en mm). Chaque zone dispose d'une méthode qui décrit la façon dont elle reçoit de la pluie, cette méthode a comme paramètre la quantité de pluie tombée sur la zone, en millimètres. Chaque zone a aussi une méthode qui décrit comment elle distribue l'eau vers les zones voisines. De plus, chaque zone doit pouvoir s'afficher en utilisant l'interface suivante :

```
public interface Affichable{
  public java.awt.Color getColor();
  public java.awt.Shape getShape();
}
```

a. Écrire du code objet pour représenter les zones.

Il existe deux types de zones (pour simplifier) :

- les zones construites, qui s'affichent en gris et n'absorbent pas du tout la pluie : quand elles reçoivent x millimètres de pluie, le niveau de l'eau sur la zone augmente de x.
- les zones humides, qui s'affichent en vert, ont une certaine capacité de stockage de la pluie dans le sol (en m^3). Quand une zone humide reçoit x millimètres de pluie, l'eau commence par remplir le sol[1]. Puis quand la capacité de stockage de la zone est atteinte, l'eau s'accumule alors à la surface et le niveau d'eau de la zone humide augmente.

b. Écrire du code objet pour représenter les zones construites et les zones humides.

Exercice 3.11 : on veut réaliser un programme de gestion d'une bibliothèque. Un livre possède un titre et un ensemble de thèmes (qui sont des mots-clés). Une bibliothèque est une liste de livre et possède un nom.

a. Écrivez du code objet pour représenter les livres et les bibliothèques.

On veut pouvoir rechercher des livres en fonction d'un thème. On va utiliser un mécanisme d'itérateur, c'est-à-dire des objets qui implémentent l'interface suivante :

```
public interface Iterator<E>{
  // renvoie true s'il existe encore un élément à parcourir
  public boolean hasNext();
  // renvoie l'élément courant et passe au suivant
  public E next();
}
```

1 rappel : s'il tombe x mm de pluie sur une surface de y km², la quantité d'eau totale reçue sur la zone, en m^3, est x*y*1000

retour au sommaire

Il faut pouvoir créer un itérateur avec un thème, et le parcours d'une bibliothèque avec un tel itérateur ne va parcourir que les livres ayant pour thème celui de l'itérateur.

b. Écrire du code objet pour représenter les itérateurs sur bibliothèque

Exercice 4.1 : indiquez si ce code compile et, s'il ne compile pas, quelle(s) erreur(s) va afficher le compilateur.

```
package p;
interface I {
  public int getI();
}

package q;
abstract class C implements I {
  private int i;
  C(int i) {
    this.i = i;
  }
}

package q;
abstract class D extends C {
  D(int x) {
    super(x);
    this.i = 0;
  }
}
```

Exercice 4.2 : ces classes compilent t-elles et, si non, quelle(s) erreur(s) le compilateur va t-il indiquer?

```
public class A {
  public int i;
  private int a;
  protected int b;
  public void afficher() {
    System.out.println(i+a+b);
  }
}

public class B extends A {
  public void afficher() {
    System.out.println(i+a+b);
  }
}
```

retour au sommaire

```
class D {
  public A a = new A();
  public void afficher() {
    System.out.println(a.i+a.a+a.b);
  }
}

public class E {
  public B b = new B();
  public void afficher() {
    b.afficher();
  }
}

public class F {
  public B b = new B();
  public void afficher() {
    System.out.println(b.i+b.a+b.b);
  }
}
```

Exercice 4.3 : Indiquez quelles lignes de la méthode main sont incorrectes.

```
package toto;
public class C {
  public C tutu;
  private int i = 0;
  int j = 1;
  static protected int k = 2;
}

package toto.titi;
public class Test extends toto.C {
  public static void main(String[] t) {
    toto.C m = new toto.C();
    m.tutu = m;
    m.tutu.i = 4;
    m.j = 5;
    m.tutu.k = 6;
  }
}
```

Exercice 4.4 : indiquez quelles lignes du programme suivant sont incorrectes.

```
package plante;
```

retour au sommaire

```
interface Plante{
  double getDureeVie();
  int getTaille();
}

package plante;
public class Paquerette implements Plante{
  static int petales=20;
  double getDureeVie(){return Double.POSITIVE_INFINITY;}
  public int getTaille(){return 15;}
}

package toto;
public class Test{
  public static void main(String[] t){
    System.out.println(plante.Paquerette.petales);
  }
}
```

Exercice 4.5 : les trois classes suivantes compilent t-elles? Si non, indiquez les erreurs. On suppose que les classes sont définies dans des fichiers correctement stockés (A et B sont dans des fichiers A.java et B.java situés dans un répertoire p1 et C dans un fichier C.java situé dans un répertoire p2).

```
package p1;
class A{
  protected int i;
  private int j;
  public void m(){System.out.println("Somme : "+(i+j));}
}

package p1;
class B extends A{
  public void m(){System.out.println("Produit : "+(i*j));}
}

package p2;
import p1.B;
class C extends B {
  protected void m(){System.out.println("Carré : "+(i*i));}
}
```

Exercice 4.6 : les classes suivantes sont placées dans différents répertoires : B et C sont dans un répertoire p1, A est dans un répertoire p3 qui est un sous-répertoire de p1 et D est dans un répertoire p2.

```
package p1.p3;
```

retour au sommaire

```
public class A{
  protected int i;
  int j;
  void m(){System.out.println("Je suis un A");}
}

package p1;
class B extends p2.D{
  private int i=3;
  void m(){System.out.println("Je suis un B");}
}

package p1;
class C extends p1.p3.A{
  static protected int i = 1;
}

package p2;
public class D extends A{
  protected void m(){System.out.println("Je suis un D");}
}
```

Les classes compilent-elles et si non, quelles sont les erreurs?

Exercice 4.7 : les classes suivantes sont placées dans différents répertoires : une classe D dans un répertoire p2, trois classes B, C et D dans un répertoire p1.

```
package p2;
public class D{
  private int i=2;
  protected void m(){System.out.println("Je suis un D de p2");}
}

package p1;
public class D{
  int i=3;
  protected void m(){System.out.println("Je suis un D de p1");}
}

package p1;
import p2.*;
class C extends D{
  static protected int i=1;
  public void m(){super.m(); System.out.println("Je suis un C");}
  public static void main(String[] toto){
    C c = new C(); B b = new B(); c.m(); b.m();
```

retour au sommaire

```
  }
}

package p1;
class B extends p2.D{
  private int i=3;
  public void m(){super.m(); System.out.println("Je suis un B");}
}
```

Les classes compilent t-elles? Si non, indiquez les erreurs affichées par le compilateur et proposez des corrections. À quel affichage conduit l'exécution du programme (éventuellement corrigé)?

Exercice 4.8 :

```
class A{
  int i;
  int getI(){return this.i;}
}

class B{
  int j;
  int getJ(){return this.j;}
}

class C extends A{
  int k;
  int getK(){return this.k;}
}

class D extends B{
  int l;
  int getL(){return this.l;}
}
```

Modifiez ces 4 classes de manière à ce que les propriétés suivantes soient vérifiées *en même temps* :

- l'attribut i est visible dans A et B mais pas dans C
- l'attribut k n'est visible que dans C
- la méthode getI est visible dans A, B et C mais pas dans D
- la méthode getL est visible dans toutes les classes

Exercice 4.9 : le code suivant compile t-il? Si non, indiquez les erreurs. Proposez éventuellement des corrections et indiquez quel sera l'affichage lors de l'exécution une fois ces corrections effectuées.

```
public class A{
  protected int a = 1;
  public A(){
    System.out.println("A");
  }
  public void a(){
    System.out.println(a);
  }
}

public interface C{
  public final int a = 3;
  public void a();
```

retour au sommaire

```
}

public class B extends A implements C{
  private int a = 2;
  public B(){
    System.out.println("B");
  }
  public void a(){
    System.out.println(a);
  }
  public static void main(String[] t){
    A o1 = new A();
    A o2 = new B();
    B o3 = new B();
    System.out.println(o1.a);
    System.out.println(o2.a);
    System.out.println(o3.a);
    o1.a();
    o2.a();
    o3.a();
  }
}
```

Exercice 5.1 : on veut écrire un programme pour gérer des graphes d'entiers (chaque sommet est une valeur entière). On veut pouvoir représenter des graphes de n'importe quel type (orienté ou non, simple ou non, etc) mais tout graphe doit cependant offrir des méthodes pour :

- ajouter un entier comme sommet du graphe
- supprimer un sommet du graphe
- ajouter une arête entre deux sommets donnés
- supprimer une arête entre deux sommets donnés
- savoir s'il existe une arête entre deux sommets donnés

NB : La façon dont ces méthodes vont être implémentées peut dépendre du type de graphe. Par exemple, ajouter une arête entre deux sommets d'un graphe simple n'est possible que si les deux sommets ne sont pas déjà liés par une arête, alors qu'on peut toujours ajouter une arête dans un multi-graphe.

a. Proposez du code objet pour représenter les sommets et les arêtes d'un graphe d'entiers.

b. Proposez du code objet pour représenter les graphes d'entiers en général.

c. Proposez du code objet pour représenter les graphes non orientés. Ce code doit permettre d'ajouter ou de supprimer un sommet et de savoir s'il existe une arête entre deux sommets donnés.

retour au sommaire

d. Proposez du code objet pour représenter les graphes simples non orientés, c'est-à-dire les graphes non orientés où il existe au plus une arête entre deux sommets. Ce code doit permettre d'ajouter et de supprimer une arête entre deux sommets donnés.

Exercice 5.2 : on veut écrire un programme gérant le coût des enseignants d'une université. Chaque enseignant a un nom, un prénom et un certains nombre d'heures de cours assurées dans l'année. Il existe plusieurs catégories d'enseignants.

Les enseignants-chercheurs sont payés avec un salaire fixe (2000 euros par mois pour simplifier) plus le revenu des heures complémentaires (40 euros de l'heure). Les heures complémentaires sont les heures effectuées au-delà des 192h. Ainsi, un enseignant-chercheur qui donne 200h de cours dans l'année recevra 2000*12+8*40=24320 euros sur l'année.

Les vacataires sont des enseignants venant d'un organisme extérieur à l'université. Cet organisme est décrit par une chaine de caractères. Les vacataires sont payés 40 euros de l'heure pour toutes leurs heures.

Les étudiants de 3e cycle (doctorants) peuvent assurer des cours et sont payés 30 euros de l'heure, mais ne peuvent dépasser 96h de cours. Un doctorant qui fait plus de 96h ne recevra que le salaire correspondant à 96h de cours. Finalement, sur tous les salaires versés, on applique des charges égales à un certain pourcentage du salaire, ce pourcentage est le même pour tous les enseignants (par exemple 100%, ce qui signifie que le montant des charges est égal au montant des salaires). On veut, pour chaque enseignant, pouvoir connaitre ce qu'il coûte à l'université (salaire + charges).

Proposez du code Java qui représente ces informations et soit le plus conforme aux principes de la programmation objet.

Exercice 5.3 : on veut développer un logiciel pour équiper un robot manipulateur dans un laboratoire de chimie. Une classe représentant les contenants (récipients de type tube à essai, béchers, etc) a déjà été écrite (elle utilise une classe de l'API Java représentant les formes tridimensionnelles).

```
import javafx.scene.shape.Shape3D;
public abstract class Contenant extends Shape3D{
  private int x, y, z, opacite; // (x,y,z)=position, opacite de 0 à 100%
  public Contenant(int x, int y, int z, int opacite){
    super();
    this.x = x;
    this.y = y;
    this.z = z;
    this.opacite = opacite;
  }
  // renvoie la forme occupée dans le contenant par un liquide de volume v
  public abstract Shape3D getSubShape(double v);
}
```

Il faut représenter dans le logiciel les produits chimiques manipulées. Certains produits sont déformables (liquides et gaz), c'est-à-dire qu'on peut changer leur forme (Shape3D). D'autres produits (les solides) ont une forme fixe. Les liquides et gaz ne se déforment pas de la même façon : un liquide occupe un volume fixe alors qu'un gaz occupe tout l'espace disponible dans un contenant (on peut

retour au sommaire

supposer qu'il existe une méthode double getVolume() dans la classe Shape3D). Un liquide a une viscosité (taux de résistance à la déformation). Tous les produits chimiques ont un nom, et un poids (réel).

a. Proposez du code objet pour représenter les différents produits chimiques

Il faut pouvoir ajouter des produits dans des contenants.

b. Ajoutez du code pour pouvoir ajouter des produits dans n'importe quel contenant en respectant les changements éventuels de formes (on suppose qu'il est toujours possible de placer un produit dans un contenant, quelques soient leurs volumes et formes respectifs).

Exercice 5.4 : on veut écrire un programme pour un jeu vidéo sur le thème du Seigneur des Anneaux de Tolkien. Un programmeur a déjà écrit la classe Personnage qui suit, et il est impossible de la modifier.

```
public abstract class Personnage{
  private String nom;
  private int pointsVie, x, y, vitesse, sous;
  public Personnage(String n, int x, int y, int v){
    this.nom = n;
    this.x = x; this.y = y;
    this.pointsVie = 100;
    this.vitesse = v;
    this.sous = 0;
  }
  public int getSous(){return this.sous;}
  public void setSous(int s){this.sous = s;}
  public int getPointsVie(){return this.pointsVie;}
  public void setPointsVie(int pv){this.pointsVie = pv;}
  /** Le personnage se deplace dans la direction (dx,dy) durant un temps t. */
  public void seDeplacer(int dx, int dy, int t){
    this.x = this.x + dx*this.v*t/Math.sqrt(dx*dx+dy*dy);
    this.y = this.y + dy*this.v*t/Math.sqrt(dx*dx+dy*dy);
  }
  public abstract String parler();
}
```

Il faut représenter dans le programme différents types de personnages : les humains qui se déplacent à 5km/h et parlent en disant "Bonjour", les elfes qui se déplacent à 7km/h et parlent en disant "Eldarie" et les nains qui se déplacent à 2km/h et parlent en disant "Groumpf".

a. Écrire du code permettant de représenter les humains, elfes et nains.

On veut également représenter les compétences des personnages :

retour au sommaire

- les guerriers sont des personnages qui peuvent attaquer un autre personnage et possèdent une certaine force.
- les magiciens sont des personnages qui peuvent lancer un sort sur un autre personnage et possèdent un certain niveau de magie.
- les voleurs sont des personnages qui peuvent voler les sous d'un autre personnage et possèdent une certaine dextérité.

Un personnage donné peut avoir plusieurs compétences (par exemple être à la fois guerrier et magicien).

b. Écrire du code qui permettent de représenter ces compétences dans le programme.

c. Écrire du code permettant de représenter les personnages nains qui sont des guerriers. Un nain guerrier attaque un autre personnage en lui retirant un nombre de points de vie égal à sa propre force.

d. Écrire du code permettant de représenter les personnages nains qui sont à la fois guerriers et voleurs. Un nain voleur vole un autre personnage en lui retirant un nombre de sous égal à sa propre dextérité (ou moins si le personnage volé possède un nombre de sous inférieur à la dextérité du nain).

Exercice 5.5 : on veut écrire un programme pour un jeu vidéo sur le thème du Seigneur des Anneaux de Tolkien. Un programmeur a déjà écrit la classe Personnage qui suit.

```
abstract class Personnage{
  private String nom;
  private int pointsVie, x, y, v, sous;
  private boolean visible = true;
  public Personnage(String n, int x, int y, int v){
    this.nom = n;
    this.x = x; this.y = y;
    this.pointsVie = 100;
    this.v = v;
    this.sous = 0;
  }
  public int getSous(){return this.sous;}
  public void setSous(int s){this.sous = s;}
  public int getPointsVie(){return this.pointsVie;}
  public void setPointsVie(int pv){this.pointsVie = pv;}
  public void setVisible(boolean b){this.visible = b;}
  public void seDeplacer(int dx, int dy, int t){
    this.x = (int) (this.x + dx*this.v*t/Math.sqrt(dx*dx+dy*dy));
    this.y = (int) (this.y + dy*this.v*t/Math.sqrt(dx*dx+dy*dy));
  }
  public abstract String parler();
}
```

retour au sommaire

On veut représenter les anneaux magiques que peuvent porter les personnages, mais avec comme contrainte qu'un anneau ne peut exister qu'en lien avec son porteur, qui est un personnage (un anneau a toujours le même porteur tout au long du jeu). Il existe plusieurs types d'anneaux, qui ont des effets différents lorsqu'on les passe au doigt, mais on veut imposer que chaque anneau ait un nom et puisse être passer au doigt de son porteur, ou enlevé du doigt de son porteur.

a. Écrivez du code (en modifiant éventuellement la classe Personnage) permettant de représenter les anneaux.

b. Écrivez du code qui permet de représenter l'anneau de Sauron. Cet anneau est d'un type particulier qui permet à son porteur de devenir invisible quand il le passe au doigt, et de redevenir visible quand il l'enlève.

c. Écrivez du code permettant de représenter les hobbits, qui sont des personnages particuliers. Les hobbits parlent en disant "Belle journée ma foi" et se déplacent à 5km/h. Ecrire une méthode main qui crée un hobbit appelé Frodon, situé aux coordonnées (200,100) et un anneau de Sauron dont Frodon est le porteur.

d. L'anneau de Sauron est unique (c'est le seul de son type). Écrivez du code permettant de rendre l'anneau de Sauron unique dans le jeu. Attention : il n'est pas possible de définir d'attribut ou de méthode static dans une classe interne.

Exercice 5.6 : on veut écrire un programme de jeu vidéo qui a pour cadre le monde du Seigneur des Anneaux de Tolkien. Une classe représentant les personnages du jeu a déjà été écrite, elle permet de gérer les positions et déplacement des personnages.

```
public abstract class Personnage{
  protected String nom;
  protected int pointsVie, connaissances, x, y, v;
  public Personnage(String n, int x, int y, int v){
    this.nom = n;
    this.x = x; this.y = y;
    this.pointsVie = 100;
    this.connaissances = 0;
    this.v = v;
  }
  public int getPointsVie(){return this.pointsVie;}
  public void setPointsVie(int pv){this.pointsVie = pv;}
  public int getConnaissances(){return this.connaissances;}
  public void setConnaissances(int co){this.connaissances = co;}
  public void seDeplacer(int dx, int dy, int t){
   this.x = (int) (this.x + dx*this.v*t/Math.sqrt(dx*dx+dy*dy));
    this.y = (int) (this.y + dy*this.v*t/Math.sqrt(dx*dx+dy*dy));
  }
  public abstract String parler();
}
```

retour au sommaire

On veut pouvoir gérer dans le jeu des objets qui peuvent être des armes, des documents (parchemins, grimoires, messages, etc) ou d'autres objets (bijoux, clés, etc). Un objet a un nom et un prix.

a. On veut pouvoir représenter les personnages humanoides (humains, nains, elfes, hobbits, etc) qui, en plus de pouvoir se déplacer, peuvent acquérir des objets ou s'en séparer. On veut aussi que chaque objet puisse être donné par un humanoide à un autre humanoide. *Sans modifier la classe* Personnage, proposez du code permettant de représenter les objets et les humanoides.

b. On veut que chaque arme ait une certaine puissance et puisse être utilisée par son propriétaire contre un autre personnage (humanoide ou non). Une arme ne peut pas être utilisée par un personnage autre que son propriétaire. Quand un personnage utilise une arme contre un autre, il retire à ce dernier un nombre de points de vie égal à la puissance de l'arme. Proposez du code pour représenter les armes.

c. On veut que chaque document contienne une certaine quantité de connaissances et puisse être lu par son propriétaire. Un document ne peut pas être lu par un autre personnage que son propriétaire. Quand un personnage lit un document, ses connaissances sont augmentées d'une quantité égale à la quantité de connaissances du document. Mais cet effet ne peut avoir lieu qu'une seule fois pour un même document : si un personnage lit un document qu'il a déjà lu, ses connaissances ne sont pas modifiées. Proposez du code permettant de représenter les documents.

On veut maintenant représenter les monstres (troll, etc), qui ne peuvent utiliser des objets. L'interface suivante a été définie pour cela, ne peut être modifiée, et doit obligatoirement être utilisée pour représenter les monstres.

```
public interface Monstre{
  public void attaque(Personnage p);
  public int getPuanteur();
}
```

d. On veut représenter les trolls, qui sont des personnages (pouvant se déplacer) et en même temps des monstres. Un troll a une certaine force et peut attaquer un personnage en lui faisant perdre un nombre de points de vie égal à sa force plus le dixième de sa puanteur. Mais un troll n'attaque jamais un autre troll. Proposez du code permettant de représenter les trolls.

Exercice 5.7 : on veut écrire un programme pour un jeu vidéo sur le thème de Star Wars. Un programmeur a déjà écrit la classe Personnage qui suit.

```
public abstract class Personnage{
  protected String nom;
  protected int pointsVie, x, y, v, force;
  public Personnage(String n, int x, int y, int v, int force){
    this.nom = n; this.x = x; this.y = y;
    this.pointsVie = 100; this.v = v; this.force = force;
  }
  public String getNom(){return this.nom;}
```

retour au sommaire

```
  public int getX(){return this.x;}
  public int getY(){return this.y;}
  public int getV(){return this.v;}
  public void setX(int x){this.x = x;}
  public void setY(int y){this.y = y;}
  public int getForce(){return this.force;}
  public void estBlesse(){this.pointsVie--;}
  /** Le personnage bouge dans la direction (dx,dy) durant un temps t. */
  public void seDeplacer(int dx, int dy, int t){
    this.x = (int) (this.x + dx*this.v*t/Math.sqrt(dx*dx+dy*dy));
    this.y = (int) (this.y + dy*this.v*t/Math.sqrt(dx*dx+dy*dy));
   }
  public abstract String parler();
}
```

On veut représenter les Jedi, qui sont des sortes de personnages qui parlent en disant "Que la force soit avec vous" et qui ont tous une vitesse (attribut v de la classe Personnage) de 5km/h.

a. Écrivez une classe Jedi pour représenter ces personnages.

b. Écrivez une méthode principale qui (1) crée un Jedi nommé Maitre Yoda, situé aux coordonnées (45,56), avec une force de 120 et (2) fait parler ce personnage en écrivant ce qu'il dit à l'écran.

Un Jedi a toujours un sabre laser, qui a une couleur et qu'on peut allumer ou éteindre. Un sabre laser ne peut exister qu'en lien avec un Jedi et un sabre est possédé par le même Jedi tout au long du jeu. Pour la couleur, on peut utiliser la classe java.awt.Color.

c. Écrivez du code (en modifiant éventuellement la classe Jedi) permettant de représenter les sabres laser dans le programme. Il faut modifier le constructeur de la classe Jedi pour qu'un sabre soit créé quand on crée un Jedi.

d. Modifiez et complétez la méthode principale écrite auparavant pour que Maitre Yoda soit créé avec un sabre laser de couleur verte et qu'il allume son sabre.

On veut maintenant représenter dans le programme les soldats de l'Empire, qui sont tous des clones d'un même personnage nommé Jango Fett. Un programmeur a déjà écrit l'interface ci-dessous qui instancie le personnage de Jango Fett et fixe le comportement des soldats clonés qui peuvent se déplacer et tirer sur un autre personnage. La classe Aventurier est une classe qui hérite de Personnage.

```
public interface Clone{
  public static final Aventurier DJANGO_FETT = new Aventurier("Jango Fett");
  public void tireSur(Personnage p);
  public void seDeplacer(int dx, int dy, int t);
}
```

retour au sommaire

e. Écrivez une classe SoldatClone qui représente les soldats de l'Empire avec un constructeur qui crée un clone de Jango Fett (c'est-à-dire un soldat ayant les mêmes caractéristiques que Jango Fett). Les soldats n'ont pas de nom et ne disent rien (leurs noms et paroles sont des chaines de caractères vides). Quand un soldat tire sur un personnage, le personnage est blessé (voir la méthode estBlesse dans la classe Personnage).

Exercice 5.8 : on veut écrire un programme de jeu vidéo qui a pour cadre l'univers de Star Wars. Une classe représentant les personnages du jeu a déjà été écrite.

```
public abstract class Personnage{
  protected String nom;
  protected int pointsVie, x, y, v, precision; // precision de 0 à 100
  public Personnage(String n, int x, int y, int v, int precision){
    this.nom = n;
    this.x = x; this.y = y;
    this.pointsVie = 100;
    this.v = v;
    this.precision = precision;
  }
  public int getPointsVie(){return this.pointsVie;}
  public void setPointsVie(int pv){this.pointsVie = pv;}
  public int getPrecision(){return this.precision;}
  public void seDeplacer(int dx, int dy, int t){
    this.x = (int) (this.x + dx*this.v*t/Math.sqrt(dx*dx+dy*dy));
    this.y = (int) (this.y + dy*this.v*t/Math.sqrt(dx*dx+dy*dy));
  }
}
```

On veut pouvoir gérer des armes dans le jeu. Une arme a une certaine puissance (nombre entier) et peut être possédée par un personnage. Elle permet de tirer sur une cible. Tout personnage ou objet peut être une cible. Une cible doit avoir un certain comportement lorsqu'elle est touchée par une arme, par exemple être détruite s'il s'agit d'un objet, perdre des points de vie s'il s'agit d'un personnage, être abimé s'il s'agit d'un vaisseau spatial.

a. Proposez du code objet pour représenter les armes et les cibles dans le jeu, en modifiant si nécessaire la classe Personnage.

b. On veut maintenant représenter les vaisseau spatiaux, qui sont à la fois des armes et des cibles. Un vaisseau est dans un certain état de 100 (état neuf) à 0 (détruit). Il a aussi un certain blindage (entier de 1 à 5). Quand un vaisseau est touché par une arme, son état diminue d'une valeur égale à la puissance de l'arme divisée par son propre blindage. Proposez du code pour représenter les vaisseaux spatiaux.

c. Il existe deux catégories de vaisseaux : les drones, non pilotés et les vaisseaux pilotés par un personnage. Un drone a une précision intrinsèque alors qu'un vaisseau piloté tire sur les cibles avec la précision de son pilote. Dans les deux cas, quand un vaisseau tire sur une cible, il la touche si un entier

retour au sommaire

généré aléatoirement est inférieur à la précision du vaisseau (ou de son pilote). Proposez du code pour représenter les vaisseaux pilotés et les drones.

d. On veut représenter l'Etoile de la Mort, un vaisseau spatial piloté particulier qui a une puissance de 1000 et un blindage de 5. Il ne peut exister qu'une seule Etoile de la Mort dans le jeu. Proposez du code pour représenter l'Etoile de la Mort.

Exercice 5.9 : on veut écrire un programme pour un jeu vidéo sur le thème de Star Wars. Un programmeur a déjà écrit la classe Personnage qui suit :

```
public abstract class Personnage{
  protected String nom;
  protected int pointsVie, force;
  public Personnage(String n, int f){
     this.nom = n;
     this.force = f;
     this. pointsVie = 100;
  }
  public void mange(){if(this.pointsVie <100) this.pointsVie ++;}
  public abstract void rencontre(Personnage p);
}
```

On veut représenter dans le programme les personnages du coté obscur de la Force, qui tuent tous les personnages qu'ils rencontrent s'ils sont de force strictement inférieure à la leur. On veut aussi représenter les personnages du coté lumineux de la Force, qui, lorsqu'ils rencontrent un autre personnage, ne le tuent que s'il est du coté obscur de la Force et qu'il a une force strictement inférieure à la leur. Tuer un personnage consiste à mettre ses points de vie à 0, et un personnage ne peut en tuer un autre que si ses points de vie sont supérieurs à 10.

a. Ecrivez du code objet pour représenter ces personnages du coté obscur et du coté lumineux avec leurs différentes façon de traiter les gens qu'ils rencontrent.

b. Ecrivez une méthode principale qui crée un personnage du coté obscur nommé "Dark Vador" avec une force de 54, un personnage du coté lumineux nommé "Luc Skywalker" avec une force de 47. Faites tuer "Luc Skywalker" par "Dark Vador".

On veut maintenant gérer le positionnement des entités (personnages et objets) dans le jeu. Pour cela, on doit, pour chaque entité, pouvoir connaitre sa localisation (donnée par deux coordonnées entières) et pouvoir la déplacer (un déplacement est représente par deux entiers qui donnent les déplacements en abscisse et ordonnée).

c. Ecrivez du code objet permettant de gérer le positionnement des entités, au besoin en complétant la classe Personnage (en ce cas, indiquer comment elle est modifiée).

retour au sommaire

d. Ecrivez du code objet permettant de créer des Jedi, qui sont des personnages du coté lumineux, avec une couleur de sabre laser, et pouvant déplacer par la pensée n'importe quelle entité. Un Jedi ne peut cependant effectuer un déplacement que si ses points de vie sont supérieurs à 10.

e. On veut que les personnages puissent passer d'un coté à l'autre de la force. Sans forcément écrire tout le code nécessaire, expliquez comment il faudrait modifier le code écrit pour permettre cela.

Exercice 5.10 : on veut écrire un programme pour manipuler des tableaux associatifs. Ces tableaux doivent permettre de stocker des éléments de type ObjetNomme (type défini par l'interface qui suit). Les indices doivent être des objets qu'on puisse comparer, et doivent donc être de type Ordonnable. On veut pouvoir proposer plusieurs implémentations pour de tels tableaux (paires d'objets, table de hachage, etc).

```
public interface ObjetNomme{
  public String getNom();
}

public interface Ordonnable{
  // retourne un entier <0 si this est plus petit que o, >0 si this
  // est plus grand que o et 0 si this==o
  public int compare(Ordonnable o);
}
```

Un tableau associatif doit permettre :
- d'ajouter un élément en précisant son indice (la méthode doit renvoyer true si on peut ajouter l'élément, false sinon)
- de récupérer un élément en donnant son indice (la méthode doit renvoyer null si l'élément n'est pas présent)
- de supprimer un élément donné (la méthode doit renvoyer true si l'élément est présent, false sinon)
- de savoir si un élement donné est dans le tableau (la méthode renvoie un booléen)
- de connaitre le nombre d'éléments stockés

a. Écrivez du code objet pour représenter les tableaux associatifs.

b. Une façon (très peu efficace) d'implémenter les tableaux associatifs est d'utiliser une liste de paires (Ordonnable/ObjetNomme). Ajoutez un élément revient à ajouter une paire à la liste, etc. Écrivez du code objet qui représente les tableaux associatifs implémentés par des paires.

c. On suppose qu'il existe deux classes concrètes A et B, implémentant respectivement les interfaces Ordonnable et ObjetNomme, et dotées de constructeurs sans paramètre. Écrivez une méthode main qui crée un tableau associatif implémenté par des paires, ajoute un élément de type B indicé par A, affiche le nombre d'éléments stockés dans le tableau, puis retire l'élément du tableau.

d. On veut introduire un type particulier de tableau associatif ayant une taille limitée. Dans ces tableaux, l'ajout d'un élément ne se fait que si le nombre d'éléments stockés est inférieur ou égal à la

retour au sommaire

taille maximum du tableau. Écrivez du code objet pour représenter les tableaux associatifs à taille limitée.

e. Écrivez du code objet pour représenter les tableaux associatifs à taille limitée implémentés par des paires.

Exercice 5.11 : on veut écrire un programme pour représenter des données manipulées selon le modèle des bases de données relationnelles. Tout le code doit appartenir au même paquetage database. Les valeurs manipulées doivent toutes être comparables, au sens de l'interface Comparable donnée plus bas, elles doivent toutes avoir une méthode String toString() et une méthode de copie qui renvoie une copie de la valeur. Les clés sont des valeurs particulières de type entier. À partir d'une clé, on doit pouvoir obtenir la clé suivante (ie l'entier suivant).

```
public interface Comparable<Type>{
  // retourne un entier négatif, positif ou nul selon que this est
  // plus petit, plus grand ou égal à o
  public int compareTo(Type o);
}
```

a. Écrire du code objet pour représenter les valeurs et les clés.

Il faut représenter les tables de données. Une table peut être créée à partir d'une liste d'identifiants (chaines de caractères) de ses colonnes. On doit pouvoir aussi ajouter une ligne à une table, en spécifiant les valeurs de chaque colonne, et enlever la dernière ligne d'une table.

b. Écrire du code objet qui représente les tables.

Les tables indexables sont des tables particulières qui ont une première colonne contenant des valeurs de type clé. Dans ces tables, on peut changer la valeur d'une case en spécifiant la clé de la ligne et l'identifiant de la colonne. On peut aussi récupérer la valeur d'une case à partir d'une clé et d'un identifiant.

c. Écrire du code objet qui représente les tables indexables.

Exercice 5.12 : on veut écrire un programme pour gérer et afficher des formes géométriques. L'interface graphique d'affichage a déjà été codée et elle peut afficher tout objet qui implémente l'interface suivante :

```
public interface Drawable{
  public final static int minX=0, maxX=100, minY=0, maxY=100;
  public void draw(Graphics g);
}
```

Chaque forme géométrique doit être dessinable sur l'interface graphique et possède un identifiant (entier), une couleur (on peut utiliser la classe java.awt.Color), une position (coordonnées 2D entières). Les identifiants et couleurs sont fixés à la création des formes et ne doivent pas changer

retour au sommaire

au cours de l'exécution du programme. Par contre, chaque forme doit pouvoir être déplacée. Les positions des formes doivent cependant toujours restées entre les valeurs min et max fixées par l'interface Drawable.

a. Écrivez une classe pour représenter ces formes.

On veut représenter les formes 2D qui sont des formes ayant en plus une largeur et une hauteur. Parmi ces formes 2D, on distingue les rectangles et les cercles.

NB: les méthodes suivantes, issues de la classe Graphics, peuvent vous être utiles :

- *public void setColor(java.awt.Color c)* // change la couleur courante de dessin
- *public void fillRect(int x, int y, int width, int height)* // dessine, avec la couleur courante, un rectangle de dimensions width*height dont le coin en haut à gauche est à la position (x,y)
- *public void fillOval(int x, int y, int width, int height)* // dessine, avec la couleur courante, un ovale de dimensions width*height dont le coin en haut à gauche est à la position (x,y)

b. Proposez du code objet pour représenter ces formes 2D. Écrire une méthode main qui crée un rectangle et un cercle et les dessine sur un objet Graphics qu'on peut supposer exister.

On veut maintenant représenter des formes 3D (mais qui seront dessinées en 2D en perspective). Une forme 3D est une forme qui possède une largeur, une hauteur et une profondeur (entières). Parmi ces formes 3D, on distingue les cubes. On peut supposer qu'il existe dans la classe Graphics une méthode fill3DRect(int x, int y, int width, int height, int depth) qui dessine, avec la couleur courante, un parallèlepipède de dimensions width*height*depth dont le coin en haut à gauche est à la position (x,y).

c. Écrivez du code pour représenter les formes 3D en général et les cubes en particulier.

Exercice 1.1 - solution : le code ne compile pas car la variable toto à la 9e ligne n'est pas static et doit donc être accéder via un objet (par exemple t1.toto).

Exercice 1.2 - solution : le code compile, il affiche 12, 12, 12 ... et finit par planter à cause d'une récursion infinie.

Exercice 1.3 - solution : 4e ligne, il est impossible d'affecter une valeur à this. On peut corriger avec this.e = e;. L'exécution affiche alors "Bonjour le monde" puis "LLUN" puis "Bonjour le monde" puis "NULL".

Exercice 1.4 - solution : le code compile, il affiche 13,14.

Exercice 1.5 - solution : le code compile, il affiche 1,1,1,2.

Exercice 1.6 - solution : le code compile, il affiche 3,3.

Exercice 1.7 - solution : le code compile et affiche 11.

retour au sommaire

Exercice 1.8 - solution : Bonjour, Hello, Bonjour, Bonjour, false, false

Exercice 1.9 - solution :
a.i=5, b.i=1 ou 2
a.i=5, b.i=3 ou 5

Exercice 1.10 - solution : le code compile sans problème.

Exercice 1.11 - solution :

```
class PoupeeRusse {
  int taille;
  boolean ouverte;
  PoupeeRusse dans, contient;
  PoupeeRusse(int taille) {
    this.taille = taille;
    this.ouverte = false;
  }
  void ouvrir() {
    if(!this.ouverte && this.dans==null) this.ouverte = true;
  }
  void fermer() {
    if(this.ouverte && this.dans == null) this.ouverte = false;
  }
  void placerDans(PoupeeRusse p) {
    if(!this.ouverte && this.dans == null && p.ouverte && p.contient == null && p.taille> this.taille)
{
      this.dans = p;
      p.contient = this;
    }
  }
  void sortirDe(PoupeeRusse p) {
    if(p.ouverte && p.contient == this) {
      this.dans = null;
      p.contient = null;
    }
  }
}
```

Exercice 1.12 - solution :

```
class  Montre{
  int  heure,  min;
  Personne  p;  //  pour  2.3
  Montre(int  h,  int  m){
    this.heure  =  h;
    this.min  =  m;
```

retour au sommaire

```
  }
  Montre(Montre m){
    this.heure = m.heure;
    this.min = m.min;
  }
  void avance(){
    if(this.min==59){
      if(this.heure == 23) this.heure = 0;
      else this.heure = this.heure++;
      this.min = 0;
    }
    else this.min++;
  }
  public static void main(String[] toto){
    Montre m = new Montre(13,45);
    Montre n = new Montre(m);
  }
}

class Personne{
  String nom;
  Montre m;
  Personne(String nom){
    this.nom = nom;
    this.m = null;
  }
  boolean porteMontre(Montre m){
    if(m.p!=null && this.m==null){
      this.m = m;
      m.p = this; // pour 2.3
      return true;
    }
    else return false;
  }
  Montre enleveMontre(){
    Montre m = this.m;
    this.m = null;
    if(m!=null) m.p=null; // pour 2.3
    return m;
  }
  String donneHeure(){
    if(this.m != null) return this.m.heure+"h"+this.m.min;
    else return "";
  }
}
```

Exercice 1.13 - solution :

retour au sommaire

```
class Plat{
  String nom;
  ArrayList<Ingredient> ingredients;
  Plat(String n){
    this.nom = n;
    this.ingredients = new ArrayList<Ingredient>();
  }
  String getNom(){
    return this.nom;
  }
  ArrayList<Ingredient> getIngredients(){
    return this.ingredients;
  }
  void addIngredient(Ingredient i){
    this.ingredients.add(i);
  }
  // pour b.
  protected boolean equals(Object o){
    if(o instanceof Plat){
      for(Ingredient i:this.ingredients){
        if(!((Plat) o).ingredients.contains(i)) return false;
      }
      return this.ingredients.size() ==
              ((Plat) o).ingredients.size();
    }
    else return false;
  }
  public static void main(String[] toto){
    Plat p = new Plat("Choucroute");
    p.addIngredient(new Ingredient("choucroute","cuite",500,"g"));
    p.addIngredient(new Ingredient("lard","cuit_entier",150,"g"));
    p.addIngredient(new Ingredient("saucisse","cuite_entiere",2,
                    "cardinalite"));

  }
}

// pour b.
class Ingredient{
   ...
  protected boolean equals(Object o){
    return (o instanceof Ingredient) &&
        this.nom_aliment.equals(((Ingredient) o).nom_aliment) &&
        this.etat.equals(((Ingredient) o).etat);
  }
  ...
```

retour au sommaire

```
}

class IngredientACuire extends Ingredient{
  int temperature;
  IngredientACuire(String n, String e, int q, String unite, int t){
    super(n,e,q,unite);
    this.temperature = t;
  }
  void cuire(){
    this.etat = "cuit";
  }
}

class IngredientADecouper extends Ingredient{
  IngredientADecouper(String n, String e, int q, String unite){
    super(n,e,q,unite);
  }
  void decouper(){
    this.etat = "decoupe";
  }
}
```

Exercice 2.1 - solution : les deux premières instructions entrainent une erreur de compilation (m2 n'est pas une méthode de la classe A) et la troisième n'entraine aucune erreur.

Exercice 2.2 - solution :
x=3
ajoute(2) = 5
x=3
ajoute(2) = 7
x=3
z=5
ajoute(2) = 9

Exercice 2.3 - solution :
a. le programme ne compile pas (t.k inconnu)
b. le programme compile et affiche 2,1,3
c. le programme ne compile pas (Toto n'est pas une sorte de Titi, affectation impossible)
d. le programme compile et affiche 2,1,3

Exercice 2.4 - solution : a. compile et affiche 1,3 ; b. compile et affiche 2,3 ; c. compile et affiche 2,0

Exercice 2.5 - solution : le code compile et affiche 2,1

Exercice 2.6 - solution : Ahah, Héhé, Ohoh, Ohoh.

retour au sommaire

Exercice 2.7 - solution : le code compile, il affiche "A et A", "A et A", "B et A", "B et A", "B et B" et "B et A".

Exercice 2.8 - solution : le code ne compile pas car il faut un constructeur sans paramètre dans A (pour que le constructeur sans paramètre de B puisse faire l'appel automatique au super-constructeur sans paramètre). On peut ajouter dans A le constructeur suivant : A(){ this.i=1;} . Dans ce cas, l'exécution du programme produit l'affichage : 2,3,5,2,1,1

Exercice 2.9 - solution : le code compile et affiche 2,0,0,B,A

Exercice 2.10 - solution :

```
class Reponse{
  String texte;
  boolean validite;
  Reponse(String t, boolean v){
    this.texte = t;
    this.validite = v;
  }
}

class QuestionQCM extends Question{
  ArrayList<Reponse> reponses;
  QuestionQCM(String e){
    super(e);
    this.reponses = new ArrayList<Reponse>();
  }
  void addReponse(String texte, boolean v){
    this.reponses.add(new Reponse(texte,v));
  }
  String getText(){
    String result = this.enonce + ":";
    for(Reponse s : this.reponses) result = result + "/" + s.texte;
    return result;
  }
}

class QuestionNotee{
  Question q;
  int points;
  QuestionNotee(Question q, int points){
    this.q = q;
    this.points = points;
  }
  int getPoints(){
    return this.points;
  }
```

retour au sommaire

```
  int getDifficulte(){
    return this.q.getDifficulte();
  }
  void setDifficulte(int d){
    this.q.setDifficulte(d);
  }
  String getEnonce(){
    return this.q.getEnonce();
  }
}

class SujetExamen{
  ArrayList<QuestionNotee> questions;
  SujetExamen(){
    this.questions = new ArrayList<QuestionNotee>();
  }
  void addQuestion(QuestionNotee q){
    this.questions.add(q);
  }
  void removeQuestion(QuestionNotee q){
    this.questions.remove(q);
  }
  int difficulteMoyenne(){
    int n = 0;
    for(QuestionNotee q:this.questions){
      n = n + q.getDifficulte();
    }
    return n/this.questions.size();
  }
}
```

Exercice 2.11 - solution :

```
class Elu extends Personne{
  ArrayList<Personne> assistants;
  Elu(String n, String p){
    super(n,p);
    this.assistants = new ArrayList<Personne>();
  }
  void embaucheAssistant(String n, String p){
    this.assistants.add(new Personne(n,p));
  }
  void licencieAssistant(Personne a){
    this.assistants.remove(a);
  }
  void depenseDotation(int montant){
    for(Personne p:this.assistants) p.addSous(montant/this.assistants.size());
```

retour au sommaire

```
  }
}

class EluMafieux extends Elu{
  EluMafieux(String n, String p){
    super(n,p);
  }
  void embaucheAssistant(String n, String p){
    if(n.equals(this.nom)) super.embaucheAssistant(n,p);
  }
}

class EluEconome extends Elu{
  EluEconome(String n, String p){
    super(n,p);
  }
  void depenseDotation(int montant){
    for(Personne p:this.assistants){
      if(montant>1480){
        p.addSous(1480);
        montant = montant - 1480;
      }
    }
  }
}

class EluEscroc extends Elu{
  int compteSuisse;
  EluEscroc(String n, String p){
    super(n,p);
    this.compteSuisse = 0;
  }
  void depenseDotation(int montant){
    for(Personne p:this.assistants){
      if(montant>1480){
        p.addSous(1480);
        montant = montant - 1480;
      }
    }
    if(montant>0) this.compteSuisse = this.compteSuisse + montant;
  }
}
```

Exercice 2.12 - solution :

```
class Acte{
  String soignant;
```

retour au sommaire

```
  Instant date;
  Acte(String s, Instant d){
    this.soignant = s;
    this.date = d;
  }
}

class Diagnostic extends Acte{
  Maladie maladie;
  int validite; // entre 0 et 100%
  Diagnostic(String s, Instant d, Maladie m, int validite){
    super(s, d);
    this.maladie = m;
    this.validite = validite;
  }
}

class Soin extends Acte{
  int amelioration; // taux d'amélioration entre 0 et 100%
  Soin(String s, Instant d, int a){
    super(s, d);
    this.amelioration = a;
  }
}

class Patient{
  String nom, prenom;
  Instant naissance, mort;
  int sexe; //0=homme, 1=femme
  ArrayList<Acte> parcours;
  Patient(String n, String p, int sexe, Instant nait){
    this.nom = n;
    this.prenom = p;
    this.naissance = nait;
    this.sexe = sexe;
    this.parcours = new ArrayList<Acte>();
  }
  long getAge(){
    return this.naissance.until(Instant.now(),ChronoUnit.YEARS);
  }
  void addActe(Acte a){
    this.parcours.add(a);
  }
  int nbMedocJour(){
    int n=0;
    for(Acte a:this.parcours){
      if(a instanceof Medoc) n = n+((Medoc) a).frequence;
```

retour au sommaire

```
    }
  }
}
class Medoc extends Soin{
  int posologie, frequence;
  Medoc(String s, Instant d, int a, int poso, int f){
    super(s, d, a);
    this.posologie = poso;
    this.frequence = f;
  }
}
```

Exercice 3.1 - solution : dans la classe Machin, ligne 4, la méthode doit être déclarée abstract. Dans la classe Bidule, la méthode bidule() n'est pas concrétisée (et la classe n'est pas abstraite).

Exercice 3.2 - solution : le code compile sans problème.

Exercice 3.3 - solution : 3e ligne, il faut initialiser i, qui est une constante (non modifiable). 7e ligne, la méthode doit ne rien retourner.

Exercice 3.4 - solution : le code compile sans problème.

Exercice 3.5 - solution : ligne 15, Toto est une classe abstraite donc non instanciable. Ligne 23, il n'y a pas de super constructeur sans paramètre dans Toto. Ligne 27, le i de Toto est privé, donc invisible ici. Une fois corrigé, l'affichage est 2, 1.

Exercice 3.6 - solution :

```
public interface List {
  public Comparable get(int i);
  public void swap(int i, int j);
}

public interface Comparable {
  public int compare(Comparable o);
}

public void quickSort(List tab, int a, int b) {
  int pivot;
  if (b > a) {
    pivot = a;
    for (int i = a+1; i <= b;i++) {
      if (tab.get(i).compare(tab.get(pivot)) < 0) {
        tab.swap(i,pivot+1);
        tab.swap(i,pivot);
        pivot = pivot+1;
      }
```

retour au sommaire

```
    }
    quickSort(tab,a,pivot-1);
    quickSort(tab,pivot+1,b);
  }
}
```

Exercice 3.7 - solution :

```
class PoupeeCreuse extends PoupeeRusse{
  private PoupeeRusse contenu;
  public PoupeeCreuse(int taille){
    super(taille);
    this.contenu = null;
  }
  public void ajoutePoupee(PoupeeRusse pr){
    if(this.contenu == null) this.contenu = pr;
  }
  public void retirePoupee(){
    if(this.contenu != null) this.contenu = null;
  }
  public static void main(String[] toto){
    PoupeeCreuse p1 = new PoupeeCreuse(10);
    PoupeeCreuse p2 = new PoupeeCreuse(8);
    PoupeeRusse p3 = new PoupeeRusse(6);
    p1.ajoutePoupee(p2);
    p2.ajoutePoupee(p3);
  }
}
```

Exercice 3.8 - solution :

a.

```
public class Voyage implements Localisable{
  private int x,y,dx,dy;
  public Voyage(int x, int y, int dx, int dy){
    this.x = x; this.y = y; this.dx = dx; this.dy = dy;
  }
  public java.awt.Point getPosition(){
    return new java.awt.Point(x,y);
  }
  public void avance(){
    this.x = (int) (this.x+dx*PAS/Math.sqrt(dx*dx+dy*dy));
    this.y = (int) (this.y+dy*PAS/Math.sqrt(dx*dx+dy*dy));
  }
}
```

b.

```
public class Personnage{
```

retour au sommaire

```
  private Localisable l;
  private String nom;
  public Personnage(String n, int x, int y){
    this.nom = n;
    this.l = new Exploration(x,y);
  }
  public java.awt.Point getPosition(){
    return this.l.getPosition();
  }
  public void setLocalisable(Localisable l){
    this.l = l;
  }
  public static void main(String[] t){
    Personnage p = new Personnage("Toto",0,0);
    p.setLocalisable(new Voyage(p.getPosition().x,p.getPosition().y,6,8));
  }
}
```

Exercice 3.9 - solution :

```
public class Arbre {
  private int hauteur, diametre;
  public Arbre(int h, int d, Parcelle p) {
    this.hauteur = h; this.diametre = d;
    p.add(this);
  }
  public class Resineux extends Arbre { }
  public class Feuillu extends Arbre { }
}

public class Parcelle {
  private ArrayList<Arbre> arbres;
  private int surface;
  public Parcelle(int s) {
    this.arbres = new ArrayList<Arbres>();
    this.surface = s;
  }
  public void addArbre(Arbre a) {
    this.arbres.add(a);
  }
  public void setSurface(int s) {
    this.surface = s;
  }
  public int getNbArbres() {
    return this.surfaces.size();
  }
}
```

retour au sommaire

Exercice 3.10 - solution :

```
public abstract class Zone implements Affichable{
   private String nom;
   protected int superficie, niveau; // superficie en km2, niveau en mm
   protected java.util.ArrayList<Zone> voisines;
  public Zone(String n, int s){
    this.nom = n;
    this.superficie = s;
    this.niveau = 0;
    this.voisines = new java.util.ArrayList<Zone>();
  }
  public void addVoisine(Zone z){
    this.voisines.add(z);
  }
  /** retourne en m3 le volume d'eau reçu par la zone en fonction
  des mm de pluie tombés */
   protected int volume(int pluie){
     return this.superficie*pluie*1000;
   }
   public abstract void recoit(int pluie);
   public abstract void distribue();
}

public abstract class ZoneConstruite extends Zone{
  public ZoneConstruite(String n, int s){
      super(n,s);
  }
  public void recoit(int pluie){
    this.niveau = this.niveau+pluie;
  }
  public java.awt.Color getColor(){
    return java.awt.Color.GRAY;
  }
}

public abstract class ZoneHumide extends Zone{
  private int capacite, contient; // en m3
  public ZoneHumide(String n, int s, int c){
      super(n,s);
      this.capacite = c;
      this.contient = 0;
  }
  public void recoit(int pluie){
    if(this.niveau>0) this.niveau = this.niveau+pluie;
    else{
```

retour au sommaire

```
      int surplus = this.volume(pluie)-(this.capacite-                this.contient);
      if(surplus>0){
        this.contient = this.capacite;
        this.niveau = this.niveau+(surplus/                (this.superficie*1000));
      }
      else this.contient = this.contient+this.volume(pluie);
    }
  }
  public java.awt.Color getColor(){
    return java.awt.Color.GREEN;
  }
}
```

Exercice 3.11 - solution :

```
public class Livre{
  private String titre;
  private ArrayList<String> themes;
  public Livre(String titre){
    this.titre = titre;
    this.themes = new ArrayList<String>();
  }
  public void addTheme(String t){
    this.themes.add(t);
  }
  public boolean aPourTheme(String t){
    for(String s:this.themes){
      if(s.equals(t)) return true;
    }
    return false;
  }
}

public class Bibliotheque extends ArrayList<Livre>{
  private String nom;
  public Bibliotheque(String n){
    super();
    this.nom = n;
  }
  public Iterator<Livre> iterator(String t){
    return this.new MonIterateur(t);
  }
  class MonIterateur implements Iterator<Livre>{
    int index;
    String theme;
    private MonIterateur(String theme){
      this.index=0;
```

retour au sommaire

```
    this.theme = theme;
  }
  public boolean hasNext(){
    int j = this.index;
    while(j<Bibliotheque.this.size()){
      if(!Bibliotheque.this.get(j).aPourTheme(this.theme)) return true;
      j++;
    }
    return false;
  }
  public Livre next(){
    if(this.index>=Bibliotheque.this.size()) throw new NoSuchElementException("On est au bout de
                                                                                la liste!");
    else{
      this.index++;
      if(!Bibliotheque.this.get(this.index-1).aPourTheme(this.theme)) return this.next();
      else return Bibliotheque.this.get(this.index-1);
    }
  }
  public void remove(){}
 }
}
```

Exercice 4.1 - solution : l'interface I est invisible en dehors de son paquetage et ne peut donc pas être implémentée par la classe C. L'attribut i est invisible en dehors de sa classe et ne peut donc être accédé dans la classe D.

Exercice 4.2 - solution : dans la classe B l'attribut a n'est pas accessible car privé dans A. Dans la classe D, 4e ligne, l'attribut a est privé dans A, donc non accessible. Dans la classe F, 4e ligne, l'attribut a est privé dans A, donc non accessible.

Exercice 4.3 - solution : les 13e et 14e lignes sont incorrectes (i n'est pas visible en dehors de C et j n'est pas visible en dehors du paquetage de C).

Exercice 4.4 - solution : 10e ligne la méthode doit être publique, 17e ligne l'attribut petales ne peut être accédé.

Exercice 4.5 - solution : erreurs à la compilation. 8e ligne, j n'est pas visible dans B. Lignes 11, 12 et 13, B n'est pas visible dans le paquetage p2 donc i non plus.

Exercice 4.6 - solution : erreurs à la compilation. La méthode m() de B doit avoir une visibilité supérieure ou égale à celle de D. Dans la classe D, il faut ajouter import p1.p3.A; en tête de fichier, ou nommé A avec son nom complet.

Exercice 4.7 - solution : les classes compilent. L'affichage est : Je suis un D de p1 /Je suis un C / Je suis un D de p2 / Je suis un B

retour au sommaire

Exercice 4.8 - solution : A et B doivent être mises dans le même paquetage, k en privé, getI en protégé et getL en public.

Exercice 4.9 - solution : le code compile et affiche A,A,B,A,B, 1,1,2,1,2,2.

Exercice 5.1 - solution :

```
public class Node{
  private int value;
  public Node(int i){
    this.value = i;
  }
}

public class Edge{
  private Node from, to;
  public Edge(Node from, Node to){
    this.from = from;
    this.to = to;
  }
  public boolean links(Node n){
    return this.from.equals(n) || this.to.equals(n);
  }
  public boolean fromTo(Node from, Node to){
    return this.from.equals(from) && this.to.equals(to);
  }
}

public interface Graph{
  public boolean addNode(int i);
  public boolean addEdge(Node n1, Node n2);
  public boolean removeNode(Node n);
  public boolean removeEdge(Node n1, Node n2);
  // return null if there is no edge between from and to
  public Edge getEdge(Node from, Node to);
}

import java.util.ArrayList;
public abstract class UndirectedGraph implements Graph{
  protected ArrayList<Node> nodes;
  protected ArrayList<Edge> edges;
  public UndirectedGraph(){
    this.nodes = new ArrayList<Node>();
    this.edges = new ArrayList<Edge>();
  }
  public boolean addNode(int i){
    return this.nodes.add(new Node(i));
```

retour au sommaire

```
  }
  public boolean removeNode(Node n){
    boolean b = this.nodes.remove(n);
    if(b){
      ArrayList<Edge> newEdges = new ArrayList<Edge>();
      for(Edge e:this.edges){
        if(!e.links(n)) newEdges.add(e);
      }
      this.edges = newEdges;
    }
    return b;
  }
  public Edge getEdge(Node n1, Node n2){
    for(Edge e:this.edges){
      if(e.fromTo(n1,n2) || e.fromTo(n2,n1)) return e;
    }
    return null;
  }
}

public class SimpleUndirectedGraph extends UndirectedGraph{
  public SimpleUndirectedGraph(){
    super();
  }
  public boolean addEdge(Node n1, Node n2){
    if(this.getEdge(n1,n2)==null)
      return this.edges.add(new Edge(n1,n2));
    else return false;
  }
  public boolean removeEdge(Node n1, Node n2){
    Edge ed = this.getEdge(n1,n2);
    if(ed!=null) return this.edges.remove(ed);
    else return false;
  }
}
```

Exercice 5.2 - solution :

```
public abstract class Enseignant {
  protected String nom,prenom;
  protected int heuresCours;
  protected static int charges = 1;
  public Enseignant(String n, String p, int h) {
    this.nom = n; this.prenom = p; this.heuresCours = h;
  }
  public abstract float cout();
}
```

retour au sommaire

```
public class EnseignantChercheur extends Enseignant {
  public EnseignantChercheur(String n, String p, int h) {
    super(n,p,h);
  }
  public float cout() {
    return (2000*12+(this.h-192)*40)*charges;
  }
}

public class Vacataire extends Enseignant {
  private String organisme;
  public Vacataire(String n, String p, int h, String o) {
    super(n,p,h);
    this.organisme = o;
  }
  public float cout() {
    return (40*this.h)*charges;
  }
}

public class Doctorant extends Enseignant {
  public Doctorant(String n, String p, int h) {
    super(n,p,h);
  }
  public float cout() {
    if (this.h > 96) return (96*35)*charges;
    else return (35*this.h)*charges;
  }
}
```

Exercice 5.3 - solution :

```
public abstract class Produit{
  protected String nom;
  protected float poids;
  protected Shape3D forme;
  protected double volume;
  public Produit(String n, float poids, Shape3D f, double v){
    this.nom = n;
    this.poids = poids;
    this.forme = f;
    this.volume = v;
  }
}
```

retour au sommaire

```
public interface Deformable{
  public void setForme(Contenant s);
}

public class Liquide extends Produit implements Deformable{
  private int viscosite;
  public Liquide(String n, float p, Shape3D f, double v, int visco){
    super(n,p,f,v);
    this.viscosite = visco;
  }
  public void setForme(Contenant s){
    this.forme = s.getSubShape(this.volume);
  }
}

public class Gaz extends Produit implements Deformable{
  public Gaz(String n, float p, Shape3D f, double v){
    super(n,p,f,v);
  }
  public void setForme(Contenant s){
    this.forme = s;
  }
}

public class Solide extends Produit{
  public Solide(String n, float p, Shape3D f, double v){
    super(n,p,f,v);
  }
}

import javafx.scene.shape.Shape3D;
import java.util.ArrayList;
public abstract class Contenant extends Shape3D{
  private int x, y, z, opacite; // opacite de 0 à 100%
  private ArrayList<Produit> contenu;
  public Contenant(int x, int y, int z, int opacite){
    super();
    this.x = x;
    this.y = y;
    this.z = z;
    this.opacite = opacite;
    this.contenu = new ArrayList<Produit>();
  }
  // renvoie la forme occupée dans le contenant par un liquide de volume donné
  public abstract Shape3D getSubShape(double volume);
  public void addProduit(Produit p){
    this.contenu.add(p);
```

retour au sommaire

```
    if(p instanceof Deformable){
      ((Deformable) p).setForme(this);
    }
  }
}
```

Exercice 5.4 - solution :

```
public class Humain extends Personnage{
  public Humain(String nom, int x, int y){
    super(nom, x, y, 5);
  }
  public String parler(){return "bonjour";}
}

public class Elfe extends Personnage{
  public Elfe(String nom, int x, int y){
    super(nom, x, y, 7);
  }
  public String parler(){return "Eldarie";}
}

public class Nain extends Personnage{
  public Nain(String nom, int x, int y){
    super(nom, x, y, 2);
  }
  public String parler(){return "groumpf";}
}

public interface Guerrier{
  public void attaque(Personnage p);
  public int getForce();
}

public interface Magicien{
  public void lancerSort(Personnage p);
  public int getMagie();
}

public interface Voleur{
  public void voler(Personnage p);
  public int getDexterite();
}

public class NainGuerrier extends Nain implements Guerrier{
  private int force;
  public NainGuerrier(String nom, int x, int y, int force){
```

retour au sommaire

```
    super(nom, x, y);
    this.force = force;
  }
  public void attaque(Personnage p){
    p.setPointsVie((p.getPointsVie()*(100-this.force))/100);
  }
  public int getForce(){
    return this.force;
  }
}

public class NainGuerrierVoleur extends NainGuerrier implements Voleur{
  private int dexterite;
  public NainGuerrierVoleur(String nom, int x, int y, int force, int dexterite){
    super(nom, x, y, force);
    this.dexterite = dexterite;
  }
  public void voler(Personnage p){
    if(p.getSous() >= this.dexterite){
      p.setSous(p.getSous()-this.dexterite);
      this.setSous(this.getSous()+this.dexterite);
    }
    else{
      this.setSous(this.getSous()+p.getSous());
      p.setSous(0);
    }
  }
  public int getDexterite(){return this.dexterite;}
}
```

Exercice 5.5 - solution :

```
public abstract class Personnage{
  private static int nbSauron = 0;
  abstract class Anneau{
    protected String nom;
    public abstract void metAuDoigt();
    public abstract void enleveDuDoigt();
  }
  class AnneauDeSauron extends Anneau{
    private AnneauDeSauron(){
      this.nom = "Anneau de Sauron";
    }
    public void metAuDoigt(){
      Personnage.this.setVisible(false);
    }
    public void enleveDuDoigt(){
```

retour au sommaire

```
      Personnage.this.setVisible(true);
    }
  }
  public void creeAnneauDeSauron(){
    if(nbSauron == 0){
      this.new AnneauDeSauron();
      nbSauron++;
    }
  }
  private String nom;
  private int pointsVie, x, y, v, sous;
  private boolean visible = true;
  public Personnage(String n, int x, int y, int v){
    this.nom = n;
    this.x = x; this.y = y;
    this.pointsVie = 100;
    this.v = v;
    this.sous = 0;
  }
  public int getSous(){return this.sous;}
  public void setSous(int s){this.sous = s;}
  public int getPointsVie(){return this.pointsVie;}
  public void setPointsVie(int pv){this.pointsVie = pv;}
  public void setVisible(boolean b){this.visible = b;}
  public void seDeplacer(int dx, int dy, int t){
    this.x = (int) (this.x + dx*this.v*t/Math.sqrt(dx*dx+dy*dy));
    this.y = (int) (this.y + dy*this.v*t/Math.sqrt(dx*dx+dy*dy));
  }
  public abstract String parler();
}

public class Hobbit extends Personnage{
  public Hobbit(String n, int x, int y){
    super(n,x,y,5);
  }
  public String parler(){
    return "Belle journée ma foi";
  }
  public static void main(String[] tutu){
    Hobbit frodon = new Hobbit("Frodon",200,100);
    frodon.creeAnneauDeSauron();
  }
}
```

Exercice 5.6 - solution :

```
public class Objet{
```

retour au sommaire

```
  private String nom;
  private int prix;
  public Objet(String n, int p){
    this.nom = n; this.prix = p;
  }
}

import java.util.LinkedList;
public abstract class Humanoide extends Personnage{
  private LinkedList<Objet> objets;
  private LinkedList<Document> lus;
  public Humanoide(String n, int x, int y, int v){
    super(n,x,y,v);
    this.objets = new LinkedList<Objet>();
    this.lus = new LinkedList<Document>();
  }
  public void acquiertObjet(Objet o){
    this.objets.add(o);
  }
  public void perdObjet(Objet o){
    this.objets.remove(o);
  }

  public void donnneObjet(Objet o, Humanoide h){
    if(this.objets.remove(o)) h.acquiertObjet(o);
  }
  public void attaque(Personnage p, Arme a){
    if(this.objets.contains(a)) p.setPointsVie(Math.max(0,p.getPointsVie()-a.getPuissance()));
  }
  public void lit(Document d){
    if(this.objets.contains(d) && !this.lus.contains(d)){
      this.connaissances = this.connaissances+d.getConnaissances();
      this.lus.add(d);
    }
  }
}

public class Arme extends Objet{
  private int puissance;
  public Arme(String n, int p, int pui){
    super(n,p);
    this.puissance = pui;
  }
  public int getPuissance(){
    return this.puissance;
  }
}
```

retour au sommaire

```
public class Document extends Objet{
  private int connaissances;
  public Document(String n, int p, int con){
    super(n,p);
    this.connaissances = con;
  }
  public int getConnaissances(){
    return this.connaissances;
  }
}

public class Troll extends Personnage implements Monstre{
  private int force;
  public Troll(String n, int x, int y, int v, int force){
    super(n,x,y,v);
    this.force = force;
  }
  public int getPuanteur(){
    return 200;
  }
  public void attaque(Personnage p){
    if(!(p instanceof Troll)) p.setPointsVie(Math.max(0,p.getPointsVie() - this.force - this.getPuanteur()/
10));
  }
  public String parler(){
    return "Hiark";
  }
}
```

Exercice 5.7 - solution :

```
public class Jedi extends Personnage{
  private final Sabre s;
  class Sabre{
    private Color c;
    private boolean on;
    public Sabre(Color c){this.c = c; this.on = false;}
    public void allumer(){this.on = true;}
    public void eteindre(){this.on = false;}
  }
  public Jedi(String n, int x, int y, int force, Color c){
    super(n,x,y,5,force);
    this.s = this.new Sabre(c);
  }
  public void allumerSabre(){
    this.s.allumer();
```

retour au sommaire

```
  }
  public void eteindreSabre(){
    this.s.eteindre();
  }
  public String parler(){
    return "Que la force soit avec vous";
  }
  public static void main(String[] toto){
    Jedi yoda = new Jedi("Maitre Yoda", 45, 56, 120, Color.GREEN);
    System.out.println(yoda.parler());
    yoda.allumerSabre();
  }
}

public class SoldatClone extends Personnage implements Clone{
  public SoldatClone(){
    super("", DJANGO_FETT.getX(), DJANGO_FETT.getY(), DJANGO_FETT.getV(),
                                                              DJANGO_FETT.getForce());
  }
  public void tireSur(Personnage p){
    p.estBlesse();
  }
  public String parler(){
    return "";
  }
}
```

Exercice 5.8 - solution :

a.

```
public interface Cible{
  public void estTouchePar(Arme a);
}

public abstract class Arme{
  protected int puissance;
  private Personnage detenteur;
  public Arme(int puissance){
    this.puissance = puissance;
  }
  public void setDetenteur(Personnage p){
    this.detenteur = p;
  }
  public int getPuissance(){return this.puissance;}
  public abstract void tireSur(Cible c);
}
```

on ajoute à l'entête de la classe Personnage : *implements Cible*

retour au sommaire

b.

```
public abstract class VaisseauSpatial extends Arme implements Cible{
  private int blindage /* de 1 à 5 */, etat /* de 0 à 100 */;
  protected VaisseauSpatial(int puissance, int blindage){
    super(puissance);
    this.etat = 100;
    this.blindage = blindage;
  }
  public void estTouchePar(Arme a){
    this.etat = this.etat - a.getPuissance()/this.blindage;
  }
}
```

c.

```
public class VaisseauPilote extends VaisseauSpatial{
  private Personnage pilote;
  public VaisseauPilote(int puissance, int blindage, Personnage pilote){
    super(puissance, blindage);
    this.pilote = pilote;
  }
  public void tireSur(Cible c){
    if(Math.random()*100<this.pilote.getPrecision())  c.estTouchePar(this);
  }
}
public class Drone extends VaisseauSpatial{
  private int precision;
  public Drone(int puissance, int blindage, int precision){
    super(puissance, blindage);
    this.precision = precision;
  }
  public void tireSur(Cible c){
    if(Math.random()*100<this.precision) c.estTouchePar(this);
  }
}
```

d.

```
public class EtoileMort extends VaisseauPilote{
  private static EtoileMort  em = null;
  private EtoileMort(Personnage pilote){
    super(1000,5,pilote);
  }
  public static EtoileMort getEtoileMort(Personnage pilote){
    if(em == null) em = new EtoileMort(pilote);
    return em;
  }
}
```

retour au sommaire

Exercice 5.9 - solution :

```
public abstract class Personnage implements Localisable{
  protected String nom;
  protected int pointsVie,force;
  protected int x,y; // pour question c
  public Personnage(String n, int f){
    this.nom = n;
    this.force = f;
    this.pointsVie = 100;
    this.x = 0; this.y = 0;
  }
  public void mange(){if(this.pointsVie<100) this.pointsVie++;}
  public abstract void rencontre(Personnage p);
  public int getX(){return this.x;}
  public int getY(){return this.y;}
  public void bouge(int dx, int dy){this.x = this.x+dx; this.y = this.y+dy;}
  public static void main(String[] toto){
    CoteObscur dv = new CoteObscur("Dark Vador",54);
    CoteLumineux ls = new CoteLumineux("Luc Skywalker",47);
    dv.rencontre(ls);
  }
}

public class CoteObscur extends Personnage{
  public CoteObscur(String n, int p){super(n,p);}
  public void rencontre(Personnage p){
    if(this.pointsVie>10 && this.force>p.force) p.pointsVie=0;
  }
}

public class CoteLumineux extends Personnage{
  public CoteLumineux(String n, int p){  super(n,p);}
  public void rencontre(Personnage p){
    if(p instanceof CoteObscur && this.pointsVie>10 && this.force>p.force) p.pointsVie=0;
  }
}

public interface Localisable{
  public int getX();
  public int getY();
  public void bouge(int dx, int dy);
}

public class Jedi extends CoteLumineux{
  private java.awt.Color sabre;
  public Jedi(String n, int p, Color c){
```

retour au sommaire

```
    super(n,p);
    this.sabre = c;
  }
  public void deplacer(Localisable l, int dx, int dy){
    if(this.pointsVie>=10) l.bouge(dx,dy);
  }
}
```

Question e : on créer une interface *Rencontre* qui ne contient qu'une méthode *public void rencontre(Personnage p)* et deux classes *RencontreLumineux* et *RencontreObscur* qui implémentent *Rencontre* des deux façons prévues. Dans la classe *Personnage*, on ajoute un attribut de type *Rencontre* et on modifie la méthode rencontre pour qu'elle appelle celle de l'attribut.

Exercice 5.10 - solution :

a.

```
public interface TabAsso{
  public boolean put(Ordonnable o, ObjetNomme on);
  public ObjetNomme get(Ordonnable o);
  public boolean remove(ObjetNomme on);
  public boolean contains(ObjetNomme on);
  public int size();
}
```

b.

```
import java.util.ArrayList;
public class ListTabAsso implements TabAsso{
  class Paire{
    private ObjetNomme on;
    private Ordonnable o;
    Paire(ObjetNomme on, Ordonnable o){
      this.on = on; this.o = o;
    }
  }
  private ArrayList<Paire> tab;
  public ListTabAsso(){
    this.tab = new ArrayList<Paire>();
  }
  private boolean contains(Ordonnable o){
    for(Paire p:this.tab){
      if(p.o==o) return true;
    }
    return false;
  }
  public boolean put(Ordonnable o, ObjetNomme on){
    if(this.contains(o)) return false;
    else this.tab.add(new Paire(on,o));
```

retour au sommaire

```
    return true;
  }
  public ObjetNomme get(Ordonnable o){
    for(Paire p:this.tab){
      if(p.o==o) return p.on;
    }
    return null;
  }
  public boolean remove(ObjetNomme on){
    Paire x = null;
    for(Paire p:this.tab){
      if(p.on==on) x = p;
    }
    if(x!=null){
      this.tab.remove(x);
      return true;
    }
    else return false;
  }
  public boolean contains(ObjetNomme on){
    for(Paire p:this.tab){
      if(p.on==on) return true;
    }
    return false;
  }
  public int size(){
    return this.tab.size();
  }
  public static void main(String[] t){
    ListTabAsso tab = new ListTabAsso();
  }
}
```

c.

```
public static void main(String[] t){
  ListTabAsso tab = new ListTabAsso();
  ObjetNomme on = new ObjetNomme();
  tab.put(new Ordonnable(),on);
  System.out.println("taille "+tab.size());
  tab.remove(on);
}
```

d.

```
public interface TabAssoLimite extends TabAsso{
  public int getTailleMax();
}
```

retour au sommaire

e.

```
import java.util.ArrayList;
public class ListTabAssoLimite extends ListTabAsso implements TabAssoLimite{
  private int max;
  public ListTabAssoLimite(int tailleMax){
    super();
    this.max = tailleMax;
  }
  public int getTailleMax(){
    return this.max;
  }
  public boolean put(Ordonnable o, ObjetNomme on){
    if(this.size()==this.max) return false;
    else return super.put(o,on);
  }
}
```

Exercice 5.11 - solution :

```
package database;
public interface Value extends Comparable<Value>{
  public int compareTo(Value v);
  public String toString();
  public Value copy();
}
```

```
package database;
public class Key implements Value{
  private int i;
  public Key(int i){
    this.i=i;
  }
  public int compareTo(Value v){
    if(v instanceof Key) return this.i-((Key) v).i;
    else throw new ClassCastException();
  }
  public String toString(){
    return this.i+"";
  }
  public Value copy(){
      return new Key(this.i);
  }
  public Key next(){
    return new Key(this.i+1);
  }
  public Key previous(){
    return new Key(this.i-1);
```

retour au sommaire

```
  }
}

package database;
import java.util.List;
import java.util.ArrayList;
public class Table{
  protected String[] ids;
  protected ArrayList<Value[]> rows;
  protected Table(){
    this.ids = new String[0];
    this.rows = new ArrayList<Value[]>();
  }
  public Table(List<String> cols){
    this.ids = new String[cols.size()];
    int i = 0;
    for(String s:cols){
      this.ids[i] = s;
      i++;
    }
    this.rows = new ArrayList<Value[]>();
  }
  protected void setIDs(List<String> cols){
    this.ids = new String[cols.size()];
    int i = 0;
    for(String s:cols){
      this.ids[i] = s;
      i++;
    }
    this.rows = new ArrayList<Value[]>();
  }
  public void addRow(List<Value> l){
    Value[] t = new Value[this.ids.length];
    int i = 0;
    for(Value v:l){
      t[i] = v;
      i++;
    }
    this.rows.add(t);
  }
  public void removeLastRow(){
    this.rows.remove(this.rows.size()-1);
  }
  protected int getIndex(String id){
    for(int i=0;i<this.ids.length;i++){
      if(this.ids[i].equals(id)) return i;
    }
```

retour au sommaire

```
    return -1;
  }
}

package database;
import java.util.List;
public class IndexableTable extends Table{
  private Key last = null;
  public IndexableTable(List<String> cols){
    super();
    cols.add(0,"CLE");
    this.setIDs(cols);
  }
  public void addRow(List<Value> l){
    Value[] t = new Value[this.ids.length];
    int i = 1;
    for(Value v:l){
      t[i] = v;
      i++;
    }
    if(this.last==null) this.last = new Key(0);
    else this.last = this.last.next();
    t[0] = this.last;
    this.rows.add(t);
  }
  public void removeLastRow(){
    this.rows.remove(this.rows.size()-1);
    this.last = this.last.previous();
  }
  public void put(Key k, String id, Value v){
    for(Value[] row:this.rows){
      if(row[0].compareTo(k)==0){
        int index = this.getIndex(id);
        if(index!=-1) row[index] = v;
      }
    }
  }
  public Value get(Key k, String id){
    for(Value[] row:this.rows){
      if(row[0].compareTo(k)==0){
        int index = this.getIndex(id);
        if(index!=-1) return row[index];
      }
    }
    return null;
  }
}
```

retour au sommaire

Exercice 5.12 - solution :

```
public abstract class Shape implements Drawable{
  protected final int id;
  protected final Color col;
  protected int x,y;
  public Shape(int id, Color c){
    this.id = id;
    this.col = c;
    this.x = 0; this.y = 0;
  }
  public int getId(){ return this.id; }
  public boolean moveTo(Point p){
    if(p.x>=minX && p.x<=maxX && p.y>=minY && p.y<=maxY){
      this.x = p.x;
      this.y = p.y;
      return true;
    }
    else return false;
  }
}

public abstract class Shape2D extends Shape{
  public Shape2D(int id, Color c){
    super(id,c);
  }
  public abstract int getWidth();
  public abstract int getHeight();
}

public class Rectangle extends Shape2D{
  protected int width, height;
  public Rectangle(int id, Color c, int w, int h){
    super(id,c);
    this.width = w;
    this.height = h;
  }
  public void draw(Graphics g){
    g.setColor(this.col);
    g.fillRect(this.x,this.y,this.width,this.height);
  }
  public int getWidth(){ return this.width; }
  public int getHeight(){  return this.height;}
}

public class Circle extends Shape2D{
```

retour au sommaire

```
  protected int diameter;
  public Circle(int id, Color c, int d){
    super(id,c);
    this.diameter = d;
  }
  public void draw(Graphics g){
    g.setColor(this.col);
    g.fillOval(this.x,this.y,this.x+this.diameter,this.y+this.diameter);
  }
  public int getWidth(){ return this.diameter; }
  public int getHeight(){  return this.diameter; }
}

public abstract class Shape3D extends Shape2D{
  public Shape3D(int id, Color c){
    super(id,c);
  }
  public abstract int getDepth();
}

public class Cube extends Shape3D{
  protected int width;
  public Cube(int id, Color c, int w){
    super(id,c);
    this.width = w;
  }
  public void draw(Graphics g){
    g.setColor(this.col);
    g.fill3DRect(this.x,this.y,this.width,this.width,true);
  }
  public int getWidth(){return this.width;}
  public int getHeight(){  return this.width;}
  public int getDepth(){return this.width;}
}
```

retour au sommaire