Énoncés 9

Exercices sur les interfaces et la conversion

1 Exercice sur les animaux

Certain animaux peuvent crier, d'autres sont muets. On représentera le fait de crier au moyen d'une méthode affichant à l'écran le cri de l'animal.

— écrire une interface contenant la méthode permettant de crier.
— écrire les classes des chats, des chiens et des lapins (qui sont muets)
— écrire un programme avec un tableau pour les animaux qui savent crier, le remplir avec des chiens et des chats, puis faire crier tous ces animaux. Décrire ce qui s'affiche à l'écran à l'exécution de ce programme.

2 Exercice sur les formes vagabondes

Nous allons reprendre et enrichir l'exmple des formes géométriques vues en cours. Nous allons doter les quatre classes de l'exemple d'une méthode `translation` permettant de déplacer une forme sur le plan et ajouter une nouvelle interface `AvecTranslation`. Cela donne le code suivant.

```
public interface AvecTranslation{
    void translation(double deplHor, double deplVer);
}
public class Point implements AvecTranslation{
    protected double x, y;
    public Point (double xi, double yi){
        x = xi;
        y = yi;
    }
    public double getX(){
        return x;
    }
    public double getY(){
        return y;
    }
    public double distance(Point p2){
      return Math.sqrt((this.x-p2.x)*(this.x-p2.x) +
```

```
            (this.y-p2.y) *(this.y-p2.y));
    }
    public void translation(double deplHor, double deplVer){
        x = x + deplHor;
        y = y + deplHor;
    }
}
public interface AvecSurface{
    double surface(); // Pas de corps de méthode
}
public class Cercle implements AvecSurface, AvecTranslation{
    protected Point centre;
    protected double rayon;
    public Cercle(Point ctr, double r){
        centre = ctr;
        rayon = r;
    }
    public double surface(){
        return Math.PI *rayon *rayon;
    }
    public void translation(double deplHor, double deplVer){
        centre.translation(deplHor,deplVer);
    }
}
public class Rectangle implements AvecSurface, AvecTranslation{
    protected Point basGauche;
    protected double dimHor, dimVer;
    public Rectangle(Point bg, double dh, double dv){
        basGauche = bg;
        dimHor = dh;
        dimVer = dv;
    }
    public double surface(){
        return dimHor *dimVer;
    }
    public void translation(double deplHor, double deplVer){
        basGauche.translation(deplHor,deplVer);
    }
}
public class Triangle implements AvecSurface, AvecTranslation{
    protected Point p1, p2, p3;
    public Triangle(Point p1i, Point p2i, Point p3i){
        p1 = p1i;
        p2 = p2i;
        p3 = p3i;
    }
    public double surface(){
        double a = p1.distance(p2);
        double b = p1.distance(p3);
        double c = p2.distance(p3);
        double demiper = (a+b+c)/2;
        return Math.sqrt(demiper*(demiper-a)*(demiper-b)*(demiper-c));
    }
```

```
    public void translation(double deplHor, double deplVer){
        p1.translation(deplHor,deplVer);
        p2.translation(deplHor,deplVer);
        p3.translation(deplHor,deplVer);
    }
}
```

On veut à présent pouvoir mettre en mouvement des pièces, c'est à dire effectuer une translation donnée à chaque top d'horloge, ce top étant matérialisé par l'invocation d'une méthode appelée `top`. A l'appel de cette méthode, la translation est effectuée.

On va écrire une classe contenant dans une variable un objet à déplacer au top, cet objet étant n'importe quel objet proposant l'interface AvecTranslation. Les données du déplacement seront stockées dans autant de variables que nécessaire.

Question 1

Écrire cette classe.

Question 2

Ecrire une méthode `main` avec un tableau contenant quatre objets de la classe écrite à la question précédente, chacun d'entre eux contenant une instance de figure ou de point différente.

Question 3

Ajouter à l'interface AvecTranslation une méthode renvoyant sous forme de chaîne la position d'un objet. Utiliser cette méthode pour afficher la situation à trois top horloge successifs.

3 Exercice sur les conversions de type

Pour cet exercice, nous reprenons les quatre classes de mots utilisées dans le cours sur les conversions de types. Le code est rappelé ci-dessous.

Question 1

Écrivez dans une nouvelle classe une méthode qui prend un paramètre de type `Mot` et teste si ce mot est une instance de la sous-classe `MotGenreNombre`. Pour ce faire la méthode utilisera une conversion explicite qui peut lever une exception et rattrapera éventuellement cette exception au moyen d'un `catch`.

L'entête de ctte méthode est le suivant :

```
public static boolean estMotGenreNombre(Mot unmot){
```

Question 2

En utilisant plusieurs conversions explicites, écrivez une méthode qui reçoit en paramètre un objet de type `Mot` et renvoie une chaîne de caractère contenant le nom de la classe dont l'objet est une instance (la classe qui a été utilisée avec `new` pour créer l'objet).

Le code des quatre classes est le suivant :

```
public class Mot{
    protected String formeMot;
    protected String categorie;
    public Mot(String fm, String cat){
        formeMot = fm;
        categorie = cat;
    }
    public String getForme(){
        return formeMot;
    }
    public String getCategorie(){
        return categorie;
    }
    public String toStringBis(){
        return "Mot␣" + formeMot + "␣categorie:␣" + categorie;
    }
}
public class MotNombre extends Mot{
    public static final int SINGULIER = 4;
    public static final int PLURIEL = 5;
    protected int nombre;
    public MotNombre(String forme, String cat, int nomb){
        super(forme,cat);
        if (nomb != SINGULIER && nomb != PLURIEL)
            throw new IllegalArgumentException("nombre␣ni␣SINGULIER␣ni␣PLURIEL");
        nombre = nomb;
    }
    public boolean estSingulier(){
        return nombre == SINGULIER;
    }
    public boolean estPluriel(){
        return nombre == PLURIEL;
    }
    public String toStringBis(){
        String res = "Mot␣variable␣en␣nombre␣" + formeMot + "␣categorie:␣" +
            categorie + "␣nombre:␣";
        if (nombre == SINGULIER){
            res = res + "␣singulier";
        }else{
            res = res + "␣pluriel";
        }
        return res;
    }
}
public class MotGenreNombre extends MotNombre{
    public static final int MASCULIN = 6;
    public static final int FEMININ = 7;
    protected int genre;
    public MotGenreNombre(String form, String cat, int gen, int nombre){
        super(form, cat, nombre);
        if (gen != MASCULIN && gen != FEMININ)
            throw new IllegalArgumentException();
```

```
        genre = gen;
    }
    public boolean estMasculin(){
        return genre==MASCULIN;
    }
    public boolean estFeminin(){
        return genre==FEMININ;
    }
    public String toStringBis(){
        String res = "Mot␣" + formeMot + "␣categorie:␣" +
            categorie + "␣nombre:";
        if (nombre == SINGULIER){
            res = res + "␣singulier";
        }else{
            res = res + "␣pluriel";
        }
        if (genre == MASCULIN)
            res =res + "␣genre:␣masculin";
        else
            res =res + "␣genre:␣feminin";
        return res;
    }
}
public class MotPersonneNombre extends MotNombre{
    public static final int PERS1 = 1;
    public static final int PERS2 = 2;
    public static final int PERS3 = 3;
    protected int personne;
    public MotPersonneNombre(String forme, String cat, int pers, int nombre){
        super(forme, cat, nombre);
        if (pers<PERS1 || pers>PERS3)
            throw new IllegalArgumentException();
        personne = pers;
    }
    public int getPersonne(){
        return personne;
    }
    public String toStringBis(){
        String res = "Mot␣" + formeMot + "␣categorie:␣" +
            categorie + "␣nombre:␣";
        if (nombre == SINGULIER){
            res = res + "␣singulier";
        }else{
            res = res + "␣pluriel";
        }
        res = res + "␣personne:␣" + personne;
        return res;
    }
}
```