Travaux pratiques d'informatique N° 10

Le but de cette séance est de vous permettre d'approfondir vos connaissances concernant les principaux aspects de l'**héritage** (surtout la dérivation d'une classe fille à partir d'une classe mère existante, la surcharge et la redéfinition des méthodes héritées, les hiérarchies de classes).

1. Créer un nouveau projet Eclipse, appelé **PrTP10Exo1**, contenant (dans le package **cms_tp10**) les trois fichiers sources Java suivants :

- le fichier **PBD.java** qui se trouve sur le bureau virtuel du CMS et qui définit une classe de base publique **PBD** ;
- le fichier **PTD.java** qui sera écrit par vous-même et qui doit contenir une classe dérivée publique **PTD**, fille de la classe de base **PBD** ;
- le fichier **CP_TP10Exo1.java** qui se trouve sur le bureau virtuel du CMS et qui définit la classe principale **CP_TP10Exo1**, contenant la méthode **main( )** dans laquelle on teste la conception de la hiérarchie de classes précisée.

Indications :

- *la classe de base **PBD** sert à instancier des objets correspondant à des points dans le plan (caractérisés par deux coordonnées cartésiennes) ;*
- *la classe dérivée **PTD** doit permettre l'instanciation d'objets correspondant à des points dans l'espace (caractérisés par trois coordonnées cartésiennes) ;*
- *afin de concevoir la classe **PTD**, utilisez, éventuellement, le canevas proposé sur le bureau virtuel du CMS ;*
- *commentez avec générosité le code source écrit (en précisant, par exemple, si une méthode définie dans la classe fille est une version surchargée ou redéfinie de la méthode héritée de la classe mère) ;*
- *anticipez les résultats qui seront affichés à l'exécution du projet ;*
- *comparez les résultats anticipés avec ceux obtenus effectivement suite à l'exécution du projet.*

2. Soit un fichier source Java appelé **CP_TP10Exo2.java**, contenant les trois classes définies ci-après. Le but de l'exercice est de comprendre le programme présenté ci-dessous et de préciser les messages affichés lors de son exécution.

```
class Mere {
    private int champMere;
    Mere( ) {
        champMere = 0;
        System.out.println("Je suis le constructeur Mere sans arguments !");          }
    Mere(int arg) {
        champMere = arg;
        System.out.println("Je suis le constructeur Mere avec un argument !");        }
    int getChampMere( ) { return champMere;                                           }
    void setChampMere(int arg) { champMere = arg;                                     }
    boolean verifierConcordance(int arg) {
        System.out.print("Je suis la methode Mere d'origine ! ");
        if(champMere == arg) return true;          else return false;                 }
    void affichage( ) { System.out.print(champMere);                                  }
}       //fin de la classe Mere

class Fille extends Mere{
    private char champFille;
    Fille( ) {
        champFille = 'a';
        System.out.println("Je suis le constructeur Fille sans arguments !");         }
    Fille(char arg) {
        champFille = arg;
        System.out.println("Je suis le constructeur Fille avec un  argument !");      }
    Fille(int arg1, char arg2) {
        super(arg1);          champFille = arg2;
        System.out.println("Je suis le constructeur Fille avec deux arguments !");    }
    char getChampFille( ) { return champFille;                                        }
    void setChampFille(char arg){ champFille = arg;                                   }
    @Override
    boolean verifierConcordance(int arg) {
        System.out.print("Je suis la methode Fille redefinie, avec un argument int ! ");
        return super.verifierConcordance(arg);
    }
```

```
        boolean verifierConcordance(char arg) {
                System.out.print("Je suis la methode Fille surchargee, avec un argument char !");
                if(champFille == arg) return true;   else return false;                    }
        boolean verifierConcordance(int arg1, char arg2) {
                System.out.print("Je suis la methode Fille surchargee, avec deux arguments ! ");
                if(getChampMere( ) == arg1 && champFille == arg2) return true;
                else return false;                                                          }
        @Override
        void affichage( ) {
                super.affichage( );     System.out.print(", " + champFille);                }
}       //fin de la classe Fille

public class CP_TP10Exo2 {
        public static void main(String args[ ]) {
                Mere refObjMere1 = new Mere( );          Mere refObjMere2 = new Mere(10);
                System.out.println("-------------------------------------------------------------------------");

                Fille refObjFille1 = new Fille( );          Fille refObjFille2 = new Fille('H');
                Fille refObjFille3 = new Fille(20, 'Z');
                System.out.println("-------------------------------------------------------------------------");

                System.out.println(refObjMere1.verifierConcordance(0));
                System.out.println(refObjMere2.verifierConcordance(20));

                System.out.println(refObjFille2.verifierConcordance(0));
                System.out.println(refObjFille3.verifierConcordance('Z'));
                System.out.println(refObjFille2.verifierConcordance(0,'H'));
                System.out.println("-------------------------------------------------------------------------");

                refObjMere1 = refObjFille3;
                System.out.println(refObjMere1.getChampMere( ));
                //System.out.println(refObjMere1.getChampFille( ));              FAUX
                System.out.println(((Fille)refObjMere1).getChampFille( ));
                System.out.println(refObjMere1.verifierConcordance('Z'));
                System.out.println(((Fille)refObjMere1).verifierConcordance('Z'));
        }       //fin de la méthode main( )
}       //fin de la classe CP_TP10Exo2
```