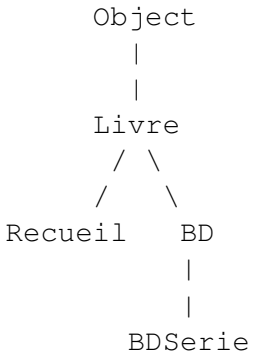
NFA032 : TP n°9 (héritage: masquage et surcharge)

29 mai 2018

On définit une architecture de classes représentant différentes catégories de livres :

— Classe des livres ayant un titre et un auteur.
— Classe des bandes dessinées ayant en plus un dessinateur.
— Classe des bandes dessinées appartenant à une série qui ont en plus un nom de série et un numéro dans la série.
— Classe des recueils comportant plusieurs œuvres du même auteur.

Cela n'épuise évidemment pas toutes les catégories de livres existantes. L'arbre d'héritage pour ces classes est le suivant :



Vous trouverez une archive zip contenant le code des quatre classes.

1 Affectations d'objets à des variables

Considérez le code suivant :

```
package livres;

public class MainLivres {
    public static void main(String[] args) {
        Object obj;
        Livre livre;
        BD bd;
        Recueil recueil;
        BDSerie bdserie;
        String[] tab = {"Le cousin Pons", "La cousine Bette"};
```

```
        obj = new Livre("Balzac","Gobseck");
        obj = new BD("Gotlib","Dans la joie jusqu'au cou","Alexis");
        obj = new BDSerie("Gosciny","La zizanie","Uderzo","Astérix",15);
        obj = new Recueil("Balzac","Oeuvres",tab);
        livre = new Livre("Balzac","Gobseck");
        livre = new BD("Gotlib","Dans la joie jusqu'au cou","Alexis");
        livre = new BDSerie("Gosciny","La zizanie","Uderzo","Astérix",15);
        livre = new Recueil("Balzac","Oeuvres",tab);
        bd = new Livre("Balzac","Gobseck");
        bd = new BD("Gotlib","Dans la joie jusqu'au cou","Alexis");
        bd = new BDSerie("Gosciny","La zizanie","Uderzo","Astérix",15);
        bd = new Recueil("Balzac","Oeuvres",tab);
        bdserie = new Livre("Balzac","Gobseck");
        bdserie = new BD("Gotlib","Dans la joie jusqu'au cou","Alexis");
        bdserie = new BDSerie("Gosciny","La zizanie","Uderzo","Astérix",15);
        bdserie = new Recueil("Balzac","Oeuvres",tab);
        recueil= new Livre("Balzac","Gobseck");
        recueil = new BD("Gotlib","Dans la joie jusqu'au cou","Alexis");
        recueil = new BDSerie("Gosciny","La zizanie","Uderzo","Astérix",15);
        recueil = new Recueil("Balzac","Oeuvres",tab);
    }
}
```

Une variable n'a pas forcément pour type le nom de la classe utilisée pour créer l'objet qu'elle contient (opération d'instanciation). Ci-dessous, toutes les combinaison entre un type pour la déclaration de variable et un type pour créer un objet sont envisagées. Certaines de ces combinaisons ne sont pas légales en Java et produisent une erreur de compilation.

- sans encore utiliser Eclipse, prédisez quelles lignes vont provoquer une erreur.
- copiez-collez le code dans Eclipse et vérifiez si votre prédiction était exacte.
- énoncez la règle qui permet de déterminer si une affectation d'un objet à une variable est légale (dans le cas d'un objet).

2 Affectations d'objets à des variables avec des conversions explicites

Certaines affectations d'objets à des variables qui ne passent pas la compilation deviennent possible avec des conversions de type explicites (cast). Le code suivant propose un certain nombre de telles affectations.

```
package livres;

public class MainLivre2 {
    public static void main(String[] args) {
        Object obj, objinst;
        Livre livre, livreinst;
        BD bd, bdinst;
        Recueil recueil, recueilinst;
        BDSerie bdserie, bdserieinst;
```

```
        String[] tab = {"Le cousin Pons", "La cousine Bette"};
        livreinst = new Livre("Balzac","Gobseck");
        bdinst = new BD("Gotlib","Dans la joie jusqu'au cou","Alexis");
        bdserieinst = new BDSerie("Gosciny","La zizanie","Uderzo","Astérix",15);
        recueilinst = new Recueil("Balzac","Oeuvres",tab);
        obj = (Object) livreinst;
        obj = (Object) bdinst;
        obj = (Object) bdserieinst;
        obj = (Object) recueilinst;

        livre = (Livre) livreinst;
        livre = (Livre) bdinst;
        livre = (Livre) bdserieinst;
        livre = (Livre) recueilinst;

        bd = (BD) livreinst;
        bd = (BD) bdinst;
        bd = (BD) bdserieinst;
        bd = (BD) recueilinst;

        bdserie = (BDSerie) livreinst;
        bdserie = (BDSerie) bdinst;
        bdserie = (BDSerie) bdserieinst;
        bdserie = (BDSerie) recueilinst;

        recueil = (Recueil) livreinst;
        recueil = (Recueil) bdinst;
        recueil = (Recueil) bdserieinst;
        recueil = (Recueil) recueilinst;
    }
}
```

On ne s'intéresse pour le moment qu'à la phase de compilation.

— sans encore utiliser Eclipse, prédisez quelles lignes vont provoquer une erreur de compilation.
— copiez-collez le code dans Eclipse et vérifiez si votre prédiction était exacte.
— énoncez la règle qui permet de déterminer si une conversion de type explicite est légale. Cette règle doit faire intervenir le type déclaré pour l'objet et le type mentionné entre parenthèse pour la conversion.

3 Erreur de conversion à l'exécution

Nous allons ne garder ici que les lignes qui passent à la compilation et tester si elles peuvent s'exécuter. Pour ce faire, nous utilisons le code suivant.

```
package livres;
```

```
public class LivreMain3 {
    public static void main(String[] args) {
        Object obj, objinst;
        Livre livre, livreinst;
        BD bd, bdinst;
        Recueil recueil, recueilinst;
        BDSerie bdserie, bdserieinst;
        String[] tab = {"Le cousin Pons", "La cousine Bette"};
        livreinst = new Livre("Balzac","Gobseck");
        bdinst = new BD("Gotlib","Dans la joie jusqu'au cou","Alexis");
        bdserieinst = new BDSerie("Gosciny","La zizanie","Uderzo","Astérix",15);
        recueilinst = new Recueil("Balzac","Oeuvres",tab);
        try{
            obj = (Object) livreinst;
        }catch(ClassCastException ex){
            System.out.println("Erreur  (Object) livreinst; ");
        }
        try{
            obj = (Object) bdinst;
        }catch(ClassCastException ex){
            System.out.println("Erreur (Object) bdinst;");
        }
        try{
            obj = (Object) bdserieinst;
        }catch(ClassCastException ex){
            System.out.println("Erreur (Object) bdserieinst;");
        }
        try{
            obj = (Object) recueilinst;
        }catch(ClassCastException ex){
            System.out.println("Erreur (Object) recueilinst;");
        }

        try{
            livre = (Livre) livreinst;
        }catch(ClassCastException ex){
            System.out.println("Erreur (Livre) livreinst;");
        }
        try{
            livre = (Livre) bdinst;
        }catch(ClassCastException ex){
            System.out.println("Erreur (Livre) bdinst;");
        }
        try{
            livre = (Livre) bdserieinst;
```

```
}catch(ClassCastException ex){
    System.out.println("Erreur (Livre) bdserieinst;");
}
try{
    livre = (Livre) recueilinst;
}catch(ClassCastException ex){
    System.out.println("Erreur (Livre) recueilinst;");
}
try{
    bd = (BD) livreinst;
}catch(ClassCastException ex){
    System.out.println("Erreur bd = (BD) livreinst;");
}
try{
bd = (BD) bdinst;
}catch(ClassCastException ex){
    System.out.println("Erreur (BD) bdinst;");
}
try{
    bd = (BD) bdserieinst;
}catch(ClassCastException ex){
    System.out.println("Erreur (BD) bdserieinst;");
}

try{
    bdserie = (BDSerie) livreinst;
}catch(ClassCastException ex){
    System.out.println("Erreur pour (BDSerie) livreinst;");
}
try{
bdserie = (BDSerie) bdinst;
}catch(ClassCastException ex){
    System.out.println("Erreur pour (BDSerie) bdinst");
}
try{
bdserie = (BDSerie) bdserieinst;
}catch(ClassCastException ex){
    System.out.println("Erreur pour (BDSerie) bdserieinst");
}

try{
recueil = (Recueil) livreinst;
}catch(ClassCastException ex){
    System.out.println("Errreur pour(Recueil) livreinst" );
}
```

```
    }
}
```

— sans encore utiliser Eclipse, prédisez quelles lignes vont provoquer une erreur à l'exécution (ce code se compile sans erreur).
— copiez-collez le code dans Eclipse, exécutez-le et vérifiez si votre prédiction était exacte.
— énoncez la règle qui permet de déterminer quand une conversion de type explicite provoque une erreur à l'exécution. Cette règle doit faire intervenir le type déclaré pour l'objet et le type mentionné entre parenthèse pour la conversion.

4 Quelle méthode est exécutée ?

Voici un code qui illustre certains phénomènes qui interviennent dans l'invocation des méthodes.

```
package livres;

public class LivreMain4 {
    public static void main(String[] args) {
        BD bdinst = new BD("Gotlib","Dans la joie jusqu'au cou","Alexis");
        BDSerie bdserieinst = new BDSerie("Fred","Time is money", "Alexis", "Timoléon",1);
        Livre declaredAsLivreButBDInstance = bdinst;
        System.out.println("bdinst == declaredAsLivreButBDInstance ? " + (bdinst == declaredAsLivreButBDInstance ));
        System.out.println(bdinst);
        System.out.println(declaredAsLivreButBDInstance);
        System.out.println(bdinst.compare(bdserieinst));
        System.out.println(declaredAsLivreButBDInstance.compare(bdserieinst));
    }
}
```

Ligne 8, on vérifie que les deux variables bdinst et declaredAsLivreButBDInstance contiennent le même objet (ce qui se voit aussi ligne 7). Ces deux variables ont deux types différents mais contiennent le même objet. Sur les deux variables, on appelle successivement les méthodes `toString` et `compare`. Ces deux méthodes sont définies dans chacune des deux classes `Livre` et `BD`.

— sans encore utiliser Eclipse, prédisez pour chacune des lignes 9 à 12, prédisez laquelle des méthodes est exécutée (par exemple *toString de Livre* ou *toString de BD*).
— exécutez le code pour vérifier vos prédictions.
— pourquoi le comportement est différent entre `toString` et `compare` ?

5 Explications

La méthode `toString` de `BD` redéfinit et masque la méthode `toString` de `Livre`. Un objet instance de la classe `BD` ne contient qu'une seule méthode `toString`, celle de la classe `BD`. C'est elle qui s'exécute chaque fois qu'on appelle cette méthode sur cette instance.

La méthode `compare` de `BD` ne rédéfinit pas et ne masque pas la méthode `compare` de `Livre` parce qu'elle n'a pas le même type. L'une attend un paramètre de type `BD`, l'autre un paramètre de type `Livre`. Dans un objet instance de la classe `BD`, il y a deux méthodes `compare` avec le même nom mais de stypes

de paramètres différents. Pour des raisons qu'on ne veut pas détailler, ce n'est pas celle qu'on voudrait qui est exécutée ligne 12 du code précédent.

Une méthode masque et redéfinit une méthode héritée de sa super-classe si et seulement si elle a le même type pour ses paramètres et la valeur qu'elle renvoie.

6 La directive @Override

Vous avez peut-être remarqué que la méthode `toString` est précédée d'une ligne contenant le code `@Override`. Il s'agit de ce qu'on appelle une *annotation*. Cela ne modifie pas le code de la méthode et ne change pas son comportement à l'exécution. C'est une information donnée au compilateur à propos de notre intention de redéfinir et masquer une méthode héritée. Dès lors le compilateur peut vérifier que c'est bien le cas.

Dans notre exemple, la méthode `compare` est définie pour la première fois dans `Livre`. On ne doit pas utiliser `@Override` à cet endroit car il n'y a pas de méthode `compare` héritée de la classe `Object`.

Dans la classe `BD`, si notre intention est de redéfinir et masquer `compare` on peut utiliser l'annotation. Essayez de le faire et constatez que dans ce cas, le compilateur ou Eclipse signale une erreur.

C'est une utile précaution d'utiliser l'annotation systématiquement lorsqu'on désire redéfinir une méthode.

7 Correction du code

On voudrait que la méthode `compare` de `BD` masque et redéfinisse la méthode `compare` de sa super-classe. Pour ce faire, il faut changer le type du paramètre : ce paramètre doit être de type `Livre`.

Nous vous proposons la démarche suivante :

— Recevoir un paramètre de type `Livre`.
— Essayer d'affecter ce paramètre à une variable de type `BD` au moyen d'une conversion explicite.
— Si la conversion réussit, comparer les dessinateurs des deux objets.
— Si la conversion échoue (une exception est levée par la conversion), rattrapper cette exception et ne rien faire de spécial.
— Dans tous les cas, comparer l'auteur et le titre.

8 Comparaison de recueils

En utilisant le même principe, ajoutez une méthode `compare` à la classe `Recueil` qui détecte les œuvres communes à deux recueils. Attention, les œuvres ne sont pas nécessairement dans la même position dans les deux recueils.

9 À rendre

Rendez le code du projet au moyen du formulaire prévu à cet effet.