# TP11 – Générisation suite et fin

Avant l'important TP12 de sécurisation, il reste ce court (mais technique) TP où on va finir de génériser :

- La création d'un objet,
- La modification d'un objet.

Dupliquez cotre code du `TP10/ex6` en `TP11/ex1`.

## Exercice 1 – les méthodes liées à la création d'un objet

### 1.a – la méthode addObjet du modèle Objet

Voici deux exemples de méthode d'ajout dans la base de données.

```php
public static function addAuteur($n,$p,$a) {
    $requetePreparee = "INSERT INTO Auteur(`nom`,`prenom`,`anneeNaissance`) VALUES(:n,:p,:a);";
    $req_prep = Connexion::pdo()->prepare($requetePreparee);
    $valeurs = array("n" => $n,"p" => $p,"a" => $a);
    try {
        $req_prep->execute($valeurs);
        return true;
    } catch(PDOException $e) {
        return false;
    }
}
```

```php
public static function addCategorie($l,$n) {
    $requetePreparee = "INSERT INTO Categorie(`libelle`,`nbLivresAutorises`) VALUES(:l,:n);";
    $req_prep = Connexion::pdo()->prepare($requetePreparee);
    $valeurs = array("l" => $l,"n" => $n);
    try {
        $req_prep->execute($valeurs);
        return true;
    } catch(PDOException $e) {
        return false;
    }
}
```

Elles sont conçues de la même façon. Elles se différencient par leur nom (ce qui va s'arranger car elles seront générisées en addObjet), et surtout par les données transmises :

- Le nombre de champs transmis,
- Le nom des champs en question.

Nous allons résoudre ceci :

- en passant non pas des arguments, mais un tableau d'arguments (un peu comme pour le constructeur générique),
- en construisant la requête d'insertion à partir de ce tableau d'arguments,
- le tableau contenant les valeurs à donner aux tags sera le tableau d'arguments.

Dans la classe `Objet`, commencez le codage d'une méthode

```
// méthode d'insertion générique
public static function addObjet($tableauDonnees) {

}
```

qui aura pour mission d'insérer un objet dans la bonne table, en utilisant le paramètre `$tableauDonnees` qui contiendra tous les renseignements pour l'insertion. Pour le moment, on suppose que le contrôleur, qui agit en amont, a bien construit `$tableauDonnees`. Mettons en place la méthode du modèle `Objet`. Pour cela :

- Récupérez comme d'habitude les attributs `protected static` qui donnent le nom de la table et la clé primaire.

- Construisez, dans un parcours `foreach` de `$tableauDonnees`, la requête préparée. Vous aurez à fabriquer quelque chose qui soit cohérent avec les deux captures précédentes. Attention :
  - Les noms des champs (cf captures) sont encadrés de guillemets « alt gr 7 ».
  - Le dernier nom de champ n'est pas suivi d'une virgule, mais d'une parenthèse fermante.
  - Idem pour le dernier tag : pas de virgule mais parenthèse fermante.

- Exécutez la requête préparée avec, comme paramètre de la méthode `execute`, le tableau `$tableauDonnees`. On renverra `true` si ça se passe bien, `false` sinon.

## 1.b – la méthode creerObjet du controleurObjet

Si vous examinez, dans les contrôleurs, le code de `creerAuteur` ou celui de `creerCategorie`, vous devez remarquer que tous les éléments récupérés du `$_GET`, à l'exception de « `action` » et de « `controleur` », sont listés comme clés du tableau static `$tableauChamps`. Créez dans `ControleurObjet` la méthode `creerObjet` qui :

- Récupère le nom de la table,
- Récupère le tableau des champs,
- Structure un tableau `$tableauDonnees` qui est la sous-partie vraiment utile de `$_GET`. Attention, tout n'est pas à prendre dans le `$_GET` : ni `action`, ni `controleur`…
- Appelle la méthode `addObjet` correspondant à la table récupérée en static (attention, l'erreur classique est d'appeler `Objet::addObjet`, ce qui ne fonctionne pas car dans ce cas le serveur ne lit pas les attributs `static` nécessaires). Le booléen résultat est récupéré dans une variable.
- Si ce booléen est `true`, alors on affiche la liste des objets (où apparaîtra le nouvel objet créé), sinon on retourne au formulaire de création.

Enfin, détails importants :

- dans le formulaire de création d'un objet, n'oubliez pas de modifier l'action en `creerObjet` !
- dans la méthode `creerCompteAdherent`, corrigez l'appel à `addAdherent` en `addObjet`, avec un envoi non pas d'arguments, mais d'un tableau d'arguments ! Ce tableau est une trace non générique du code : il devra reprendre toutes les paires clé/valeur d'un Adhérent :

```php
$b = Adherent::addObjet(
  array(
    "login" => $l,
    "mdp" => $m,
    "nomAdherent" => $n,
    "prenomAdherent" => $p,
    "email" => $e,
    "dateAdhesion" => $d,
    "isAdmin" => 0,
    "numCategorie" => $c,
    "chaineValidationEmail" =>$ch
  )
);
```

Faites ensuite quelques essais de création, et quand tout fonctionne, supprimez toutes les méthodes spécifiques des modèles et des contrôleurs qui servaient précédemment à la création.

## Exercice 2 – les méthodes liées à la modification d'un objet

Dupliquez cotre code du `TP11/ex1` en `TP11/ex2`.

### 2.a – la méthode updateObjet du modèle Objet

Codez cette méthode, qui ressemble comme une jumelle à `addObjet`. Elle va gérer un argument `$tableauDonnees`, et construire cette fois une requête préparée de type `UPDATE`, au moyen d'un parcours de l'argument `$tableauDonnees`.

La seule petite différence est que le champ identifiant n'intervient que dans le `WHERE`. Donc on fera attention au moment de la construction de la requête préparée.

Conseil : dans la méthode `updateObjet`, faire un `echo` de la requête préparée pour voir si elle est bien construite, car il y a de grandes chances que vous n'y arriviez pas du premier coup.

Cette requête préparée sera exécutée avec comme paramètre le tableau `$tableauDonnees`, et retournera, elle aussi, un booléen (`true` si la mise à jour s'est bien passée, `false` sinon).

### 2.b – la méthode modifierObjet du controleurObjet

Comme pour creerObjet :

- On récupère en static le nom de la table, le tableau des champs, mais aussi le nom de la clé primaire.

- On structure un tableau `$tableauDonnees` qui est la sous-partie vraiment utile de `$_GET`. Attention, tout n'est pas à prendre dans le `$_GET` : ni `action`, ni `controleur`… Mais il y a un détail supplémentaire : si le champ est « `identifiant` », alors on l'ajoute dans `$tableauDonnees` sous la forme

  ```
  $tableauDonnees[$identifiant] = $valeur;
  ```

  sinon, on l'ajoute sous la forme

  ```
  $tableauDonnees[$champ] = $valeur;
  ```

  Ainsi, dans la méthode addObjet du modèle, on pourra faire le tri entre ce qui va dans le WHERE (le champ qui identifie) et le reste.

- On appelle la méthode `updateObjet` correspondant à la table récupérée en static (attention, l'erreur classique est d'appeler `Objet::updateObjet`). Le booléen résultat est récupéré dans une variable.

- Si ce booléen est `true`, alors on affiche la liste des objets (où apparaîtra le nouvel objet créé), sinon on retourne au formulaire de modification.

Enfin, détail important : dans le formulaire de modification d'un objet, n'oubliez pas de modifier l'action en `modifierObjet` !

Faites ensuite quelques essais de modification, et quand tout fonctionne, supprimez toutes les méthodes spécifiques des modèles et des contrôleurs qui servaient précédemment à la modification.

Si vous en êtes là, vous pouvez être content de vous !