TP de JAVA – Série 3. Introduction à la notion d'objets et de classes

Université de Mons-Hainaut

2004 - 2005

Préliminaires

Téléchargez sur le site des tp le fichier : `BankAccount.java`

Notez bien dans ce code la syntaxe des commentaires : ceux-ci sont destinés à produire de la documentation au format html.
À l'avenir il faudra utiliser ces commentaires.

Exercices de révision

R3.1 Expliquer la différence entre :

```
BankAccount b;
```

et :

```
BankAccount b = new BankAccount(500);
```

Exercices de révision

R3.1 Expliquer la différence entre :

```
BankAccount b;
```

et :

```
BankAccount b = new BankAccount(500);
```

R3.2 Quels sont les paramètres de construction pour un objet de la classe `BankAccount` ?

Exercices de révision

R3.1 Expliquer la différence entre :

`BankAccount b;` et :

```
BankAccount b = new BankAccount(500);
```

R3.2 Quels sont les paramètres de construction pour un objet de la classe `BankAccount` ?

R3.3 Donner le code qui construit l'objet suivant :
un compte bancaire avec un solde de 500 euros,
(d'abord sans initialiser de variable, ensuite avec).

Exercices de révision

R3.4 Trouver les erreurs dans le code suivant :

```
double x = BankAccount(1000).getBalance();

BankAccount b;
b.deposit(1000);

b = new BankAccount(1000);
b.add("one million bucks");
```

Exercices de révision

R3.4 Trouver les erreurs dans le code suivant :

```
double x = BankAccount(1000).getBalance();

BankAccount b;
b.deposit(1000);

b = new BankAccount(1000);
b.add("one million bucks");
```

R3.5 Décrire tous les constructeurs de la classe `BankAccount`. Lister toutes les méthodes qui peuvent être utilisées pour changer un objet de type `BankAccount`. Lister toutes celles qui ne changent pas un objet de type `BankAccount`.

Exercices de révision

R3.6 Quelle est la valeur de b après les opérations suivantes :

```
BankAccount b = new BankAccount(10);
b.deposit(5000);
b.withdraw(b.getBalance() / 2);
```

Exercices de révision

R3.6 Quelle est la valeur de b après les opérations suivantes :

```
BankAccount b = new BankAccount(10);
b.deposit(5000);
b.withdraw(b.getBalance() / 2);
```

R3.7 Soient b1 et b2 des variables qui référencent des objets de la classe BankAccount. Considérer les instructions suivantes :

```
b1.deposit(b2.getBalance());
b2.deposit(b1.getBalance());
```

Est-ce qu'après ceci les soldes (bancaires) des objets référencés par b1 et b2 sont égaux ? Expliquer.

Exercices de révision

R3.8 Expliquer la signification de la référence `this`.

Exercices de révision

R3.8 Expliquer la signification de la référence `this`.

R3.9 Que fait la méthode suivante ? Donner un exemple de comment on peut appeler cette méthode.

```
public class BankAccount
{
   public void mystery (BankAccount that, double amount)
   {
   this.balance = this.balance - amount;
   that.balance = that.balance + amount;
   }
   ... //other BankAccount methods
}
```

Exercices de Programmation

P3.1 Ajoutez une méthode `sayGoodbye` à la classe `Greeter`.

Exercices de Programmation

P3.1 Ajoutez une méthode `sayGoodbye` à la classe `Greeter`.

P3.2 Ajoutez une méthode `refuseHelp` à la classe `Greeter`.
Elle doit retourner une chaîne de caractère semblable à :
`"Desole Damien :  cela est impossible."`

Exercices de Programmation

P3.1 Ajoutez une méthode `sayGoodbye` à la classe `Greeter`.

P3.2 Ajoutez une méthode `refuseHelp` à la classe `Greeter`. Elle doit retourner une chaîne de caractère semblable à :
`"Desole Damien :  cela est impossible."`

P3.3 Écrire un programme qui construit un compte bancaire, fait un dépôt de 1000 euros, retire 500 euros, puis encore retire 400 euros, et alors affiche le solde restant.

Exercices de Programmation

P3.1 Ajoutez une méthode `sayGoodbye` à la classe `Greeter`.

P3.2 Ajoutez une méthode `refuseHelp` à la classe `Greeter`.
Elle doit retourner une chaîne de caractère semblable à :
`"Desole Damien :  cela est impossible."`

P3.3 Écrire un programme qui construit un compte bancaire, fait un dépôt de 1000 euros, retire 500 euros, puis encore retire 400 euros, et alors affiche le solde restant.

P3.4 Ajouter une méthode :
`void addInterest(double rate)`
à la classe `BankAccount` qui ajoute un intérêt à un taux donné au solde du compte.

Exercices de Programmation

P3.5 Écrire une classe **SavingAccount** similaire à la classe **BankAccount** avec une variable d'instance supplémentaire **interest**.

Fournir un constructeur qui fixe le solde initial (**balance**) et le taux d'interet (**interest**).

Fournir une méthode (**addInterest**) (sans paramètre explicite) qui ajoute les intérêts au compte.

Ecrire un programme qui construit un objet **SavingAccount** avec un solde initial de 1000 euros et un taux d'intérêts de 10%.

Appliquer la méthode **addInterest** 5 fois et afficher le solde résultat.

Exercices de Programmation

P3.6 Ecrire une classe **Employee.**

Un employé possède un nom (**String**) et un salaire (**double**).

Fournir un constructeur par défaut avec 2 paramètres (nom et salaire), et des méthodes pour retourner lce nom et ce salaire.

Ecrire un petit programme pour tester cette classe.

Exercices de Programmation

P3.7 Compléter la classe de l'exercice précédent en ajoutant une méthode **raiseSalairy(double percent)** qui augmente le salaire de l'employé d'un certain pourcentage. Exemple d'utilisation:

```
Employee harry = new Employee("Hacker, Harry", 55000);
harry.raise(10);//Harry a une augmentation de 10%
```

Exercices de Programmation

P3.8 Ecrire une classe **Car** avec les propriétés suivantes:
Un objet **Car** a une certaine consommation (mesuré en kilomètres parcourus avec un litre) et un certain volume de carburant dans le réservoir. La consommation est spécifiée par le ocntrusteur, et le niveau initial de carburant dans le réservoir est de 0.
Fournir une méthode **drive** qui simule la conduite du véhicule pour une certaine distance, réduisant la quantité de carburant disponible dans le réservoir; une méthode **getGas** qui retourne le niveau de carburant actuel dans ce réservoir, et une méthode **addGas** pour faire le plein.

Exercices de Programmation

Exemple d'utilisation:

```
Car myBeemer = new Car(19);
        //19 kilomètres avec un litre
myBeemer.addGas(20);
        //on ajoute 20 litres
myBeemer.drive(100);
        //on parcoure 100 kilomètres
System.out.println(myBeemer.getGas());
        //affiche le nombre de litres de carburant restant
```

Exercices de programmation

P3.9 Ecrire une classe **Student.**

Pour les besoins de l'exercice, un étudiant possède un nom et un score total.

Fournir un constructeur approprié et les méthodes **getName()**, **addQuiz(int score)**, **getTotalScore()** et **getAverageScore().**

Il sera nécessaire de conserver le *nombre de cotes* obtenues par l'étudiant.

Exercices de programmation

P3.10 Ecrire une classe **Product.** Un produit possède un nom et un prix, par exemple new Product("Toaster", 29.95).
Fournir 2 constructeurs, dont l'un ne prendra aucun paramètre et fixera le nom à "" et le prix à 0 euro.
Fournir des méthodes **getName()**, **getPrice()** et **setPrice().**
Ecrire un programme qui créé 2 produits, afficher leurs noms et leurs prix, réduire leurs prix de 5 euros et réafficher les informations de noms et de prix.

Exercices de programmation

P3.11 Ecrire une classe **Square** avec les méthodes **getArea()** et **getPerimeter()**.

Le constructeur prend en paramètre le côté du carré.

Exercices de programmation

P3.12 Ecrire une classe **PopulationMouche** qui simule l'accroissement d'une population de mouches.
Le constructeur prend en paramètre la taille de la population initiale de mouches.
La méthode **waitForDoubling** simule une période durant laquelle la population de mouches double.
La méthode **spray** simule la vaporisation avec un insecticide, ce qui réduit la population de 10%.
La méthode **getMouches** retourne le nombre courant de mouches.
Ecrire cette classe ainsi qu'un programme de test qui simulera la situation dans une cuisine avec 10 mouches au départ.

Attendre, vaporiser, afficher le nombre de mouches. Répéter cette opération 3 fois.

Exercices de programmation

P3.13 Ecrire une classe **PopulationLapin** qui simule l'accroissemnt d'une population de lapins. Les règles sont les suivantes:
Commencer avec un couple de lapins. Les lapins sont capablent de s'accoupler à partir de l'âge de 1 mois. Un mois plus tard la femelle donne naissance à un autre couple de lapins.
Supposons que les lapins ne meurent jamais et que la femelle donnent toujours naissance à un couple de lapins (un mâle et une femelle) chaque mois à partir de son deuxième mois de vie.

Exercices de programmation

Ecrire une méthode **waitAMonth** qui attend un mois, et une méthode **getPairs** qui affiche le nombre courant de couples de lapins.
Ecrire un programme de test qui simulera l'accroissement de la population de lapins sur 10 mois.

Astuce: Utiliser un champ d'instance pour repérer les couples de lapins qui viennent de naître et ceux qui ont plus d'un mois d'existence.