POLYTECH NICE-SOPHIA
Université Nice Sophia Antipolis
Membre de UNIVERSITÉ CÔTE D'AZUR

Découvrir la programmation orientée objet avec Java

Travaux pratiques

frederic.rallo@univ-cotedazur.fr

TP4 : Exceptions

L'objectif des exercices ci-dessous est de mettre en œuvre les collections, les types énumérés (enum) et les exceptions en Java.

Exercice 1

Le but de cet exercice est de réaliser une classe EnsembleEntierBorne. Ajoutez une classe dans votre projet, package ex4. Cette classe devra permettre de manipuler des ensembles de taille bornée de nombres entiers .

1. **Attributs :** Chaque instance de EnsembleEntierBorne dispose au moins des attributs :
   a. une constante entière MAXIMUM qui contient le nombre maximum d'éléments de l'ensemble,
   b. un tableau de valeurs.

2. **Constructeurs :** l'unique constructeur prend en paramètre la taille de l'ensemble ainsi que la plus grande valeur que peut contenir l'ensemble . En effet, l'ensemble que vous construisez ne peut contenir que des entiers compris entre 0 (inclus) et cet entier (inclus).

3. **Méthodes :**
   - toString() Affiche l'ensemble sous forme {2,4,6,7,99}. Ce sera l'occasion de regarder la javadoc de la classe String et notamment la méthode substring de la classe.
   - add(…) Cette procédure prend un entier en paramètre et le rajoute à l'ensemble (rappel : un ensemble ne peut pas contenir de doublon).
   - remove(…) qui permet de supprimer un élément.
   - findOccurence(…) Cette fonction vérifie si une valeur est dans l'ensemble. Elle retourne la valeur -2 si l'ensemble est plein val n'est pas dans l'ensemble ; -1 si l'ensemble n'est pas plein mais que val n'est pas dans l'ensemble ; ou l'indice si la valeur cherchée est dans l'ensemble (indice est la première occurrence)
   - union(…) Cette fonction renvoie un nouvel ensemble correspondant à l'union entre l'ensemb le courant et celui entré en paramètre.
   - intersect(…) Cette fonction renvoie un nouvel ensemble correspondant à l'intersection entre l'ensemble courant et celui entré en paramètre.

4. **Main :** Créez la classe `TestEntiersBornes` dans le package miseEnOeuvre, la méthode main qui pourra instancier des objets EnsembleEntierBorne et tester les méthodes développées.

Découvrir la programmation orientée objet avec Java

Travaux pratiques

frederic.rallo@univ-cotedazur.fr

```
* ------------------------------------------------------------ *
* TP java                                                      *
*                                                              *
* @author frédéric rallo - frederic.rallo@univ-cotedazur.fr   *
* @version TD2   - ex4                                         *
* ------------------------------------------------------------ *

ensemble e1 = { }
ensemble e2 = { }

TEST DE LA METHODE add()
on ajoute --> 5 à e1
 -->  ensemble  e1 = { 5 }
 -->  ensemble  e2 = { }

on ajoute --> 5 (encore) à e1
 -->  ensemble  e1 = { 5 }
 -->  ensemble  e2 = { }

on ajoute --> 1 à e1
 -->  ensemble  e1 = { 5,1 }
 -->  ensemble  e2 = { }

on ajoute --> 2 à e1
 -->  ensemble  e1 = { 5,1,2 }
 -->  ensemble  e2 = { }

on ajoute --> 1 à e2
 -->  ensemble  e1 = { 5,1,2 }
 -->  ensemble  e2 = { 1 }

on ajoute --> 2 à e2
 -->  ensemble  e1 = { 5,1,2 }
 -->  ensemble  e2 = { 1,2 }

on ajoute --> 6 à e2
 -->  ensemble  e1 = { 5,1,2 }
 -->  ensemble  e2 = { 1,2,6 }

on ajoute --> 3 à e2
 -->  ensemble  e1 = { 5,1,2 }
 -->  ensemble  e2 = { 1,2,6 }

on ajoute --> 4 à e2
 -->  ensemble  e1 = { 5,1,2 }
 -->  ensemble  e2 = { 1,2,6 }

on ajoute --> 5 à e2
 -->  ensemble  e1 = { 5,1,2 }
 -->  ensemble  e2 = { 1,2,6 }

TEST DE LA METHODE remove()
```

Découvrir la programmation orientée objet avec Java

Travaux pratiques

frederic.rallo@univ-cotedazur.fr

```
on supprime --> 4 à e2
 -->  ensemble  e1 = { 5,1,2 }
 -->  ensemble  e2 = { 1,2,6 }
on supprime --> 3 à e2
 -->  ensemble  e1 = { 5,1,2 }
 -->  ensemble  e2 = { 1,2,6 }

TEST DE LA METHODE union()
ensemble e1 = { 5,1,2 }
ensemble e2 = { 1,2,6 }
 --> ensemble e1 UNION e2 = { 5,1,2,6 }

TEST DE LA METHODE intersect()
ensemble e1 = { 5,1,2 }
ensemble e2 = { 1,2,6 }
 --> ensemble e1 INTER e2 = { 1,2 }
```

Exercice 2

Cet exercice fait suite au TP3. Les menus sont caractérisés par leur nom, un créateur (qui est un Restaurant non récent), des distributeurs (qui sont des Restaurants). Parmi les distributeurs on peut y retrouver le restaurant créateur (ou pas !). Les distributeurs de Menus seront décrits dans une collection

1. Écrire la classe Menu.java avec les attributs, accesseur(s) et constructeur(s) en respectant les bonnes pratiques du java

2. Écrire la méthode toString() dans la classe Menu.java qui indiquera son créateur, le nom du menu et l'année de création de tous les restaurants qui le distribuent.

3. Écrire une classe TestMenu.java disposant d'un point d'entrée du programme
   - Créer le menu « Rapid » inventé par chez fred
   - Créer le menu « Prestige » inventé par l« la cantine »
   - Les restaurants « La cantine » et « Le miam » distribuent le menu rapid

```
* ------------------------------------------------------- *
* TP java                                                 *
*                                                         *
* @author frédéric rallo - frederic.rallo@univ-cotedazur.fr *
* @version TD3 - ex7                                      *
* ------------------------------------------------------- *
Menu "Prestige", created by "Jarrerie" (2019) is deal by []
Menu "Rapid", created by "Chez Fred" (1970) is deal by ["La cantine" (1972), "Le miam"
(2020)]
```

Découvrir la programmation orientée objet avec Java

Travaux pratiques

frederic.rallo@univ-cotedazur.fr

Exercice 3

Il existe une base de données dans laquelle sont répertoriés tous les Restaurants et tous les Menus. Cette base de données permet d'ajouter un Restaurant, de le Supprimer, d'ajouter un Menu et de le supprimer

1. Écrire la classe Database.java

2. Précisez en commentaire de la javadoc le type de relation avec les autres objets

3. Écrire la méthode getRecentRestaurants() qui retourne la liste de tous les Restaurants récents

4. Écrire la méthode getProspects(Menu menu) qui retournera la liste des Restaurants qui n'affichent pas le menu passé en paramètre

5. Tester la base de données en créant :
   - le Restaurant « chez Fred » (1970)
   - le Restaurant « la cantine» (1972)
   - le Restaurant « la jarrerie » (2019)
   - le Restaurant « le miam » (2020)
   - le Menu « rapide» distribué chez « la cantine » et « le miam »

   → Affichez les prospects pour le menu « rapide »

```
* ------------------------------------------------------- *
* TP java                                                 *
*                                                         *
* @author frédéric rallo - frederic.rallo@univ-cotedazur.fr *
* @version TD3 - ex8                                      *
* ------------------------------------------------------- *
["Chez Fred" (1970), "La jarrerie" (2019)]
```

Découvrir la programmation orientée objet avec Java

Travaux pratiques

frederic.rallo@univ-cotedazur.fr

Exercice 4

Reprenez les classes du TP3 et écrivez la classe **TownException** héritant de la classe Exception. La classe **TownException** ne contient pas d'attribut1 (mails elle pourrait en contenir). Son rôle sera d'afficher une message d'erreur adéquat.

- Reprenez vos classe **Villes.java Capitale.java et Patelin.java et Main.java du dernier exercice du TP précédent.**
- Vos constructeurs seront le plus simple possible (utilisez this())
- Vous devrez respecter les règles suivante
    - un Patelin a au moins 500 habitants et au plus 5000 habitants
    - une Ville a au moins 5001 habitants et au plus 100000 habitants
    - une Capitale a au moins 100001 habitants
    - Les instances de Ville, Patelin et Capitale ont un nom
    - Les instances de Capitale représentent un Pays

Découvrir la programmation orientée objet avec Java

Travaux pratiques

frederic.rallo@univ-cotedazur.fr

```
* ------------------------------------------------------- *
* TP java                                                 *
*                                                         *
* @author frédéric rallo - frederic.rallo@univ-cotedazur.fr *
* @version TD4 - ex1                                     *
* ------------------------------------------------------- *
tabVille[0] = Capitale("Paris",558889,"France")
tabVille[1] = Capitale("Londres","Angleterre")
TownException error: le nombre d'habitant ne permet pas de créer la Capitale Londres
tabVille[2] = Ville("Nice",354000)
tabVille[3] = Capitale("Cagnes-sur-Mer",154000)
TownException error: la Capitale Cagnes-sur-Mer doit représenter un pays
tabVille[4] = Patelin("Séranon",400)
TownException error: le nombre d'habitant ne permet pas de créer le Patelin Séranon
tabVille[5] = Patelin("Ovignon")
TownException error: le nombre d'habitant ne permet pas de créer le Patelin Ovignon
tabVille[6] = Ville()
TownException error: le nom est obligatoire pour créer une Ville
tabVille[7] = Patelin()
TownException error: le nom est obligatoire pour créer un Patelin
tabVille[8] = Capitale()
TownException error: le nom est obligatoire pour créer une Capitale
tabVille[9] = Capitale("Grasse")
TownException error: le nombre d'habitant ne permet pas de créer la Capitale Grasse

* ------------------------------------------------------- *
voici les villes :
      - Paris est une .Capitale de 558889 habitants. La catégorie est : C
      - Nice est une .Ville de 354000 habitants. La catégorie est : A
```

Découvrir la programmation orientée objet avec Java

Travaux pratiques

frederic.rallo@univ-cotedazur.fr

Exercice 5

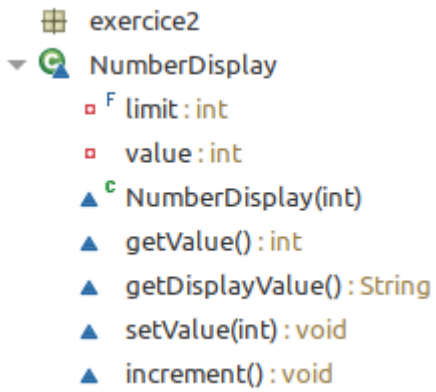
On souhaite réaliser un dispositif pour afficher l'heure. Vous devrez créer les Classes ClockDisplay.java et NumberDisplay.java

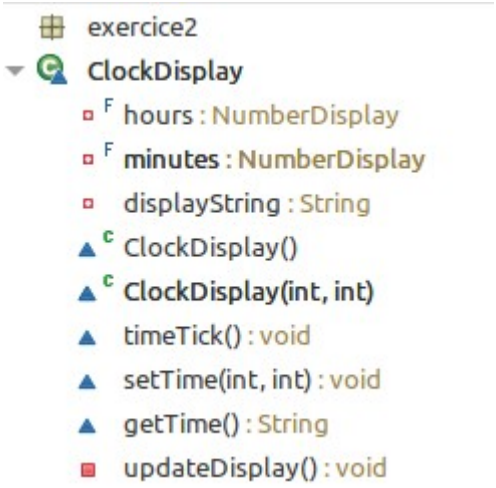
Pour ce projet, on vous fournit l'interface

```
/**
 * This interface must be implemeted to display the time
 * @author fred
 *
 */
interface IDisplay {
	/**
	* Update the internal string that represents the display.
	*/
	void updateDisplay();

	/**
	* Set the time of the display to the specified hour and minute.
	*/
	void setTime(int hour, int minute) throws ClockDisplayException;

	/**
	* This method should get called once every minute -
	* it makes the clock display go one minute forward.
	*/
	void timeTick();
}
```





Découvrir la programmation orientée objet avec Java

Travaux pratiques

frederic.rallo@univ-cotedazur.fr

Votre main devra

- créer une instance de ClockDisplay,
- effectuer 1000 fois la méthode timeTick()
- mettre à jour l'heure avec la valeur 14:55
- mettre à jour l'heure avec la valeur 14:75
- mettre à jour l'heure avec la valeur 24:55
- afficher l'heure courante

```
* ------------------------------------------------------- *
* TP java                                                       *
*                                                         *
* @author frédéric rallo - frederic.rallo@univ-cotedazur.fr *
* @version TD4 - ex2                                     *
* ------------------------------------------------------- *

clockDisplay.getTime() = 00:00
clockDisplay.getTime() = 16:40
clockDisplay.getTime() = 14:55
ClockDisplayException error: this time (14:75) is impossible
clockDisplay.getTime() = 14:55
ClockDisplayException error: this time (24:55) is impossible
clockDisplay.getTime() = 14:55
```

Découvrir la programmation orientée objet avec Java

Travaux pratiques

frederic.rallo@univ-cotedazur.fr

Exercice 6

On désire de créer un jeu de cartes de Pokémon. Chaque personnage ayant
- un nom
- un type (eau, feu...)
- des points de vie
- des points d'attaque
- des points de défense

- Piochez 2 cartes dans le paquet
- Afficher la description des combattants (pt de vie, pt attaque et pt défense)
- Simulez une succession de 10 combats :
  → Aux points de vie de chaque combattant on soustrait les point d'attaque de l'adversaire et on ajoute ses propres points de défense

voici un exemple de trace :

```
* ---------------------------------------------------------- *
* TP java                                                    *
*                                                            *
* @author frédéric rallo - frederic.rallo@univ-cotedazur.fr *
* @version TD4 - ex3                                        *
* ---------------------------------------------------------- *

This is packet contains:
SALAMECHE
ODDISH
VENONAT
TENTACOOL

ODDISH (life=10, attack=6, defense=2) vs SALAMECHE (life=10, attack=4, defense=5)
   ===> ODDISH (life=8, attack=6, defense=2) and SALAMECHE (life=9, attack=4, defense=5)

VENONAT (life=10, attack=9, defense=1) vs SALAMECHE (life=9, attack=4, defense=5)
   ===> VENONAT (life=7, attack=9, defense=1) and SALAMECHE (life=5, attack=4, defense=5)

SALAMECHE (life=5, attack=4, defense=5) vs VENONAT (life=7, attack=9, defense=1)
   ===> SALAMECHE (life=1, attack=4, defense=5) and VENONAT (life=4, attack=9, defense=1)

ODDISH (life=8, attack=6, defense=2) vs SALAMECHE (life=1, attack=4, defense=5)
SALAMECHE is dead!
   ===> ODDISH (life=6, attack=6, defense=2)

ODDISH (life=6, attack=6, defense=2) vs VENONAT (life=4, attack=9, defense=1)
ODDISH is dead!
VENONAT is dead!
   ===> ODDISH (life=6, attack=6, defense=2)
```