Introduction à la programmation et au développement d'applications en Java

Membres statiques

Introduction

- Membre d’instance vs membre statique
- Champ statique
- Méthode statique
- Blocs d’initialisation statiques

Membres statiques

- Jusqu'à présent, en définissant une classe, nous n'avons rencontré que des **membres d'instance** (**champs et méthodes d'instance**), c'est-à-dire qui sont **directement associés à une instance** spécifique de la classe (objet)
- Parfois, on a besoin de **membres de classes**, qui seront accessibles et partagés par toutes les instances de la classe : ce sont les **membres statiques**
- Un membre statique n'est donc pas associé à une instance spécifique
- La déclaration d'un membre statique est similaire à une déclaration de membre d'instance, il faut juste ajouter le mot-clé `static`

Membres statiques

- **Champ statique** :
  - sa valeur n'est pas associée à une instance, mais à toute la classe ;
  - toutes les instances accèdent donc à la même valeur partagée ;
  - accès en nommant la classe, pas l'instance :

```
MaClasse.monChampStatique = 1;
```

- **Méthode statique** :
  - exécute une action qui n'est pas spécifique à une instance précise ;
  - ne peut donc accéder qu'aux champs statiques, pas aux champs d'instance ;
  - accès en nommant la classe, pas l'instance :

```
MaClasse.maMethodeStatique();
```

```
public class Vol {
  // ... divers autres membres ...
  private int nbPassagers;


  private void ajouterUnPassager() {
    if (placeDispo()) {
      nbPassagers++;

    } else {
      gererTropNombreux();
    }
  }









}
```

```
public class Vol {
  // ... divers autres membres ...
  private int nbPassagers;
  private static int nbTotalPassagers;

  private void ajouterUnPassager() {
    if (placeDispo()) {
      nbPassagers++;

    } else {
      gererTropNombreux();
    }
  }









}
```

```
public class Vol {
  // ... divers autres membres ...
  private int nbPassagers;
  private static int nbTotalPassagers;

  private void ajouterUnPassager() {
    if (placeDispo()) {
      nbPassagers++;
      nbTotalPassagers++;
    } else {
      gererTropNombreux();
    }
  }

}
```

```
public class Vol {
  // ... divers autres membres ...
  private int nbPassagers;
  private static int nbTotalPassagers;

  private void ajouterUnPassager() {
    if (placeDispo()) {
      nbPassagers++;
      nbTotalPassagers++;
    } else {
      gererTropNombreux();
    }
  }

  public static int getNbTotalPassagers() {
    return nbTotalPassagers;
  }

  public static void resetNbTotalPassagers(){
    nbTotalPassagers = 0;
  }
}
```

```
Vol.resetNbTotalPassagers();
System.out.println(
 Vol.getNbTotalPassagers());// 0
```

```
public class Vol {
  // ... divers autres membres ...
  private int nbPassagers;
  private static int nbTotalPassagers;

  private void ajouterUnPassager() {
    if (placeDispo()) {
      nbPassagers++;
      nbTotalPassagers++;
    } else {
      gererTropNombreux();
    }
  }

  public static int getNbTotalPassagers() {
    return nbTotalPassagers;
  }

  public static void resetNbTotalPassagers(){
    nbTotalPassagers = 0;
  }
}
```

```
Vol.resetNbTotalPassagers();
System.out.println(
 Vol.getNbTotalPassagers());// 0

Vol v1 = new Vol();
v1.ajouterUnPassager();
v1.ajouterUnPassager();
```

```
public class Vol {
  // ... divers autres membres ...
  private int nbPassagers;
  private static int nbTotalPassagers;

  private void ajouterUnPassager() {
    if (placeDispo()) {
      nbPassagers++;
      nbTotalPassagers++;
    } else {
      gererTropNombreux();
    }
  }

  public static int getNbTotalPassagers() {
    return nbTotalPassagers;
  }

  public static void resetNbTotalPassagers(){
    nbTotalPassagers = 0;
  }
}
```

```
Vol.resetNbTotalPassagers();
System.out.println(
 Vol.getNbTotalPassagers());// 0

Vol v1 = new Vol();
v1.ajouterUnPassager();
v1.ajouterUnPassager();

Vol v2 = new Vol();
v2.ajouterUnPassager();
```

```
public class Vol {
  // ... divers autres membres ...
  private int nbPassagers;
  private static int nbTotalPassagers;

  private void ajouterUnPassager() {
    if (placeDispo()) {
      nbPassagers++;
      nbTotalPassagers++;
    } else {
      gererTropNombreux();
    }
  }

  public static int getNbTotalPassagers() {
    return nbTotalPassagers;
  }

  public static void resetNbTotalPassagers(){
    nbTotalPassagers = 0;
  }
}
```

```
Vol.resetNbTotalPassagers();
System.out.println(
 Vol.getNbTotalPassagers());// 0

Vol v1 = new Vol();
v1.ajouterUnPassager();
v1.ajouterUnPassager();

Vol v2 = new Vol();
v2.ajouterUnPassager();

System.out.println(
      v1.getNbPassagers()); // 2

System.out.println(
      v2.getNbPassagers()); // 1
```

```
public class Vol {
  // ... divers autres membres ...
  private int nbPassagers;
  private static int nbTotalPassagers;

  private void ajouterUnPassager() {
    if (placeDispo()) {
      nbPassagers++;
      nbTotalPassagers++;
    } else {
      gererTropNombreux();
    }
  }

  public static int getNbTotalPassagers() {
    return nbTotalPassagers;
  }

  public static void resetNbTotalPassagers(){
    nbTotalPassagers = 0;
  }
}
```

```
Vol.resetNbTotalPassagers();
System.out.println(
 Vol.getNbTotalPassagers());// 0

Vol v1 = new Vol();
v1.ajouterUnPassager();
v1.ajouterUnPassager();

Vol v2 = new Vol();
v2.ajouterUnPassager();

System.out.println(
      v1.getNbPassagers()); // 2

System.out.println(
      v2.getNbPassagers()); // 1

System.out.println(
 Vol.getNbTotalPassagers());// 3
```

```
public class Vol {
  // ... divers autres membres ...
  private int nbPassagers;
  private static int nbTotalPassagers;

  private void ajouterUnPassager() {
    if (placeDispo()) {
      nbPassagers++;
      nbTotalPassagers++;
    } else {
      gererTropNombreux();
    }
  }

  public static int getNbTotalPassagers() {
    return nbTotalPassagers;
  }

  public static void resetNbTotalPassagers(){
    nbTotalPassagers = 0;
  }
}
```

Bloc d'initialisation statique

- Un **bloc d'initialisation statique** effectue une initialisation de **type** (de la **classe**, et non d'une instance spécifique) **une fois et une seule**
- Il est exécuté **automatiquement juste avant la première utilisation** du type
- Rappel : un bloc d'initialisation *d'instance* est exécuté une fois pour *chaque* création d'instance (juste avant le constructeur pour instancier l'objet)
- Comme pour un bloc d'initialisation d'instance, bloc entre accolades en dehors de toute méthode ou constructeur, mais précédé du mot-clé `static`
- Ne peut accéder qu'aux membres statiques
- **Doit récupérer toutes les exceptions *checked*** éventuelles

personnel_disponible.txt

```
Pilote, Franck
Pilote, Noémie
Copilote, Zoé
Copilote, Jeoffrey
Hotesse, Lucille
Hotesse, Cathy
Hotesse, Sylvia
Hotesse, Naïma
OfficierMecanicien, Romain
OfficierMecanicien, Samantha
...
```

- La liste du personnel disponible est conservée dans un fichier texte
- Format :
  - une ligne par personne
  - **`Role, Nom`**

`personnel_disponible.txt`

```
Pilote, Franck
Pilote, Noémie
Copilote, Zoé
Copilote, Jeoffrey
Hotesse, Lucille
Hotesse, Cathy
Hotesse, Sylvia
Hotesse, Naïma
OfficierMecanicien, Romain
OfficierMecanicien, Samantha
...
```

- La liste du personnel disponible est conservée dans un fichier texte
- Format :
  - une ligne par personne
  - **`Role, Nom`**
- Ce fichier va être lu pour alimenter un tableau *pool* statique de personnel disponible
- A chaque fois qu'on aura besoin d'un type de personnel précis, on appelera la méthode statique **`trouverDisponible`** qui retournera le première personne encore disponible qui correspond au rôle

`personnel_disponible.txt`

```
Pilote, Franck
Pilote, Noémie
Copilote, Zoé
Copilote, Jeoffrey
Hotesse, Lucille
Hotesse, Cathy
Hotesse, Sylvia
Hotesse, Naïma
OfficierMecanicien, Romain
OfficierMecanicien, Samantha
...
```

```
MembreDEquipage m = EquipageManager.trouverDisponible(RoleDEquipage.CoPilote);
```

- La liste du personnel disponible est conservée dans un fichier texte
- Format :
  - une ligne par personne
  - **`Role, Nom`**
- Ce fichier va être lu pour alimenter un tableau *pool* statique de personnel disponible
- A chaque fois qu'on aura besoin d'un type de personnel précis, on appelera la méthode statique **`trouverDisponible`** qui retournera le première personne encore disponible qui correspond au rôle

`personnel_disponible.txt`

```
Pilote, Franck
Pilote, Noémie
Copilote, Zoé
Copilote, Jeoffrey
Hotesse, Lucille
Hotesse, Cathy
Hotesse, Sylvia
Hotesse, Naïma
OfficierMecanicien, Romain
OfficierMecanicien, Samantha
...
```

```
MembreDEquipage m = EquipageManager.trouverDisponible(RoleDEquipage.CoPilote);
```

Le rôle d'équipage est défini dans une **énumération** :

- mot-clé **enum**
- liste finie de valeurs possibles séparées par des virgules
- ici chaque valeur correspond exactement à un rôle dans le fichier

```
public enum RoleDEquipage {
  Pilote,
  Copilote,
  Hotesse,
  OfficierMecanicien
}
```

personnel_disponible.txt

```
Pilote, Franck
Pilote, Noémie
Copilote, Zoé
Copilote, Jeoffrey
Hotesse, Lucille
Hotesse, Cathy
Hotesse, Sylvia
Hotesse, Naïma
OfficierMecanicien, Romain
OfficierMecanicien, Samantha
...
```

```
public class EquipageManager {
  private final static String NOM_FICHIER = “c:\\personnel_disponible.txt”;









}
```

```
public class EquipageManager {
  private final static String NOM_FICHIER = “c:\\personnel_disponible.txt”;

  private static MembreDEquipage[] pool;









}
```

```
public class EquipageManager {
  private final static String NOM_FICHIER = “c:\\personnel_disponible.txt”;

  private static MembreDEquipage[] pool;

  public static MembreDEquipage trouverDisponible(RoleDEquipage roleVoulu) {









  }


}
```

```
public class EquipageManager {
  private final static String NOM_FICHIER = "c:\\personnel_disponible.txt";

  private static MembreDEquipage[] pool;

  public static MembreDEquipage trouverDisponible(RoleDEquipage roleVoulu) {

    MembreDEquipage choisi = null;

    for (int i = 0; i < pool.length; i++) {
      if (pool[i] != null && pool[i].getRole() == roleVoulu) {
        choisi = pool[i];
        pool[i] = null;
        break;
      }
    }

    return choisi;
  }


}
```

```
public class EquipageManager {
  // --- suite de la classe ---









}
```

```
public class EquipageManager {
  // --- suite de la classe ---

  // bloc d'initialisation static
  static {









  }
}
```

```
public class EquipageManager {
  // --- suite de la classe ---

  // bloc d'initialisation static
  static {
    BufferedReader reader = null;

      reader = new BufferedReader(new FileReader(NOM_FICHIER));
      String ligne = null;
      int idx = 0;
      pool = new MembreDEquipage[100];
      while ((ligne = reader.readLine()) != null) {









      }


  }
}
```

```
public class EquipageManager {
  // --- suite de la classe ---

  // bloc d'initialisation static
  static {
    BufferedReader reader = null;

      reader = new BufferedReader(new FileReader(NOM_FICHIER));
      String ligne = null;
      int idx = 0;
      pool = new MembreDEquipage[100];
      while ((ligne = reader.readLine()) != null) {
        String[] parties = ligne.split(",");
        String nom = parties[1];
        RoleDEquipage role = RoleDEquipage.valueOf(parties[0]);
        pool[idx] = new MembreDEquipage(nom, role);
        idx++;
      }


  }
}
```

```
public class EquipageManager {
  // --- suite de la classe ---

  // bloc d'initialisation static
  static {
    BufferedReader reader = null;
    try {
      reader = new BufferedReader(new FileReader(NOM_FICHIER));
      String ligne = null;
      int idx = 0;
      pool = new MembreDEquipage[100];
      while ((ligne = reader.readLine()) != null) {
        String[] parties = ligne.split(",");
        String nom = parties[1];
        RoleDEquipage role = RoleDEquipage.valueOf(parties[0]);
        pool[idx] = new MembreDEquipage(nom, role);
        idx++;
      }
    }
    catch (IOException e) { ... }
  }
}
```

```
public class EquipageManager {
  private final static String NOM_FICHIER = "c:\\personnel_disponible.txt";
  private static MembreDEquipage[] pool;
  public static MembreDEquipage trouverDisponible(RoleDEquipage roleVoulu) {...}
  static { ... }
}
```

```
MembreDEquipage pilote =
                    EquipageManager.trouverDisponible(RoleDEquipage.Pilote);

MembreDEquipage co =
                    EquipageManager.trouverDisponible(RoleDEquipage.Copilote);

MembreDEquipage hotesse1 =
                    EquipageManager.trouverDisponible(RoleDEquipage.Hotesse);

MembreDEquipage officierMecanicien =
          EquipageManager.trouverDisponible(RoleDEquipage.OfficierMecanicien);

MembreDEquipage hotesse2 =
                    EquipageManager.trouverDisponible(RoleDEquipage.Hotesse);
```

Résumé

- Les membres statiques (champs et méthodes) sont **partagés** entre toutes les instances de la classe
- Et ne sont donc pas associés à une instance particulière de la classe
- Les blocs d'initialisation statiques permettent l'**initialisation automatique de la classe** (et non d'une instance) avant la toute première utilisation de la classe
- Les exceptions *checked* dans un bloc d'initialisation statique doivent être récupérées tout de suite (pas d'appelant « réel »)