JAVA – INTERFACE

Wijin

v1.0 2024

Plan

- Présentation de la notion d'interface
- Apports dans les interfaces
- Interfaces et injection de dépendance

Java – Présentation de la notion d'interface

Java – Interface ?

- **Notion importante** :
  - Une **interface** peut représenter un **point d'échange** entre un fournisseur et un ou plusieurs clients (**contrat**)
  - Exemple : Si un composant fourni des fonctionnalités, des clients doivent pouvoir accéder à ces fonctionnalités, via un point d'accès → **interface**
    - On doit fournir au(x) client(s) l'interface lui permettant de mettre au point son code source
    - Uniquement les **définitions** des fonctions présentes, le client n'a pas besoin du code

Java – Interface

- **Autre exemple avec métaphore :**

  - Plusieurs fabricants décident de proposer une façade de machine à laver standard
  - L'objectif est de retrouver les mêmes fonctionnalités au même endroit sur n'importe quelle machine à laver
  - Les fabricants définissent **l'interface** de la façade, qu'il peuvent donner à un sous-traitant pour la fabrication (plan, modèle)
  - Le sous-traitant va savoir quoi faire pour construire les façades, **sans connaître les fonctionnalités propres à chaque fabricant** (démarrage, arrêt, programmes, ouverture, …)

Java – Interface

- **Autre exemple avec métaphore :**
  - Chaque fabricant pourra utiliser la façade pour construire ses modèles de machine à laver
  - La façade est **l'interface** (contrat)
  - La machine à laver propre à un fabricant est une **implémentation**

Java – Interface

- En java, mot clé **interface**
  - La classe qui implémentera cette interface devra utiliser la syntaxe suivante :
    - ***class*** *MaClasse* ***implements*** *MonInterface*
- Une classe qui implémente une interface doit implémenter **toutes les méthodes** de l'interface
  - Sinon erreur de compilation
  - Ou alors il est possible de déclarer la classe abstraite

Java – Interface

```java
package interfaces;

public interface IMachineALaver {

    enum Reglage {
        PRELAVAGE,
        LAVAGE,
        SECHAGE;
    }

    public void setReglage(Reglage reglage);

    public Reglage getReglage();

    public Boolean stopStart();

    public Boolean fermerHublot();

    public Boolean ouvrirHublot();
}
```

Ici, déclaration d'une **énumération** pour avoir une constante (il est **interdit** de déclarer des variables dans une interface)

Déclaration des méthodes, sans code associé

Java – Interface

```java
package interfaces;

public class MaMachineALaver implements IMachineALaver {

    private Boolean fonctionnement = false;
    private Reglage etat = Reglage.PRELAVAGE;
    private Boolean hublot = false;

    public void setReglage(Reglage reglage) {
        this.etat = reglage;
    }

    public Reglage getReglage() {
        return this.etat;
    }

    public Boolean stopStart() {
        this.fonctionnement = !this.fonctionnement;
        return this.fonctionnement;
    }

    public Boolean fermerHublot() {
        this.hublot = false;
        return this.hublot;
    }

    public Boolean ouvrirHublot() {
        this.hublot = !this.fonctionnement;
        return this.hublot;
    }

}
```

Mot clé **implements**

Un exemple d'implémentation

Implémentation des méthodes de l'interface

Java – Interface

- En java, il est possible d'implémenter **plusieurs interfaces** dans une même classe
  - **Utile**, car Java ne supporte pas l'héritage multiple
  - On peut faire que la classe hérite d'une autre classe **et** implémente une interface (ou plusieurs)

Java – Interface

```java
package interfaces;

public interface IWrite {
    public Boolean write (String filename, String content);
}
```

Première interface

```java
package interfaces;

public interface IRead {
    public String read(String filename);
}
```

Deuxième interface

Classe qui implémente les 2 interfaces

```java
package interfaces;

public class File implements IWrite, IRead {

    public Boolean write(String filename, String content) {
        System.out.println("file " + filename + " avec le contenu " + content + " a été écrit");
        return true;
    }

    public String read(String filename) {
        return "file " + filename + " a été lu";
    }

}
```

Java – Interface

Classe

```java
package interfaces;

public class Output {

    public Boolean write(String filename, String content) {
        System.out.println("file " + filename + " avec le contenu " + content + " a été écrit");
        return true;
    }
}
```

Interface

```java
package interfaces;

public interface IRead {
    public String read(String filename);
}
```

Classe qui hérite de **Output** et qui implémente l'interface **IRead**

```java
package interfaces;

public class File2 extends Output implements IRead {

    public String read(String filename) {
        return "file " + filename + " a été lu";
    }
}
```

Java – Interface

- **Avantages** supplémentaire des interfaces
  - **Ajouter une ou plusieurs fonctionnalités à une classe**
  - Positionner **l'implémentation souhaitée** pour chaque interface pour l'exécution

Java – Apports dans les interfaces

Java – Apports dans les interfaces

- Il est possible depuis la version 8 de Java d'ajouter des méthodes **statiques** dans des interfaces

  - **Particularité** : ces méthodes statiques au sein d'interface peuvent contenir une **implémentation**

Java – Apports dans les interfaces

```java
package interfaces;

public interface IFile {

    enum FileInfo {
        SEQUENTIAL,
        RELATIVE
    }

    static FileInfo getInfo(Boolean type) {
        return type?FileInfo.RELATIVE:FileInfo.SEQUENTIAL;
    }

    public FileInfo read(Boolean type);
}
```

Méthode **static** avec implémentation (code)

Java – Apports dans les interfaces

- Il est possible depuis la version 8 de Java d'ajouter des méthodes « **default** » dans des interfaces

  - **Particularité** : une méthode **default** est une méthode avec une **implémentation**, qu'il va être possible d'utiliser **en fonction d'un choix**, dans le cadre d'une implémentation multiple

    - Si 2 méthodes de même nom dans 2 interfaces différentes, il n'est pas possible pour une classe d'implémenter 2 fois la même méthode

Java – Apports dans les interfaces

```java
package interfaces;

public interface IRadio {

    public void setVolume(Integer volume);

    default void start() {
        System.out.println("Sonne comme une radio");
    }
}

```

Méthode **default** nommée *start()*

```java
package interfaces;

public interface IReveil {

    public void setHeure(String heure);

    default void start() {
        System.out.println("Sonne comme un réveil");
    }
}
```

Méthode **default** nommée *start()*

Java – Apports dans les interfaces

```java
package interfaces;

public class RadioReveil implements IRadio, IReveil {

    private Boolean typeReveil;
    private String heure;
    private Integer volume;

    public RadioReveil(Boolean typeReveil) {
        this.typeReveil = typeReveil;
    }

    @Override
    public void start() {
        if (typeReveil.booleanValue()) {
            IReveil.super.start();
        } else {
            IRadio.super.start();
        }
    }

    @Override
    public void setHeure(String heure) {
        this.heure = heure;
    }

    @Override
    public void setVolume(Integer volume) {
        this.volume = volume;
    }

}
```

La classe implémente les 2 interfaces

La méthode ***start()*** implémentée ici va choisir en fonction de la valeur de « **typeReveil** » quelle méthode **default** exécuter :

Utilisation de super sur l'interface pour appeler la méthode default voulue

Java – Interfaces et injection de dépendance

Java – Interfaces et injection de dépendance

- La notion d'interface peut permettre de rendre les composants **indépendants** les uns des autres
  - **Objectifs** : faciliter la **maintenance** et l'**évolution** des applications
  - Mécanisme d'**injection de dépendance** : différer la dépendance d'un composant à un autre à l'exécution

Java – Interfaces et injection de dépendance

```java
package interfaces;

public class Traducteur {

    public String traduit(String mot) {
        switch(mot) {
        case "maison":
            return "house";
        case "homme":
            return "man";
        default:
            return "unknown";
        }
    }
}
```

1er exemple avec **dépendance forte**

```java
package interfaces;

public class TestTraducteur {

    Traducteur t = new Traducteur();

    public  String testTraduction() {
        return t.traduit("maison");
    }

}
```

Couplage fort entre la classe **TestTraducteur** et la classe **Traducteur**

```java
package interfaces;

public class ProgrammeTraducteur {

    public static void main(String[] args) {
        TestTraducteur tt = new TestTraducteur();
        System.out.println(tt.testTraduction());
    }
}
```

Si le service de traduction n'est plus disponible, l'application ne fonctionne plus !

Java – Interfaces et injection de dépendance

```java
package interfaces;

public interface ITraducteur {

    public String traduit(String mot);
}
```

2ème exemple avec **dépendance faible**

1ère étape : création d'une **interface** qui **expose** la méthode de traduction **(*traduit()*).**

```java
package interfaces;

public class Traducteur implements ITraducteur {

    public String traduit(String mot) {
        switch(mot) {
        case "maison":
            return "house";
        case "homme":
            return "man";
        default:
            return "unknown";
        }
    }
}
```

2ème étape : création d'une **implémentation de l'interface**

Java – Interfaces et injection de dépendance

```java
package interfaces;

public class TestTraducteur {

    ITraducteur it;

    public TestTraducteur(ITraducteur it) {
        this.it = it;
    }

    public String testTraduction() {
        return it.traduit("maison");
    }

}
```

2ème exemple avec **dépendance faible**

Utilisation du type de l'**interface** et passage de l'objet, via son **interface**, au constructeur

Java – Interfaces et injection de dépendance

2ème exemple avec **dépendance faible**

```java
package interfaces;

public class ProgrammeTraducteur {

    public static void main(String[] args) {

        TestTraducteur tt = new TestTraducteur(new Traducteur());
        System.out.println(tt.testTraduction());

    }
}
```

Dans le **main**, on positionne l'**implémentation** souhaitée

Java - Quiz

Java – Quiz

- A partir de Java 8, les interfaces peuvent contenir des méthodes avec du code, en utilisant le mot clé **default** ?

  - **Vrai**
  - **Faux**

Java – Quiz

- Peut-on hériter d'une classe et implémenter plusieurs interfaces ?
  - Oui
  - Non

Java – Quiz

- Peut-on hériter d'une interface ?
  - Oui
  - Non

Java – Quiz

- Une classe qui implémente une interface est-elle toujours instanciable ?

  - Non, si elle n'implémente pas toutes les méthodes de l'interface

  - Oui, si elle implémente toutes les méthodes de l'interface

  - Oui, même si elle n'implémente pas toutes les méthodes de l'interface, mais il faudra déclarer la classe abstract

Java – Quiz

- Une classe abstraite dont toutes les méthodes seraient abstraites s'apparente à une interface

  - Vrai

  - Faux

MERCI POUR VOTRE ATTENTION

Faites-moi part de vos remarques concernant le cours afin qu'il soit amélioré pour les prochaines sessions : nicolas.sanou@wijin.tech