

Factory, service object, configuration

Leleux Laurent

2022 - 2023

Instanciación

- Création d'instances
- « new »
- Pas d'encapsulation
- Pas de polymorphisme

Polymorphisme

« Le polymorphisme est l'idée de permettre à un même code d'être utilisé avec différents types, ce qui permet des implémentations plus abstraites et générales. »

Polymorphisme

« En proposant d'utiliser un même nom de méthode pour plusieurs types d'objets différents, le polymorphisme permet une programmation beaucoup plus générique. Le développeur n'a pas à savoir, lorsqu'il programme une méthode, le type précis de l'objet sur lequel la méthode va s'appliquer. Il lui suffit de savoir que cet objet implémentera la méthode. »

Encapsulation

« L'encapsulation est l'idée de protéger l'information contenue dans un objet et de ne proposer que des méthodes de manipulation de cet objet. Ainsi, les propriétés contenues dans l'objet seront assurées/validées par les méthodes de l'objet et ne seront plus de la responsabilité de l'utilisateur extérieur. »

Factory

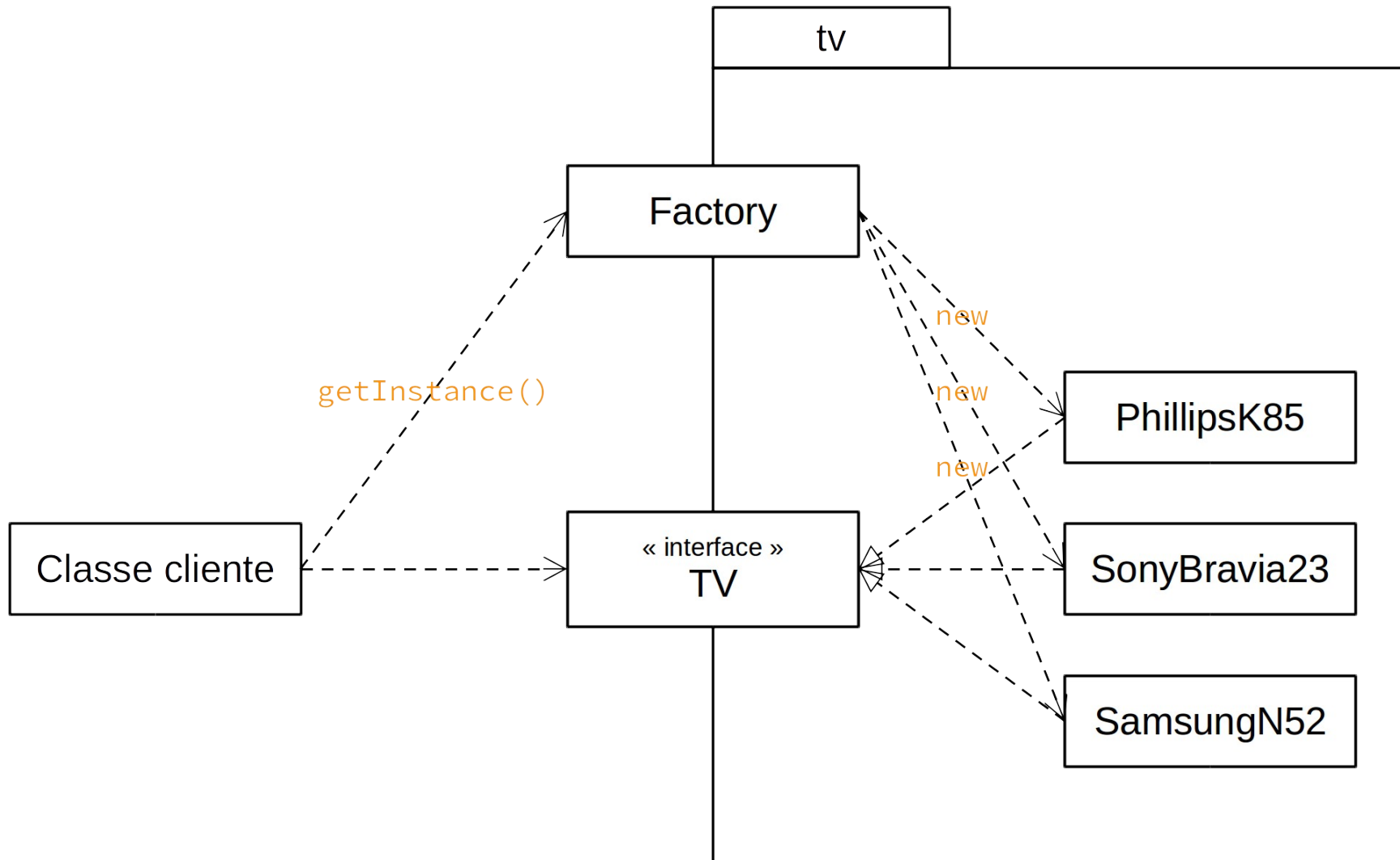
- Encapsuler l'instanciation
- Masquer le « new »
- Ajouter une stratégie d'instanciation
- Dépendances abstraites

Méthodes

- `TvFactory.getInstance()`
- `DvdFactory.getInstance(caracteristiques)`
- `ReveilFactory.getInstance(type)`
- `...getInstance(type, caracteristiques)`

- `/!\ Type de retour`

Exemple



Bénéfices

- Le client
 - ne connaît aucune classe
 - connaît que des abstractions
 - est indépendant des implémentations
 - utilise le « dynamic binding »
- Stratégies

A retenir !

« Soyez le plus possible insensible aux changements qui se produisent dans les autres packages ! »

« Déléguez la logique d'instanciation des objets éloignés aux factories des packages éloignés »

« Utilisez les interfaces proposés par les packages éloignés, qui exposent les services rendus par les objets éloignés »

Problème

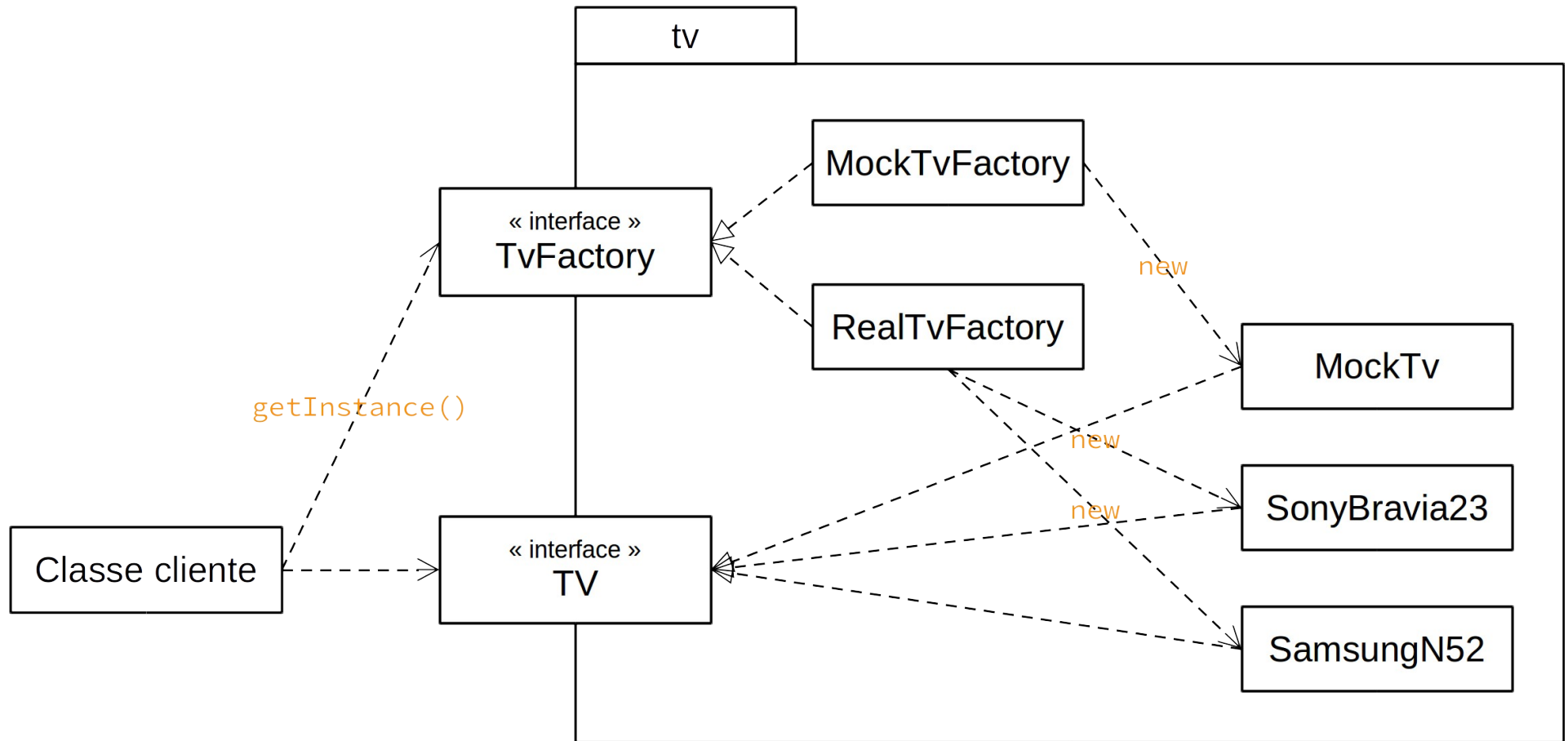
`getInstance()` n'est pas polymorphe

- Toujours le même code
- Pas de changement possible
- Toujours la même factory

Service Object

- Classe / Objet utilitaire
- Plus de static !
- Objet caché par une interface
- Liaison à l'exécution
- Stateless

Multiple implementations



Contexte

- Développement
- Production
- Changement dynamique
- Au lancement ?

Properties

- Système clef/valeur
- Map<String, String>

```
#Fri Jan 17 22:37:45 MYT 2014  
dbpassword=password  
database=localhost  
dbuser=gerard
```

Example

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.Properties;

public class Main {

    public static void main(String[] args) {
        // Create and load properties
        Properties props = new Properties();
        FileInputStream input = null;
        try {
            input = new FileInputStream("src/props.properties");
            props.load(input);
        } catch (FileNotFoundException e) {
            e.printStackTrace();
            System.exit(1);
        } catch (IOException e) {
            e.printStackTrace();
            System.exit(2);
        } finally {
            if (input != null) {
                try {
                    input.close();
                } catch (IOException ignore) {
                }
            }
        }
        // Print properties
        System.out.println(props.getProperty("dbpassword"));
    }
}
```


Documentation

- Recherche dans la JAVADOC...