UNIVERSITÉ CÔTE D'AZUR

UTILISATION DU PATTERN SINGLETON

@.fR Frédéric RALLO

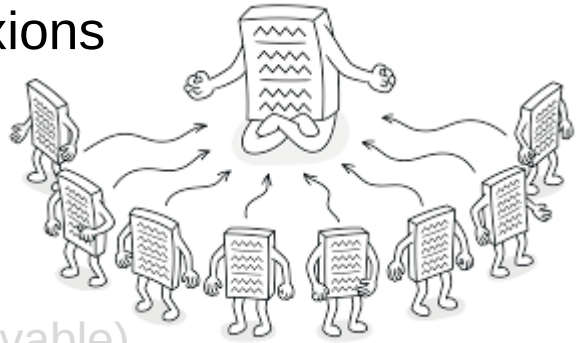
CLASSIFICATION DES PATRONS DE CONCEPTION

Création

- Comment un objet peut être créé
- Indépendance entre la manière de créer et la manière d'utiliser

Structure

- Comment les objets peuvent être combinés
- Indépendance entre les objets et les connexions



Comportement

- Comment les objets communiquent
- Encapsulation de processus (ex : observer/observable)

LA PROBLÉMATIQUE QU'IL CHERCHE À RÉSOUDRE

Structure

- Comment les objets peuvent être combinés
- Indépendance entre les objets et les connexions

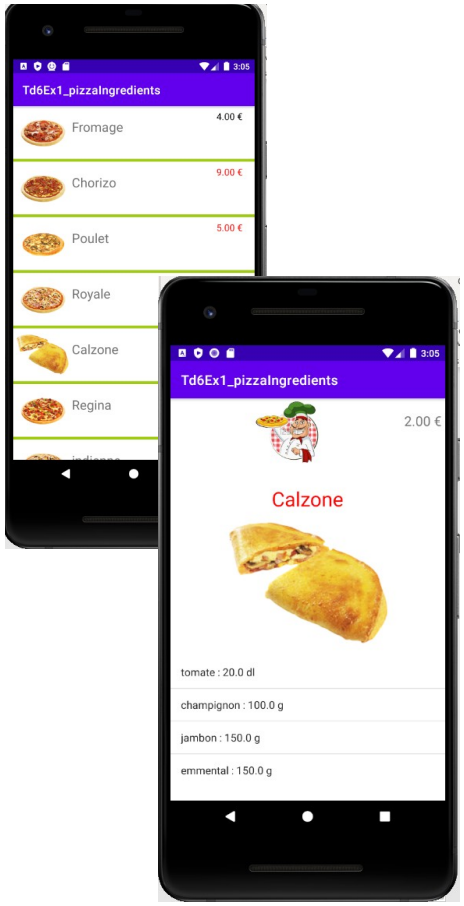
Il est utilisé lorsqu'on a besoin exactement d'un objet pour coordonner des opérations dans un système.
Le modèle est parfois utilisé pour son efficacité, lorsque le système est plus rapide ou occupe moins de mémoire avec peu d'objets qu'avec beaucoup d'objets similaires. .

WIKIPÉDIA
L'encyclopédie libre

ETUDE DE LA PROBLÉMATIQUE SUR ANDROID

Cadre d'utilisation sur Android

- Echanger des objets entre 2 Activités
- *Mais on peut aussi utiliser l'interface Parcelable*
- exemple :
  - Depuis Activity1 on sélectionne 1 Pizza
  - Cette Pizza est issue d'une liste de Pizza
  - On envoie la Pizza sélectionnée à Activity2

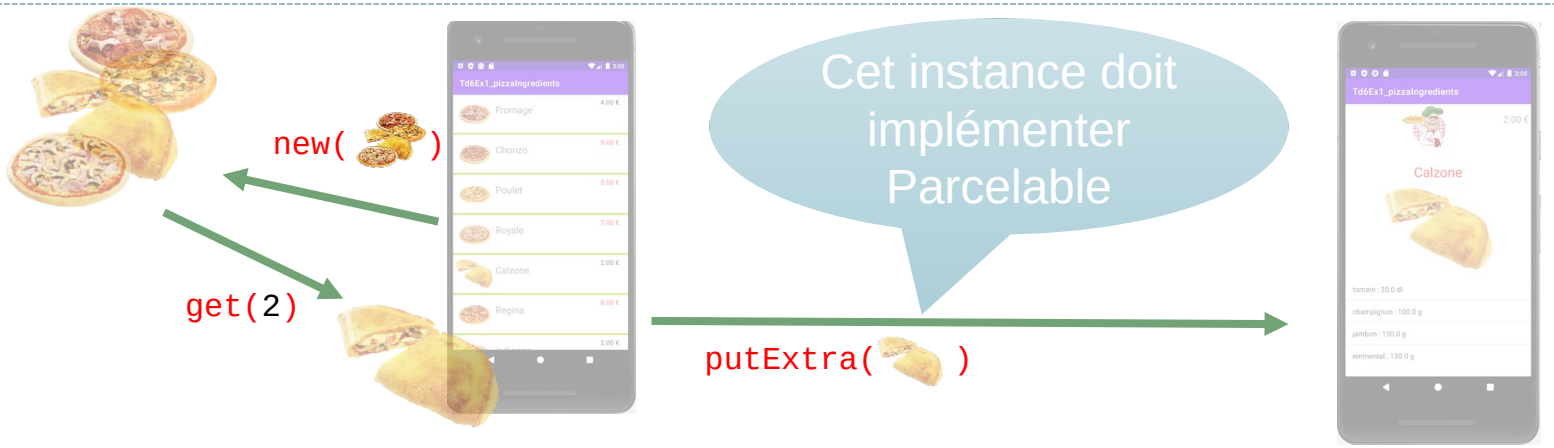


Cadre d'utilisation sur Android

- Echanger des objets entre 2 Activités

Idée Avec un Parcelable

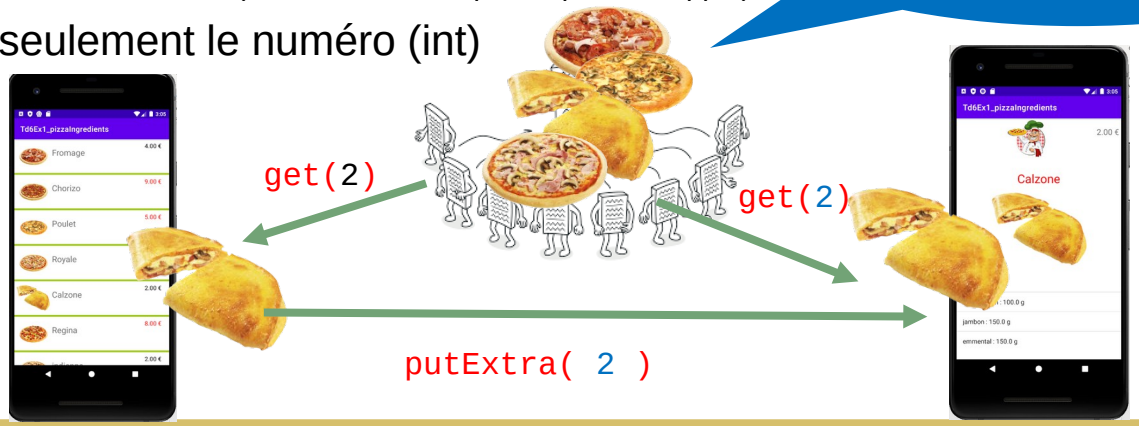
- Activité1 crée une liste de Pizza et récupère l'une des Pizza
- Puisque la Pizza est Parcelable, on peut la passer à Activité2



Idée avec un Singleton !

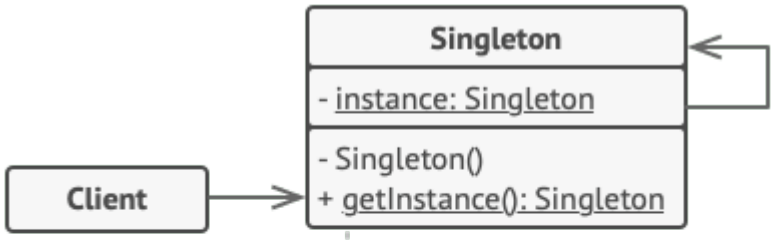
- Les 2 Activités connaissent la liste unique de Pizzas
- Personne ne crée la liste (ce sera automatique au premier appel)
- On échange seulement le numéro (int)

Cette liste est un Singleton !
Accessible depuis les 2 activités existe une seule instance



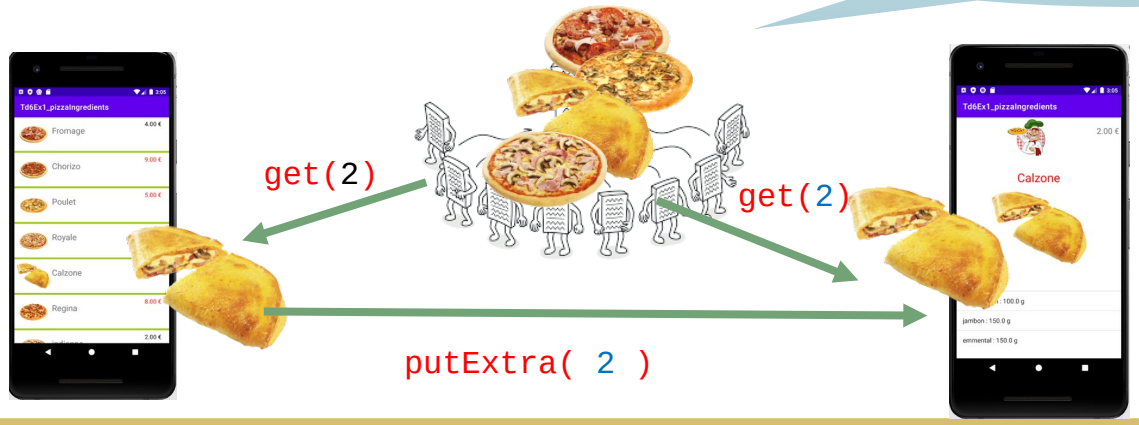
ECRIRE UN SINGLETON EN JAVA

Solution 1 !
rendre le constructeur privé
créer un attribut (privé) instance
créer un accesseur getInstance()




```
if (instance == null) {
    // Note: if you're creating an app with
    // multithreading support, you should
    // place a thread lock here.
    instance = new Singleton()
}
return instance
```

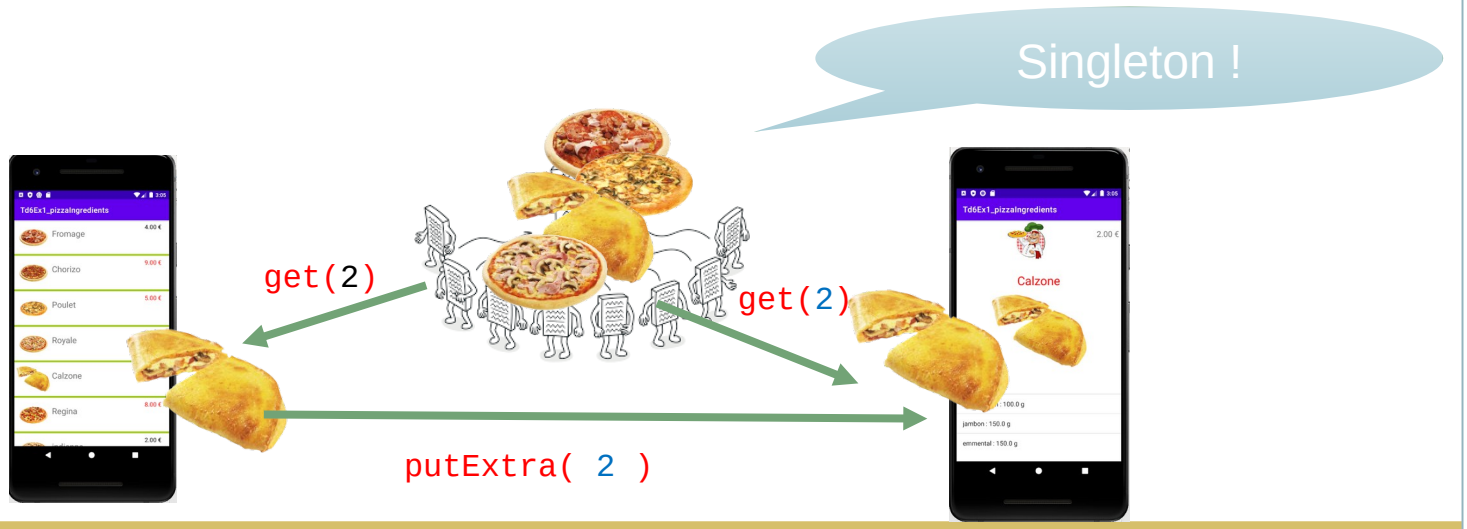
Singleton !



ECRIRE UN SINGLETON EN JAVA

Solution 2 (non threadSafe) !

Créer une classe qui ne contient QUE des méthodes statiques

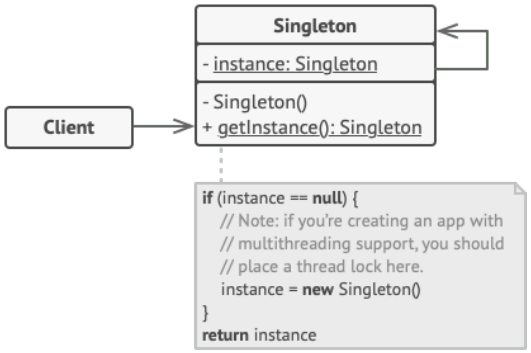


ECRIRE UN SINGLETON EN JAVA

- **Interdire de créer des instances**
  - Le constructeur est privé
- **Permettre d'accéder à l'unique instance**
  - Attribut privé instance
  - Accesseur (getter)

Accesseur getInstance() static !

- **Interdire de modifier l'instance**
  - Sécurité !
  - On peut déclarer instance en final (mais ce n'est pas la seule façon de faire)




```
public class Singleton {
    private String objectState;
    private static final Singleton instance = new Singleton();
    private Singleton() {
        this.objectState = "ABCD...XYZ";
        // Setting some random object objectState
    }
    public static Singleton getInstance() {
        return instance;
    }
    public String getObjectState() {
        return objectState;
    }
    public void setObjectState(String objectState) {
        this.objectState = objectState;
    }
}
```

PLUSIEURS VARIANTES DE RÉALISATION (PLUS OU MOINS COMPLETES)

« final » devant l'attribut assure l'unicité !

```
public class Singleton {
    private String objectState;
    private static final Singleton instance = new Singleton();
    private Singleton() {
        this.objectState = "ABCD...XYZ";
        // Setting some random object objectState
    }
    public static Singleton getInstance() {
        return instance;
    }
    public String getObjectState() {
        return objectState;
    }
    public void setObjectState(String objectState) {
        this.objectState = objectState;
    }
}
```

La classe est « final » et toutes les méthodes sont statiques…

```
public final class Singleton {
    private static String login;
    private static String passwd;

    public static String getLogin() { return login; }
    public static void setLogin(String login) { Singleton.login = login; }
    public static String getPasswd() { return passwd; }
    public static void setPasswd(String passwd) { Singleton.passwd = passwd; }
}
```

```
public final class Singleton {
    private static volatile Singleton instance = null;

    private Singleton() { super();  }

    public final static Singleton getInstance() {
        if (Singleton.instance == null) {
            synchronized(Singleton.class) {
                if (Singleton.instance == null) { Singleton.instance = new
Singleton();  }
            }
        }
        return Singleton.instance;
    }
}
```

Ce singleton est threadSafe !