ENCART CAMÉRA

Patrons de conception
Créationnels

Sébastien Mosser - INF5153
Chapitre 7 - Capsule 1
Automne 2020

ace

ENCART CAMÉRA

Prototype

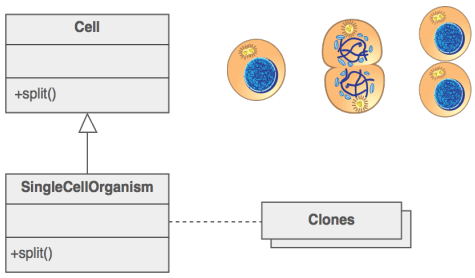
ENCART CAMÉRA

Problème

Comment **utiliser un objet comme exemple d'instantiation** pour les autres, par exemple quand sa **création est coûteuse** mais sa **copie aisée** ?

Exemple

ENCART CAMÉRA

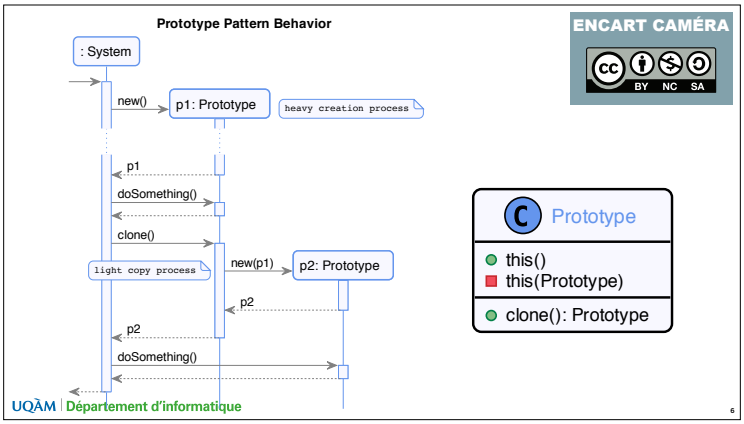


Intention

ENCART CAMÉRA

- Spécifier une **classe d'objets** constructibles par **clonage**
- Permettre de **se passer du new**
- **Co-opter une instance** comme étant **représentative** des suivantes

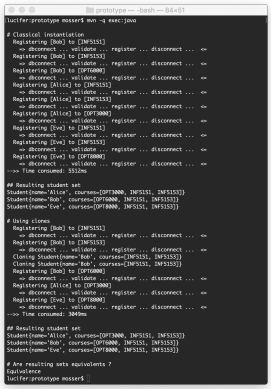
ENCART CAMÉRA



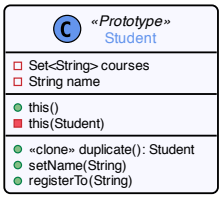
Utilisation dans le code


```java
private static Set<Student> usingConstructor(){
  Set<Student> result = new HashSet<>();

  Student bob = new Student("Bob");
  bob.registerTo("INF5151");
  bob.registerTo("INF5153");
  bob.registerTo("OPT6000");
  result.add(bob);

  Student alice = new Student("Alice");
  alice.registerTo("INF5151");
  alice.registerTo("INF5153");
  alice.registerTo("OPT3000");
  result.add(alice);

  Student eve = new Student("Eve");
  eve.registerTo("INF5151");
  eve.registerTo("INF5153");
  eve.registerTo("OPT8000");
  result.add(eve);

  return result;
}
```

```java
private static Set<Student> usingClones(){
  Set<Student> result = new HashSet<>();

  Student bob = new Student("Bob");
  bob.registerTo("INF5151");
  bob.registerTo("INF5153");

  Student alice = bob.duplicate();
  alice.setName("Alice");

  Student eve = bob.duplicate();
  eve.setName("Eve");

  bob.registerTo("OPT6000");
  result.add(bob);

  alice.registerTo("OPT3000");
  result.add(alice);

  eve.registerTo("OPT8000");
  result.add(eve);

  return result;
}
```

```
lucifer:prototype mosser$ mvn -q exec:java

# Classical instantiation
  Registering [Bob] to [INF5151]
    => dbconnect ... validate ... register ... disconnect ... <=
  Registering [Bob] to [INF5153]
    => dbconnect ... validate ... register ... disconnect ... <=
  Registering [Bob] to [OPT6000]
    => dbconnect ... validate ... register ... disconnect ... <=
  Registering [Alice] to [INF5151]
    => dbconnect ... validate ... register ... disconnect ... <=
  Registering [Alice] to [INF5153]
    => dbconnect ... validate ... register ... disconnect ... <=
  Registering [Alice] to [OPT3000]
    => dbconnect ... validate ... register ... disconnect ... <=
  Registering [Eve] to [INF5151]
    => dbconnect ... validate ... register ... disconnect ... <=
  Registering [Eve] to [INF5153]
    => dbconnect ... validate ... register ... disconnect ... <=
  Registering [Eve] to [OPT8000]
    => dbconnect ... validate ... register ... disconnect ... <=
--=> Time consumed: 551ms

## Resulting student set
Student{name='Alice', courses=[OPT3000, INF5151, INF5153]}
Student{name='Bob', courses=[OPT6000, INF5151, INF5153]}
Student{name='Eve', courses=[OPT8000, INF5151, INF5153]}

# Using clones
  Registering [Bob] to [INF5151]
    => dbconnect ... validate ... register ... disconnect ... <=
  Registering [Bob] to [INF5153]
    => dbconnect ... validate ... register ... disconnect ... <=
  Cloning Student{name='Bob', courses=[INF5151, INF5153]}
  Cloning Student{name='Bob', courses=[INF5151, INF5153]}
  Registering [Bob] to [OPT6000]
    => dbconnect ... validate ... register ... disconnect ... <=
  Registering [Alice] to [OPT3000]
    => dbconnect ... validate ... register ... disconnect ... <=
  Registering [Eve] to [OPT8000]
    => dbconnect ... validate ... register ... disconnect ... <=
--=> Time consumed: 3044ms

## Resulting student set
Student{name='Alice', courses=[OPT3000, INF5151, INF5153]}
Student{name='Bob', courses=[OPT6000, INF5151, INF5153]}
Student{name='Eve', courses=[OPT8000, INF5151, INF5153]}

# Are resulting sets equivalents ?
Equivalence : ok
lucifer:prototype mosser$
```

Conséquences

- On peut **factoriser les opérations coûteuses** dans le prototype
- **Le clonage est une opération difficile**
  - Clonage superficiel ?
  - Clonage en profondeur ?
    - *Ça vous rappelle les problèmes de fuite de données du début ?*
- C'est un **modèle de programmation** auquel on est peu habitué
  - *C'est le modèle de construction des objets de Javascript par exemple*

Où le rencontrer dans la "vraie-vie"(c) ?

ENCART CAMÉRA

- Modèle objet de :
  - Javascript,
  - Maple,
  - Object Lisp,
  - R, ...
- Interface *Cloneable* de Java

Module java.base
Package java.lang

Interface Cloneable

All Known Subinterfaces:
`AclEntry`, `Attribute`, `AttributedCharacterIterator`, `Attributes`, `CertPathBuilderResult`, `CertPathParameters`, `CertPathValidatorResult`, `CertSelector`, `CertStoreParameters`, `CharacterIterator`, `CRLSelector`, `Descriptor`, `ExtendedGSSCredential`, `GSSCredential`, `Name`

`public interface Cloneable`

A class implements the `Cloneable` interface to indicate to the `Object.clone()` method that it is legal for that method to make a field-for-field copy of instances of that class.

Invoking Object's clone method on an instance that does not implement the `Cloneable` interface results in the exception `CloneNotSupportedException` being thrown.

By convention, classes that implement this interface should override `Object.clone` (which is protected) with a public method. See `Object.clone()` for details on overriding this method.

Note that this interface does not contain the clone method. Therefore, it is not possible to clone an object merely by virtue of the fact that it implements this interface. Even if the clone method is invoked reflectively, there is no guarantee that it will succeed.

Since:
1.0

See Also:
`CloneNotSupportedException`, `Object.clone()`

ENCART CAMÉRA

Builder

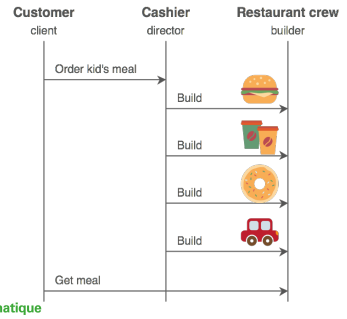
Problème

ENCART CAMÉRA

Comment **créer par assemblage** un **objet complexe**, **indépendamment des parties** qui le composent ?

Exemple

ENCART CAMÉRA

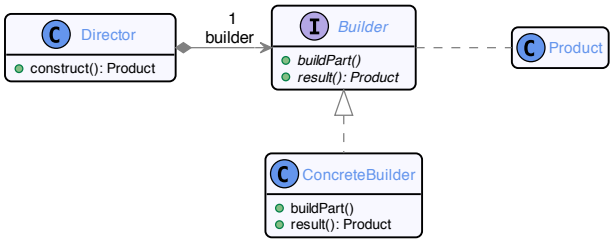


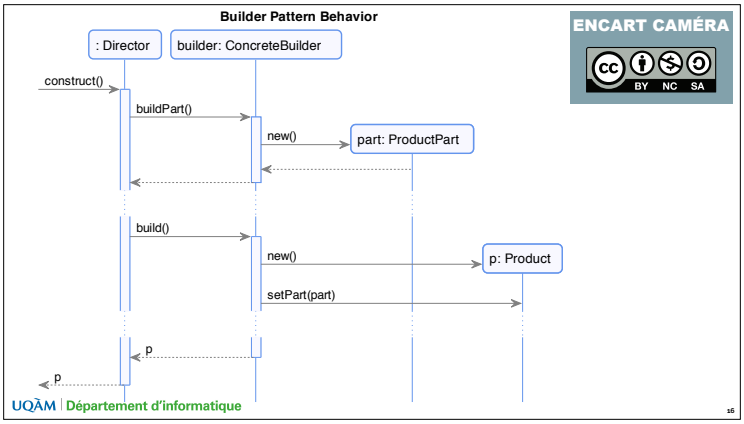
Intention

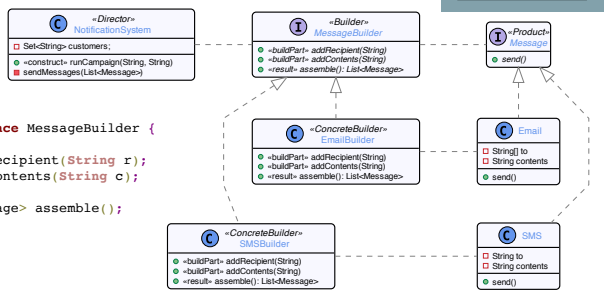
ENCART CAMÉRA

- **Isoler** la **construction** de l'objet de sa **représentation**
- **Contrôler** le procédé de **construction**
- **Changer au besoin** la représentation interne du produit

ENCART CAMÉRA








```
public interface MessageBuilder {

    void addRecipient(String r);
    void addContents(String c);

    List<Message> assemble();
}
```


Utilisation dans le code

```
public class SMSBuilder implements MessageBuilder {

    private Set<String> recipients = new HashSet<>();
    private String contents = null;

    @Override public void addRecipient(String r) { this.recipients.add(r); }

    @Override public void addContents(String c) { this.contents = c; }

    @Override
    public List<Message> assemble() {
        if (recipients.isEmpty() || contents == null)
            throw new IllegalStateException();
        return recipients.stream()
                .map((r) -> new SMS(r,contents))
                .collect(Collectors.toList());
    }
}
```

Conséquences

ENCART CAMÉRA

- **Définition abstraite** du processus de construction
  - **Réutilisation du processus**, mais **produits différents**
- On peut utiliser du **polymorphisme** pour construire des produits différents
- **Le client n'a pas besoin de connaitre le processus de fabrication**

Où le rencontrer dans la "vraie-vie"(c) ?

ENCART CAMÉRA

- Dès qu'on parle de "Fluent API"
  - Mockito
  - EntityBuilder (spring, .Net, …)
- Dans l'API Java (StringBuilder, StringBuffer, …)
- Interfaces graphiques (composition de composants)

ENCART CAMÉRA

Factory

Problème

ENCART CAMÉRA

- **Masquer la complexité** d'un cadriciel
  - Reposer uniquement sur les interfaces (abstractions)
- **Configurer des objets** potentiellement complexes
  - Ne pas faire dépendre le client de cette complexité
- **Impossible d'anticiper ce qu'il faut construire** a priori

Exemple

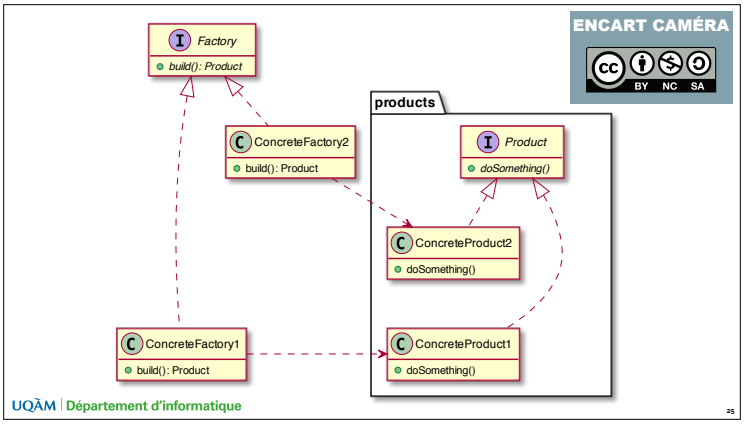
ENCART CAMÉRA

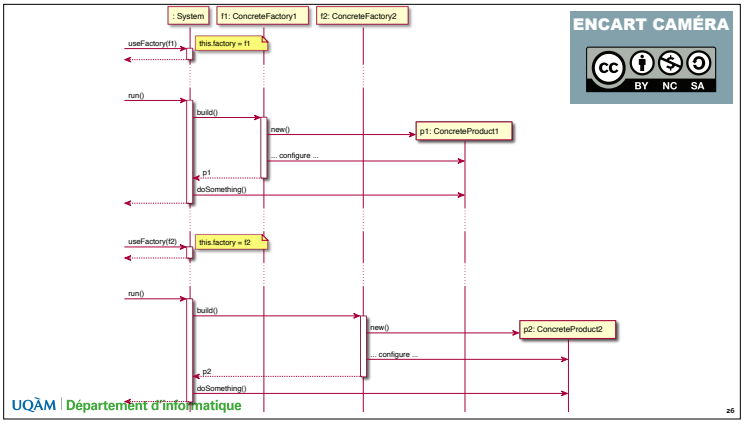
- Utilisation d'un système de journalisation (Log)
  - Le journal peut-être local ou distant
  - On peut journaliser en JSON, XML, TXT, …
- On peut utiliser Log4J, SLF4J, util.Logger, …
- On peut vouloir changer d'avis

Intention

ENCART CAMÉRA

- Définir une **interface pour la création** d'un objet
- Laisser aux **sous-classes le choix** de l'instantiation
- **Déléguer à une classe** "qui sait faire" les instantiations complexes





Utilisation dans le code

```
public interface Shape {
   void draw();
}

public class Rectangle implements Shape {
   @Override
   public void draw() { … }
}

public class Square implements Shape {
   @Override
   public void draw() { … }
}

public class Circle implements Shape {
   @Override
   public void draw() { … }
}
```

```
public class ShapeFactory {

   public Shape getShape(String shapeType){
      if(shapeType == null){
         return null;
      }
      if(shapeType.equalsIgnoreCase("CIRCLE")){
         return new Circle();

      } else if(shapeType.equalsIgnoreCase("RECTANGLE")){
         return new Rectangle();

      } else if(shapeType.equalsIgnoreCase("SQUARE")){
         return new Square();
      }

      return null;
   }
}
```

Exemple de Fabrique correcte

ENCART CAMÉRA

- **Une interface "buildX", avec X ce qu'on veut construire**
  - *buildRectangle, buildTriangle, …*
- **Des familles de produits différentes :**
  - *Des triangles en JavaFX, en Swing, en HTMLCanvas, …*
- **On choisit la fabrique et on l'utilise pour cette famille donnée.**
- **Les fabriques qui prennent des "strings" en paramètres et font un switch sont souvent de "mauvaises" fabriques.**

Conséquences

ENCART CAMÉRA

- **La fabrication n'est pas adhérente** aux classes spécifiques dans le code
- Lien avec le patron GRASP de "**Création**"
- **Gain en flexibilité** dans les interfaces des objets construits

Où le rencontrer dans la "vraie-vie"(c) ?

ENCART CAMÉRA

- Dans le JDK:
  - Les API utilisant getInstance() et valueOf()
    - (souvent attachée à des singletons)
- Dans les cadriciels réseaux
  - Construire la connexion au système distant
- Dans les approches de modélisation / génération de code
  - Une interface pour plusieurs implémentations possibles

ENCART CAMÉRA

Abstract Factory

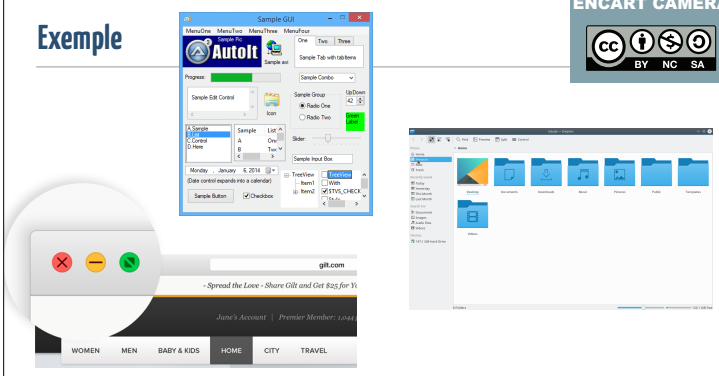
ENCART CAMÉRA

Problème

Comment **garantir la consistance** de **familles de systèmes**, qui sont à la fois **proches mais différentes** ?

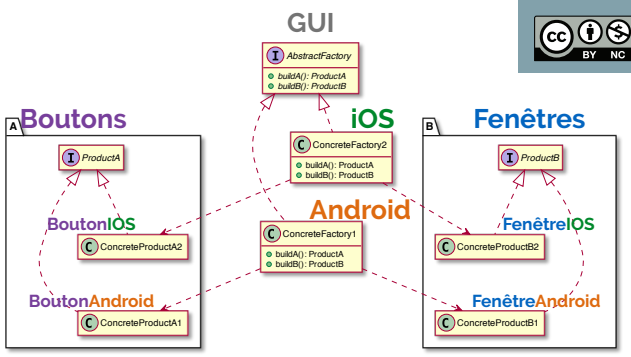
ENCART CAMÉRA

Exemple



Intention

- Fournir une **interface** pour créer des **familles d'objets**
- **Pas de couplage** avec les implémentations réelles
- **Configuration** aisée d'une **multitude de produits** cohérents



Conséquences

- **Isolation** des classes concrètes
- **Échange / Remplacement** facile de familles de produits
- Maintien de la **cohérence**
- **Difficile d'introduire de nouvelles familles de produits**
  - *Doit modifier toutes les fabriques …*

Où le rencontrer dans la "vraie-vie"(c) ?

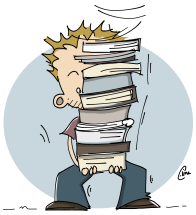
ENCART CAMÉRA

- Dans les cadriciels multi-platformes
- Générateurs de code visant plusieurs cibles
- API gérant des collections uniformes
  - Python (defaultdic), Java Collections (presque)

UQÀM | Département d'informatique

FACULTÉ DES SCIENCES
Université du Québec à Montréal

ENCART CAMÉRA



https://mosser.github.io/

ace

https://ace-design.github.io/

Abonne toi à la chaine,
et met un pouce bleu !