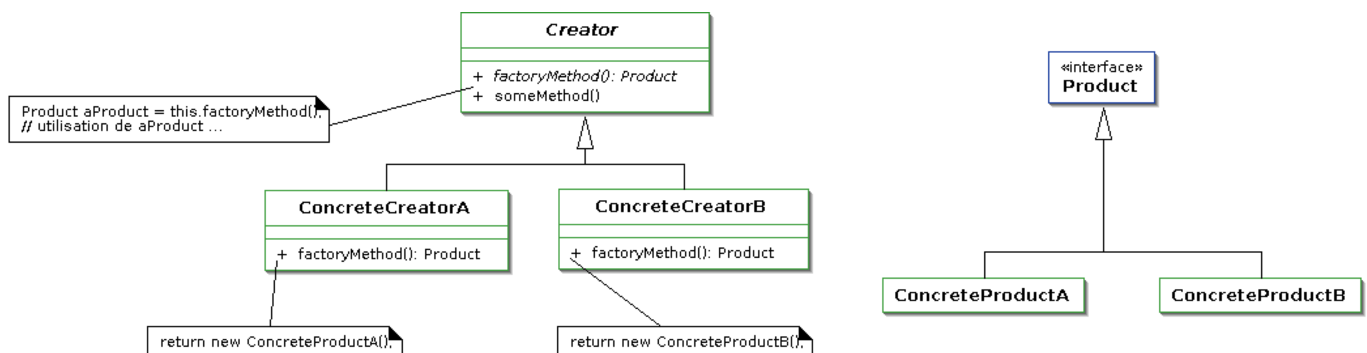
Design Pattern : factory method

Intent

Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses.

Structure



Eléments caractéristiques

- une méthode dont la fonction est de créer un objet, cet objet est renvoyé comme résultat.

  Cette méthode est surchargée dans les sous-classes pour modifier la classe de l'objet qu'elle crée.

Exemples rencontrés en TD

Dans l'api java

- la méthode `iterator()` de `java.util.Collection` crée et fournit un itérateur dont l'implémentation diffère selon la classe concrète de la collection considérée.

Tests : héritage de tests

Dans le cours, on indique que `YellowSqaure` doit passer avec succès les tests de `Square` dont elle hérite. C'est évidemment aussi le cas des autres classes qui héritent de `Square`. Elles doivent toutes en respecter le contrat : *principe de Liskov*. Les tests doivent donc pouvoir être exécutés (et validés) par les sous-types, mais avec leurs propres instances.

Une telle situation se résout généralement en mettant en place une factory method qui est spécialisée dans les sous-classes des tests et qui permet d'initialiser une référence utilisée pour les tests.

Dans l'exemple suivant, `createSquare` est cette *factory method*.

```
public abstract class AsbtractSquareTest {

  // la référence de type Square qui est utilisée pour les tests
  protected Square square;
  // la factory method
  public abstract Square createSquare();

  @Before
  public void initSquare() {
    // initialisation de la référence par appel de la factory method
    this.square = this.createSquare();
  }

  @Test
  public void testAddPlayer() {
    // square sera adaptée par chaque sous-classe de test
    assertTrue(square.getPlayers().isEmpty());
    Player p = new Player(null);
    square.addPlayer(p);
    assertTrue(square.getPlayers().contains(p));
    assertEquals(1,square.getPlayers().size());

  }
}
```

CC BY NC SA

Les sous-classes de test implémentent donc `createSquare` pour préciser l'instance utilisée pour `square`. Les tests de cette classe sont passés avec succès si les tests de la classe mère sont en succès. On vérifie ainsi bien que `YellowSquare` respecte le contrat de `addPlayer` défini par le type `Square`.

```
public class YellowSquareTest extends AbstractSquareTest {
  // concrétisation de la fatory method avec création d'une instance adaptée
  public Square createSquare() {
    return new YellowSquare(1);
  }
}
```

Tests : héritage de tests (bis)

On retrouve le même pattern dans le TD Transports :

```
public abstract class ShipmentTest {

  protected abstract Shipment createShipment();

  protected Shipment shipment;

  @Before
  public void setupBefore() {
    this.shipment = this.createShipment();
  }
  ...
}
```

Avec, par exemple:

```
public class AirShipmentTest extends ShipmentTest {
  public Shipment createShipment() {
    return new AirShipment(10);
  }
}
```