Appendix: Creational Design Patterns

Simple Factory

- vytváření objektů konkrétního typu je zapouzdřené do metod ve specializované třídě
- uživatel odstíněn od komplexity vytváření objektů

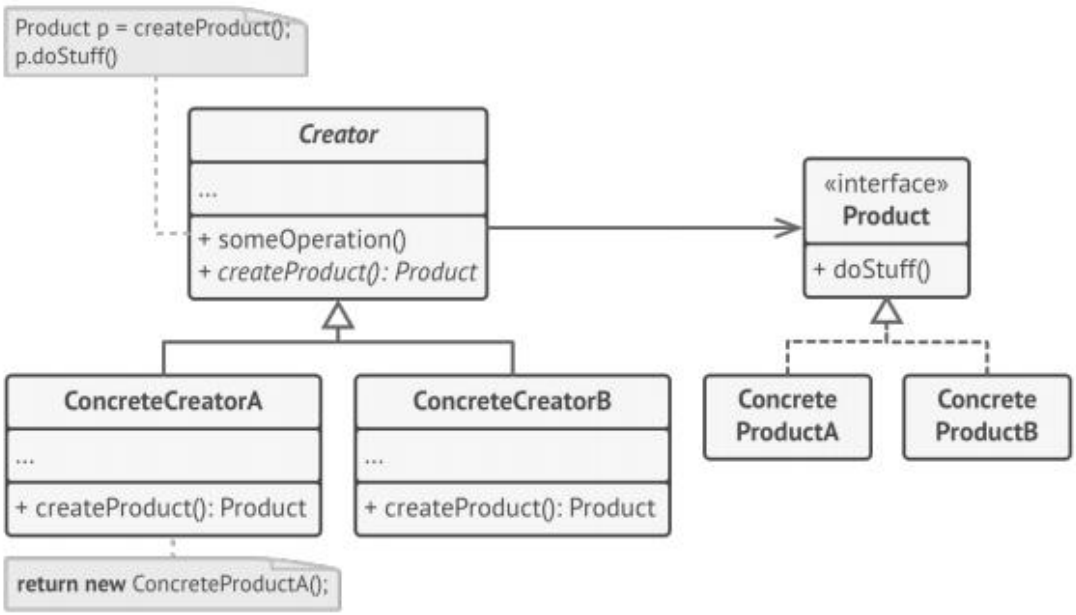
- **příklad (PlantFactory):**
  - může sloužit dalším vývojářům jako API na vytváření ovoce
  - vymezuje všechny typy ovoce, co chci vyrábět
  - vymezuje metody umožňující vytváření ovoce
  - mohu přidat aspekty typu „Kolik ovoce jsem vyrobil?“, logování,…

```
class PlantFactory {
      public Apple makeApple(int size, Boolean hasGoodTaste) {
            // Code for creating an Apple here.
      }
      public Orange makeOrange(int size) {
            // Code for creating an orange here.
      }
}

...
PlantFactory plantFactory = new PlantFactory()
Apple bigTastyApple = plantFactory.makeApple(10, true)
Apple smallUglyApple = plantFactory.makeApple(2, false)
Orange orange = plantFactory.makeOrange(5)
```

Factory method

- použití za účelem vytvoření objektu, nechceme-li specifikovat konkrétní třídu vytvářeného objektu
- místo konstruktoru se volá factory metoda (implementována/redefinována v potomkovi)



- druh objektu i počáteční vlastnosti jsou dané přijatými parametry (případně stavem factory)
- užíváme tam, kde je vytvářený objekt nějak odvozený od aktuální instance třídy poskytující Factory metodu
- časté hlavně u immutable tříd

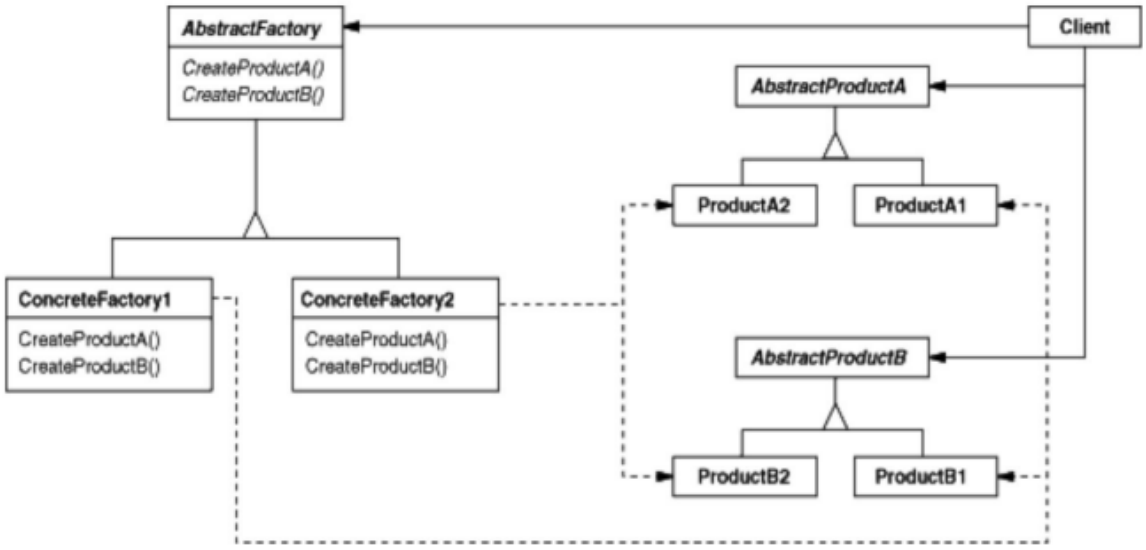
```
abstract class FruitCreator {
      protected abstract Fruit makeFruit(int size);
      public void pickFruit() {
            private final Fruit f = makeFruit();
       }
}

class OrangeCreator extends FruitCreator {
      @Override protected Fruit makeFruit(int size){
            if (size > 5)
                  return new BigOrange();
            else
                  return new SmallOrange();

       }
}
...
FruitCreator fruitCreator = new FruitCreator();
Fruit orange = fruitCreator.makeFruit(4) //returns
SmallOrange instance
```

Abstract Factory

- abstraktní rozšíření SimpleFactory
- chceme-li vytvářet celou rodinu objektů, ale v různých variantách
- z FM – není třeba přesně specifikovat produkt, který vytváříme (jen typy)



Vytváření produktů:

```
interface PlantFactory {
      Plant makePlant();
      Picker makePicker();
}
public class AppleFactory implements PlantFactory {
      Plant makePlant() {
            return new Apple();
      }
      Picker makePicker() {
            return new ApplePicker();
      }
}
public class OrangeFactory implements PlantFactory {
      Plant makePlant() {
            return new Orange();
      }
      Picker makePicker() {
            return new OrangePicker();
      }
}
```

Produkty:

```
abstract class Plant {
      ...
}

class Apple extends Plant {
      ...
}

class Orange extends Plant {
      ...
}

abstract class Picker {
      ...
}

abstract class ApplePicker extends Picker {
      ...
}
```

Client:

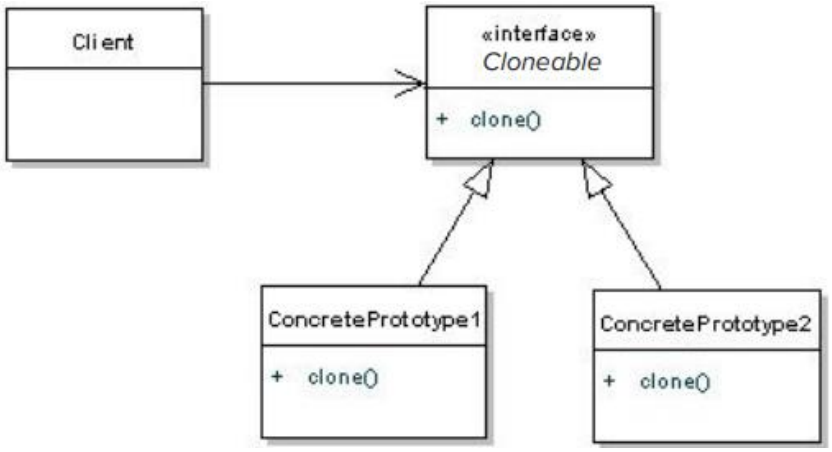
```
PlantFactory appleFactory = new AppleFactory();
PlantFactory orangeFactory = new OrangeFactory();
Plant apple = appleFactory.makePlant();
...
```

Prototype

- kontrolované vytváření kopií objektů

- **Shallow copy**
  - clone() vytvoří novou instanci téže třídy
  - nastaví všechny atributy na hodnoty atributů z volající instance

- **Deep copy**
  - clone() provede shallow copy, pak i clone() všech navázaných objektů
  - naklonuje celý objektový graf



```java
public interface Cloneable {
   Cloneable clone();
}
public class Address implements Cloneable {
   private String street;
   public Address(String street){
       this.street = street;
   }
   @Override
   public Cloneable clone() {
       return new Address(this.street);
   }
}
```

```java
public class Employee implements Cloneable {
   private String name;
   private Address address;
   public Employee(String name, Address address){
       this.name = name;
       this.address = address;
   }
   @Override
   public Cloneable clone() {
       return new Employee(this.name, (Address)
address.clone());
   }
                                                }
```

Client:

```java
...
Employee employee = new Employee("Karel Novak", new Address("Karlova 12"));
Employee employeeCopy = (Employee) employee.clone();
```