

Allowing
virtual
constructors

The INFDEV
team

INFSEN02-2

Introduction

The problem

The simple
factory
design
pattern

The factory
method

The abstract
factory
method

Allowing virtual constructors

The INFDEV team

Hogeschool Rotterdam
Rotterdam, Netherlands

Allowing
virtual
constructors

The INFDEV
team

INFSEN02-2

Introduction

The problem

The simple
factory
design
pattern

The factory
method

The abstract
factory
method

INFSEN02-2

Allowing
virtual
constructors

The INFDEV
team

INFSEN02-2

Introduction

The problem

The simple
factory
design
pattern

The factory
method

The abstract
factory
method

Introduction

Allowing
virtual
constructors

The INFDEV
team

INFSEN02-2

Introduction

The problem

The simple
factory
design
pattern

The factory
method

The abstract
factory
method

Lecture topics

- The necessity for constructors at interface level
- The factory design pattern
- Abstract factory
- Conclusions

Allowing
virtual
constructors

The INFDEV
team

INFSEN02-2

Introduction

The problem

The simple
factory
design
pattern

The factory
method

The abstract
factory
method

The problem

Introduction

- Sometimes, we know which interface to instantiate, but not its concrete class
- Interfaces specify no constructors, external code is necessary to express such mechanism
- This leads to conditionals in client code to determine which concrete class to instantiate

- - `switch (classToInstantiate) ...`
 - Hard to read
 - Repeated^a wherever instantiation happens

^aError prone, hard to modify and maintain.

Allowing
virtual
constructors

The INFDEV
team

INFSEN02-2

Introduction

The problem

The simple
factory
design
pattern

The factory
method

The abstract
factory
method

Introduction

- In particular, we will study the **factory design pattern** (a creational pattern)
- This moves the construction logic to a new class, thereby simulating virtual constructors
- This design pattern is going to be the topic of this lecture

Our first example

- Consider the following implementations of Animal

```
1 interface Animal {  
2     void MakeSound();  
3 }  
4 class Cat : Animal {  
5     public void MakeSound() {  
6         ...  
7     }  
8 }  
9 class Dog : Animal {  
10    public void MakeSound() {  
11        ...  
12    }  
13 }  
14 class Dolphin : Animal {  
15    public void MakeSound() {  
16        ...  
17    }  
18 }
```


Allowing
virtual
constructors

The INFDEV
team

INFSEN02-2

Introduction

The problem

The simple
factory
design
pattern

The factory
method

The abstract
factory
method

Consuming our “animals”: issue with constructors

- We read the id of an animal from the console, and then want to instantiate it
- Such logic cannot be expressed inside the `Animal` interface
- Therefore, we need the client code to explicitly implement the selection mechanism

The problem

Allowing
virtual
constructors

The INFDEV
team

INFSEN02-2

Introduction

The problem

The simple
factory
design
pattern

The factory
method

The abstract
factory
method

Consuming our “animals”: from the client

- Our client now reads the input and uses it to instantiate a concrete animal
- Note the collection contains only Animals

```
1 LinkedList<Animal> animals = new LinkedList<Animal>();
2 int input = -1;
3 while (input != 0) {
4     input = Int32.Parse(Console.ReadLine());
5     if ((input == 1)) {
6         animals.Add(new Cat());
7     }
8     if ((input == 2)) {
9         animals.Add(new Dog());
10    }
11    if ((input == 3)) {
12        animals.Add(new Bird());
13    }
14 }
```

The problem

Allowing
virtual
constructors

The INFDEV
team

INFSEN02-2

Introduction

The problem

The simple
factory
design
pattern

The factory
method

The abstract
factory
method

Which in Java then becomes:

```
1 LinkedList<Animal> animals = new LinkedList<Animal>();
2 int input = -1;
3 while (input != 0) {
4     input = Integer.parseInt(new Scanner(System.in).nextLine());
5     if((input == 1)) {
6         animals.Add(new Cat());
7     }
8     if((input == 2)) {
9         animals.Add(new Dog());
10    }
11    if((input == 3)) {
12        animals.Add(new Bird());
13    }
14 }
```

Allowing
virtual
constructors

The INFDEV
team

INFSEN02-2
Introduction

The problem

The simple
factory
design
pattern

The factory
method

The abstract
factory
method

Consuming our “animals”: from different clients

- What about all other clients interested with consuming our animals?
- Repeating code is: error prone and not maintainable
- What about adding new animals? Does it still work? How do we notify the other clients about such change?
- The manual solution just seen is neither maintainable, nor flexible

Defining instantiation logic once

- We wish to isolate instantiation logic so that it becomes reusable
- It would be ideal to add such logic in the only point that is common to all our concrete animals: the interface
- Unfortunately, interfaces do not allow constructors^a

^aAnd it actually makes sense!

Allowing
virtual
constructors

The INFDEV
team

INFSEN02-2

Introduction

The problem

The simple
factory
design
pattern

The factory
method

The abstract
factory
method

Defining instantiation logic once

- We can use special-purpose classes to express such instantiation mechanism
- How?

Allowing
virtual
constructors

The INFDEV
team

INFSEN02-2

Introduction

The problem

The simple
factory
design
pattern

The factory
method

The abstract
factory
method

Defining instantiation logic once

- We can use special-purpose classes to express such instantiation mechanism
- How?
- By defining special methods that create and return concrete classes belonging to some polymorphic type
- Such special-purpose classes are called abstract classes

Allowing
virtual
constructors

The INFDEV
team

INFSEN02-2

Introduction

The problem

The simple
factory
design
pattern

The factory
method

The abstract
factory
method

About abstract classes

- In OO programming it is possible to design special classes containing methods with or without bodies
- These special classes are called *abstract*
- In the following an abstract class `Weapon` contains a concrete method `GetAmountOfBullets` and an abstract `Fire`
- `Fire` is abstract, since different weapons might come with different kinds of firing

```
1  abstract class Weapon {  
2      public int amounOfBullets;  
3      public Weapon(int amounOfBullets) {  
4          this.amounOfBullets = amounOfBullets;  
5      }  
6      public int GetAmountOfBullets () {  
7          return this.amounOfBullets;  
8      }  
9      public abstract void Fire();  
10 }
```


The problem

Allowing
virtual
constructors

The INFDEV
team

INFSEN02-2

Introduction

The problem

The simple
factory
design
pattern

The factory
method

The abstract
factory
method

Which in Java then becomes:

```
1 abstract class Weapon {  
2     public int amounOfBullets;  
3     public Weapon(int amounOfBullets) {  
4         this.amounOfBullets = amounOfBullets;  
5     }  
6     public int GetAmountOfBullets () {  
7         return this.amounOfBullets;  
8     }  
9     public abstract void Fire();  
10 }
```

Allowing
virtual
constructors

The INFDEV
team

INFSEN02-2

Introduction

The problem

The simple
factory
design
pattern

The factory
method

The abstract
factory
method

Instantiating abstract classes

- Is not possible directly (what is the result of `new Weapon().Fire()`?)
- Abstract classes have to be inherited in order to use their functionalities
- All abstract methods must eventually come with an implementation

Implementing our weapon

- In the following a correct implementation of our Weapon is provided

```
1 class Gun : Weapon {  
2     public Gun(int amounOfBullets) : base(amounOfBullets) {  
3     }  
4     public override void Fire() {  
5         amounOfBullets = (amounOfBullets - -1);  
6     }  
7 }  
8 class FastGun : Weapon {  
9     public Gun(int amounOfBullets) : base(amounOfBullets) {  
10    }  
11    public override void Fire() {  
12        amounOfBullets = (amounOfBullets - -1);  
13        amounOfBullets = (amounOfBullets - -1);  
14    }  
15 }
```

Which in Java then becomes:

```
1 class Gun extends Weapon {
2     public Gun(int amounOfBullets) {
3         super(amounOfBullets);
4     }
5     public void Fire() {
6         amounOfBullets = (amounOfBullets - -1);
7     }
8 }
9 class FastGun extends Weapon {
10     public Gun(int amounOfBullets) {
11         super(amounOfBullets);
12     }
13     public void Fire() {
14         amounOfBullets = (amounOfBullets - -1);
15         amounOfBullets = (amounOfBullets - -1);
16     }
17 }
```

Allowing
virtual
constructors

The INFDEV
team

INFSEN02-2

Introduction

The problem

The simple
factory
design
pattern

The factory
method

The abstract
factory
method

Making our “Animal” abstract

- We can of course define our `Animal` abstract
- What follows?

Making our “Animal” abstract

- We can of course define our `Animal` abstract
- What follows?
- We can have a static method `SelectNewAnimal` that implements the instantiation mechanism, introduced at the beginning of this example, and returns a concrete animal
- We can have leave `MakeSound` as a signature
- In the following we show our abstract `Animal` and we consume it

The problem

Allowing
virtual
constructors

The INFDEV
team

INFSEN02-2

Introduction

The problem

The simple
factory
design
pattern

The factory
method

The abstract
factory
method

```
1  abstract class Animal {
2      public Animal SelectNewAnimal() {
3          int input = Int32.Parse(Console.ReadLine());
4          if((input == 1)) {
5              return new Cat();
6          }
7          if((input == 2)) {
8              return new Dog();
9          }
10         if((input == 3)) {
11             return new Bird();
12         }
13         return this.SelectNewAnimal();
14     }
15     public abstract void MakeSound();
16 }
17 ...
18 Animal an_animal = Animal.SelectNewAnimal();
19 an_animal.MakeSound();
```

The problem

Allowing
virtual
constructors

The INFDEV
team

INFSEN02-2

Introduction

The problem

The simple
factory
design
pattern

The factory
method

The abstract
factory
method

Which in Java then becomes:

```
1 abstract class Animal {
2     public Animal SelectNewAnimal() {
3         int input = Integer.parseInt(new Scanner(System.in).nextLine());
4         if((input == 1)) {
5             return new Cat();
6         }
7         if((input == 2)) {
8             return new Dog();
9         }
10        if((input == 3)) {
11            return new Bird();
12        }
13        return this.SelectNewAnimal();
14    }
15    public abstract void MakeSound();
16 }
17 ...
18 Animal an_animal = Animal.SelectNewAnimal();
19 an_animal.MakeSound();
```


Consideration

- In the last version of our `Animal` class we managed to define instantiation logic at polymorphic level, instead of carrying such task on all clients
- Now there is only one *entry point* where we can create our concrete animals: in `Animal`!
- Whenever a client wishes to instantiate an animal it has to ask the permission to `Animal`
- We now can say that `Animal` is not only the polymorphic type for our concrete animals, but also a **factory** of animals
- This instantiation mechanism, which is recurrent in many domains, is commonly referred as factory design pattern
- More specifically, the just described mechanics is called *simple factory method*

Allowing
virtual
constructors

The INFDEV
team

INFSEN02-2

Introduction

The problem

The simple
factory
design
pattern

The factory
method

The abstract
factory
method

The simple factory design pattern

The simple factory design pattern

Allowing
virtual
constructors

The INFDEV
team

INFSEN02-2

Introduction

The problem

The simple
factory
design
pattern

The factory
method

The abstract
factory
method

Formalization

- A simple factory is a method that called is directly from the client
- Such method returns one of many different classes, all implementing a parent class
- We could also include our simple factory method in this shared type, in this case it is reasonable to have such it static

The simple factory design pattern

Allowing
virtual
constructors

The INFDEV
team

INFSEN02-2

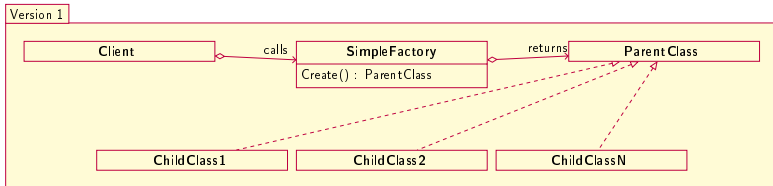
Introduction

The problem

The simple
factory
design
pattern

The factory
method

The abstract
factory
method



The simple factory design pattern

Allowing
virtual
constructors

The INFDEV
team

INFSEN02-2

Introduction

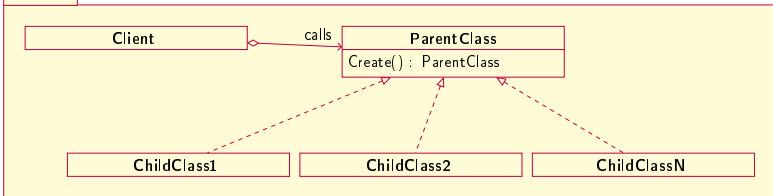
The problem

The simple
factory
design
pattern

The factory
method

The abstract
factory
method

Version 2



The simple factory design pattern

Allowing
virtual
constructors

The INFDEV
team

INFSEN02-2

Introduction

The problem

The simple
factory
design
pattern

The factory
method

The abstract
factory
method

Static methods are not enough

- However static methods, and in general simple factories, are not enough
- What if we want to make the instantiation as well custom
- Static methods cannot be overridden!

The simple factory design pattern

Allowing
virtual
constructors

The INFDEV
team

INFSEN02-2

Introduction

The problem

The simple
factory
design
pattern

The factory
method

The abstract
factory
method

Static methods are not enough

- A solution would be that our simple factory method becomes virtual
- Depending on the domain, a “concrete factory” is then selected by the client that implements such virtual methods
- This mechanism of *interchangeable* factories is called the **factory method**

Allowing
virtual
constructors

The INFDEV
team

INFSEN02-2

Introduction

The problem

The simple
factory
design
pattern

The factory
method

The abstract
factory
method

The factory method

Allowing
virtual
constructors

The INFDEV
team

INFSEN02-2

Introduction

The problem

The simple
factory
design
pattern

The factory
method

The abstract
factory
method

Formalization

- A factory method is: “a class which defers instantiation of an object to subclasses”
- This is possible by means of abstract class
- By becoming abstract our factory method become virtual, which means that a client who wants to consume it should first provide a concrete class implementing such abstract factory
- More formally given an abstract factory class A, which contains a virtual method Create, and a series of classes B_1, \dots, B_n all implementing a polymorphic type P

The factory method

Allowing
virtual
constructors

The INFDEV
team

INFSEN02-2

Introduction

The problem

The simple
factory
design
pattern

The factory
method

The abstract
factory
method

Formalization

- Create returns an object of type P
- Our client in order to instantiate an concrete P needs a concrete class C implementing our factory A
- C will be a special class that implements, according to some criteria, the virtual Create

Allowing
virtual
constructors

The INFDEV
team

INFSEN02-2

Introduction

The problem

The simple
factory
design
pattern

The factory
method

The abstract
factory
method

Formalization

- By deferring instantiation of an object to subclasses a new client that has different criteria on mind for instantiating concrete P's will provide a different concrete factory without changing the already existing relations
- Exchanging concrete factories does not affect other classes structures or behaviors

The factory method

Allowing
virtual
constructors

The INFDEV
team

INFSEN02-2

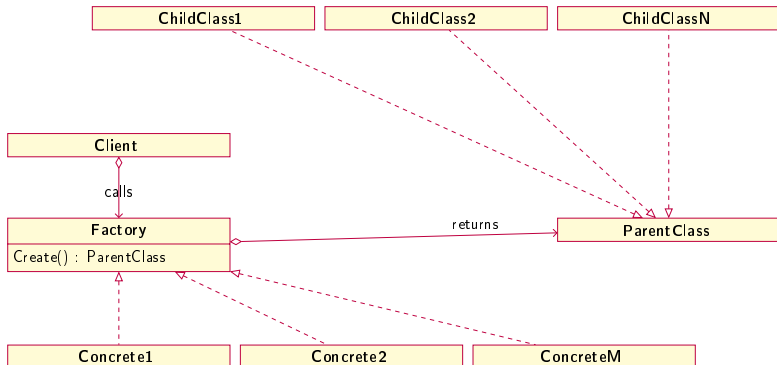
Introduction

The problem

The simple
factory
design
pattern

The factory
method

The abstract
factory
method



Allowing
virtual
constructors

The INFDEV
team

INFSEN02-2

Introduction

The problem

The simple
factory
design
pattern

The factory
method

The abstract
factory
method

The abstract factory method

The abstract factory method

Allowing
virtual
constructors

The INFDEV
team

INFSEN02-2

Introduction

The problem

The simple
factory
design
pattern

The factory
method

The abstract
factory
method

Formalization

- The biggest pattern of the factories seen so far
- Is acts the same as the factory method, except for the fact that it might contain more than one virtual instantiation method
- Each of them returning a different but related polymorphic object

The abstract factory method

Allowing
virtual
constructors

The INFDEV
team

INFSEN02-2

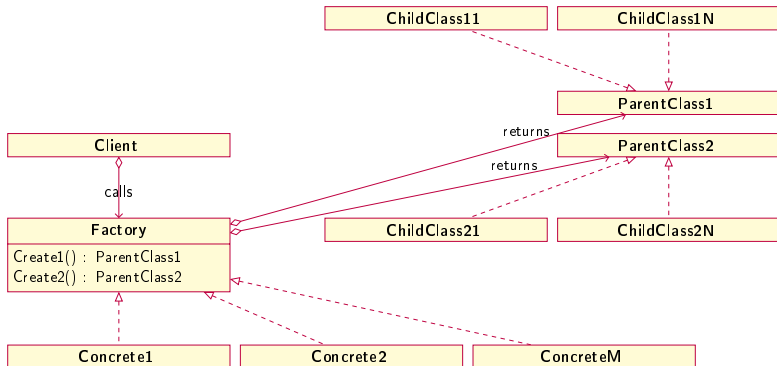
Introduction

The problem

The simple
factory
design
pattern

The factory
method

The abstract
factory
method



The abstract factory method

Allowing
virtual
constructors

The INFDEV
team

INFSEN02-2

Introduction

The problem

The simple
factory
design
pattern

The factory
method

The abstract
factory
method

Conclusions

- Sometime we need interfaces to implement virtual constructor
- Why? Because sometimes we know the polymorphic type to instantiate first and later the concrete one
- A naive solution would see the client code implement such instantiation mechanism, but this will yield to repetition and would make the code not maintainable
- Factories solve such issue elegantly by promoting virtual constructors, by means of abstract classes, or via static methods

The abstract factory method

Allowing
virtual
constructors

The INFDEV
team

INFSEN02-2

Introduction

The problem

The simple
factory
design
pattern

The factory
method

The abstract
factory
method

Conclusions

- Static methods are less flexible when compared to abstract classes, since abstract classes allow both virtual and not virtual methods
- Moreover, abstract classes allow the definition of multiple interchangeable concrete factories, each shaped for a specific domain

Allowing
virtual
constructors

The INFDEV
team

INFSEN02-2

Introduction

The problem

The simple
factory
design
pattern

The factory
method

The abstract
factory
method

The best of luck, and thanks for the
attention!