Design Pattern : Factory Patterns

daybreaker@SPARCS, KAIST

2007. 5. 1

Contents

- Abstract Factory
- Factory Method
- Discussion
  - Reflection

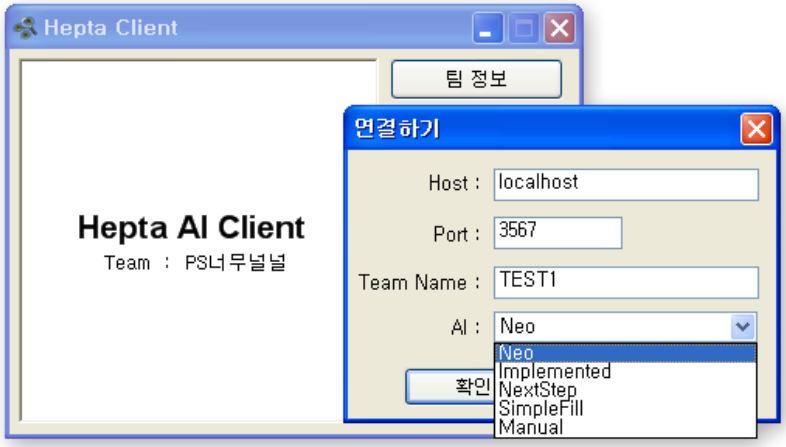
※ 이번 스터디는 사전 발표 자료를 준비하지 못한 관계로 토론 형태로 진행되었음
(이 자료는 스터디 후 토론된 내용을 기록한 것임)

Getting Started: Creational Patterns

- Use when?
  - 어떤 class가 자신이 사용할 class를 생성하고자 할 때
- Patterns in the textbook
  - Abstract Factory
  - Builder
  - Factory Method
  - Prototype
  - Singleton

Abstract Factory

- 경험담: CS202 Problem Solving AI 프로젝트



What to do:

사용자가 AI를 선택하게 하고 싶다.

What we have:

하나의 AI interface를 서로 다른 방법으로 구현한 여러 AI class들

```
HeptaAI ai;
switch (si.AIName) {
case "SimpleFill":
            ai = new HeptaAI_SimpleFill(f.gd);
            break;
case "Implemented":
            ai = new HeptaAI_Implemented(f.gd);
            break;
case "NextStep":
            ai = new HeptaAI_NextStep(f.gd);
            break;
case "Neo":
            ai = new HeptaAI_Neo(f.gd);
            break;
case "Manual":
default:
            ai = new HeptaAI_Manual(f.gd);
            break;
}
...
Step next = ai.thinkTheNextStep();
```

Factory로 정리 가능하지 않을까?!

Abstract Factory

- 자신이 사용할 class : product
- product 생성 작업을 다른 제3자(factory)에게 맡기자.
  - 그리고 그 factory를 *갈아 끼울* 수 있게 하자.

- 깔아 끼우는 방법
  - 사용하는 언어의 다형성(polymorphism) 이용
  - Java/C++ 등은 subtyping

Abstract Factory

- Minimal Java example

```
interface Product {
    void doSomething();
}

interface Factory {
    Product createProduct();
}

class FactoryA implements Factory {
    Product createProduct() { return new ProductA(); }
}

class FactoryB implements Factory {
    Product createProduct() { return new ProductB(); }
}

class ProductA implements Product {
    void doSomething() {
        // A's implementation
    }
}

class ProductB implements Product {
    void doSomething() {
        // B's implementation
    }
}
```

```
/* usage example */
Factory f;
if (useA)
    f = new FactoryA();
else if (useB)
    f = new FactoryB();
// ...
Product p = f.createProduct();
p.doSomething();
```

Abstract Factory

- 응용 예
  - 앞서 소개한 AI 프로젝트
  - Database Backend 추상화
    - JDBC : factory
    - Oracle JDBC Driver : product
    - MySQL JDBC Driver : product

Factory Method

- Product 생성을 제3자에게 맡기지 않고, 자기 자신의 method로 구현
  - '갈아 끼우기'를 위해서 자기 자신을 상속
  - 잘 사용되지 않음

Discussion

- 갈아 끼우기
  - 어떻게 갈아 끼우지?
  - 코드를 작성할 때 생성될 product가 한정되어 있다면?
    - 그냥 subtyping과 switch case / if ~ else로 해결 가능
    - String-factory mapping
  - 처음 코드작성자가 아닌 임의의 누군가가 factory를 제공, 나름대로의 product를 제공한다면?
    - Plug-in architecture
    - 컴파일할 때 미래의 모든 가능성을 다 담을 수 없다.
    - *Reflection* 이용!

Discussion

- What is Reflection?
  - 코드가 코드를 inspect할 수 있다.
  - 코드가 코드를 만들 수 있다.
  - 코드가 코드를 실행할 수 있다.

???

- Reflection 활용 예(inspection)
  - object type으로 받은 임의의 object가 가진 method/property를 알고 싶을 때
  - 그러한 method/property를 호출/사용하고 싶을 때
    - 언어에 따라 Class, Method, Type 등과 같은 class를 제공한다.

Discussion

- C에서 reflection이 가능한가?
  - DLL 등을 이용한 plug-in 구조도 '미리 정의된 형식'을 지켜야 함
- .NET Framework, Java, Python, …
  - (사실상의) Virtual Machine 기반 언어
  - '동적' 코드 실행 가능
  - 언어 기능만으로 외부 class loading, 필요한 method가 있는지 검사, 있으면 사용, 없으면 자체 처리하는 등의 동작 가능
  - Python, ECMA-Script, Ruby 등의 경우 임의의 object에 실시간으로 method, property를 추가할 수 있다.
  - Higher-order function?

끝

THE END