Factory Method Pattern

드라이 뱅(Dry Bang)씨가 그들의 인기 게임에 새로운 레벨를 만들자고 제안한다. 레벨이 많으면 그만큼 돈도 번다.

페드로: 새 레벨들을 어떻게 만들 수 있나요?

이브: 그저 리소스를 바꾸고 블럭들을 새로 만들고, 종이, 나무, 철…

페드로: 그건 너무 바보같지 않아요?

이브: 게임 전체가 바보같죠. 유저들이 자신의 캐릭터가 쓸 색깔 모자에 돈을 지불한다면, 나무 블럭에도 지불할 거에요.

페드로: 말도 않되는 거 같지만, 어쨌든, MazeBuilder를 일반적으로 만들고 블럭의 각 타입에 맞는 빌더를 추가합시다. 그건 팩토리 메소드 패턴이죠.

```
class Maze { }
class WoodMaze extends Maze { }
class IronMaze extends Maze { }

interface MazeBuilder {
  Maze build();
}

class WoodMazeBuilder {
  @Override
  Maze build() {
    return new WoodMaze();
  }
}

class IronMazeBuilder {
  @Override
  Maze build() {
    return new IronMaze();
  }
}
```

이브: IronMazeBuilder가 IronMazes를 리턴하는 것은 당연하지 않나요?

페드로: 프로그램한테는 당연한 게 아니죠. 하지만 봅시다, 다른 블럭으로부터 미로를 만들기 위해 우리는 블럭 생성에 책임이 있는 구현체를 단지 변경했어요.

```
MazeBuilder builder = new WoodMazeBuilder();
Maze maze = builder.build();
```

이브: 전에 비슷한 것을 봤어요.

페드로: 무엇을요?

이브: 저에게는 이것은 전략 패턴이나 상태 패턴과 비슷해 보여요.

페드로: 말도 안돼요. 전략 패턴은 특정 동작을 수행하는 것에 대한 것이고, 팩토리 패턴은 특정 객체를 만들기 위한 것이에요.

이브: 하지만 생성도 또한 동작이죠.

```
(defn maze-builder [maze-fn])

(defn make-wood-maze [])
(defn make-iron-maze [])

(def wood-maze-builder (partial maze-builder make-wood-maze))
(def iron-maze-builder (partial maze-builder make-iron-maze))
```

페드로: 흠, 비슷해 보이네요.

이브: 생각해 보세요.

페드로: 사용 예제 같은 게 있나요?

이브: 아니요, 모든 것이 여기서는 뻔한거라, 그저 전략 패턴이나 상태 패턴, 그리고 템플릿 메소드 패턴 에피소드를 다시 읽으면 돼요.

관련 문서

Plugin Backlinks: 아무 것도 없습니다.