创建模式
(Creational Patterns)

概述

- 控制对象的的创建过程
- 提供更好的可扩展性
- 提高对象的可管理性
- 降低耦合性

案例分析-1

颜色（Color）对象属于资源消耗型对象，但在基于可视化界面的应用程序中广泛使用，如何设计才能有效避免重复创建资源消耗型对象?

案例分析-2

不同的浏览器对于DOM、CSS和Javascript的支持程度不同。对于同样功能的对象往往有着不同的创建方法。假设在一个基于浏览器的应用需要创建一组与平台相关的对象{O1,...On}。请问，如何设计才能降低应用程序对于具体平台的依赖程度，提升平台无关性?

案例分析-3

一款RPG游戏中包含丰富的迷宫设计，假设迷宫由墙、陷阱、门、树（障碍物）、怪物等概念组成。迷宫可以有不同的风格（丛林、地宫、都市、魔堡等，数量不定），决定了迷宫组成元素的视觉效果。作为迷宫模块的设计人员，如何设计一个能够从配置文件创建迷宫对象的接口?

案例分析-4

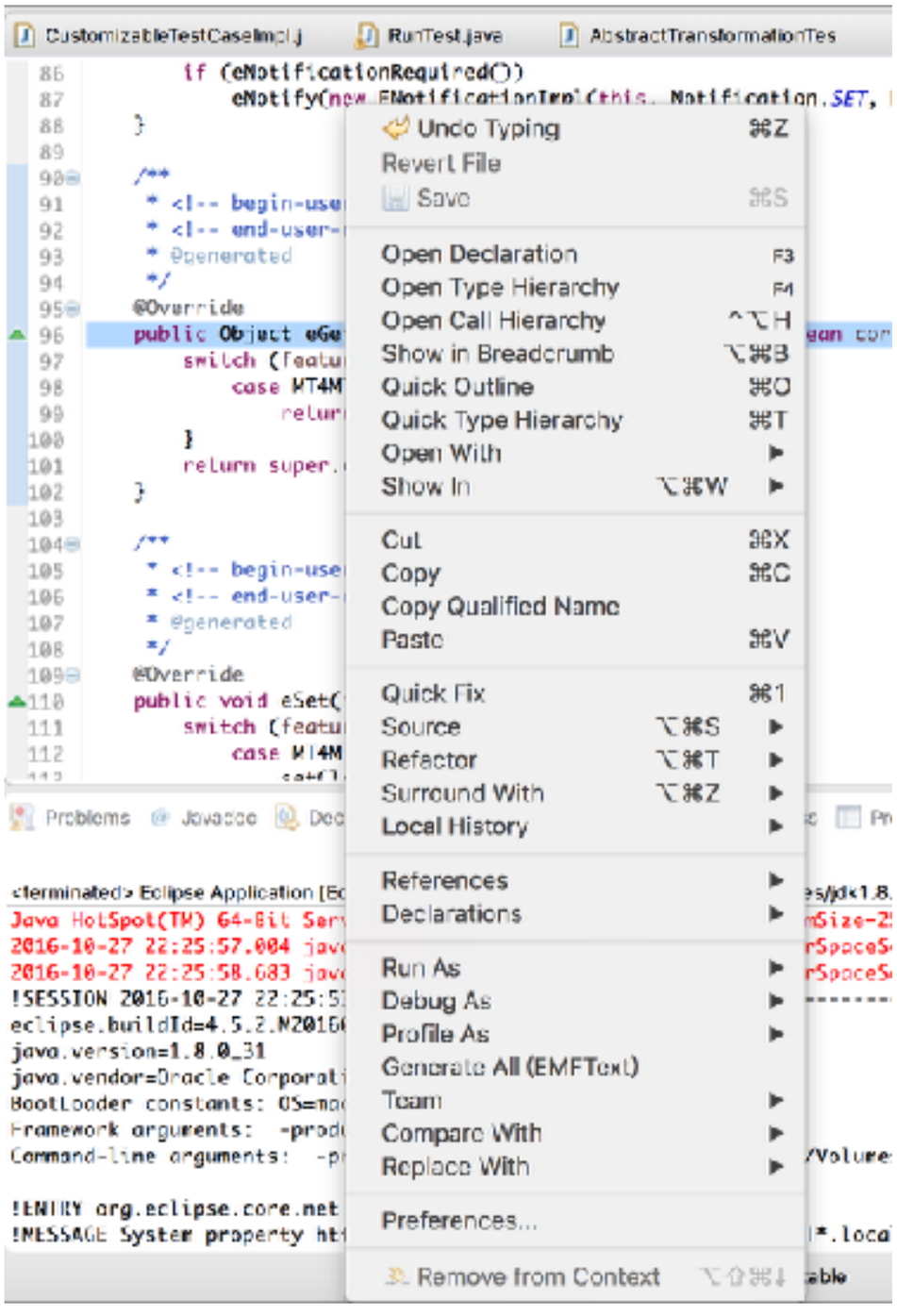
一款应用对于同样的对象提供不同的展示方式和控制机制（MVC），用户可以通过切换不同的视图风格进行选择。从基础框架的角度看，预先并不知道每种应用具有哪些视图风格。如何设计（视图和控制器部分对象的创建策略），才能有效支持这种类型的需求?

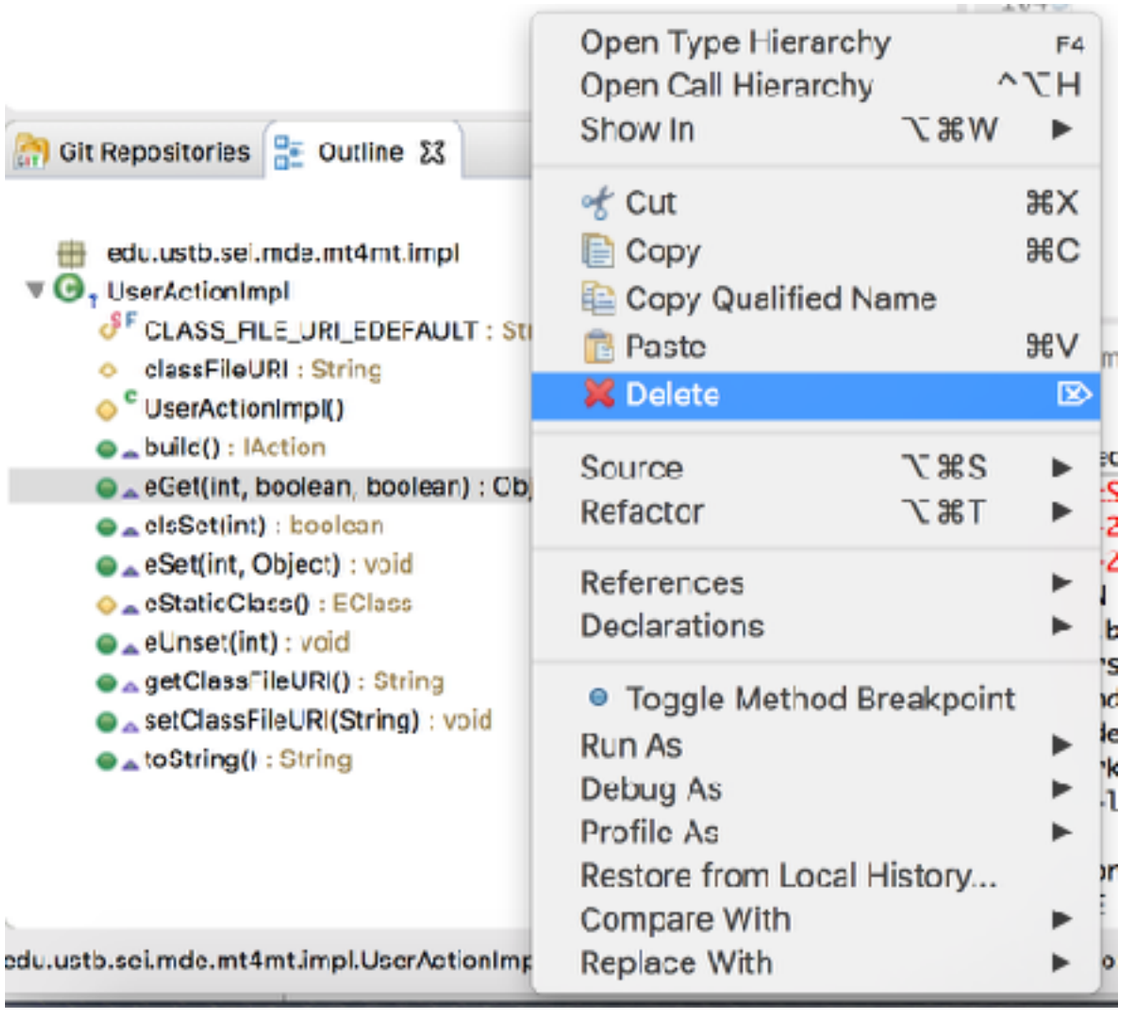
案例分析-5

一款应用对于同样的对象提供不同的展示方式和控制机制（MVC）。对象的表示方式决定了该对象的控制策略（或者控制方式决定了表示方式）。从框架的角度，对于每个对象都需要获知其表示方式和控制策略。

应如何设计，才能提高系统结构灵活性?

案例分析-5


CustomizableTestCaseImpl.j
RunTest.java
AbstractTransformationTes
Undo Typing ⌘Z
Revert File
Save ⌘S
Open Declaration F3
Open Type Hierarchy F4
Open Call Hierarchy ^⌥H
Show in Breadcrumb ⌥⌘B
Quick Outline ⌘O
Quick Type Hierarchy ⌘T
Open With
Show In ⌥⌘W
Cut ⌘X
Copy ⌘C
Copy Qualified Name
Paste ⌘V
Quick Fix ⌘1
Source ⌥⌘S
Refactor ⌥⌘T
Surround With ⌥⌘Z
Local History
References
Declarations
Run As
Debug As
Profile As
Generate All (EMFText)
Team
Compare With
Replace With
Preferences...
Remove from Context ⌥⇧⌘↓
Problems
Javadoc
<terminated> Eclipse Application [Ec
Java HotSpot(TM) 64-Bit Ser
2016-10-27 22:25:57.004 jav
2016-10-27 22:25:58.083 jav
!SESSION 2016-10-27 22:25:5
eclipse.buildId=4.5.2.M2016
java.version=1.8.0_31
java.vendor=Oracle Corporati
BootLoader constants: OS=ma
Framework arguments: -produ
Command-line arguments: -p
!ENTRY org.eclipse.core.net
!MESSAGE System property ht


Git Repositories
Outline
edu.ustb.sei.mde.mt4mt.impl
UserActionImpl
CLASS_FILE_URI_EDEFAULT : St
classFileURI : String
UserActionImpl()
build() : IAction
eGet(int, boolean, boolean) : Ob
eIsSet(int) : boolean
eSet(int, Object) : void
eStaticClass() : EClass
eUnset(int) : void
getClassFileURI() : String
setClassFileURI(String) : void
toString() : String
edu.ustb.sei.mde.mt4mt.impl.UserActionImp
Open Type Hierarchy F4
Open Call Hierarchy ^⌥H
Show In ⌥⌘W
Cut ⌘X
Copy ⌘C
Copy Qualified Name
Paste ⌘V
Delete
Source ⌥⌘S
Refactor ⌥⌘T
References
Declarations
Toggle Method Breakpoint
Run As
Debug As
Profile As
Restore from Local History...
Compare With
Replace With

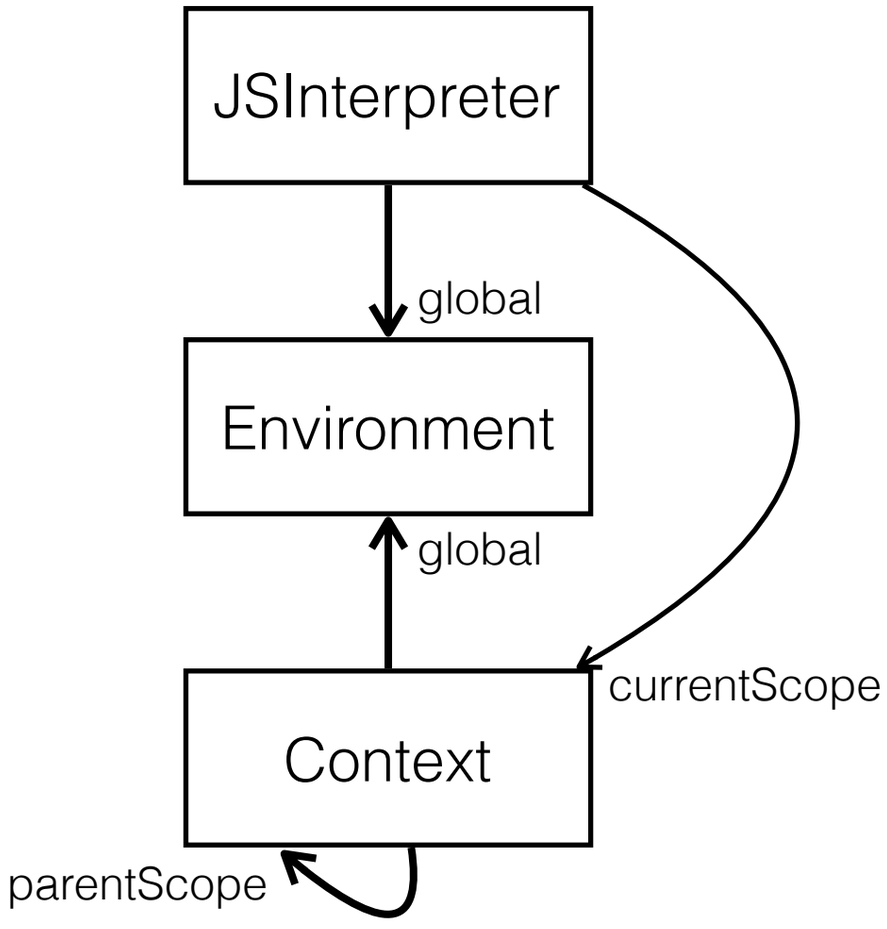
案例分析-6

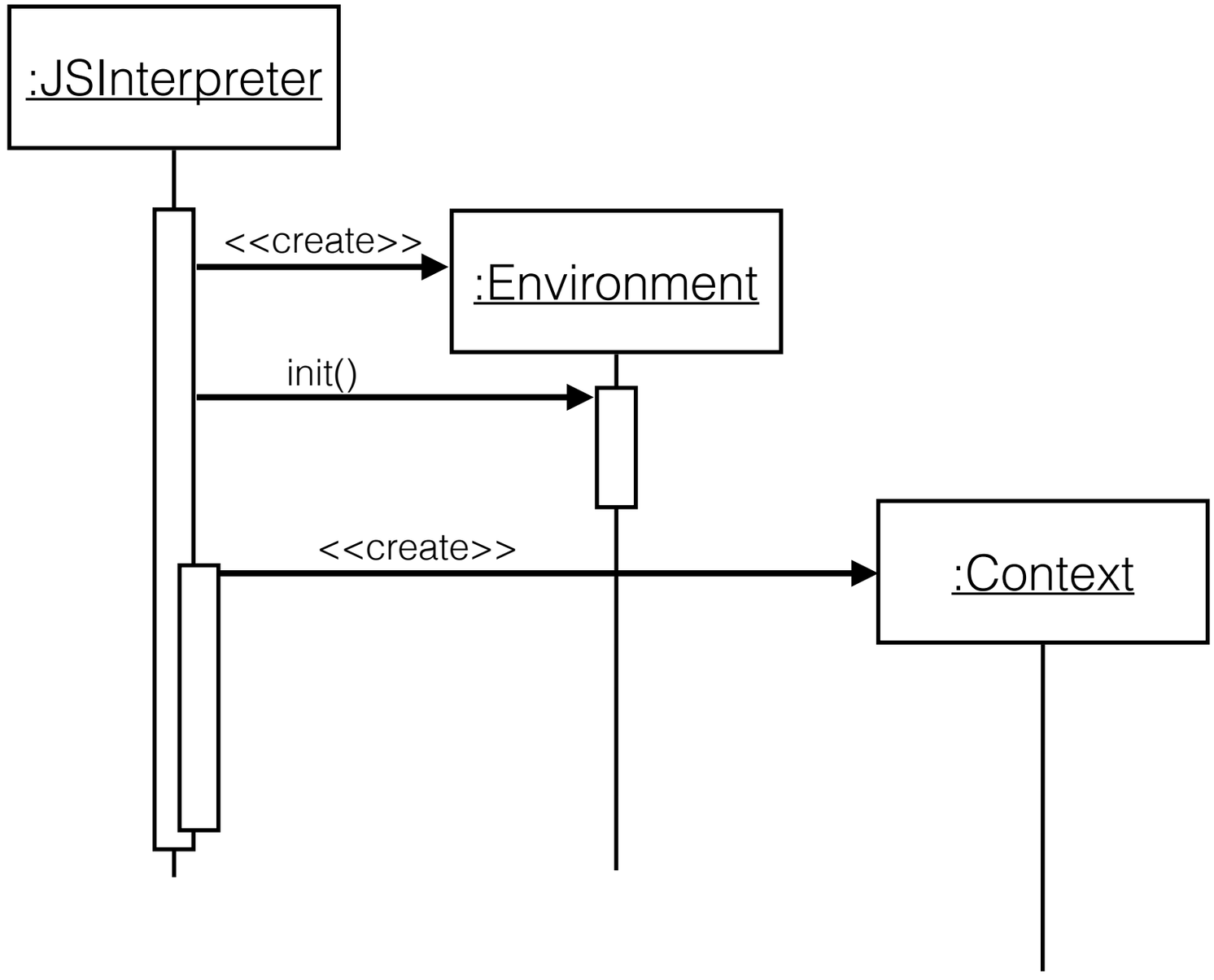
应用程序需要对所创建的对象进行自动跟踪和管理（例如分配ID或者注册事件监听器），如何设计才能实现上述功能（即避免开发人员绕过对象监管）？

案例分析-7

一款针对Javascript的程序分析器能够解释并执行Javascript语句。在进行上述工作时，该分析器需要使用两种对象Environment和Context。现有另一款工具需要扩展该Javascript分析器，并且设计了自己的MyEnvironment和MyContext类。作为Javascript分析器的设计者，应该如何设计才能提高整个工具的可扩展性?

案例分析-7


JSInterpreter
global
Environment
global
currentScope
Context
parentScope


:JSInterpreter
<<create>>
:Environment
init()
<<create>>
:Context

案例分析-8

文件选择窗口包含一组基本控件和一组可选控件。基本控件如文件路径输入框，确定按钮，取消按钮。可选控件如目录树、导航栏，预览视图等。如何设计能够简化文件选择窗口的构造过程?

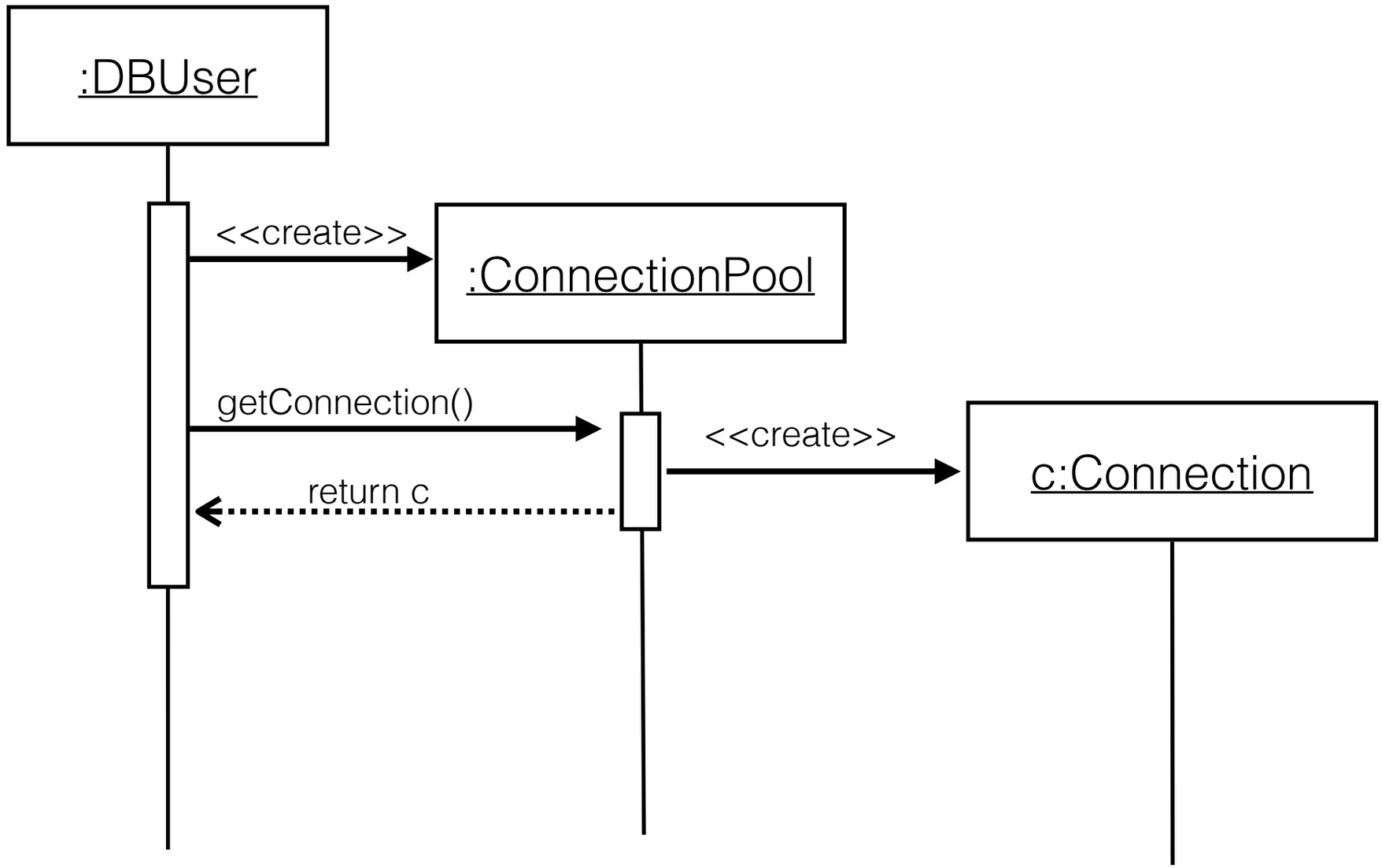
案例分析-9

一款矢量图编辑器允许用户绘制一个矢量图型后，将其存储到工具栏中。如何设计工具栏模块，以便支持用户的自定义?

案例分析-10

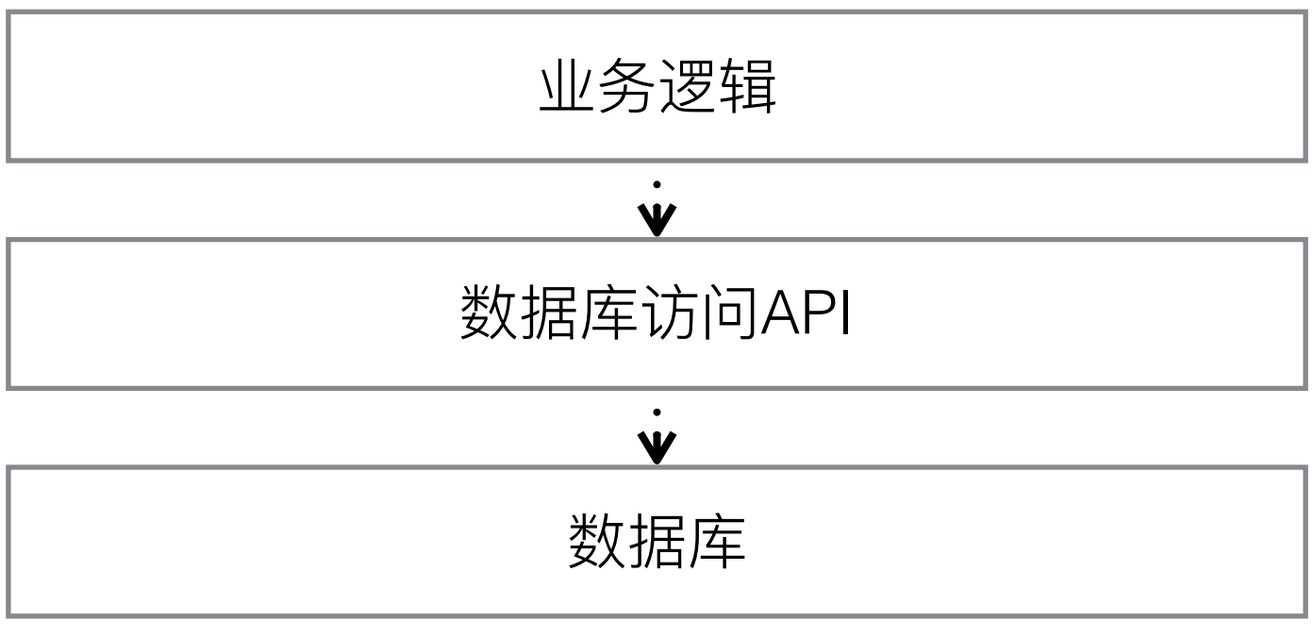
为了管理整个应用系统中的数据库连接，设立一个连接池供整个系统使用。由于连接池十分占用资源，必须确保该系统只能有一个连接池对象存在。如何设计以保证这一点?

案例分析-10


:DBUser
<<create>>
:ConnectionPool
getConnection()
<<create>>
c:Connection
return c

案例分析-11

一款基于数据库的应用系统，为了控制数据库的访问，需要提供一组基础的API（方法）供整个应用系统使用。为了在未来支持多种控制策略，应该如何设计?

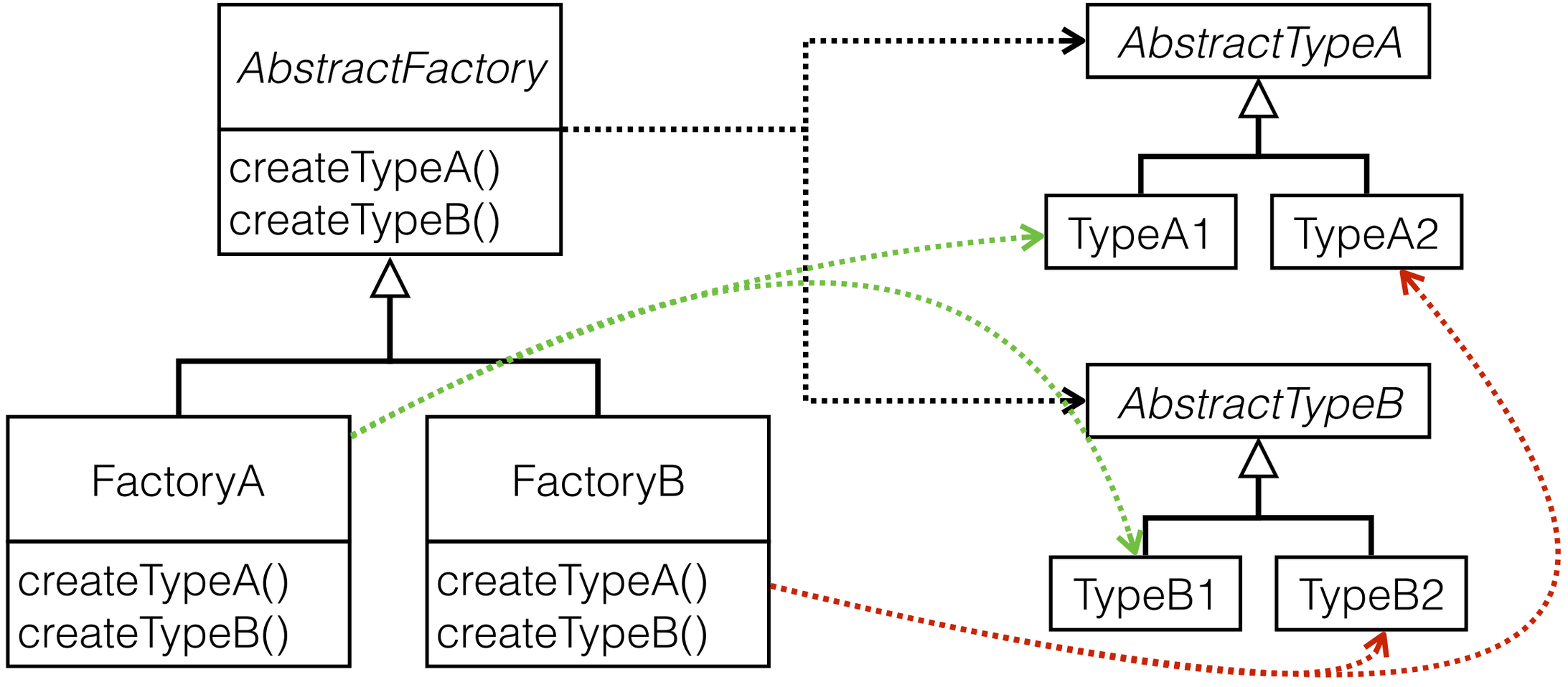


案例小结

- 这几个案例都涉及到对象创建
- 变化性：创建对象的过程具有变化性，不能通过硬编码的方式实现
- 管理性：由于某些管理需求，对象创建过程需要被控制

抽象工厂
Abstract Factory

- 意图：
  - 提供一个能够创建一组对象的接口，而不需要知道这些对象的具体类



抽象工厂
Abstract Factory

- 例子
  - javax.xml.transform.TransformerFactory

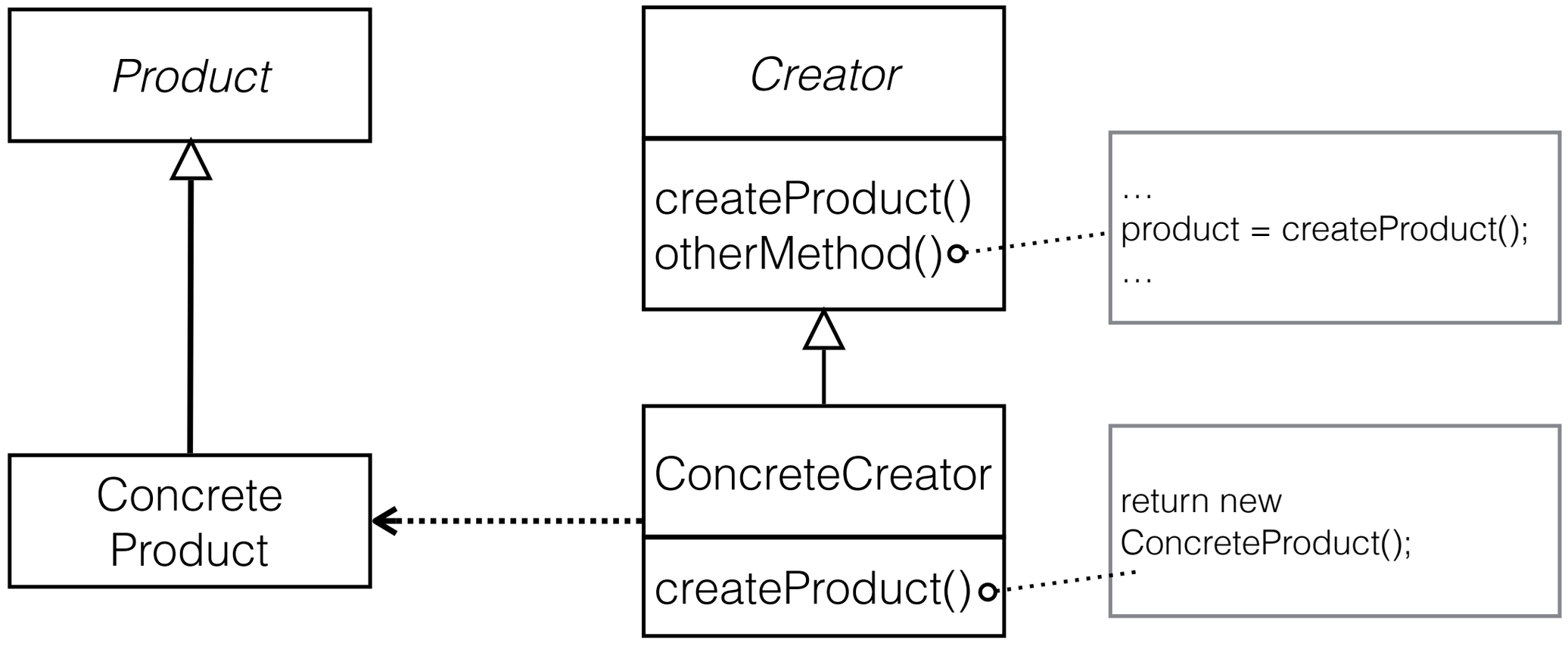
```
public static void output(Node node) {
        TransformerFactory transFactory=TransformerFactory.newInstance();
        try {
            Transformer transformer = transFactory.newTransformer();
            transformer.setOutputProperty("encoding", "gb2312");
            transformer.setOutputProperty("indent", "yes");

            DOMSource source=new DOMSource();
            source.setNode(node);
            StreamResult result=new StreamResult();
            result.setOutputStream(System.out);
            transformer.transform(source, result);
        } catch (TransformerConfigurationException e) {
            e.printStackTrace();
        } catch (TransformerException e) {
            e.printStackTrace();
        }
}
```

```
    Source xmlSource = new StreamSource(xmlFile);
    Source xsltSource = new StreamSource(xsltFile);
    // the factory pattern supports different XSLT processors
    TransformerFactory transFact =
TransformerFactory.newInstance();
    Transformer trans = transFact.newTransformer(xsltSource);
    trans.transform(xmlSource, new StreamResult(System.out));
```

工厂方法
Factory Method

- 意图：
  - 为创建一个对象提供一个接口（方法），但由子类确定应该创建哪个具体对象?



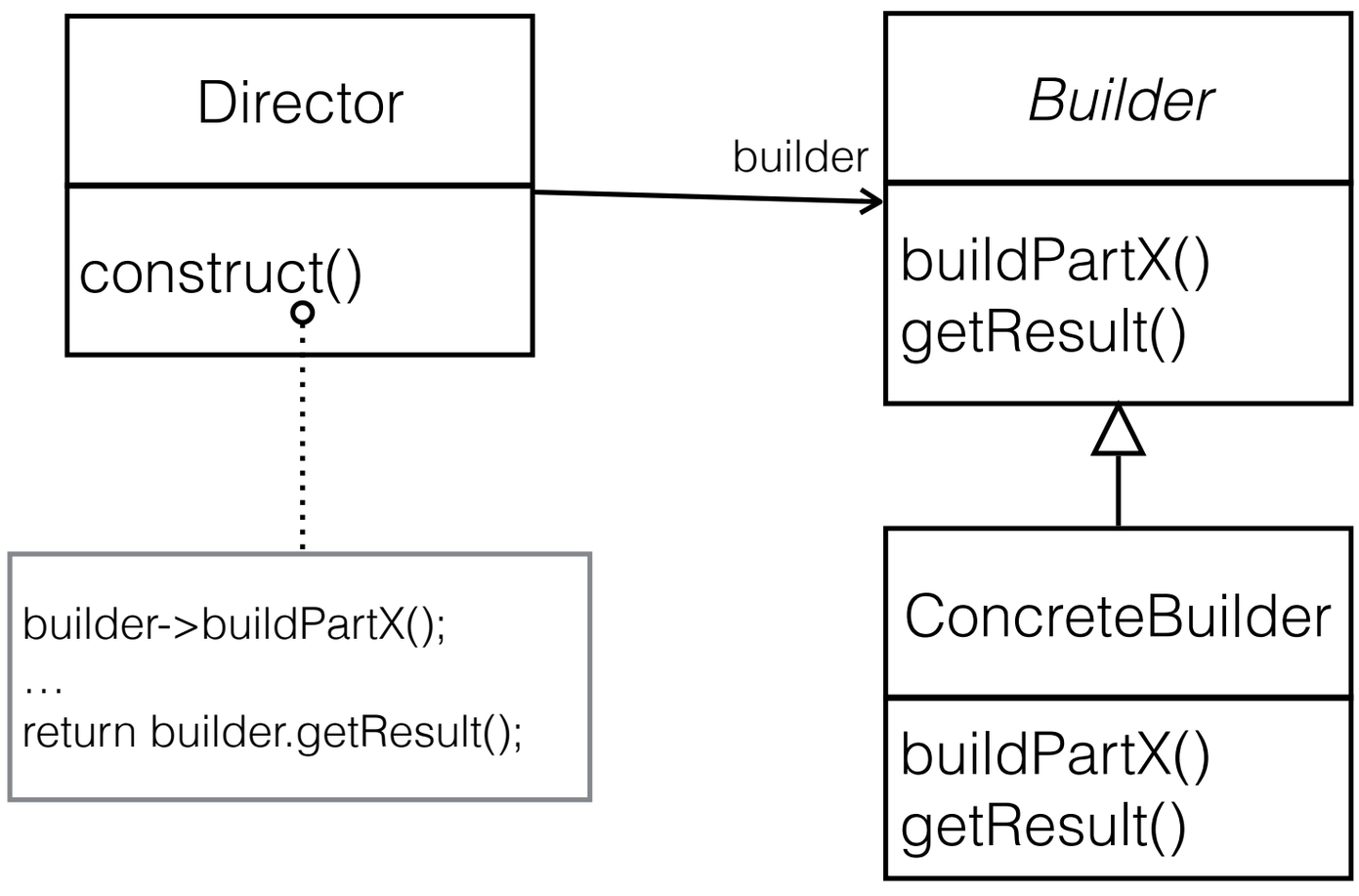
工厂方法
Factory Method

- 例子
  - javax.xml.bind.JAXBContext

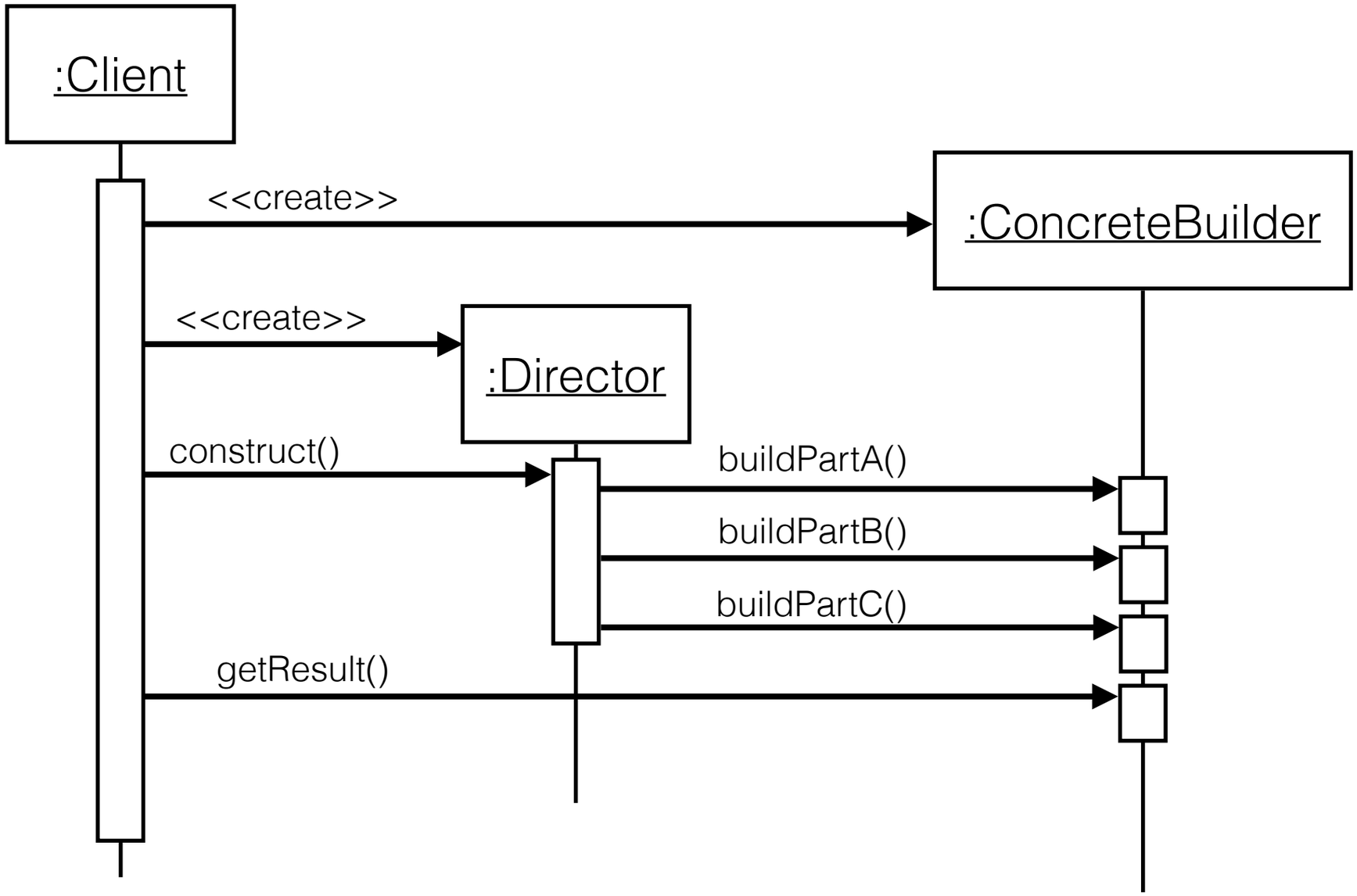
```
JAXBContext jc = JAXBContext.newInstance( "com.acme.foo:com.acme.bar" );
Unmarshaller u = jc.createUnmarshaller();
FooObject fooObj = (FooObject)u.unmarshal( new File( "foo.xml" ) );
BarObject barObj = (BarObject)u.unmarshal( new File( "bar.xml" ) );
```

建造者模式
Builder

- 意图:
  - 将复杂对象的构造过程与实现方法分离



建造者模式
Builder



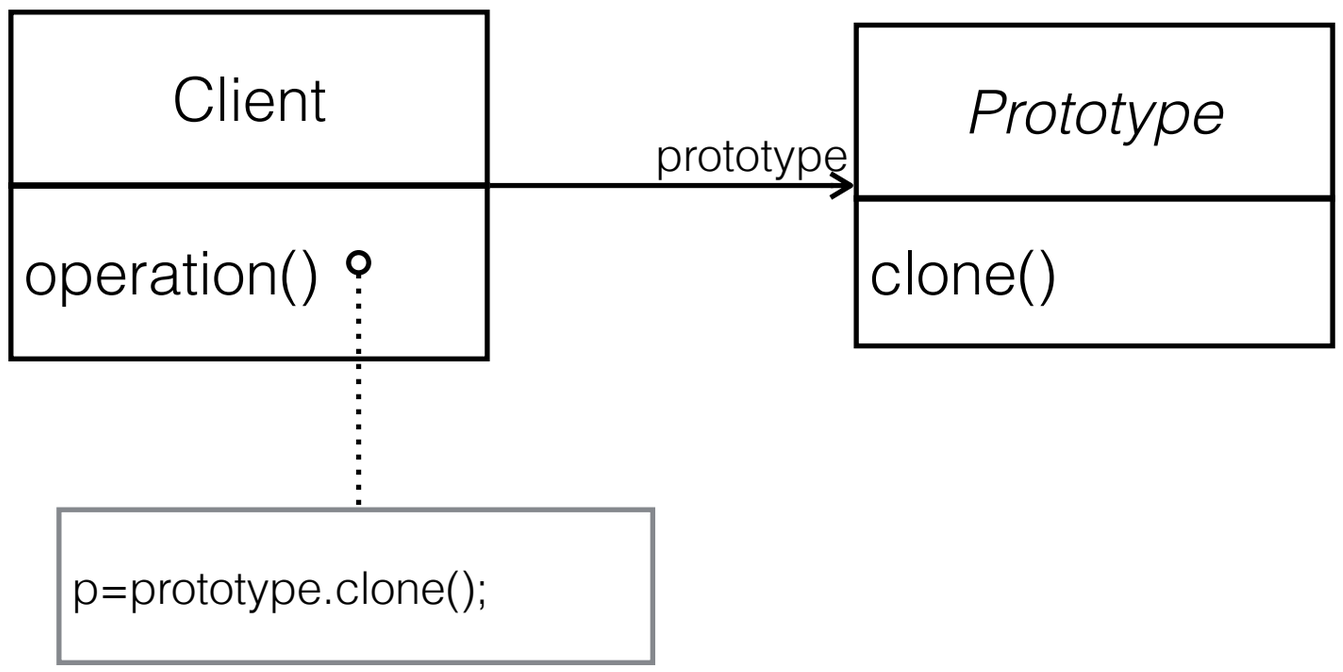
建造者模式
Builder

- 例子
  - java.lang.StringBuilder

```
StringBuilder builder = new StringBuilder();
builder.append(“str”);
builder.append(3);
return builder.toString();
```

原型模式
Prototype

- 意图
  - 使用原型刻画需要创建的对象，通过克隆的方式实现对象的创建



单件模式
Singleton

- 意图
  - 确保一个类只有一个实例