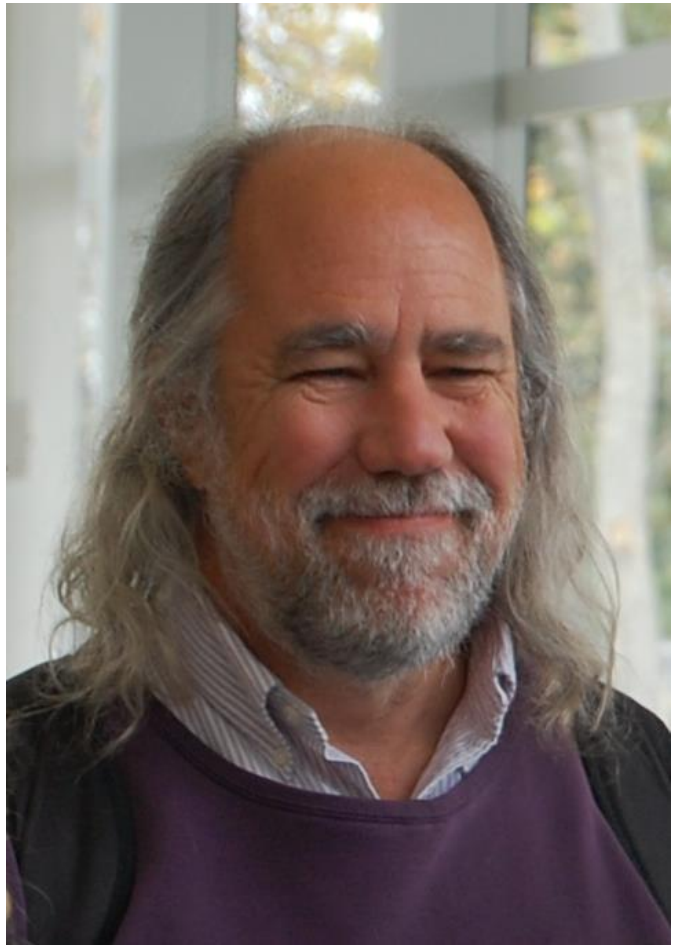
简单工厂

波波老师~研发总监/资深架构师

什么是架构

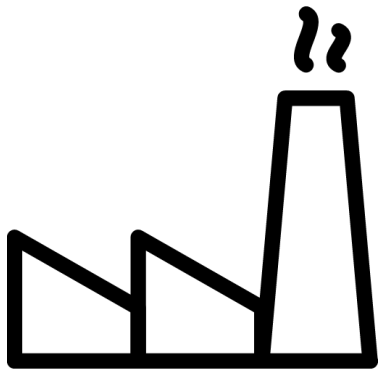


Grady Booch，UML创始人

- Architecture represents the significant design decisions that shape a system, where significant is measured by cost of change.
- 架构表示对一个系统的成型起关键作用的**设计决策**，架构定系统基本就成型了，这里的关键性可以由**变化的成本**来决定
- +**质量反馈的速度**

简单工厂定义

- 具有产品制造方法的工厂类，该方法能够根据不同的输入制造输出不同的产品



不用工厂案例

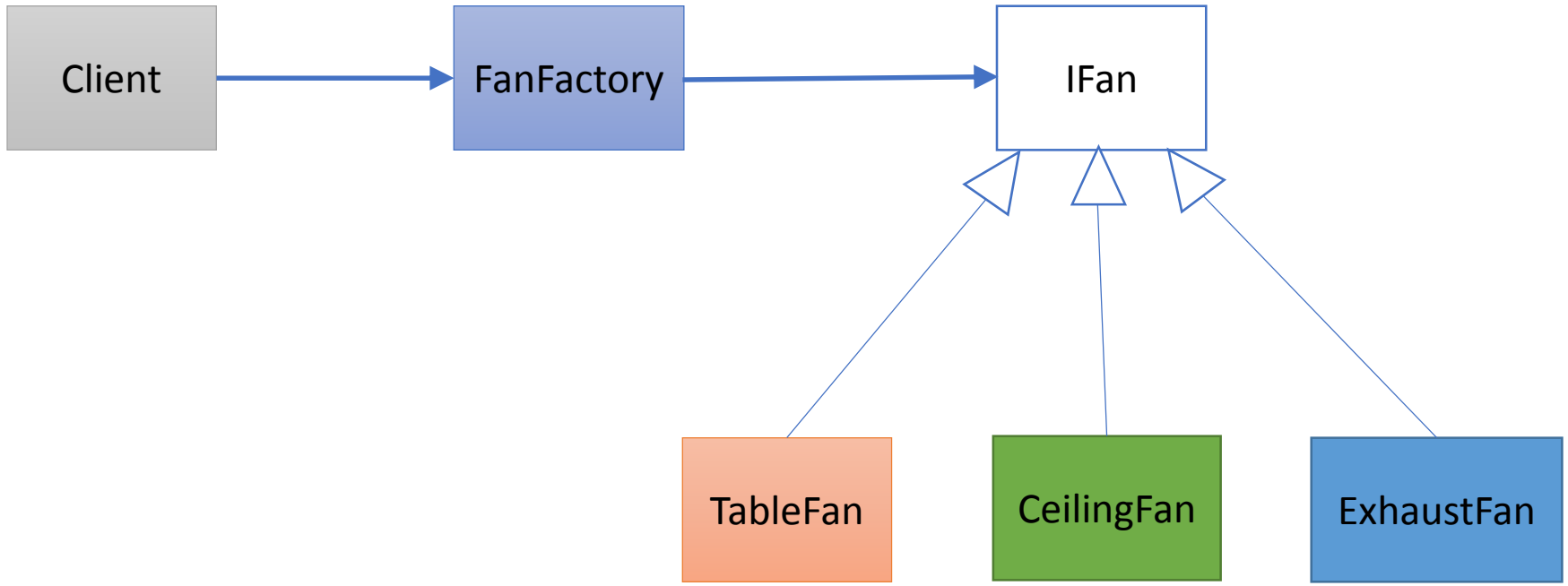
NoFactoryMain.java

```java
package io.spring2go.corespring.nofactory;

public class NoFactoryMain {

    public static void main(String[] args) {
        TableFan fan = new TableFan();
        fan.switchOn();
    }

}

class TableFan {
    public void switchOn() {
        System.out.println("The TableFan is switched on ...");
    }
}

```

问题

- 客户知道类的构造细节，耦合变更问题
  - 类名变更
  - 参数变更
- 难于优化对象创建流程
  - 缓存

简单工厂~关系图



实现~接口

```java
package io.spring2go.corespring.simplefactory;

public interface IFan {

    public void swithOn();

    public void switchOff();

}

```

```java
package io.spring2go.corespring.simplefactory;

public interface IFanFactory {
    IFan createFan(FanType type);
}

```

实现~产品类型

```java
package io.spring2go.corespring.simplefactory;

public enum FanType {

    TableFan, //台扇
    CeilingFan, //吊扇
    ExhaustFan //  排风扇
}
```

实现~产品类

```java
package io.spring2go.corespring.simplefactory;

public class TableFan implements IFan {

    @Override
    public void swithOn() {
        System.out.println("The TableFan is swithed on ...");
    }

    @Override
    public void switchOff() {
        System.out.println("The TableFan is swithed off ...");
    }

}
```

```java
package io.spring2go.corespring.simplefactory;

public class ExhaustFan implements IFan {

    @Override
    public void swithOn() {
        System.out.println("The ExhaustFan is swithed on ...");
    }

    @Override
    public void switchOff() {
        System.out.println("The ExhaustFan is swithed off ...");
    }
}
```

```java
package io.spring2go.corespring.simplefactory;

public class CeilingFan implements IFan {

    @Override
    public void swithOn() {
        System.out.println("The CeilingFan is swithed on ...");
    }

    @Override
    public void switchOff() {
        System.out.println("The CeilingFan is swithed off ...");
    }
}
```

实现~工厂

```java
package io.spring2go.corespring.simplefactory;

public class FanFactory implements IFanFactory {

    @Override
    public IFan createFan(FanType type) {
        switch (type) {
        case TableFan:
            return new TableFan();
        case CeilingFan:
            return new CeilingFan();
        case ExhaustFan:
            return new ExhaustFan();
        default:
            return new TableFan();
        }
    }

}
```

实现~客户端

```java
package io.spring2go.corespring.simplefactory;

// 客户端代码
public class SimpleFactoryMain {

    public static void main(String[] args) {
        IFanFactory simpleFactory = new FanFactory();
        // 使用简单工厂创建一个电扇
        IFan fan = simpleFactory.createFan(FanType.TableFan);
        fan.swithOn();
        fan.switchOff();
    }

}
```

好处

- 产品制造流程集中到工厂，客户只和工厂打交道
  - 易于变更
  - 易于优化

思考

- 简单工厂解决了一部分问题，它有引入什么问题？违反了那一条SOLID原理？

参考和预习

- Factory Patterns – Simple Factory Pattern(by Snesh Prajapati)
  - https://www.codeproject.com/Articles/1131770/Factory-Patterns-Simple-Factory-Pattern
- SOLID面向对象设计原理

代码

- https://github.com/spring2go/core-spring-patterns

波波微课

- 关于波波微课
  - 十多年研发经验老司机波波老师主导
  - 致力于使用新媒体技术提升学习成效
  - 主题面向Java，Spring，面向对象开发和微服务等
  - 关注工程师的成长
- 理念
  - 交互式的课程体验
  - 贴近一线企业实践
- 方法
  - 短视频，平均10分钟，最长不超过15分钟
  - 一个视频专注讲清楚一个主题
  - 50%原理+50%案例代码
  - 所有代码和ppt在github上可免费获得