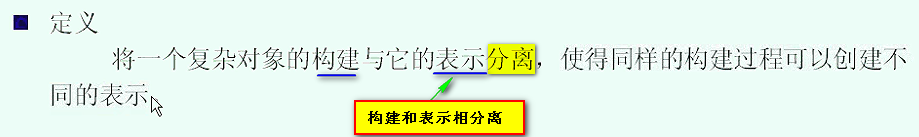
构造器模式的定义



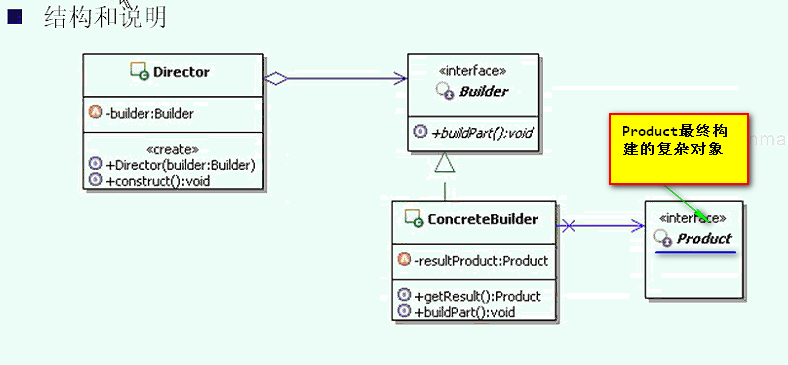
**场景理解**

相同的创建过程 创建出不同的表示 --- 根据这个定义 我们就知道 拼接字符串生成最终的字符串的时候 **就是一个相同的创建过程 append.append.append** 但是传递的可能是任意类型的数据 Object String StringBuilder Integer 等等

所以 StringBuilder是典型的构造器模式

**StringBuilder如何和构造器模式挂钩？**

1. 标准的构造器模式的uml类图



\*\*\*说明一下 getResult为什么出现在具体Builder的实现类里面 而没有出现在Builder接口中？ 因为如果暴露在接口中 用户就会都能访问到，那么 没有装配的过程 直接getResult()就会出问题。

所以 一般直接放到Builder的实现类中

\*\*查了一些资料 构造器模式在使用的时候 通常进行简化

因为Builder直接构造对象 所以 第一个 就是Builder接口省略

**Director和Client合二为一 ------ 从之前Client跟Director要Product【这个过程 就是 Director组织Builder构建出Product】 简化到 Client直接操作Builder来生成Product**

**这样 实际上角色就剩下了 Client, ConcreteBuilder和Product**

所以 依据构造器模式的应用背景 + 简化的uml类图

就知道了 我们直接使用的Client就是集原始Client和Director合并的client

编程写法

StringBuilder sBuilder = new StringBuilder();

String s =

**sBuilder**

**.append(“xxxx”)**

**.append(new Object())**

**.append(new ArrayList<String>())……**

**.toString();**

这个一个劲的append()就是Director中的骨架算法 ---- 简化之后再客户端调用

**最后的toString方法就是类图中ConcreteBuilder中的getResult()方法**

【原本append都应该在Director中操作，现在直接客户端操作】

在append中返回的都是 return this; ------- 这样 就是为了构造链式运算 ----- 不用一步步去写多个语句调用 ---- 简化Director骨架算法的写法

