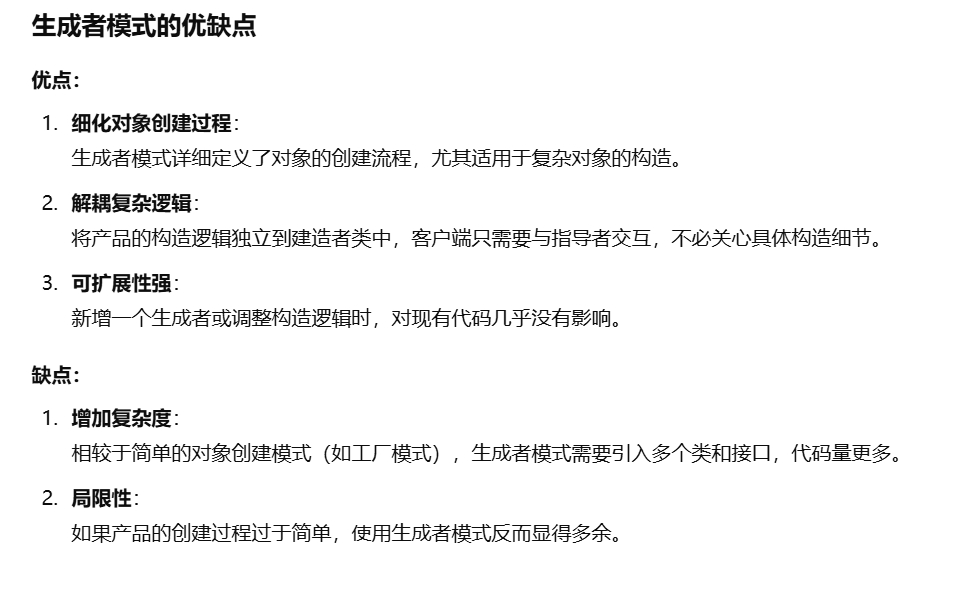
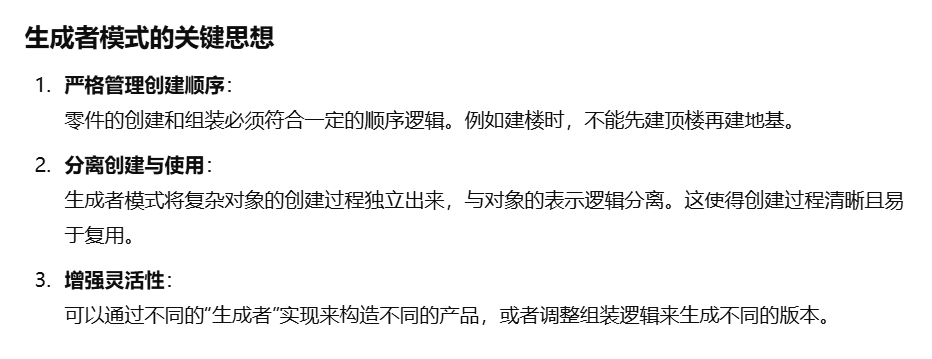
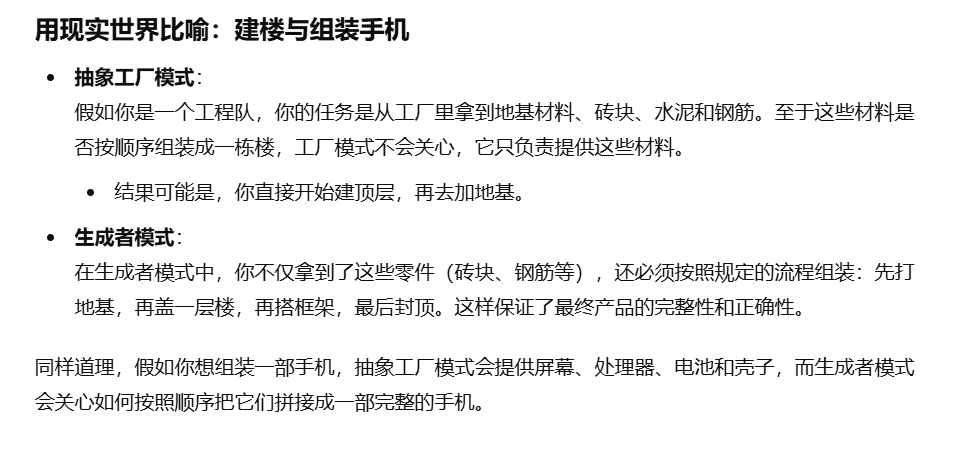
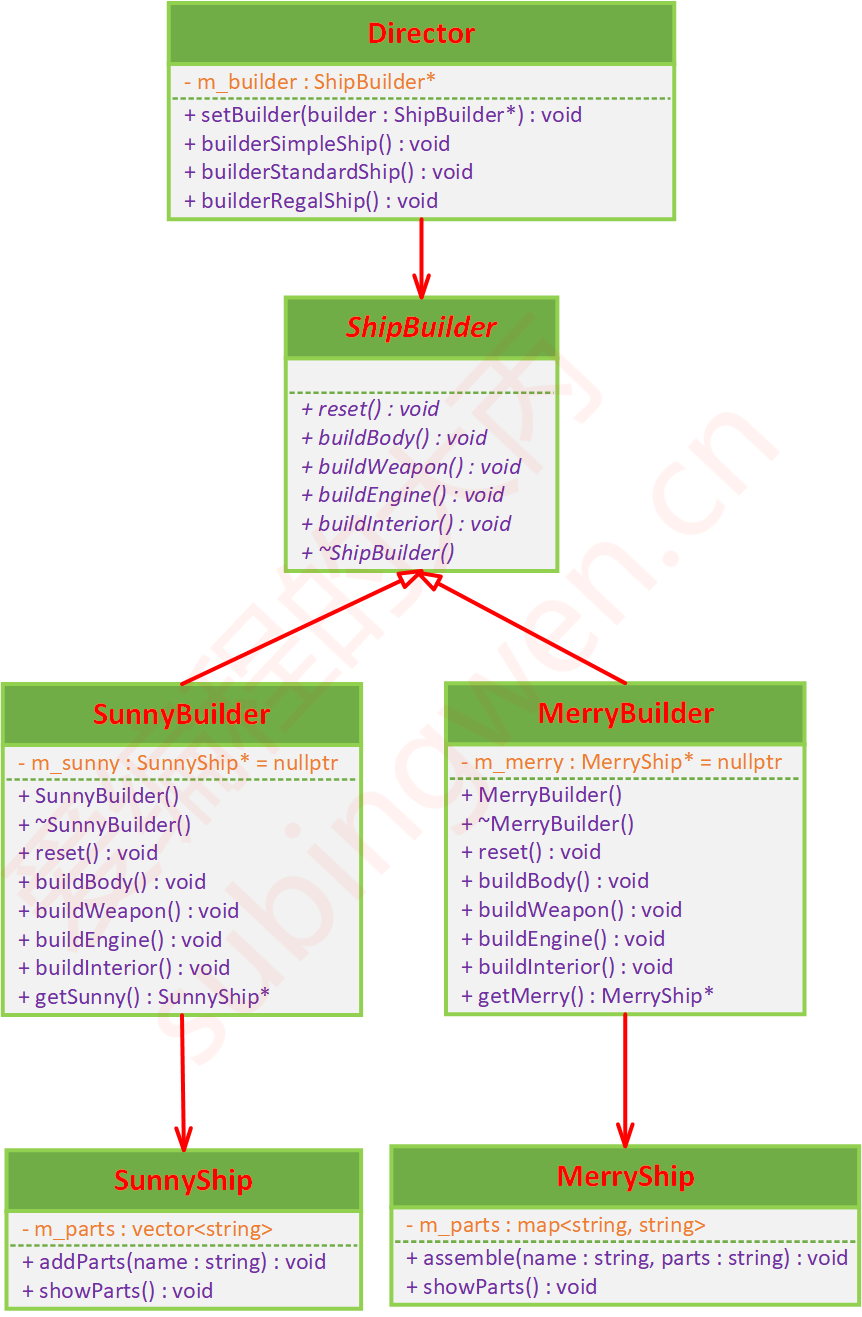
接下来我们进入到建造者模式中最麻烦的模式，生成者模式(**建造者模式**)。这种模式可以看做是对于抽象工厂模式的一个方面的改良。我们先回来看到抽象工厂模式，在抽象工厂模式中，一个工厂只需要去负责对应零件的获取即可，只需要获取到了这一系列一个具体产品需要的具体零件，我们就默认这些零件已经组合好成为了我们需要的一个具体产品。

看到这个过程，我们其实可以看到它其实省略了一个过程，就是零件是怎么组装形成我们最后的产品的。在现实世界中，我们仅仅获取到了一些零件是远远不够的，毕竟，就算给你一个手机所需要的所有零件，你也不知道这些零件要怎么组装才能形成我们需要的具体产品。这时就引入了一种新的创建型模式：生成者模式(**建造者模式**)。

既然它是创建型模式，那么它最后的目的就还是一个具体类对象的创建，但是正如我们前面所说，这个模式相对于抽象工厂模式来说进行了一些改进，其相对于不注重细节的抽象工厂模式进行了在零件组装上的细化。使得在对象创建时能够更加细致的对整个过程进行管理。就比如把，你要建造一栋楼，那么你至少需要地基才能建造第一层乃至更多层的楼吧。在抽象工厂模式中，其不关注你到底怎么建造的。哪怕你先提供了最高层的楼，再提供一个地基，它也是允许的，毕竟它忽略了一个组装的细节。

但是我们建造者模式不允许这样的创建。建造者模式将一个创建对象的过程进行细化和规定，其规定了一个具体产品的组成零件，以及这些个零件之间的组装细节。就比如，你需要先有地基，你才能建第一层楼，才能接着建造接下来的一系列楼层。





来简单看一个生成器模式下的UML类图。简单来分析下建造者类中的层次，首先我们应该看到的就是ShipBuilder这个类，可以看到，这个类是接下来的俩个具体类的子类。也可以这么说，这个是一个抽象工厂层，这个层次规定了一个产品所需要的步骤，我们所有具体的工厂类都应该是从这些个抽象工厂类中派生出来的。在这个类中，你可以看到其实相对于抽象工厂它是相似的。在演示中由于设计的复杂性这里不会让其去下辖抽象工厂层次中的零件类，但我们需要知道其实是有的。

在这些具体的建造者工厂类中，我们一般会存在一个成员来对我们建造的规则进行一个记录，这个的具体类型取决于实现。

说完了船的建造这一块，我们来说指令发出这一块的。在生成一个船的指令发出时，可能经过多层的处理。一般来说，为了好看，我们都会考虑增加一个管理者类，在用户层可以通过这个类去跟具体的工厂进行间接的交互，而不必直接去找到对应的工厂，其实就是加了一层中介。在这个管理者类中，需要这个管理者类可以管理我们需要的一系列产品的生产。所以这里就需要这个管理者类包含一个抽象工厂类的指针，令其可以指向所有的子类对象。再者，如果说一个产品中还有细分的品种，我们就需要让这个管理者类中去添加对应的生产函数，毕竟一个指针只能指向对应的对象，但是对应对象的生产是必须通过方法的(即函数)。也就是说，一个管理者类的方法数量，取决于我们一个产品的种类。

好，接下来就是最后一层，用户层，在我们这里中，由于存在管理者类，所以与用户层直接进行交互的将会是管理者类，用户层通过想管理者类提出要求，就比如要那艘船，要什么品质的船等相对来说更加细致的要求，就可以获取对应的产品了，不用关注底层的实现。

也就是说，生产者模式其实还可以看做是一种对于抽象工厂模式的扩充。其更加注重了各个零件组装的细节，能够相对于抽象工厂类去实现更加细致化的操作。

好了，我好像对建造者模式有一定的理解了，让我们启动。

首先，在建造者模式中，存在几个比较重要的层次需要我们了解。首先是用户层，这个其实对应的调用，我们不多赘述。

但是吧，我决定建造者模式从下往上去建模比较好，来吧。我们首先需要看到我们整个模式需要生产的东西，就是一个具体的类对象。用上面造船为例子吧。船的类就是整个模式要生产的东西的板子。对于这一层，可能是由一个抽象类来规定我们生产的产品的基本属性，然后通过一个具体的派生类来实现具体的一个船对象所需要的属性。当然，也有可能不是使用多态而是穷举所有的船类。这个取决于我们具体的实现。但是无论是哪种实现，我们需要注意的是，在这一层的船类中，对船的约束是相当宽泛的。就好比说，这个东西我给你标签为船，但是这个东西的具体并不会在这个标签上就注明，这里也是如此。

接下来需要看到建造者层，这是在整个层次中的底层。这一层中我们还是可以抽象出来俩层，一层是抽象建造者层，一层是具体建造者层。这个抽象建造者层规定了这一系列的建造者所要建造的物品的种类。就比如，你如果是一个船建造者，那么它要派生出来的类就是具体的船的建造者，但是，并不规定到底是哪一个船。

一般来说，在抽象建造者层中，一般只存在着一系列的建造方法，不会存在一个成员对象用于储存。这个其实是为了方便一个扩展。因为我们不确定在设计中我们的船类是否是使用多态的，如果你的具体船类是使用多态的，那么使用成员指针自然是一个极好的方法，但是，如果你是穷举出一系列的船的，你使用成员指针的话，你就必须包括所有的可能创建的船，这个其实不是很符合单一职责原则的。

所以我们可以看到，在建造者类中，我们特别需要注意的就是其所遵守的**单一职责原则**，这个原则使得建造者类在设计上是关注与船的功能的构建即可。其在设计中一般来说都是提供一系列的对应的要创建的对象的组装方法的。就比如组装船。在建造者类中，其可能存在的方法就包括武器，能源，装饰等等。但是，一般来说，在建造者类中，不会包括这些方法的组合。也就是说，建造者类负责提供一些(较大块的)零件，让其他模块来组合形成一个具体的类对象。

接下来我们需要来看到更接近用户层的类，**管理者层**。这个层在一些资料中说是可有可无的，但是吧，现实设计中一般都是包含这个类的。

这个类可以看做是对用户需求提出的层次和建造者层次的缓冲。管理者类一般是可以存在多个的，一个管理者类负责一类船的生产。

用现实生活来举例吧，一个船厂一般有自己生产的船的方向，但是并不固定。当一个公司甲方想要去委托一个船厂去生产一个具有特定功能的船的时候，其一般都不会直接去找到制造部门，而是去找到船厂的销售部门，向这个部门去阐述自己的需求，或者说选择自己需要的船型号。接下来这个销售部门会去解析这个需求，将这个需求解析成一步步可以落地的步骤，然后让这个船厂的制造部门去做。

在这个例子中，甲方扮演的角色就是用户层，销售部门扮演的就是管理者层，制造部门扮演的就是建造者层，而我们最后的需求产品，就是我们一个具体的船类的对象。

可以看出来，我们管理者层这里负责一个需求的解析，就比如甲方去A船厂提出我需要你们一个标准型号的船，那么这个管理者类就会对这个需求进行解析，就假设一个标准型号的船需要一个船身，一个发动机，一个武器这个配置，那么管理者层就会把这个配置发送给我们具体的建造者类中去，然后这个建造者类就会依照这个管理者类发来的需求按顺序去一个个实现到一个船对象中。当执行完后，一个标准A船就被制造出来了。

接下来，就需要船厂能够提供一个接口让用户能够提船。这个一般应该在制造者类中去实现，至于是在抽象类中就有还是每个类维护一个自己的方法取决于各自的实现。其实，这些个制造者类就好像一个公司一个部门的不同项目，其各自有着一套对接流程，甲方需要依照这些接口流程去获取自己需要的东西。

同时，我们可以考虑到如果不想要A船厂的船而想要B船厂的船。那我们需要怎么做，那我们需要去找到对应船厂的中介(管理者)，对应船厂需要特定的船产品(具体的产品类)，还需要对应船厂有自己的制造部门(建造者类)，这是最简单的配置了，这也是我们在建造者模式中想要进行扩展需要添加的几个方面。

我再来对制造者类来进行下分析。我们抽象这个建造者类为具体的生产部门。在这个部门中，其的目的是生成出一个管理者可能会要求生产的产品，也就是说，在制造者类中，应该存在一个可以用来储存当前生产的对象的地方，在类中就抽象为一个成员指针。这个可以玩笑为”新建文件夹”。在制造者类中，其一开始并不知道这个类将要被怎么个设计，但是这个制造者类中应该包含着对于这个新建文件夹的所有操作。

既然这个制造者类不能自己决定要怎么个制作方法，那么就需要外部来给他提出要求，在生产者模式中就是管理者类起到了这方面的作用。前面已经提到了，管理者类会将用户提出的需求转化为具体的组装步骤转发给制造者类去进行实现。

也就是说，管理者类会提供一个完整的设计流程给制造者，制造者类只需要按这个提供的流程去进行组装即可。

差不多了，那么接下来我们来讨论下管理者类的必要性。其实理论上管理者类的存在不是必须的，但是正如设计模式存在的意义一般，这个层次的存在也是为了方便我们的设计和使用。我们当然可以把这个管理者要要提供的组装流程直接封装在制造者类中。但是吧，这就像你一个甲方要去直接与对应乙方的技术部门进行交流一样。这通常是一份吃力不讨好的工作。为了更高效的交流，我们应该考虑这个公司中的对接部门，这个部门能够协调甲方的需求和自己方公司的实际能力，这种缓存能更好的提高效率。

也就是说，对于设计模式的学习，我们一定得结合实际来看，毕竟设计模式其实就是一种对于现实世界的抽象建模，其旨在解决现实生活中的问题，结合现实问题能让我们更好的理解设计模式。