设计原理

“恐龙”类和“玩家”类扩展了“演员”类。这是因为它们都有相同的名称，hitPoints字段等。 “恐龙”类定义了恐龙具有的常见属性，例如行为，foodLevel（指从“演员”继承的命中点）和性别。通过从“演员”类继承，“恐龙”和“玩家”类可以使用现有的字段和方法，从而避免了重复生成代码。如果需要，他们可以覆盖这些方法以创建其唯一属性。这种继承方式实现了“不要自己重复”（DRY）原则。

有3个类扩展了“恐龙”类，例如“剑龙”，“腕龙”和“异特龙”。由于不同类型的恐龙与“恐龙”密切相关，因此创建抽象类“恐龙”比创建界面“恐龙”更好。

“恐龙”类可确保不同类型的恐龙具有上述共同的属性。这减少了重复的代码并实现了DRY原理。这样，这3个子类将分别定义其属性，同时仍被视为“恐龙”，并允许创建不同类型的恐龙。此外，将来在不更改代码主体结构的情况下（很昂贵）向“恐龙”添加功能更加容易，这提高了代码的易懂性和可维护性。

为了减少依赖性，将“恐龙”类及其子类（“恐龙”，“剑龙”，“腕龙”和“异龙”）放入名为game.dinosaur的程序包中。这种打包紧密相关的类的方式为这些类提供了良好的封装边界，并提高了代码的可读性和可理解性。

有一些serval类实现接口<Behaviour>。 <行为>

界面会创建“动作”的不同实例，并允许执行“动作”。

这些类实现<Behaviour>接口与“ Dinosaur”类（的子类）交互，“ Dinosaur”具有不同的<Behaviour>作为字段存储在其中。 “恐龙”目前具有以下行为：

“ wanderBehaviour”，返回一个移动动作，允许“ Dinosaur”漫游到随机位置；

“ TrackFoodBehavior”为“ Dinosaur”创建移动动作，使其靠近目标食物一步（此处的食物是指从“ Tree”或“ Bush”生成的“ Fruit”取决于不同类型的恐龙） ;

当恐龙试图寻找具有不同性别的相同物种的伴侣并使其目标伴侣靠近时，将调用“ TrackSpouseBehavior”类。

“跟随行为”允许异特龙靠近目标剑龙进行狩猎和攻击。

同样，将创建一个名为game.behavior（<Behavior>，“ WanderBehavior”，

“ TrackFoodBehavior”，“ TrackSpouseBehavior”和“ FollowBehavior”）保存<Behavior>及其实现DRY和ReD原理的类。

“水果”，“素食餐盒”，“ CarnivoreMealKit”，“蛋”（“ StegosaurusEgg”，“腕龙”，“异龙”）和“尸体”是应用DRY原理的“便携式项目”的子类。原因是它们都可以被恐龙吃掉以增加食物水平，并且可以被拾取并放入玩家的库存中。 “ Item”类是“ PortableItem”类的父级。结果，上述子类继承了“ PortableItem”，而子类也继承自“ Item”类，这意味着它们可以具有不同的功能，例如被不同类型的恐龙吞噬和寿命。通过重新使用预定义的代码，继承可以节省大量工作。

可能会认为“ Corpse”类太大，无法容纳在玩家的广告资源中，因此不应携带。在这种情况下，假定“尸体”是便携式的。否则，“ Corpse”应该扩展“ Item”类，因为“ Item”类具有布尔型可移植字段，通过将其设置为false，“ Corpse”将被识别为不可携带的“ Item”。

“ StegosaurusEgg”，“ BrachiosaurusEgg”和“ AllosaurusEgg”扩展了抽象类“ Egg”。恐龙蛋都具有孵化时间，并且在孵化时间之后会属于“恐龙”类型。在这里使用抽象类“ Egg”的原因是通过将相关类组合在一起来提高可读性和可理解性。此外，通过使用“ Egg”类对恐龙蛋进行分组，可能会出现许多“ Egg”实例，从而使整个游戏保持一致，从而避免了某些可能在整个编码过程中出现的错误代码。

创建软件包（game.portableItem）来容纳那些“ PortableItem”（“水果”，“ VegetarianMealKit”，“ CarnivoreMealKit”，“ Corpse”和“ Egg”）。在此程序包中，有一个game.portableItem.egg程序包，其中包括“ Egg”，“ StegosaurusEgg”，“ BrachiosaurusEgg”和“ AllosaurusEgg”类。这种创建软件包的方法符合ReD原则。

此外，还会创建一个包（game.ground）来保存与“地”相关的类，包括“土”，“灌木”，“树”和“自动贩卖机”。

“污垢”，“灌木”，“树”和“自动贩卖机”类扩展了抽象类“地面”。 “地面”类提供了不同的方法，例如检查“演员”是否可以站立在其上，并允许子类定义其功能。然后，这些子类可以通过应用多态性覆盖“地面”类中的方法来改变行为，从而实现DRY原理。

“污垢”对象有机会长大为“灌木”对象。 “树”和“布什”都负责种植水果（“水果”的一个实例）。在整个游戏中，“演员”与“地”的那些子类进行交互。这些子类在“演员”站在旁边时提供了不同的允许动作。例如，

播放器可以站在与“树”或“灌木”对象相同的位置，并在其中搜索

看看它是否能结出果实。

“自动售货机”维护着一个可出售的“ PortableItem”的列表（ArrayList）（“水果”，“素食餐包”，“食肉动物餐包”，“蛋”（“ StegosaurusEgg”，“ BrachiosaurusEgg”，“ AllosaurusEgg”）稍后通过“ SellAction”类允许玩家购买这些物品。

“ LaserGun”继承自发动机设计中的“ WeaponItem”，这会损坏“剑龙”。

上面说明的“ PortableItem”通过“动作”与“演员”互动。 “操作”类具有服务子类，即“ BreedAction”，“ EatAction”，“ FeedAction”，“ AttackAction”，“ FireAction”，“ PickUpItemAction”，“ HaverestAction”和“ SellAction”。

通过“ BreedAction”，“恐龙”可以与另一个“物种”相同但性别不同的“恐龙”繁殖。

“饮食行动”负责“演员”在

“ PortableItem”。

通过“ FeedAction”，玩家（“演员”）可以将“ PortableItem”提供给“恐龙”。 “攻击行动”允许“异特龙”攻击原始设计中的“剑龙”。 “ FireAction”允许玩家使用“ WeaponItem”向恐龙射击。

“ PickUpItemAction”采用代码库的原始设计，允许玩家从“地面”拾取物品（主要是“水果”）。

“收割动作”是玩家在当前位置搜索“灌木”或“树”以查看是否有“水果”的时候。

“ SellAction”用于“ VendingMachine”，以出售“ PortableItem”和

“ WeaponItem”（“ LaserGun”）。

以上课程都集中在“动作”课程中。为了ReD的目的，创建了一个名为game.action的包，其中包括上述子类，“ BreedAction”，“ EatAction，“ AttackAction”，“ PickUpItemAction”，“ HarvestAction”，“ SellAction”，“ FeedAction” “ FireAction”。

还有一个“ EcoPoint”类，可跟踪游戏中的环保积分，并在可能的情况下负责增加或降低环保积分。它将具有一个私有整数属性，该属性存储生态点，并具有获取器和设置器来修改该值。如果不使用设置器，外部世界就无法简单地修改环保积分值。这样可以避免由于整数的不可变属性而导致的隐私泄漏。