题主说的<? extends T>和<? super T>是Java泛型中的***“通配符（Wildcards）”***和***“边界（Bounds）”***的概念。

* <? extends T>：是指 **“上界通配符（Upper Bounds Wildcards）”**
* <? super T>：是指 **“下界通配符（Lower Bounds Wildcards）”**

**1. 为什么要用通配符和边界？**

使用泛型的过程中，经常出现一种很别扭的情况。比如按照题主的例子，我们有*Fruit*类，和它的派生类*Apple*类。

**class** **Fruit** **{}**

**class** **Apple** **extends** Fruit **{}**

然后有一个最简单的容器：*Plate*类。盘子里可以放一个泛型的“*东西*”。我们可以对这个东西做最简单的“*放*”和“*取*”的动作：*set( )*和*get( )*方法。

**class** **Plate<**T**>{**

**private** T item**;**

**public** **Plate(**T t**){**item**=**t**;}**

**public** **void** **set(**T t**){**item**=**t**;}**

**public** T **get(){return** item**;}**

**}**

现在我定义一个“*水果盘子*”，逻辑上水果盘子当然可以装苹果。

Plate**<**Fruit**>** p**=new** Plate**<**Apple**>(new** Apple**());**

但实际上Java编译器不允许这个操作。会报错，“*装苹果的盘子*”无法转换成“*装水果的盘子*”。

error: incompatible types: Plate<Apple> cannot be converted to Plate<Fruit>

所以我的尴尬症就犯了。实际上，编译器脑袋里认定的逻辑是这样的：

* 苹果 ***IS-A*** 水果
* 装苹果的盘子 ***NOT-IS-A*** 装水果的盘子

所以，就算容器里装的东西之间有继承关系，但容器之间是没有继承关系的。所以我们不可以把*Plate<Apple>*的引用传递给*Plate<Fruit>*。

为了让泛型用起来更舒服，Sun的大脑袋们就想出了<? extends T>和<? super T>的办法，来让”*水果盘子*“和”*苹果盘子*“之间发生关系。

**2. 什么是上界？**

下面代码就是**“上界通配符（Upper Bounds Wildcards）”**：

Plate**<**？ **extends** Fruit**>**

翻译成人话就是：**一个能放水果以及一切是水果派生类的盘子**。再直白点就是：**啥水果都能放的盘子。**这和我们人类的逻辑就比较接近了。Plate<？ extends Fruit>和Plate<Apple>最大的区别就是：**Plate<？ extends Fruit>是Plate<Fruit>以及Plate<Apple>的基类。**直接的好处就是，我们可以用“*苹果盘子*”给“*水果盘子*”赋值了。

Plate**<?** **extends** Fruit**>** p**=new** Plate**<**Apple**>(new** Apple**());**

如果把Fruit和Apple的例子再扩展一下，食物分成水果和肉类，水果有苹果和香蕉，肉类有猪肉和牛肉，苹果还有两种青苹果和红苹果。

*//Lev 1*

**class** **Food{}**

*//Lev 2*

**class** **Fruit** **extends** Food**{}**

**class** **Meat** **extends** Food**{}**

*//Lev 3*

**class** **Apple** **extends** Fruit**{}**

**class** **Banana** **extends** Fruit**{}**

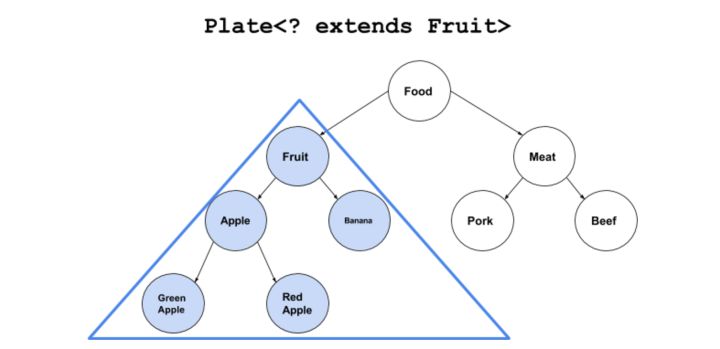
**class** **Pork** **extends** Meat**{}**

**class** **Beef** **extends** Meat**{}**

*//Lev 4*

**class** **RedApple** **extends** Apple**{}**

**class** **GreenApple** **extends** Apple**{}**

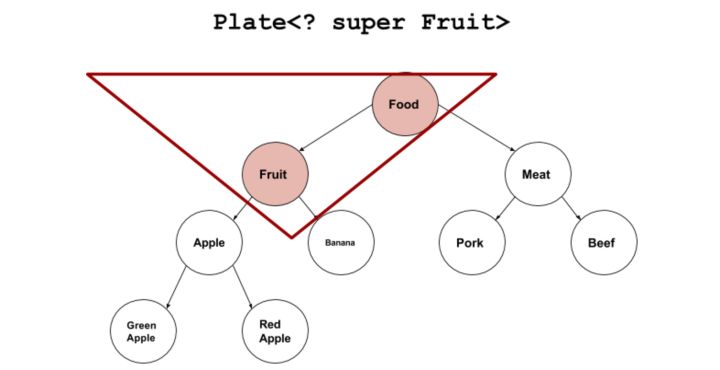
在这个体系中，上界通配符 “Plate<？ extends Fruit>” 覆盖下图中蓝色的区域。  


**3. 什么是下界？**

相对应的，**“下界通配符（Lower Bounds Wildcards）”**：

Plate**<**？ **super** Fruit**>**

表达的就是相反的概念：**一个能放水果以及一切是水果基类的盘子**。**Plate<？ super Fruit>是Plate<Fruit>的基类，但不是Plate<Apple>的基类。**对应刚才那个例子，Plate<？ super Fruit>覆盖下图中红色的区域。



**4. 上下界通配符的副作用**

边界让Java不同泛型之间的转换更容易了。但不要忘记，这样的转换也有一定的副作用。那就是容器的部分功能可能失效。

还是以刚才的Plate为例。我们可以对盘子做两件事，往盘子里set( )新东西，以及从盘子里get( )东西。

**class** **Plate<**T**>{**

**private** T item**;**

**public** **Plate(**T t**){**item**=**t**;}**

**public** **void** **set(**T t**){**item**=**t**;}**

**public** T **get(){return** item**;}**

**}**

**4.1 上界<? extends T>不能往里存，只能往外取**

**<? extends Fruit>会使往盘子里放东西的set( )方法失效。但取东西get( )方法还有效。**比如下面例子里两个set()方法，插入Apple和Fruit都报错。

Plate**<?** **extends** Fruit**>** p**=new** Plate**<**Apple**>(new** Apple**());**

// Plate**<**Fruit**>表示装水果的盘子，**Plate**<?** **extends** Fruit**>表示装水果或者装某种具体水果的盘子，但不知道是哪种水果**

*//不能存入任何元素*

p**.**set**(new** Fruit**());**

*//Error，假如*Plate**<?** **extends** Fruit**>** p**=new** Plate**<**Apple**>(new** Apple**()); 则这个盘子必须是装苹果的，不能装new** Fruit**()**

p**.**set**(new** Apple**());**

*//Error，假如*Plate**<?** **extends** Fruit**>** p**=new** Plate**< Banana** **>(new** **Banana** **()); 则这盘子必须是装香蕉的，不能装new** Apple**()**

*//读取出来的东西只能存放在Fruit或它的基类里。*

Fruit newFruit1**=**p**.**get**(); //** Fruit newFruit1可以指向所有Fruit、Fruit派生类

Object newFruit2**=**p**.**get**(); //** Object newFruit可以指向所有Fruit、Fruit派生类

Apple newFruit3**=**p**.**get**();** *//Error，*Apple newFruit3不能指向所有Fruit、Fruit派生类

原因是编译器只知道容器内是Fruit或者它的派生类，但具体是什么类型不知道。可能是Fruit？可能是Apple？也可能是Banana，RedApple，GreenApple？编译器在看到后面用Plate<Apple>赋值以后，盘子里没有被标上有“苹果”。而是标上一个占位符：**CAP#1**，来表示捕获一个Fruit或Fruit的子类，具体是什么类不知道，代号CAP#1。然后无论是想往里插入Apple或者Meat或者Fruit编译器都不知道能不能和这个CAP#1匹配，所以就都不允许。

所以通配符<?>和类型参数<T>的区别就在于，对编译器来说**所有的T都代表同一种类型**。比如下面这个泛型方法里，三个T都指代同一个类型，要么都是String，要么都是Integer。

**public** **<**T**>** List**<**T**>** **fill(**T**...** t**);**

但通配符<?>没有这种约束，Plate<?>单纯的就表示：**盘子里放了一个东西，是什么我不知道**。

所以题主问题里的错误就在这里，Plate<？ extends Fruit>里什么都放不进去。

**4.2 下界<? super T>不影响往里存，但往外取只能放在Object对象里**

**使用下界<? super Fruit>会使从盘子里取东西的get( )方法部分失效，只能存放到Object对象里。set( )方法正常。**

Plate**<?** **super** Fruit**>** p**=new** Plate**<**Fruit**>(new** Fruit**());**

**//** Plate**<?** **super** Fruit**> 装水果、水果父类的盘子，比如可以装**Fruit、**Food、Object**

*//存入元素正常*

p**.**set**(new** Fruit**()); //** Fruit属于Fruit、**Food、Object**

p**.**set**(new** Apple**()); //** Apple属于Fruit、**Food、Object**

*//读取出来的东西只能存放在Object类里。*

Apple newFruit3**=**p**.**get**();** *//Error*

Fruit newFruit1**=**p**.**get**();** *//Error*

Object newFruit2**=**p**.**get**();**

//从p中取出来的可能是Fruit、**Food、Object，为了匹配所有可能，p.get()的类型只能是Object**

因为下界规定了元素的最小粒度的下限，实际上是放松了容器元素的类型控制。既然元素是Fruit的基类，那往里存粒度比Fruit小的都可以。但往外读取元素就费劲了，只有所有类的基类Object对象才能装下。但这样的话，元素的类型信息就全部丢失。

**5. PECS原则**

最后看一下什么是**PECS（Producer Extends Consumer Super）原则，**已经很好理解了：

1. **频繁往外读取内容的，适合用上界Extends。**
2. **经常往里插入的，适合用下界Super。**