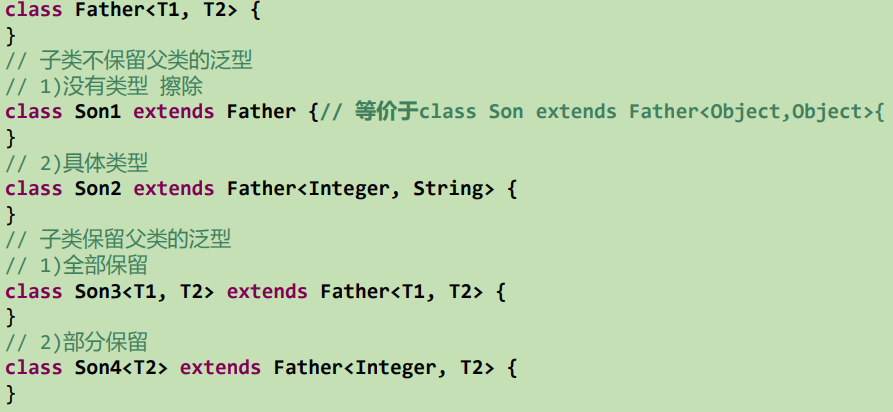
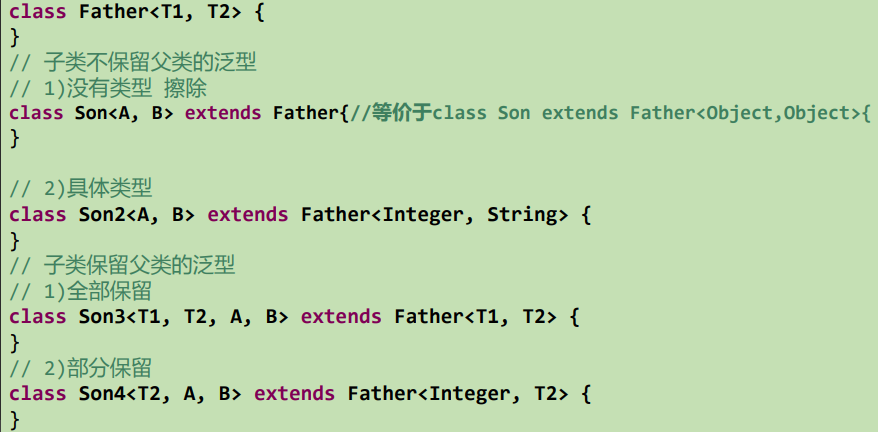
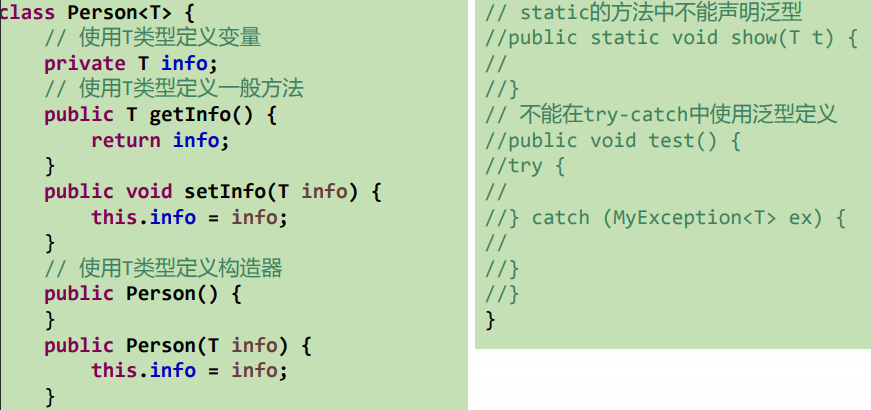
子类泛型**不可以**转换为父类泛型

父类泛型不可以转换为子类泛型

如果希望只取出，不插入，就使用? extends Hero  
如果希望只插入，不取出，就使用? super Hero  
如果希望，又能插入，又能取出，就不要用通配符？

1. 自定义泛泛型类：
   1. 声明：Interface XXX<T> 和 class XXX<T>
   2. 实例化：在类名后面<>指定类型
   3. 注意：T只能是类，不能用基本数据类型填充，但可以使用包装类
      1. <>尖括号内可以有多个参数，<X,A,S>
      2. 泛型类的构造器public xxx<E> {} ， 需添加<>
      3. 实例化后，操作泛型位置的结构必须与指定的泛型一致
      4. 泛型不同的引用不能相互赋值
      5. 泛型不指定将被擦除，按照Object处理，但不等价于Objcet
      6. 泛型结构是一个接口或抽象类，则不可以创建泛型类对象
   4. 核心：把内容限制为一个特定的数据类型
   5. 在类或接口上声明泛型：
      1. 在本类中代表某种类型，可以作为非静态属性的类型，参数类型，返回值类型
      2. 在静态方法中不能使用类的泛型
      3. 异常类不能泛型
      4. 不能创建泛型数组 new T[]
   6. 子类不保留父类的泛型：按需要实现
      1. 没有类型，擦除
      2. 具体类型
   7. 子类保留父类的泛型：泛型子类
      1. 全部保留
      2. 部份保留
   8. 
   9. 
   10. 
2. 泛型方法：
   1. 方法可以被泛型化，参数的类型就是传入的数据的类型
   2. 格式：【访问权限】 <泛型> 返回类型 方法名（【泛型标识 参数名称】） 抛出异常
   3. 静态方法中的泛型同理 在访问权限后加上static
3. 泛型在继承上的体现：
   1. A←B，G<T>，G<B>与G<A>并列
4. 通配符：？
   1. 例如List<?>是List<String>等各种泛型List的父类
   2. 读取：
      1. List<？> 的对象中元素，被读取时永远安全，因为不管list的真是类型是什么，他都包含的是Object
   3. 写入：不能写入，应为知道list的元素是什么类型，但是null是例外
   4. 注意：
      1. 不能在返回值类型使用
      2. 不能再泛型类声明上使用
      3. 不能再创建对象时使用
5. 有限制的通配符：
   1. ？extends xxx
      1. 允许泛型为xxx或xxx的子类
   2. ？super xxx
      1. 运行泛型为xxx或xxx的父类