**泛型**

**类型擦除：**泛型信息只存在于代码编译阶段，在进⼊ JVM 之前，与泛型相关的信息会被擦除掉，即泛型类与普通类没有区别

泛型类被类型擦除的时候，之前泛型类中的类型参数如果没有指定上限， 则会被转译成普通的 Object 类型，如果指定了上限如 <T extends String> 则类型参数就被替换成类型上限。



**泛型的限制：**

1. 类型参数不支持基本数据类型

2. 无法实例化泛型类型的对象

3. 无法使用泛型类型声明静态的属性

4. 无法使用 instanceof 判断带类型参数的泛型类型（instance发生在运行时期）

5. 无法创建泛型类数组

7. 泛型无法重载

**1.泛型类：**在类定义的时候并不会设置类中的属性或方法中的参数的具体类型，而是在类使用时再进行定义（编译时期的一种机制）

class MyClass<T> {
T value1;
}

尖括号 <> 中的 T 被称作是类型参数，用于指代任何类型。

泛型类作用：

可以自动进行类型检查

可以自动进行类型转换

改变了向下转型的需求

**2.泛型方法：**类型参数写在返回值前面，<T> 中的**T** 被称为参数化类型，它不是运行时真正的参数，可以作为方法返回值。

class MyClass{
public <T> T testMethod(T t) {
System.out.println(t);
}
}

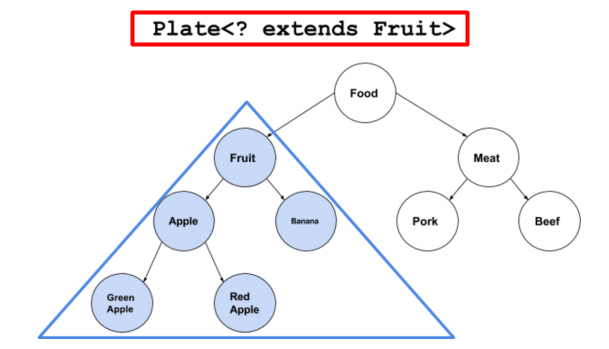
**泛型避免了 ClassCastException 问题，但是产生了新的情况：参数的统⼀问题。**

**3.泛型接口：子类在实现接口时可以继续使用泛型，也可以明确类型**

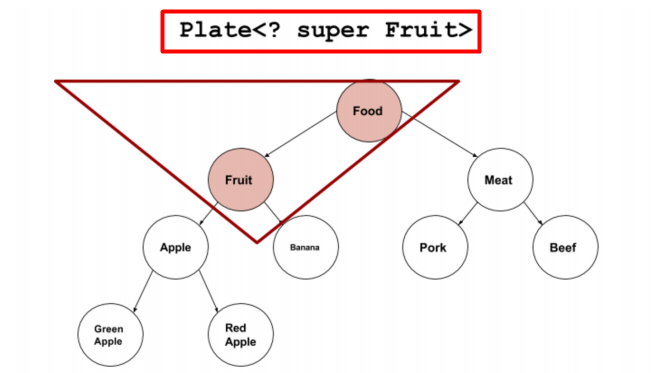
interface IMessage<T> { // 在接⼝上定义了泛型
public void print(T t) ;
}

**通配符（？）：**可以接收所有的泛型类型，但是不能随意修改

**? extends 类：设置泛型上限，用在声明，不能修改**



**? super 类：设置泛型下限，用在方法参数，可以修改**



**可变参数(...)**

public [static] [final] 返回值 ⽅法名称([参数类型 参数名称][参数类型 ... 参数名称])
{}

如果要传递多类参数，可变参数⼀定放在最后，并且只能设置⼀个可变参数

**静态导入：**从JDK1.5开始，如果类中房法全是static方法，可以直接把这个类导⼊进来，被主方法直接调用。