第一组：

问题一：hashcode和equals的具体区别和分别的应用场景，具体是指两者是在哪些情况下等价或者可以相互替代，什么时候相互替代会引起一些错误和问题？

https://www.cnblogs.com/gieriy/p/15638293.html
https://zhuanlan.zhihu.com/p/347342971

实际项目中一般只使用equals。用来做变量之间的比较。

问题二：枚举这个手段在程序中有什么好处吗?如何利用enum枚举类来和其他类中的方法以及成员函数相互组合来进行开发，比如说枚举类在各级类之间所应处于的位置和具体的一些使用方。

枚举的好处：
1，将无意义的数字变量，变成了有意义的类型；
2，解决意义不明确的问题。比如调试程序时，本来想输出性别男，结果输出了个0，不是自己写的完全不知道其意义，用枚举类就完美解决了
3，代码更优雅。一个大一些的程序里面，可能要用到成百上千的静态常量，如果全写在一个文件里面，容易造成命名混淆，程序读起来也比较麻烦。
4，定义自己的类型。如完全可以用enum XXX来包含枚举类型，可以避免错误的参数输入。

https://blog.csdn.net/Kisushotto/article/details/123225096

问题三：书上对于Java访问权限一般分为四种，若不加任何权限控制符表示的是该类或方法在包内和类内可见，这种写法是规范的嘛，我们写程序时需要注意去避免吗？

Java的四种访问权限：

(1).public：最大访问控制权限，对所有的类都可见。
(2).protect：修饰的，在类内部、同一个包、子类中能访问
(3).default：包访问权限，即同一个包中的类可以可见。默认不显式指定访问控制权限时就是default包访问控制权限。
(4).private：最严格的访问控制权限，仅该类本身可见。

实际写程序时，尽量清晰的加上访问权限的修饰符，提高程序可读性。

问题四：如何通过构造函数去输入一个 Date 类时间，比如我初始化一个手机实例将其出厂时间设定在 2022-1-2，具体的格式和书写方式是怎么样的？

构造一个date对象并对其进行初始化以反映当前时间
java.util.Date date = new java.util.Date();

构造一个date对象，并根据相对于gmt 1970年1月1日00:00:00的毫秒数对其进行初始化
date(long date)

接受一个时间字符串，设定某个时间，jdk1.1后不推荐使用了。
java.util.Date date = new java.util.Date("2016-10-26");

(int year, int month, int date)，(int year,int month, int date, int hrs, int min)等。在使用上面两个构造函数的时候 year 参数需要理想年-1900，例如2016年，作为参数用的话需要先减去1900年，也就是：
java.util.Date date = new java.util.Date (116,10,26);

第二组：

组内讨论与总结：

本周学习内容主要有：

1、继承
继承通俗说即一个子类 is-a 父类的一种，特点有：子类可以扩展父类的功能,但不能改变父类原有的功能，即里氏替换原则(也是对扩展开放的体现)。在对类加载上，其类加载器

遵从双亲委派机制，即当一个类加载器收到了类加载的请求的时候，他不会直接去加载指定的类，而是把这个请求委托给自己的父加载器去加载。只有父加载器无法加载这个类的时候，才会由当前这个加载器来负责类的加载。这些方式都是为了系统的安全考量。

在子类中可以使用 super，显式的调用父类中的方法。

2、多态扩展

方法的覆盖（又称重写）是指，子类在实现父类功能的时候，重新实现父类的方法，其特点为，参数列表，返回值等必须与父类相同且权限访问修饰符必须＞父类。且静态方法不能被重写。

方法的重载为参数类型，个数，顺序有不同。

3、继承修饰符及 final

继承可以用 protected 修饰即包可见，且子类可见。

final 关键字可以用来修饰一个类，一个方法，一个局部变量和一个成员变量。

使用 final 修饰的类不能有子类，方法不能被覆盖，其修饰的基本变量不会变，对于引用变量，其引用地址不会变，但内容可以改变。

4、hashcode,equals,toString 方法

皆为 Object 方法

hashcode 方法：对值(例如内存地址)进行计算，得到哈希码。

equals 方法：Obj 底层与==相同，但 String 重写了 Obj 的 equals 方法，会对引用对象的地址值进行计算比较。

toString 方法：返回一个文本表示的字符串，在 Obj 中该方法返回类名+@+哈希值。

5、接口

Interface.接口中的方法隐式的为 public 以及 abstract。这意味着实现该接口的类必须重写方法。接口的方法没有成员体，没有构造方法，若其有变量必须为常量。与抽象类的不同：

区别点	抽象类	接口
定义组成	包含抽象方法的类 具体方法 变量 构造方法	抽象方法和常量
使用	子类继承抽象类	实现类实现接口
关系	抽象类实现接口	接口不能继承抽象类，可以继承多个接口
对象	通过抽象类的多态性产生实例对象	无
局限	只能单继承	没有

本周问题：

Method threw 'java.lang.NullPointerException' exception. Cannot evaluate XXXX.toString()

Debug 模式下，检测到值为 null，就会报错，但程序可以正常执行。

==> 因为 Debug 模式下需要显示变量信息，这个信息就是要调用 toString()方法得到的，所以如果 toString()方法在对 null 变量进行操作，就会出现这种异常。然而，出现这种情况并没有关系，因为这是 Debug 下看到的，正常执行下来没问题就不用管

第三组：

1、抽象类是否也算是一个子类，只是它是适用于重新实现接口？对于抽象类概念感觉还是有点不太理解。

==> 第四组 2.1

2、Java 中如何获取以前的时间，例如：一个月之前或一年之前的时间。

==> 例程

第四组：

1.Java 静态（static）方法的调用有哪些方式？

（1） 通过 **类名.方法名（）**直接调用 **推荐**

（2） 通过类的实例对象 **对象名.方法名（）**调用

（3） 引用调用

（4） null 引用调用

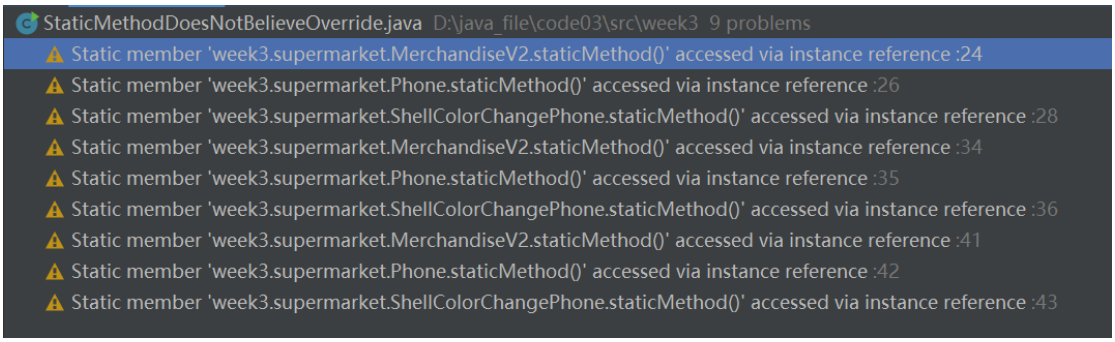
静态方法没有覆盖，静态方法没有 this 自引用，不访问成员变量。

This 自引用是多态的核心，静态方法不多态。

```java
System.out.println("-----1------");
//  MerchandiseV2（父类） > Phone > ShellColorChangePhone   继承关系
// >> TODO 静态方法可以被继承.    类名.方法名（）
MerchandiseV2.staticMethod();
Phone.staticMethod();
ShellColorChangePhone.staticMethod();

System.out.println("-----2------");
// >> TODO 通过类的实例对象       对象名.方法名（）
MerchandiseV2 a = new MerchandiseV2();
a.staticMethod();
Phone b = new Phone();
b.staticMethod();
ShellColorChangePhone c = new ShellColorChangePhone();
c.staticMethod();


System.out.println("-----3------");
// >> TODO 用引用调用静态方法没有覆盖,使用引用调用静态方法的内容
MerchandiseV2 m = superMarket.getMerchandiseOf( merchandiseIndex: 100);
m.staticMethod();
((Phone) m).staticMethod();
((ShellColorChangePhone) m).staticMethod();


System.out.println("-----4------");
// TODO 使用有类型的null引用调用静态方法的内容
((MerchandiseV2) null).staticMethod();
((Phone) null).staticMethod();
((ShellColorChangePhone) null).staticMethod();
```

```
-----1------
staticMethod in MerchandiseV2
staticMethod in Phone
staticMethod in ShellColorChangePhone
-----2------
staticMethod in MerchandiseV2
staticMethod in Phone
staticMethod in ShellColorChangePhone
-----3------
staticMethod in MerchandiseV2
staticMethod in Phone
staticMethod in ShellColorChangePhone
-----4------
staticMethod in MerchandiseV2
staticMethod in Phone
staticMethod in ShellColorChangePhone

Process finished with exit code 0
```

当使用后面三种调用方法时，会有下图标灰部分的警告：



2、抽象类和接口的关系：

抽象类和接口看起来功能相似，他们各自的特点和之间的区别有哪些？

2.1 抽象类是什么？

抽象类和具体类是相对的概念。“抽象”是一种存在思想逻辑中的概念，而“具体”是一种可见可触摸的现实对象。简单说，比如“人”比“男人”抽象一点，“动物”又比“人”更抽象一点，而“生物”又比“动物”更抽象。

抽象类的设计目的是代码复用，在抽象类中是先有子类再有父类，将子类的共性抽取

成父类。

抽象是对类本质的抽象，表达的是 is a 的关系，抽象类包含并实现子类的通用特性，将子类存在的差异化的特性进行抽象，交给子类去实现。

2.2 接口是什么？

接口的设计是对行为的约束，可以强制要求不同的类具有相同的行为，他只约束了行为的有无，并不对行为是如何实现的进行约束。

接口是对行为的抽象，表达的是 like a 的关系，接口的核心是定义行为，即实现类可以做什么，至于实现类主体是谁，如何实现，接口不关心。

2.3 使用场景

当你关注一个事物的本质时，使用抽象类！

当你关注一个操作的时候，使用接口！

2.4、接口和抽象的相同点

1、抽象类和接口都不能实例化，但可以定义抽象类和接口类的引用。

2、一个类如果继承了某个抽象类，或实现了某个接口，都需要对其中的抽象方法全部进行实现，否则该类仍然需要被声明为抽象类。

2.5、接口和抽象类的不同点

1、接口比抽象类更加抽象，因为抽象类中可以定义构造器，可以有抽象方法和具体方法。接口中不能定义构造器，而且其中的方法全部是抽象方法。

2、抽象类中的成员可以是 private、default、protected 和 public 的。接口中的成员全是 public 的。

3、抽象类可以定义成员变量，接口中定义的成员变量都是常量。

4、有抽象方法必须被声明为抽象类，抽象类不一定有抽象方法。

5、单继承，extends。多实现，implements。

6、接口成员变量默认为 public static final，必须赋初值，不能被修改；其所有的成员方法都是 public、abstract 的。抽象类中成员变量默认 default，可在子类中被重新定义，也可被重新赋值；

7、抽象方法被 abstract 修饰，不能被 private、static、synchronized 和 native 等修饰，必须以分号结尾，不带花括号。

8、抽象类作为很多子类的父类，它是一种模板式设计。而接口是一种行为规范，它是一种辐射式设计。

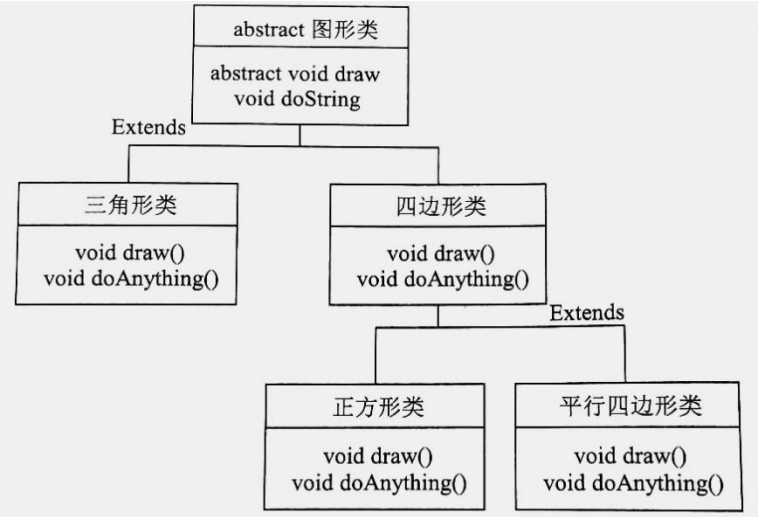


图 7.10　抽象类继承关系

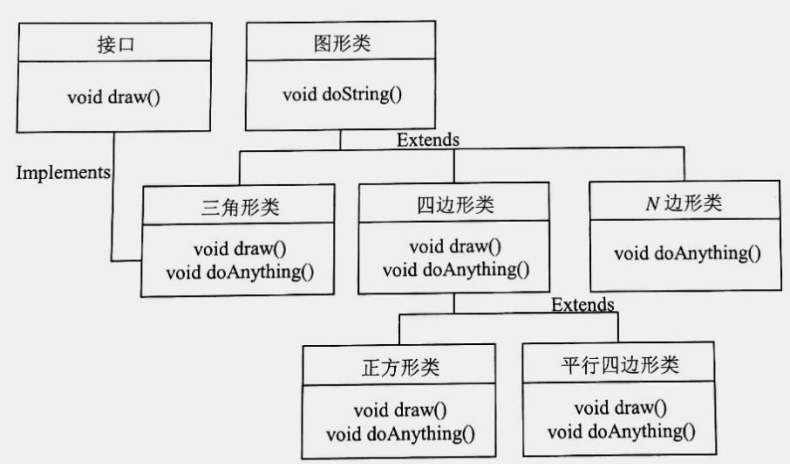


图 7.11　使用接口继承关系

3、封装、继承和多态的关系：

面向对象编程有三大特点：封装、继承和多态，这三者相互联系。

3.1 封装

封装是面向对象编程的核心思想。将对象的属性和行为封装起来，其载体就是类，类通常对客户隐藏其实现细节，这就是封装的思想。采用封装的思想保证了类内部数据结构的完整性，应用该类的用户不能轻易地直接操作此数据结构只能执行类允许公开的数据。这样就避免了外部操作对内部数据的影响，提高了程序的可维护性。

封装的具体方法步骤：

1.属性私有化

2.对外提供访问接口（即提供 setXXX() 方法和 getXXX() 方法）

3.2 继承

继承的主要目的是减少代码复写，提高程序的扩展性。

继承的几个特点：

1、子类拥有父类非private 的属性和方法。

2、子类可以拥有自己属性和方法，即子类可以对父类进行扩展。

3、子类可以用自己的方式实现父类的方法。（即多态）。

3.3 多态

多态: 指不同对象接收到同一消息时会产生不同的行为（一个接口，多种方法）

简单来说,就是在同一个类或继承体系结构的基类与派生类中，用同名函数来实现各种不同的功能

多态的三个条件：

a） 继承的存在(继承是多态的基础,没有继承就没有多态).

b） 子类重写父类的方法(多态下调用子类重写的方法).

c） 父类引用变量指向子类对象(子类到父类的类型转换).

多态性主要体现在：向不同的对象发送同一个消息，不同对象接收到消息时产生不同的行为，即每个对象以自己的方式响应同样的消息。

重载和重写的区别：

重写是子类对父类的允许访问的方法的实现过程进行重新编写, 返回值和形参都不能改变。即外壳不变，核心重写！

重载(overloading) 是在一个类里面，方法名字相同，而参数不同。返回类型可以相同也可以不同。

方法的重写(Overriding)和重载(Overloading)是 java 多态性的不同表现，重写是父类与子类之间多态性的一种表现，重载可以理解成多态的具体表现形式。