2.1-面向对象方法的特性

1.面向对象方法的四个基本特征：**抽象、封装、继承、多态**。
抽象：忽略问题中与当前目标无关的方面，只关注有关的方面。面向对象方法采用（**数据抽象**）方法来构建程序的类与对象。
过程抽象是将整个系统的功能划分为若干部分，强调功能完成的过程和步骤，而隐藏其具体实现；数据抽象是将系统中需要处理的数据和这些数据上的操作结合在一起，抽象成不同的抽象数据类型，每个抽象数据类型既包含了数据，也包含了针对这些数据的操作。
封装：信息隐蔽技术。利用抽象数据类型将数据和基于数据的操作封装在一起；用户只能看到对象的封装界面信息，对象的内部细节对用户隐蔽；目的在于将对象的使用者和设计者分开，使用者不必知道具体细节。Java 的封装性是通过访问（**控制权限**）来实现的。
继承：基于已有类产生新类的机制。新的类可以获得已有类的属性和行为，称新类为已有类的子类或者派生类。在继承过程中，子类继承了超类的特性，包括方法和实例变量。子类可修改继承的方法或者增加新的方法。
继承分为单继承和多继承。单继承是一个子类只有一个超类；多继承是可以用多个超类。（**Java 只能使用单继承**）
多态：在有继承情况下，超类和子类的对象们都可以响应同名的消息，但这些对象对这些消息的具体响应方式可以是不一样的。这是通过子类用的时候覆盖来实现的。

2.2 类与对象

2.2.1 类声明与对象的创建

类是具有相同操作功能和相同的数据格式（属性）的对象的集合与抽象；对象是类的具体实例。
类的声明：
[public][abstract|final] class 类名称 [extends 父类名称][implements 接口名称列表]{
数据成员声明及初始化;
方发声明及方法体;
}
说明:
extends：如果所声明的类是从某父类派生而来，需要将父类名字写在 extends 后面。
implements：声明的类实现某些接口，那么接口的名字应写在 implements 之后。
public 代表公有类，abstract 为抽象类。final 是终结类，不可被继承。
类的使用：构造实例（对象）
对象引用声明：
1.语法:
（1）定义一个引用变量名： 类名 引用变量名。例如 Clock aclock;
此时并没有生成对象

（2）创建对象： new <类名>() 例如： aclock = new Clock();

new 作用是为 clock 类型的对象分配内存空间，返回对象的引用。引用变量可以被赋空值。

2.2.2 数据成员

数据成员是用来表示对象的状态, 也可以用来表示在整个类所有对象之间要共享的数据。
数据成员的声明:
[public|protected|private][static][final][transient][volatile]数据类型 变量名 1[=变量初值]，变量名 2[=变量初值],...;
说明:
1.必须有的是数据类型、变量名;
2.public protected private 为访问控制符;
3. static 指这是一个静态成员变量（类的变量）;
4.final 指变量的值不能被修改;
5.transient 指明变量不需要序列化;
6.volatile 指是一个共享变量。

实例变量-属于对象的属性：
1.没有 static 修饰的变量（数据成员）称为实例变量;
2 存储所有实例都需要的属性，不同实例的属性值可能不同;
3.可以通过下面的表达式来访问：<实例名>.<实例变量名>
类变量-静态变量
1.用 static 修饰;
2.在整个类只有一个值;
3.类初始化时就被赋值;
4.使用情况：类中所有对象都相同的属性；经常需要共享数据；系统用到的一些常量值。
引用格式:
<类名|实例名>.<类变量名>

2.2.3 方法成员

行为表示一个对象能做的事情，或者能够从一个对象取得的信息。通过声明方法成员可以定义类的行为。

方法分为类的方法和实例的方法。

实例方法表示每一个实例的功能或行为。类方法是表示类的公共的行为或者功能。

方法成员语法形式:
[public|protected|private][static][final][abstract][native][synchronized]返回类型 方法名（[参数列表]）[throws exceptionList]
{
方法体
}
解释:

方法体是真正要执行的语句。返回类型、方法名都是必须要有的

（1）public,protected,private 为访问权限控制符；static 指明方法是一个类方法；final 是终结方法；abstract 是抽象方法；native 是用来将 java 码和其它语言代码集成起来；synchronized 用来控制多个并发线程对共享数据的访问。

（2）返回类型可以是任意的 java 了数据类型，不需要返回类型时用 void。

（3）参数类型可以是简单数据类型，也可以是引用类型（数据、类或接口）。

（4）方法体是对方法的具体实现，包括局部变量声明以及所有合法 java 语句。

（5）throws exceptionList 列出异常列表。

1.**实例方法**-表示特定对象的行为。定义时不用 static。

调用：给对象发消息，使用对象某个行为功能：调用对象的某个方法。

调用格式：

<对象名>.<方法名>([参数列表])

<对象名>为消息的接收者。注意：在类内，直接用方法名调用。

参数传递：

（1）值传递：参数类型为基本数据类型时；

（2）引用传递：参数类型为对象类型或者数据时，设计传递的是引用。实参名和形参名指向了同一个对象。

2.**类方法**-公共或共享的功能。它也被称为静态方法，声明前需要加 static，它不能被声明为抽象的，类名直接调用，也可以使用实例调用。

3.**可变长参数**：可变长参数使用省略号表示，实质上是数组。利用，“String...s”表示 String[] s。对于可变长参数的方法，传递给可变长参数的实际参数可以是多个对象，也可以是一个对象或者没有对象。

课本一个例题的解释：

```
static double maxArea(Circle c, Rectangle... varRec){
        Rectangle[] rec =varRec;
        for( Rectangle r: rec)
    {
         程序;
    }
```

rec 就是一个对象应用数组，多少元素不确定，要看给几个就是几个。程序里面如何处理呢？用这个增强型的 for 循环，定义一个 rectangle 的应用，在循环的每一次，在这个数组中循环去从这个数组取出一个元素赋值给 r。

2.2.4 包

包是一组类的集合，一个包可以包含多个文件或者子包。

包的作用：将相关的源代码文件组织在一起；类名的空间管理，利用包来划分名字空间可以避免类名冲突；提供了包一级的封装及存取权限。

包命名：要求独一无二，使用小写字母表示。推荐使用机构的 Internet 域名反序作为包名前导，但是，若包名有任何不可用于标识符的字符，用下划线代替；若包名中的任何部分与关键字冲突，后缀下划线；若包名中任何部分以数字或其他不能用作标识符起始的字符开头，前缀下划线。

1.编译单元与类空间

编译单元：一个 java 源代码文件为一个编译单元，由三部分组成：

所属包的声明;

import 包声明，用于导入外部的类;

类和接口的声明。

一个编译单元中只能有一个 public 类，该类名与文件名相同，编译单元中的其它类往往是 public 类的辅助类，经过编译，每一个类都会产生一个 class 文件。

包的声明:

命名的包： package My package;

默认包： 不包含声明的包都放入默认包中了。

2.包与目录

每个包对应一个目录，也就是文件夹。但是每个目录不一定对于一个目录。

引入包：为了使用其它包中的类，需要使用 import 语句引用所需要的类。

java 编译器为所有的程序自动引入包 java.lang。

import 语句的格式:

import package1[.package2...].(classname |*);

这里面，package1[.package2...]表明包的层次，对应于文件目录;

classname 代表所引入的类名，如果引入所有的类，使用*。

3.静态引入

单一引入:

引入某一指定的静态成员，例如 import static java.lang.Math.PI;

全体引入：引入全体静态成员，例如：import static java.lang.Math.*;

2.2.5 类的访问权限控制

表 2.1 类访问控制符与访问能力之间的关系

类型	无修饰符	Public
同一包中的类	是	是
不同包中的类	否	是

在一个编译单元中至多有一个公共类（public class），且源程序文件名必须与公共类的名字相同。

类的访问权限有:

（1）公有（public）：可以被其它任何方法访问（前提是类成员所属的类有访问权限）;

（2）保护（protected）：只可被同一类及其子类的方法访问;

（3）私有（private）：只可被同一类的方法访问;

（4）默认（default）：仅允许同一个包内的访问。

具体如表 2.2 所示。

表 2.2 类成员在不同范围是否可以被访问

类型	private	无修饰	protected	Public
同一类	是	是	是	是
同一包中的子类	否	是	是	是
同一包中的非子类	否	是	是	是
不同包中的子	否	否	是	是

类				
不同包中的非子类	否	否	否	是

get 方法: 功能是取得属性变量的值。类的数据成员设置成私有的，另一个类就无法读取了。这个时候就是使用 get 方法和 set 方法。set 用于设置对象的值或属性。
get 方法以 get 开头，后面是实例变量的名字，实例变量名字在这里一般首字母要大写。
例如：我们需要得到 circle 里面 radius 的值，那么需要写这样一个方法：
public int getRadius(){ return radius;}
set 方法：用于修改属性变量的值。set 方法名以 set 开头，后面是实例变量的名字，，实例变量名字在这里一般首字母要大写。
例如：在圆中，我们可以写一个 set Radius，然后用参数值 R 去设置这个圆的半径值。
public void setRadius(int r){ radius =r; }
set 和 get 方法都是公有接口。
this 关键字：如果方法内的局部变量（包括形参）名与实例变量名相同，则方法体内访问实例变量时需要使用 this 关键字。
例如：
public void setRadius(int **radius**){ **this.radius** = radius;}
这个里面形参为 radius，然后实例变量也是 radius，为了解决冲突问题，实例变量使用了 this 关键字。

2.3.1 对象初始化

系统在生成对象时，会为对象分配内存空间，并自动调用（构造方法）对实例变量进行初始化。
构造方法特点：
（1）方法名与类名相同；（2）不定义返回类型；（3）通常被声明为 public；（4）可以由任意多个参数；(5) 主要完成对象初始化；(6) 不能在程序显式调用；(7) 在生成一个对象时，会自动调用该类的构造方法为新对象初始化。
1.系统提供的默认构造方法
没有参数（内部类除外），方法体为空；
使用默认的构造方法初始化对象时，如果在类生命中没有给实例变量赋初值，则对象的属性值为 0 或空。
2.自定义构造方法与方法重载
在生成对象时给构造方法传送初始值，对对象进行初始化。
构造方法可以被重载：一个类可以由两个或两个以上同名方法，但参数表不同。在方法调用时，可以通过参数列表不同来辨别应调用哪个方法。
只要显式声明构造方法，编译器不会生成默认的构造方法。
也可以显式声明无参数构造方法，方法体中可以定义默认初始化方式。
this 关键字
可以使用 this 关键字在一个构造方法中调用另外的构造方法；
代码更简洁，维护更容易；
通常使用参数比较少的方法调用参数更多的方法。
final 关键字-一经声明不修改

实例变量和类变量都可以被声明为 final；

final 实例变量可以在类中给出初始值，或者在每个构造方法结束之前初始化；

final 类变量必须在声明的同时初始化。

2.3.2 内存回收技术

当对象不再使用时，它就可能被 JAVA 内存回收。无用对象指离开作用域的对象或者无引用指向的对象。

Java 运行时系统通过垃圾收集器周期性地释放无用对象所使用的内存。

Java 运行时系统会在对象进行自动垃圾回收前，自动调用对象的 finalize()方法。

垃圾收集器：自动扫描对象的动态内存区，对不再使用的对象做上标记进行垃圾回收；作为一个后台线程运行，通常在系统空闲时异步执行。

finalize()方法：

在 java.lang.Object 中声明，因此 Java 中每一个类都有该方法

用于释放资源

类可以覆盖（重写）finalize()方法。

finalize 方法有可能在任何时机以任何次序执行。我们在使用是应使其无论何时都不影响程序的正确执行。

2.4 枚举类

可取值是有限的，可取值是可以一一列出来的。

声明类型：

[public] enum 枚举类型名称[implements 接口变量列表]

{

枚举值;

变量成员声明及初始化;

方发声明及方法体;

}

说明：

1.枚举定义实际上是定义一个类;

2.所有枚举类型都隐含继承自 java.lang.Enum，因此枚举类不能再继承其他任何类;

3.枚举类型的类体可以包括方法和变量;

4.枚举类型的构造方法必须是包内私有或者私有的。定义在枚举开头的常量会被自动创建，不能显式地调用枚举类的构造方法。

2.4.1 枚举类型的默认方法

静态的 values()方法用于获得枚举类型的枚举值的数组;

toString 方法返回枚举值的字符串描述;

valueOf 方法以字符串形式表示枚举值转化为枚举类型的对象;

ordinal 方法获得对象在枚举类型中的位置索引。

2.5 应用举例

2.5.2 声明 toString()方法

如果需要自己覆盖 toString()方法，那么：

（1）必须声明为 public;

（2）返回类型必须为 String;

（3）方法的名称必须为 toString，且没有参数;

（4）方法体中不要使用输出方法 System.out.println()。

最后一个注意点：

静态方法可以直接通过类名调用，任何的实例也都可以调用，

因此静态方法中不能用 this 和 super 关键字，**不能直接访问所属类的实例变量和实例方法(就是不带 static 的成员变量和成员成员方法)，只能访问所属类的静态成员变量和成员方法。**

因为实例成员与特定的对象关联！这个需要去理解，想明白其中的道理，不是记忆!!!

因为 static 方法独立于任何实例，因此 static 方法必须被实现，而不能是抽象的 abstract。例如为了方便方法的调用，Java API 中的 Math 类中所有的方法都是静态的，而一般类内部的 static 方法也是方便其它类对该方法的调用。

静态方法是类内部的一类特殊方法，只有在需要时才将对应的方法声明成静态的，一个类内部的方法一般都是非静态的。