5. 面向对象编程2

本章技能点列表

技能点名称	难易程度	认知程度	重要程度
封装：定义和优点	中	理解	**
封装：权限修饰符	难	应用	***
继承：定义和优点	中	理解	**
继承：方法重写	中	应用	***
继承：Object	中	理解	***
继承：super	难	理解	**
继承：构造方法执行过程	难	理解	**
继承：==和equals	难	应用	***
多态：定义和优点	中	理解	**
多态：向上转型	难	应用	***

本章技能点列表

技能点名称	难易程度	认知程度	重要程度
多态：向下转型 instanceof	难	应用	**
多态：应用父类做形参	中	应用	***
多态：简单工厂模式	中	应用	***
final	中	应用	***
抽象类和抽象方法	中	应用	***
接口	中	应用	***
接口应用：内部比较器Comparable	难	应用	***
接口应用：外部比较器Comparator	难	应用	***
内部类	难	应用	**
垃圾回收器	易	了解	*

面向对象的三大特征

- 继承 inheritance
  - 子类 父类
  - 子类可以从父类继承属性和方法
  - 子类可以提供自己单独的属性和方法
- 封装/隐藏encapsulation
  - 对外隐藏某些属性和方法
  - 对外公开某些属性和方法
- 多态 polymorphism
  - 为了适应需求的多种变化，使代码变得更加通用！
- 面向过程只有封装性（功能的封装，而没有数据的封装），没有继承和多态

隐藏/封装(encapsulation)

- 为什么需要封装？封装的作用和含义？
  - 我要看电视，只需要按一下开关和换台就可以了。有必要了解电视机内部的结构吗？有必要碰碰显像管吗？
  - 我要开车，....
- 隐藏对象内部的复杂性，只对外公开简单的接口。便于外界调用，从而提高系统的可扩展性、可维护性。
- 我们程序设计要追求“高内聚，低耦合”。
  - 高内聚：就是类的内部数据操作细节自己完成，不允许外部干涉；
  - 低耦合：仅暴露少量的方法给外部使用。

使用访问控制符，实现封装

- 成员（成员变量或成员方法）访问权限共有四种：
  - public 公共的
    - 可以被项目中所有的类访问。(项目可见性)
  - protected 受保护的
    - 可以被这个类本身访问；同一个包中的所有其他的类访问；被它的子类（同一个包以及不同包中的子类）访问
  - default / friendly 默认的/友好的（包可见性）
    - 被这个类本身访问；被同一个包中的类访问。
  - private 私有的
    - 只能被这个类本身访问。（类可见性）
- 类的访问权限只有两种
  - public 公共的
    - 可被同一项目中所有的类访问。(必须与文件名同名)
  - default / friendly 默认的/友好的
    - 可被同一个包中的类访问。

使用访问控制符，实现封装

	同一个类	同一个包中	子类	所有类
private	*			
default	*	*		
protected	*	*	*	
public	*	*	*	*

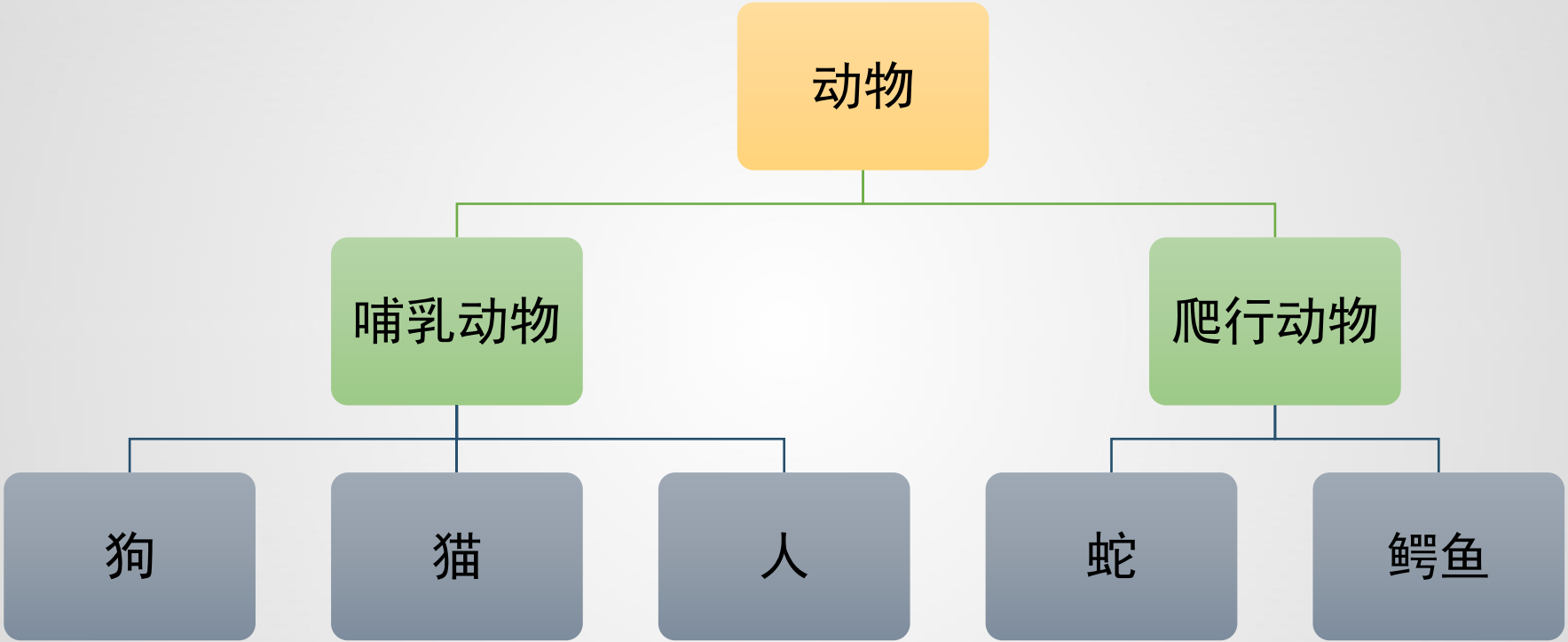
- 封装要点：
- 类的属性的处理:
  - 一般使用private. (除非本属性确定会让子类继承)
  - 提供相应的get/set方法来访问相关属性. 这些方法通常是public，从而提供对属性的读取操作。
    （注意：boolean变量的get方法是用：is开头!）
- 一些只用于本类的辅助性方法可以用private，
- 希望其他类调用的方法用public

继承(1)

- 类是对对象的抽象，继承是对某一批类的抽象，从而实现对现实世界更好的建模。
- 提高代码的复用性！
- extands的意思是“扩展”。子类是父类的扩展

- 不同的叫法：超类、父类、基类、子类、派生类

继承(2)



继承(3)

```
public class TestExtends {
        public static void main(String[] args) {
                Mammal m1 = new Mammal();
                m1.puru();
                m1.eat();
        }
}
class Animal {
        String eyes="眼睛";
        String name="无名";

        public void eat(){
                System.out.println("动物吃东西！");
        }
}
class Mammal extends Animal {
        //哺乳
        public void puru(){
                System.out.println("小动物吃奶！");
        }
}
```

继承(4)

- 小结：
  - 通过继承可以简化类的定义，实现代码的重用
  - 子类继承父类的成员变量和成员方法，但不继承父类的构造方法
  - java中只有单继承，没有像c++那样的多继承。多继承会引起混乱，使得继承链过于复杂，系统难于维护。就像我们现实中，如果你有多个父母亲，那是一个多么混乱的世界啊。多继承，就是为了实现代码的复用性，却引入了复杂性，使得系统类之间的关系混乱。
  - java中的多继承，可以通过接口来实现
  - 如果定义一个类时，没有调用extends，则它的父类是：java.lang.Object。

课堂练习(10分钟)

- 熟悉继承实现方式!

方法的重写（override）

- 在子类中可以根据需要对从基类中继承来的方法进行重写。
- 重写方法必须和被重写方法具有相同方法名称、参数列表和返回类型。
- 重写方法不能使用比被重写方法更严格的访问权限。（由于多态）

重写（override）举例代码

```
public class TestOverride {
        public static void main(String[] args) {
                Animal animal = new Animal();
                animal.shout();
                Dog dog = new Dog();
                dog.shout();
        }
}
class Animal{
    void shout(){
        System.out.println("发出声音！");
    }
}
class Dog extends Animal {
    void shout(){
        System.out.println("旺旺旺！");
    }
}
```

Object类

- Object类是所有Java类的根基类
- 如果在类的声明中未使用extends关键字指明其基类，则默认基类为Object类

```
public class Person {
        …
}
```



```
public class Person extends Object {
        …
}
```

- 重写：toString方法：
  - 默认返回：包名+类名+@+哈希码
  - 可以重写！
- 打开API文档，开始熟悉！

根据对象内存位置生成，唯一不重复！

课堂练习(10分钟)

- 熟悉方法重写
- 熟悉Object
- 重写toString方法
- 打开API文档，开始学着看看

作业

课下将课堂讲授多复习。一定要非常熟悉！

super关键字

- super是直接父类对象的引用。
- 可以通过super来访问父类中被子类覆盖的方法或属性。

super 示例代码

```
public class Test {
         public static void main(String[] args) {
                  new ChildClass().f();
         }
}
class FatherClass {
    public int value;
    public void f(){
        value = 100;
        System.out.println
        ("FatherClass.value="+value);
    }
}
class ChildClass extends FatherClass {
    public int value;
    public void f() {
        super.f();
        value = 200;
        System.out.println
             ("ChildClass.value="+value);
        System.out.println(value);
        System.out.println(super.value);
    }
}
```

课堂练习 10分钟

熟悉super用法

继承深化

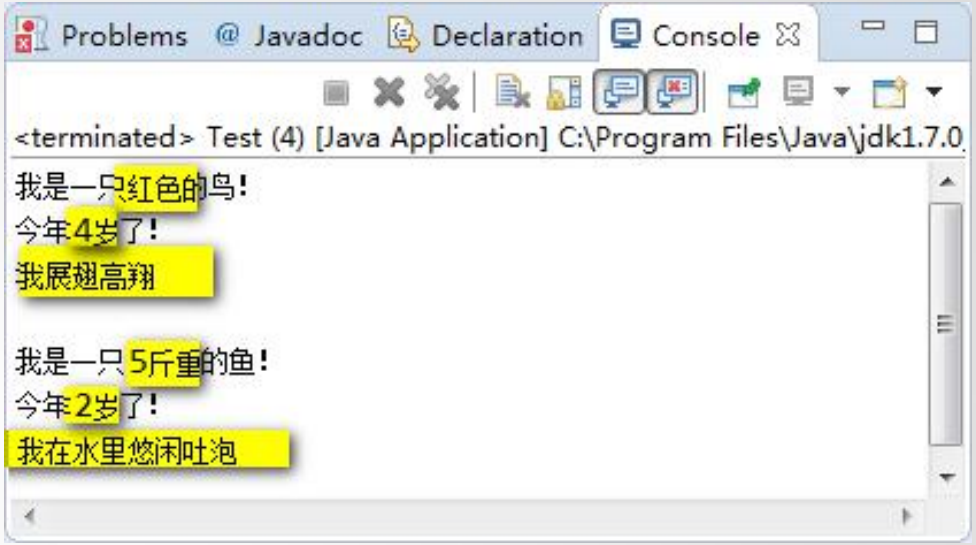
- 父类方法的重写：
  - “==”：方法名、形参列表相同。
  - “≤≤”：返回值类型和异常类型，子类小于等于父类。
  - “≥”：访问权限，子类大于等于父类
- 构造方法调用顺序：
  - 根据super的说明，构造方法第一句 总是：super(...)来调用父类对应的构造方法。
  - 先向上追溯到Object，然后再依次向下执行类的初始化块和构造方法，直到当前子类为止。

对象的比较—==和equals()

- ==：
  - 比较两基本类型变量的值是否相等
  - 比较两个引用类型的值即内存地址是否相等，即是否指向同一对象。
- equals()：
  - 两对象的内容是否一致
- 示例
  - object1.equals(object2)　如：p1.equals(p2)
    - 比较所指对象的内容是否一样
    - 是比较两个对象，而非两个基本数据类型的变量
  - object1 == object2　　如：p1==p2
    - 比较p1和p2的值即内存地址是否相等，即是否是指向同一对象。
- 自定义类须重写equals()，否则其对象比较结果总是false。

示例

- 需求
  - 1.鱼类
    - 属性 ： 年龄 重量
    - 方法：自我介绍 游泳
  - 2.鸟类
    - 属性：年龄 颜色
    - 方法：自我介绍 飞
- 分析
  - 使用继承：
    - 抽取出动物类：属性（年龄） 方法（自我介绍）
    - 鱼类继承动物类，提供特有属性 重量和特有方法 游泳
    - 鸟类继承动物类，提供特有属性 颜色和特有方法 飞
    - 开发测试类，进行测试
  - 使用封装
    - 属性私有 方法public 提供对应的构造方法



多态 polymorphism

港台明星是多态。有：刘德华、曾志伟！

男朋友长得怎样?

有点像港台明星

哇，好！明天带回家看看

我的女朋友

丈母娘大人

女朋友把我带到家，丈母娘一看，疯了！原来是像“曾志伟”！

多态 polymorphism

- 多态性是OOP中的一个重要特性，主要是用来实现动态联编的，换句话说，就是程序的最终状态只有在执行过程中才被决定而非在编译期间就决定了。这对于大型系统来说能提高系统的灵活性和扩展性。
- java中如何实现多态?使用多态的好处?
- 引用变量的两种类型：
  - 编译时类型（模糊一点，一般是一个父类）
    - 由声明时的类型决定。
  - 运行时类型（运行时，具体是哪个子类就是哪个子类）
    - 由实际对应的对象类型决定。
- 多态的存在要有3个必要条件：
  - 要有继承，要有方法重写，父类引用指向子类对象

多态示例代码

```
package object;
public class TestPolym {

public static void main(String[] args) {
            Animal animal = new Dog();  //向上可以自动转型

            System.out.println(animal.age);    //属性调用时，仍然是基类的属性。属性没有多态！
//          animal.shout();

            animalCry(new Dog());
            //传的具体是哪一个类就调用哪一个类的方法。大大提高了程序的可扩展性。
            //如果没有多态，我们这里需要写很多重载的方法。如果增加一种动物，就需要重载一种动物的喊叫方法。非常麻烦。

               //有了多态，只需要增加这个类继承Animal基类就可以了。
            animalCry(new Cat());

            Dog dog = (Dog) animal;  //编写程序时，如果想调用运行时类型的方法，只能进行类型转换。不然通不过编译器的检查。
            dog.gnawBone();
                                    System.out.println(dog instanceof Animal);
            System.out.println(animal instanceof Cat);
            System.out.println(animal instanceof Dog);

            }

            static void animalCry(Animal a){
                        a.shout();
            }

}
```

```
class Animal {
        int age=10;
        public void shout(){
            System.out.println("叫了一声！");
        }
}
class Dog extends Animal {
        int age=28;
        public void shout() {
            System.out.println("旺旺旺！");
        }

        public void gnawBone(){
            System.out.println("我在啃骨头");
        }

}
class Cat extends Animal {
        int age=18;
        public void shout() {
        System.out.println("喵喵喵喵！");
        }
}
```

引用数据类型的类型转换

- 子类转换为父类：自动转换
  - 上转型对象不能操作子类新增的成员变量和方法。
  - 上转型对象可以操作子类继承或重写的成员变量和方法
  - 如果子类重写了父类的某个方法，上转型对象调用该方法时，是调用的重写方法。
- 父类转换为子类：强制转换
  - （绝不是做手术，而是父类的真面目就是一个子类，否则会出现类型转换错误）

final关键字

- final可以用来修饰变量，方法，类。
- 修饰变量：变量一旦被初始化便不可改变，相当定义了一常量。
  - final int x=3;
  - x=4;
- 修饰方法：final方法是在子类中不能被覆盖的方法
  - final returnType methodName(paramList){...}
  - final void eat() { … }
- 修饰类：final类是无法被任何类继承的。
  - final class finalClassName{ … }
  - final class Person{ … }
- 举例

抽象类

- 为什么需要抽象类? 如何定义抽象类?
  - 是一种模版模式。抽象类为所有子类提供了一个通用模版，子类可以在这个模版基础上进行扩展。
  - 通过抽象类，可以避免子类设计的随意性。通过抽象类，我们就可以做到严格限制子类的设计，使子类之间更加通用。

```
abstract class Animal {
	abstract void shout(); //抽象方法没有方法体!

}
class Dog extends Animal {

	void shout() {  //必须重写父类的抽象方法否则编译通不过
		// TODO Auto-generated method stub
		System.out.println("旺旺旺! ");
	}
}
```

抽象类

- 要点：
  - 抽象方法和抽象类均必须用abstract来修饰。
  - 抽象方法没有方法体，只需要声明不需实现。

  - 有抽象方法的类只能定义能抽象类
  - 相反抽象类里面的方法不一定全是抽象方法，也可能没有抽象方法。

  - 抽象类可以包含属性、方法、构造方法。
  - 抽象类不能实例化，及不能用new来实例化抽象类，只能用来被子类调用。
  - 抽象类只能用来继承。
  - 抽象方法必须被子类实现。抽象类的子类必须覆盖所有的抽象方法才能被实例化，否则还是抽象类

接口 interface

- 我们前面用继承关系，描述了动物、哺乳动物、爬行动物的各种关系。

- 现在我们要描述：
  - 飞机 导弹 子弹 篮球 石头的关系?

接口 interface

- 为什么需要接口?接口和抽象类的区别?
  - 接口就是比“抽象类”还“抽象”的“抽象类”，可以更加规范的对子类进行约束。全面地专业地实现了：规范和具体实现的分离。
  - 接口就是规范，定义的是一组规则，体现了现实世界中“如果你是...则必须能...”的思想。如果你是天使，则必须能飞。如果你是汽车，则必须能跑。如果你好人，则必须干掉坏人；如果你是坏人，则必须欺负好人。
  - 接口的本质是契约，就像我们人间的法律一样。制定好后大家都遵守。
  - 项目的具体需求是多变的，我们必须以不变应万变才能从容开发，此处的“不变”就是“规范”。因此，我们开发项目往往都是面向接口编程！

接口 interface

- 接口相关规则
  - 接口中所有方法都是抽象的。
  - 即使没有显式的将接口中的成员用public标示，也是public访问类型的
  - 接口中变量默认用 public static final标示，所以接口中定义的变量就是全局静态常量。

  - 可以定义一个新接口，用extends去继承一个已有的接口
  - 可以定义一个类，用implements去实现一个接口中所有方法。
  - 可以定义一个抽象类，用implements去实现一个接口中部分方法。

接口 interface

- 如何定义接口?
  - 格式：
    - [访问修饰符] interface 接口名 [extends 父接口1，父接口2...] {
      - 常量定义 //总是public static final
      - 方法定义 //总是：public abstract
    - }

- 如何实现接口
  - 子类通过implements来实现接口中的规范
  - 接口不能创建实例，但是可用于声明引用变量类型。
  - 一个类实现了接口，必须实现接口中所有的方法，并且这些方法只能是public的。
  - Java的类只支持单继承，接口支持多继承

接口 interface

- C++支持多重继承，Java支持单重继承
- C++多重继承的危险性在于一个类可能继承了同一个方法的不同实现，会导致系统崩溃。
- Java中，一个类只能继承一个类，但同时可以实现多个接口，既可以实现多重继承的效果和功能，也避免的多重继承的危险性。

- class Student extents Person implements Runner， Flyer
- {…}
- 注意：extends 必须位于implements之前

内部类

- 将一个类定义置入另一个类定义中就叫作“内部类”
  - 类中定义的内部类特点
  - 内部类作为外部类的成员，可以直接访问外部类的成员（包括private成员），反之则不行。
  - 内部类做为外部类成员，可声明为private、默认、protected或public。
  - 内部类成员只有在内部类的范围之内是有效的。
  - 用内部类定义在外部类中不可访问的属性。这样就在外部类中实现了比外部类的private还要小的访问权限。
  - 编译后生成两个类： OuterClass.class 和OuterClass$InnerClass.class

- 内部类分类
- 成员内部类 静态内部类 方法内部类 匿名内部类

内部类

- 匿名内部类Anonymous
  - 可以实现一个接口，或者继承一个父类
  - 只能实现一个接口
  - 适合创建那种只需要一次使用的类，不能重复使用。比较常见的是在图形界面编程GUI里用得到。
  - 匿名内部类要使用外部类的局部变量，必须使用final修饰该局部变量

内部类

```
class Outer{
      int outer_i = 100;
      void test(){
            Inner in = new Inner();
            in.display();
            System.out.println(in.a);
      }
      class Inner{
          int a=5;
       void display(){
                  System.out.println("display: outer_i = " + outer_i);
             }
      }
}
```

1、Inner类是在Outer内部定义的
2、在Inner类中可以访问Outer类中的成员属性outer_i;
3、在Outer类中可在方法test()中创建内部类Inner的对象;
4、通过Outer类的对象调用test（）方法最终就可以执行Inner类中的方法

垃圾回收机制

- 对象空间的分配：
  - 使用new关键字创建对象即可

- 对象空间的释放：
  - 传统的C/C++语言，需要程序员负责回收已经分配内存。显式回收垃圾回收的缺点：
    - 程序忘记及时回收，从而导致内存泄露，降低系统性能。
    - 程序错误回收程序核心类库的内存，导致系统崩溃。
  - Java语言不需要程序员直接控制内存回收，是由JRE在后台自动回收不再使用的内存，称为垃圾回收机制(Garbage Collection)。
    - 可以提高编程效率。
    - 保护程序的完整性。
    - 其开销影响性能。Java虚拟机必须跟踪程序中有用的对象，确定哪些是无用的。

垃圾回收机制关键点

- 垃圾回收机制只回收JVM堆内存里的对象空间。
- 对其他物理连接，比如数据库连接、输入流输出流、Socket连接无能为力
- 现在的JVM有多种垃圾回收实现算法，表现各异。
- 垃圾回收发生具有不可预知性，程序无法精确控制垃圾回收机制执行。
- 可以将对象的引用变量设置为null，暗示垃圾回收机制可以回收该对象。
- 程序员可以通过System.gc()或者Runtime.getRuntime().gc()来通知系统进行垃圾回收，会有一些效果，但是系统是否进行垃圾回收依然不确定。
- 垃圾回收机制回收任何对象之前，总会先调用它的finalize方法（如果覆盖该方法，让一个新的引用变量重新引用该对象，则会重新激活对象）。
- 永远不要主动调用某个对象的finalize方法，应该交给垃圾回收机制调用。

学习面向对象章节的注意事项

- 大家不要希望学到现在就很精通面向对象，这只是开始。

- 事实上，很多人如果要对面向对象很精通，至少工作两年之后

总结

- 封装
  - 成员权限修饰符 private 默认 protected public
  - 类权限修饰符 默认 public
- 继承
  - 方法重写
  - 构造方法执行过程 super关键字
  - Object类 重写toString() equals();
- 多态
  - 使用父类做形参 使用父类做方法返回值
  - 向上转型 向下转型
- final
  - final修饰类 、成员变量、成员方法