## Java基础概念及特性

* 内容描述：学习Java基础的类，接口等结构，熟悉Java三大特性，了解Java对象在内存的分配方式
* 练习：写一个Animal接口，并实现一部分动物的实体类，在子类中需要扩展相关动物的特性

### 内容学习笔记

在java中，万物皆对象，将复杂的业务处理逻辑和实体属性分别作为方法和成员封装在类中，对外暴露清晰、简单的操作视图，可以让代码充分复用。

接口可以认为是java中没有多继承的补充，一般接口为功能性接口，而不是实体的抽象。接口的使用场景较多，可以通过接口对实体类进行扩展，通过接口可以实现类的门面模式等，面向接口编程可以使代码拥有更好的扩展性。

Java的三大特性：封装、继承、多态。封装，将一些操作或实体封装成类，对外仅仅暴露相关的方法用来对实体进行配置或进行运算。继承，通过对类的抽象，对可复用属性或方法进行提取，单独作为上层类，方便依据不同实体类的特有属性快速生成大量实体类，节省代码量，代码结构层次清晰。多态，即接口的多种不同实现，使程序依赖于抽象而不依赖于具体，方便扩展。

Java内存的分配方式：jvm在启动时，会向操作系统申请一块内存区域进行自我管理，该块内存大小可以通过jvm启动参数进行设定。jvm主要将申请来的内存分为堆、堆栈、方法区、程序计数器四大块。其中，程序计数器为线程私有，记录该线程执行到的指令地址，若发生线程切换，则记录中断的位置及线程上下文，当线程得到CPU的时间片后，恢复上下文，继续从中断的位置开始执行；方法区线程共享，主要用于存放类信息、常量池、静态变量；堆栈为线程私有，jvm是基于栈的线程执行模型，当一个线程（或方法）被执行时，堆栈上分配栈帧结构，用于保存该方法的传入参数、返回值引用等信息，当方法执行完成后，栈帧出栈，执行上一个栈帧中的运算；堆区域为线程共享，也是对象实际分配空间的地方，该区域是垃圾回收主要的关注区域，采用分代算法，将区域分为年轻代和老年代，永久带，年轻代分为Eden，Survivor, Survivor分为from，to，对象创建时内存的分配遵循以下规则：

1. 对象优先分配在Eden区，如果Eden区没有足够的空间，虚拟机执行一次Minor GC，此时会将Eden区中仍然存活的对象和from（或to）区的对象转移到to（或from区）。
2. 大对象直接进入老年代，比如大数组，可用来避免发生Minor GC时，大对象在from和to区之间的多次复制操作。
3. 长期存活的对象进入老年代。虚拟机为每个对象定义了一个年龄计数器，如果对象经过1次MinorGC后依然存活，那么对象进入Survivor，年龄加1，当年龄到达阈值后，进入老年代。
4. 如果Survivor区中存在大量相同年龄的对象，并且这些对象所占用的空间总和大于Survivor区的一半，则将该组对象直接放入老年代。
5. 空间分配担保。每次进行Minor GC时，JVM会计算Survivor区移至老年区的对象的平均大小，如果这个值大于老年区的剩余值大小则进行一次Full GC，如果小于检查HandlePromotionFailure设置，如果true则只进行Monitor GC,如果false则进行Full GC。

### **练习**

Animal作为顶层接口，定义如下方法：

|  |
| --- |
| **public interface** Animal {  **boolean** isExtinct();   AnimalPhylum getAnimalPhylum();*//门* AnimalClass getAnimalClass(); *// 纲* AnimalOrder getAnimalOrder();*//目* AnimalFamily getAnimalFamily();*//科* AnimalGenus getAnimalGenus(); *// 属* } |

仅仅是获得物种属性的方法。

物种的属性定义了AnimalPhylum 、AnimalClass 、AnimalOrder 、AnimalFamily 、AnimalGenus ，因为这些属性都是数量较少的有限集合，使用枚举类型进行定义，枚举类型中包含该属性的一般生物特征，既方便扩展，也方便实例化具体物种，例如：

|  |
| --- |
| **public enum** AnimalPhylum {  ***Chordata***(**"脊索动物门"**, **"特征是有脊椎"**, 1),***Arthropoda***(**"节肢动物门"**, **"特征是节肢"**,2);   **private** String **name**;  **private** String **characteristic**;  **private int index**;   **private** AnimalPhylum(String name, String characteristic, **int** index){  **this**.**name** = name;  **this**.setCharacteristic(characteristic);  **this**.**index** = index;  }   **private boolean** setCharacteristic(String characteristic){  **if** (characteristic.length() > ConfigRule.***CHARACTERISTIC\_MAX\_LEN***){  **this**.**characteristic** = characteristic.substring(0, 1024);  **return false**;  }   **this**.**characteristic** = characteristic;  **return true**;  }   @Override  **public** String toString() {  **return "AnimalPhylum{"** +  **"name='"** + **name** + **'\''** +  **", characteristic='"** + **characteristic** + **'\''** +  **'}'**;  } } |

接下来通过实现Animal接口，创建抽象类AbstractAnimal，定义共有属性，实现公有方法，最后通过继承AbstractAnimal，派生不同具体物种。

|  |
| --- |
| **public class** Bee **extends** AbstractAnimal{  **private static final** String ***DEFAULT\_UNITES*** = **"m/s"**;   **private double flySpeed;**  **private** String **units** = ***DEFAULT\_UNITES***;  **public** Bee(**double** flySpeed) {  **super**(**true**, AnimalPhylum.***Arthropoda***, **null**,  **null**, **null**, **null**, **null**);   **this**.**flySpeed** = flySpeed;  }   @Override  **public** String toString() {  **return "Wolf{"** + **super**.toString() +  **"flyspeed='"** + **flySpeed** + **units** + **'\''** +  **'}'**;  }   @Override  **public void** showCharactor() {  System.***out***.println(**"采蜜"**);  } } |

|  |
| --- |
| **public class** Wolf **extends** AbstractAnimal {  **private** String **color**;   **public** Wolf(String color) {  **super**(**true**, AnimalPhylum.***Chordata***, **null**,  **null**, **null**, **null**, **null**);   **this**.**color** = color;  }   @Override  **public** String toString() {  **return "Wolf{"** + **super**.toString() +  **"color='"** + **color** + **'\''** +  **'}'**;  }   @Override  **public void** showCharactor() {  System.***out***.println(**"四条腿，野生，食肉"**);  } } |