[读取配置文件 1](#_Toc528248167)

[Properties配置文件 1](#_Toc528248168)

[在Java中访问Properties配置文件 1](#_Toc528248169)

[简单工厂 2](#_Toc528248170)

[多态的转型 3](#_Toc528248171)

[向上转型 3](#_Toc528248172)

[向下转型 4](#_Toc528248173)

[Instance Of 运算符 4](#_Toc528248174)

[抽象 5](#_Toc528248175)

# 读取配置文件

项目中一些具有动态效应的内容是很少写死在代码中的.因为这样的代码不够灵活且增加维护成本.所以我们会把这些内容写出独立的配置文件(一般都是文本).常见的配置文件格式有

* Properties
* XML
* JSON
* Yaml

有专门配置文件格式,那在代码中就有与之对应的访问方式.今天说的是Properties.

## Properties配置文件

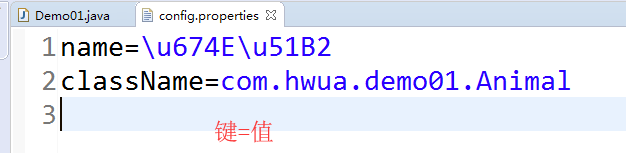
Properties是java.util包下的一个工具类.专门用于来读取后缀为properties的配置文件.在该配置文件中.所有的数据采用键值对的形式(甚至其它格式的配置文件也是采用键值对格式.只是写法不一样).

我们在src下新建一个File.因为只有src下的内容才会被编译.

## 在Java中访问Properties配置文件

在Java中想要访问Properties配置文件是有一定的代码流程.是固定的写法

* 提供Properties对象.并指向实际的配置文件.
* 通过类加载读取资源(写法是固定的)
* 根据键读取值





# 简单工厂

在一个完整的软件应用中.它一定会有多个模块.模块和模块之间的联系程度我们可以把它叫做耦合度.一个模块内部,功能代码的完善,紧凑我们把这种关系叫做内聚.一个优秀的软件它的设计应该是高内聚,低耦合的.

在Java中有一个很严重的现象就是对象的创建.无论是什么样的对象,它的创建都是很耗资源的(时间,内存等).并且我们创建一个类的对象时还要考虑该类中是否有其它类作为成员变量存在.这样代码我们认为模块和模块之间耦合度比较高.为了解决这个问题.我们选择把对象的创建统一放在一个地方.这个地方就叫做工厂.

假设在构建A的时候需要构建B,那A和B之间存在耦合.现在我们把对象的创建委托给工厂那再A需要B时,就不再是找B了,而是去找工厂,这样我们就降低A和B的耦合.

工厂有很多种.比如简单共产(静态工厂),抽下工厂等.这里学的是简单.在使用工厂时有两套写法

* 根据反射来构建对象(后期框架所采用的方式)
* 根据字符串匹配来构建对象(现在用的)



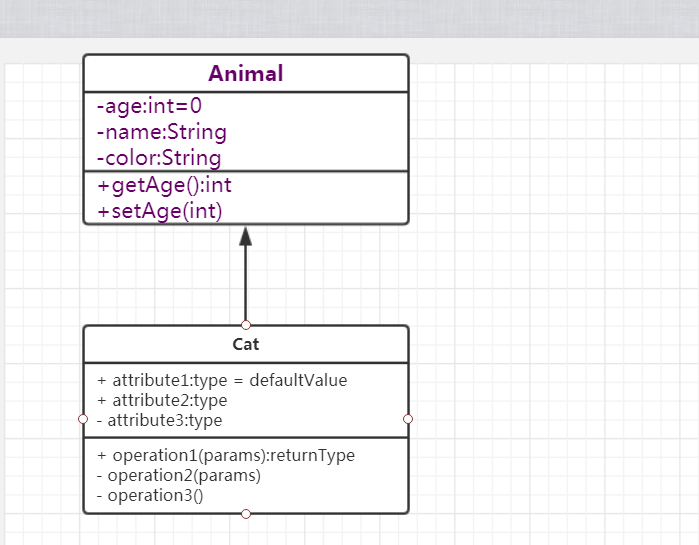
工厂的核心就是依赖于多态.通过用父类型来保存任意子类型的对象.实现动态绑定效果.配置文件中配置的什么样的内容就创建与之对应的子类对象.

# 多态的转型

多态是有向上转型和向下转型的.

## 向上转型

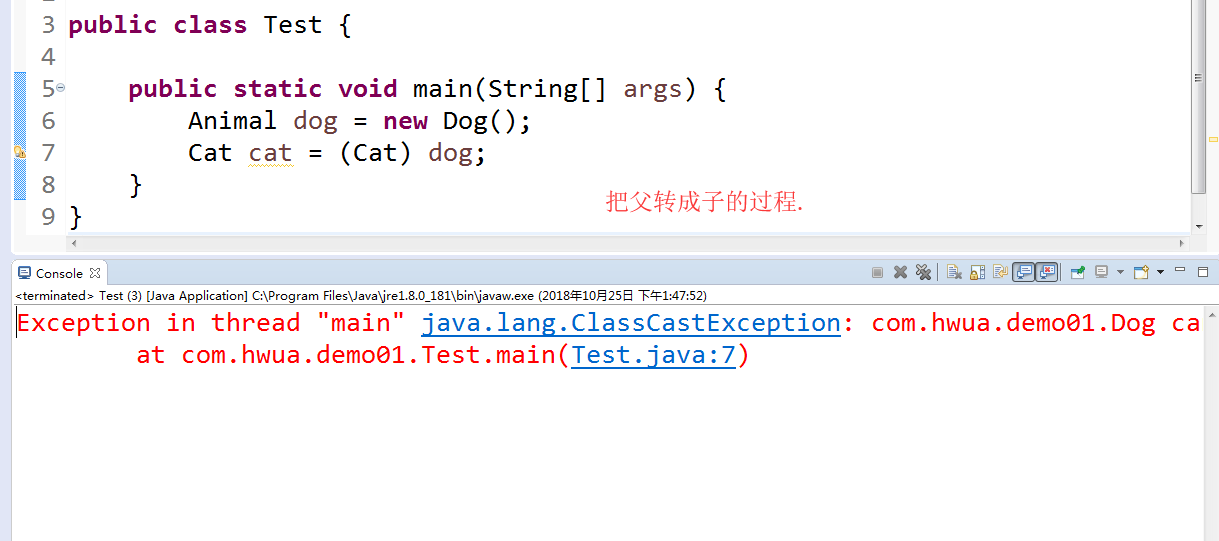
子类的对象转换为父类类型称为向上转型.在UML制图中我们一般把父类画在上面



向上转型是不会有类型偏差的.不会产生类型转换错误:ClassCastException.

## 向下转型

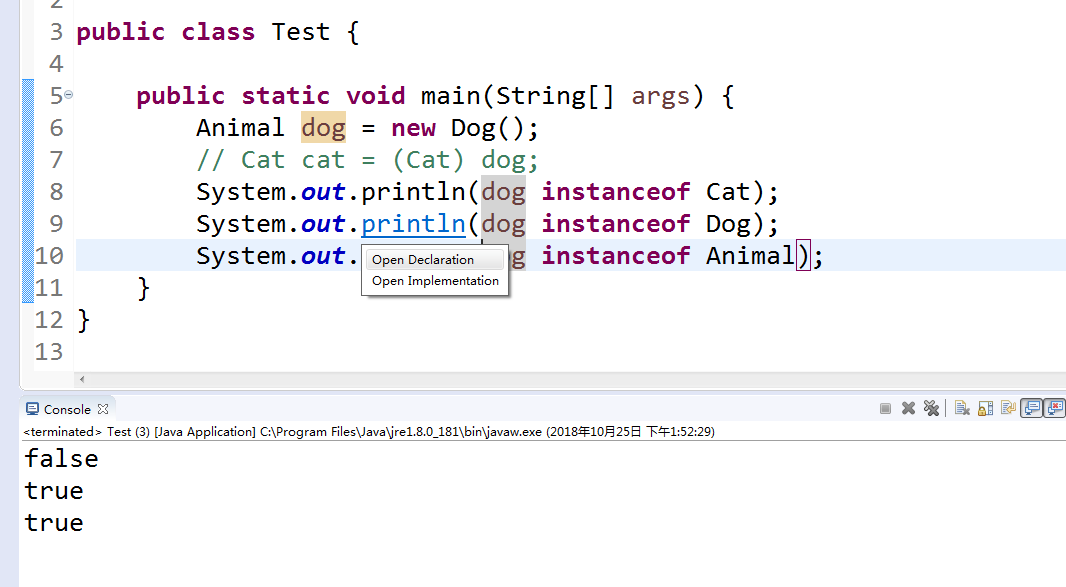
与向上转型相反,把父类转向子类就是向下转型.在使用向下转型时必须保障对象的类型和要要转成的类型一致,不然会报错



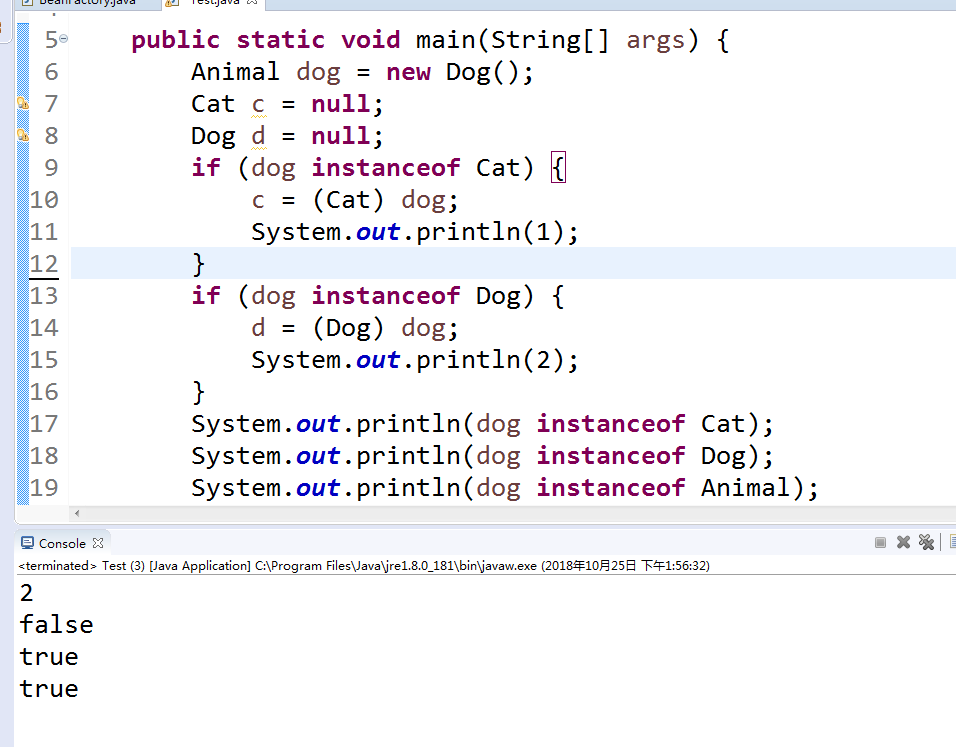
## Instance Of 运算符

InstanceOf是一个很特殊的存在.在Java中被归类到运算符类型.作用是验证对象是不是某个类型的对象

该运算符的结果是布尔类型.代表是与不是.



一般情况下当我们想要做向下转型时,都需要先用instanceof来验证一下类型是否匹配.

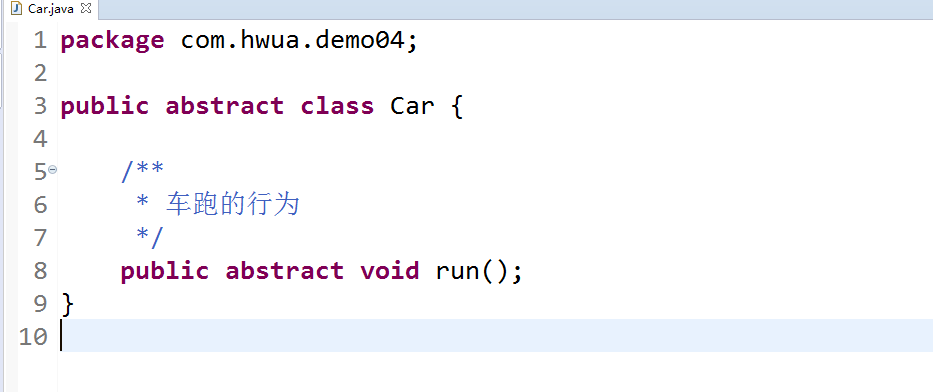


# 抽象

在使用多态时,有没有发现一个现象:一旦子类重写了父类的方法.那父类的方法基本不被使用.

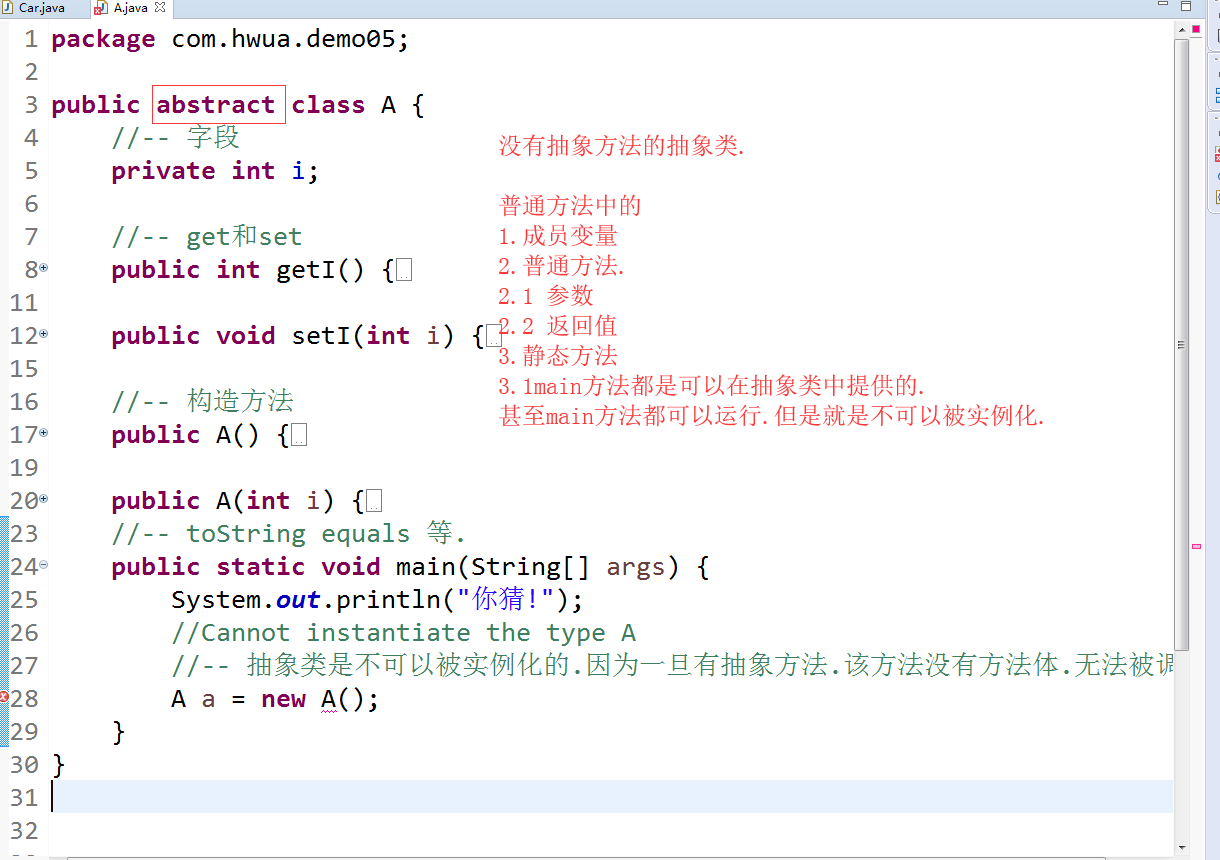
基于这一点我们思考可否只声明某类事物具备某个行为,但是在声明时不提供任何具体的实现(方法体),由各自的实现类(子类)自己提供具体的实现方式.这种行为我们把它叫做抽象.把行为从类中抽象出来.

在Java中使用关键字:abstract来代表抽象.如果一个类被该关键字修饰了,那这个类被叫做抽象类.



关于抽象有一个注意点:有抽象方法的类一定是抽象类,但是抽象类不一定有抽象方法.

抽象类也是类.所以一个类所具备的功能,抽象类都拥有,唯独没有的就是对象.即抽象类是不可以被实例化的



# 接口

具有相同特征的个体多了,我们总结出了类.类多了把具有共性的部分抽取出来组成父类.考虑子类重写父类的行为,我们把共性特征抽取出来形成抽象.当不同类型的抽象,都具备相同的行为时,我们把该行为在进行抽象化形成接口.

在Java中接口使用关键字interface.比如飞机可以飞,鸟可以飞.问是让飞机继承鸟.还是鸟继承飞机.

为了解决这个问题,我们引入接口.接口只管声明行为和类型无关.

