继承/抽象类/接口/多态

公告

2019-12-19 今天学习内容：

- 继承/抽象类/接口/多态
  - 继承
    - 定义
    - 阻止继承
    - 类型转换
    - 受保护访问
    - 所有类的超类 Object
  - 抽象类
  - 接口
    - 定义
    - 接口特性
    - 标记接口
    - 对象克隆
  - 多态
  - 常见面试问题
- 课后练习

继承

继承（inheritance），就是基于已经存在的类构造一个新类。新类继承了已存在类的方法和域，并在此基础上还可以添加新的方法和域。

在 Java 中使用 `extends` 关键字来表明一个类继承自另一个类，已存在的类称为超类（superclass）、基类（base class）或父类（parent class）；新创建的类称为子类（subclass）、派生类（derived class）或孩子类（child class）。

继承的语法结构：

```
class SubClass extends ParentClass {
    子类新定义的方法和域;
}
```

那什么时候需要用到继承，我们通过一个例子来说明：一个公司中普通员工和部分经理都属于公司雇员，他们有很多相同的地方，比如：有姓名、入职时间、要领薪水等；而经理除了具有普通员工具有的特性外，如果本部门完成了预期的业绩目标，那么经理还可以得到额外的绩效奖金。这时我们就把普通雇员设计为一个类 `Employee`，而把经理定义为另一个类 `Manager`，它继承自 `Employee`，因此具有了普通雇员的方法和域，然后只需要再加上经理具有的额外的一些方法和域。如下代码所示：

```
public class Employee {
    private String name;
    private Date hireDay;
    private Integer salary;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Date getHireDay() {
        return hireDay;
    }

    public void setHireDay(Date hireDay) {
        this.hireDay = hireDay;
    }

    public Integer getSalary() {
        return salary;
    }

    public void setSalary(Integer salary) {
        this.salary = salary;
    }

    @Override
    public String toString() {
        StringBuilder str = new StringBuilder();

        str.append("员工: ");
        str.append(getName());
        str.append(", 入职时间: ");
        str.append(getHireDay());
        str.append(", 薪水: ");
        str.append(getSalary());

        return str.toString();
    }
}
```

```
public class Manager extends Employee {
    private Integer bonus;

    public Integer getBonus() {
        return bonus;
    }
```

```
    public void setBonus(Integer bonus) {
        this.bonus = bonus;
    }

    @Override
    public Integer getSalary() {
        return super.getSalary() + bonus;
    }
}
```

```
Employee employee = new Employee();
employee.setName("普通员工");
employee.setHireDay(new Date());
employee.setSalary(100);
System.out.println(employee.toString());

Manager manager = new Manager();
manager.setName("经理");
manager.setHireDay(new Date());
manager.setSalary(120);
manager.setBonus(30);
System.out.println(manager.toString());
```

上述代码执行结果：

```
员工：普通员工，入职时间：Thu Dec 19 17:45:49 CST 2019，薪水：100
员工：经理，入职时间：Thu Dec 19 17:45:49 CST 2019，薪水：150
```

像上述示例代码，我们可以在子类中增加域、增加方法或者覆盖父类的方法，但是不能覆盖父类的域，也不能删除继承过来的任何域和方法。如果需要父类的方法可以使用 `super` 关键字。

子类在构造的时候，执行子类构造器之前必定会先执行父类的构造器，如果子类的构造器中没有显式的调用父类的构造器，系统将自动调用父类的没有参数的默认构造器，子类调用父类构造器时使用 `super` 关键字，且语句只能出现在子类构造器方法中的第一句。

由子类声明的子类对象也可以赋值给父类定义的对象变量，这种一个对象变量可以指向多个实际类型的现象我们称为多态（polymorphism）。在执行的时候能够自动地选择实际所属对象的方法进行执行，这种现象叫动态绑定（dynamic binding）。例如：

```
Employee employee = new Employee();
employee.setName("普通员工");
employee.setHireDay(new Date());
employee.setSalary(100);
System.out.println(employee.toString());

Employee manager2 = new Manager();
manager2.setName("经理2");
```

```
manager2.setHireDay(new Date());
manager2.setSalary(120);
System.out.println(manager2.toString());
```

注意： Java 不支持多重继承，一次只能继承自一个类；但继承不于一个层次，我们可以再创建一个类表示总经理，继承自前边的 `Manager` 类：

```
public class MasterManager extends Manager {
    总经理特有的方法和域;
}
```

由一个类派生出来的所有类的集合称为`继承层次（inheritance hierarchy）`。

在继承层次中，从某个特定的子类到其祖先的路径被称为该类的`继承链（inheritance chain）`。

阻止继承

如果我们定义一个类后，不希望人们再给它定义子类，这时我们可以使用 `final` 修饰符，表明这个类不能被继承。

如果类中的某些方法我们不希望被子类进行覆盖也可以声明为 `final`。

类型转换

类型转换，顾名思义就是将一个类型转换成另一个类型，我们可以将子类对象直接赋值给父类定义的对象变量而不用做任何额外的事情，就像前边将 Manager 对象赋值给 Employee 定义的变量。

如果我们需要将父类对象赋值给子类定义的变量，这时就需要进行强制类型转换，例如：

```
Manager manager = (Manager) employee;
```

在进行强制类型转换时，运行时会检测这个对象的实际类型是否是被转换类型的对象，如果不是则会抛出 `ClassCastException` 异常。

```
Object object = new Date();
Date date = (Date) object;
Manager manager = (Manager) object;
```

综上所述，我们总结类型转换的规则如下：

- 只能在继承层次内进行类型转换
- 在将父类转换成子类之前，应该使用 `instanceof` 进行类型检查

受保护访问

父类中定义的 private，只能在父类方法中被访问，子类中是不能直接访问定义为 private 的域。

如果我们希望父类中某些域或者方法允许被子类访问，只需要将这些域或方法声明为 protected。

我们再来看一下控制权限访问的 4 个访问控制修饰符：

- `private` 仅对本类可见
- `public` 对所有类可见
- `protected` 对本包和所有子类可见
- `默认(没有修饰符)` 对本包可见

所有类的超类 Object

Object 类是 Java 中所有类的始祖，每个类都是由它扩展而来，并且不需要像下边这样显式的指明：

```
class YourObject extends Object {
}
```

在 Java 中，除了基本数据类型，其余所有类型（包括数组、基本类型数组），都是扩展于 Object 类。

equals 和 hashCode方法

Object 类中的 equals 方法用于检测一个对象是否等于另外一个对象，默认判断的是两个对象是否具有相同的引用。

散列码（hash code）是由对象导出的一个整数值，散列码是没有规律的，如果是两个不同的对象，那么它们的 hashCode 方法返回的数值基本上也不会相同。

如果重新定义 equals 方法，就必须重新定义 hashCode 方法，以便可以将对象插入到散列表中。

抽象类

在类的继承层次中，位于上层的类更具有通用性，甚至更加抽象。有时我们希望在父类中只进行功能定义而不实现它，具体的实现由子类来实现。这时我们只需要将方法声明为 `abstract`，这种方法就叫做抽象方法。由 abstract 可以没有方法体，子类继承它时则必须实现该方法。

包含一个或多个抽象方法的类必须被声明为抽象类，也使用 `abstract` 修饰符；当然类不包括抽象方法时也可以被声明为抽象类。

继承抽象类的子类也可以被定义为抽象类，这时可以不实现父类中的抽象方法；但如果未被定义为抽象类，则子类必须实现父类中所有的抽象方法。

抽象类不能被实例化，也就是不能用于 new 操作符。

接口

在 Java 语言中，接口不是类，是一种新的引用类型，使用关键字 `interface` 进行定义，它的成员可以有类、接口、常量和方法。接口不能实例化，也就是不能用于 new 操作符。

下面是一个接口声明示例：

```

```

```
public interface Interface {
    class InnerClass {
    }

    interface InnerInterface {
    }

    int MAX_COUNT = 100;

    void method();
}
```

下面我们来看一下接口有哪些特性：

- 接口不是类，不能使用 new 操作符进行实例化。
- 接口变量必须引用实现了接口的类对象。
- 可以使用 `instanceof` 来检查一个对象是否实现了某个特定的接口。
- 接口不能包含实例域和静态方法，但可以包含常量；接口中的方法都自动被设置为 `public`，而接口中的域则被自动设置为 `public static final`。
- 接口中定义的方法不能有方法体。
- 类通过 `implements` 关键字来实现一个或多个接口。
- **特别注意：Java 8 版本开始对接口做了以下增强**
    - 在方法声明前添加 default 关键字时，则可以为该方法编写一个默认实现，这个特征叫默认方法或者扩展方法；但是默认方法不能覆盖 Object 中的方法，却可以重载 Object 中的方法。
    - 接口里可以声明静态方法，静态方法必须有方法体。

下面我们使用代码来演示验证接口上述这些特性。

接口的作用： 接口是对行为的抽象，实现来了约定和实现相分离，它主要解决的问题是：可以降低代码之间的耦合，提高代码的可扩展性。

标记接口（tagging interface）： Java 中有一类比较特殊的接口，接口中没有任何方法，使用它的唯一目的就是可以用 `instanceof` 来进行类型检查，看某个类是否实现类特定的接口。例如：

- `java.util.RandomAccess` 用来标记对象是否支持快速随机访问，比如：java.util.ArrayList 实现类此接口。
- `java.lang.Cloneable` 对象是否支持拷贝

下边示例如果没有实现 `Cloneable` 接口，调用 Object 类的 clone 方法就会抛出 CloneNotSupportException 异常：

```
public class CopyTest implements java.lang.Cloneable {
    String name = "";

    public CopyTest(String name) {
        this.name = name;
    }

    public String getName() {
```

```
        return this.name;
    }

    @Override
    public CopyTest clone() throws CloneNotSupportedException {
        return (CopyTest) super.clone();
    }
}

CopyTest copy = new CopyTest("标记接口作用验证").clone();
System.out.println(copy.getName());
```

对象克隆

在 Java 中，引用类型变量指向的是一个对象的引用，因此将引用变量直接赋值给另一个变量时，两个变量指向的是同一个对象的引用，因此当对象发生变化时，会对两个变量都产生影响。

当我们需要创建一个新的对象，它的最初状态和原对象的状态一样，这时我们就需要用到对象克隆，也称为对象拷贝。Object 类提供了一个 protected 的 clone 方法来实现拷贝，用户自定义的类中必须覆写它后，才能被调用。Object 类的 clone 默认提供的是浅拷贝，对于对象中的数据域是基本类型或不可变对象时是没有问题的，但对于对象中的子对象的引用，拷贝的是子对象的引用。

- 浅拷贝
- 深拷贝

多态

前边继承部分我们已经讲过多态的概念：一个对象变量可以指向多个实际类型的现象我们称为多态（polymorphism）。而在执行的时候能够自动地选择它实际所属对象的方法进行执行，这种现象叫动态绑定（dynamic binding）。

通俗来说，就是一个父类对象变量，可以指向父类对象，也可以指向它的所有子类对象；而在调用签名相同的方法的时候，运行时会根据该变量实际的类型来决定调用谁的方法。

下边我们就来看一下调用对象方法的执行过程：

1. 编辑器查看对象的声明类型和方法名，查找所有同名的方法（包括继承链上的父类）作为候选方法。
2. 根据参数类型查找名字和参数类型完全匹配的方法，如果未找到或者经过类型转换后有多个方法匹配时则报错。
3. 如果是 private 方法、static 方法、final 方法或者构造器，那么编辑器就知道应该调用哪个方法，我们称这种调用方式为静态绑定（static binding）。与此对应，调用的方法依赖于隐式参数 `this` 的实际类型，并在运行时实现动态绑定。
4. 程序运行时，采用动态绑定调用方法时，JVM 虚拟机会调用与所引用对象的实际类型最合适的那个类的方法。

每次调用都要进行搜索，时间开销很大，因此，虚拟机预先为每个类创建了一个方法表（method table），其中列出了所有方法的签名和实际调用的方法。这样，每次调用方法时，虚拟机仅查询这个方法表就行了。

常见面试问题

抽象类和接口的定义、特性与区别

抽象类和接口主要是解决什么编程问题

抽象方法可以被 private 修饰吗？为什么

接口能不能有方法体?

课后练习

1. 自己编写练习一下课程中的示例和练习。
2. 根据课程内容回答常见面试问题。