"如果你还没有对象，你现在就可以new一个出来"——中泰姐姐

面向对象的基本概念 Object Oriented Programming（oop）

面向过程and面向对象

面向过程：比如c这样的语言就是面向过程的语言

面向对象：C++，Java是面向对象的语言

|**面向过程**|思想思考问题时，是一种**执行**的思维，我们首先思考"怎么按步骤实现？"并将步骤对应成方法，一步一步，最终完成。比如说蛋炒饭的制作，这是一个一步到位，浑然一体的过程。

而|**面向对象**|思想思考问题，是一种**设计**的思维，首先是把解决整个问题的各个部分设计好，不需要自己亲力亲为，而是通过不同的对象来完成问题，举个栗子，番茄 脆皮鸡 饭，黑椒 烤肉 饭，这样分成了几个部分，我们拥有了一个 **XX XX 饭** 的**模板**，假如你不喜欢吃黑椒，那么将黑椒这一部分去掉换一个就行，而不是像蛋炒饭一样需要打倒重来。

> 栗子：
>
> 王者荣耀中 互相攻击的过程
>
> 面向过程方式思考：
>
> 1、player1控制英雄
>
> 2、player1控制英雄释放技能
>
> 3、实现技能特效
>
> 4、实现伤害血量扣除
>
> 5、player2控制英雄
>
> 6、player2控制英雄释放技能
>
> 7、实现技能特效
>
> 8、实现伤害血量扣除
>
> 面向对象的思考：
>
> （上面的过程中有很多的共性，我们把这些共性集中起来）
>
> 1、玩家控制系统（player1和player2的控制）
>
> 2、英雄系统（包括释放技能，血量展示，移动等）
>
> 3、展示系统（包括地图、技能特效等）
>
> 4、攻击判断系统（对战斗的攻击进行处理）
>
> 我们把繁琐重复的步骤，通过行为、功能、特点来进行**分类构造**，最后抽象出一个模板，这就是面向对象的思维，通过这样4个抽象出来的系统，我们可以构建出两个英雄的战斗，也可以快速构建其他所有的战斗，而不是每一场战斗都重新开始面向过程的1-8的过程。

类和对象的定义

面向对象的思想是如何在java展现的呢？ 就是通过**类**和**对象**。

类是一组相关的属性和行为的集合。是一个抽象的概念。

对象是该类事物的具体表现形式。具体存在的个体。

类

成员变量：成员变量所属于对象。所以也称为实例变量。

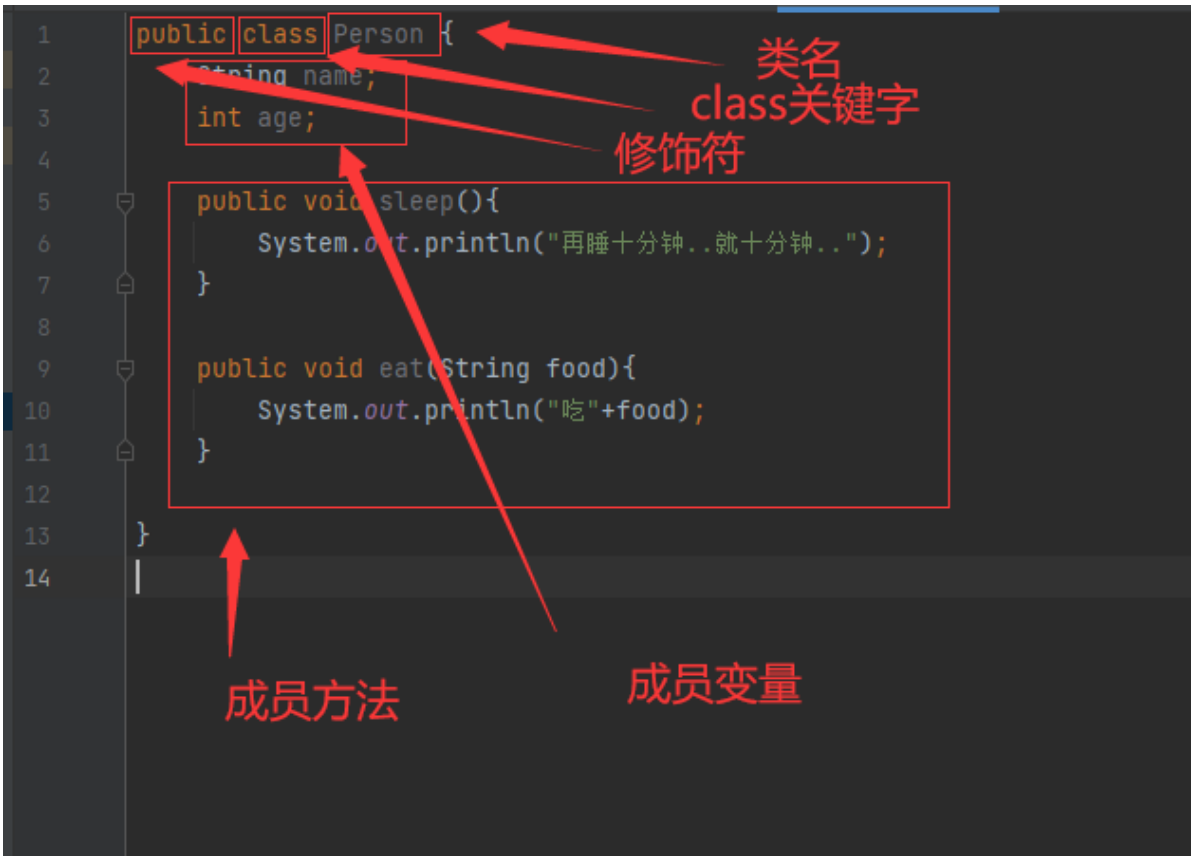
成员方法：

```
public class Person {
    String name;
    int age;

    public void sleep(){
        System.out.println("十分钟..再十分钟我就睡觉了");
    }

    public void eat(String food){
        System.out.println("吃"+food);
    }

}
```



类就是对一些具有**共性特征**，并且**行为相似**的个体的描述。

比如张三和李四都有姓名、年龄等一些**属性**，并且两人都能够进行吃饭、睡觉等**相似的行为**。

由于这两个人具有这些共性的地方，所以我们把它抽象出来，定义为一个**类——人类**，而张三、李四正是这个类的一个实例化的个体（**对象**），而个体才是真正具体的存在，光提到人类，你只知道应该有哪些属性行为，但你不知道他具体的一些值，比如你知道他属于“人类”所以他应该有姓名，年龄等属性，但你并不知道他具体叫什么，年龄多大了。而张三和李四这两个具体的对象，却能够实实在在的知道张三今年30岁了。

> 类是一种抽象的类型，是一个不可分割的部分，你可以说一个人有名字，年龄，性别，但是你不能把一个人拆开。
>
> 与之相对比的就是结构体，结构体是紧紧把数据捆起来了，并没有达到一个抽象的作用

对象

对象的创建：

```
//类名 对象名 = new 类名();
Person zhangSan = new Person();
//这里的Person zhangSan = new Person();其实是拆成两步的
Person zhangSan2;//声明
zhangSan2 = new Person();//赋值
```

> 想一想`zhangSan`到底是什么，对象又是存放在哪个地方的
>
> 类是引用类型，使用堆存储，其实zhangSan是一个指向堆空间的地址

对象的使用：

```
//对象名.成员变量
//对象名.成员方法()
zhangSan.sleep();
zhangSan.name = "张三";
System.out.println(zhangSan.name);
```

这样是不是感觉很简单，你已经学会了如何创建使用对象了，快去new一个属于自己的对象！

访问修饰符

访问修饰符	本类	同包	子类	不同包
private	√			
默认(default)	√	√		
protected	√	√	√	
public	√	√	√	√

private:私有的，被private修饰的类、方法、属性、只能被本类的对象所访问。

> 我什么都不跟别人分享。只有自己知道。

default:默认的，在这种模式下，只能在同一个包内访问。

> 我的东西可以和跟我一块住的那个人分享。

protected:受保护的，被protected修饰的类、方法、属性、只能被本类、本包、不同包的子类所访问。

> 我的东西我可以和跟我一块住的那个人分享。另外也可以跟不在家的儿子分享消息，打电话。

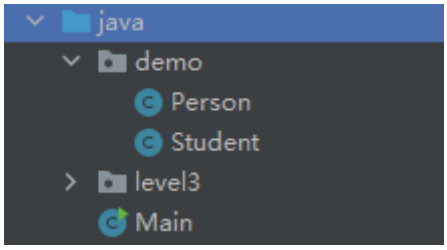
public:公共的，被public修饰的类、方法、属性、可以跨类和跨包访问。

> 我的东西给大家随便用。

包?

包的作用就是管理Java文件，小写命名，可以用`.`隔开

```
package demo;
```



可以看到，java这个文件夹是蓝色的，蓝色是源代码文件夹，这个目录下就是默认的文件夹

那么在demo文件夹下，每一个类就会有一个 package demo;

如果想在其他地方使用demo这个文件夹的类，就需要使用下列代码来导入

```
import demo.Person;
```

或者说使用`demo.*`来导入这个文件夹下所有的类

(通常我们会使用alt+回车来自动导入)

方法的定义

方法就是一段可重复调用的代码段。在C语言中有函数，在java中我们有方法。(两者有细微的区别)

为什么要使用方法呢，因为这样可以避免我们成为CV工程师，在一个项目里面会有很多重复的代码在使用，这样会导致**可读性**变差，同时**可维护性**也会变差，因为如果要有改动的话，每一个地方都需要改。通过使用方法，把重复使用的代码写为一个方法，然后调用方法就是在执行方法内的代码了，如果有改动就只需要改动方法内的代码一次就行。

方法的命名以及一些类和包的命名规则:

在Java中我们使用驼峰命名法，在定义方法、变量、包名的时候，都是第一个单词首字母小写，之后每一个单词的首字母大写，例如doHomeWork(),而类的命名则是每个单词首字母都是大写。一个优秀的开发者，都需要遵循相应的开发规范。

方法的结构:

```
public int exam(String subject){
System.out.println("考了"+subject);
return 60;
}
修饰符 返回值 方法名(类型1 参数1.类型2 参数2....){
    方法体
    return 返回值(如果有返回值的话)
}
//如果返回值为void的话，就是无返回值，不需要return
```

> c函数和Java方法的区别
>
> “函数”是一段实现某种“功能”的代码，函数的操作是对输入数据的处理。
>
> “方法”也是一段完成某项功能的代码，也通过名字来进行调用，但它依赖于某个特定的对象。
>
> (细心的同学可以发现，Java的main函数是在一个类里面的)
>
> 所有传递给函数的数据都是显式传递的。而类里面方法的参数传递可以是隐式的，它可以直接操作类内部的数据。
>
> 我们可以说“调用对象X的Y方法”，而不能说“调用Y方法”。
>
> 简单来说，方法和对象相关；而函数和对象无关。

this关键字

在了解this关键词之前，我们需要先知道什么是局部变量，什么是成员变量（此处仅讨论对象的）

成员变量：成员变量无需显示初始化，只要为一个类定义了实例变量，系统就会在创建该类的实例(对象)时进行默认初始化。

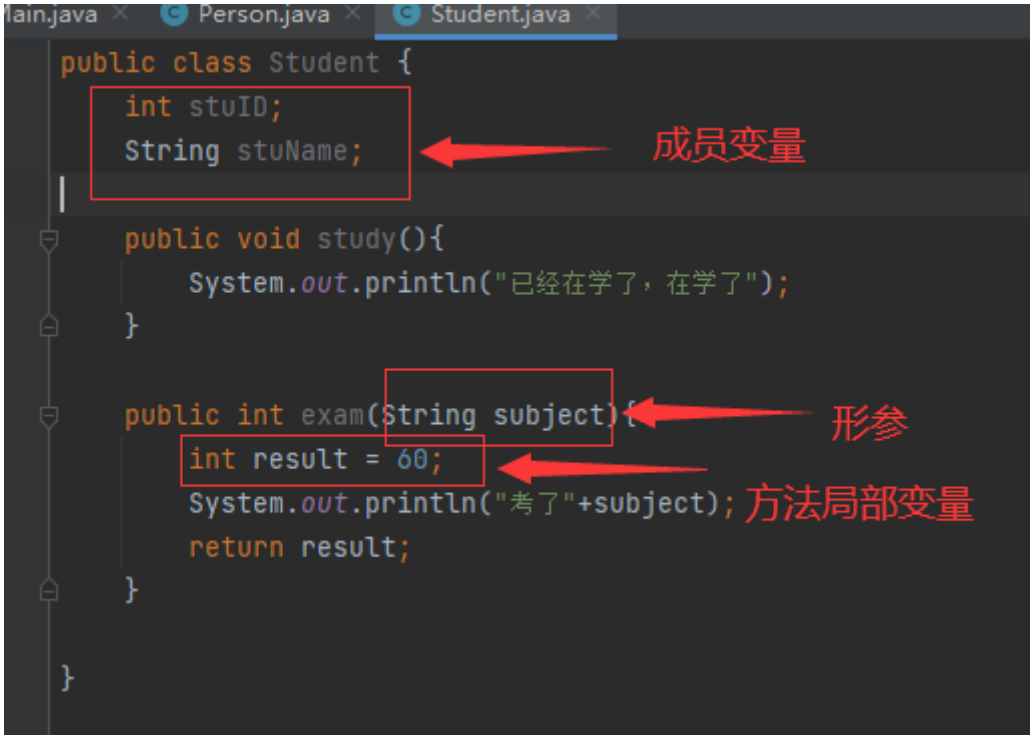
只要实例存在，对象就可以访问成员变量　对象.成员变量

局部变量：一般分为三种情况

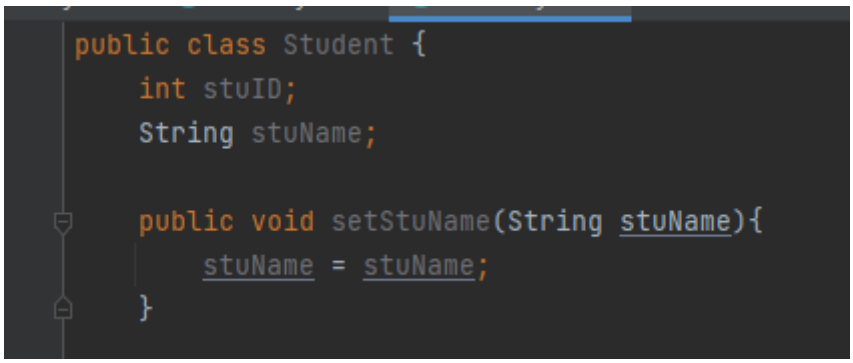
形参：在定义方法时定义的变量，形参的作用域在整个方法中都有效

方法局部变量：在方法体内定义的局部变量，它的作用域是从定义该变量的地方生效，到该方法结束时失效

代码块局部变量：这个局部变量的作用域从定义该变量的地方生效，到该代码结束时失效。(暂时忽略，用得少)



在Java中允许局部变量与成员变量同名，那么就会出现这样的情况




```
public class Student {
    int stuID;
    String stuName;

    public void setStuName(String stuName){
        stuName = stuName;
    }
}
```

当我想通过一个方法改变我的参数的时候，就会出现这样的问题

那么此时我们就可以使用`this`关键字来解决这个问题

`this`:是对自身对象的一个地址引用，就是指自身。

1、通过this调用另一个构造方法，用法是this(参数列表)，这个仅仅在类的构造方法中，别的地方不能这么用。
2、函数参数或者函数中的局部变量和成员变量同名的情况下，成员变量被屏蔽，此时要访问成员变量则需要用“this.成员变量名”的方式来引用成员变量。当然，在没有同名的情况下，可以直接用成员变量的名字，而不用this，用了也不为错。
3、在函数中，需要引用该函所属类的当前对象时候，直接用this。

这样我们就可以写成这个样子解决问题

```
public class Student {
    int stuID;
    String stuName;

    public void setStuName(String stuName){
        this.stuName = stuName;
    }
}
```

super，一个与this相似，但不相同的东西。

1、在子类构造方法中要调用父类的构造方法，用“super(参数列表)”的方式调用，参数不是必须的。同时还要注意的一点是：“super(参数列表)”这条语句只能用在子类构造方法体中的第一行。

2、当子类方法中的局部变量或者子类的成员变量与父类成员变量同名时，也就是子类局部变量覆盖父类成员变量时，用“super.成员变量名”来引用父类成员变量。当然，如果父类的成员变量没有被覆盖，也可以用“super.成员变量名”来引用父类成员变量，不过这是不必要的。

3、当子类的成员方法覆盖了父类的成员方法时，也就是子类和父类有完全相同的方法定义（但方法体可以不同），此时，用“super.方法名(参数列表)”的方式访问父类的方法。

构造方法的定义和使用

在刚刚的this中，我们好像看见了构造方法这个名词，那么这是个什么东西呢

解决问题场景：我们对象创建之后，有成员属性，那么我们每次都需要给成员变量赋值吗，这样会不会有疏忽，然后出现了一个没有名字的人？

于是有了构造方法

构造方法：方法名和类名相同，没有返回值类型

每一个类都会有一个默认的无参构造方法(不会显示，编译的时候会默认加上)

当写了一个构造函数，默认无参构造就会消失

会在对象创建的时候执行

使用：

```
public class Person {
    public String name;
    public int age;
    //这个方法就是构造方法
    public Person(){

    }
}
```

那么，我们可以利用构造方法来达到一些操作，比如说给成员属性赋初值

```
public class Person {
    public String name;
    public int age;

    public Person(String name,int age){
        this.name = name;
        this.age = age;
    }
}
```

那么我们创建的时候就需要如下写，在Person()括号里面传入参数给构造方法

```
Person zhangSan = new Person("张三",18);
```

PS:

构造方法名称必须和类名一致

构造方法没有返回值，也不能return

每个类都有构造方法，如果代码里面没有构造方法，那是编译的时候直接生成了一个无参构造

方法的重载和重写

方法的重载: 就是方法名称相同，但参数的类型和参数的个数不同。通过传递参数的个数及类型的不同可以完成不同功能的方法调用。（返回值不会算成判断方法重载的要素）

```
    public void eat(String food){
        System.out.println("吃"+food);
    }

    public void eat(){
        System.out.println("喝西北风");
    }

    public void eat(String food,int heat){
        System.out.println("吃了包含"+heat+"热量的"+food);
    }
```

方法的重写:

1、父类与子类之间的多态性，对父类的函数进行重新定义。如果在子类中定义某方法与其父类有相同的名称和参数，我们说该方法被重写 (Overriding)。在java中，**子类可继承父类中的方法，而不需要重新编写相同的方法。**但有时子类并不想原封不动地继承父类的方法，而是想作一定的修改，这就需要采用方法的重写。方法重写又称方法覆写

2、若子类中的方法与父类中的某一方法具有相同的方法名、返回类型和参数表，则新方法将覆盖原有的方法。如需父类中原有的方法，**可使用super关键字，该关键字引用了当前类的父类;**

3、子类函数的访问修饰权限不能少于父类的.

匿名对象的使用

创建匿名对象：

```
//创建匿名对象
new Person();
//使用
needPerson(new Person());
```

简单的理解就是 没有名字的对象

一般使用于 仅仅调用一次的时候

匿名对象调用完就是垃圾，可以被垃圾回收器回收，并且这样写比较简化。

面向对象的三大特性（封装 继承 多态）

刚刚是还是起飞过程，现在是飞行过程

封装

将类的某些信息隐藏在类内部，不允许外部程序直接访问，而是通过该类提供的方法来实现对隐藏信息的操作和访问。

作为一个黑盒，仅提供愿意暴露的部分

原则：

将不需要对外提供的内容都隐藏起来

把属性隐藏，提供方法对其访问

封装的好处：
1，隐藏类的实现细节
2，方便加入控制语句
3，方便修改实现
4，只能通过规定方法访问数据

封装的实现：

封装前：

```
public class Person {
    String name;
    int age;

    public Person(String name,int age){
        this.name = name;
        this.age = age;
    }
}
```

封装后（简单封装）：

```
public class Person {
    private String name;
    private int age;
```

```
    public Person(String name,int age){
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }

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

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }
}
```

我们做了什么呢，我们将成员属性变为了私有`private`，然后使用方法来给成员变量赋值或者得到他

首先我们失去了`zhangSan.age`这样的权限，而转为使用`zhangSan.getAge()`,`zhangSan.setAge(xx)`的方法

举个栗子（隐藏类的实现细节）：

如果说学生有作业，那么老师要收学生的作业，老师只需要关注作业是否完成，而不需要关注作业是怎样完成的，这个完成作业的过程被类隐藏起来了

```
public HomeWork doHomeWork(){
    //1.自己独立完成
    //2.大家相互帮助
    //3.....
    return HomeWork;
}
```

举个栗子（方便加入控制语句）：

```
public String getStudy(){
    if(this.name == "王中泰"){
        return "往死里学！";
    }else if(this.name == "廖俊枭"){
        return "开躺！";
    }else{
        return "在学了在学了！"
    }
}
```

举个栗子（方便修改实现）：

如果说食堂想给每一种食物便宜一块钱，但是他又不想去修改每一种食物，那么就可以

```
class Food{
    private int price;
    //修改前
    public int getPrice(){
        return price;
    }
    //修改后
    public int getPrice(){
        return price-1;
    }
}
```

举个栗子（只能通过规定方法访问数据）：

如果说我们给一个人设定年龄，那么人的年龄肯定在一个合理的范围内，我们就可以限制这个范围

```
public void setAge(int age){
    if(age >= 0 | age <= 150){
        this.age = age;
    }else{
        System.out.println("年龄超出范围");
    }
}
```

我们使用封装，来将类和对象的使用更加规范、简单、安全，增加拓展性

看完了封装，你可能已经领悟到了对象的一部分美妙，让我们再仔细品味一下。

> 可能对于现在的你们来说，程序一个程序的组成就是 数据 和 如何使用数据，你对数据的某种行为和意图，便可以定义为一个方法，使属性(数据)和方法组成了一个类，将数据隐藏起来，而仅暴露行为，这样就可以把一个程序通过一个个类来组成，可以脱离数据和一堆奇怪操作的深坑，而通过对抽象之后生成的类来操作，比如你们用的`Scanner`，你可以不用知道他是怎么达成功能的，你只需要调用他。

继承

继承是子类对父类的拓展，延伸

Java没有多继承，只有单继承（只能有一个爹）

不同的叫法：

父类：基类，超类

子类：派生类

继承关键字 使用

```
public class Student extends Person {

}
```

这样Student就继承了Person，拥有了Person的成员变量和成员方法，

```
public class Student extends Person {
    int stuID;

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

    public void study(){
        System.out.println("已经在学了，在学了");
    }

    public int exam(String subject){
        int result = 60;
        System.out.println("考了"+subject);
        return result;
    }

}
```

我们就可以使用父类中的成员变量和方法

```
new Student().getName();
```

在继承了父类之后，我们可以对父类的方法进行重写，可以使用super来给父类传参

```
public class Student extends Person {
    int stuID;

    public Student(String name,int age){
        super(name,age);
    }
    //这里的@Override(注解)是一个重写的标志
    @Override
    public void eat() {
        System.out.println("我是学生，人在食堂");
        super.eat();//这里是执行父类里面的eat()方法，你也可以删掉不用父类的
    }
}
```

这里的super都是相当于调用父类里面的东西

这样使用继承，会简化很多代码的编写，避免很多的不必要的代码，

多态

对象可以在特定的情况下，表现不同的状态，从而对应着不同的属性和方法

多态有对象向上转型和向下转型

向上转型：这里的向上转型 指得是由子类向上转型为父类

```
Person zhangSan = new Student();
```

举个栗子：

你是一个大学生，那么你必然上过高中，你有高中的知识，你在知识上是高中生的子类，那么这个意思就是说高中生能做的题你也能做

子类是继承父类的，有比父类更强大的功能（更多的属性与方法），但是父类能做的事情你也能做，那么你就可以成为父类，但是这样的话，你子类中额外的方法和属性就不能够使用了

```
高中生 你 = new 大学生();
//当然在做高中题的时候你只能使用高中的知识
人 你 = new 大学生();
```

使用场景

```
public void doRedrockHomeWork(Person person){
    //这个方法，红岩的作业是人都能做！
}
//调用
Person zhangSan = new Student();
doRedrockHomeWork(zhangSan);//这里当然学生也可以做，因为他是继承人的
```

向下转型：

比如说你去打工，打完工回来读书还是得变成大学生

打工：是人都可以去

上大学：你得是大学生

```
Person zhangSan = new Student();//向上转型
work(zhangSan);
//打完工了
Student zhangSanStu = (Student)zhangSan;//向下转型
//此时你作为大学生的东西都能还使用
```

一个类在不同的情况的可以表现为不同的形态，有不同的身份，这就是多态

> 假如
>
> 你是一个地球人，中国人，重庆人，南岸区的人
>
> 你可以在这些身份之间转换，那么我们可以理解为
>
> 南岸的人 继承 重庆人
>
> 重庆人 继承 中国人
>
> 中国人 继承 地球人
>
> 然后你可以在这些之间转换，拥有不同身份应该有的不同能力
>
> 你是一个重庆人，但是你也肯定拥有身份证，是一个中国人
>
> 但是你说你是一个四川人，这样就有问题，因为你不具备四川人特有的能力（说四川话）

一个面向对象的应用栗子

把一个大象放进冰箱需要几步：

1、打开冰箱

2、把大象放进去

3、关闭冰箱

用面向对象的方法

```
冰箱{
    高;
    宽;
    温度;
    开门();
    关门();
    调节温度();
}
```

```
大象{
    高;
    宽;
    体重;
    进冰箱();
}
```

这样就可以做到使用抽象的对象来完成

```
冰箱 我的冰箱 = new 冰箱();
大象 我的大象 = new 大象();

我的冰箱.开门();
我的大象.进冰箱();
我的冰箱.关门();
```

这样大家就不需要仔细的，来个小白都能根据对象的方法名来调用

现在，老板突然说要把大象放进洗衣机里面

```
冰箱{
    高;
    宽;
    开门();
    关门();
}
```

```
大象{
    高;
    宽;
    体重;
    进冰箱();
    进洗衣机();
}
```

我连忙熬夜又CV出来了一个洗衣机类，把大象类也更新了

现在写了个调用来实现

```
冰箱 我的冰箱 = new 冰箱();
洗衣机 我的洗衣机 = new 洗衣机();
大象 我的大象 = new 大象();

我的冰箱.开门();
我的大象.进冰箱();
我的冰箱.关门();

我的洗衣机.开门();
我的大象.进洗衣机();
我的洗衣机.关门();
```

当我刚刚写完准备睡觉的时候，老板说，现在要把大象放进烤箱里面

我觉得CV太累了，气得直写了一个箱子类

```
箱子{
    高;
    宽;
    开门();
    关门();
}
```

啪的一下，写了几百个需求

```
冰箱 extends 箱子{
    温度;
    调节温度();
}
```

```
洗衣机 extends 箱子{

}
```

```
烤箱 extends 箱子{

}
```

```
XXX extends 箱子{

}
```

现在无论要装进什么东西，我都不用幸苦的CV了，只要继承了箱子这个父类，就一定有高宽，开门，关门这些东西。

但是我又发现大象里面的方法每次都要写，好苦恼

```
大象{
    高;
    宽;
    体重;
    进冰箱();
    进洗衣机();
    ...
}
```

我反手运用多态，写了一个通用的方法，以后就再也不用在大象类里面添加了

```
大象{
    高;
    宽;
    体重;
    进入(箱子 xx);
}
```

所有的继承了箱子的类，我们都认可他箱子这个身份，那么就可以使用这个进入的方法。

以前的我：一天CV10个需求，都写不完

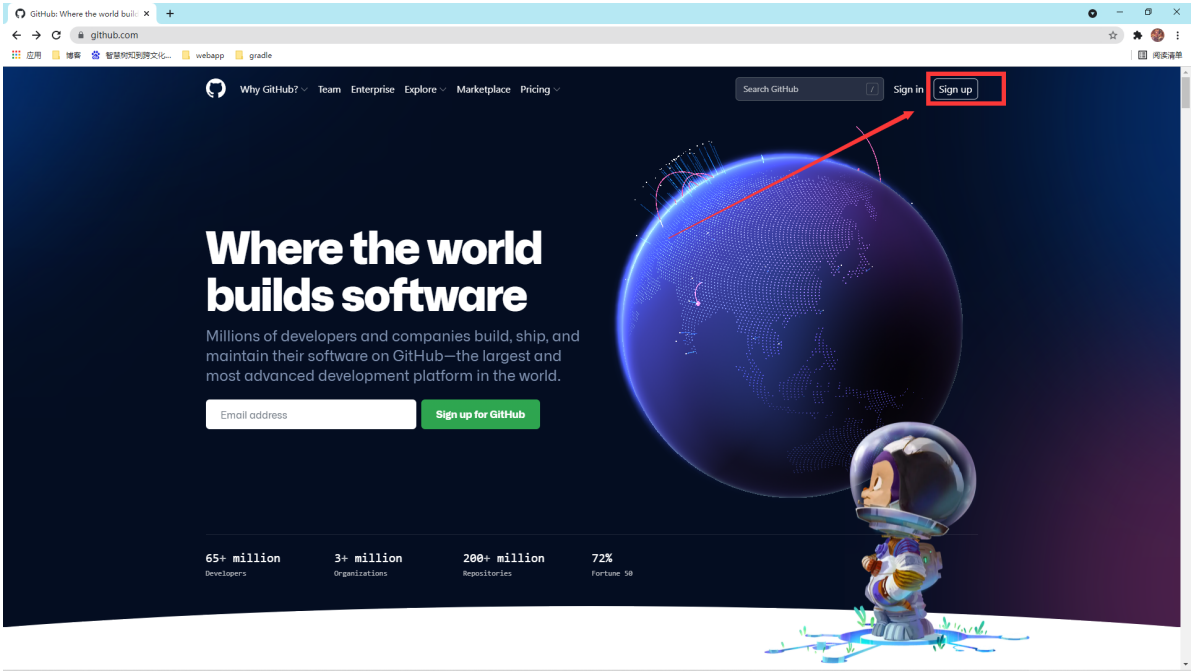
现在的我：一天写100个需求，还不带喘口气的

github的注册

github!,程序员的快乐天堂

github是一个代码托管网站，是一个基于git的网站，可以做版本管理，也可以公开自己的仓库，开源让其他人学习，寻找他人的公开库学习。

github



根据引导完成自己账号的注册

可以自行了解安装git，将自己的代码上传到github仓库