面向对象概述

面向对象概述

可重用性

- 减少软件中的重复代码，避免重复编程
- 软件要修改时，不用重复修改同样的问题

可扩展性

- 软件增加功能时，能够在当前系统结构上方便地创建新系统
- 增加新系统时，原有系统和结构尽量保持稳定不变

可维护性

- 用户需求变化时，只需要修改局部子系统的少量代码
- 修改代码尽量不牵一发而动全身，尽量不动多个子系统

结构化开发：面向过程(POP)

子系统按功能来划分

- 功能：输入数据，进行相应处理，输出结果
- 软件设计：功能划分，每个模块完成一个功能，功能内聚，功能间松耦合
- 程序主体方法
- 方法是最小的功能模块

不足：

- 软件设计难度大，设计阶段要考虑功能实现
- 当系统需求变化，也就是输入输出关系改变，功能实现要变化
- 当系统增加功能时需要改变代码架构

例：面向过程设计一个画图板

JAVA

```java
package paintpanel;

public class PaintPanel {
        public void drawLine() {
                System.out.println("Drawing a Line");
        }
        public void drawCircle() {
                System.out.println("Drawing a Circle");
        }
        public void drawSquare() {
                System.out.println("Drawing a Square");
        }

        public void selectShape(char shape) {
                switch(shape) {
                        case'L':
                        drawLine();break;
                        case'C':
                        drawCircle();break;
                        case'S':
                        drawSquare();break;
                        }
                }

        public void checkShape(char shape) {
                if(!(shape == 'L'|| shape =='C'|| shape =='S')) {
                        System.out.println("Error shape");
                        return;
                }
        }

        public static void main(String[] args) {
                System.out.println(args[0]);
                char shape = args[0].charAt(0);
                PaintPanel pp = new PaintPanel();
                pp.checkShape(shape);
                pp.selectShape(shape);
        }
}
```

面向对象开发(OO)

- 把软件系统可看成是各种对象的集合
- *对象就是最小的子系统*，对象组合成更复杂的子系统，接近人类描述世界的过程
- 需求变动，导致功能变动，但*功能的执行者(对象)一般不变；
- 对象包括属性（数据）和行为（方法）
  - 对象把*数据及方法的具体实现方式*一起封装起来，这使得方法和与之相关的数据不再分离
  - 提高了每个子系统的*相对独立性*，从而提高了软件可维护性

- 支持*封装、抽象，继承和多态*
- *自底而上*的抽象
  从问题领域的陈述入手，建立正确的对象模型
  把问题领域的事物抽象为具有特定属性和行为的对象
  把具有相同属性和行为的对象抽象为类
  当多个类中具有一些共性(相同的属性和行为)，把这些共性抽象到父类中
- *自项向下*的分解
  对于每个对象进一步分解，包含有哪些对象组成，每个对象有什么属性和行为
  由于分解过程有具体对象为依据，分解过程比较明确，相对容易
  既能控制系统的复杂性，又避免了分解的困难和不确定性

提高软件性能的途径

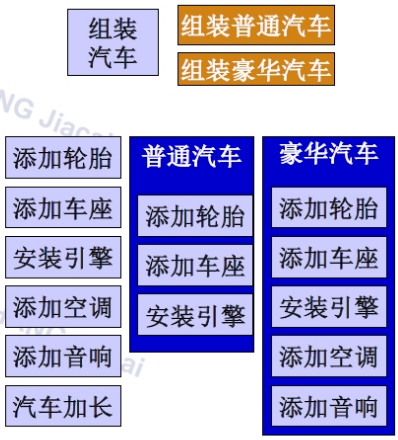
- 结构稳定性
- 可扩展性
- 内聚性
- 可组合性
- 松耦合

代码复用

◈ 代码复用是面向对象的首要优点

◈ 预先写好的代码（类），可在整个工程中使用，如使用继承等面向对象技术可进一步减少代码

◈ 编写多个小程序胜过写一个大程序，面向对象提倡使用多个类



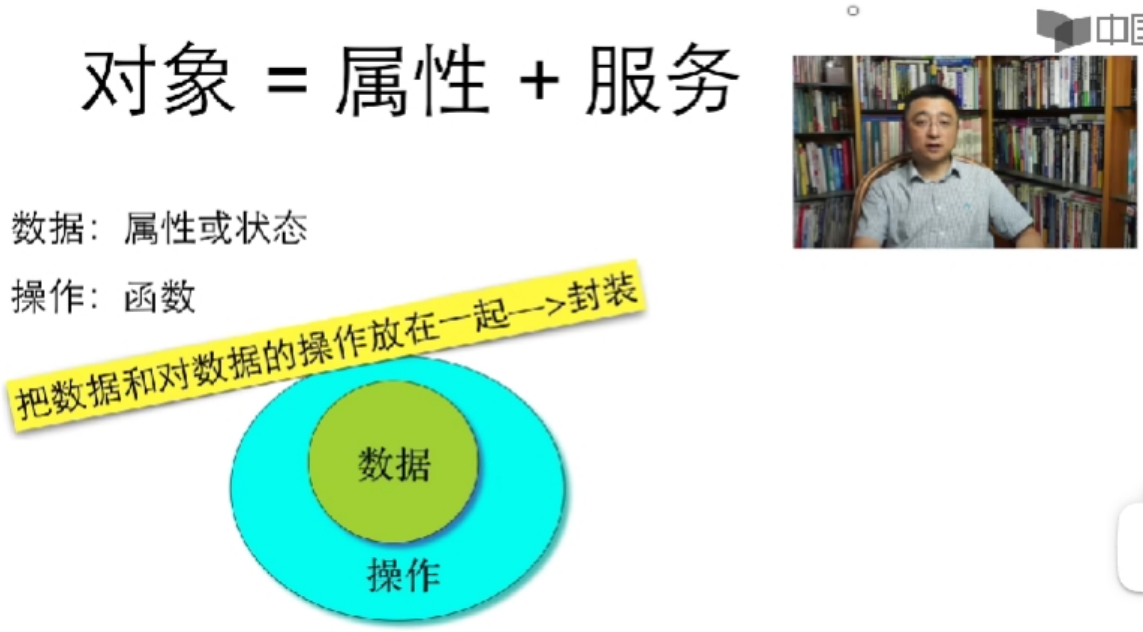
面向对象与面向过程的区别

- OO首先关心的是*所要处理的数据*，OP首先关心的是*功能*
- 面向对象并不是说可以解决这些问题，而是说，与传统方法相比，OO在*稳定性、可修改性和可重用性*方面更具优势
- 对象是以面向对象方法构造的系统的基本单位

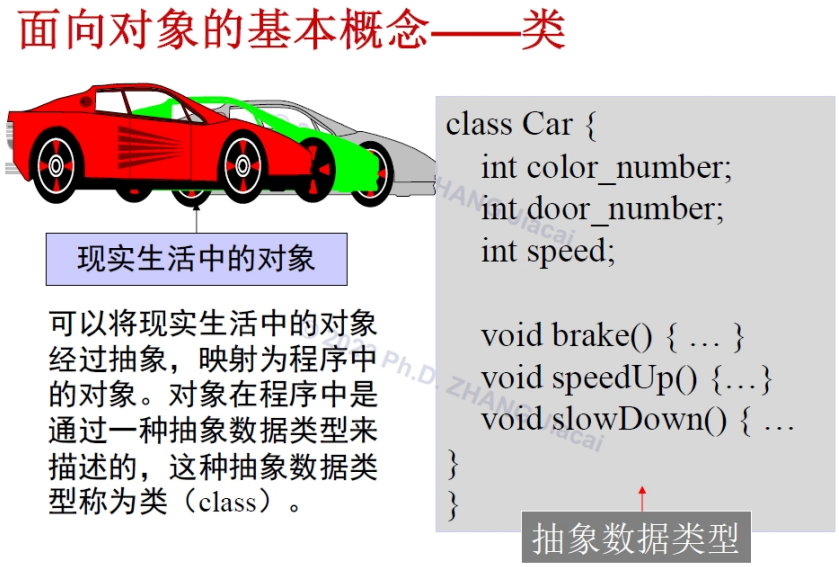
认识对象

- 面向对象编程好比给系统建模，如在线购物系统
- 认识**对象**
  - 对象可繁可简
  - 对象可实可虚
- 对象的成员(属性和操作)
  - *属性*也可以是其他对象，订单中的用户对象

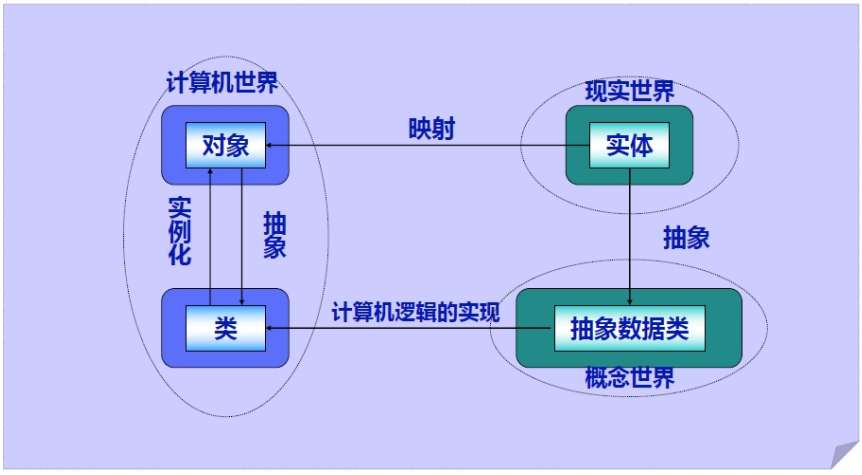
- *操作*是对象要完成的功能



- **类**
  - 类是人类抽象思维的产物
  - 类是*具有相同属性和操作的一组对象的集合*，它为属于该类的全部对象提供了*统一的抽象描述*
  - 类可以看作是用户自定义的数据类型，跟结构体不同的是类中除了*定义成员变量*，还*定义了一组行为*



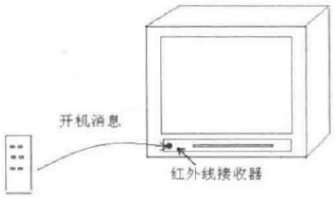
- 对象与类
  - 对象是类的一个具体实现(数据)，一个类的实体；
  - 类是对象的模板，对象的规范
  - 类中的属性不是给类用的，而是给对象使用



- **消息、服务**
  - 软件系统的复杂功能由各种对象的*协同工作*共同完成
  - 相对于其他对象，每个对象都有**特定的功能**，这就是提供服务
  - **接口**：对外提供的所有服务
    - 对象通过接口对外提供服务

消息、服务

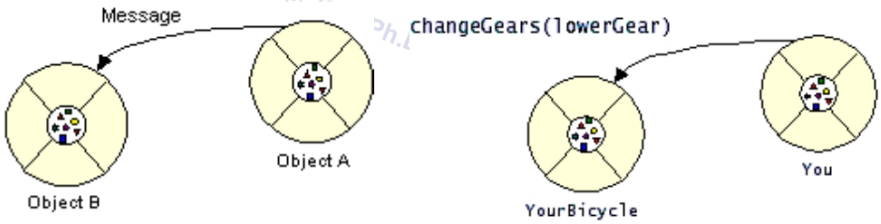
- **软件系统的复杂功能由各种对象的协同工作共同完成**
  - 当用户遥控器向电视机对象发送“open”消息
  - 电视机对象接收到这个“open”执行相应开机操作
  - 此外，遥控器对象还向电视机发消息，如换频道，调节音量等，收到消息执行对应操作



- **每个对象都具有特定的功能，相对于其它对象，这就是提供服务**
  - 电视机对遥控器提供开机，换频道，调节音量等服务
  - 对象提供的服务就是对象的方法来实现的，消息不是方法的调用
    - Television.open()
- **接口：对外提供的所有服务**
  - 对象通过接口对外提供服务

对象之间的通信

- 单个对象通常不是很有用的，通过对象的交互作用，程序员可以获得高阶的功能以及更为复杂的行为
  - 对象与其它对象进行交互与通讯是利用发送消息给其它对象来实现的。当对象A向对象B来执行一个B中的方法，对象A就会消息给对象B。
  - 可能接收的对象需要更多的信息才可以正确知道该如何做。比如，当你想改变自行车的齿轮，通过参数指出哪个齿轮。
  - 接收消息的对象（YourBicycle）、要执行方法的名字(changeGears）、这个方法需要的所有参数（lowerGear）



对象的方法调用

对象的方法调用

思考这能有什么好处:
thing1.setX(47)
thing1.x=47

- 方法
  - 方法可以将参数的值传入方法中
  - 方法可以按照返回类型返回值；返回类型代表不返回任何值
  - Java要求严格的返回类型
- 对象的成员访问
  - <object>.<member>，访问对象成员（属性和方法）
  - 对于类中的本地成员，可以直接访问

```
public class TestThing{
   public static void main (String []args){
      Thing thing1=new Thing();
      thing1.setX(47);
      //thing1.x=47;
      System.out.println(thing1.getX());
   }
}
```

```
public class Thing{
   private int x;
   public int getX(){
      return x;
   }
 public void setX(int new_x){
      x=new_x;
   }
}
```

构造器(constructor)

- 并非类的成员方法，而是一个**用来创建对象的特殊方法**，用来*初始化对象的属性*
- **没有返回类型**
- 由类实例化生成对象时，实际调用类的构造器
  - 构造器**名字与类型相同**
  - 构造器完成*实例初始化*，可以传入参数
  - 不是方法，不可以返回值，也*不可以继承*
  - 每个类*至少有一个构造器*，Java提供一个*无参数*的空构造器(缺省)，*可以有多个构造器*(传入的**参数不同**)
  - 如果给一个类定义了一个构造器，缺省构造器*自动丢失*。下面如果没有人为定义无参数构造器，`new Thing()`就不能工作

```
public class Thing{
   private int x;
   public Thing(){  x=47;  }
   public Thing(int new_x){  x=new_x;  }
}
```

```
调用构造器：
Thing thing1=new Thing();
Thing thing1=new Thing(25);
```

  - 注意构造器前面**没有函数`void`之类的符号**，直接`public + class`名即为构造器
    例子：

```java
public VendingMachine(){
        total = 0;
}
public VendingMachine(int price){
        this.price = price
}
```

构造器与方法的区别
- 构造器要通过`new Constructor()`的方式调用
- 方法通过`object.method()`的方式调用
创建和初始化对象的过程
- 分配对象的**内存空间**，并初始化对象的域：`0/null/false`
- 外部初始化赋值
- 调用构造器
```java
public class Employee{
        private String name;
        private double salary = 1500.00;
        private Date birthDay;

        public Employee(String n, Date DoB){
                name = n;
                birthDay = DoB;
        }

        public Employee(String n){this(n, null);} // 调用其他构造器
}
```

变量初始化

- **成员变量**在定义时可以设为默认值，系统**默认为0**
- 可以通过*构造函数(构造器)对成员变量进行赋值*
- 一个类中构造器*可以有多个*，只要他们的参数表不同
- 创建对象时给出不同的参数值，就会调用不同的构造函数
- 通过`this()`还可以*调用其他构造函数*

```java
VendingMachine(int price){
        this();  // 只能使用一次，只能作为构造器的第一句
        this.price = price;
}
```

- 一个类里的同名但参数表不同的函数构成了**重载**关系

此处会报错，因为当`x<=50`时，`y`没有初始值
封装和透明
- 封装是指隐藏对象的属性和实现细节，仅仅对外公开接口
- 一个设计良好的系统会封装所有的实现细节，把它的接口接口和实现清晰地隔开，系统之间只通过接口进行通信

对象封装
- 对象封装将外部的调用和对象本身分开。对象的访问者不管对象的实现，对外部隐藏具体细节
- OO可以将属性与对属性的操作封装在一个类，类中的操作确保对属性的操作符合逻辑

统一建模语言(UML)

Java对象
引用类型所指内存中保存的数据就是对象
```java
package test1;

public class Teacher {
	/**attributes of a teacher*/
	private String name;
	private int age;
	private double salary;
	/**Creates a new instance of Teacher*/
	public Teacher(String name, int age, double salary) {
		this.salary = salary;
		this.age = age;
		this.name = name;
	}
	/**get the name of this teacher*/
	public String getName() {return name;}
	/**get the salary of this teacher*/
	public double getSalary() {return salary;}
	/**get the age of teacher teacher*/
	public int getAge() {return age;}

	public static void main(String[] args) {
		/**通过new从已经定义的类Teacher创建对象*/
		Teacher jason = new Teacher("Jason", 30, 10000);
		System.out.println("Teacher:" + jason.getName());
		System.out.println("\tAge:" + jason.getAge());
		System.out.println("\tSalary:" + jason.getSalary());
```

```
        }
}
```

对象由类创建：

- 方法是某个对象的行为
- 一般的方法，如果不生成对象，就不能调用对应的方法
- this：`this`就是这个对象
- 除**八种基本类型**以外，其他对象必须要用`new`创建

引用类型

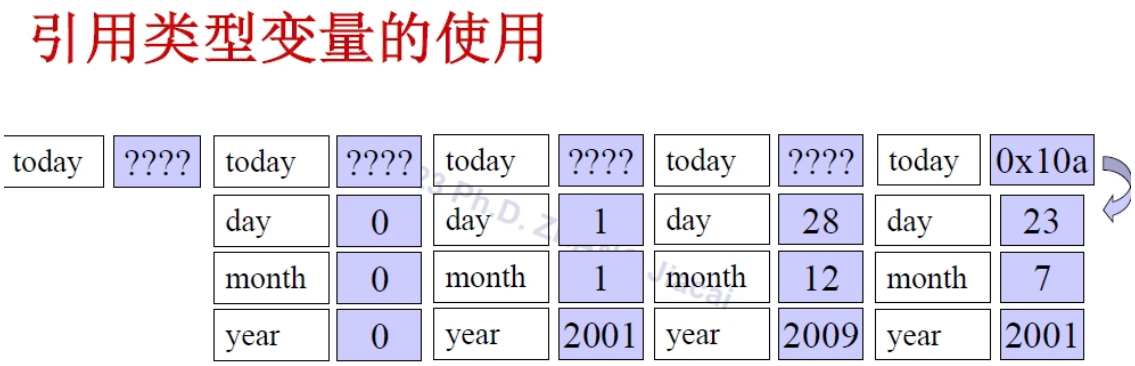
引用类型指向一个**对象**，不是原始值，指向对象的变量是引用变量
例子：

JAVA

```
class MyDate{
        private int day=1;
        private int month=1;
        private int year=2001;
        public MyDate(int day, int month, int year){…}
        public void print(){…}
}

public class TestMyDate{
        public static void main(String args[]){
        MyDate today = new MyDate(28,12,2009);
        }
}
```

- 在MyDate这段代码中，首先在`pubilc class`中对`MyDate`变量进行了赋初值，令`day=1, month=1, year=2001`，但是没有对象进行实例化
- 在TestMyDate这段代码，通过`new`方法实例化了一个对象`today`，并将其重新赋值，则整个的流程为：



这里要注意`today`存放的是一个地址，它指向了后面的MyDate类型包裹的内容

引用类型变量的使用

引用变量也可以赋值

◈ 引用变量（referrence variable）也可以赋值

```
int x = 7;
int y = x;          //y=7,x=7
MyDate today;
today = new MyDate(28,12,2009);
MyDate t = s;
y = 9;              //y=9,x=7
t = new MyDate(17,2,1975);
```

在红字下面的操作中，将一个已赋值变量 `s` 赋值给 `t` ，此时`t`虽然没有传入参数，但此时它内部已经不是初值，而是与 `s` 一样的值。同样的，`t` 也可以通过`new`方法重新构造一个MyDate变量进行赋值操作

JAVA

```java
public class Mydate {
        private int day = 1;
        private int month = 1;
        private int year = 2001;
        public Mydate(int day, int month, int year) {
                this.day = day;
                this.month = month;
                this.year = year;
        }

        public void print() {
                System.out.println("day = " + day + " month = " + month + " year = "
+ year);
        }
}

public class TestMyDate {
        public static void main(String[] args) {
                Mydate today = new Mydate(28,12,2009);
                today.print();
                today = new Mydate(31, 8, 2023);
                today.print();
                Mydate yesterday = today;
                yesterday.print();
        }
}
// 打印:
// day = 28 month = 12 year = 2009
// day = 31 month = 8 year = 2023
// day = 31 month = 8 year = 2023
```

对象构造与初始化：

- 声明引用变量不会立刻分配内存，而是只有一个`null`的指针
- 只有使用`new`后才会给变量分配内存
- 基本变量的拷贝是按值传递的，改变一个不会改变另一个

- 引用变量的拷贝是按照地址传递的，改变一个另一个也会改变

对象构造与初始化

- 当我们声明一个基本数据类型的变量时，系统会自动为变量分配内存。
- 声明一个String或其它自定义类型变量时，系统不立即为其分配内存。
  - 声明为“类”类型的变量不是数据，而是对数据的引用（指针）；
  - 使用引用变量前，调用new Xxx()为一个新对象分配存储空间
- 一个new MyDate(23, 7, 2001)语句引起如下动作：
  - 给新对象分配空间，并将成员变量执行缺省初始化，局部变量不会初始化；
  - 执行外部的初始化；
  - 执行构造函数，new中的参数传入构造函数中；
  - 对象的引用赋给变量。

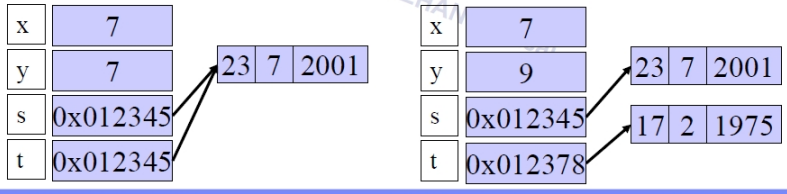
```
int x = 7;
int y = x;              //y=7,x=7
MyDate s = new MyDate(23,7,2001);
MyDate t = s;
y = 9;                  //y=9,x=7
t = new MyDate(17,2,1975);
```

北京邮电大学

引用类型变量的使用

- 上面四条赋值语句，两条是基本类型变量，两条是引用变量。
  - 基本变量赋值直接拷贝值，变量x和y互相独立，进一步的赋值互相不影响。
  - 引用变量赋值，s和t指向同一个MyDate对象，进一步的修改会互相影响。
  - 可以给变量t单独赋给一个新生成的MyDate对象。

```
int x = 7;
int y = x;              //y=7,x=7
MyDate s;
s = new MyDate(23,7,2001);
MyDate t = s;
y = 9;                  //y=9,x=7
t = new MyDate(17,2,1975);
```



AI School, Course for Undergraduate　面向对象方法与技术　© 2023 Ph.D. ZHANG Jiacai

成员变量:
类定义了*对象中所具有的变量*，这些变量称作成员变量
每个对象有自己的变量，和同一个类的其他对象是分开的

函数与成员变量:
- 在函数中可以直接写成员变量的名字来访问成员变量
- *访问的是哪一个对象？*
- 函数是通过对象来调用的，如`v.insertMoney()`
- 这次调用临时建立了`insertMoney()`和`v`之间的关系，让`insertMoney()`内部的成员变量指的是`v`的成员变量

值传递

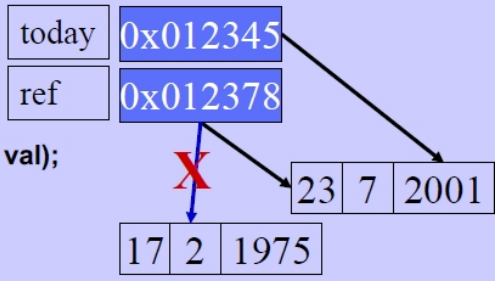
值传递

◈ Java仅仅支持参数的“值传递（by value）”，也就是说调用过程不可以修改参数。当传递的参数是对象实例时，函数可以修改对象内容，但对象的引用不可修改。

```
public class PassTest{
  public static void changeInt(int value) {value=5;}
  public static void changeObjectRef(MyDate ref) {ref = new MyDate(17,2,1975);}
  public static void changeObjectAtt(MyDate ref) {ref.setDate(4);}

  public static void main(String args[]){
    MyDate today = new MyDate(23,7,2001);
    int val = 11;

    changInt(val); System.out.println("value:" + val);
    changeObjectRef(today); today.print();
    changeObjectAtt(today); today.print();
  }
}
```



This引用

- `this`是成员函数的一个特殊的固有的本地变量，它表达了调用这个函数的那个对象
- 由于私有变量命名与方法中传入的变量命名一样，为了避免混淆，所以用`this.value`代替原来的`value`

This引用

◈ This关键字代表对象自己：

◈ 在本地方法或构成器中引用本地属性或方法成员；

◈ 将对象自己作为参数传递给其它方法。

```
public class MyDate{
  private int day=1;    private int month=1;    private int year=2001;
  public MyDate(int day, int month, int year){
    this.day=day; this.month=month; this.year=year;
  }
  public MyDate(MyDate date){
    this.day=date.day; this.month=date.month; this.year=date.year;
  }
  public MyDate addDays(int more_days){
    MyDate new_date = new MyDate(this);
    new_date.day= new_date.day + more_days;
    return new_date;
  }
}
```

```
Mydate today=new MyDate(23,7,2001)
MyDate newDay=today.addDays(7);
newDay.print()
```

JAVA

```java
public class Test{
public static void main(String[] args){
        Person p = new Person();
        p.display;
        p.setAge(18);
        p.display();
        }
}

class Person{
        private int age;
        public Person(){}
        public Person(int i){
                age = i; // 等价于this.age = i;
        }
        /*public Person(int age){
                this.age = age; // 不同于age = age
        }*/
        publicc void setAge(int i){
                age = i;
        }
        public int getAge(){return age;}
        public void display(){
                System.out.println(this.age);
        }
}
```

JAVA

```java
public class VendingMachine {
        int price = 20;
        int balance = 0;
        int total = 0;

        void setPrice(int price) {
                this.price = price;
                }

        void showPrompt() {
                System.out.println("Welcome");
                }

        void insertMoney(int amount) {
                balance = balance + amount;
                }

        void showBalance() {
                System.out.println(balance); // 相当于this.balance
                }

        void getFood() {
                if(balance >= price) {
                        System.out.println("Here you are");
                        balance -= price;
                        total += price;
                        }
                }

        public static void main(String[] args) {
                VendingMachine vm = new VendingMachine();
                vm.showPrompt();
                vm.showBalance();
                vm.insertMoney(100);
                vm.getFood();
                }
}
```

调用函数

- 通过 `.` 运算符调用某个对象的函数
- 在成员函数内直接调用自己的其他函数

```java
void insertMoney(int amount) {
                balance = balance + amount;
                showBalance();  // 由于在成员函数内部，所以这个地方不用写
this.showBalance()直接这么写就好
}
```

本地变量

- 定义在**函数内部的变量是本地变量**
- 本地变量的生成期和作用域都是**函数内部**
- **成员变量的生存期是对象的生存期**(不用关心怎么销毁，java会自动垃圾回收)，作用域是类内部的成员函数

Java方法使用举例

```java
public class Test{
        public int method1(int a , int b, int c){
                int k = a + b + c;
                return k;
        }

        public static void main(String[] args){
                Test t = new Test();
                int j = t.method1(3,4,5);
                System.out.println("the result is:" + j);
        }
}
```

Java值传递举例

JAVA

```
package test1;
public class Example {
        public static void main(String args[]){
                Example ex = new Example();
                int date = 9;
                BirthDate d1= new BirthDate(7,7,1970);
                BirthDate d2= new BirthDate(1,1,2000);

                ex.change1(date);
                // 这里传进去了date，但不会影响date的值，因为只是函数内部的局部变量，不会影响date的值
                ex.change2(d1);
                // 同理，这里虽然传入了d1，但也不会改变d1的内容
                ex.change3(d2);
                // 通过对象的方法(setDay())进行设置才能改变对象属性

                System.out.println("date=" + date);
                d1.display();
                d2.display();
                }

        public void change1(int i){i = 1234;}
        public void change2(BirthDate b){b = new BirthDate(22,2,2004);}
        public void change3(BirthDate b){b.setDay(22);}
}

class BirthDate {
        private int day;
        private int month;
        private int year;

        // 构造器
        public BirthDate(int d,int m,int y){day = d; month = m; year = y;}

        // 函数，注意一定要有这个，不能直接访问day之类的私有变量
        public void setDay(int d){day = d; }
        public void setMonth(int m){month = m; }
        public void setYear(int y){year = y; }

        public int getDay(){ return day; }
        public int getMonth(){return month; }
        public int getYear(){return year;}

        public void display(){
                System.out.println(day + " - " + month + " - " + year);
```

```
        }
    }
```