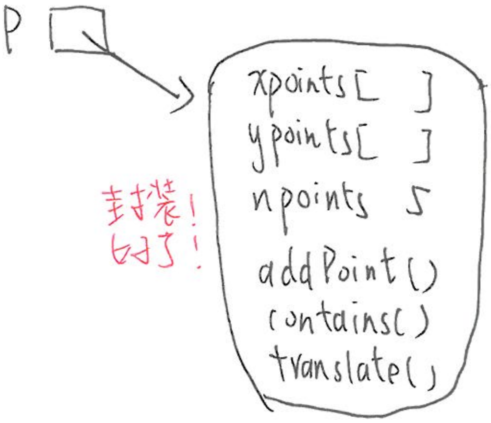
↓在面向对象编程中，该操作从逻辑上实质为多边形的一部分.

```
int [] theXs = {0, 4, 6, 5, 2};
int [] theYs = {0, 0, 2, 4, 3};
int num = 3;
Polygon P = new Polygon(theXs,theYs, num);
P.addPoint(0, 2);
if (P.contains(2, 1))
        System.out.println("Inside P");
else
        System.out.println("Outside P");
P.translate(2, 3);
```

→移动组成P的点

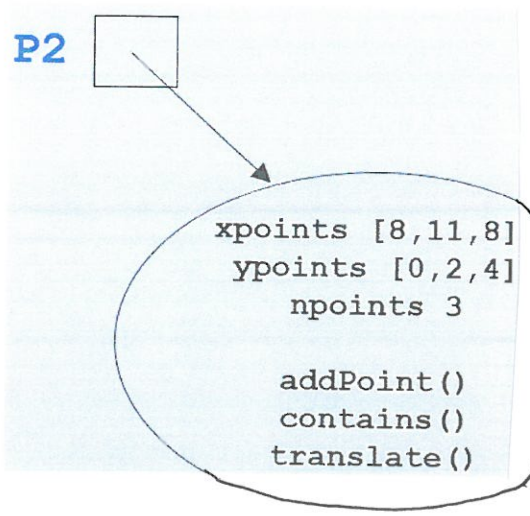
我们从多边形中调用方法 ← 多边形的一部分

Objects enable us to combine data and operation of a type into a entity：封装 Encapsulation



方法也在对象内

方法总是执行于object's data



对于同一类的多个对象，操作实施于特定对象.

```
int [] moreXs={8,11,8};
int [] moreYs={0,2,4};
Polygon P2 =new Polygon(moreXs, moreYs, 3);
```

综上：class是多边形的蓝图 — 一个类决定任意一个多边形的数据类型和方法.

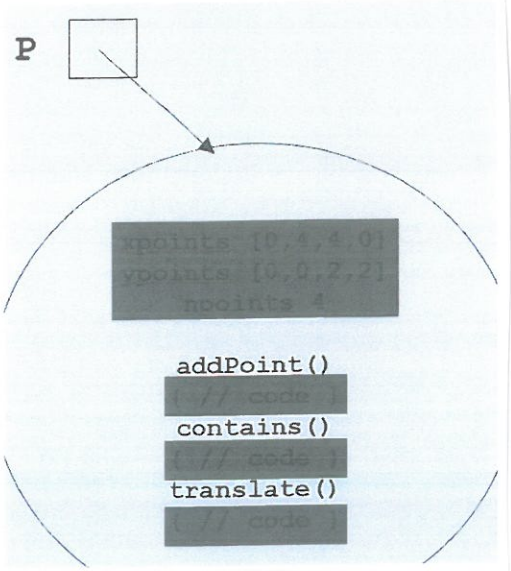
（类：决定了多边形的general property）.

objects是多边形的instance — 对象有一个多边形的具体数据（几个点，点在哪里）.

data abstraction

我们不需知道一个对象的所有细节. 对象为包含数据与方法的容器

The ones we can see give us the functionality of the objects.



Data Abstraction.

X know how the data is stored & how the method is implemented.

But we can use the Polygon.

一对象中的数据被称为 instance variable (因为这是与类的实例相关联的variable

一对象中的方法被称为 instance method (因为这是与类的实例相关联的 method

如果想要用别人写的，我们得知道 < Basic idea of class
Method headers and what they do

另例：数据的读入

```
Scanner myScanner = new Scanner(System.in);
int x = myScanner.nextInt();
```

第一行建立了 Scanner 对象

第二行调用了读取并返回下一个token作为int型的方法.

↓不需知道

Lecture 8

复习：Data abstraction allows us to use a class without knowing < how the data is stored
how the method is realized

Encapsulation allows the author of class make details inarressible to users — must do abstraction

在一个类中，可以使用关键字 private 声明 instance variables 实例变量.

=> They cannot be directly accessed outside the class itself (类外不能直接访问

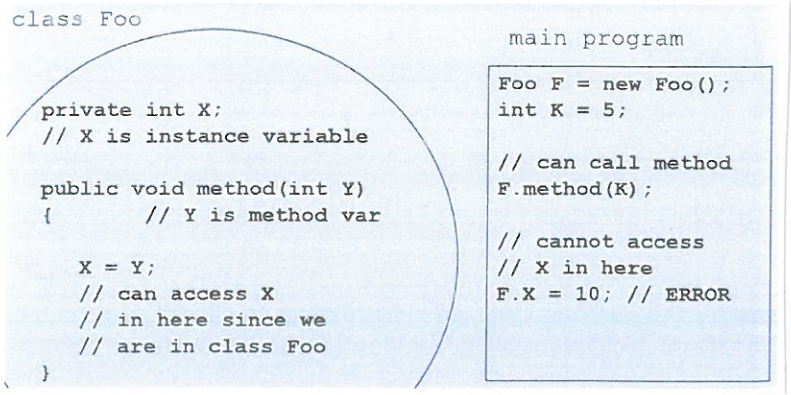
方法变量
Method variable 在某种方法内被声明

```
public void method()
{
  int Y;
}
```

实例变量
Instance variable 在一个类内被声明，而不是在一个方法内

```
public class Foo
{
  private int X;
```

Scope is the object that encapsulates the variables. ⇒not accessible outside the context of ob



之前所讨论的方法是 class method 类方法 ← 不与任何 object 相关联
被视作独立的方法

涉及OOP时，讨论 instance method 实例方法 — 与类的独立实例相关

```
StringBuilder B = new StringBuilder("this is ");
B.append("really fun stuff!");
System.out.println(B.toString());
```

一个对象内的操作

Class Method 没有隐式数据 — 所有数据都由实参传递.

形式 ClassName.methodName(param list)

Instance Method 有与对象相关的隐式数据

VariableName.methodName(param list)

↘ reference to an object

Sum ① 对象将数据与操作封存进单一实体

Instance variable 定义对象内的 data

Instance method 定义对象使用的 operation

② 类定义中指定了实例变量和方法

private 声明限制了一个对象的使用者所能"看"的变量/方法.

③ 对象是包含了特定的数据和方法的类的实例

基于 Data abstraction，我们应当知道：
- general idea of data
- instance method (name)
- general function
- parameters they need

封装和数据抽象紧密相联

将数据和方法封装在对象里使程序员限制对抽象的访问和要求

Instance method 实例方法

1° Constructors 创建者

- 这是当对象首次被建立时的特殊的实例方法
- 叫他它们没有 return value (not even void)
- 通常用来来初始化对象的实例变量

```
StringBuilder B = new StringBuilder("hello there");
B = new StringBuilder(); // default constructor
B = new StringBuilder(10); // capacity 10
```

创建者默认化

容量为10

2° Accessors 访问者 / getter

- 以某种方式访问对象而不改变对象
- 从 accessors 中获得信息
- 没有特殊语法

```
StringBuilder B = new StringBuilder("hello there");
char c = B.charAt(4);  // c == 'o'
String S = B.substring(3, 9);  // S == "lo the"
           // note that end index is NOT inclusive
int n = B.length();  // n == 11
```

第4个字母

4-9的字母

字符长度

3° Mutator 见武值 / setter

- 以某种方式改变对象
- instance variables are private ⇒ 使用 Mutator 改变对象而不知道 instance variables

```
B.setCharAt(0, 'j');  // B == "jello there"
B.delete(6,7);  // B == "jello here"
B.insert(6, "is "); // B == "jello is here";
```

第0字符改为j

删除 7,8

7位插入

改变了内容

Constructor 初始化 instance variable

Accessor & Mutator 间接访问改变 instance variable

一个简例

IntCircle
- Instance variable: private int radius
  > Cannot directly access it from outside the class
- Constructor: take an int argument and initialize a new circle with the given radius
- Accessors:
  public double area();
  public double circumference();
  public String toString();
- Mutator:
  public void setRadius(int newRadius);

Lecture 9

创建一个类 PlayList — 歌曲的集合
- 定义 Song type
- use ArrayList
- build class via composition

搭建一个类的基石出构架
① 决定一些 instance variables，1~2个 constructor，一个用于输出数据的 accessor
例 tostring() method
② 写一个简单的驱动来测试
③

Unit testing & Assertions

如何实现
- Save a large amount of data
- initialize a database that is used
- output from one program must be input to another

We need to use files
- Text Files 文本
- Binary Files 二进制

Java Arrays

- They are objects with certain properties (像其它引用类型一样.
- Array在逻辑上是个单变量名但允许访问多变量位置
- 所访问的位置应当是 contiguous & homogeneous. (相邻且同类)
  - contiguous: 每个都跟随在内存中的前一个
  - homogeneous: 数组中所有的引用具有相同的类型.

创建Java数组有2步

① prim_type [] var name; //建立数组变量,而非实际数组

② var-name = new prim_type [arr-size]; //创建数组对象

实例 ① int [] myArray

② myArray = new int [10]

2步可以合二为一: int [] myArray = new int [10]

prim_type [] var_name = new prim_type [arr_size] .

一个新的myArray数值类型初始为0,布尔值初始为false

- location within an array被视作数组对象内的 instance variables

Indexing an array 改变数组内特定位置的属性

使用operator[]

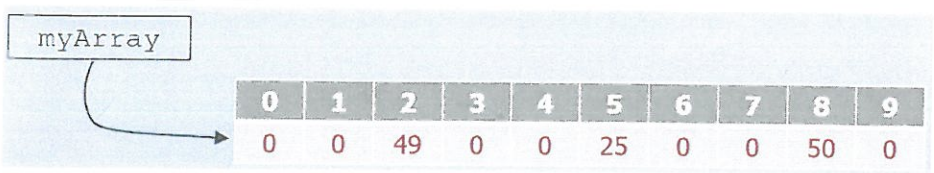
例: int [] myArray = new int [10];

↓ 位置号从0至9,个数值为0

index [] 操作

```
myArray[5] = 25;
myArray[8] = 2 * myArray[5];
myArray[2] = myArray[8] - 1;
if (myArray[2] < myArray[8])
    System.out.println("Less Than!");
```

[]操作产生的结果



Text Files

- ASCII字母序列
- 更大的数据类型同样以字母储存
  Strings . ints .

- nextInt(): convert characters into an integer
- 将数据读取进 object with many instance variables
  从文件中读取数据值，通过 constructor & mutator 分配
- 读取任意所需数值来填充数组.

```
Scanner keyIn = new Scanner(System.in);
System.out.print("Input file name: ");
String fName = keyIn.nextLine();
File inFile = new File(fName);
Scanner fileIn = new Scanner(inFile);
while (fileIn.hasNextLine())
{
    String line = fileIn.nextLine();
    System.out.println(line);
}
```

Get file name using a regular keyboard Scanner

Create a File object from the name, then create a Scanner object from the File

use a Scanner with an input file
与 keyboard 类似

将项目内的data存为 text file 使用 toString() 方法

PrintWriter 类 — 支持将基本类型和字符串写进 text file

目的: store all value being read, then sort them and print
实现方法: 存储若任意数的值，而不需要对等数量的变量.

& Array

Lec 10

Iterating through an array (遍历数组)

一使用 For Loop 遍历数组

为使循环结束，访问数组的长度属性

```
for (int i = 0; i < myArray.length; i++)
{
    System.out.print("Value " + i + " = " + myArray[i]);
}
```

- Or we can iterate on the values without a counter

```
for (int value : myArray)
{
   System.out.println("Next value is : " + value);
}
```

访问数组有两个方式：

① 直接访问

使用 index

② Sequential Access 有序访问

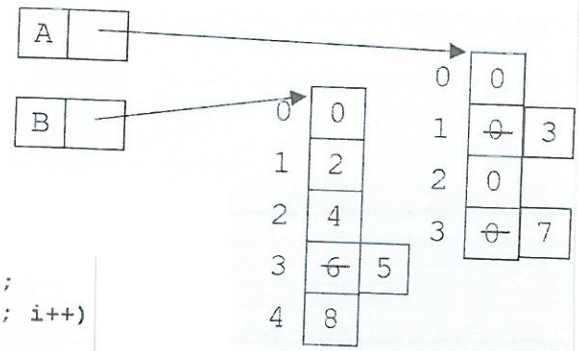
正序 or 逆序

Java数组是 Reference

数组变量是实际数组的引用

通过索引改变数组的内容

例




```
int [] A = new int[5];
for (int i = 0; i < 5; i++)
    A[i] = 2*i;
int [] B = A;
A[3] = 5;
A = new int[4]; 新创建的数组
A[1] = 3;
A[3] = 7;
```

回忆：Java的形参都是 value

实参的备份传给形参

形参的改变不影响实参

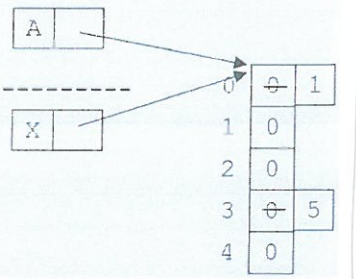
对于Java数组

cannot change the argument variable

但可以改变 array object

```
int [] A = new int[5];
foo(A);

-----------------------------------------
public static void foo(int [] X)
{
   X[0]++;
   X[3] = 5;
}
```



数组中，形参、实参都是对同一对象的引用

直接改变 object

Sequential Search 有序查找

从头开始检查每一个目标直到数组尾或找到所需目标.

有两种情况 { ① failure. loop结束，没找到
② success. 找到匹配 item

Lecture 12

Java arrays of primitive type

```
int [] data; // declare variable (reference) ①
data = new int[20]; // create array object ②
...
data[4] = 77; // index array to access locations
```

与对象数组的不同

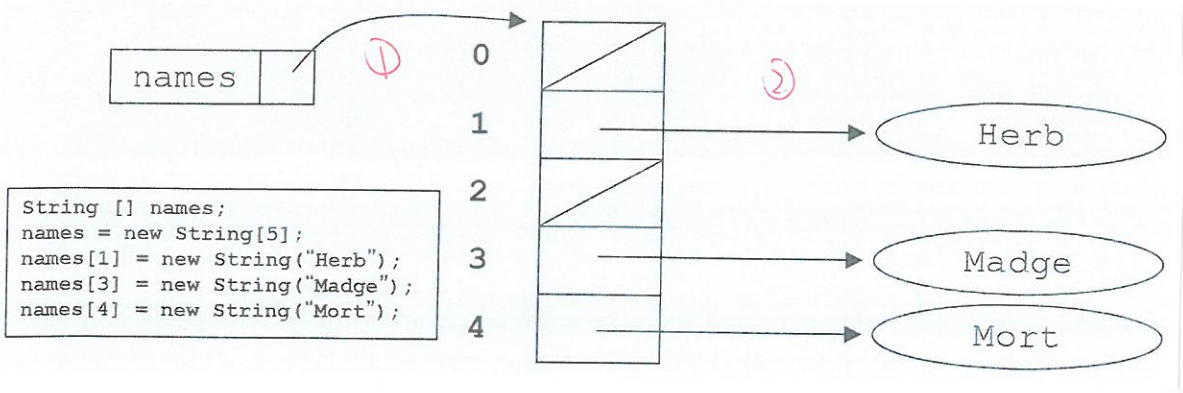
前两步一样的（上图 int[] data = new int [20]; ），

对象由 reference type 访问

但，当array创建时，内部无对象 { 所有地址 initialized to null
create objects

```
String [] names;  // declare array var
names = new String[5];  // make array object
names[1] = new String("Herb"); // make String
names[3] = new String("Madge");
names[4] = new String("Mort");
```

names[0] names[2] null



两层reference

一个新class里可以包含任意实例变量

↓ 为reference type时，via composition

以已有的中组成新类 (aggregation)

简例：Scramble class

Scramble有以下实例变量：String currWord、String scrambledWord、Scanner wordFile、Random R

↓ Scramble class包含Strings，a Scanner，a Random

对于within Scramble

- 知道所有实现细节，但只能访问public method

对于outside Scramble

- 使用者不知道变量，只知道公共方法

综上，当数组被用作instance variable {
- 类内，拥有相同的访问权限，正如程序中其它地方
- 类外，由于封装，并不清楚数组是否被使用

数组对象可以是任何大小，但创建后大小无法调整。

```
System.out.println("How many integers?");
int size = inScan.nextInt();
int [] theInts = new int[size];
```

知道item的数量，[但并非所有时候]

数组满了？

Logically，resize it.

Physically，
1. 创建一个新的、更大的数组对象
2. 从旧的数组复制数据
3. 布署新的reference到新对象

此操作不难，但是花时间。