OOP with Java

Yuanbin Wu
cs@ecnu

OOP with Java

- 通知
  - Project 4: 11 月 3 日晚 9 点

- 复习
  - 类的复用
  - 组合 (composition):
    - has-a 关系

```
class MyType {
    public int i;
    public double d;
    public char c;
    public void set(double x) { d = x;}
    public double get() { return d; }
}
```

```
public class MyCompType {
    private MyType m = new MyType();
    private String s;
    public MyCompType(){
        s = new String(“Hello”);
    }
}
```

- 复习
  - 继承 (inheritance)
    - is-a 关系

```
class MyType {
    public int i;
    public double d;
    public char c;
    public void set(double x) { d = x;}
    public double get() { return d; }
}
```

```
public class MySubType extends MyType{

    String s = new String("Hello");
    public double add(double d){return this.d + d;}
    public double add(String s){return this.s + s;}

    public void set(double x){ i = (int)x; }
    public double get() { return i; }

    public static void main(String [ ]args){
        MySubType ms = new MySubType();
        System.out.println(ms.get());
        System.out.println(ms.add(1.0));
        System.out.println(ms.add("World"));
    }
}
```

- 复习
  - 继承
    - 子类有父类的所有方法和数据
    - 子类可以定义新的方法和数据
    - 子类可以重写 (override) 父类的方法
  - super 关键字
    - 每一个子类对象都隐含包含一个父类对象
  - Object 对象
    - Single root class hierarchy tree
    - 方法 :

```
boolean equals(Object o)

String toString()
```

```
class MyType {

    public int i;
    public double d;
    public char c;
    public void set(double x) { d = x;}
    public double get() { return d; }

    public static void main(String [ ]args){
        MyType m = new MyType();
        MyType n = new MyType();
        String s = “hello”;
        m.equals(n);
        m.equals(s);
    }
}
```

OOP with Java

- protected
- upcasting
- final 关键字

protected

- 访问控制
  - package access
  - public
  - private

protected

- 函数重写

```
class MyType {
    public int i;
    public double d;
    public char c;
    public void set(double x) { d = x;}
    public void set(int y) {i = y;}
    public double get() { return d; }
}
```

```
public class MySubType extends MyType{

    public double foo(){ return get(); }
    public void set(double x){ i = (int)x; }
    public void set(char z) {c = z; }

    public static void main(String [ ]args){
        MySubType ms = new MySubType();
        ms.set(1.0);
        System.out.println(ms.get());
        System.out.println(ms.i);
        System.out.println(ms.d);
    }
}
```

protected

- 函数重写？

```
class MyType {
    public int i;
    public double d;
    public char c;
    private void set(double x) { d = x;}
    private void set(int y) {i = y;}
    private double get() { return d; }
}
```

```
public class MySubType extends MyType{

    // can not access!!
    // public double foo(){ return get(); }
    public void set(double x){ i = (int)x; }
    public void set(char z) {c = z; }

    public static void main(String [ ]args){
        MySubType ms = new MySubType();
        ms.set(1.0);
        System.out.println(ms.get());
        System.out.println(ms.i);
        System.out.println(ms.d);
    }
}
```

Protected

- 父类的方法
  - public
  - private
  - 是否有可能被子类访问而不被外界访问？

protected

- protected
  - 可以被子类 / 同一包中的类访问，不能被其他类访问
    - 弱化的 private
    - 同时赋予 package access

protected

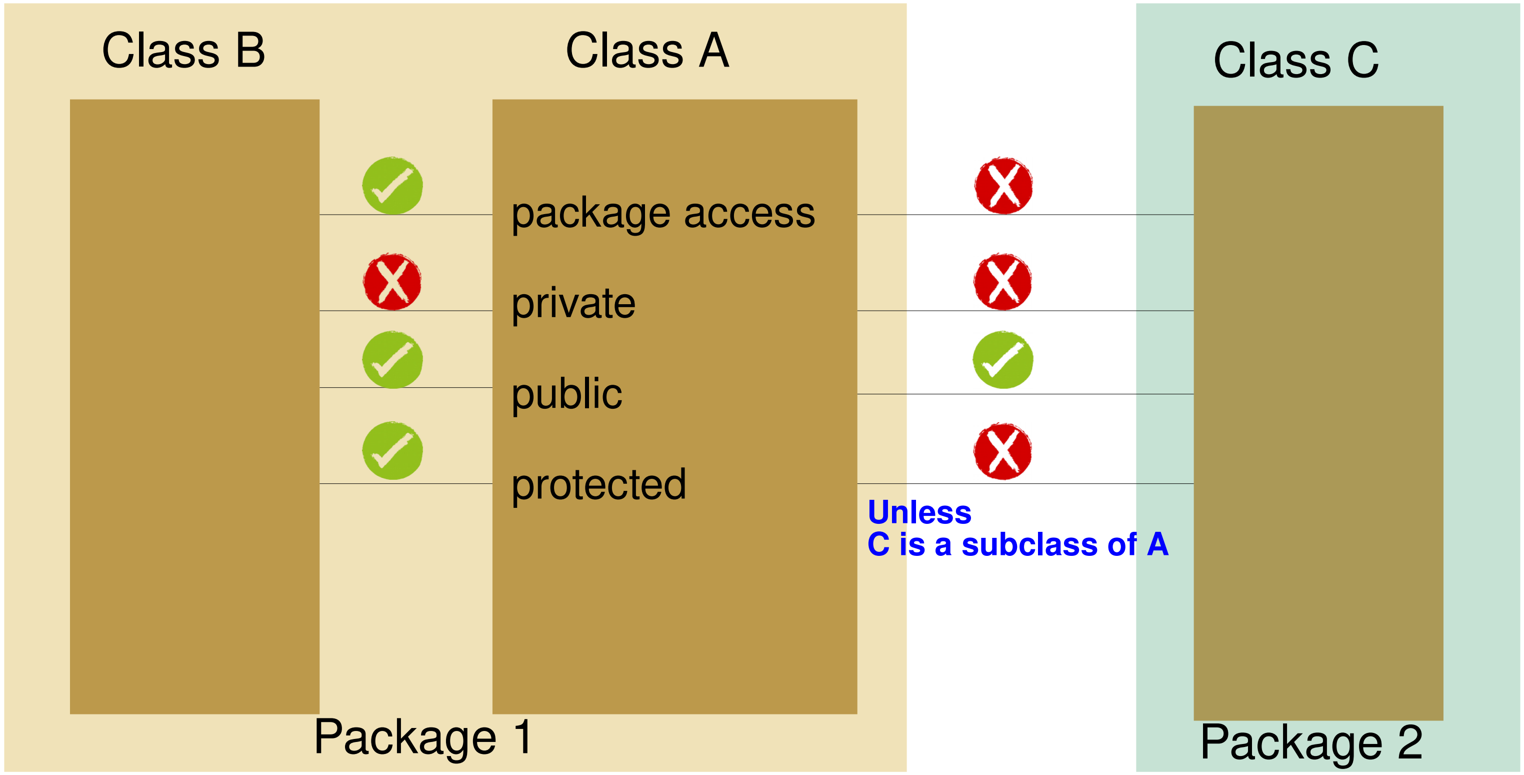
```
class MyType {
    public int i;
    public double d;
    public char c;
    protected void set(double x) { d = x;}
    protected void set(int y) {i = y;}
    protected double get() { return d; }
}
```

```
public class MySubType extends MyType{

    public double foo(){ return get(); }
    public void set(double x){ i = (int)x; }
    public void set(char z) {c = z; }

    public static void main(String [ ]args){
        MySubType ms = new MySubType();
        ms.set(1.0);
        System.out.println(ms.get());
        System.out.println(ms.i);
        System.out.println(ms.d);
    }
}
```

Protected

- 访问控制
  - package access
  - public
  - private
  - protected

Protected

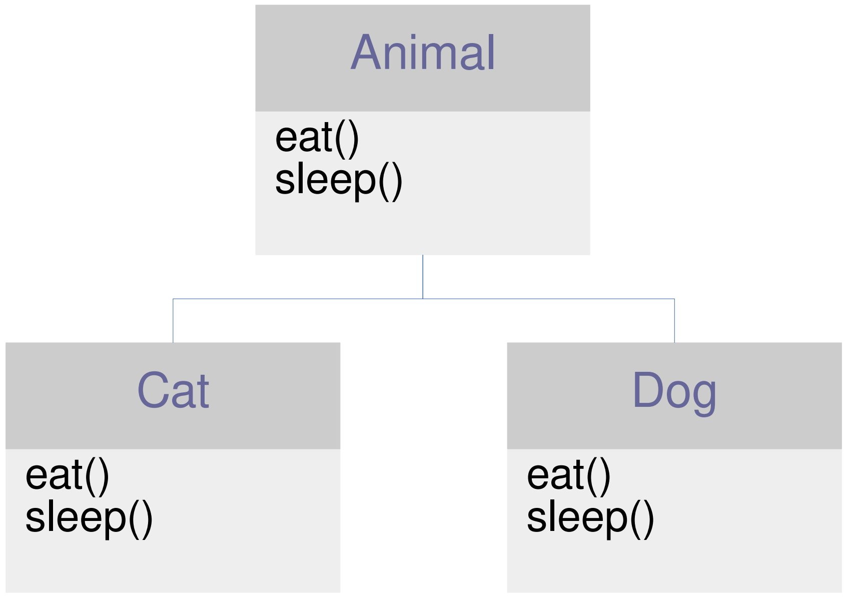

Class B
Class A
Class C
✓
package access
✗
✗
private
✗
✓
public
✓
✓
protected
✗
Unless
C is a subclass of A
Package 1
Package 2

Protected

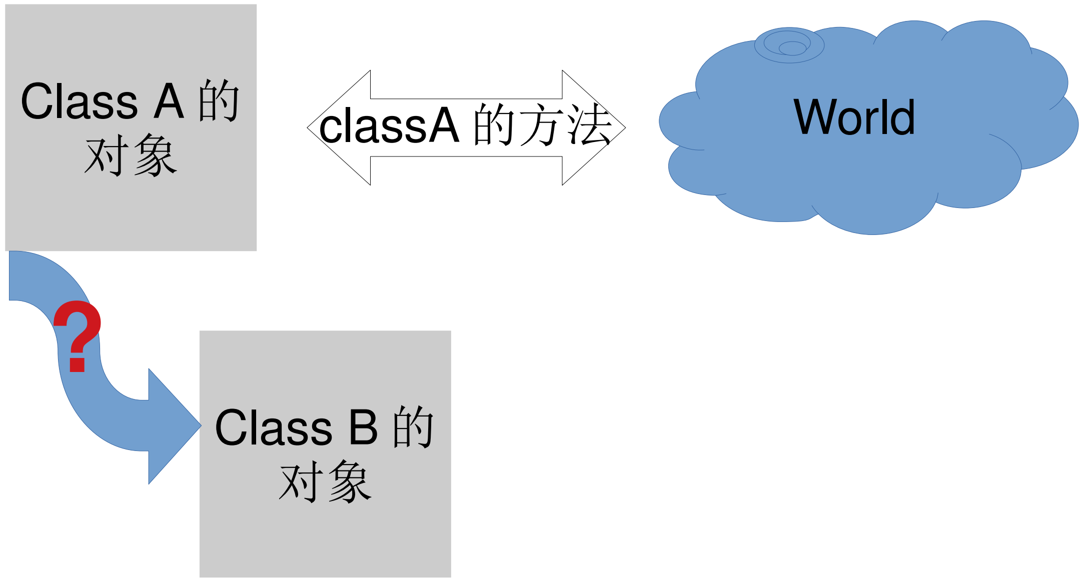
Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N

Upcasting

- 继承
  - 子类拥有父类所有的数据和方法



Upcasting

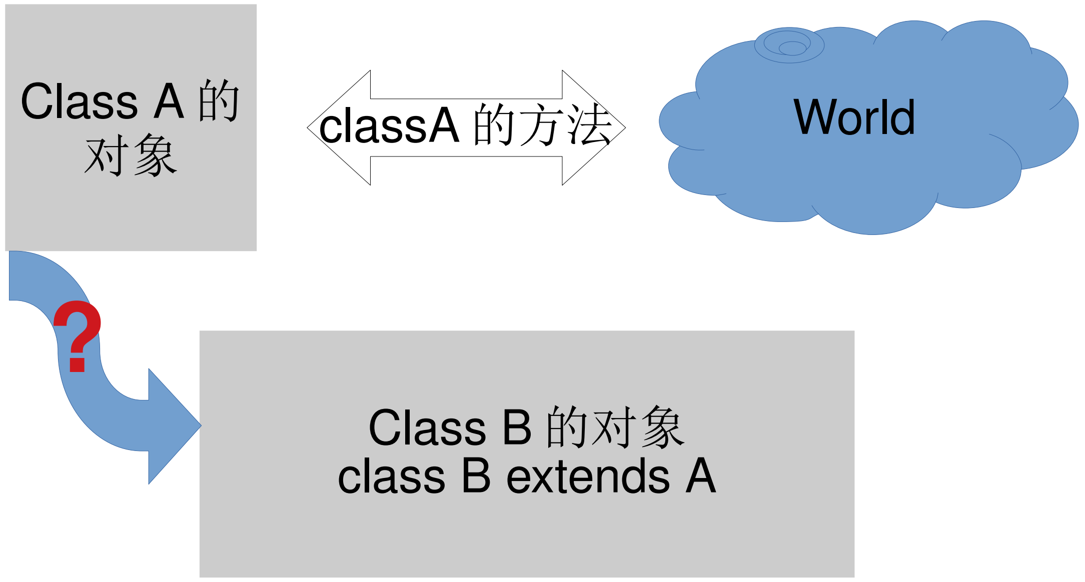


Cannot

A 与 B 有不同的类型

```
class B {
    // ...
}

class A {
    // …
    public void foo() {}
}

class C {
    public void bar(A a) { a.foo(); }

    public static void main(String []argv){
        A a = new A();
        B b = new B();
        C c = new C();
        a.foo();
        c.bar(a);
        // replace a with b
        b.foo();
        c.bar(b);
        A a1 = b;
    }
}
```

Upcasting



Yes

子类拥有父类所有的数据和方法

```
class B extends A {
    // ...
}

class A {
    // …
    public void foo() {}
}

class C {
    public void bar(A a) { a.foo(); }

    public static void main(String []argv){
        A a = new A();
        B b = new B();
        C c = new C();
        a.foo();
        c.bar(a);
        // replace a with b
        b.foo();
        c.bar(b);
        A a1 = b;
    }
}
```

Upcasting

- 类型关系：
  - 子类是一种父类 （“is-a 关系”）
  - the sub-class is a type of the base class

Upcasting

- 例子

```
class Instrument {
    public void play() {}
    static void tune(Instrument i) {
        // ...
        i.play();
        // ...
    }
}

public class Wind extends Instrument {
    public static void main(String[] args) {

    Wind flute = new Wind();

    Instrument.tune(flute);
    }
}
```

Upcasting

Upcasting

- 例子

```
public class MySubType extends MyType{

    String s = new String(“Hello”);
    public double add(double d){return this.d + d;}
    public double add(String s){return this.s + s;}

    public static void main(String [ ]args){
        MySubType ms = new MySubType();
        MyType m = ms;

        System.out.println(m.get());
        System.out.println(ms.add("World"));

        m.set(1.0);
        System.out.println(m.get());
        System.out.println(ms.get());
    }
}
```

Upcasting

- Upcasting（向上转换）
  - 需要父类对象的地方可以用子类对象带入
    - 引用，函数参数
  - 一种类型转换
    - 安全的
    - 子类拥有父类所有的数据和方法
  - 其他的类型转换？
  - 类型间关系

Upcasting

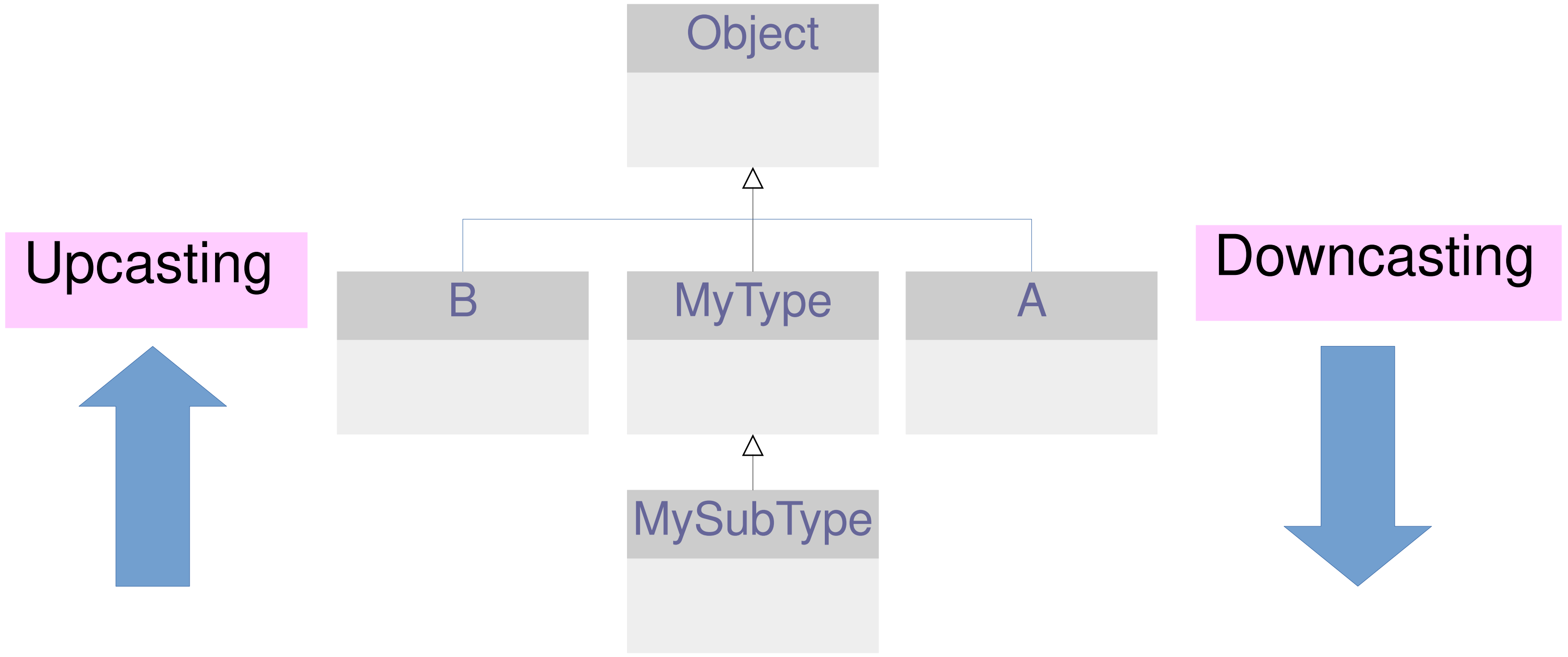
- 例子

```
public class MySubType extends MyType{

    String s = new String(“Hello”);
    public double add(double d){return this.d + d;}
    public double add(String s){return this.s + s;}

    public static void main(String [ ]args){

        MySubType ms = new MySubType();
        MyType m = ms;

        System.out.println(m.get());
        System.out.println(ms.add("World"));

        m.set(1.0);
        System.out.println(m.get());
        System.out.println(ms.get());
    }
}
```

```
int i = 5;
double d = i;
```

Upcasting

- Upcasting



Upcasting

- 子类重写了父类方法？

```
class MyType {
    public int i;
    public double d;
    public char c;
    protected void set(double x) {
       System.out.println(“base class”);
       d = x;
    }
    protected void set(int y) {i = y;}
    public double get() { return d; }
}
```

```
public class MySubType extends MyType{
    public void set(double x){
        System.out.println(“sub class ”);
        d = x;
    }
    public static void main(String [ ]args){
        MySubType ms = new MySubType();
        MyType m = ms;
        m.set(1.0);
    }
}
```

多态

Upcasting

- 类型转化
  - 基本类型
    - int → double (安全 , 自动转换)
    - double → int (损失精度，强制转换)
  - 基本类型与 wrapper
    - int → Integer (autoboxing)
    - Integer → int (unboxing)
  - 类
    - 不支持强制转化
    - 子类 → 父类 (安全 , upcasting)
    - 父类 → 子类 (downcasting)

Upcasting

- Downcasting
  - MySubType ms = (MySubType)m;
  - 仅在 m 确实指向子类对象时才能进行
  - 运行时类型信息 (RTTI)

```
public class MySubType extends MyType{
    public void set(double x){
        System.out.println("sub class ");
        d = x;
    }
    public static void main(String [ ]args){
        MySubType ms = new MySubType();

        MyType m = ms;
        m.set(1.0);

        MySubType n = (MySubType)m;
        m.set(1.0);
    }
}
```

Upcasting

- 总结
  - 子类是一种父类 (is-a)
  - 父类的引用可以指向子类对象

final 关键字

- final 关键字
  - 不同的环境下有不同含义
  - 基本意义为: **不能被改变**

final 关键字

- final 数据
  - 编译时常数
  - 一旦被赋值就不能被修改

final 关键字

- final 数据
  - 例子

```
class MyType {
    public int i;
    public final double d = 1;
    public char c;
    public double get() { return d; }
    public void set(double x) {d = x;}

    public static void main(String []args){
        MyType m = new MyType();
        // m.d = 2.0;
    }
}
```

final 关键字

- final 数据
  - final 引用

```
class MyType {
    public int i;
    public final double d = 1;
    public char c;
    public final int [ ] a = new int[10];

    public double get() { return d; }
    public void set(double x) {d = x;}
    public static void main(String []args){
        MyType m = new MyType();
        m.a[0] = 1.0;
        //m.a = new int[10];
    }
}
```

final 关键字

- final 数据
  - final + static
    - static final int i = 1;
  - 仅有一个不可变的存储空间

final 关键字

- final 数据
  - Blank final

final 成员在定义时可以不给初值
必须在构造函数中初始化

```
class MyType {
    public int i;
    public final double d;
    public char c;
    public double get() { return d; }
    public MyType(double x){ d = x; }

    public static void main(String []args){
        MyType m = new MyType(1.0);
        System.out.println(m.get());
        // m.d = 2.0;
    }
}
```

final 关键字

- final 参数
  - 函数不能修改参数的引用 .

```
class FinalArgs {
    public static void set(final int [ ] a) {
        a[0] = 1;
        // a = new int [10];
    }
    public static void main(String []args){
        int [ ]a = new int[10];
        FinalArgs.set(a);
    }
}
```

final 关键字

- final method
  - 不能被重写

```
class MyType {
    public int i;
    public double d;
    public char c;
    final void set(double x) { d = x;}
    protected void set(int y) {i = y;}
    public double get() { return d; }
}
```

```
public class MySubType extends MyType{
    // can't overide
    /* public void set(double d){
        System.out.println("Sub-class set");
        i = int(d);
    } */
    public static void main(String [ ]args){
        MySubType ms = new MySubType();
        MyType m = ms;
        m.set(1.0);
    }
}
```

final 关键字

- final class
  - 不能被继承

```
final class MyType {
    public int i;
    public double d;
    public char c;
    final void set(double x) { d = x;}
    protected void set(int y) {i = y;}
    public double get() { return d; }
}
```

```
// can not be extended
/*
public class MySubType extends MyType{
    public void set(double d){
        System.out.println(“Sub-class set”);
        i = int(d);
    }
    public static void main(String [ ]args){
        MySubType ms = new MySubType();
        MyType m = ms;
        m.set(1.0);
    }
}*/
```

不可变类型

- 不可变类型 (immutable)
  - 类型的对象一旦创建就不能被改变
  - 例子 String 类 , Integer 类 , Float 类 ...

```
String s = “Hello World”;
System.out.println(s.toUpperCase());
System.out.println(s);
```

- 可变类型 (mutable)
  - 例子 MyType, 数组

```
MyType m = new MyType();
System.out.println(s.get());
m.set(1.0);
System.out.println(s.get());
```

```
int []a = {1, 2, 3};
System.out.println(a[0]);
a[0] = 1
System.out.println(a[0]);
```

final 关键字

- 不可变 (immutable)
  - 优点：易于使用，易于 debug，易于维护
  - 缺点：空间 / 时间消耗

final 关键字

- final 关键字
  - 帮助构造不可变对象
  - Let’s try it.

```
class MyType {
    final public int i;
    final public double d;
    final public char c;

    public MyType set(double x) {
        return new MyType(i, x, c);
    }
    public double get() { return d; }

    public MyType(int x, double y, char z){
        i = x;
        d = y;
        c = z;
    }

    public static void main(String []argv){
        MyType m = new MyType(1, 2, '\0');
        MyType n = m.set(3);
        System.out.println(n.d);
        System.out.println(m.d);
    }

}
```

http://new-play.tudou.com/v/571411926.html?