方法的重载

- 方法的重载是指一个类中可以定义有相同的名字，但参数不同的多个方法。 调用时，会根据不同的参数表选择对应的方法。
- 例如：在 Person 类中添加如下方法：

```
void info() {
    System.out.println("My id is "+id);
}
void info(String t) {
    System.out.println(t+" "+id+" "+age);
}
```

TestOverload1.java

```
//运行如下程序
public class Test {
    public static void main(String args[]) {
        Person p = new Person(1,20);
        p.info();
        p.info("hello");
    }
}
```

TestOverload2.java

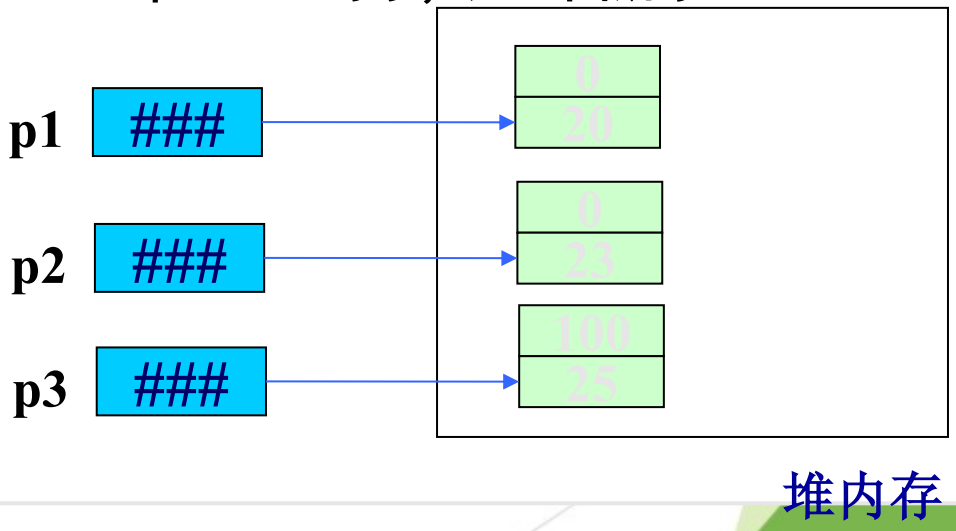
构造方法的重载

- 与普通方法一样，构造方法也可以重载：
- 例如：修改 Person 类的构造方法为：

```
Person() {
    id = 0;
    age = 20;
}
Person(int i) {
    id = 0;
    age = i;
}
Person(int n,int i){
    id = n;
    age = i;
```

课堂练习

利用修改过后的 Person 类；编写程序，分别用三种构造方法创建三个 person 对象，如下图所示

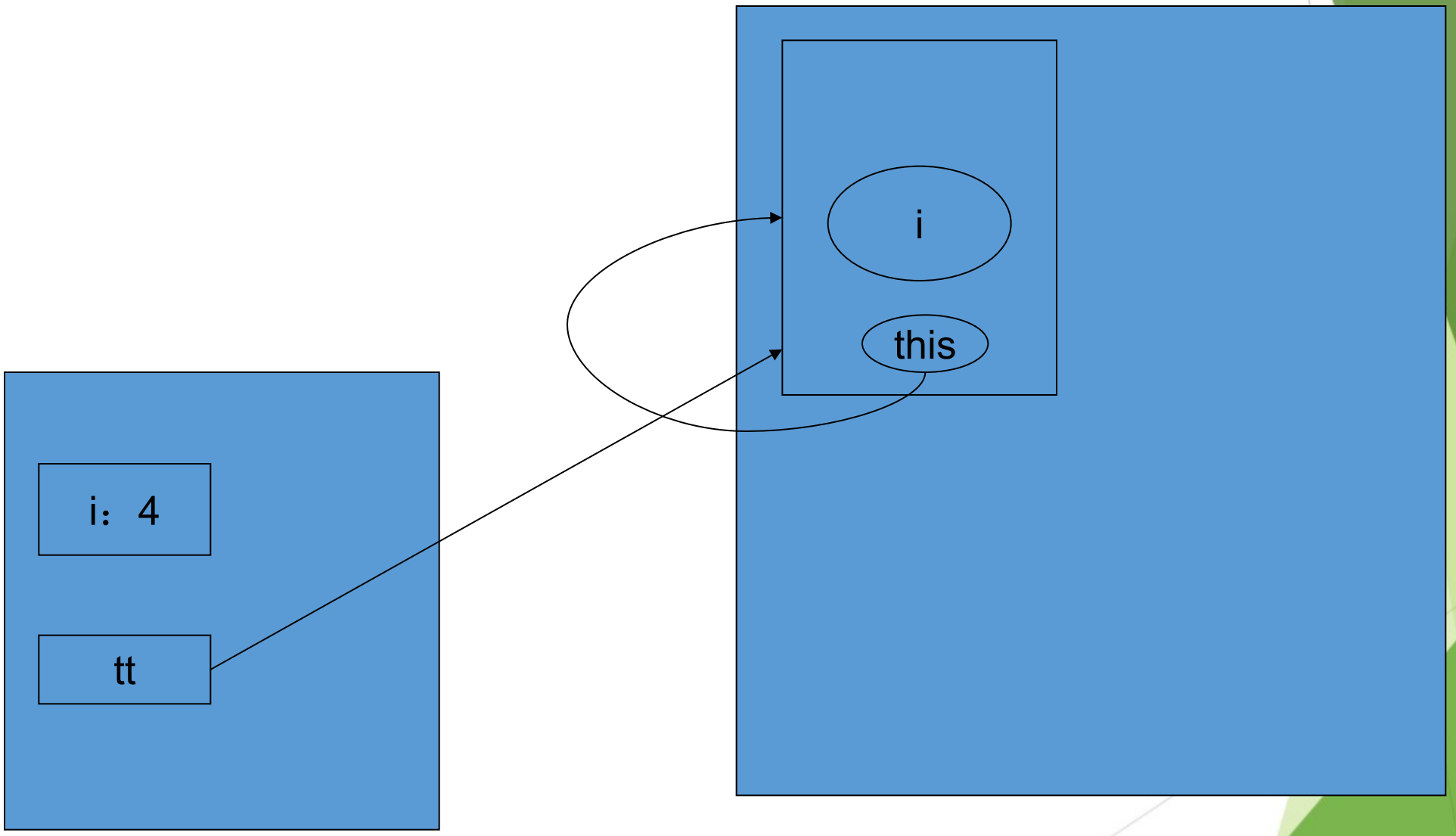


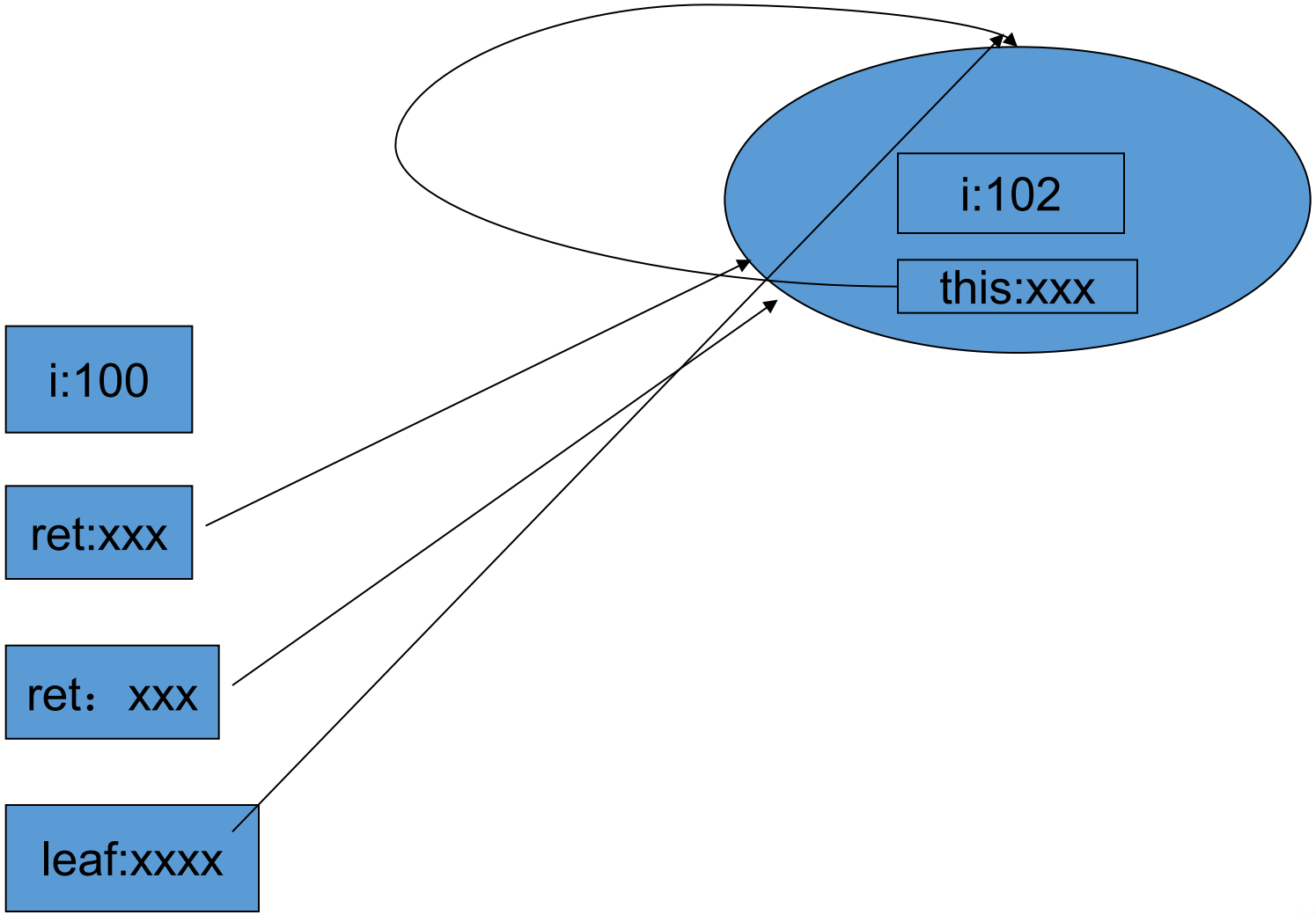
this 关键字

TestThis.java

- 在类的方法定义中使用的 this 关键字代表**使用该方法的对象**的引用。
- 当必须指出当前使用方法的对象是谁时要使用this。
- 有时使用this可以处理方法中成员变量和参数重名的情况。

```
public class Leaf{
    int i = 0;
    Leaf(int i) { this.i = i; }
    Leaf increament(){
        i++;
        return this;
    }
    void print(){ System.out.println("i = "+i); }
    public static void main(String[] args){
        Leaf leaf = new Leaf(100);
        leaf.increament().increament().print();
    }
}
```


i
this
i：4
tt


i:102
this:xxx
i:100
ret:xxx
ret：xxx
leaf:xxxx

static 关键字

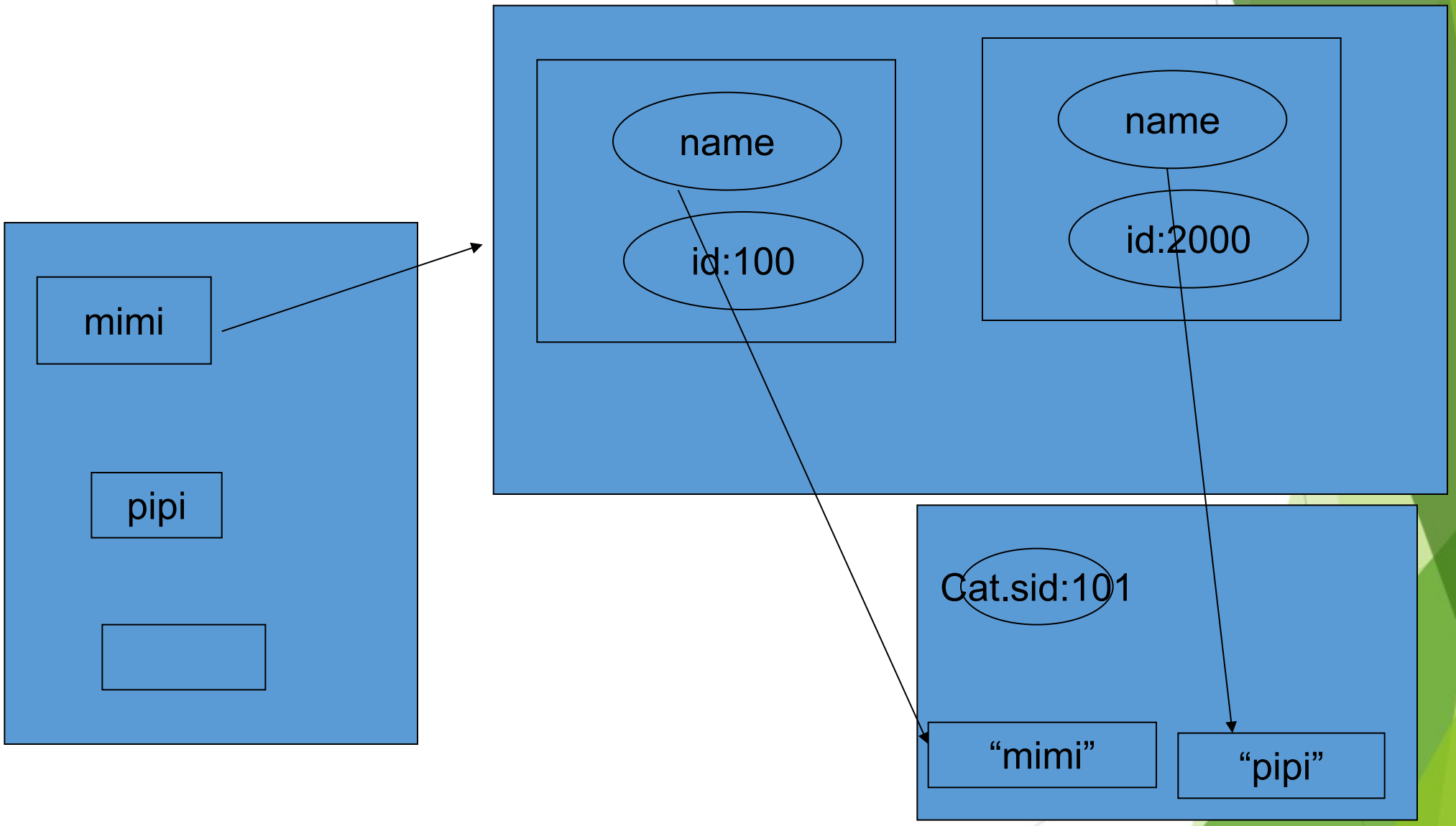
- 在类中，用static声明的**成员变量**为静态成员变量，它为该类的公用变量，在第一次使用时被初始化，对于该类的所以对象来说，static成员变量只有一份。

- 用static声明的方法为**静态方法**，在调用该方法时，不会将对象的引用传递给它，所以在static方法中不可访问非static的成员。

➢静态方法不再是针对于某个对象调用，所以不能访问非静态成员。

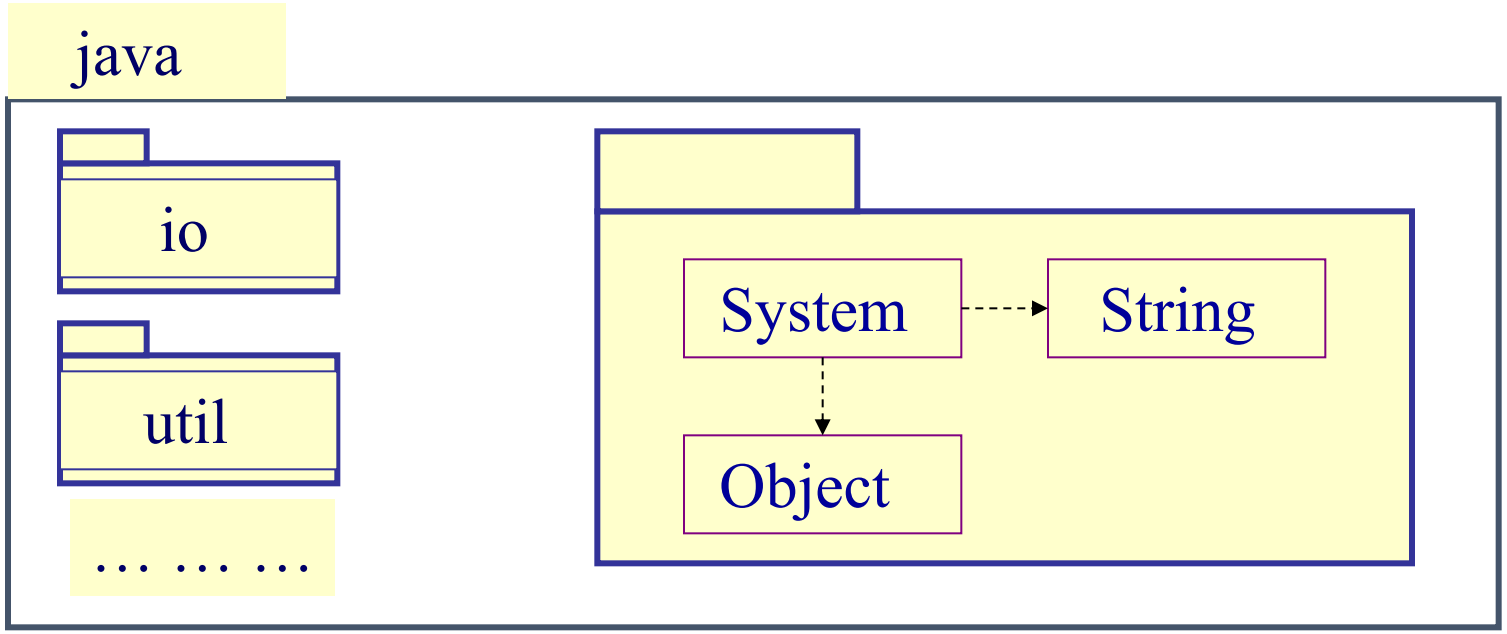
- 可以通过对象引用或类名（不需要实例化）访问静态成员。

Cat.java


mimi
pipi
name
id:100
name
id:2000
Cat.sid:101
“mimi”
“pipi”

package 和 import语句

- 为便于管理大型软件系统中数目众多的类，解决类的命名冲突问题，Java引入包（package）机制，提供类的多重类命名空间。



package 和 import语句

- package 语句作为 Java 源文件的第一条语句，指明该文件中定义的类所在的包。（若缺省该语句，则指定为无名包）。
  它的格式为：
  - package pkg1[.pkg2[.pkg3…]];
- Java编译器把包对应于文件系统的目录管理，package语句中，用‘.’来指明包(目录)的层次，例如使用语句
  - package com.sxt;
    - 则该文件中所以的类位于. \com\sxt 目录下

package 和 import语句

- 如果将一个类打包，则使用该类时，必须使用该类的全名（例如：com.sxt.MyClass），Java编译器才会在找到该类。
- 也可以使用 import 在文件的开头引入要使用到的类；例如：

```
import com.sxt.MyClass;
import java.util.*;//引入java.util包中所有的类
        ... ... ...
  MyClass myClass = new MyClass(); //可以直接使用类名
        ... ... ...
```

- 可以不需要用import语句直接使用 java.lang 包中的类。

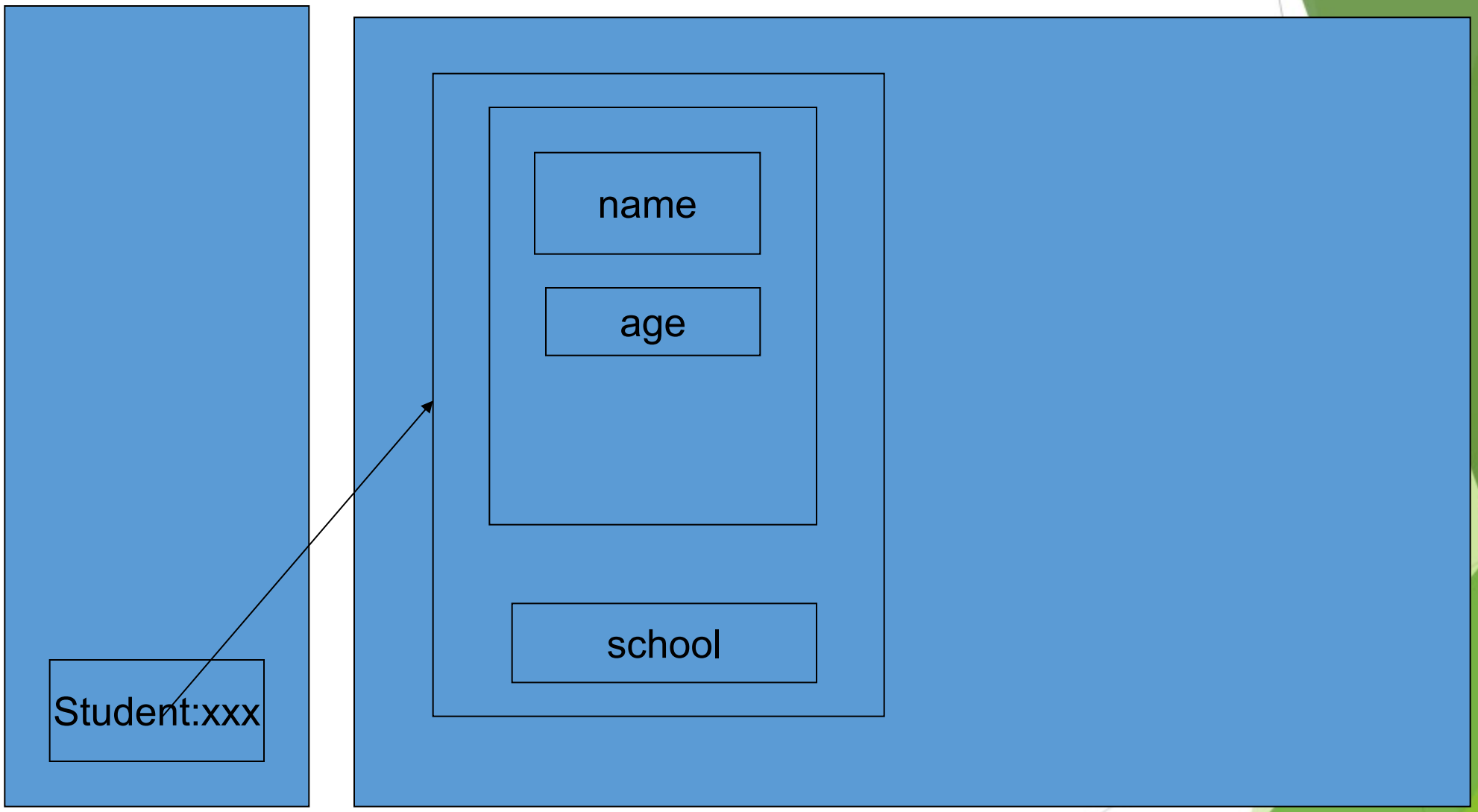
类的继承与权限控制

- Java 中使用 extends 关键字实现类的继承机制
- 通过继承，子类自动拥有了基类（superclass）的所有成员（成员变量和方法）。
- Java只支持单继承，不允许多继承：
  - 一个子类只能有一个基类 一个基类可以派生出多个子类

TestPerson.java

```
//继承中的权限控制

class Parent{
    private     int  n_private =1;
                int  n_friendly = 2;
    protected   int  n_protected = 3;
    public      int  n_public = 4;
}
class Child extends Parent {
    public void f(){
        //n_private =10;
        n_friendly = 20;
        n_protected = 30;
        n_public = 40;
    }
}
```


name
age
school
Student:xxx

访问控制（Access Control）

- Java权限修饰符 public protected private 置于类的成员定义前，用来限定其他对象对该类对象成员的访问权限。

修饰符	类内部	同一个包	子类	任何地方
private	Yes			
default|package	Yes	Yes		
protected	Yes	Yes	Yes	
public	Yes	Yes	Yes	Yes

- 对于class的权限修饰只可以用 public 和 default。（inner class除外）
  - public类可以在任意地方被访问
  - default类只可以被同一个包内部的类访问

TestAccess.java　　**编写程序验证Java中的访问控制。**

方法的重写(OverWrite OverRide)

- 在子类中可以根据需要对从基类中继承来的方法进行重写。
- 重写方法必须和被重写方法具有相同方法名称、参数列表和返回类型。
- 重写方法不能使用比被重写方法更严格的访问权限。

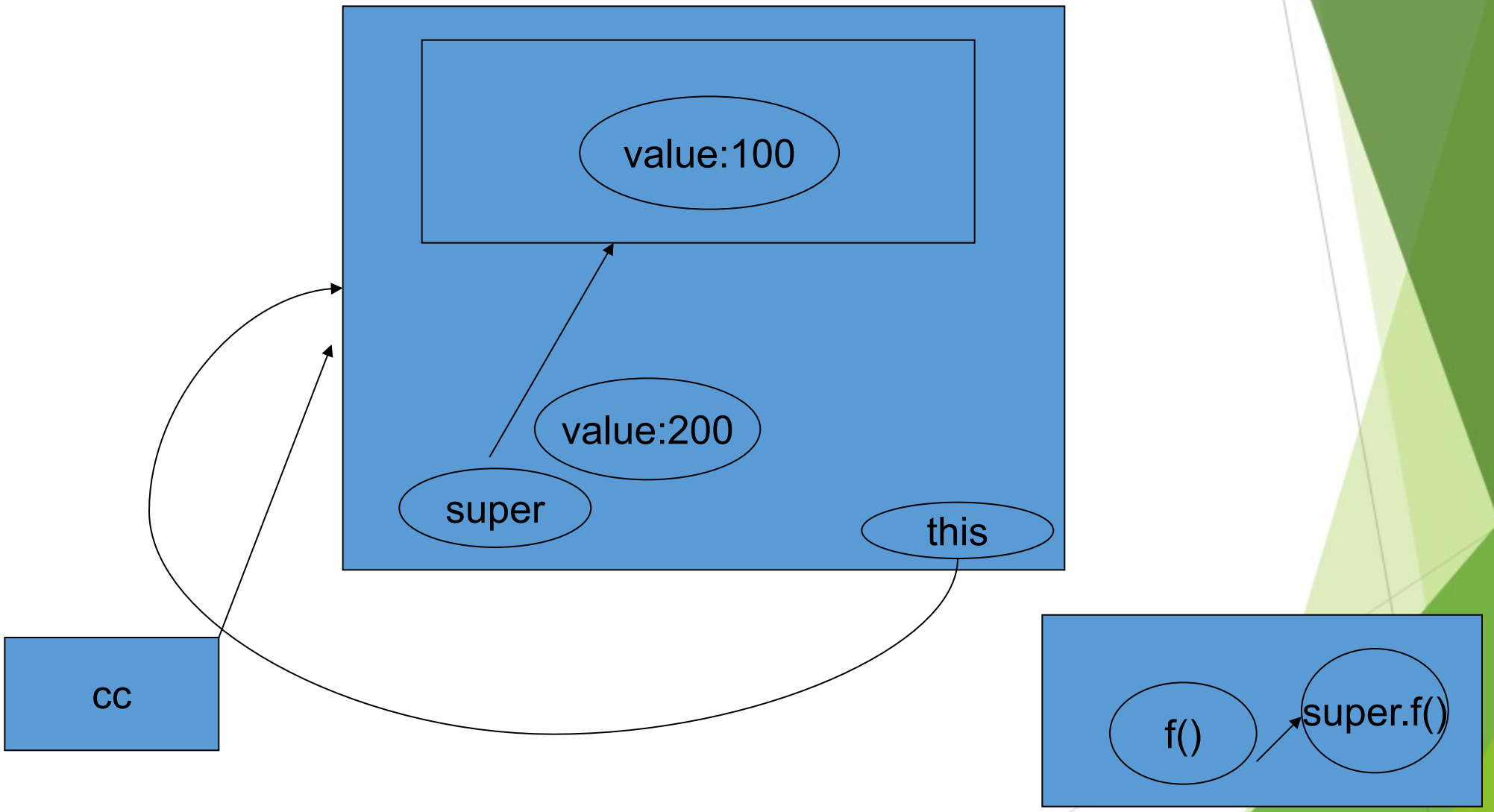
TestOverWrite/TestOverWrite.java

super 关键字

- 在Java类中使用super来引用基类的成分；例如：

```
class FatherClass {
    public int value;
    public void f(){
        value = 100;
        System.out.println
        ("FatherClass.value="+value);
    }
}
class ChildClass extends FatherClass {
    public int value;
    public void f() {
        super.f();
        value = 200;
        System.out.println
              ("ChildClass.value="+value);
        System.out.println(value);
        System.out.println(super.value);
    }
}
```

TestInherit.java


value:100
value:200
super
this
cc
f()
super.f()

继承中的构造方法

- 子类的构造的过程中**必须**调用其基类的构造方法。
- 子类可以在自己的构造方法中使用super(argument_list)调用基类的构造方法。
  - 使用this(agument list)调用本类另外的构造方法
  - 如果用super,必须写在方法的第一行
- 如果子类的构造方法中没有显式地调用基类构造方法，则系统默认调用基类无参数的构造方法。
- 如果子类构造方法中既没有显式调用基类构造方法，而基类中又没有无参的构造方法，则编译出错。

TestSuperSub.java

课 堂 练 习

```
class A {
    protected void print(String s){
        System.out.println(s);
    }
    A(){print("A()");}
    public void f() {print("A:f()");}
}

class B extends A{
    B(){print("B()");}
    public void f() {print("B:f()");}
    public static void main(String arg[]) {
        B b = new B(); b.f();
    }
}
```

TestPerson2/Test.java

分析上题的输出结果，体会构造函数和一般成员函数在继承中的区别。