day6作业：

1. 写出下列代码的运行结果

public class ExerciseTest {

    public static void main(String[] args){

        ExerciseTest f = new ExerciseTest();

     System.out.println(f.add("4", "5"));

    }

    public int add(int x, int y) {

        return x + y;

    }

    public String add(String x,String y) {

        return x + y;

    }

}

输出 45

2. 关于构造方法，下面说法正确的是

A.构造方法不能带有参数

B.构造方法的名称必须和类名相同

C.构造方法可以定义返回值

D.构造方法不能重载

答案： B.

3. 指出下面代码的编译错误，并说明原因

//哺乳动物

public class Mammals {

}

//鸟类

public class Birds {

}

//蝙蝠

public class Bat extends Mammals,Birds{

}

答：java 只能单继承，不能同时继承两个父类

4. JVM如何管理内存，分成几个部分？分别有什么用途？说出下面代码的内存实现原理：

main

Foo foo = new Foo();

foo.f();

new Foo(); 是创建了一个对象，将对象存储至堆内存

Foo foo； 是一个局部变量，存储在栈内存

= 赋值运算符是把创建的对象地址 赋值给foo变量

foo.f() 调用对象中的f()：会使用栈内存

答： 堆内存 （所有new 对象）、栈内存 （方法运行时用，存放方法运行时的局部变量、参数值）、方法区（类的定义）

 5. 说出下面代码的输出结果，并解释原因

public class Sub extends Base {

    String color; // = null 第五步

    public Sub(double size, String name, String color) {

        super(size, name); // 第四步

         this.color = color; // 第六步

    }

    public static void main(String[] args) {

     Sub s = new Sub(5.6, "测试对象", "红色"); // 第一步

  System.out.println(s.size + "--" + s.name + "--" + s.color);//七

    }

}

class Base {

    double size; //= 0 第二步

    String name; //=null 第三步

public Base(double size, String name) { // 第四步

       this.size = size;

        this.name = name;

    }

}

输出结果 : 5.6--测试对象--红色

6. 说出下面代码的输出结果，并解释原因

//鸵鸟

public class Ostrich extends Bird{

    public void fly(){

        System.out.println("我只能在地上奔跑...");

    }

    public static void main(String[] args) {

        Bird os=new Ostrich();

        os.fly();

    }

}

class Bird {

    public void fly(){

        System.out.println("我在天空里自由自在的飞翔...");

    }

}

输出： 我只能在地上奔跑...

7. 说出下面代码的输出结果，并解释原因

public class SlowPoint extends Point {

    public void move(int dx, int dy) {

        System.out.println("SlowPoint move parameter");

        move();

    }

    public static void main(String[] args) {

        SlowPoint sp=new SlowPoint();

        sp.move(10,20);

    }

}

class Point {

    public void move(int dx, int dy) {

        System.out.println("Point move parameter");

    }

    public void move(){

        System.out.println("Point move ");

    }

}

SlowPoint move parameter

Point move

8.关于package和import语句，下面说法错误的是：

A. package 提供了一种命名机制，用于管理类名空间

B. 定义类时，除了定义类的名称以外，必须要指定一个包名

C. import语句用于导入所需要的类

D.同时使用不同包中相同类名的类，包名不能省略

B.

类名有冲突的情况下，必须写完整的包名.类名

9. 关于public和private，下面说法错误的是：

A.private修饰的成员变量和方法仅仅只能在本类中访问

B. public修饰的成员变量和方法可以在任何地方访问

C. private修饰的成员变量和方法可以在本类和子类中访问 （protected）

D.public 修饰的成员变量和方法只能在同一个包中访问 (default)

答案： C,D

10. 关于protected 关键字，下面说法错误的是：

A. 用protected修饰的成员变量和方法可以被子类及同一个包中的类使用

B.使用protected 关键字修饰的访问控制，即默认访问控制

C.默认访问控制的成员变量和方法可以被同一个包中的类访问

D. 用protected修饰的成员变量和方法只能被子类使用

答案： B. D

11. 关于static 关键字，下面说法正确的是：

A. 用static修饰的成员变量是属于对象的数据结构 // 属于类

B.在static方法中，可以访问非static成员(对象成员)

C. static成员变量存储在堆中 //方法区

D. 一个类的static成员变量只有“一份”，无论该类创建了多少对象

public class Test{

int age;

static void test() {

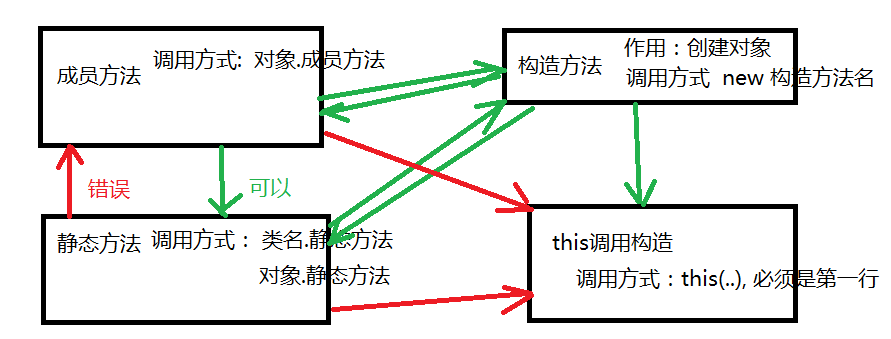
System.out.println(age)

}

}

答案： D

## 方法之间访问的限制:



day5 作业：

1. 有以下代码

class MyClass{

int value;

}

public class TestRef{

public static void main(String args[]){

int a = 10; int b = a; b ++ ;

System.out.println(a);

MyClass mc1 = new MyClass();

mc1.value = 10; // mc1.value = 10

MyClass mc2 = mc1; // 地址赋值，mc2与mc1都指向同一个对象

mc2.value ++;

System.out.println(mc1.value);

}

}

请写出编译运行后的结果

10

11

1. 有以下代码

class ClassA{

int value = 10;

}

public class TestReturnRef{

public static void main(String args[]){

ClassA ca = new ClassA(); // 001122

ca = getObject(); // 001133

ca = getObject(); // 001144

ca = getObject(); // 001155

System.out.println(ca.value);

}

public static ClassA getObject(){

ClassA newObject = new ClassA();

newObject.value += 10; // 20

return newObject;

}

}

编译运行TestReturnRef 程序，结果为?

20

3. 有以下代码,请选择正确答案：

class ClassA{ public ClassA(){ System.out.println("ClassA()"); } } class ClassB{ public ClassB(){ System.out.println("ClassB()"); } } class ClassC{

ClassA a = new ClassA(); // 1

ClassB b; // 2

public ClassC(){

System.out.println("ClassC()");

b = new ClassB();

}

}

public class TestConstructor{

public static void main(String args[]){

ClassC cc = new ClassC();

}

}

1. 编译不通过
2. 输出ClassA() ClassB() ClassC()
3. 输出 ClassA() ClassC() ClassB()
4. 输出 ClassC() ClassB() ClassA()

答案： C

4. 请选择正确答案：

class ClassA {

int value;

}

public class Test4 {

public static void main(String args[]) {

int value = 10;

changeInt(value);

System.out.println(value);

ClassA ca = new ClassA();

ca.value = 10;

changeObject(ca);

System.out.println(ca.value);

}

public static void changeInt(int value) {

value++;

}

public static void changeObject(ClassA ca) {

ca.value++;

}

}

编译运行Test4时，结果是

1. 编译出错

B. 输出 10 11

C. 输出 10 10

D. 输出 11 11

答案： B

5.请选择正确答案：

class ClassA {

int value;

}

public class Test4 {

public static void main(String args[]) {

ClassA ca = new ClassA();

changeValue(ca);

System.out.println(ca.value);

changeRef(ca);

System.out.println(ca.value);

}

public static void changeValue(ClassA ca) {

ca.value = 100;

}

public static void changeRef(ClassA ca) {

ca = new ClassA();

ca.value = 200;

}

}

编译运行Test4，结果为

1. 编译不通过
2. 输出100 200
3. 输出100 100
4. 输出0 200

答案： C

## 类型转换:

## 基本类型转换:

从小到大....

byte short int long float double

char

小的类型可以直接赋值给大的类型

short b = 10;

int a = b;

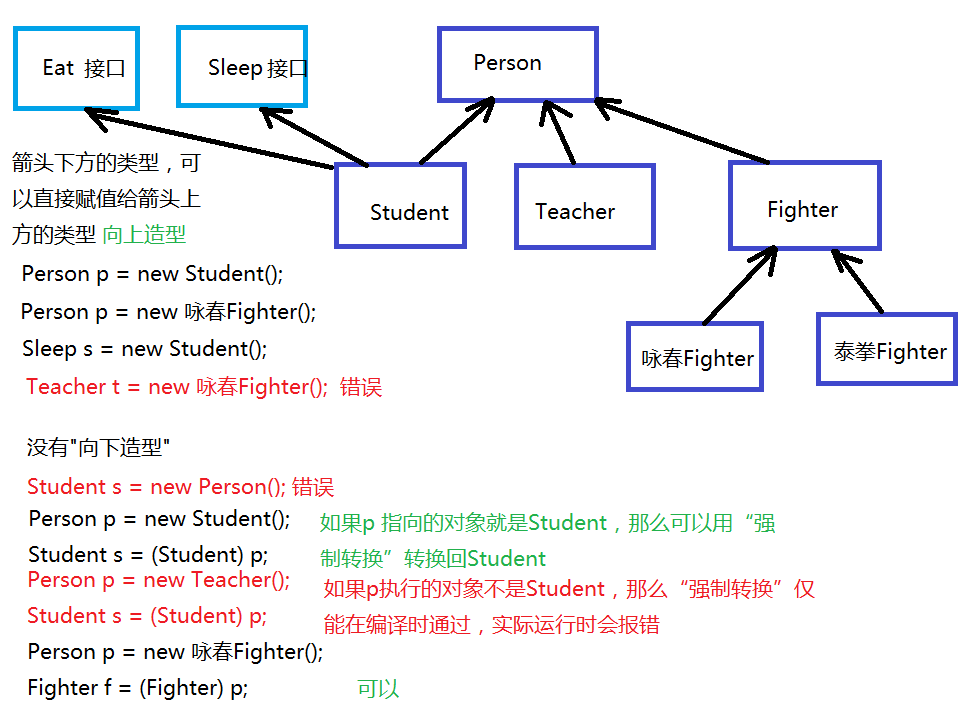
大的类型不能直接赋值给小的类型，需要强制转换

short b = 129;

byte a = b; //不行

byte a = (byte) b; // 会有精度损失

## 引用类型：



## final

### 修饰类

final加在类上，表示类不能被继承

public final class Person {

}

不能与abstract关键字一起用,他们有冲突

## 修饰变量

final 加在变量上，表示这个变量就变为一个常量（初始化之后就不能改动了）

public class TestConstant {

// 修饰全局变量

final static int d = 10;

// 在静态代码块赋值

final static int e;

static {

e = 20;

}

// final 修饰成员变量必须手动赋初值

// 1.可以直接赋值，

final int a = 10;

// 2. 可以用初始化代码块赋值

final int b;

{

b = 20;

}

// 3. 可以用构造方法赋值

final int c;

public TestConstant() {

c = 30;

}

public static void main(String[] args) {

// final 修饰的是常量，不能重新赋值

final int i = 10;

// i = 20; // 错误

TestConstant test = new TestConstant();

// test.a = 20; // 错误

// test.c = 1; // 错误

TestConstant.d = 10; //错误

}

}

public interface MyInterface {

public final static int i = 10;

int j = 20; // 会帮你加上public final static 关键字

void test(); // 会帮你加上public 关键字

}

### 修饰方法

final 加在方法，则此方法不能被重写

public class Animal {

final public void test() {

System.out.println();

}

}

class Dog extends Animal {

public void test() { // 错误

System.out.println("子类");

}

}

## java的垃圾回收机制

new 类型() 存储于堆内存： 分代垃圾回收机制

方法调用时，使用栈内存: 方法参数，局部变量占用的内存，在方法调用结束被释放

类的定义，全局变量，使用的是方法区: 直到程序运行结束，方法区内存不会被释放

### 垃圾回收过程

刚创建的对象，放入伊甸园区，

当垃圾回收发生时，如果伊甸园区的对象仍有被使用，把他放入幸存区

当经过若干次垃圾回收后，如果幸存区对象仍然有人使用，会把它放入老生代

在新生代发生的垃圾回收 minor GC (GC Garbage Collect 就是垃圾回收)

在老生代发生的垃圾回收 full GC （如果发生，可能会导致应用程序暂停一会）

### 提前标记对象可以被回收的技巧

Test a = new Test();

...

a = null; // 给变量赋一个null值，就是告诉jvm a不再引用 Test()对象了

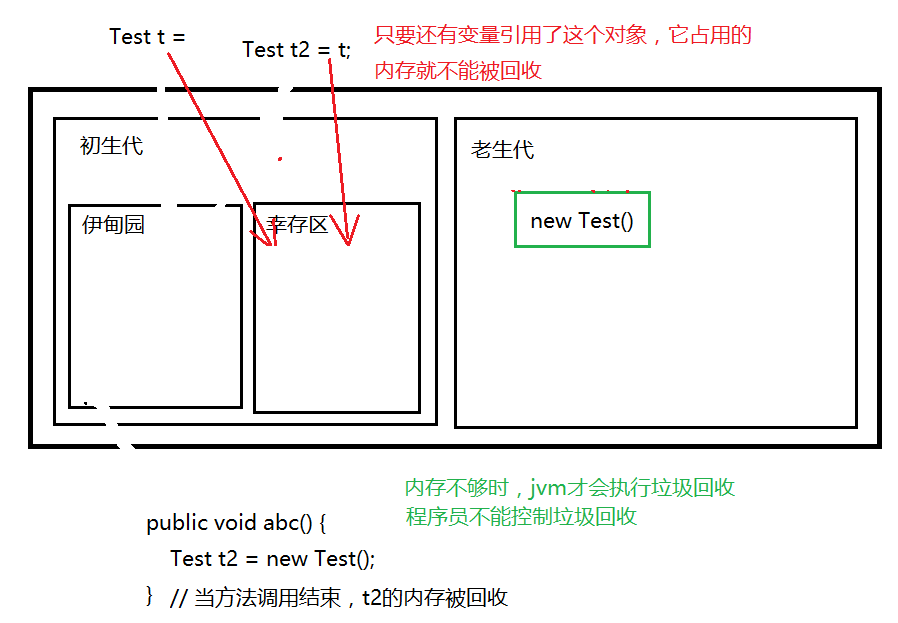
好多代码

// a 如果不加 a= null 必须等到方法运行结束，jvm 才知道没有人使用Test()对象

### System.gc();

// 只是建议jvm ，做一次垃圾回收, jvm 不一定执行垃圾回收

垃圾回收也是java 相对于原来C++语言的优点，C++得编程实现内存回收



## 内部类

在一个类内部定义其他类，新定义的类称为内部类

### 成员内部类

// 外部类

public class Outer {

private int i;

private void test() {

System.out.println("外部类方法");

}

// 成员内部类, 提供封装

class Inner {

void inner() {

System.out.println("访问外部类的成员变量： " + i);

// 调用外部类方法

test();

}

}

}

特点： 成员内部类可以直接访问外部类的私有的方法和属性

了解：

import day08.inner.Outer.Inner;

public class Use {

public static void main(String[] args) {

// 创建外部类实例:

Outer outer = new Outer();

// 创建成员内部类实例， 与外部类不一样

Inner inner = outer.new Inner();

}

}

对外部类，类上的访问修饰符只能是public 或是default

对内部了，类上的访问修饰符可以是public ,protected, default , private

### 静态内部类

public class StaticOuter {

static int i = 10;

// 其他的使用者向创建静态内部类。创建方法不同于成员内部类

static class StaticInner {

public void test() {

System.out.println(StaticOuter.i);

}

}

}

// 创建静态内部类的实例

StaticInner inner2 = new StaticOuter.StaticInner();

### 方法内部类

// 演示方法内部类

public class MethodInner {

public void test() {

// 方法内部类

//类的定义只能在方法内部引用，出了方法，类的定义就不可见了

class Inner {

int x;

void test2() {

}

}

Inner inner = new Inner();

System.out.println(inner.x);

inner.test2();

}

public void test2() {

new Inner(); // 错误： 其他方法中不能访问

}

}

### 匿名内部类

匿名：没有自己的名字

public interface Eat {

public void eat();

}

class Person{

String name;

void test() {

System.out.println("父类方法");

}

}

public class Niming {

public void test() {

// 匿名内部类和方法类，可以引用方法内的局部常量

final int v = 10;

// 必须基于父类，或者基于接口，来创建匿名内部类

Person p = new Person() {

void test() {

System.out.println("子类方法");

System.out.println("引用方法的局部变量：" + v);

}

};

/\*

Person() {} 其实是Person的子类

new Person() {} 是创建了子类的对象

\*/

System.out.println(p.name);

p.test();

// 基于接口创建匿名内部类，创建了一没有名字的类，它实现了接口中的抽象方法

Eat e = new Eat() {

public void eat() {

System.out.println("实现了抽象方法");

}

};

e.eat();

}

public static void main(String[] args) {

Niming n = new Niming();

n.test();

}

}

JDK API (application programming interface)(应用程序编程接口)

比较重要的包

1） java.lang.\*

Boolean

Float

Double

Integer

Short

Byte

Long

Character

String

Math 数学运算的方法

java.lang 包下的类，不需要用import语句导入就能使用

1. java.util.\* 工具类，日期，集合
2. java.io.\* 输入输出
3. javax.sql.\* x表示扩展，可以让我们访问数据库

## java.lang.Object

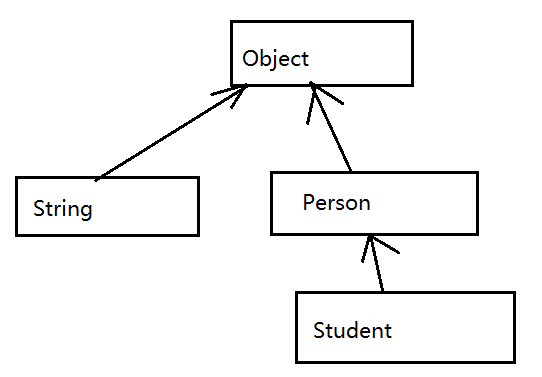
它是除了基本类型以外，所有类的顶层父类，会由编译器给我们定义的类添加extends Object

class Person{

}

class Student extends Person｛

｝



### 可以把任意一个对象赋值给Object类型的变量：

Object o = new Studnet();

String s = "sss";

Object o2 = s;

### .clone() 方法

创建一个新的对象，把旧对象的属性拷贝新的对象中

前提：类要实现一个Cloneable （可以被克隆）

public class Dog implements Cloneable {

int age;

String name;

public static void main(String[] args) throws CloneNotSupportedException {

Dog d1 = new Dog();

d1.name = "藏獒";

d1.age = 8;

Dog d2 = (Dog)d1.clone();

System.out.println(d2.name);

System.out.println(d2.age);

// d2 是一个新对象，只是把d1的值复制了一份

System.out.println("d2是否是新的对象:" + (d1 == d2));

}

}

### .getClass() 方法

用来获取对象的类型

### .equals() 方法

用来判断两个对象的值是否相等

public boolean equals(Object obj) {

return (this == obj);

}

返回是boolean值，表示对象是否相等

this == obj this是对象自己、obj是其他任意对象, == 是判断他们的地址是否相等，默认是地址一样认为两个对象相等、如果地址不一认为两个对象不等

自己重写equals方法(为了判断值是否相等)

public boolean equals(Object obj) {

// 地址相等，肯定相等

if (this == obj)

return true;

// 对方为null，肯定不等

if (obj == null)

return false;

// 类型不等，肯定不等

if (getClass() != obj.getClass())

return false;

// 类型肯定能够转换为Teacher类型

Teacher other = (Teacher) obj;

// 判断每个属性是否相等

if (age != other.age)

return false;

if (name == null) {

if (other.name != null)

return false;

} else if (!name.equals(other.name))

return false;

return true;

// 如果这些属性的值都相信，就可以认为两个对象相等

}

### .finalize()

当对象被垃圾回收时，会调用这个对象的finalize方法

因为无法确定对象什么时候被垃圾回收，因此fianlize方法的内容什么时候被调用，程序员无法控制

protected void finalize() {

System.out.println("在这个对象被垃圾回收时调用");

}

### public String toString() {

### return "转换结果" }

将对象用字符串表示, 主要方便调试

public class Cat {

String name;

int age;

public static void main(String[] args) {

Cat c = new Cat();

c.name = "波斯猫";

c.age = 2;

String s = c.toString(); // 继承的toString

System.out.println(s);

// Object 父类中toString返回这个对象的地址

// 简化

System.out.println(c); // 打印对象时，会自动调用该对象的toString方法

// System.out.println(c.toString());

String str = "hello, ";

System.out.println(str + c);

// System.out.println(str + c.toString());

}

@Override

public String toString() {

return "Cat [name=" + name + ", age=" + age + "]";

}

}

### 包装类

Double 称为double 的包装类

double 基本类型

Double d = new Double(4.5); // 创建了Double对象, 占用都是堆内存

// 测试包装类

public class Wrapper {

public static void main(String[] args) {

// 作用1： 用来转换字符类型到Double类型

String a = "4.5";

Double d = Double.valueOf(a);

System.out.println(d + 0.5);

double d2 = Double.parseDouble(a);

System.out.println(d2 + 1.0);

// 作用2: 让 Double 类型也能当作Object对象来使用

Object o = new Double(5.6);

//

double d3 = 5.6;

Double d4 = d3; // 自动装箱：把基本double转换为Double对象

// 上面的代码，被替换为下方的代码

// Double d4 = Double.valueOf(d3);

Double d5 = new Double(1.1);

double d6 = d5; // 自动拆箱,把Double对象转换为基本double

// 上面的代码，被替换为下方的代码

// double d6 = d5.doubleValue();

}

}

int => Integer

double => Double

float => Float

long => Long

short => Short

byte => Byte

char => Character

boolean => Boolean

自动拆箱装箱：基本类型可以被直接赋值给包装类型，反之也可以