一、选择题

1.知识点：

(1)this不能用在静态方法中,可以在非静态方法中用this：

示例1：

**public** **class** Test1 {

**private** **float** f =10.6f;

**int** i=16;

**static** **int** *si*=10;

**void** a() {

**this**.*si*=15;//使用this关键字

}

**public** **static** **void** main(String[] args) {

Test1 test1=**new** Test1();

test1.a();

System.***out***.println(*si*);//输出15

}

}

示例2：

**public** **class** Test1 {

**private** **float** f =10.6f;

**int** i=16;

**static** **int** *si*=10;

**static** **void** a() {

**this**.si=15;//报错，不能在静态方法中用this

}

**public** **static** **void** main(String[] args) {

Test1 test1=**new** Test1();

test1.*a*();

System.***out***.println(*si*);

}

}

(2)静态方法可以直接调用，非静态方法必须依赖于具体的对象调用.

示例1：

**public** **class** Test1 {

**private** **float** f =10.6f;

**int** i=16;

**static** **int** *si*=10;

**static** **void** a() {

*si*=15;

}

**public** **static** **void** main(String[] args) {

Test1 test1=**new** Test1();

*a*();//静态方法可以直接调用，也可以test1.*a*()这样调用

System.***out***.println(*si*);//输出15

}

}

示例2：

**public** **class** Test1 {

**private** **float** f =10.6f;

**int** i=16;

**static** **int** *si*=10;

**void** a() {

*si*=15;

}

**public** **static** **void** main(String[] args) {

Test1 test1=**new** Test1();

a();//报错，必须是test1.*a*()

System.***out***.println(*si*);

}

}

2.复习static的使用范围：成员变量(类变量)、方法(类方法)、内部类

3.A.接口中方法默认是公有抽象的，即public abstract，不能有方法体，否则会报错。若用default修饰方法，方法才能有方法体。

B.接口中方法可以使用的修饰符有：public，abstract，default，static，strictfp

D.接口中的变量默认是公有常量，即public static final类型变量，接口中变量可用修饰符有：public，static，final，接口中的变量必须初始化。

4.重载和重写(覆盖)的详解

参考链接:<https://www.runoob.com/java/java-override-overload.html>

注意重写的第二个bark()示例很有迷惑性。尤其要注意重写的规则。有关super用法也可参见上面链接。

6、7、8题:

注意6、7题的选项混淆了多重继承和多继承，这两者是不同的概念，注意区分，参考链接：

<https://www.runoob.com/java/java-interfaces.html>

<https://www.runoob.com/java/java-inheritance.html>

要点总结：一个类可以实现(implements)多个接口(8题);

一个子类只能继承(extends)一个父类,即Java语言中类间的继承关系是单继承(6题);

一个子接口可以继承(extends)多个父接口, 即Java语言中接口间的继承关系是多继承(7题)。

9.参考(4)组知识点总结

10.另外参考(5)组第一题对this的总结,super用法参考4题。

二、填空题

2. 在面向对象程序设计中,消息分为两类:公有消息和私有消息。假设有一批消息同属于一个对象,其中一部分消息是由其它对象直接向它发送的,称为公有消息(我的理解:在其他类调用该类的消息);另一部分消息是它向自己发送的,称为私有消息(我的理解:在该类中调用该类的消息)。

公有消息与私有消息的确定, 与消息要求调用的方法有关。如果被调用的方法在对象所属的类中是在public下说明的, 则为公有; 是在private下说明的,即为私有。

根据我的理解给几个示例：

例1：

**public** **class** Test1 {

**public** **static** **void** main(String[] args){

A o = **new** A();

System.***out***.println(o.test());//报错,提示方法不可见

}

}

**class** A{

**private** **int** test(){

System.***out***.println("test1");

**return** 1;

}

}

例2：公有消息

**public** **class** Test1 {

**public** **static** **void** main(String[] args){

A o = **new** A();

System.***out***.println(o.test());

}

}

**class** A{

**public** **int** test(){

System.***out***.println("test1");

**return** 1;

}

}

例3：私有消息

**public** **class** Test1 {

**public** **static** **void** main(String[] args){

Test1 o = **new** Test1();

System.***out***.println(o.test());

}

**private** **int** test(){

System.***out***.println("test1");

**return** 1;

}

}

9.抽象类相关：<https://www.runoob.com/java/java-abstraction.html>

17.示例一：

**public** **class** Test1 {

**public** **static** **void** main(String[] args){

Person p=**new** Person();//输出Dick 12

Person p1=**new** Person("Jack");//输出Jack 16

}

}

**class** Person {

**private** String name;

**private** **int** age;

**public** Person() {

**this**("Dick", 12);

System.***out***.println(name+" "+age);

}

**public** Person(String n) {

**this**(n, 16);

System.***out***.println(name+" "+age);

}

**public** Person(String n, **int** a) {

name = n;

age = a;

}

}

示例二：

**public** **class** Test1 {

**public** **static** **void** main(String[] args){

Person p=**new** Person();

Person p1=**new** Person("Jack");

}

}

**class** Person {

**private** String name;

**private** **int** age;

**public** Person() {

**this**("Dick", 12);

System.***out***.println(name+" "+age);

}

**public** Person(String n) {

System.***out***.println(name+" "+age);

**this**(n, 16);//报错:构造函数的调用必须是构造函数中的第一条语句

}

**public** Person(String n, **int** a) {

name = n;

age = a;

}

}

18、19题:

示例一:

**public** **class** Test1 {

**public** **static** **void** main(String[] args){

Son1 s = **new** Son1();//输出: 子类的无参构造方法

Son1 s1 = **new** Son1(5); //输出: 子类中有参构造方法5

}

}

**class** Father1{

}

**class** Son1 **extends** Father1{

**public** Son1() {

System.***out***.println("子类的无参构造方法");

}

**public** Son1(**int** i) {

System.***out***.println("子类中有参构造方法"+i);

}

}

[解释] 当父类没有显式定义构造方法时，编辑器会默认为父类添加一个隐式无参构造函数。此时子类可以有自己的无参和有参构造方法。不论实例化时调用的是子类的有参还是无参构造函数，先隐式调用父类的无参构造函数，再执行子类的构造函数。

示例二:

**public** **class** Test1 {

**public** **static** **void** main(String[] args){

Son1 s = **new** Son1();

Son1 s1 = **new** Son1(5);

}

}

**class** Father1{

**public** Father1() {

System.***out***.println("父类的无参构造方法");

}

}

**class** Son1 **extends** Father1{

**public** Son1() {

System.***out***.println("子类的无参构造方法");

}

**public** Son1(**int** i) {

System.***out***.println("子类中有参构造方法"+i);

}

}

[输出]

父类的无参构造方法

子类的无参构造方法

父类的无参构造方法

子类中有参构造方法5

[解释]

当父类有显式定义的无参构造方法时，此时子类也可以有自己的无参和有参构造方法。类似于示例一，先隐式调用父类的无参构造函数，再执行子类的构造函数。

示例三:

**public** **class** Test1 {

**public** **static** **void** main(String[] args){

Son1 s = **new** Son1();

Son1 s1 = **new** Son1(5);

}

}

**class** Father1{

**public** Father1(**int** i) {

System.***out***.println("父类中的有参构造方法"+i);

}

}

**class** Son1 **extends** Father1{

**public** Son1() {

**super**(1);

System.***out***.println("子类的无参构造方法");

}

**public** Son1(**int** i) {

**super**(i);

System.***out***.println("子类中有参构造方法"+i);

}

}

[输出]

父类中的有参构造方法1

子类的无参构造方法

父类中的有参构造方法5

子类中有参构造方法5

示例四:

**public** **class** Test1 {

**public** **static** **void** main(String[] args){

Son1 s = **new** Son1();

Son1 s1 = **new** Son1(5);

}

}

**class** Father1{

**public** Father1(**int** i) {

System.***out***.println("父类中的有参构造方法"+i);

}

}

**class** Son1 **extends** Father1{

**public** Son1() {//报错

System.***out***.println("子类的无参构造方法");

}

**public** Son1(**int** i) {//报错

System.***out***.println("子类中有参构造方法"+i);

}

}

[报错信息]隐式调用的构造函数Father1()未定义

[解释]综合以上4个例子看，首先需要说明的是，如果子类的构造函数没有使用super显式调用父类的构造方法，那么子类的构造方法会隐式调用父类的无参构造方法，即子类会在执行时自动在构造方法的第一行补充super()语句。如果子类的构造函数使用super显式调用父类的构造方法，那么子类的构造方法不再隐式调用父类的无参构造方法。

此外，结合(4)组选择6题可知：如果用户自己没有定义构造函数，则编译器自动生成一个默认构造函数(无参构造函数)；若用户自定义了构造函数，那么编译器不会生成默认构造函数(无参构造函数)，也就是说在这种情况下可用的构造函数只有用户自定义的构造函数。

结合这两点，就可以知道上面四个例子的结果的产生逻辑了。

示例5：

**public** **class** Test1 {

**public** **static** **void** main(String[] args){

Son1 s = **new** Son1();

Son1 s1 = **new** Son1(5);

}

}

**class** Father1{

**public** Father1() {

System.***out***.println("父类的无参构造方法");

}

**public** Father1(**int** i) {

System.***out***.println("父类中的有参构造方法"+i);

}

}

**class** Son1 **extends** Father1{

**public** Son1() {

System.***out***.println("子类的无参构造方法");

}

**public** Son1(**int** i) {

**super**(i);

System.***out***.println("子类中有参构造方法"+i);

}

}

[输出]

父类的无参构造方法

子类的无参构造方法

父类中的有参构造方法5

子类中有参构造方法5

输出的产生逻辑参考示例4中的解释即可。

示例6：

**public** **class** Test1 {

**public** **static** **void** main(String[] args){

Son1 s = **new** Son1();

}

}

**class** Father1{

**public** Father1() {

System.***out***.println("父类的无参构造方法");

}

**public** Father1(**int** i) {

System.***out***.println("父类中的有参构造方法"+i);

}

}

**class** Son1 **extends** Father1{

}

[输出] 父类的无参构造方法

[解释]该示例中，子类自己没有构造方法，则编译器为子类生成一个隐式无参构造函数，该隐式无参构造函数会隐式调用父类的无参构造函数，因此产生了如上的输出，由该例也可以看出，18题中所谓的“如果子类自己没有(显示定义)构造方法，那么父类也一定没有带参构造方法”是错误的，只需为父类显示定义无参和有参构造方法即可。当然，若父类只有有参构造方法是会报错的，例子见示例7，原因见示例4中的第二点说明。18题中的“子类将继承父类的无参构造方法作为自己的构造方法”更是无稽之谈，Java中构造方法是不可以继承的，发生的只是隐式调用而已。

示例7：

**public** **class** Test1 {

**public** **static** **void** main(String[] args){

Son1 s = **new** Son1();

}

}

**class** Father1{

**public** Father1(**int** i) {

System.***out***.println("父类中的有参构造方法"+i);

}

}

**class** Son1 **extends** Father1{//报错

}

[报错信息] 隐式调用的构造函数Father1()未定义。

三、判断题

1.构造函数的另一点说明：构造函数不用定义返回值类型（不同于void类型返回值，void是没有返回值；构造函数是连类型都没有），若定义了返回值类型，则该函数不是构造函数。例子：

**public** **class** Test1 {

**public** **static** **void** main(String[] args){

Son1 s = **new** Son1();

}

}

**class** Father1{

**public** **void** Father1() {//这不是构造函数

System.***out***.println("父类中的有参构造方法");

}

}

**class** Son1 **extends** Father1{

}

以上代码段没有输出，所有构造函数都将由编译器自动生成。

2. 子类继承了其父类中不是私有的成员变量和成员方法，作为自己的成员变量和方法。因此是不一定的。

6.参考5组选择6、7、8题。

7.示例一:使用非抽象类实现接口

**public** **class** Dog **implements** Animal {

**public** **void** eat(){

System.***out***.println("Dog eats");

}

**public** **void** travel(){

System.***out***.println("Dog travels");

}

**public** **static** **void** main(String args[]){

Dog d = **new** Dog();

d.eat();

d.travel();

}

}

[输出]

Dog eats

Dog travels

[说明]

使用非抽象类实现接口时，必须实现接口中的所有方法，如果只实现接口中的部分方法，比如只实现上例中的eat()方法，则会报错

示例2:使用抽象类实现接口中的部分方法

**public** **abstract** **class** Dog **implements** Animal {

**public** **void** travel(){

System.***out***.println("Dog travels");

}

**public** **static** **void** main(String args[]){

//注意这段注释如果放开会报错，这是因为抽象类不能被实例化，而Dog是一个抽象类

/\*

Dog d = new Dog();

d.eat();

d.travel();

\*/

}

}

[说明]可见，使用抽象类可以只实现接口中的部分方法，这与示例1形成了对比。

示例3: 使用抽象类实现接口中的所有方法

**public** **abstract** **class** Dog **implements** Animal {

**public** **void** eat(){

System.***out***.println("Dog eats");

}

**public** **void** travel(){

System.***out***.println("Dog travels");

}

**public** **static** **void** main(String args[]){

}

}

[说明]该抽象类实现了接口中的所有方法，注意，再次复习，抽象类中可以没有抽象方法，有抽象方法的类一定是抽象类。此外，类中的抽象方法必须加abstract修饰符，否则会报错，提示你添加函数体或者添加abstract修饰符；与此相对的，接口中的方法默认就是抽象方法，不添加abstract修饰符就默认它是抽象方法了，当然添加abstract也没问题，接口中的方法若添加了default关键字，则方法必须有函数体，否则会报错。

示例4:抽象类+继承实现接口中的所有方法

**public** **abstract** **class** Dog **implements** Animal {

**public** **void** travel(){

System.***out***.println("Dog travels");

}

**public** **static** **void** main(String args[]){

Dog1 d = **new** Dog1();

d.eat();

d.travel();

}

}

**class** Dog1 **extends** Dog **implements** Animal{

**public** **void** eat() {

System.***out***.println("Dog eats");

}

}

[输出]

Dog eats

Dog travels

[说明]由于Dog1是非抽象类，因此该类必须实现接口中的所有方法，此处Dog1继承了Dog中已实现的travel方法并自己实现了eat方法。

另外注意Dog1 d = new Dog1();可以改成Dog d = new Dog1();输出是完全相同的。

另外注意复习(5)组选择4中所提到的第二个bark示例，我们在此也给出这样的示例再次说明这一知识点。

示例5:

**public** **abstract** **class** Dog **implements** Animal {

**public** **void** travel(){

System.***out***.println("Dog travels");

}

**public** **static** **void** main(String args[]){

Dog d = **new** Dog1();

d.eat();

d.travel();

d.hi();//报错

}

}

**class** Dog1 **extends** Dog **implements** Animal{

**public** **void** hi(){

System.***out***.println("Dog hi");

}

**public** **void** eat() {

System.***out***.println("Dog eats");

}

}

[说明]编译时报错的原因是，Dog没有hi方法。若将程序改成:

示例6:

**public** **abstract** **class** Dog **implements** Animal {

**public** **void** travel(){

System.***out***.println("Dog travels");

}

**public** **static** **void** main(String args[]){

Dog d = **new** Dog1();

d.eat();

d.travel();

}

}

**class** Dog1 **extends** Dog **implements** Animal{

**public** **void** eat() {

System.***out***.println("Dog eats");

}

}

[输出]

Dog eats

Dog travels

[说明]首先Dog有eat和travel方法，因此d.eat()和d.travel()通过了编译时检查，在运行时按Dog1对象调用eat和travel方法。

示例7:

**public** **abstract** **class** Dog **implements** Animal {

**public** **void** travel(){

System.***out***.println("Dog travels");

}

**public** **static** **void** main(String args[]){

Dog d = **new** Dog1();

d.eat();

d.travel();

}

}

**class** Dog1 **extends** Dog{//报错

}

[说明]提示报错信息:Dog1应实现继承来的抽象方法Aniaml.eat() 实际上，implements是另一种形式的继承，在本例中Dog继承了Animal的抽象方法eat,Dog重写了Aniaml中的方法travel，完成了travel方法的实现。结合规则:非抽象类必须实现所有抽象方法(非抽象类不能有抽象方法)，可知Dog1必须实现eat方法。

8.不理解这句话要表达什么，但我认为它是错的。

示例:

**public** **class** Test1 {

**int** i;

**static** **int** *k*;

Test1(**int** i){

**this**.i = i;

*k*=*k*+1; //写成this.k=this.k+1也行，输出完全相同

}

**int** j=i+1;

**public** **static** **void** main(String[] args){

Test1 t = **new** Test1(10);

/\*

\* 写System.out.println(i);会报错不能使用

\* 静态方式引用非静态的i

\* 写System.out.println(k);是完全没问题的，

\* 输出也相同

\*/

System.***out***.println(t.i);

System.***out***.println(t.j);

System.***out***.println(t.*k*);

Test1 t1 = **new** Test1(10);

System.***out***.println(t1.*k*);

}

}

[输出]

10

1

1

2

[解释]i是实例成员变量,k是类变量(静态变量),第1、3、4个输出的结果是显而易见的。现重点解释第二个输出: 在新建对象时，先执行成员变量初始化再调用构造函数,i的默认初值为0，因此j初始化为1。

实际上使用构造方法可以给定义在该构造方法中的局部变量赋初值，也可以给类变量赋初值，不知道这道题为什么这么说还算对。

这里赋初值我认为可以理解为:在一个变量的生命周期内，第一次显式地将一个右值赋给某变量。