**Java 面向对象**

1. **Java封装**

类：属性（静态属性）、方法（动态属性）

一个java文件可以有多个class，但是名称必须与唯一的public main所在的class名称相同,或者是public class

Test.java: class Test(public static void main)

Class Test2(static void main)

1. **Java继承**

初始化顺序：1.如果当前类没有加载，要加载类，每个类只加载一次。

2.类的加载过程中，static成员变量和static语句块会被初始化——Demo1

3.生成对象过程中，先初始化对象的成员变量，再执行构造器（变量在定义与方法之间）

**Demo1:**

public class Test {

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

        Bread bread1 = new Bread();

        Bread bread2 = new Bread();

    }

}

class Bread {

    static{

        System.out.println("Bread is loaded");

    }

    public Bread() {

        System.out.println("bread");

    }

}

结果：Bread is loaded.只输出一次。

**Demo2:**

public class Test {

    public static void main(String[] args)  {

        new Meal();

    }

}

class Meal {

    public Meal() {

        System.out.println("meal");

    }

    Bread bread = new Bread();

}

class Bread {

    public Bread() {

        System.out.println("bread");

    }

}

结果：bread

meal

继承：如果没有明确的指出继承的类，隐式的继承根类Object。

继承原则（变量）：1.可以继承public和protect成员变量，private不可以。

2.对于父类的包访问权限成员变量，子类与父类只有在同一个包下可以访问。

3.对于子类可继承成员变量，若子类出现同名成员变量，则会发生隐藏，即屏蔽掉 父类成员变量，可以使用super来进行引用。（即父类与子类可分别计算存在）

继承原则（方法）：1.可以继承public和protect成员方法，private不可以。

2.对于父类的包访问权限成员变方法，子类与父类只有在同一个包下可以访问。

3.对于子类可继承成员方法,若子类出现同名成员方法,则会发生覆盖,即覆盖掉 同名成员方法,可以用super引用。

隐藏：成员变量、静态成员。 覆盖：普通函数。

构造器：若父类是带参构造器，子类要用super调用父类构造器，并且在第一行。若父类有无参构

构造器，则可以不显示调用。

**Demo3:**

public class Test {

public static void main(String[] args) {

new Circle();

}

}

class Draw {

public Draw(String type) {

System.out.println(type+" draw constructor");

}

}

class Shape {

private Draw draw = new Draw("shape");

public Shape(){

System.out.println("shape constructor");

}

}

class Circle extends Shape {

private Draw draw = new Draw("circle");

public Circle() {

System.out.println("circle constructor");

}

}

结果：shape draw constructor

shape constructor

circle draw constructor

circle constructor

**Demo4：**

public class Test {

public static void main(String[] args) {

Shape shape = new Circle();

System.out.println(shape.name);

shape.printType();

shape.printName();

}

}

class Shape {

public String name = "shape";

public Shape(){

System.out.println("shape constructor");

}

public void printType() {

System.out.println("this is shape");

}

public static void printName() {

System.out.println("shape");

}

}

class Circle extends Shape {

public String name = "circle";

public Circle() {

System.out.println("circle constructor");

}

public void printType() {

System.out.println("this is circle");

}

public static void printName() {

System.out.println("circle");

}

}

结果：shape constructor

circle constructor

shape

this is circle

shape

1. **抽象类与接口**

抽象方法：abstract void fun()没有具体的实现方法。

抽象类：[public] abstract class ClassName{}含有抽象方法的类。没有抽象方法有abstrac也是

抽象类。抽象方法只能为public(缺省),protected。抽象类不能用来创建对象，即只用来继

承。若子类未实行抽象方法，则必须也定义为抽象方法。

接口：[public] interface InterfaceName{} 接口中可以有变量(隐式的指定为public static

final) 、方法(隐式的指定为public abstract)。

对比：1.抽象类可以提供成员方法的实现细节，而接口中只能存在public abstract 方法；

2.抽象类中的成员变量可以是各种类型的，而接口中的成员变量只能是public static final类型的；

3.接口中不能含有静态代码块以及静态方法，而抽象类可以有静态代码块和静态方法；

4.一个类只能继承一个抽象类，而一个类却可以实现多个接口。

1. **Java单例模式**

单例模式：保证一个类仅有唯一的实例，并提供一个全局访问点。

单例类：只有一个实例；自己创建自己的实例；为其他对象提供唯一的实例。

懒汉模式：只有在用的时候才生成实例。类加载快，运行时慢。轻量级对象使用。

饥汉模式：在类加载的时候生成实例。类加载满，运行时快。重量级对象使用。

好处：1.控制资源的使用，通过线程同步来控制资源的并发访问。

2.控制实例的数量，节约资源。

3.数据共享，在不建立直接关联的条件下，让不相关的线程或进程实现通信。

使用：打印机、数据库连接池。

Java Synchronized：避免同一个数据对象同时被多个线程访问。可以保证线程可见性(一个线程的共享数据的变化可以被其他线程看到)

* 可以修饰函数，也可以作为函数语句（同步方法、同步语句块）。
* 无论修饰的是对象还是方法，取得的锁都是对象，而不是一段代码，而且同步方法很可能还会被其他线程的对象访问(静态可访问)。
* 每个对象只有一个Lock与之关联。
* 很容易造成死锁，尽量避免应用。

作用域：1. synchronized aMethod(){}同一个对象只要有一个同步方法被一个线程访问，其他

同步方法不能被其他线程访问。但不同对象之间互不干扰。

2.synchronized static aMethod{}对类的所有实例起作用。

对象锁和类锁并不互斥,可能产生线程安全问题

Static void myMethod(){

Synchronized(MyClass.class){ //code }

} === static synchronized void myMethod(){ //code }

Void myMethod(){

Synchronized(this){ //code }

} === synchronized void myMethod(){ //code }

Synchronized方法：每个实例对应一把锁，每个同步方法获得类实例的锁才能执行，否则线程阻塞直至锁释放。确保每一个类实例，其所有同步方法至多有一个处于可执行状态。（若可能访问成员变量的成员函数军备声明为synchronized,则避免成员变量的访问冲突。对于静态变量和静态成员函数也可以应用，因为每个类页拥有一个锁）。

缺陷：基于多线程学习

Synchronized 代码块：作用于方法的某个区块中，减小锁的粒度，用法是synchronized(this){/区块/}，作用域为当前对象。若没有明确定的对象作为锁，可以创建一个instance变量来充当锁。例如：

class Foo implements Runnable {

private byte[] lock = new byte[0]; // 特殊的instance变量

Public void methodA() {

synchronized(lock) { //生成0长度的byte[]需要3条操作码，Object需要7条 }

}}

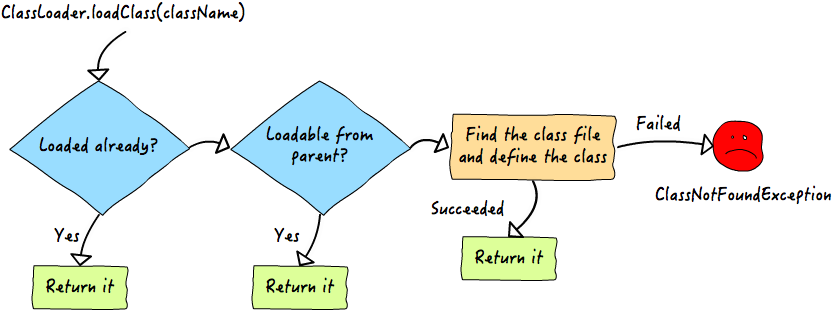
Synchronized monitor:monitor由ObjectMonitor实现（含有\_WaitSet和\_EntryList两个队 列），monitor对象存在于每个java对象的对象头（对象在内存中分为三块：对象头、实力数据、对齐填充），所以任意对象都可以作为锁（即加入一个代码块对数组加锁，尽管该数组跟代码块没关系，但其他线程运行到这里时发现数组不可访问，则会等待数组被释放再执行加锁的代码块。） Monitorenter和monitorexit指令指明同步代码块开始和结束的位置，永远配对，即使线程异常结束也会有异常处理器执行monitorexit。

Class Loader: jvm中的类装载器，把类文件从硬盘读取到内存中，jvm在加载类的时候通过ClassLoader（抽象类）的loadClass()方法来加载class。loadClass使用双亲委派模式。

三个类加载器：Bootstrap : core java libraries(java.\*, javax.\*)

Extension : load classes from “java.ext.dirs”

System : load classes from “java.class.path”



Object reference：可以为null，看作一个地址。

* 强引用StrongReference：Object obj = new Object(); obj=null;释放
* 软引用SoftReference：

SoftReference<Object> sr = new SoftReference<Object>(obj);当只有软引用指向obj且内存紧张时释放，可用于数据缓存。

* 弱引用WeakReference:

WeakReference<Object> wr = new WeakReference<Object>(obj);当只有弱引用指向它时释放。

* 虚引用PhantomReference:

ReferenceQueue<Object> rq = new ReferenceQueue<Object>();

PhantomReference<Object> pr = new PhantomReference<Object>(obj, rq);

当只有虚引用指向它时释放。

Garbage Collections:

1. **Java多态**