Day1

方法 原理 实战 项目

数据库->持久层->业务层->视图层

静态变量（类变量）

```
block块
每次调用就执行一次
{
    构造方法
}
静态初始化块
static{

}只调用一次 在实例初始化前 属于类方法
```

override重写

重写父类方法 覆盖

overload重载

构造方法 有多个相同名字函数参数不同来区分

POP&OOP

POP：面向过程编程 procedure oriented programming

面向过程就是分析出解决问题所需要的步骤，然后用函数把这些步骤一步一步实现，使用的时候一个一个依次调用就可以了

OOP：面向对象编程 object oriented programming

面向对象就是将我们的程序模块化，对象化，把具体事物的特性属性和通过这些属性来实现一些动作的具体方法放到一个类里面

AOP：面向切面编程 日志 事务管理 非原子操作 Aspect Oriented Programming

通过预编译方式和运行期动态代理实现程序功能的统一维护的一种技术

抽象类&接口类

比较抽象的东西放在抽象类中 水果

```
所有类首字母都大写 默认空为NULL
java中的type有两种：引用类型 （有地址指向对象）基本类型(原始类型 8种)
引用类型首字母大写 String
原始类型 int 未初始化为0
bool 默认为false
```

静态变量 静态类型 类类型 static

局部变量 方法变量 必须在使用前初始化

Java中上一级类只能有一个

extend:

抽象类可以有实现的方法

```
abstract void drawcircle();
```

带括号的即为实现方法

```
abstract void drawcircle;
```

不带括号只声明 未实现

接口：百分百未实现

抽象类：至少有一个未实现

接口变量

```
final static ...
```

面向对象三个特性

继承

继承父类的属性和方法

虚调用 virtual invoke 设计模式

```
Fruit a=new Apple();
```

封装

```
public ...
private ...
provide ...
```

extend：

内部类

```
public class LinkedList<T> {
    class Node {
        T data;
        Node prev;
        Node next;
    }

    Node fst;
    Node lst;
}
//高内聚
```

```
$ 文件名中带有dollar符号
```

外部类

先定义外部类才能定义内部类

多态

通过重载实现多态

抽象类的方法一定是public方法

抽象类可以有实现的方法

String 是引用类型

String与StringBuffer

```
String x1="aaa";
String x2="aaa";
String x3=new String("aaa");
//x1=x2/x1=x3.intern()
```

String 在传参数过程中

例如

```
public void String(s1){
    s1+="bbb";
}
```

会创造新的指向在main中定义的s1的实体并进行更改

StringBuffer在传参数过程中

例如

```
public void StringBuffer(s1){
    s1.append("bbb");
}
```

不会创造新的指向在main中定义的s1的实体 在原本的实体的s1上更改进行更改

VO、DTO

VO 视图对象

DTO 数据传输对象

类是对象的模板 对象是类的实例

消息：对象方法名参数

Others

Putty

Hadoop生态 环境 搭建

JDK

开发工具

SVN

MVVM

Vue.js React Angular

MVC

EJB

POJO不需要实现特定借口的JAVA对象

Shiro

JPBM/Activiti

Spring SpringMVC

上溯与下溯造型

上溯相当于是常见的类型转换 小转大 默认可以

扩大类别 安全 自动转换 子类转父类

比如 苹果->水果

下溯是大转小 会提示异常 最好进行类型判断转换

缩小类别 不安全 强制转换

比如 水果->苹果

拆箱与装箱

```
Interger int 类型类
```

```
int x=10;
Interger y=x;
float z=10;
float z=10.0f;
```

~~float z=10.0;~~

设计模式

单例模式

通过对构造函数重写来使得每个类只有一个实体

线程不安全

```
public class SingletonDemo1 {
    private static SingletonDemo1 instance;
    private SingletonDemo1(){}
    public static SingletonDemo1 getInstance(){
        if (instance == null) {
            instance = new SingletonDemo1();
        }
        return instance;
    }
}
```

线程安全

```
public class SingletonDemo2 {
    private static SingletonDemo2 instance;
    private SingletonDemo2(){}
    public static synchronized SingletonDemo2 getInstance(){
        if (instance == null) {
            instance = new SingletonDemo2();
        }
        return instance;
    }
}
```

静态内部类

```
public class SingletonDemo5 {
    private static class SingletonHolder{
        private static final SingletonDemo5 instance = new SingletonDemo5();
    }
    private SingletonDemo5(){}
    public static final SingletonDemo5 getInsatance(){
        return SingletonHolder.instance;
    }
}
```

Exception

异常通过

```
try{
...
}catch(){
...
    throw()

}finally{
...
}
来进行异常操作与处理
定义时注意throws ...
与打印信息的显示
```

线程程序执行优先权

main的优先权默认为5

```
public class Main {
    public void setPrioritiesOnThreads() {
        Thread thread1 = new Thread(new TestThread(1));
        Thread thread2 = new Thread(new TestThread(2));
        thread1.start();
        thread2.start();
        try {

            //Wait for the threads to finish
            thread1.join();
            thread2.join();
        } catch (InterruptedException ex) {
            ex.printStackTrace();
        }
        System.out.println("Done.");
    }

    public static void main(String[] args) {
        new Main().setPrioritiesOnThreads();
    }
    class TestThread implements Runnable {
        int id;
        public TestThread(int id) {
            this.id = id;
        }
        public void run() {
            for (int i = 1; i <= 10; i++) {
                System.out.println("Thread" + id + ": " + i);
            }
        }

    }
```

```
}
```

join放置某处提前执行 插队操作

setPriorities设置优先权

this&super

super 父对象，super第一种用法是调用父类的构造方法构建父类对象，第二种用法是调用被子类重写的父类对象成员

```
class Animal {
    public Animal() {
        System.out.println("An Animal");
    }
}
class Dog extends Animal {
    public Dog() {
        super();
        System.out.println("A Dog");
        //如果子类构造函数中没有写super()函数，编译器会自动帮我们添加一个无参数的super()
    }
}
class Test{
    public static void main(String [] args){
        Dog dog = new Dog();
    }
}
```

this 当前对象，当局部变量与成员变量重名的时候，访问成员变量要用this

```
class Mini extends Car {
    Color color;
    public Mini() {
        this(color.Red);
    }
    public Mini(Color c){
        super("mini");
        color = c;
    }
    public Mini(int size) {
        super(size);
        this(color.Red);
}
```

典型课后题

给你一个链表，用递归的方法找出链表中最大的节点?

```
class Node {
    int data;

    Node next;
```

```
    public Node(int data) {
        this.data = data;
    }
}
public class Link {
    public static Node max(Node node) {
        if(node.next == null) return node;
        if (node.data>node.next.data) {
            node.next = node.next.next;
            return max(node);
        }
        else {
            return max(node.next);
        }
    }
    public static void main(String[] args) throws Exception{
        Node n1 = new Node(4);
        Node n2 = new Node(5);
        Node n3 = new Node(2);
        Node n4 = new Node(1);
        n1.next = n2;
        n2.next = n3;
        n3.next = n4;
        Node n = max(n1);
        System.out.println(n.data);
    }
}
```

大学生创新创业环境分析及政策解读

TMD 头条 美团 DiDi

IOT 物联网 Internet of things

IP 知识产权 Intellectual property

GEM 全球创业观察 Global entrepreneurships monitor

TEA 中国早期创业活动指数

客户服务产业 客户服务业（加附加值）

在教育培训、商务环境和研发转移方面改善缓慢或停滞不前

创业心态+创业资源+创业能力

SWOT：Strength weakness opportunity threat