1. Object类，方法:
2. Public Object() 构造
3. Public equals() 对比：
   1. 对比== 和 equals 对比两个相同基础变量==返回true，不同类型的基础变量（int 和 short对比依旧为true：short提升为short）（int 和 double对比依旧为true：int提升为double）(String a=”abc”,String b=new String(“abc”) 对比为false，应为a存在常量池中，b存在于一块新开辟的对象空间)(new的对象内容相同，依旧为false，因为地址值不同，是不同的对象)得出结果：==类型可以用于比较基本类型(笔记数据的值)，和引用数据类型（比较数据的地址(string看创建方式)）
   2. Equals调用方式是通过对象名.equals的方式调用，比较的是对象a和对象b的值；String object的equals类方法：1.比较二者是否地址相同2.比较二者是否都是String类或其子类，强转(string)传入进来的对象3.比较二者长度是否形同4.一次比较每一个字符是否相同
   3. 继承关系下子类对父类的方法进行覆盖：
      1. 规则：返回类型不大于父类的返回类型：父类的返回类型是void，子类也必须是void；父类返回值是基本数据类型，子类的重写方法也必须是基本类型，父类的返回值是引用类型子类的返回类型不得大于父类的返回类型。。。还有
      2. 子类无法重写父类的private方法
      3. 子类方法抛出的异常类型不得大于父类的异常类型
4. 关键字
   1. Static关键字
      1. 作用：描述的属性就是属于这一类的内容(属性)，不会改变的(不会因为对象的改变而改变)；这一类公用的方法
      2. 利用static修饰过后内存空间发生改变
      3. 利用static声明属性，方法，代码块，内部类，被修饰过后有以下特点
         1. 随着类的加载而加载
         2. 优先于对象存在
         3. 修饰的成员，被所有对象所共享
         4. 访问权限允许时可不创建对象，直接被类调用
      4. 静态的方法中不可以访问非静态的方法，反之可以
5. 单例模式
   1. 单例类只有一个实例
   2. 单例类必须自己创建自己的唯一实例
   3. 单例类必须给其他类提供这一实例
   4. 实现步骤：
      1. 构造器私有化
      2. 提供静态方法返回实例对象
6. 代码块
   1. 对java类或对象进行初始化
      1. 不显示进行初始化，java类的属性有默认值
      2. 显示初始化//和代码块初始化冲突
      3. 构造器初始化
      4. 调用方法初始化
      5. 通过代码块对对象进行初始化//和显示初始化冲突

Final不能被继承

抽象类可以定义构造器

抽象类规范子类的行为

1. Exception
   1. 三元运算符：自动转化类型true?Integeer(1)结果输出float:1.0
   2. 当给Integer用=赋值时若超过127则转成Integer对象
   3. 因为static创建过早无法调用非静态的xxx
   4. 单例创建：私有化构造器：用静态方法返回对象
   5. 抽象类和接口的区别：
      1. 抽象类由常量，变量，方法都有，接口。。。
      2. 抽象类继承，接口实现
2. 接口
   1. 所有成员public，变量static final ，方法abstract，无构造器
   2. 代理模式，一个类调用另一个类的实现方法
   3. 若方法名相同，则会只实现一个，返回类型必须相同，否则报错

实现类内部可以使用 实习那列的实例对象可以通过对象名，默认方法使用

* 1. 非静态内部类实例化需要先创建父类的对象，确保子类可以被创造，或者使用static修饰符让内部类可以直接构造，但是其中不可以使用非静态变量
  2. 匿名类xx，要实例化该类则：new 父类构造器(实参列表)|  
     实现接口(){

//匿名内部类的类体部分

}——————经典案例：监听器的使用setOnClickListener(new OnClickListener(){

@Override

Public void onClick(View v){

//do something

}})

1. 异常
2. 泛型
   1. 定义方法：public static <E> List<E> functionName(E[] arr){}//
      1. 对传入的数据进行处理，返回对应的数据类型
   2. 定义泛型类class costumer<T>{}在类里面通过T来定义泛型方法
   3. 用另一个类继承Another extends costumer<Adult>;此时该类就具体了，泛型方法也具体了；方法只能传入Adult类的对象了
   4. List<String> List<Object>之间属于平行关系，不能进行赋值操作，相同类型可以
   5. 泛型不同的类可以通过通配符?来确定唯一父类-🡪List<?>list 此时可以将Sting 和 Object的泛型类列表直接赋值给通配符类
   6. 通配符类不能添加任何除了Null以外的数据，取出来必须转成Object
   7. 通配符加以限制：<? Extends Person> 添加的必须小于等于Person

<? Super Person>添加的必须大于等于Person

1. IO流
   1. File.list()File.listFiles()：：目录获取子文件/夹相对/绝对路径，返回一个String数组/File数组
   2. File.isFile/isDictionary判断是否是文件/目录//////绝对路径
   3. File.makdir()返回boolean；；；java删除目录得目录为空得递归删除所有子文件及目录
   4. 涉及文件内容操作使用IO流：
      1. 流有很多：四个基本的🡪inputSteam,outputStream,Reader,Writter
      2. 用接水管的方式读取水流🡪FileReader(new File())
         1. 读取:按照字符循环读取，当读取到末尾时，返回-1，可写为：while((data = rd.read())!=-1){print((char)data)} if rd!=null rd.close：手动关闭阀门
         2. 一次一个字符太少了：char buffer = new char[5];while(len = fr.read(buffer)!=-1)for int i=0;i<len;i++{print(buffer[i])}
         3. New FileWriter(new File):写的操作无需判断文件存在与否，若没有则新建一个文件，不带参数则直接覆盖—带参数：append:ture--添加不覆盖，将读取的数组写入时可带参数：fr.write(buf,0,len)
         4. 字节流：FileInputStream字节读取文本文件时不要打印，中文装不下，可能会断掉出现乱码；读取时使用byte数组存
         5. 缓冲流：BufferedInputStream(fr)传入字节流，因为提供了缓冲区的原因要比字节流快好几倍
         6. BufferdRead/Writer 定义一个String data reader.readLine进行行读取，写入data,换行可以使用writer.nextLine();,在初始化时，跟参数可以将读取或写入方法改变🡪new InputStreamReader(rd,”utf-8”)
         7. 流进行加密解密🡪最简单：对每一个写入字节进行异或处理，解密：对加密文件进行异或处理
      3. 网络编程
         1. 基础传输字符串：
            1. 客户端：

定义socket,参数：对方端口号

定义输出流 = socket.getOutputStream()

输出流输出字符串的字节数组::os.write(“str”.getBytes)

* + - * 1. 服务端：

定义ServerSocket,参数(己方端口号) ，在定义socket = ss.accept(),获取输入流：socket.getInputStream();

如果想打印：可以将每轮的字节流存入ByteArrayOutputStream,最后转成字符传

一个完整的交互过程：

客户端：定义socket，定义文件输入流，定义socket输出流，文件输入流循环读取文件，socket输出流循环发送buffer，结束后socket. shutdownOutput;

服务端：定义serverSocket，定义socket=服务socket.accept()，定义文件输出流，定义socket输入流，循环读取socket输入流，文件输出流循环在本地写入，完成后，定义socket输出流，输出已接收

客户端：定义socket接受流，定义字节数组流，socket接受流循环读取，字节数组流循环写入，最后打印

1. 反射
   1. ***基本创建(弃用)：***
      1. Class clazz = Person.class;//获取类结构
      2. Constructor cons = clazz.getConstuctor(class.String,class.int);构件复制模板
      3. Object obj = cons.getInstance(“tom”,15);创建对象
      4. Person person = (Person) obj;
   2. ***动态创建(推荐)：***
      1. Class clazz = Class.forName(“com.dell.SepDay9.Person”);//路径要写全,直接得到该对象
      2. Object person = clazz.newInstance()/**/不传参一定要有一个无参构造方法！！！**
   3. ***属性/方法读取***，**(在setAccessible(true)之前模板不能修改private修饰的属性和方法:**)
      1. Field age = clazz.getDeclaredField(“age”)//获取申明属性模板
      2. age.set(person,10);//导入要修改的动态类，值
      3. Method method = clazz.getDeclaredMethod(“show“,String.class);//获取申明方法模板
      4. method.invoke(person,”arg”);//导入要修改额动态类
   4. ***是否破坏封装性？***：我理解的反射破坏封装性:私有化属性是为了不让不了解这个类的人不要乱改里面的属性\n导致功能紊乱\n但是当你使用了反射的穿透机制,说明此时的封装有问题，必须要修改里面的私有化属性了,必要时不得不用
   5. ***使用场景***：
      1. 前端登录或注册提供一个参数regest || login 通过这两个参数名来构造对应的对象，同时调用里面的方法
   6. ***读取配置文件***：
      1. Properties pros = new Properties();;;;;;;FileInputStream fis = new FileInputSream(“src\\jdbc1.roperties”);;;;;; pros,load(fis);;;;;;;String s = pros.getProperties(“userName”);
   7. ***方法中的信息***(先获取所有方法clazz.getMethods)：
      1. Annotation[] annotations = method.getAnnotations();//该方法可能的注解
      2. //获取权限修饰符,将对应的数字转化成字符串 System.out.println(Modifier.toString(method.getModifiers()) + "\t");
      3. //返回值类型 System.out.println(method.getReturnType().getName()+"\t");
      4. System.out.println(method.getName()+"\t");//方法名
   8. ***获取所有构造器*** M.getConstructors()
   9. ***获取父类***：
      1. 普通父类Class superClass = clazz.getSurperclass();
      2. 带泛型的父类：getGenericSuperClass()
      3. 如何获取泛型：ParameterizedType paramType = (ParameterizedType) generiSuperclass;;;;;;;;;;;;;;;;;;;;;;Type[] actualTypeArguments = paramType.getActualTypeArguments();;;;;;;;;;;遍历即可
   10. 获取接口：getInterfaces()
2. 动态规划
3. Es6
   1. 在while循环中，var定义的全局变量，一直存在，在函数回调时，会导致结果出错，这里使用let局部变量解决
   2. 常量结构：定义一个const对象块const zhao{name=”zhao”,age=12.xiaoping:function(){console.log(“I can show);}

利用let赋初值，let{name,age,xiaoping} = zhao;

* 1. 反引号``直接得到一整段的字符串,将需要动态添加的元素放入
  2. =>函数的使用
     1. Let getName = (传参数)=>{console.log(this.name)}箭头函数的第一个传入参数不可变,argument(形参)未定义
     2. 定义一个数组进行过滤const arr=[1,3,4,5]; const result = arr.filter(item=>item%2===0)
  3. Rest参数：如果函数传入多个参数：使用rest参数,let fuc = (a,b,…args){console.log(args)}此时的args是一个数组，可以使用数组相关的api例如filter(item=>执行操作)

1. Vue