**一、类**

**1定义**

package xxx.yyy //指定所在包(命名空间)

public class OneClass extends SuperClass implements Interface1, Interface2...

{

public static void main(String[] args) {} //main方法

public OneClass() {} //构造器

public static void classMethod(int a, String b) {} //类方法

public String instanceMethod() {...} //实例方法

private String name = "youName"; //实例变量

private static int nextID; //类变量

public static final double PI = 3.141592653;

}

按访问级

·公开类：public声明，且与文件名相同，可在任何类环境访问

·同包类：缺省声明，可在同包类环境访问

继承性

·一般类：缺省声明，可独立使用，亦可被继承

·最终类：final声明，不可用于继承。见继承

·抽象类：abstract声明，仅用于继承。区别为，可有abstract方法，不可实例化

关键字规范：访问级(4种) + 继承性(abstract|final|无)

**2方法**

按访问级

·公开级：public声明，可在任何类访问

·保护级：protected声明，可在同包类或子类访问

·同包级：缺省声明，可在同包类访问

·私有级：private声明，仅在本类中访问

按用途

·类方法：static声明，可访问类变量、类方法

·实例方法：缺省声明，可访问实例/类变量、实例/类方法

·抽象方法：abstract声明，需子类定义的实例方法(与访问及private冲突)

重写性

·可重写：缺省声明

·非重写：final声明，方法禁止被子类重写

方法参数：

·形式参数：采用拷贝基础类型值，或拷贝引用类型指针实现

·可变参数：public void method(Object... args){}，args作为Object[]类型传入

·隐式参数：this指代实例域当前对象，super指代实例父对象(静态域无super隐参)

初始方法

·main方法：类方法，虚拟机初始化时最先执行

·构造器：以类同名无返回值的实例方法，实例化时执行

·super方法：调用父类重载构造器，仅在子类构造器第一行实现(否则报错)  
 ·类域块：static{...}定义，类被加载时执行

·实例块：{...}定义，实例化时执行

关键字规范：访问级(4种) + 用途(static|无|abstract) + 重写性(abstract|final|无) + 其他

**3变量(java中称域)**

按访问级：见方法

按用途

·类变量：static声明，类加载时被初始化(显式|null|0)，作用域为类域

·实例变量：实例化时被初始化(显式|null|0)，作用域为实例域

读写性

·只读变量：final声明，初始化后其值/引用不可改变

·读写变量：缺省声明，其值/引用可改变

关键字规范：访问级(4种) + 用途(static|无) + 读写性(final|无) + 其他

**4继承**

继承

·类的继承：所有非final类(异包被继类需public声明)

·方法继承：所有非private方法(异包被继方法需public/protected声明)

·变量继承：所有非private变量(异包被继变量需public/protected声明)

重写：

·方法：覆盖继承方法，final方法不可重写。

·构造器：同方法重写，super()方法调用父构造器(必须在重写第一行调用)

父类重载了构造器，子类必须显式super(...)调用父类重载的构造器

·变量：覆盖继承变量的访问级、显式初始化

实现：抽象类中的abstract方法，具体子类必须实现

**二、对象与类**

**1实例化与初始化**

实例化方法：

OneClass oc = new OneClass(); //可将对象赋值给抽象变量

类初始化：类加载时按如下顺序

·静态块：static{}定义

·类变量显式定义

·若未被初始化自动赋值：null、0、false

实例初始化

·递归实例初始化父类对象

·实例块：{}定义

·构造器：OneClass(){}，可指定不同参数重载构造器

·实例变量显式定义

·若未被初始化自动赋值：null、0、false

**2对象结构**

类结构：super指针(父类)、静态变量组、实例类型组、方法组(实例|静态)

对象结构：kind\_class指针(类)、super\_obj指针(父对象)、实例变量组



**3调用链**

构造器调用：被调用时，隐式递归调用默认父构造器，或显式将super(...)放于第一行

类方法调用：

·用类调用类方法：入口类为本类，若不存在则遍历父类方法

·对象调用类方法，入口类为变量的声明类(与对象无关)

实例方法调用：入口类为对象的类，若不存在则遍历父类实例方法

变量访问：调用继承方法访问的是父类或父对象中对应的变量，不存在则向上遍历

三、**接口(类似OC中的协议)**

**1定义**

定义：用于对类的多肽扩展

public interface Worker extends People

{

int abc = 123; //接口变量，public static final类型

static void aaa() {}; //接口静态方法

void bbb(); //接口抽象方法

defualt void bbb() {}; //接口扩展方法

private void ccc() {}; //接口私有方法

}

用途：对类的扩展、对类的规范以实现多态调用(强类型语言特性)

访问级

·公开接口：public声明，与文件名相同，可在任何类访问

·同包接口：缺省声明，可在同包类访问

抽象性：接口都为抽象性，即缺省等同abstract声明

**2方法与变量**

接口方法：除private都为公有方法，无需public声明

·静态方法：static声明，接口自身的方法(Java8)

·抽象方法：缺省声明，约定实现类的实例方法

·扩展方法：default声明，扩展实现类的实例方法(Java8)

·私有方法：private声明，仅被同接口default方法中调用(Java9)

接口变量：都为常量(public static final类型)，缺省声明即可，必须显式初始化

**3接口继承**

继承

·方法继承：可继承default、抽象方法

·变量继承：可继承所有接口变量

重写

·方法重写：即重写default、抽象方法

·变量重写：可重写所有接口常量

**4接口实现**

public class Employee implements Worker, Comparable

{ public void bbb() {...} //实现接口的方法 }

实现：类需定义接口的抽象方法

继承：继承接口default方法和常量，并置于继承链末端(实现类或其父类均可重写)

**5匿名类**

定义：通过类/接口的匿名子类创建一个对象，可重写方法

Thread t = new Thread() { //匿名类

public void run() { //code... }

}

Runnable r = new Runnable() { //匿名接口类

public void run() { //code... }

}

用途：实现对象独有方法(ruby中obj.extend)，如Thread的匿名

**四、泛型**

**1泛型类**

定义：使类具有类型通用性，从而避免乱用Object类型

public class Pair<T, U> //指定泛型类，多个泛型用","隔开

{

public Pair(T f, U s) { first = f; second = s; }

public T getFirst() { return first; }

public U GetSecond() {return second; }

private T first; private U second;

}

使用

Pair<String, int[]> p = new Pair("key", new int[]{1,2,3});

String s = p.getFirst(); //无强制类型转换

**2泛型方法**

定义：使方法具有类型通用性，可定义在非泛型类

public static <T> T getLast(T[] a) { return a[a.length - 1]; } //指定泛型方法

使用

String[] arr = {"John", "Q."};

String last = ArrayAlg.getLast(arr);

**3泛型限定**

定义：可限定泛型的继承连，属于某个接口(父类)

<T extends Comparable & Serializable, U> //多个限定用&连接

**4泛型作参数的通配**

泛型参数通配：作为参数时，使用<?>可接受任意泛型，否则必须与指定泛型一致

public void print(Collection<?> c){ System.out.println(s); }

泛型参数通配限定：<? extends XXX>

**五、注解**

**1注解原理**

类、方法、域可包含多个注解对象。

通过标注将注解绑定到类、方法、域上

利用反射实现相关项的配置

**2注解类型**

元注解：定义注解的注解

·@Target：指可注解的项，见下属

ElementType.Type 可用于类、接口、注解类型

ElementType.METHOD 可用于方法

ElementType.CONSTRUCTOR 可用于构造器

ElementType.FIELD 可用于域

ElementType.PARAMETER 方法或构造器参数

·@Retention：注解的生命周期

RetentionPolicy.SOURCE 类源文件(编译时被抛弃)

RetentionPolicy.CLASS 类文件(加载器加载时被抛弃)

RetentionPolicy.RUNTIME 虚拟机(一直存在)

·@Inherited：指定注解的继承性，缺省不继承

·@Document：提供Javadoc文档

编译注解：编译后抛弃

·@Deprecated：标注全部，指示为过时(编译时若引用过时项将发出警告)

·@SuppressWarnings("xxx")：标注非(包/注解)，指示屏蔽指定警告

·@Override：标注方法，指示为重写方法(若父类无此方法将抛出异常)

资源注解：系统提供或自定义的注解

·@PostConstruct：标注方法，在类构造后执行

**3注解定义**

注解定义：注解都隐式扩展了java.lang.annotation.Annotation接口

@Target({ElementType.Type, ElementType.METHOD})

@Retention(RetentionPolicy.RUNTIME)

public @interface AnnotationName1{

String abc() default "123"; //元素abc

int count() = 0; //元素count

}

public @interface AnnotationName2{ String value(); } //单值注解定义

元素类型：基本类型、String、Class、enum、注解、数组

注解标注

@AnnotationName1(abc="hello", count=123) //多值标注

java code ...

@AnnotationName2("nihao") //单值标注

java code ...

**4处理注解**

注解反射：见反射，@Retention为运行时的可被反射(其他的被虚拟机抛弃)

注解元素访问：同方法调用

AnnotationName1 a = xxx.getAnnotation(AnnotationName1.class)

a.abc()

**六、包**

作用：实现Java类的命名空间(OC用前缀，Ruby用module)

实现包类

·类文件加入package xxx.yyy语句(见类)

·类文件置于package文件夹xxx/yyy

·将包放入类加载路径(见JDK环境)

用它包类

·全包名调用：java.util.Date today = new java.util.Date();

·导包后调用

import java.util.\*;

Date today = new Date();