**【第六章 抽象类和接口】**

1、abstract修饰的类称为抽象类

抽象类的特点：

* 1. 不能实例化对象；
  2. 类中可以定义抽象方法（abstract修饰的方法）；
  3. 抽象类中可以没有抽象方法

2、abstract修饰的方法称为抽象方法，抽象方法只有方法的声明没有方法实现，即没有方法体。包含抽象方法的类本身必须被声明为抽象的。

3、子类继承抽象类必须实现抽象类中所有的抽象方法，否则子类也必须定义为抽象类。

4、Java中的接口是一系列的方法的声明，是一些方法特征的集合，接口可以看作是一种特殊的抽象类，其中包含常量和方法的声明，为没有变量和方法的实现。

接口的定义语法：

interface 接口名称{

//接口中的常量声明

//接口中的抽象方法声明

}

5、类可以通过实现接口的方式来具有接口中定义的功能，基本语法：

class 类名 implements 接口名{

}

要点：① 一个类可以同时实现多个接口

② 一个接口可以被多个无关的类实现

③ 一个雷实现接口必须实现接口中所有的抽象方法，否则必须定义为抽象类。

6、接口的继承：

（1）Java中接口可以继承接口，与类的继承概念一致，一个接口继承一个父接口就会继承父接口中定义的所有方法和属性。

（2）Java中接口的继承是多继承机制，即一个接口可以同时继承多个接口。

（3）接口继承的基本语法 :

interface 接口名 extends 父接口1,父接口2,……{

}

7、接口的意义：

① 弥补Java中单继承机制的不足。

② 接口只有方法的定义没有方法的实现，即都是抽象方法，这些方法可以在不同的地方被不同的类实现，而这些实现可以具有不同的行为（功能）。

8、抽象类：

* 1. 不能实例化，但是可以声明抽象类的引用
  2. 包含抽象方法的类必须定义为抽象类
  3. 不包含抽象方法的类不一定不是抽象类（抽象类中可以不含抽象方法）
  4. 抽象方法不含方法体，必须显示定义为abstract

（不同于void display(){}）

* 1. 抽象类的子类必须实现父类所有的抽象方法，否则也必须定义为抽象类

9、接口：

①弥补Java单一继承的不足

②不能实例化，但是可以声明接口变量

③包含常量和方法的声明，不含变量和方法的实现（常量可以没有final修饰，必须初始化，在接口实现类中不能修改值；方法可以没有abstract，在接口实现类中必须实现；static方法和default方法有方法体）

1. 接口中的抽象方法不能有方法体（即便是空的方法体{ }也不行）
2. 接口不能继承类
3. 接口可以继承（extends）1个或多个接口（ ,分隔开）

⑦一个类可以实现（implements）1个或多个接口（ ,分隔开）

10、Object类

Object类是所有Java类的祖先。每个类都会使用Object作为父类。

在不明确给出父类的情况下，java会自动把Object作为要定义类的父类

可以使用类型为Object的变量指向任意类型的对象

11、Object类有一个默认的构造方法public Object()，在构造派生类实例时，都会先调用这个默认构造方法。

Object类的变量只能用作各种值的通用持有者。若要对他们进行任何专门的操作，则须知道他们原始数据类型并进行类型的转换。

12、equals()方法：用于测试某个对象是否同另一个对象相等。它在Object类中的实现是判断两个对象是否指向同一块内存区域。

**\*** Object中的equals方法是说两个对象是否具有相同的引用

**\*** String中重写了equals方法，判断对象的值是否相同

**\*** == 判断地址是否相同

13、Java语言规范要求equals方法具有下面的特点：

①自反性：对于任何非空引用值 x，x.equals(x) 都应返回 true。

②对称性：对于任何非空引用值 x 和 y，当且仅当 y.equals(x) 返回 true 时，x.equals(y) 也应返回 true。

③传递性：对于任何非空引用值 x、y 和 z，如果 x.equals(y) 返回 true，并且 y.equals(z) 返回 true，那么 x.equals(z) 应返回 true。

④一致性：对于任何非空引用值 x 和 y，多次调用 x.equals(y) 始终返回 true 或始终返回 false，前提是对象上 equals 比较中所用的信息没有被修改。

⑤对于任何非空引用值 x，x.equals(null) 都应返回 false。

14、hashCode方法：

* public int hashCode() 返回该对象的哈希码值。
* hashCode 的常规协定是：
  1. 在 Java 应用程序执行期间，在同一对象上多次调用 hashCode 方法时，必须一致地返回相同的整数，前提是对象上 equals 比较中所用的信息没有被修改。
  2. 如果根据 equals(Object) 方法，两个对象是相等的，那么在两个对象中的每个对象上调用 hashCode 方法都必须生成相同的整数结果。

15、toString方法：

* public String toString() 返回该对象的字符串表示。
* 通常，toString 方法会返回一个“以文本方式表示”此对象的字符串。建议所有派生类都重写此方法。
* Object 类的 toString 方法返回一个字符串，该字符串由类名（对象是该类的一个实例）、at 标记符“@”和此对象哈希码的无符号十六进制表示组成。
* 该方法返回一个字符串，它的值等于：

getClass().getName()+'@'+Integer.toHexString(hashCode())

**【第七章 多态】**

1. 多态是指：同一操作作用于不同的类的对象，将产生不同的执行结果 。
2. 多态的优势：①增强了程序的灵活性；②增强了程序的扩展性
3. 多态的条件：
   1. 继承：存在有继承关系的子类和父类
   2. 重写：子类对父类中某些方法进行重新实现
   3. 向上转型：需要子类的对象赋给父类的引用
4. 多态的实现：

1）向上转型就是父类引用指向子类对象；“向上转型”既可以使用子类强大的功能，又可以抽取父类的共性。

例：定义了一个派生类Teacher，继承自Person。

* 通过Teacher teacher = new Teacher()实例化Teacher对象；
* 通过Person person= new Teacher();表示定义了一个Person类型的引用，指向新建的Teacher类型的对象，这就称为“向上转型”。

2）通过将子类对象赋值给父类变量来实现动态方法调用

5、在运行时根据对象的实际类型调用方法叫动态绑定，又叫后期绑定、运行时绑定。

Java中的多态是通过动态绑定实现的。

一个父类引用，指向不同的子类对象，执行该方法时，表现出不同的行为。

6、

<1>对于父类中定义的方法，如果子类中重写了该方法，那么父类类型的引用将会调用子类中定义的这个方法。

<2>父类中的一个方法只有在父类中定义而在派生类中没有重写的情况下，才可以被父类类型的引用调用。

<3>对于子类中定义而父类中没有的方法无法调用。

**【第八章 包装器类】**

1、Java的八种基本数据类型对应的包装器类分别为：Byte、Short、**Character**、**Integer**、Long、Float、Double、Boolean。

2、创建包装器类型对象的两种方式：

1. 构造方法：new

例：Integer i = new Integer(9);

int num = 10;

Integer i = new Integer(num);

//相当于Integer i = new Integer(10);

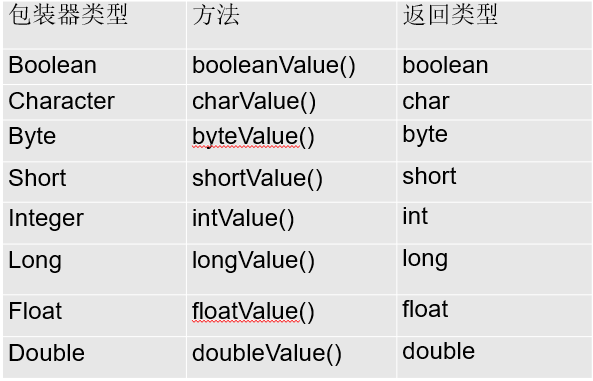
1. 调用包装器类型的valueOf方法

例：Double d = **Double.**valueOf(3.14);

Float f = **Float.**valueOf(10.2f);

3、包装器类型对象共同的特点：对象一旦赋值，其值不能再改变

4、包装器类型与基本数据类型的转换：



5、自动装箱：

实现Integer类型数据++/--操作。

Integer y = new Integer (567);

y++;

**注意：y++之前和之后所引用的内存地址不同**

等价于：

Integer y = new Integer (567);

int x = y.intValue();

x++;

y = Integer.valueof(x);

6、基本数据类型会被加宽到更宽泛的基本数据类型

- 基本数据类型和对应的包装器类型是不同的数据类型

自动装箱给方法的重载带来了一定的难题。

可变元参数列表给方法的重载带来了一定的难题。

7、选择重载时应匹配哪个方法时遵循以下原则：

* 1. 能否直接匹配
  2. 加宽后能否匹配
  3. 装箱后能否匹配
  4. 先装箱再向上转型能否匹配

可以先装箱，再向上转型（例如：int可以装箱成Integer提升到Object）

* 1. 有不定长参数能否匹配

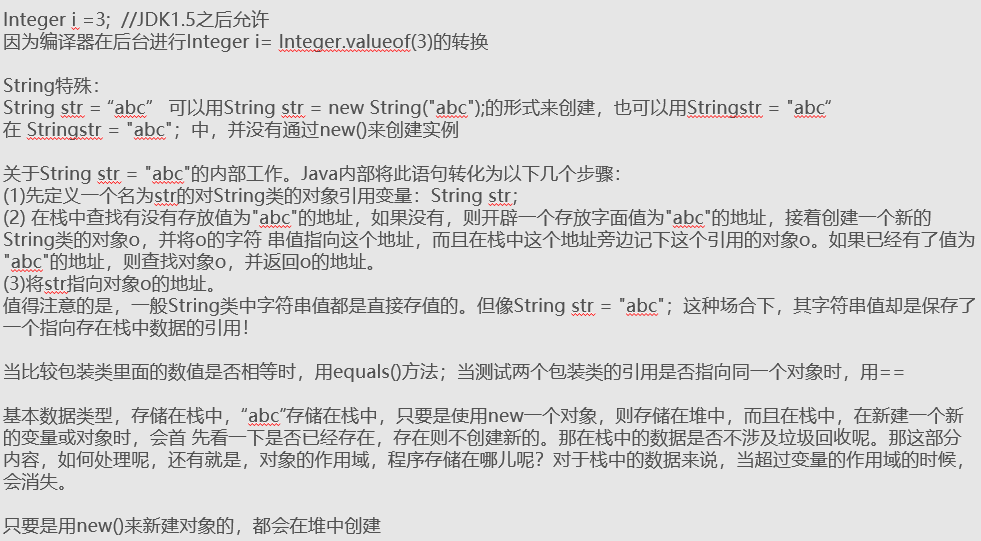
基本数据类型参数先加宽，次数没限制

装箱后不加宽匹配（装箱只是对原有数据类型一次装箱）

不定长参数的匹配是基本数据类型的不定长参数（可以是直接匹配，也可以是加宽后匹配）

如果参数是包装器类型，只能直接匹配（或加宽成Object类型后匹配）

8、



**【第九章 内部类】**

1. 内部类：定义在另一个类中的类，也称内置类或嵌套类
2. 内部类的特点：
3. 内部类是一种编译时的语法，编译后生成的两个类是独立的两个类。

（对于一个名为outer的外部类和其内部定义的名为inner的内部类。编译完成后出现outer.class和outer$inner.class两类）

1. 内部类可以访问外部类的任何成员（包括私有成员），但外部类不能直接访问内部类的成员
2. 内部类可为静态，可以用protected、private修饰，二外部类只能使用public和缺省的包访问权限
3. 用内部类定义在外部类中不可访问的属性，这样就在外部类中实现了比外部类的private还要小的访问权限

内部类知道外部类中有什么，但外部类不知道内部类具体有什么，所以外部类访问内部类时必须先实例化该内部对象

1. 广泛意义上，内部类的形式：
   1. 成员内部类
   2. 局部内部类
   3. 匿名内部类
   4. 静态内部类
2. 注意事项：

①在内部类中访问内部类自己的变量直接用变量名或者用this.变量名

②在内部类中访问外部类中与内部类同名的实例变量用外部类名.this.变量名

③如果内部类中没有与外部类同名的变量，则可以直接用变量名访问外部类变量（包括静态变量）

④内部类和外部类的实例变量可以共存：允许内部类中有和外部类相同的成员变量（如内部类和外部类都有成员变量name）

1. 成员内部类：作为外部类的一个成员存在，与外部类的属性和方法并列。内部类和外部类的实例变量可以共存。

例：class OuterClass{

OuterClass() {

System.out.println("Outer class");

}

class InnerClass { //定义内部类

InnerClass(){

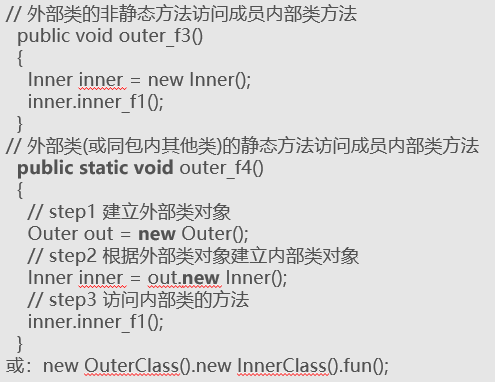
System.out.println("Inner class");

}

}

}

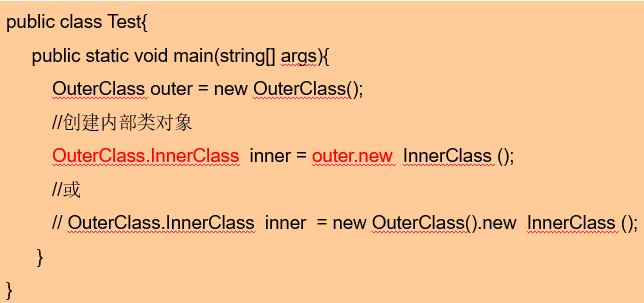
成员内部类中不能定义静态成员；成员内部类中可以访问外部类的所有成员。



1. 成员内部类-创建对象：

建立内部类对象时应注意：

* 在外部类的内部可以直接使用Inner s = new Inner();（因为外部类直到Inner是哪个类，所以可以生成对象）
* 在外部类的外部要生成（new）一个内部类对象需要手写爱你建立一个外部类对象（外部类可用），然后再生成一个内部类对象。

例：

1. 成员内部类-访问：
2. 在内部类访问内部类变量：[内部类名.]this.属性
3. 内部类作为外部类的成员，可以访问外部类的私有成员或属性。

访问方式：外部类名.this.属性

1. 在外部类的外部访问内部类：内部类的对象.成员名
2. 局部内部类：在方法中定义的内部类称为局部内部类。

局部内部类不能加修饰符public、protected和private，其范围为定义它的代码块。

例：class OuterClass{

//称为外包方法

public void display() {

//定义局部内部类

class InnerClass {

public void displayInner() {

System.out.println("display inner class");

}

}

InnerClass in = new InnerClass();

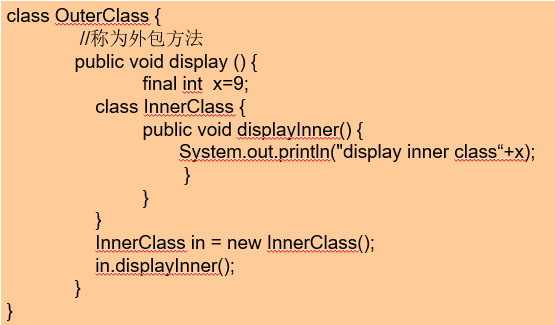
in.displayInner();

}

}

1. 局部内部类-特点：
   1. 可以访问外包方法之外的外部类之内的所有成员。还可以访问所在外包方法中的参数。
   2. 局部内部类只能访问局部中被final修饰的内部变量
   3. 局部内部类不能声明为接口
   4. 要想使用局部内部类时，需要生成外部类对象，通过外部类对象调用外包方法，在方法中才能调用局部内部类。

例：



10、匿名内部类：将局部内部类的使用再深入一步，假如只创建这个类的一个对象，就不必对内部类命名了。这种类被称为匿名内部类。

例：interface anonymous{ //定义一个接口

void display();

}

class NoClassName{

public void print(){

anonymous anon = new anonymous (){ //定义匿名类

public void display() {

System.out.println(“implement anonymous “);

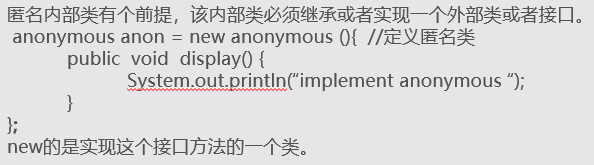
}

}**;**

anon.display();

}

}



11、匿名内部类-特点：

1. 匿名内部类没有类名，它必须继承一个类或是实现一个接口。不能有显示的extends和implements子句
2. 匿名内部类不能有构造函数，因为它没有类名。可以通过new<父类名>的方法创建对象，**匿名类的创建和定义同时进行**
3. 匿名类只能一次性的创建其对象
4. 匿名类可以在方法体中，也可以在参数列表中(在方法的参数中)

匿名内部类就是一个匿名子类对象。

格式： new 父类OR接口（）{子类内容}

12、匿名内部类-注意事项：

①匿名内部类一定是在new的后面其隐含实现一个接口或继承一个类，没有类名

②匿名内部类不能是抽象类

**③匿名内部类必须实现它的抽象父类或接口里面的所有抽象方法**

使用场景：

当函数参数是接口类型时，而且接口中的方法不超过三个，可以使用匿名内部类作为实际参数进行传递。

13、静态内部类：静态内部类定义在类中，在任何方法之外，用static定义。

例：class Outer {

public void display(){ }

protected static class Inner{ }//静态成员类

}

静态内部类的对象可以直接生成。

例：class Test {

public static void mian(string[] args) {

Outer.Inner in = new Outer.Inner ();

}

}

如果内部类是静态的，它就相当于一个外部类，直接实例化内部类对象，不用建立外部类对象。

静态成员属性可以由类名直接调用。

14、静态内部类-特点：

1. 静态内部类能直接访问外部类的静态成员，不能直接访问外部类的非静态属性成员（外部内的对象访问）
2. 静态内部类里面可以定义静态成员，其他内部类不可以
3. 如果内部类中定义了静态内部成员，该内部类也必须定义为静态的。

**【第十章 static修饰符和枚举】**

1. 静态属性

语法定义：权限修饰符 **static** 数据类型 属性名称;

使用方法：类名.属性名;

1. 为什么要引入静态属性？

使各对象之间的数据有沟通的渠道，实现数据的共享

1. 静态属性的特点：
   * 1. 声明为static的属性也被称为类属性
     2. 当创建一个对象时，并不产生static属性的拷贝
     3. 该类所有的实例共享同一个static属性
     4. 在类装载时，只分配一块存储空间，所有此类的对象都可以操控此块存储空间。

4、静态属性属于类和这个类所实例化出来的所有对象而不属于具体的对象，为不同的对象所共有，因此public（共有的）静态属性在类外有两种访问方式。

* 1. 通过类名.公有静态属性名
  2. 通过对象名.公有静态属性名

5、静态属性值一旦改变，所有类的对象均共享改变

6、静态方法：public static void main(String[] args){ }

语法定义：

权限修饰符 **static** 返回值类型 方法名(形式参数列表){

…方法体

}

使用方法：类名.方法名(实际参数列表)

7、静态方法：

1. 功能：用于访问静态属性

2）公有的静态方法在类外也有两种访问方式：

①通过类名.公有静态方法名（[参数]）

②通过对象名.公有静态方法名（[参数]）

3）使用规则（类内部）

① 在静态方法中，仅能直接调用其他的 static 类型的方法。

② 在静态方法中，只能直接访问 static 类型的属性。

③ 在静态方法中，不能以任何方式使用 this 或 super。

**4）特点：不调用不执行**

8、静态代码块：

1）定义语法：static{

…代码块

}

2）功能：初始化static变量

**3）特点：仅在该类被加载时执行一次**

4）涉及到继承的时候，会先初始化父类的static代码块，然后是子类的静态代码块

5）子类实例化对象时的执行顺序：

1. 父类的静态代码块
2. 子类的静态代码块
3. 父类的构造方法
4. 子类的构造方法

6）执行顺序

- **静态属性和静态代码块**（按照**定义顺序**，**在类被加载的时候只执行一次**）

- **非静态属性和非静态代码块**（按照**定义顺序**，**在类被实例化的时候执行**）

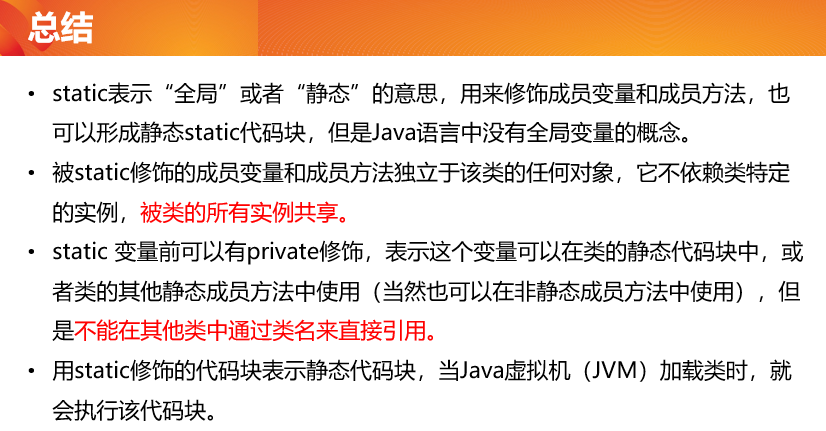
- **构造方法**

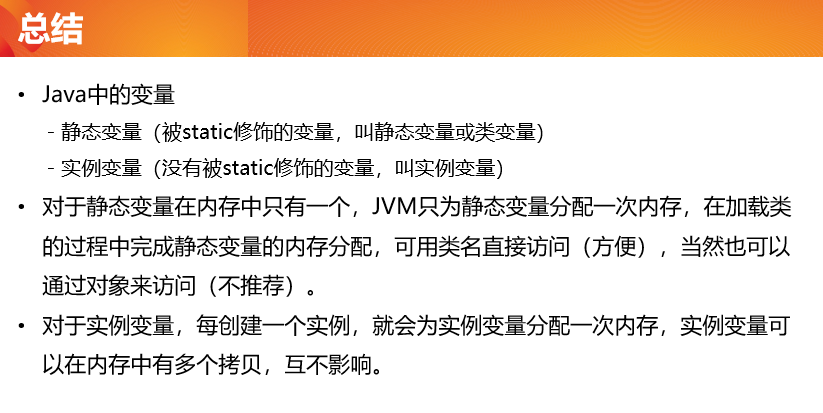
9、静态内部类：

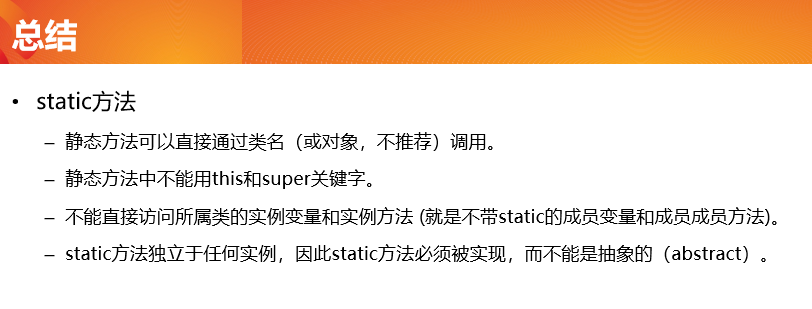
* 通常一个普通类不允许声明为静态的，只有内部类才可以。
* 声明为静态的内部类可以直接作为一个普通类来使用，而不需实例化一个外部类。

10、static和final一起使用

* static final用来修饰成员变量和成员方法，可简单理解为“全局常量”。
* 对于常量，表示一旦赋值就不可修改，并且通过类名可以访问。
* 对于方法，表示不可覆盖，并且可以通过类名直接访问。

11、





12、枚举定义：

权限修饰符 enum enum\_name{

元素1

元素2

……

}

13、Java中的enum本质就是一个class

枚举的成员就是枚举对象，只不过是静态常量。

14、枚举与类的区别：

枚举不可以实例化，编译器会自动为其构造方法加上private

通常枚举值为枚举成员变量提供get方法，而不提供set方法

15、枚举的特点：

* 数据集
  + 他们的数值在程序中是稳定的。
  + 元素个数有限。
* 所有枚举类都继承了 Enum 类的方法
  + toString
  + equals
  + hashcode
  + …
  + 注： equals、hashcode 方法是 final 的，所以不可以被枚举重写（只可以继承）。但是，可以重写 toString 方法。
  + **Java 不允许使用 = 为枚举常量赋值。**

16、枚举成员：

枚举成员也是变量，变量名不能以数字开头；

Java不允许使用 = 为枚举常量赋值。

17、