## 3.1、匿名内部类（重点）

内部类：在一个类的内部定义了另外的类，称为内部类，匿名内部类指的是没有名字的内部类。为了清楚内部类的主要作用，下面首先来观察如下的程序：

**范例：**思考程序的问题

|  |
| --- |
| interface Message { // 定义了一个接口  public void print() ; // 抽象方法  }  **class MessageImpl implements Message { // 定义接口的实现类**  **public void print() {**  **System.out.println("Hello World .") ;**  **}**  **}**  class Demo {  public static void get(Message msg) { // 接收接口对象  msg.print() ;  }  }  public class TestDemo {  public static void main(String args[]) {  Message msg = new MessageImpl() ; // 子类为接口实例化  Demo.get(msg) ; // 传递msg对象  }  } |

以上的代码是属于正常的开发范畴，但是如果说现在MessageImpl这个子类只使用那么唯一的一次，有必要按照以上的方式进行定义吗？这个时候那么MessageImpl就没有什么意义了，但是可以利用匿名内部类的概念来解决此问题。匿名内部类是在抽象类和接口的基础之上所发展起来的一种应用。

|  |
| --- |
| interface Message { // 定义了一个接口  public void print() ; // 抽象方法  }  class Demo {  public static void get(Message msg) { // 接收接口对象  msg.print() ;  }  }  public class TestDemo {  public static void main(String args[]) {  Demo.get(new Message() **{ // 匿名内部类**  **public void print() {**  **System.out.println("Hello World .") ;**  **}**  **}**) ;  }  } |

对于匿名内部类的使用，暂时只需要记住它的语法即可，因为如果要想真正的去使用它，还需要一段时间的课程。

## 3.2、Object类（重点）

在Java的定义之中，除了Object类之外，所有的类实际上都存在继承关系，即：如果现在定义了一个类，没有默认继承任何一个父类的话，则默认将继承Object类，以下两种类的最终定义效果是完全一样的。

|  |  |
| --- | --- |
| class Person {} | class Person **extends Object** {} |

那么按照这样的方式理解，即：Object类可以接收所有类的实例化对象。

|  |
| --- |
| class Person {}  public class TestDemo {  public static void main(String args[]) {  Object obj = new Person() ;  Person per = (Person) obj ;  }  } |

**如果在日后的开发之中，一个操作可能接收所有类的对象，那么使用Object作为参数最合适。**

除此之外，对于任意的一个简单Java类而言，理论上讲应该覆写Object类之中的三个方法：

· 取得对象信息：public String toString()；

· 对象比较：public boolean equals(Object obj)；

· **取得哈希码：**public int hashCode()。

### 3.2.1、取得对象信息：toString()

在之前提示过，如果说现在一个对象直接输出，那么默认情况下输出的是一个对象的地址。

|  |
| --- |
| class Person {}  public class TestDemo {  public static void main(String args[]) {  Person per = new Person() ;  System.out.println(per) ; // Person@ 1f 6226  System.out.println(per.toString()) ; // Person@ 1f 6226  }  } |

现在可以清楚地方发现，如果直接输出一个类的对象，那么和这个对象调用toString()方法的结果是完全一样的，那么就可以得出一个结论：对象直接输出默认调用了Object类之中的toString()方法，但是默认的toString()方法有一个特点：为了适用于所有的子类，那么在toString()默认情况下就是输出了对象地址，当然，每一个子类也可以自己进行修改。

|  |
| --- |
| class Person {  private String name ;  private int age ;  public Person(String name,int age) {  this.name = name ;  this.age = age ;  }  public String toString() { // 方法覆写  return "姓名：" + this.name + "，年龄：" + this.age ;  }  }  public class TestDemo {  public static void main(String args[]) {  Person per = new Person("张三",20) ;  System.out.println(per) ; // Person@ 1f 6226  }  } |

### 3.2.2、对象比较：equals()

实际上对于equals()方法应该并不陌生了，这个方法在String类中见过，String是Object类的子类，所以String类的equals()方法就是覆写了Object类中的equals()方法，在Object类之中，默认的equals()方法实现比较的是两个对象的内存地址数值，但是并不符合于真正的对象比较需要。对象比较之前也写过，但是之前是自己定义了一个新的方法名称，今天可以给出标准的方法名称：equals()。

|  |
| --- |
| class Person {  private String name ;  private int age ;  public Person(String name,int age) {  this.name = name ;  this.age = age ;  }  public String toString() { // 方法覆写  return "姓名：" + this.name + "，年龄：" + this.age ;  }  public boolean equals(Object obj) {  if (this == obj) {  return true ;  }  if (obj == null) {  return false ;  }  if (! (obj instanceof Person)) { // 不是本类对象  return false ;  }  // 因为name和age属性是在Person类中定义，而Object类没有  Person per = (Person) obj ;  if (this.name.equals(per.name) && this.age == per.age) {  return true ;  }  return false ;  }  }  public class TestDemo {  public static void main(String args[]) {  Person per1 = new Person("张三",20) ;  Person per2 = new Person("张三",20) ;  System.out.println(per1.equals("Hello")) ;  System.out.println(per1.equals(per2)) ;  }  } |

### 3.2.3、使用Object接收所有的引用数据类型

Object是所有类的父类，那么Object类可以接收所有类的对象，但是在Java设计的时候，考虑到引用数据类型的特殊性，所以Object类实际上是可以接收所有引用数据类型的数据，这就包括了数组、接口、类。

**范例：**使用Object类接收数组，数组和Object没有任何明确的定义关系

|  |
| --- |
| public class TestDemo {  public static void main(String args[]) {  Object obj = **new int [] {1,2,3}** ; // 接收数组  if (obj instanceof int []) {  int [] data = (int []) obj ; // 向下转型  for (int x = 0 ; x < data.length ; x ++) {  System.out.println(data[x]) ;  }  }  }  } |

**范例：**接收接口对象，从接口定义而言，它是不能去继承一个父类的，但是由于接口依然属于引用类型，所以即使没有继承类，也可以使用Object接收。

|  |
| --- |
| interface Message {  }  class MessageImpl implements Message { // 定义接口子类  public String toString() {  return "New Message : Hello World ." ;  }  }  public class TestDemo {  public static void main(String args[]) {  Message msg = new MessageImpl() ; // 向上转型  Object obj = msg ; // 使用Object接收接口对象，向上转型  Message temp = (Message) obj ; // 向下转型  System.out.println(temp) ; // toString()  }  } |

从代码上讲，以上只能算是一个固定的操作概念，不过从实际来讲，因为有了Object类的出现，所以的操作的数据就可以达到统一，那么之前的链表程序，就应该变的很方便了，所有的数据都使用Object接收，所有的对象比较（删除、查找）都可以使用equals()。

## 3.3、包装类（重点）

在Java的设计之中，一直提倡一个原则：一切皆对象，这个原则本身有一个漏洞 —— 基本数据类型不是对象，所以这个原则就出现了问题。那么如果说现在这个问题由我们来解决，该如何做呢？

|  |
| --- |
| class Int { // 类  private int num ; // 基本型  public Int(int num) {  this.num = num ;  }  public int intValue() {  return this.num ;  }  }  public class TestDemo {  public static void main(String args[]) {  Int temp = new Int(10) ; // 把基本类型变为类  int result = temp.intValue() ;  System.out.println(result \* result) ;  }  } |

以上的操作是将基本类型变为了一个对象的形式进行操作了，但是这里面有一个问题：基本数值型数据是可以进行数学运算的，可是以上变为了类的形式，那么肯定无法直接计算了。而且以上的问题既然我们都可以想到方法去解决，那么java也一定早已解决，为此它专门提供了八种包装类：**byte（Byte）、short（Short）、int（Integer）、long（Long）、float（Float）、double（Double）、boolean（Boolean）、char（Character）**，而这八种包装类又分为两大阵营：

· 数值型（Number子类）：Byte、Short、Integer、Float、Double、Long；

· 对象型（Object子类）：Boolean、Character。

可是对于Number的子类，就必须观察出Number类之中定义的方法：byteValue()、intValue()、doubleValue()、shortValue()、floatValue()、doubleValue()，就是从包装的类之中取得所包装的数值。

### 3.3.1、装箱与拆箱

在基本数据类型和包装类之间的转换操作之中分为两个重要概念：

· 装箱操作：将基本数据类型变为包装类，称为装箱，包装类的构造方法；

· 拆箱操作：将包装类变为基本数据类型，称为拆箱，各个类的xxValue()方法。

**范例：**以int和Integer为例

|  |
| --- |
| public class TestDemo {  public static void main(String args[]) {  Integer var = new Integer(15) ; // 装箱  int result = var.intValue() ; // 拆箱  System.out.println(result \* result) ;  }  } |

**范例：**以double和Double为例

|  |
| --- |
| public class TestDemo {  public static void main(String args[]) {  Double var = new Double(15.5) ; // 装箱  double result = var.doubleValue() ; // 拆箱  System.out.println(result \* result) ;  }  } |

**范例：**以boolean和Boolean为例

|  |
| --- |
| public class TestDemo {  public static void main(String args[]) {  Boolean var = new Boolean(true) ; // 装箱  boolean result = var.booleanValue() ; // 拆箱  if (result) {  System.out.println("Hello World .") ;  }  }  } |

以上的操作是在JDK 1.5之前所进行的必须的操作，但是到了JDK 1.5之后，Java提供了自动的装箱和拆箱机制，并且包装类的对象可以自动的进行数学计算了。

|  |
| --- |
| public class TestDemo {  public static void main(String args[]) {  Integer var = 15 ; // 自动装箱  int result = var ; // 自动拆箱  System.out.println(++ var \* result) ;  }  } |
| public class TestDemo {  public static void main(String args[]) {  Boolean var = true ; // 自动装箱  if (var) { // 自动拆箱后操作  System.out.println("Hello World .") ;  }  }  } |

那么正是因为有了这样的自动装箱和拆箱的机制，所以Object也可以接收基本数据类型的数据。

|  |
| --- |
| public class TestDemo {  public static void main(String args[]) {  Object obj = 15 ; // int --> 自动装箱 --> Object  int result = (Integer) obj ; // Object --> 包装类 --> 自动拆箱  System.out.println(result \* result) ;  }  } |

但是到此处还有一个小问题，实际上这一问题之前已经见过的。

|  |
| --- |
| public class TestDemo {  public static void main(String args[]) {  Integer x = new Integer(10) ; // 新空间  Integer y = 10 ; // 入池  Integer z = 10 ;  System.out.println(x == y) ; // false  System.out.println(x == z) ; // false  System.out.println(z == y) ; // true  System.out.println(x.equals(y)) ; // true  }  } |

以后使用包装类的时候还需要考虑equals()和==的区别。

### 3.3.2、数据转型

包装类之中所提供的最大优点还在于可以将字符串变为指定的基本数据类型，下面列出几个操作：

· **Integer类：**public static int parseInt(String s)；

· **Double类：**public static double parseDouble(String s)；

· **Boolean类：**public static boolean parseBoolean(String s)；

但是Character这个包装类之中，并没有提供一个类似的parseCharacter()，因为字符串String类之中提供了一个charAt()方法，可以取得指定索引的字符，而且一个字符的长度就是一位。

**范例：**将字符串变为int

|  |
| --- |
| public class TestDemo {  public static void main(String args[]) {  String str = "16" ;  int result = Integer.parseInt(str) ; // String --> int  System.out.println(result \* result) ;  }  } |

但是需要提醒的是，在执行这种转换的操作过程之中，字符串中的全部内容必须由数字所组成，如果有一位内容不是数字，则在转换的过程之中将出现如下的错误提示：NumberFormatException。

**范例：**将字符串变为double

|  |
| --- |
| public class TestDemo {  public static void main(String args[]) {  String str = "36." ;  double result = Double.parseDouble(str) ;  System.out.println(result \* result) ;  }  } |

**范例：**将字符串变为boolean型数据

|  |
| --- |
| public class TestDemo {  public static void main(String args[]) {  String str = "true" ;  boolean result = Boolean.parseBoolean(str) ;  if (result) {  System.out.println("Hello World .") ;  }  }  } |

提示：在使用Boolean型包装类的时候，如果字符串之中的内容不是true或者是false，统一都按照false处理。

以上的操作是通过字符串变为了一些基本类型的数据，但是反过来讲，基本类型如何变为字符串呢？

· **方式一：**任何的基本类型数据遇到了String之后都变为String型数据；

|  |
| --- |
| public class TestDemo {  public static void main(String args[]) {  int num = 100 ;  String str = num + **""** ; // int --> String  System.out.println(str.length()) ;  }  } |

· **方式二：**利用String类的方法，public static String valueOf(数据类型 b)

|  |
| --- |
| public class TestDemo {  public static void main(String args[]) {  int num = 100 ;  **String str = String.valueOf(num) ; // int --> String**  System.out.println(str.length()) ;  }  } |

字符串向各个基本数据类型的转换，是一个非常重要的内容，必须熟练。

## 3.4、思考题（核心）

一个动物园有多种动物，现在要求实现某类动物的添加、某一类动物的取消、以及关键字检索动物信息的操作

**提示：**要求代码模型，动物属性以名称和年龄为主

**分析：**

1、 首先需要分析出本程序的关系：一个动物园有多种动物，按照道理是属于一对多的操作关系，而多的一方肯定通过Link来表示，而且Link可以针对于各种类型。

2、 什么叫动物？动物是一个标准，既然是标准，肯定在类之上，那么一定就是接口，所以对于整个程序应该先把接口定义出来，那么接口只有两个功能：

**范例：**定义出动物的接口

|  |
| --- |
| interface Animal { // 动物  public String getName() ;  public int getAge() ;  } |

**范例：**定义动物园，动物园里面只关心接口，而不关心具体有什么动物。

|  |
| --- |
| class Zoo {  private Link animals = new Link() ; // 多个动物  public void add(Animal ani) { // 增加动物  this.animals.add(ani) ; // 增加动物  }  public void delete(Animal ani) {  this.animals.remove(ani) ; // 需要equals()  }  public Link search(String keyWord) {  Link result = new Link() ;  Object obj [] = this.animals.toArray() ;  for (int x = 0 ; x < obj.length ; x ++) {  Animal ani = (Animal) obj[x] ;  if (ani.getName().contains(keyWord)) { // 满足  result.add(ani) ;  }  }  return result ;  }  } |

动物园没和具体的动物有关，只和动物的标准有关，而后下面按照标准去定义动物。

**范例：**定义狗

|  |
| --- |
| class Dog implements Animal {  private String name ;  private int age ;  public Dog(String name,int age) {  this.name = name ;  this.age = age ;  }  public String getName() {  return this.name ;  }  public int getAge() {  return this.age ;  }  public boolean equals(Object obj) {  if (this == obj) {  return true ;  }  if (obj == null) {  return false ;  }  if (!(obj instanceof Dog)) {  return false ;  }  Dog dog = (Dog) obj ;  if (this.name.equals(dog.name)  && this.age == dog.age) {  return true ;  }  return false ;  }  public String toString() {  return "〖狗的信息〗名字：" + this.name + "，年龄：" + this.age ;  }  } |

**范例：**定义老虎

|  |
| --- |
| class Tiger implements Animal {  private String name ;  private int age ;  public Tiger(String name,int age) {  this.name = name ;  this.age = age ;  }  public String getName() {  return this.name ;  }  public int getAge() {  return this.age ;  }  public boolean equals(Object obj) {  if (this == obj) {  return true ;  }  if (obj == null) {  return false ;  }  if (!(obj instanceof Dog)) {  return false ;  }  Tiger t = (Tiger) obj ;  if (this.name.equals(t.name)  && this.age == t.age) {  return true ;  }  return false ;  }  public String toString() {  return "〖老虎的信息〗名字：" + this.name + "，年龄：" + this.age ;  }  } |

**范例：**定义测试类

|  |
| --- |
| public class TestDemo {  public static void main(String args[]) {  Zoo zoo = new Zoo() ; // 动物园  zoo.add(new Dog("花狗",1)) ;  zoo.add(new Dog("黄狗",1)) ;  zoo.add(new Dog("黑狗",1)) ;  zoo.add(new Dog("斑点狗",1)) ;  zoo.add(new Tiger("斑点虎",2)) ;  zoo.add(new Tiger("黑虎",2)) ;  zoo.add(new Tiger("花虎",2)) ;  zoo.delete(new Dog("斑点狗",1)) ; // 删除  Link result = zoo.search("斑点") ;  Object obj [] = result.toArray() ;  for (int x = 0 ; x < obj.length ; x ++) {  System.out.println(obj[x]) ;  }  }  } |

利用此代码模型，日后可以解决以下问题：

· 一个停车场可以停放多种车辆；

· 一列火车上可以有不同国家的人乘座；

· 一个人的书包可以放多物品；

· 一个超市里有多种商品，每个人可以购买自己的商品。

# 4、总结

1、 匿名内部类现在不使用；

2、 Object类可以接收任意的对象，是所有类的父类；

3、 包装类可以使用自动装箱和拆箱进行操作，以及可以与字符串之间进行的数据互相转换；

4、 接口和链表的应用。

# 5、预习任务

异常的捕获及处理、包及访问控制权限、Eclipse开发工具。