day02 接口 多态 集合框架(一)

1. 接口

Interface
原因：简化书写
接口是类。接口中有成员变量，抽象方法。
例子：

```
interface A{
    int a();
    void b();
}

interface B extends A{
    void c();
}

class Bb{
    void Bb(){}
}

class Aa extends Bb implements B{

    @Override
    public int a() {
        return 0;
    }

    @Override
    public void b() {
    }

    @Override
    public void c() {

    }
}
```

继承：(`extends`)使得类与类之间产生了家族联系
作用：增加代码复用性。
接口：想用某功能，但不想管人家叫爹（不想加入人家的那个家族），那就实现那个接口。

2. 多态

一件事物多种形态。具体解释，及代码如下：

1. 函数的参数想接受多种类型的对象，而不修改参数列表；
2. 编译看左边，执行看右边。

对象的多态性。
class 动物 **{}**
class 猫 **extends** 动物 **{}**

class 狗 **extends** 动物 **{}**

猫 **x = new** 猫**();**

动物 **x = new** 猫**();//**一个对象，两种形态。猫这类事物即具备者猫的形态，又具备着动物的形态。这就是对象 的多态性。

简单说： 就是一个对象对应着不同类型**.**

多态在代码中的体现： 父类或者接口的引用指向其子类的对象。

多态的好处： 提高了代码的扩展性，前期定义的代码可以使用后期的内容。

多态的弊端： 前期定义的内容不能使用**(**调用**)**后期子类的特有内容。

多态的前提： **1**，必须有关系，继承，实现。 **2**，要有覆盖**.**

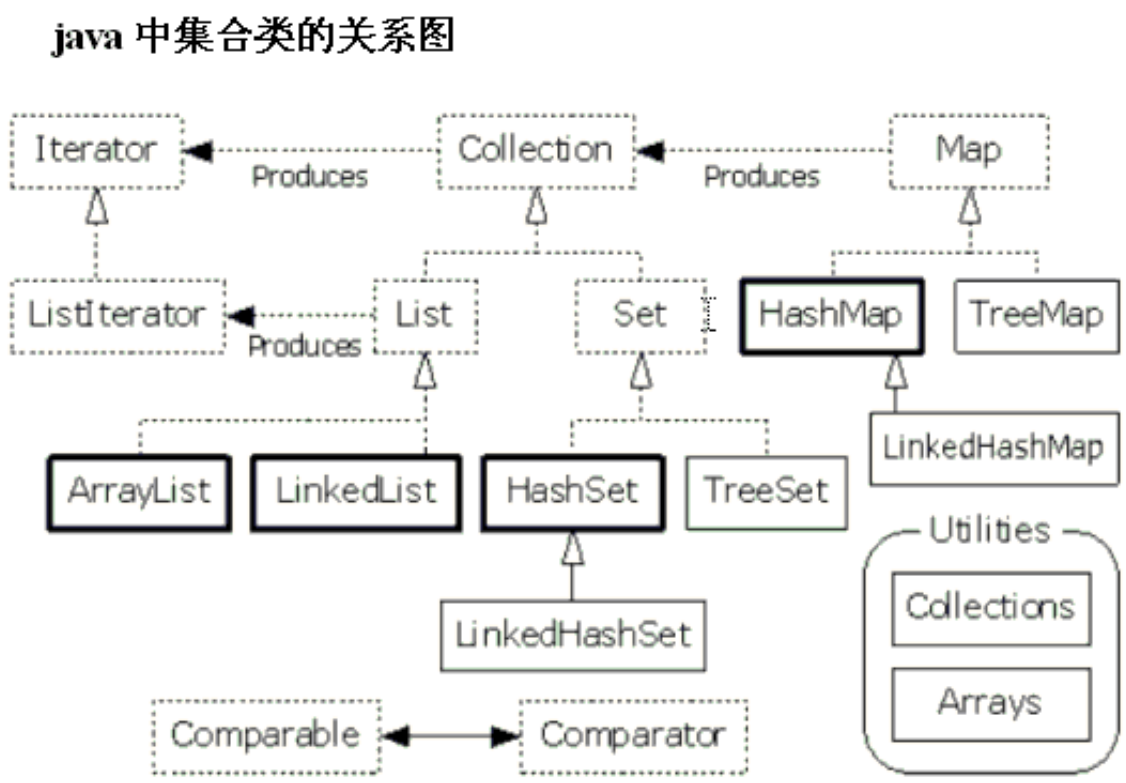
```java
package gzs_day02;

abstract class Animal {
    abstract void eat();
}

class Dog extends Animal {
    void eat() {
        System.out.println("啃骨头");
    }

    void lookHome() {
        System.out.println("看家");
    }
}

class Cat extends Animal {
    void eat() {
        System.out.println("吃鱼");
    }

    void catchMouse() {
        System.out.println("抓老鼠");
    }
}

class Pig extends Animal {
    void eat() {
        System.out.println("饲料");
    }

    void gongDi() {
        System.out.println("拱地");
    }
}

public class DuoTaiDemo {
    public static void main(String[] args) {
        Animal a = new Cat();
        method(new Pig());
    }

    public static void method(Animal a) { // Animal a = new Dog();
        a.eat();
        if (a instanceof Cat) {
            // instanceof：用于判断对象的具体类型。只能用于引用数据类型判断。 通常在向下转型前用于健壮性的判断。
            Cat c = (Cat) a;
            c.catchMouse();
        } else if (a instanceof Dog) {
```

```
            Dog d = (Dog) a;
            d.lookHome();
        } else {
            Pig p = (Pig) a;
            p.gongDi();
        }
    }
}
```

3. 集合框架

基本数据类型的变量多了，用数组存；
一维数组多了，用二维数组存；
对象多了，用集合(都在内存里)存。

框架图：



Collection: 不存键值对，只存单个元素，增加，删除，不能用角标。
- List: 可用操作角标，线性的，存取顺序一致；
- Set：去重，非线性的存储结构，存取顺序不一定一致；

Map：键值对-> <key, value>， 非线性的存储结构，存取顺序不一定一致。

Tree: 使得容器内元素有可比较性；
Hash: 查询时时间复杂度为$O(1)$。但空间复杂度增加，属于用空间换时间的技术。存、取时间复杂度是$O(1)$。

Linked: 使得容器存取顺序一致。比如：`LinkedHashSet`和`LinkedHashMap`。

学习一个家族式的框架，应先学祖宗类，使用该框架时，应用其子类。

3.1 Collection

- 添加
  `add()`：添加一个元素；
  `addAll()`：添加一个`Colleciton`对象。
- 删除
  'remove()': 一个元素
  'removeAll()': 一个集合
- 判断
  `boolean contains()`: 包含那个元素吗？
  `boolean containsAll()`: 包含那个集合吗？
  `boolean isEmpty()`: 空吗？
- 获取
  `int size()`: 元素个数
  `iterator()`: 迭代器
- 其他方法
  `boolean retainAll(Collection coll)`: 取交集
  `Object[] toArray()`: 集合转数组

例子：

```java
package gzs_day03;

import java.util.ArrayList;
import java.util.Collection;
import java.util.Iterator;

public class CollectionDemo {

    public static void main(String[] args) {
        // 1. 添加
        Collection coll = new ArrayList();
        coll.add("abcd1");
        coll.add("abcd2");
        coll.add("abcd3");
        coll.add("abcd4");

        System.out.println(coll.toString());

        // 2. 迭代方式1
        for (Iterator itt = coll.iterator(); itt.hasNext();) {
            String str = (String) itt.next();
            System.out.print(str + "  ");
        }
        System.out.println();

        // 3. 迭代方式2
        Iterator it = coll.iterator();
        while (it.hasNext()) {
            String str = (String) it.next();
            System.out.println(str);
        }

        System.out.println(coll.size());
        boolean bl = coll.remove("abcd2");
        System.out.println(bl);
        System.out.println(coll);

        // 4. 添加一个集合
        Collection coll2 = new ArrayList();
        coll2.addAll(coll);
        System.out.println(coll2);
    }
}
```

3.1.1 泛型

泛型： 约束容器内存放的元素的类型
例如：

```java
package gzs_day03;

import java.util.ArrayList;
import java.util.Collection;
import java.util.Iterator;

class A{
}

public class CollectionDemo2 {
    public static void main(String[] args) {
        // 1. <String> 泛型
        Collection<String> coll = new ArrayList<String>();
        coll.add("abcd1");
        coll.add("abcd2");
        // coll.add(new A()); 报错
        coll.add("abcd4");

        // 3. 迭代方式2
        Iterator<String> it = coll.iterator();
        while (it.hasNext()) {
            String str = (String) it.next();
            System.out.println(str);
        }
    }
}
```

3.2 ArrayList, LinkedList

带角标-> crud(增删改查的意思)

3.3 Set

Set去重功能展示：

```java
package gzs_day03;

import java.util.HashSet;
import java.util.Iterator;
import java.util.Set;

public class HashSetDemo {

    public static void main(String[] args) {
        Set s = new HashSet();
        // 去重
        s.add("abc1");
        s.add("abc2");
        s.add("abc3");
        s.add("abc2");

        Iterator it = s.iterator();
        while(it.hasNext()){
            String str = (String) it.next();
            System.out.println(str);
        }
    }
}
```

3.3.1 HashSet

HashSet 内部结构是哈希表
HashSet故事:

1. 无HashSet时，要查找元素，咋找？ 遍历 数组未排序时$O(n)$, 数组已经sorted时，$log(n)$
2. 有Hash时，有一个Hash函数，存时，针对要存储的内容，Hash函数生成地址并存入；取时，根据同样的Hash函数算出要取的那个对象的地址，直接取之，即：无需遍历。

HashSet例子：

Set中，因为要去重，所以涉及到被存储的元素间是否是相同元素的辨别问题。判断两个对象(如：Person对象)是否为相同元素涉及2个方法:
hashCode() 和 equals()

HashSet容器中，关于要存储的两个元素是否为同一个元素的判断过程：

1. 先用根据hashCode()生成两个对象的hash值，若不同，则直接认定为是不同对象；

2. 否则，若相同，则调用equals方法判断2个对象是否相等；
3. 若equals返回true，则认为是相同对象，丢弃后来者。若返回false，则认为是不同对象，既发生了碰撞。
4. 若发生碰撞，则在已有的hash地址旁边拉个小凳子，将第二个对象存入。

```java
package gzs_day03;

import java.util.HashSet;
import java.util.Iterator;

class Person {
    private String name;
    private int age;

    public Person() {
    }

    public Person(String name, int age) {
        super();
        this.name = name;
        this.age = age;
        return;
    }

    // alt + shift + s
    // 创建公有方法，供外界访问
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    // 覆盖 Object 的hashCode()和equals()
    @Override
    public int hashCode() {
        return name.hashCode() + age;
    }

    @Override
    public boolean equals(Object obj) {
        Person p = (Person) obj;
        return this.name.equals(p.name) && this.age == p.age;
    }
}
```

```
public class HashSetDemo2 {
    // 需求：存储自定义的Person类对象
    public static void main(String[] args) {
        HashSet hs = new HashSet();
        hs.add(new Person("li4", 24));
        hs.add(new Person("li4", 24));
        hs.add(new Person("li5", 6));
        hs.add(new Person("li8", 9));

        Iterator it = hs.iterator();
        while (it.hasNext()) {
            Person p = (Person) it.next();
            System.out.println(p.getName() + " : " + p.getAge());
        }
    }
}
```

上面的代码，实现了基于HashSet容器的，将2个`new Person("li4", 24)`对象去重。