

第十章 对象的关联与Object类

1. 对象的关联

- 继承：父类和子类的关系，继承的关联关系比较高，耦合性比较高
- 关联：耦合性比较低
 - 一个类的对象，是另一个类的属性
 - 比如：老师有姓名，年龄，还有电脑

代码块

```
1  package com.powernode.relevancy11;
2
3  class Computer{
4      private double cpu;
5      private int memory;
6      private int hardDisk;
7
8      public Computer(double cpu, int memory, int hardDisk) {
9          this.cpu = cpu;
10         this.memory = memory;
11         this.hardDisk = hardDisk;
12     }
13
14     public String getDetails(){
15         return "cpu:" + cpu + "\t内存: " + memory + "\t硬盘: " + hardDisk;
16     }
17 }
18 class Teacher {
19     private String name;
20     private int age;
21     //老师有电脑
22     private Computer computer;//基本类型的变量 叫变量 ， 引用类型的变量 叫对象
23     public Teacher(String name,int age,Computer computer){
24         this.name = name;
25         this.age = age;
26         this.computer = computer;
27     }
28     public String getDetails(){
29         return "姓名: " + name + "\t年龄: " + age + "\t电脑配置: " +
30             computer.getDetails();
31     }
32 }
```

```

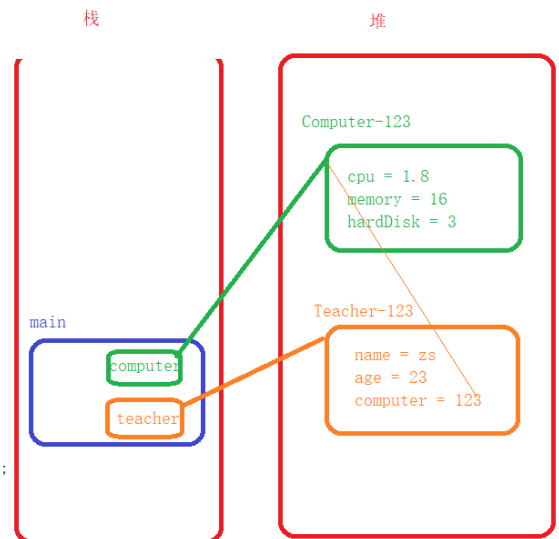
31 }
32 public class Test {
33     public static void main(String[] args) {
34         //1.创建电脑对象
35         Computer computer = new Computer(1.8,16,3);
36         //2.创建老师对象
37         Teacher teacher = new Teacher("zs",23,computer);
38         System.out.println(teacher.getDetails());
39     }
40 }

```

```

class Computer{
    private double cpu;
    private int memory;
    private int hardDisk;
    public Computer(double cpu, int memory, int hardDisk) {
        this.cpu = cpu;
        this.memory = memory;
        this.hardDisk = hardDisk;
    }
    public String getDetails(){
        return "cpu:" + cpu + "\t内存: " + memory + "\t硬盘: " + hardDisk;
    }
}
class Teacher {
    private String name;
    private int age;
    private Computer computer; //基本类型的变量 叫变量, 引用类型的变量 叫对象
    public Teacher(String name,int age,Computer computer){
        this.name = name;
        this.age = age;
        this.computer = computer;
    }
    public String getDetails(){
        return "姓名: " + name + "\t年龄: " + age + "\t电脑配置: " + computer.getDetails();
    }
}
public class Test {
    public static void main(String[] args) {
        //1.创建电脑对象
        Computer computer = new Computer(1.8,16,3);
        //2.创建老师对象
        Teacher teacher = new Teacher("zs",23,computer);
        System.out.println(teacher.getDetails());
    }
}

```



2. Object类的概述

代码块

```

1 package com.powernode.object12;
2
3 public class Test {
4     //说明继承
5     @Override
6     public String toString() {
7         return super.toString();
8     }
9 }

```

- @Override 说明有父类
- Object类是所有类的父类
- 每个类都默认继承了Object

代码块

```
1  public class Test {
2      //说明继承
3      @Override
4      public String toString() {
5          return super.toString();
6      }
7  }
8
9  等同于
10
11 public class Test extends Object {
12     //说明继承
13     @Override
14     public String toString() {
15         return super.toString();
16     }
17 }
```

3. toString方法

代码块

```
1  package com.powernode.object13;
2
3  class Cat extends Object{
4      private String name = "喵喵";
5      private int age = 2;
6
7      @Override
8      public String toString() {
9          return "Cat{" +
10              "name='" + name + '\'' +
11              ", age=" + age +
12              '}';
13      }
14  }
15  public class Test {
```

```

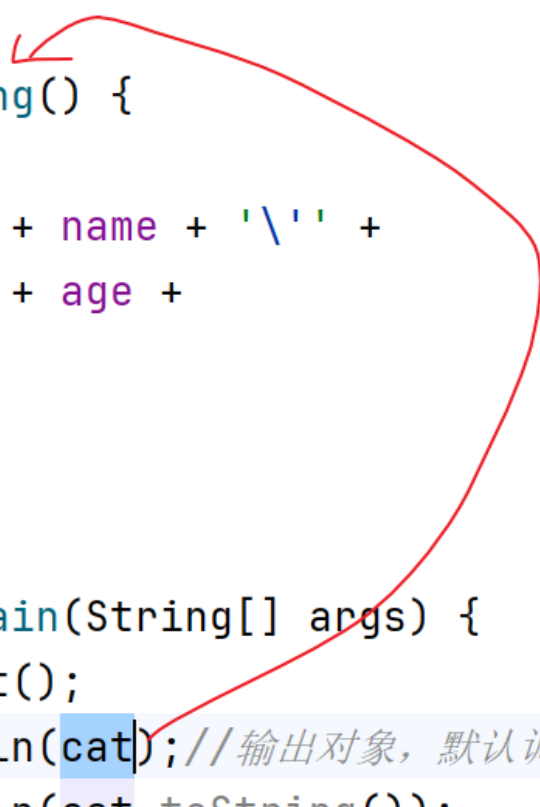
16     public static void main(String[] args) {
17         Cat cat = new Cat();
18         System.out.println(cat); //输出对象，默认调用了对象的toString方法
19         System.out.println(cat.toString());
20     }
21 }

```

```

@Override
public String toString() {
    return "Cat{" +
        "name='" + name + '\'' +
        ", age=" + age +
        '}';
}
}
public class Test {
    public static void main(String[] args) {
        Cat cat = new Cat();
        System.out.println(cat); //输出对象，默认调用了对象的toString方法
        System.out.println(cat.toString());
    }
}

```



97

4. clone方法(现阶段不要求掌握)

代码块

```

1     package com.powernode.object14;
2
3     class Cat implements Cloneable{
4         private String name = "喵喵";
5         private int age = 2;
6
7         @Override
8         public String toString() {
9             return "Cat{" +

```

```

10         "name='" + name + '\'' +
11         ", age=" + age +
12         '}'';
13     }
14
15     @Override
16     protected Object clone() throws CloneNotSupportedException {
17         return super.clone();
18     }
19 }
20
21 public class Test {
22     public static void main(String[] args) throws CloneNotSupportedException {
23         Cat cat = new Cat();
24         System.out.println(cat);
25         Cat newCat = (Cat) cat.clone();
26         System.out.println(newCat);
27
28     }
29
30 }

```

5. hashCode

代码块

```

1  package com.powernode.object15;
2
3  import java.util.Objects;
4
5  class Cat {
6      private String name;
7      private int age;
8
9      public Cat(String name, int age) {
10         this.name = name;
11         this.age = age;
12     }
13
14     @Override
15     public int hashCode() {
16         //根据name和age生成hash码, 如果两个对象的name和age相等, 那么hash码也相等
17         return Objects.hash(name, age);
18     }
19 }
20 public class Test {

```

```

21     public static void main(String[] args) {
22         Cat cat1 = new Cat("喵喵",3);
23         Cat cat2 = new Cat("喵喵",3);
24         System.out.println("cat1 = " + cat1.hashCode());
25         System.out.println("cat2 = " + cat2.hashCode());
26         /**
27          * 1.如果两个对象的内容相同,那么hashCode不一定相等
28          * 2.如果两个对象的hashCode相等,那么那么内容一定相等
29          */
30         //cat2 = cat1;
31         System.out.println(cat1);
32         System.out.println(cat2);
33         System.out.println(cat1 == cat2);//false
34     }
35 }

```

6. identityHashCode (了解)

代码块

```

1  package com.powernode.object16;
2
3  import java.util.Objects;
4
5  class Cat {
6      private String name;
7      private int age;
8
9      public Cat(String name, int age) {
10         this.name = name;
11         this.age = age;
12     }
13
14     @Override
15     public int hashCode() {
16         //根据name和age生成hashCode, 如果两个对象的name和age相等, 那么hashCode也相等
17         return Objects.hash(name,age);
18     }
19 }
20 public class Test {
21     public static void main(String[] args) {
22         Cat cat1 = new Cat("喵喵",3);
23         Cat cat2 = new Cat("喵喵",3);
24         System.out.println(cat1.hashCode());
25         System.out.println(cat2.hashCode());
26

```

```

27         //身份hashCode
28         System.out.println(System.identityHashCode(cat1));
29         System.out.println(System.identityHashCode(cat2));
30         //== 可以理解为比较的是 身份hashCode,重写了hashCode,身份hash不变
31         System.out.println(cat1 == cat2);
32     }
33 }

```

7. equals方法和内存分析

代码块

```

1     package com.powernode.object17;
2
3     class Cat {
4         private String name;
5         private int age;
6
7         public Cat(String name, int age) {
8             this.name = name;
9             this.age = age;
10        }
11
12        /**
13         * 1.比较两个对象的属性值是否相等
14         * 2.先找到两个对象
15         *     1.this:当前对象,谁调用了equals方法,谁就是当前对象,所以this = cat1
16         *     2.obj: Object obj = cat2
17         *
18         */
19        @Override
20        public boolean equals(Object obj) {
21            //this和obj的地址相等,说明指向堆中的同一个对象,内容一定相等
22            if (this == obj) return true;
23            //如果obj为null,那么this不为null,内容不具有比较性,直接返回false
24            // this的字节码文件对象 不等于 obj的字节码文件对象,说明就不是一个类,不具有可不
25            行
26            if (obj == null || this.getClass() != obj.getClass()) return false;
27            Cat cat2 = (Cat) obj;
28            return age == cat2.age && name.equals(cat2.name);//字符串比较内容是否相
29            等,使用equals
30            // return this.age == cat2.age && this.name.equals(cat2.name);//字符串比
31            较内容是否相等,使用equals
32        }
33    }
34
35    class Dog{}

```

```

32 public class Test {
33     public static void main(String[] args) {
34         Cat cat1 = new Cat("喵喵",3);
35         Cat cat2 = new Cat("喵喵",3);
36         //Dog dog = new Dog();
37         /**
38          * 1.比较两个对象的属性值是否相等
39          * 2.使用 == 无法比较
40          * 3.使用equals也无法比较, 因为Object类的equals方法比较的也是地址
41          * 4.父类的方法不满足子类的需求
42          * 5.重写equals方法, 在equals方法中比较属性值
43          *
44          */
45         System.out.println(cat1.equals(cat2));
46         /**
47          * public boolean equals(Object obj) {
48          *     return (this == obj);
49          * }
50          */
51     }
52 }

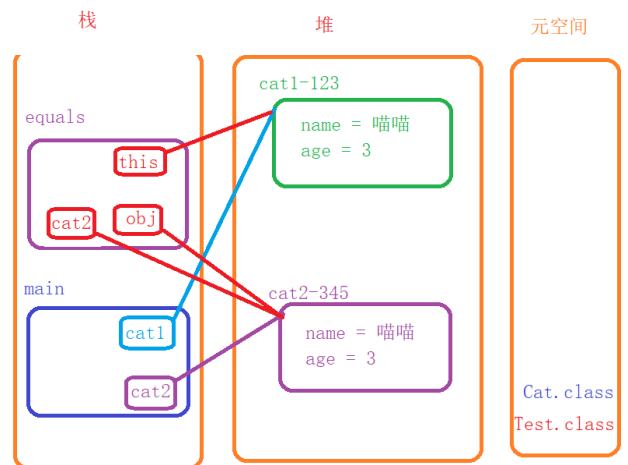
```

```

class Cat {
    private String name;
    private int age;
    public Cat(String name, int age) {
        this.name = name;
        this.age = age;
    }
    @Override
    public boolean equals(Object obj) {
        //this和obj的地址相等, 说明指向堆中的同一个对象, 内容一定相等
        if (this == obj) return true;
        if (obj == null || this.getClass() != obj.getClass()) return false;
        Cat cat2 = (Cat) obj;
        return age == cat2.age && name.equals(cat2.name);
    }
}

public class Test {
    public static void main(String[] args) {
        Cat cat1 = new Cat("喵喵",3);
        Cat cat2 = new Cat("喵喵",3);
        System.out.println(cat1.equals(cat2));
    }
}

```



作业

1:对象的关联

1. 编写Computer类, 其属性为cpu, memory, hardDisk, 提供构造器及相关方法, 以及say方法用于自我描述。
2. 编写Student类, Computer对象作为Student类的属性, 提供构造器及相关方法, 以及say方法用于自我描述。

3. 编写Test类，在main方法中创建Student对象，调用say方法打印输出结果。

2.toString

1.编写Student类，定义属性name、age和address

2.覆盖Object的toString方法用于自我描述

3.提供构造器为实例变量赋值。

4.在Test类的main方法中，创建 Student 对象，并打印该对象。

5.再打印调用toString方法的返回内容，比较两个输出结果是否相同。为什么？

3.equals方法

1.定义Dog类，定义属性name、age

2.编写Test类，在main方法中创建两个 Dog对象

3.比较两个对象是否equals （Object类的）

4.重写equals比较对象的内容是否相等

4.hashCode方法

1.在Dog类重写hashCode改变父类的hash规则，使用name和age属性生成hash值