第十讲--接口和Lambda表达式

任务目标

- ☑ 接口定义、继承和实现
- ☑ 接口默认的方法和静态方法
- ☑ Comparable接口和Comparator接口
- ☑ Lambda表达式

相关知识

1、接口≈抽象类

2、接口可以解决多继承的问题

3、使用Comparable接口和Comparator接口进行排序

1、接口

1、实现多个接口，继承一个父类

```java
abstract class Move
{
    public abstract void eat();
    public abstract void run();
    public abstract void climb();
}

interface Swim {
    public abstract void swim();
}

interface Fly {
    public abstract void fly();
}

public class Kitty extends Move implements Fly, Swim{

    public void eat()
    {System.out.print("Kitty eat");}
    public void run()
    {System.out.print("kitty run");}
    public void climb()
    {System.out.print("kitty climb");}
    public void fly()
    {
        System.out.print("kitty fly");
    }
    public void swim()
    {
        System.out.print("kitty swim");
    }
    public static void main(String[] args)
    {
        Kitty cat  = new Kitty();
        cat.eat();
        cat.fly();
        cat.swim();
    }
}
```

2、接口的静态方法

```
interface DogHeight{
    int height  =20;
    public static void getHeight()
    {
        System.out.print(height);
    }
}

public class DogHeightTest {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
       DogHeight.getHeight();
    }
}
```

3、接口的默认方法

```
interface DogA{
    int height  =20;
    public void run();
    public default void getHeight()
    {
        System.out.print(height);
    }
}

public class WangDog implements DogA{
    public void run()
    {
        System.out.print("run");
    }
}

public class DogHeightTest {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
       DogA w = new WangDog();
       w.getHeight();
       w.run();
    }
}
```

4、接口的继承

```
public interface Shape {

    public void getLength();
}


public interface Square extends Shape {

    public void getArea();
}


class SquareA implements Square{

  public void getLength()
  {
      System.out.print("4");
  }

  public void getArea()
  {
      System.out.print("20");
  }

}


public class SquareATest {
    public static void main(String[] args)
    {
        SquareA a1 = new SquareA();
        a1.getArea();
        a1.getLength();
    }

}
```

2、Comparable接口

1、JDK存在一些已定义的接口，例如Comparable接口

```
import java.util.Arrays;

class Circle implements Comparable<Circle>{
    private double radius;
    Circle(double r)
    {
        this.radius =r;
    }
    public double length()
    {
        return Math.PI*this.radius*2;
    }
    public double area()
    {
        return Math.PI*this.radius*this.radius;
    }
    @Override
    public int compareTo(Circle c)
    {
        if(this.area()<c.area())
        {
            return -1;
        }
        else
        {
            return 1;
        }
    }
}

public class CircleTest {
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Circle[] c1 = new Circle[3];
        c1[0] = new Circle(2.0);
        c1[1] = new Circle(3.0);
        c1[2] = new Circle(1.0);
        Arrays.sort(c1);
        for(Circle c: c1)
        {
            System.out.println(c.area());
        }
    }
}
```

3、Comparator接口

1、Comparator接口的定义和测试类分开的情况。

```
import java.util.Comparator;
class Rectangle implements Comparator<Rectangle>{

    private int w;
    private int h;
    Rectangle(int w, int h)
    {
        this.w=w;
        this.h=h;
    }
    public int area()
    {
        return this.w*this.h;
    }
    public int length()
    {
        return (this.w+this.h)*2;
    }
    public int compare(Rectangle c1, Rectangle c2)
    {
        return c1.area()-c2.area();
    }
}

import java.util.Arrays;
import java.util.Comparator;
public class RectangleTest {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Rectangle[] r1  = new Rectangle[3];
        r1[0] = new Rectangle(1,1);
        r1[1] = new Rectangle(2,2);
        r1[2] = new Rectangle(3,3);
        Arrays.sort(r1, new Rectangle(1,2)); //加一个Rectangle类
        for(Rectangle r: r1)
        {
            System.out.println(r.area());
        }
    }

}
```

2、重新定义compare()方法，在测试类中重新定义Compare(Rectangle c1, Rectangle c2)方法。

```
class Rectangle{

    private int w;
    private int h;
    Rectangle(int w, int h)
    {
        this.w=w;
        this.h=h;
    }
    public int area()
    {
        return this.w*this.h;
    }
    public int length()
    {
        return (this.w+this.h)*2;
    }
}

import java.util.Arrays;
import java.util.Comparator;
public class RectangleTest {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Rectangle[] r1  = new Rectangle[3];
        r1[0] = new Rectangle(1,1);
        r1[1] = new Rectangle(2,2);
        r1[2] = new Rectangle(3,3);
        Arrays.sort(r1,new Comparator<Rectangle>()
        {
    public int compare(Rectangle c1, Rectangle c2)
    {
        return c1.area()-c2.area();
    }
        });
        for(Rectangle r: r1)
        {
            System.out.println(r.area());
        }
    }

}
```

4、Lambda表达式

1、匿名内部类

```
interface Add
{
    int plus(int a, int b);
}
interface Subtract
{
    int minus(int a, int b);
}
interface Multiply
{
    int times(int a, int b);
}
interface Divide
{
    int divide(int a, int b);
}
public class Test82
{
    public static void main(String[] args)
    {
    Add op1 = new Add()
    {
        public int plus(int a, int b)
        {
            return a+b;
        }
    };
    Subtract op2 =  new Subtract()
    {
        public int minus(int a, int b)
        {
            return a-b;
        }
    };
    Multiply op3 = new Multiply()
    {
        public int times(int a, int b)
        {
            return a*b;
        }
    };
    Divide op4 = new Divide()
    {
        public int divide(int a, int b)
        {
            if (b!=0)
            {
                return a/b;
            }
            else
            {
                return 0;
            }
        }
    };
    System.out.println(op1.plus(3,4));
    System.out.println(op2.minus(3,4));
    System.out.println(op3.times(3,4));
    System.out.println(op4.divide(8,4));
    }
}
```

2、Lambda 表达式是 JDK8 的一个新特性，可以取代大部分的匿名内部类，写出更优雅的 Java 代码，尤其在集合的遍历和其他集合操作中，可以极大地优化代码结构。JDK 也提供了大量的内置函数式接口供我们使用，使得 Lambda 表达式的运用更加方便、高效。

使用 Lambda 表达式可以对某些接口进行简单的实现，但并不是所有的接口都可以使用 Lambda 表达式来实现。Lambda 规定接口中只能有一个需要被实现的方法，不是规定接口中只能有一个方法，JDK8 中有另一个新特性：default， 被 default 修饰的方法会有默认实现，不是必须被实现的方法，所以不影响 Lambda 表达式的使用。

Lambda的语法形式为 `() -> {}`，其中 `()` 用来描述参数列表，`{}` 用来描述方法体，`->` 为 lambda运算符 ，读作(goes to)。

```
@FunctionalInterface
interface Add
{
    int plus(int a, int b);
}
@FunctionalInterface
interface Subtract
{
    int minus(int a, int b);
}
@FunctionalInterface
interface Multiply
{
    int times(int a, int b);
}
@FunctionalInterface
interface Divide
{
    int divide(int a, int b);
}
public class Test82 {
    public static void main(String[] args)
    {
    Add op1 = (int a, int b) ->
        {
            return a+b;
        };
    Subtract op2 =  (int a, int b) ->
        {
            return a-b;
        };
    Multiply op3 = (int a, int b) ->
        {
            return a*b;
        };
    Divide op4 = (int a, int b) ->
        {
            if (b!=0)
            {
                return a/b;
            }
            else
            {
                return 0;
            }
        };
    System.out.println(op1.plus(3,4));
    System.out.println(op2.minus(3,4));
    System.out.println(op3.times(3,4));
    System.out.println(op4.divide(8,4));
    }

}
```

3、interface添加default 方法

```
@FunctionalInterface
interface Add
{
    int plus(int a, int b);
    default void print(int a)
    {
        System.out.println(a);
    }
}
@FunctionalInterface
interface Subtract
{
    int minus(int a, int b);
    default void print(int a)
    {
        System.out.println(a);
    }
}
@FunctionalInterface
interface Multiply
{
    int times(int a, int b);
    default void print(int a)
    {
        System.out.println(a);
    }
}
@FunctionalInterface
interface Divide
{
    int divide(int a, int b);
    default void print(int a)
    {
        System.out.println(a);
    }
}
public class Test82 {

    public static void main(String[] args)
    {
    Add op1 = (int a, int b) ->
        {
            return a+b;
        };
    Subtract op2 =  (int a, int b) ->
        {
            return a-b;
        };
    Multiply op3 = (int a, int b) ->
        {
            return a*b;
        };
    Divide op4 = (int a, int b) ->
        {
            if (b!=0)
            {
                return a/b;
            }
            else
            {
                return 0;
            }
        };
    op1.print(op1.plus(3,4));
    op2.print(op2.minus(3,4));
    op3.print(op3.times(3,4));
    op4.print(op4.divide(8,4));
    }

}
```

4、Lambda的多种类型（多参数无返回、无参无返回值、一个参数无返回、多个参数有返回值、无参有返回、一个参数有返回值）

```
/**多参数无返回*/
@FunctionalInterface
interface NoReturnMultiParam {
    void method(int a, int b);
}
/**无参无返回值*/
@FunctionalInterface
interface NoReturnNoParam {
    void method();
}
/**一个参数无返回*/
@FunctionalInterface
interface NoReturnOneParam {
    void method(int a);
}
/**多个参数有返回值*/
@FunctionalInterface
interface ReturnMultiParam {
    int method(int a, int b);
}
/*** 无参有返回*/
@FunctionalInterface
interface ReturnNoParam {
    int method();
}
/**一个参数有返回值*/
@FunctionalInterface
interface ReturnOneParam {
    int method(int a);
}

public class Test82
{
    public static void main(String[] args)
    {
        NoReturnMultiParam p1 = (a,b) ->
        {System.out.println(a+b);};
        p1.method(2,3);
        NoReturnNoParam p2 = () ->
        {System.out.println(12);};
        p2.method();
        NoReturnOneParam p3 = a ->
        {System.out.println(a);};
        p3.method(2);
        ReturnMultiParam p4 = (a,b) ->
        {return a+b;};
        System.out.println(p4.method(2,3));
        ReturnNoParam p5 = () ->
        {return 25;};
        System.out.println(p5.method());
        ReturnOneParam p6 = a ->
        { return 10+a;};
        System.out.println(p6.method(21));
    }
}
```

5、函数式接口

函数式接口（function interface）是指仅包含一个抽象方法的接口，因此也称为单抽象方法（SAM，Single Abstract Method）接口。

```
@FunctionalInterface
interface Converter<F, T>
{
T convert(F from);
}
//匿名内部类
public class ConverterTest {
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Converter<String, Integer> con = new Converter<String, Integer>()
            {
            public Integer convert(String from)
            {
                return Integer.valueOf(from);
            }
            };
            System.out.print(con.convert("1222"));
    }
}
//Lambda表达式
public class ConverterTest {
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Converter<String, Integer> con = (from) -> Integer.valueOf(from);
        System.out.print(con.convert("1222"));
    }
}
```

☑ 定义String排序

```
import java.util.Arrays;
public class StringTest {
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        String[] c1 = new String[3];
        c1[0] = "Hello";
        c1[1] = "Java";
        c1[2] = "program";
        Arrays.sort(c1,(String a, String b)->b.compareTo(a));
        for(String c: c1)
        {
            System.out.println(c);
        }
    }
}
```

☑ 定义Circle数组的排序

```
class Circle implements Comparator<Circle>{
    private double radius;
    Circle(double r)
    {
        this.radius =r;
    }
    public double length()
    {
        return Math.PI*this.radius*2;
    }
    public double area()
    {
        return Math.PI*this.radius*this.radius;
    }
    public int compare(Circle c,Circle d)
    {
        return c.area()-d.area();
    }
}
public class CircleTest {
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Circle[] c1 = new Circle[3];
        c1[0] = new Circle(2);
        c1[1] = new Circle(3);
        c1[2] = new Circle(1);
        Arrays.sort(c1,(Circle a, Circle b)->compare(a,b));
        for(Circle c: c1)
        {
            System.out.println(c.area());
        }
    }
}
```

☑ 定义Cumsum接口

```
interface Cumsum
{
    public int add(int a);
}
public class Test83
{
    public static void main(String[] args)
    {
        Cumsum cs = (int a) ->
            {
                int s = 0;
                for(int i=1;i<a;i++)
                {
                    s+=i;
                }
                return s;
            };

        System.out.print(cs.add(15)+"\n");
    }
}
```

☑ 预定义的函数式接口

函数式接口	说明
Function<T,R>	一个输入参数，一个返回值，返回值和输入参数类型可以不同
BiFunction<T,U,R>	两个输入参数，一个返回值，返回值和输入参数类型可以不同
UnaryOperator	一个输入参数，一个返回值，返回值和输入参数类型相同
BinaryOperator	两个输入参数，一个返回值，返回值和输入参数类型相同

☑ Function<T,R>接口定义了apply方法

```
interface Function<T,R>
{
    R apply(T a);
}
//上面是Function接口的原型，不需要定义
import java.util.function.Function;
public class Test84
{
    public static void main(String[] args)
    {
        Function<Integer,Integer> fc = (a) ->
                 {
            int s = 0;
            for(int i=1;i<a;i++)
            {
                s+=i;
            }
            return s;
        };
        System.out.print(fc.apply(15)+"\n");
    }
}
```

☑ BiFunction<T,U,R>接口定义了apply方法

```
import java.util.function.BiFunction;

public class Test84
{
    public static void main(String[] args)
    {
        BiFunction<Integer,Integer,Integer> fc = (a,b) ->
                 {
            int r = a%b;
            while(r!=0)
            {
                a = b;
                b = r;
                r = a%b;
            }
            return b;
        };
        System.out.print(fc.apply(18,48)+"\n");
    }
}
```

☑ UnaryOperator接口定义了apply方法

```
import java.util.function.UnaryOperator;

public class Test84
{
    public static void main(String[] args)
    {
        UnaryOperator<Integer> fc = (a) ->
                 {
            int s = 1;
            while(a>0)
            {
                s*=a;
                a--;
            }
            return s;
        };
        System.out.print(fc.apply(5)+"\n");
    }
}
```

☑ BinaryOperator接口定义了apply方法

```
import java.util.function.BinaryOperator;

public class Test84
{
    public static void main(String[] args)
    {
        BinaryOperator<Integer> fc = (a,b) ->
         {
                int r = a%b;
                while(r!=0)
                {
                    a = b;
                    b = r;
                    r = a%b;
                }
                return b;
            };
        System.out.print(fc.apply(18,48)+"\n");
    }
}
```