**每日作业卷答案**

**就业班JavaSE第12天**

传智播客.黑马程序员

# 基础题

### 练习一：函数式接口

1. 定义一个函数式接口CurrentTimePrinter,其中抽象方法void printCurrentTime()，使用注解@FunctionalInterface
2. 在测试类中定义static void showLongTime(CurrentTimePrinter timePrinter)，该方法的预期行为是使用timePrinter打印系统当前毫秒值
3. 测试showLongTime(),通过lambda表达式完成需求

答案

TimePrinter接口：

@FunctionalInterface  
public interface CurrentTimePrinter

{  
 void printCurrenTime();  
}

测试类：

public class Test01 {  
 public static void main(String[] args) {  
 showLongTime(()->System.out.println(System.currentTimeMillis()));  
 }  
  
 public static void showLongTime(CurrentTimePrinter timePrinter){  
 timePrinter.printCurrentTime();  
 }  
}

### 练习二：函数式接口

1. 定义一个函数式接口IntCalc,其中抽象方法int calc(int a , int b)，使用注解@FunctionalInterface
2. 在测试类中定义static void getProduct(int a , int b ,IntCalc calc), 该方法的预期行为是使用calc得到a和b的乘积并打印结果
3. 测试getProduct(),通过lambda表达式完成需求

答案

IntCalc接口：

@FunctionalInterface  
public interface IntCalc {  
 int calc(int a, int b);  
}

测试类：

public class Test02 {  
 public static void main(String[] args) {  
 getProduct(2,3,(a,b)->a\*b);  
 }  
 public static void getProduct(int a, int b, IntCalc intCalc){  
 int product = intCalc.calc(a,b);  
 System.out.println(product);  
  
 }  
}

### 练习三：静态方法引用

1. 定义一个函数式接口NumberToString,其中抽象方法String convert(int num)，使用注解@FunctionalInterface
2. 在测试类中定义static void decToHex(int num ,NumberToString nts), 该方法的预期行为是使用nts将一个十进制整数转换成十六进制表示的字符串，***tips:已知该行为与Integer类中的toHexString方法一致***
3. 测试decToHex (),使用方法引用完成需求

答案

NumberToString接口：

@FunctionalInterface  
public interface NumberToString {  
 String convert(int num);  
}

测试类：

public class Test03 {  
 public static void main(String[] args) {  
 decToHex(999, Integer::toHexString);  
 }  
 public static void decToHex(int num ,NumberToString nts){  
 String convert = nts.convert(num);  
 System.out.println(convert);  
 }  
}

### 练习四：成员方法引用、可变参数

1. 已知有随机数工具类如下：

import java.util.Random;  
public class RandomUtil {  
 private Random random = new Random();  
  
 public int nextInt(int... nums) {  
 if (nums == null || nums.length == 0) {  
 throw new RuntimeException("pls认真一点！");  
 }

//生成1到nums[0](包含1和nums[0])之间的随机数  
 if (nums.length == 1) return random.nextInt(nums[0]) + 1;

//生成nums[0]到nums[1](包含nums[0]和nums[1])之间的随机数  
 if (nums.length == 2) return random.nextInt(nums[1] + 1 - nums[0]) + nums[0];

//从数组nums中取出一个随机元素

return nums[random.nextInt(nums.length)];  
 }  
}

1. 定义一个函数式接口NumberSupplier,其中抽象方法int getNum(int...nums)，使用注解@FunctionalInterface
2. 在测试类中定义static void getRandomNumFromOneToNum(int num , NumberSupplier ns), 该方法的预期行为是使用ns得到一个从1到num(包含1和num)之间的随机数并打印
3. 在测试类中定义static void getRandomNumFromAToB(int a ,int b, NumberSupplier ns), 该方法的预期行为是使用ns得到一个从a到b(包含a和b)之间的随机数并打印
4. 在测试类中定义static void getRandomNumFromArray(int[] nums, NumberSupplier ns), 该方法的预期行为是使用ns从数组nums中获取一个随机元素并打印
5. 主方法中测试调用以上三个方法，使用方法引用完成需求

答案

NumberFactory接口：

@FunctionalInterface  
public interface NumberSupplier{  
 int getNum(int...nums);  
}

测试类：

public class Test04 {  
 public static void main(String[] args) {  
 RandomUtil randomUtil = new RandomUtil();  
 getRandomNumFromOneToNum(10, randomUtil::nextInt);  
 getRandomNumFromAToB(10, 20, randomUtil::nextInt);  
 int[] arr = {2, 4, 6, 8};  
 getRandomNumFromArray(arr, randomUtil::nextInt);  
 }  
  
 public static void getRandomNumFromOneToNum(int num, NumberSupplier ns) {  
 int result = ns.getNum(num);  
 System.out.println(result);  
  
 }  
  
 public static void getRandomNumFromAToB(int a, int b, NumberSupplier ns) {  
 int result = ns.getNum(a, b);  
 System.out.println(result);  
 }  
  
 public static void getRandomNumFromArray(int[] nums, NumberSupplier ns) {  
 int result = ns.getNum(nums);  
 System.out.println(result);  
 }  
  
}

### 练习五：super引用成员方法

按要求补全代码

1. 有函数式接口Helper如下

@FunctionalInterface  
public interface Helper {  
 void help();  
}

1. 有父类Father如下

public class Father {  
 public void askMoney(){  
 System.out.println("向家中的财政老大要钱购买家庭物资...");  
 }  
  
 public void spendMoney(){  
 System.out.println("去村口小卖部付钱给老板打酱油...");  
 }  
}

1. 有子类Son如下

public class Son extends Father {  
 @Override  
 public void askMoney() {  
 System.out.println("向家中的财政老大要钱交(上)学(网)费(吧)...");  
 }  
 @Override  
 public void spendMoney() {  
 System.out.println("去网吧付钱给网管开了一台机器...");  
 }  
 public void helpFather(Helper helper) {  
 helper.help();  
 }  
 //完成帮助Father打酱油的方法  
 public void daJiangYou() {  
 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;  
 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;  
 }  
}

1. 使用方法引用补全Son类中横线处的代码，帮助Father打酱油

答案

//完成帮助Father打酱油的方法  
public void daJiangYou() {  
 helpFather(super::askMoney);  
 helpFather(super::spendMoney);  
}

### 练习六：this引用成员方法

按要求补全代码

1. 有字符串包装StringWrapper接口如下

public interface StringWrapper {  
 String wrap(String str);  
}

1. 有Book类如下

public class Book {  
 //书名  
 private String name;  
 //出版社  
 private String publishingCompany;  
  
 public Book(String name, String publishingCompany) {  
 this.name = name;  
 this.publishingCompany = publishingCompany;  
 }  
  
 //bookMsg()拼接带书名号的书名  
 private String bookMsg(String bookname) {  
 return "《" + bookname + "》";  
 }  
  
 //publishingMsg()拼接出版信息  
 private String publishingMsg(String msg) {  
 return "【专柜正品<" + msg + ">原装绝版】";  
 }  
  
 //packSaleMsg()使用StringWrapper为上架销售信息包装  
 public String packSaleMsg(String msg, StringWrapper wrapper) {  
 return wrapper.wrap(msg);  
 }  
  
 //sale()上架销售  
 public void sale() {  
 //开始包装上架销售信息  
 String saleMsg = packSaleMsg( ) + packSaleMsg( );  
 System.out.println(saleMsg);

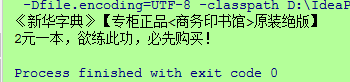
//开始售卖

System.out.println("2元一本，欲练此功，必先购买！");  
 }  
}

1. 有测试类如下

public class Test05 {  
 public static void main(String[] args) {  
 Book book = new Book("新华字典","商务印书馆");  
 book.sale();  
 }  
}

1. 填写Book类中空白横线处的代码，要求出现如下结果：



答案

String saleMsg = packSaleMsg(name, this::bookMsg) + packSaleMsg(publishingCompany, this::publishingMsg);

### 练习七：类的构造器引用

1. 有对象工厂BeanFactory接口如下

@FunctionalInterface  
public interface BeanFactory<T> {  
 T getBean();  
}

1. 有汽车类Car如下

public class Car {  
 private String brand;  
  
 public String getBrand() {  
 return brand;  
 }  
  
 public void setBrand(String brand) {  
 this.brand = brand;  
 }  
  
 public Car(String brand) {  
 this.brand = brand;  
 }  
  
 public Car() {  
  
 }  
}

1. 有测试类如下

public class Test06 {  
 public static void main(String[] args) {  
 Car Q7 = getCar(\_\_\_\_\_\_);  
 Q7.setBrand("奥迪");  
 }  
 public static Car getCar(\_\_\_\_\_\_\_\_\_\_\_\_){  
 return factory.getBean();  
 }  
}

1. 补全测试类中横线处的代码，完成原代码的需求

答案

public class Test06 {  
 public static void main(String[] args) {  
 Car Q7 = getCar(Car::new);  
 Q7.setBrand("奥迪");  
 }  
 public static Car getCar(BeanFactory<Car> factory){  
 return factory.getBean();  
 }  
}

### 练习八：数组构造器引用

1. 定义一个函数式接口ArrayBuilder<T>,提供带泛型的抽象方法T[] buildArray(int length)，使用注解@FunctionalInterface
2. 在测试类中定义static void getIntegerArray(int length , ArrayBuilder<Integer> builder), 该方法的预期行为是使用builder创建一个长度为length的Integer数组并打印其内存地址
3. 测试getIntegerArray (),使用方法引用完成需求

答案

ArrayBuilder接口：

@FunctionalInterface  
public interface ArrayBuilder<T> {  
 T[] buildArray(int length);  
}

测试类

public class Test07 {  
  
 public static void main(String[] args) {  
 getintarray(5, Integer[]::new);  
 }  
 private static void getintarray(int length,ArrayBuilder<Integer> builder){  
 Integer[] array = builder.buildArray(length);  
 System.out.println(array);  
  
 }  
}

### 练习九：Supplier接口使用

1. 给出测试类中代码如下：

import java.util.function.Supplier;  
  
public class Test01 {  
 public static <T> T getObj(Supplier<T> supplier) {  
 return supplier.get();  
 }  
  
 public static void main(String[] args) {  
   
 }  
}

1. 分别使用lambda表达式获得以下对象：
2. 长度为5的String数组
3. 包含5个1-20(含1和20)之间随机数的HashSet<Integer>集合
4. 一个代表2018年4月1日的Calendar对象

答案

import java.util.Calendar;  
import java.util.HashSet;  
import java.util.Random;  
import java.util.function.Supplier;  
  
public class Test01 {  
 public static <T> T getObj(Supplier<T> supplier) {  
 return supplier.get();  
 }  
  
 public static void main(String[] args) {  
 String[] obj1 = getObj(() -> new String[5]);  
  
 HashSet<Integer> obj2 = getObj(() -> {  
 Random random = new Random();  
 HashSet<Integer> integers = new HashSet<>();  
 while (integers.size() < 5) {  
 integers.add(random.nextInt(20) + 1);  
 }  
 return integers;  
 });  
  
 Calendar obj3 = getObj(() -> {  
 Calendar calendar = Calendar.getInstance();  
 calendar.set(Calendar.YEAR, 2018);  
 calendar.set(Calendar.MONTH, 3);  
 calendar.set(Calendar.DAY\_OF\_MONTH, 1);  
 return calendar;  
 });  
 System.out.println(obj1);  
 System.out.println(obj2);  
 System.out.println(obj3.getTime());  
 }  
}

### 练习十：Consumer接口使用

1. 给出测试类中代码如下

import java.util.HashMap;  
import java.util.function.Consumer;  
  
public class Test02 {  
 public static <T> void doJob(T t, Consumer<T> consumer) {  
 consumer.accept(t);  
 }  
 public static <T> void doJob(T t, Consumer<T> consumer1, Consumer<T> consumer2) {  
 consumer1.andThen(consumer2).accept(t);  
 }  
  
 public static void main(String[] args) {  
 HashMap<String, Integer> map = new HashMap<>();

//key:姓名 value:成绩  
 map.put("岑小村",59);  
 map.put("谷天洛",82);  
 map.put("渣渣辉",98);  
 map.put("蓝小月",65);

map.put("皮几万",70);  
   
 }  
}

1. 分别使用lambda表达式完成以下需求
2. 打印谷天洛的分数
3. 打印最高分
4. 分别以60分和70分为及格线，打印及格的人的名字(组合型消费)

答案

import java.util.Collection;  
import java.util.HashMap;  
import java.util.Map;  
import java.util.function.Consumer;  
  
public class Test02 {  
 public static <T> void doJob(T t, Consumer<T> consumer) {  
 consumer.accept(t);  
 }  
  
 public static <T> void doJob(T t, Consumer<T> consumer1, Consumer<T> consumer2) {  
 consumer1.andThen(consumer2).accept(t);  
 }  
  
 public static void main(String[] args) {  
 HashMap<String, Integer> map = new HashMap<>();  
  
 //key:姓名 value:成绩  
 map.put("岑小村", 59);  
 map.put("谷天洛", 82);  
 map.put("渣渣辉", 98);  
 map.put("蓝小月", 65);  
 map.put("皮几万", 70);  
  
 //打印谷天洛的分数  
 doJob(map, (m) -> System.out.println("谷天洛的分数：" + m.get("谷天洛")));  
  
 //打印最高分  
 doJob(map, (m) -> {  
 Collection<Integer> values = m.values();  
 Integer max = 0;  
 for (Integer value : values) {  
 if (value > max) {  
 max = value;  
 }  
 }  
 System.out.println("最高分：" + max);  
 });  
  
 //打印及格的人的名字  
 doJob(map, (m) -> {  
 System.out.print("60分及格的人：");  
 for (Map.Entry<String, Integer> entry : m.entrySet()) {  
 if (entry.getValue()>=60){  
 System.out.print(entry.getKey()+" ");  
 }  
 }  
 System.out.println();  
 }, (m) -> {  
 System.out.print("70分及格的人：");  
 for (Map.Entry<String, Integer> entry : m.entrySet()) {  
 if (entry.getValue()>=70){  
 System.out.print(entry.getKey()+" ");  
 }  
 }  
 System.out.println();  
 });  
 }  
}

# 扩展题

### 练习十一：综合练习

1. 定义学生类:
2. 成员变量 姓名：String name;
3. 成员变量 成绩：int score；
4. 无参及全参构造
5. 重写toString()
6. 学生信息与成绩如下：

|  |  |
| --- | --- |
| 姓名 | 数学 |
| 谢霆锋 | 85 |
| 章子怡 | 63 |
| 刘亦菲 | 77 |
| 黄晓明 | 33 |
| 岑小村 | 92 |

1. 在测试类中完成如下要求
2. 将五名学生添加到ArrayList集合
3. 使用Collections.sort(List<T> list, Comparator<? **super** T> c)方法将学生成绩从小到大进行排序（忽略非空判断）
4. 使用匿名内部类
5. 使用Lambda表达式
6. 使用方法引用 //tips:借助Comparator接口中静态方法comparingInt()方法

答案

Student类：

public class Student {  
 private String name;  
 private int score;  
 public String getName() {  
 return name;  
 }  
 public void setName(String name) {  
 this.name = name;  
 }  
 public int getScore() {  
 return score;  
 }  
 public void setScore(int score) {  
 this.score = score;  
 }  
 public Student() {  
 }  
 public Student(String name, int score) {  
 this.name = name;  
 this.score = score;  
 }  
 @Override  
 public String toString() {  
 return "Student{" +  
 "name='" + name + '\'' +  
 ", score=" + score +  
 '}';  
 }  
}

测试类：

import java.util.ArrayList;  
import java.util.Collections;  
import java.util.Comparator;  
  
public class Test08 {  
 public static void main(String[] args) {  
 ArrayList<Student> list = new ArrayList<>();  
 list.add(new Student("谢霆锋", 85));  
 list.add(new Student("章子怡", 63));  
 list.add(new Student("刘亦菲", 77));  
 list.add(new Student("黄晓明", 33));  
 list.add(new Student("岑小村", 92));  
  
 //使用匿名内部类将成绩从小到大排序  
 /\*Collections.sort(list, new Comparator<Student>() {  
 @Override  
 public int compare(Student o1, Student o2) {  
 // 忽略非空判断  
 return o1.getScore() - o2.getScore();  
 }  
 });\*/  
  
 //使用Lambda为英语成绩从小到大排序  
 //Collections.sort(list,(o1, o2) -> o1.getScore()-o2.getScore());

//借助comparingInt()使用方法引用 Collections.*sort*(list, Comparator.*comparingInt*(Student::getScore));

System.out.println(list);  
   
 }  
}