Java第一阶段—DAY12-JAVA案例

1. 有接口如下，请使用lambda表达式对其进行实现。

interface Drinkable{  
 void drink();  
}

**public class** Demo02Lambda {  
  
 **public static void** main(String[] args) {  
 Drinkable drinkable = ()->{  
 System.out.println(**"大口的喝..."**);  
 };  
  
 drinkable.drink();  
 }  
}

1. 有函数接口如下，请使用lambda表达式对其进行实现。

@FunctionalInterface  
interface Eatable{  
 String eat(String food);  
}

**public class** Demo03Lambda {  
  
 **public static void** main(String[] args) {  
 Eatable eatable = (food)->{  
 System.out.println(**"大口的吃"**+food);  
 **return "吃饱了"**;  
 };  
  
 String result = eatable.eat(**"牛肉"**);  
 System.out.println(result);  
  
 }  
}

1. 练习常用的函数接口。

public class Demo04Lambda {  
  
 public static void main(String[] args) {  
*// 随机产生一个整数* Supplier<Integer> supplier = ()->{  
 return new Random().nextInt();  
 };  
  
 Integer result = supplier.get();  
 System.out.println("随机产生一个整数： "+result);  
  
*// 小写字符串转成大写字符串* Consumer<String> consumer = (s)->{  
 System.out.println(s.toUpperCase());  
 };  
 consumer.accept("hello blb!");  
  
*// 将字符串转成数字* Function<String,Integer> function = (s)->{  
 return Integer.parseInt(s);  
 };  
 System.out.println(function.apply("123"));  
  
*// 判断一个整数是否是偶数* Predicate<Integer> predicate = (i)->{  
 return i%2==0;  
 };  
 System.out.println("是否是整数："+predicate.test(18));  
 }  
}

1. 定义接口Swimmable，定一个静态方法swimming方法，直接通过接口调用其实现。

public class Demo06InterfaceEnhance {  
  
 public static void main(String[] args) {  
 Swimmable.swimming();  
 }  
}  
  
interface Swimmable{  
  
 public static void swimming(){  
 System.out.println("游泳");  
 }  
}

1. 练习函数引用的4种场景。

public class Demo08FunctionReference {  
  
 public static void main(String[] args) {  
 }  
  
 public void testMethod1(){  
 Consumer<Object> c = System.out::println;  
 c.accept("bailiban");  
 c.accept(123);  
 }  
  
 public void testMethod2(){  
 Function<Integer, String> function = String::valueOf ;  
 System.out.println(function.apply(123));  
 }  
  
 public void testMethod3(){  
 Function<String,Integer> function = String::length;  
 System.out.println(function.apply("bailiban"));  
 }  
  
 @Test  
 public void testMethod4(){  
 Function<String,Student> function = Student::new;  
 Student blb = function.apply("blb");  
 System.out.println(blb);  
 }  
  
}  
  
class Student {  
  
 private String name ;  
  
 public Student(String name){  
 this.name = name ;  
 }  
  
 @Override  
 public String toString() {  
 return "Student{" +  
 "name='" + name + '\'' +  
 '}';  
 }  
}

1. 创建List集合，放入元素"刘备","张飞","赵云","诸葛亮","黄忠","黄月英"。通过stream流过滤出姓黄的元素。

public class Demo09Stream {  
  
 public static void main(String[] args) {  
*// 创建集合并添加元素* List<String> list = new ArrayList<>();  
 Collections.addAll(list,"刘备","张飞","赵云","诸葛亮","黄忠","黄月英");  
*// 通过流过滤姓名以“黄”开头的元素，然后遍历* list.stream().filter((s)->{  
 return s.startsWith("黄");  
 }).forEach((s)->{  
 System.out.println(s);  
 });  
  
 }  
}

1. 练习集合中stream流的相关API。

public class Demo10Stream {  
  
*// 获取Stream流对象* @Test  
 public void testStream01(){  
*// List获取Stream* List<String> list = new ArrayList<>();  
 Stream<String> stream = list.stream();  
  
*// Set获取Stream* Set<String> set = new HashSet<>();  
 Stream<String> stream2 = set.stream();  
  
*// Map获取Stream* Map<String, String> map = new HashMap<>();  
 Stream<String> keyStream = map.keySet().stream();  
 Stream<String> valueStream = map.values().stream();  
 Stream<Map.Entry<String, String>> entryStream = map.entrySet().stream();  
  
*// of静态方法获取Stream流* Stream<String> stringStream = Stream.of("刘备","张飞","赵云","诸葛亮","黄忠","黄月英");  
 }  
  
*// foreach* @Test  
 public void testStream02(){  
 Stream<String> stream = Stream.of("刘备","张飞","赵云","诸葛亮","黄忠","黄月英");  
*// stream.forEach(s->{  
// System.out.println(s);  
// });* stream.forEach(System.out::println);  
 }  
  
*// count* @Test  
 public void testStream03(){  
 Stream<String> stream = Stream.of("刘备","张飞","赵云","诸葛亮","黄忠","黄月英");  
 System.out.println(stream.count());  
 }  
  
 *// filter* @Test  
 public void testStream04(){  
 Stream<String> stream = Stream.of("刘备","张飞","赵云","诸葛亮","黄忠","黄月英");  
 stream.filter(s-> s.startsWith("黄")).forEach(System.out::println);  
 }  
  
 *// limit* @Test  
 public void testStream05(){  
 Stream<String> stream = Stream.of("刘备","张飞","赵云","诸葛亮","黄忠","黄月英");  
 stream.limit(3).forEach(System.out::println);  
 }  
  
*// skip* @Test  
 public void testStream06(){  
 Stream<String> stream = Stream.of("刘备","张飞","赵云","诸葛亮","黄忠","黄月英");  
 stream.skip(3).forEach(System.out::println);  
 }  
  
*// map* @Test  
 public void testStream07(){  
 Stream<String> stream = Stream.of("1","2","3","4","5","6");  
 stream.map(s->Integer.parseInt(s)).forEach(System.out::println);  
 }  
  
*// sorted* @Test  
 public void testStream08(){  
 Stream<Integer> stream = Stream.of(9,4,1,2,8);  
*// stream.sorted().forEach(System.out::println);* stream.sorted((o1,o2)->o2-o1).forEach(System.out::println);  
  
 }  
  
 *//distanct* @Test  
 public void testStream09(){  
 Stream<String> stream = Stream.of("刘备","张飞","赵云","诸葛亮","张飞","赵云");  
 stream.distinct().forEach(System.out::println);  
 }  
  
 @Test  
 public void testStream10(){  
 Stream<String> stream = Stream.of("刘备","张飞","赵云","诸葛亮","张飞","赵云");  
*// 用List集合来收集  
// List<String> list = stream.collect(Collectors.toList());  
// 用Set集合来收集  
// Set<String> set = stream.collect(Collectors.toSet());  
// System.out.println(set);  
  
// 用ArrayList集合来收集* ArrayList<String> arrayList = stream.collect(Collectors.toCollection(ArrayList::new));  
 System.out.println(arrayList);  
 }  
}

1. 练习LocalDate、LocalTime类跟LocalDateTime类的相关API。

public class Demo11LocalDateTime {  
  
 @Test  
 public void dateTime01(){  
*// 获取当前对应的日期* LocalDate now = LocalDate.now();  
 System.out.println(now);  
  
 LocalDate date = LocalDate.of(2020, 12, 13);  
 System.out.println(date);  
*// 获取年份* System.out.println(now.getYear());  
*// 获取月份，英文* System.out.println(now.getMonth());  
*// 获取月份值* System.out.println(now.getMonthValue());  
*// 获取当月中的第几天，也就是几号* System.out.println(now.getDayOfMonth());  
*// 获取当周中的第几天，也就是星期* System.out.println(now.getDayOfWeek());  
*// 获取年中的第几天* System.out.println(now.getDayOfYear());  
  
*// 修改年份为2019* System.out.println(now.withYear(2019));  
*// 修改月份为2* System.out.println(now.withMonth(2));  
*// 修改日期为3号* System.out.println(now.withDayOfMonth(3));  
  
 }  
  
 @Test  
 public void dateTime02(){  
 LocalTime time = LocalTime.of(11,12,13);  
 System.out.println(time);  
*// 获取当前时间* LocalTime now = LocalTime.now();  
*// 获取小时* System.out.println(now.getHour());  
*// 获取分* System.out.println(now.getMinute());  
*// 获取秒* System.out.println(now.getSecond());  
*// 获取纳秒* System.out.println(now.getNano());  
*// 修改小时为12* System.out.println(now.withHour(12));  
*// 修改分钟为22* System.out.println(now.withMinute(22));  
*// 修改秒为33* System.out.println(now.withSecond(33));  
 }  
  
 @Test  
 public void dateTime03(){  
 LocalDateTime time = LocalDateTime.of(2020,12,11,10,11,12);  
 System.out.println(time);  
  
 LocalDateTime now = LocalDateTime.now();  
 System.out.println(now);  
  
 *// 获取年份* System.out.println(now.getYear());  
*// 获取月份，英文* System.out.println(now.getMonth());  
*// 获取月份值* System.out.println(now.getMonthValue());  
*// 获取当月中的第几天，也就是几号* System.out.println(now.getDayOfMonth());  
*// 获取当周中的第几天，也就是星期* System.out.println(now.getDayOfWeek());  
*// 获取年中的第几天* System.out.println(now.getDayOfYear());  
*// 获取小时* System.out.println(now.getHour());  
*// 获取分* System.out.println(now.getMinute());  
*// 获取秒* System.out.println(now.getSecond());  
*// 获取纳秒* System.out.println(now.getNano());  
*// 修改年份为2019* System.out.println(now.withYear(2019));  
*// 修改月份为2* System.out.println(now.withMonth(2));  
*// 修改日期为3号* System.out.println(now.withDayOfMonth(3));  
*// 修改小时为12* System.out.println(now.withHour(12));  
*// 修改分钟为22* System.out.println(now.withMinute(22));  
*// 修改秒为33* System.out.println(now.withSecond(33));  
 }  
  
 @Test  
 public void dateTime04(){  
*// 日期转成格式化字符串* LocalDateTime now = LocalDateTime.now();  
 DateTimeFormatter dtf = DateTimeFormatter.ofPattern("yyyy年MM月dd日 HH时mm分ss秒");  
 String formatNow = now.format(dtf);  
 System.out.println(formatNow);  
  
*// 格式化字符串转成日期* LocalDateTime parse = LocalDateTime.parse("2021年01月28日 04时14分43秒", dtf);  
 System.out.println(parse);  
 }  
  
 @Test  
 public void dateTime05(){  
 LocalTime now = LocalTime.now();  
 System.out.println(now);  
 LocalTime time = LocalTime.of(17,48,12);  
 System.out.println(time);  
  
 Duration duration = Duration.between(now,time);  
*// 时间差转成小时* System.out.println(duration.toHours());  
*// 时间差转成分钟* System.out.println(duration.toMinutes());  
*// 时间差转成秒* System.out.println(duration.getSeconds());  
*// 时间差转成纳秒* System.out.println(duration.toNanos());  
  
 }  
  
 @Test  
 public void dateTime06(){  
 LocalDate now = LocalDate.now();  
 LocalDate date = LocalDate.of(2030,4,30);  
 Period period = Period.between(now,date);  
*// 获取年份差* System.out.println(period.getYears());  
*// 获取月份差* System.out.println(period.getMonths());  
*// 获取日期差* System.out.println(period.getDays());  
 }  
  
 @Test  
 public void dateTime07(){  
 LocalDateTime now = LocalDateTime.now();  
*// 设置下月1号的校正器* TemporalAdjuster firstDayOfNextMonth = temporal -> {  
 LocalDateTime time = (LocalDateTime) temporal;  
 return time.plusMonths(1).withDayOfMonth(1);  
 };  
*// 通过校正器调节now的值* System.out.println(now.with(firstDayOfNextMonth));  
  
*// TemporalAdjusters获取下一年的第1天的校正器* System.out.println(now.with(TemporalAdjusters.firstDayOfNextYear()));  
  
 }  
  
}

1. 练习重复注解跟类型注解的API。

*// 重复注解*@MyAnnotation("hello")  
@MyAnnotation("world")  
public class Demo12Annotation {  
  
 public static void main(String[] args) {  
 MyAnnotation[] annotations = Demo12.class.getAnnotationsByType(MyAnnotation.class);  
 for (MyAnnotation anno : annotations){  
 System.out.println(anno.value());  
 }  
  
 }  
}  
  
*//定义注解容器*@Target({ElementType.FIELD,ElementType.METHOD,ElementType.TYPE})  
@Retention(RetentionPolicy.RUNTIME)  
@interface MyAnnotations{  
 MyAnnotation [] value() ;  
}  
  
*// 定义注解内容*@Target({ElementType.FIELD,ElementType.METHOD,ElementType.TYPE})  
@Retention(RetentionPolicy.RUNTIME)  
@Repeatable(MyAnnotations.class)  
@interface MyAnnotation{  
 String value() default "blb" ;  
}

*// 修饰泛型类***public class** Demo13Annotation< @TypeParam T> {  
*// 在任意使用类型的地方使用，包括基本数据类型* **public** @TypeUse **int** i = 1 ;  
  
 *// 修饰泛型方法* **public** < @TypeParam E> **void** testTypeParameter(){  
 *// 在任意使用类型的地方使用，包括局部变量* @TypeUse **int** i ;  
 }  
}  
  
*//定义泛型为TYPE\_USE*@Target(ElementType.TYPE\_USE)  
@**interface** TypeUse{  
 String value() **default "blb"**;  
}  
  
*//定义泛型为TYPE\_PARAMETER*@Target(ElementType.TYPE\_PARAMETER)  
@**interface** TypeParam{  
 String value() **default "blb"**;  
}