一、函数式编程

根据实际场景，代码的多个改进版本——>购物车实例，lambda的演化

（1）创建Sku实例 ——定义商品类型枚举类型

（2）创建购物车服务类——可以获取商品列表信息

【CartService】

（3）第一个需求——（购物车服务）获取所有的电子产品类商品 version 1.0.0

【业务硬编码】（ctrl -y整行删除 ）

建立测试类，并测试这个需求Version1Test

1，加上@Test注解，他就是一个单独的测试类了

2，可以设置为静态方法，否则还需要实例化

3，导入阿里的fastjson包，调用JSON.toJSONString方法，让输出结果更美观

（4）第二个需求，要看某特定商品类型里面有哪些商品（传入参数）

【单一维度判断条件】

（5）第三个需求，支持通过商品类型或总价来过滤商品

【多维度判断条件】

（6）第四个需求：如何根据更多的条件来过滤商品？

【判断逻辑参数化】【实体类作为参数】

1，对**选择标准**进行建模，在更上层进行抽象，**谓词**，test方法返回一个布尔值判断是否符合条件。建立一个接口，【SkuPredicate】。

【TODO：问题，接口也能Override了？】

2，然后建立方法第二个参数传入SkuPredicate，对Sku的不同操作行为，根据传入的标准【即不同的Sku判断策略，Predicate的具体实现】来执行业务逻辑

3，建立具体的Sku标准，实现Predicate

【SkuBooksCategoryPredicate，SkuTotalPricePredicate】

（7）针对version4，每个过滤条件都需要一个实例类，如果只使用一次，是不是不用编写很多**多余的**Predicate的实现类了。——》匿名内部类 Version5Test

【匿名类作为参数】

（8）比匿名内部类更好用的—— Lambda表达式 Version6Test【与version5类似】

【lambda表示作为参数】

注意设置一下language level

Name: 
Stream996 
Stream996 
Project Settings 
Project 
Libraries 
Facets 
Artifacts 
Platform Settings 
SDKs 
Global Libraries 
Problems 
Sources Paths Dependencies 
Language level: 8 - Lambdas, type annotations etc. 
Mark as: • Sources — Jests Resources •e Test Resources 
V D:\ideaProject\mooc\Stream996 
target 
Excluded 
+ Add Content Root 
D:\ideaProject\moo 
Source Folders 
src\main\java 
Test Source Folders 
src\test\java 
Resource Folders 
src\main\resources 
Excluded Folders 
target 

二、Lambda表达式特点

（1）Java8 引入**函数式编程**风格（函数可以赋值给变量，作为参数或者返回值进行传递，而不单单只面向对象）

（2）可以理解为一种匿名函数的代替

（3）行为参数化传递

三、Lambda表达式的形式【参考其他文档】

（1）（parameters) -> expression

（2）（parameters) -> { statement;}

可选的类型声明；可选的参数。没有参数，小括号不能省，为 () ；只有一个参数可以省略括号；多个statement要加大括号{}；也可以对参数进行显式声明（Long x，Long y）

四、什么是函数式接口？

（1）接口中只有一个抽象方法（可以有多个非抽象方法）【SkuPredicate】

（2）Java8的函数式接口注解：@FuncitonInterface【仅作为标注，不必须】

（3）函数式接口可以被隐式转换为 lambda 表达式。

五、自定义函数式接口

【函数式接口都可以用lambda表达式代替？——好像是，似乎是】

实现读取本地文件后自定义处理逻辑功能（仅为理解函数式接口）：

（1）创建文件处理式函数式接口 FileConsumer

（2）创建文件服务类 FileService

（3）编写测试类进行测试

/当然还可以进行更复杂的操作，比如统计词频、检查错词什么的，这里仅是一个函数式接口的举例

六、常用的函数式接口及使用【到处都懵懵懂懂。。】

泛化的接口（可以直接使用lambda代替） util——function文件夹下面【可以到具体的文件下去看】

如何Funciton有两个方法参数T和R，其中的apply方法接受T参数，返回R参数

常 用 函 数 接 口 及 使 用 
接 囗 
Predicate<T> 
Consume <T> 
Function T, R> 
Supplier<T> 
UnaryOperator<T> 
BinaryOperator<T> 
参 数 返 回 类 型 
T 
T 
T 
None 
T 
boolean 
void 
描 述 
一 比 如 求 一 个 人 是 否 为 男 性 
用 接 收 一 个 对 象 进 行 处 理 但 没 有 返 回 ， 比 如 
接 收 | 
， 子 
转 换 一 个 对 象 为 不 同 类 型 的 、 
提 供 一 个 对 象 
接 收 对 象 并 返 回 同 类 型 的 对 象 
接 收 两 个 同 类 型 的 对 象 ， 并 返 回 一 个 原 类 型 对 

七、方法引用

调用特定方法的Lambda表达式的一种快捷写法，可以让你重复使用现有的方法定义，并像Lambda表达式一样传递他们。

Sku 
getSkuPrice 

（1）指向静态方法的方法引用 Integer：： parseInt

* (args) —> ClassName. staticMethod (args) ; 
* ClassName: : staticMethod; 
testl() { 
public void 
Consumer<String> 
consumerl 
= (String number) 
consumer2 
Consumer<String> 
= Integer: : parselnt; 
mooc 

（2）指向任意类型的实例方法的方法引用 String：：length

* (args) —> args . instanceMethod ( ) ; 
* ClassName: : instanceMethod; 
public void 
tes t2 
Consumer<String> 
consumerl 
= (String str) 
Consumer<String> 
consumer2 
= String: : length; 

（3）指向现有对象的实例方法的方法引用 stringBuilder：：apend

* (args) —> object. instanceMethod (args) ; 
* object: : instanceMethod; 
test3 
public void 
StringBui1der 
= new StringBui1der ( ) • 
consumerl 
Consumer<String> 
= (String str) 
Consumer<String> 
consumer2 
= stringBui1der: : append; 

八、Stream流

（1）新建StreamVs（对比原始集合操作与Stream+lambda集合操作）

购物车案例要求

\*1，先看看购物车中都有什么东西

\*2，图书类商品都给买（自我提升作用）

\*3，其余的商品买两件最贵的

\*4，最后只要这两件商品名字和总价

原始集合操作当然可以优化，但是一般来说Stream能带来更大的便利性和优雅性

（2）流Stream

1，JDK1.8引入的新成员，以声明式方式（配合lambda）处理集合数据

2，将基础操作连接起来，完成复杂的数据处理流水线

3，提供透明的并行处理【不要问这是什么意思，我也不懂。。】

定义：从支持数据处理操作（类似数据库的操作）的源生成的元素序列（类似集合）。

——java8实战

（3）流与集合的区别（自己去找）

集合遍历多次，流只能遍历一次；集合是外部迭代，流是内部迭代等等

（4）流的组成

三部分：数据源——中间操作——终端操作

举例：cart ——filter，sorted，map——collect

（5）流操作分类

( Intermediate ) 
( Terminal ) 
fil ter/map/peek$ 
dic tin c t/ sorted/ I imi 
forEach/c011ect/count$ 
anyMa tch/ f indFirs t/ f indAny$ 

流的使用

( filter ) 
( map ) 
( flatMap ) 
( peek ) 
( distinct ) 
( skip ) 
( limit ) 
( sorted ) 
( noneMatch ) 
( forEach ) 
IBÉ5 ( reduce ) 
( max ) 
( collect ) 
( min ) 
( count ) 

* 中间操作，dictinct，sorted建立在所有数据上，就是**有状态操作**；filter，map只要符合某个条件就进行过滤，不需要建立在所有数据上，就是**无状态操作**
* 终端操作，**非短路**：每个数据都要遍历一次；**短路**：只要找到就不往后执行了

（6）流操作代码举例

【StreamOperator】

1，**filter**：点击去看发现参数需要一个Predicate函数式接口，再点开Predicate，发现里面有一个test方法\_(:з」∠)\_ 。

2，**map**：点进去看发现需要传入一个Function函数接口，点开Function发现apply方法是将一个T型参数转换为R型参数。

3，**peek**：peek与forEach类似，都是遍历查看每一个元素，区别是peek是中间操作，查看完流还可用；forEach是终端操作，查看完之后流就不可用了；接受一个Consumer

注意代码：peek和forEach是交替执行的；

4，**sort**：无参就是自然排序；有参可以接受一个Comparator

注意代码：添加了sorted操作之后（有状态操作），peek就是先遍历forEach是后遍历了

5，**skip，limit可实现分页功能**

6，终端操作比较简单

（7）流的构建

1，由值创建流

Stream.of(1,2,3,4,5);

2，由数组创建流

Arrays.stream(numbers);

3，由文件生成流

Files.lines(Paths.get(url));

4，由函数生成流（可以是无限流）

Stream.iterate(0,n->n+2);

Stream.generate(Math::random);

【StreamConstructor类】

（8）收集器——创建好了流，如何把结果收集起来。

1，将流中的元素累积成一个结果

2，作用于终端操作collect()上

3，注意区分：collect 方法/ Collector / Collectors

【终端操作 / 接口 / 工具类（直接用）】

collect方法接受一个Collector接口的实现类，像是groupingBy和toList方法返回的都是Collector

（9）预定义收集器的功能Collectros

【StreamCollector类】

1，将流元素归约和汇总成一个值

.collect(Collectors.toList());

2，将流元素分组

返回一个Map，key就是条件，value是相应条件下元素的集合

.collect(Collectors.groupingBy(Sku::getSkuCategory));

3，将流元素分区

分区，是分组的特殊情况，由一个谓词作为分区的函数（返回一个布尔值），将数据分为两组，布尔值为true时和为false时。

.collect(Collectors.partitioningBy(sku->sku.getTotalPrice()>100));

总结：注意lambda与流的区别，这是两个不同的概念，lambda就是一种表达式，我自己理解是一种语法；流就是处理集合、处理数据的一种工具。有了集合，数组或文件什么的创建了流，流的中间方法或者终端方法需要的参数一般是函数式接口，这时候可以把lambda传进去，这是我的理解。