package LambdaPractice;

import java.util.Arrays;

import java.util.List;

public class TekClass {

// Javada Lambda fonksiyonları sadece Fonksiyonel interface'ler ile kullanılabilir.

// Sadece bir adet abstract metodu olan interface'lere fonksiyonel interface denilir.

// Kullanıcı isterse kendi fonksiyonel interface'ini yazabilir. Ama Javada hali hazırda

// Consumer, Function, Supplier, Predicate gibi tanımlanmış interfaceler bulunmaktadır.

// Ayrıca bu interface'leri parametre olarak alabilen High Order Function (HOF)'lar da bulunmaktadır.

// foreEach(), map(), filter(), reduce() gibi HOF'lar fonksiyonel interfaceler ile çalıştığı için

// içerisinde Lambda fonksiyonları yazmak mümkündür.

// Javada genelde bu gibi fonksiyonlar üzerinden Lambda ifadeleri kullanılmaktadır.

/\*

ForEach:

Stream içindeki veriyi tek tek tüketmekiçin oluşturulmuş bir yapıdır.

\*\*Terminal işlemi olduğundan dolayı stream artık tüketilmiş olarak kabul edilir ve

artık kullanılamayacağı anlamına gelir.

\*/

///==========================================

/\*

Filter

Dizimiz veya Collection’ımız üzerinde bizim belirteceğimiz koşullar doğrultusunda

filtreleme işlemi yapmamıza yarayan bir yapıdır.

Bu işlem sonrası belirttiğimiz koşula uygun olmayan elemanlar bir sonraki aşamada kullanılamazlar.

Artık elimizde filtrelenmiş/süzülmüş/elenmiş bir veri vardır

\*/

//===========================================

/\*

Collect

Collect metodu Stream türündeki nesneleri başka biçimdeki nesneye,

veri yapısına dönüştürmek için kullanılır.

\*/

//==========================

/\*

Map :Stream içerisinde erişilen her bir eleman üzerinde işlem yapmamıza

ve başka elemanlara dönüştürmemize imkan sağlayan

\*/

///==========================

/\*

Distinct

Stream içerisinde bulunan ve tekrar eden elemanlar varsa

bunları distinct metodu ile çıkartabilir ve

elimizde tekrar etmeyen birbirinden farklı

elemanlar barından bir veri setimiz kalır

\*/

//===========================================

/\*

Sorted

Bazı durumlarda elimizde ki veri setini belirli bir parametreye göre sıralamak isteyebiliriz.

2 türü bulunmaktadır. Birinci hali parametre almaz küçükten büyüğe şekilde sıralar.

İkinci formatı ise Comparator arayüzünden türediği için

bizim belirlemiş olduğumuz parametreye göre sıralama işlemi yapar.

\*/

////======================

/\*

reduce : indirgeme demektir.

kullanımı; elemanları teker teker işler. Bir önceki adımda elde edilen sonuç,

bir sonraki elemanla işlemle tutulur

\*/

//=====================

/\*

peek () 'in Javadoc sayfası şöyle diyor: " Bu yöntem, temelde, öğeleri bir

ardışık düzen içinde belirli bir noktadan geçerken görmek istediğiniz

yerde hata ayıklamayı desteklemek için vardır .

\*/

//===============================

/\*

AnyMath: Vereceğimiz şarta bağlı olarak Stream içerisinde gezinir ve

herhangi bir elemanla eşleşme durumunda true dönecektir.

\*/

//========================

/\*

AllMatch: Verilen şarta göre Stream içerisindeki tüm elemanların

bu şarta uyması durumunda true dönecektir.

\*/

//===========================

/\*

NoneMatch: Şarta göre Stream içindeki hiç bir

elemanın şartı sağlamaması durumunda true dönecektir.

\*/

//====================

/\*

Limit

Elimizde var olan bir veri kaynağını, göstermek istediğimiz kadarını göstermemize yarayan,

sınırlandıran bir metoddur. Parametre olarak vereceğimiz rakamla aslında

Stream içerisinde dönen çok sayıda veri var

ancak sen bize şu kadarını göster diyoruz.

\*/

//========================

/\*

// Arraylerde Stream kullanabilmek icin 2 yol vardir

// 1) Stream <Integer> isim=StreamOf(cevrilecek olan)

// 2) Arrays.stream(arr).

\*/

/\*

Count

Stream içerisinde bulunan toplam veri sayısını öğrenmemize yarar.

\*/

///========================

/\*

// dosya eklemek icin 2 yol var

//1) Files.lines(Path.of("path")

//buradan stream e eklemek icin

//Stream <String> satir=Files.lines(Path.of("src/day07\_file/javanin avantajlari"));

// bu bize tekrardan stream tanimlamadan kullanmayi saglar

//2) //Files.lines(Paths.get("src/day07\_file/javanin avantajlari")).

// kullanilir

satir.map(t->t.split(" ")).

flatMap(Arrays::stream). kelimelere ulasmamizi saglar

\*/

public static void main(String[] args) {

List<Integer> liste = Arrays.asList(1,2,3,4,5);

// forEach() bir collection'nın iterasyonu için kullanılan HOF'dur.

// Parametre olarak bir lambda fonksiyonu alabilir.

// Dizinin Elemanlarını yazdıralım

System.out.println("=== Dizinin Elamanlarını Yazdır ===");

liste.forEach((x) ->System.out.print(x));

//Listenin elemanlarını 2 arttırarak yazdıralım.

System.out.println("\n == Dizinin Elamanlarını 2 arttırarak Yazdır ==");

liste.forEach(t-> System.out.print(t+2 + " "));

System.out.println("\n=== Eğer satır sayısı 1'den fazla ise {} kullanmalıyız..=== ");

liste.forEach(t-> {

int miktar = 2;

System.out.print(t + miktar + " ");

});

System.out.println("\n === Veri tipi kullanılırsa : Explicit ==");

System.out.println("Dizinin Elemanlarının 2 katını yazdır.");

liste.forEach((Integer x) -> System.out.print(x\*2 + " "));

System.out.println("\nLambda ifadesinde dış değişkenler kullanalıbilir");

System.out.println("Ancak bu değişken Final gibi davranır.");

int deger= 6;

liste.forEach(t-> System.out.println(t+deger));

// Metot Refransı =====> Class adı :: Metot adı

System.out.println("\nJavanın Method referansını kullanabiliriz");

liste.forEach(System.out::print);

// Metot Referansı =====> Class adı :: Metot adı

System.out.println("\nKendimizin yazdığı metotları da Method referansı olarak kullanabiliriz");

liste.forEach(TekClass::yazdir);

}

// int değer yazdıran metot

public static void yazdir(int x){

System.out.print(x + " ");

}

}