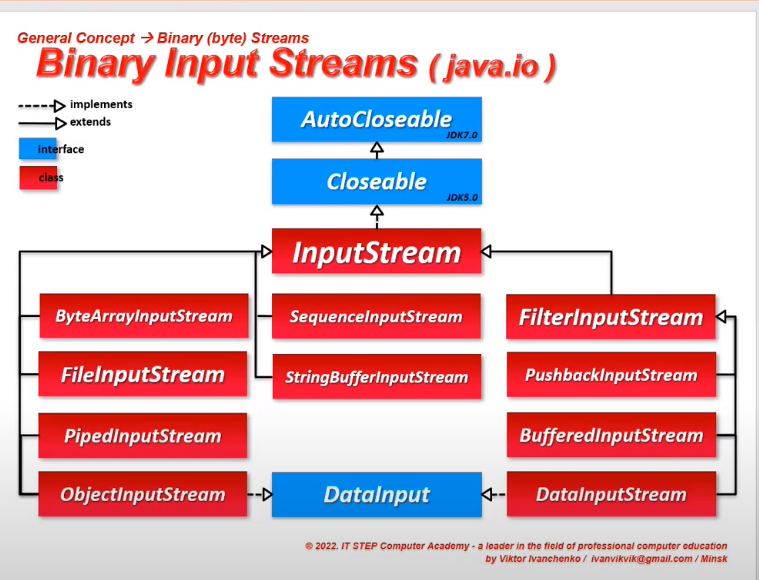
Вся библиотека потоков ввода вывода базируется на шаблоне декоратор, зная эту особенность можно разобрать библиотеку подробнее. Начнем с системы потоков ввода



Слева(всегда) конкретные имплементации

Cправа в данной схеме классы – обертки (Декоратор)

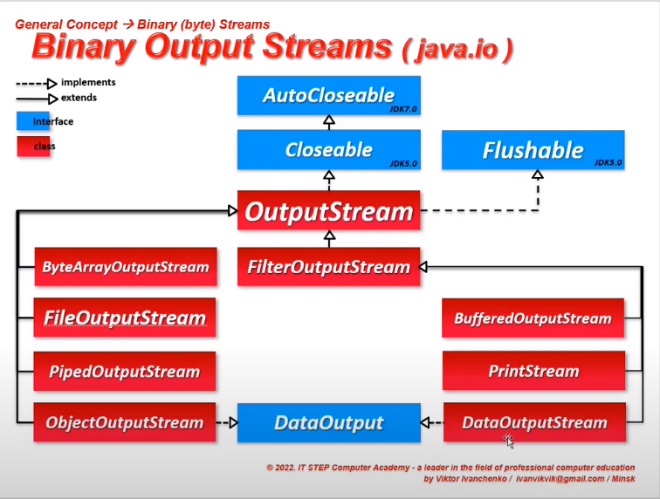
ByteArrayInputStream – позволяет в качестве входных данных использовать массив

FileInputStream – чтение из файла, уже использовали.

PipedInputStream – чтение из именованных каналов, межпроцессное взаимодействие

ObjectInputStream – сереализация (один из способов сохранения состояния системы), десереализация

Идея IF AutoCloseable возможность использования try with resources для автоматического закрытия ресурсов - close() с 7 JDK.

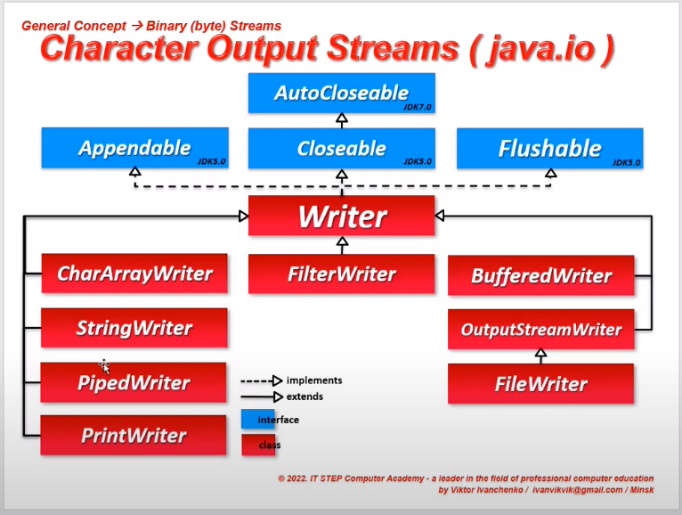
Система записи бинарных данных

BufferedOutputStream предварительная буфферизация

PrintStream конвертация бинарного потока в символьный

DataOutputStream легкий способ записи примитивных данных,

Система записи символьных данных



Рассмотрим 2 основных декоратора :

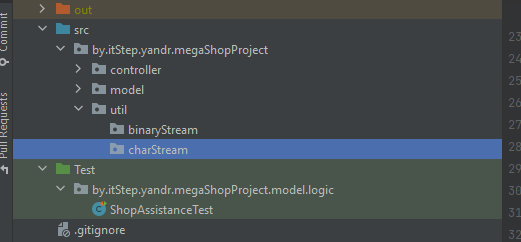
BufferedWriter/ BufferedReader – буфферизация символьных данных

package itStep.yandr.javaLessons.lesson41;  
  
import java.io.\*;  
  
public class CharacterWorker {  
  
 public static void write(String FileName) {  
  
 try (Writer stream = new BufferedWriter(new FileWriter(FileName))) {  
 for (int i = 0; i < 1000; i++) {  
 stream.write(i);  
 }  
 stream.flush();  
 } catch (IOException e) {  
 System.*err*.println(e);  
 }  
  
 }

public static String read(String FileName) {  
 StringBuilder builder = new StringBuilder();  
  
 try (Reader stream = new BufferedReader(new FileReader(FileName))) {  
 if (stream.ready()) {  
 int temp;  
 while ((temp = stream.read()) != -1) {  
 builder.append(temp).append(" ");  
 }  
 }  
 } catch (IOException e) {  
 System.*err*.println(e);  
 }  
  
 return builder + "";  
 }  
}

Реализуем долгосрочное сохранение данных программы, для получения дополнительного способа инициализации системы в 2х моделях: строки и байтовые данные.

Переходим в MegaShop:

Создаем пакет util для создания сервисных объектов и чтения и записи данных, в котором создаем 2 вложенные папки charStream (текстовые данные) и binaryStream (бинарные данные)

Начнем с создания символьного потока для Orange, задачи:1. на вход получаем массив апельсинов-сохраняем в файл; 2. Восстановление системы из файла хранящего данные о ее состоянии(метод принимающий путь к файлу)

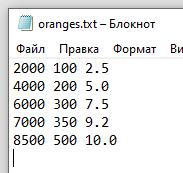
\*В символьном потоке используем стандарты JDK6.0, в бинарном – JDK7.0 и выше с использованием try with resources

package by.itStep.yandr.megaShopProject.util.charStream;  
  
import by.itStep.yandr.megaShopProject.model.entity.Orange;  
import java.io.BufferedWriter;  
import java.io.FileWriter;  
import java.io.IOException;  
import java.io.Writer;  
  
public class ProductStream {  
 public static void writeOranges(Orange[] oranges, String fileName) {  
 Writer writer = null;  
 try {  
 writer = new BufferedWriter(new FileWriter(fileName));  
 for (Orange orange : oranges) {  
 writer.write(orange.getVitaminC() + " ");  
 writer.write(orange.getDiameter() + " ");  
 writer.write(orange.getPrice() + "\n");  
 }  
 writer.flush(); гарантия закрытия потока только после получения   
 } catch (IOException e) { данных  
 System.*err*.println(e);  
 } finally {  
 try {  
 if (writer != null) {  
 writer.close();  
 }  
 } catch (IOException e) {  
 System.*err*.println(e);  
  
 }  
 }  
 }  
}

Переходим в main

public class Main {  
 public static void main(String[] args) {  
  
 Orange orange1 = new Orange(100,2000,2.5);  
 Orange orange2 = new Orange(200,4000,5);  
 Orange orange3 = new Orange(300,6000,7.5);  
 Orange []oranges = {orange1,orange2,orange3};  
 ProductStream.*writeOranges*(oranges,"D:/oranges.txt");

}

Тж мы можем добавить теперь объекты прямо в файле просто прописав их параметры, добавим параметры нескольких объектов и создадим парсер для чтения файла из динамического контейнера List<Orange>

public static List<Orange> readOranges(String fileName) {  
 List<Orange> list = new ArrayList<>();  
 BufferedReader reader = null;  
 try {  
 reader = new BufferedReader(new FileReader(fileName));  
 //int size = Integer.parseInt(reader.readLine());  
 String buf = reader.readLine();  
 int count = 0;  
 while (buf != null) {  
 count++;  
 //if (buf != "")  
 {  
 //for (int i = 0; i < size; i++) {  
 String[] parameters = buf.split(" "); //"3.5 2500 200".split() -> String[] "3.5" "2500" "200"  
 Orange orange = new Orange();  
 double price = Double.*parseDouble*(parameters[0]);  
 int vitamin = Integer.*parseInt*(parameters[1]);  
 int diameter = Integer.*parseInt*(parameters[2]);  
 orange.setPrice(price);  
 orange.setVitaminC(vitamin);  
 orange.setDiameter(diameter);  
 list.add(orange);  
 buf = reader.readLine();  
 }  
 }  
 System.*out*.println("count = " + count);  
 } catch (IOException exception) {  
 System.*err*.println(exception);  
  
 } finally {  
 try {  
 if (reader != null) {  
 reader.close();  
 }  
 } catch (IOException exception) {  
 System.*err*.println(exception);  
 }  
 }  
  
 return list;  
}

psvm

List<Orange>oranges = ProductStream.*readOranges*("D:/oranges.txt");  
 for (Orange orange : oranges) {  
 System.*out*.println(orange);  
 }

