**Міністерство Освіти І НАУКИ України**

**Національний університет "Львівська політехніка"**

Інститут **КНІТ**

Кафедра **ПЗ**

**ЗВІТ**

До лабораторної роботи №1

**На тему:** *“Типи даних, змінні та масиви у мові програмування Java”*

**З дисципліни:** *“Кросплатформне програмування”*

**Лектор:**

доц. каф. ПЗ

Дяконюк Л.М.

**Виконав:**

ст. гр. ПЗ-32

Бурець В.В.

**Прийняв:**

асист. каф. ПЗ

Шкраб Р. Р.

« \_\_\_\_ » \_\_\_\_\_\_\_\_ 2020 р.

∑= \_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Тема роботи:** Об’єктно-орієнтоване програмування мовою Java.

**Мета роботи**: Навчитися використовувати основні конструкції мови програмування Java, використовувати можливості ООП.

**Завдання**

1. Використовувати можливості ООП: класи, успадкування, поліморфізм, інкапсуляцію.

2. Кожний клас повинен мати назву, яка повністю описує його суть, і інформативний склад. Атрибути і методи класів слід визначити самостійно

3. Успадкування потрібно використовувати тільки тоді, коли воно має сенс. У випадку використання наслідування кількість класів-нащадків має бути не меншою 2 і не більшою 4-ьох.

4. При записі програми потрібно використовувати домовленості щодо оформлення коду java code convention.

5. Зображати всі пари set/get (ака сеттери/геттери) для атрибутів класу не потрібно з метою уникнення засмічення діаграми

6. Для реалізації операцій пошуку/сортування слід реалізувати окремий клас (в назві якого має бути присутнє слово Manager)

7. Реалізувати один з методів сортування з використанням компаратора, реалізованого як статичний вкладений клас (static inner class)

8. Реалізувати наступний з методів сортування з використанням компаратора, реалізованого як вкладений клас (inner class)

9. Додати ще один метод сортування (додатковий), який реалізує сортування з використанням анонімного класу (anonymous inner class)

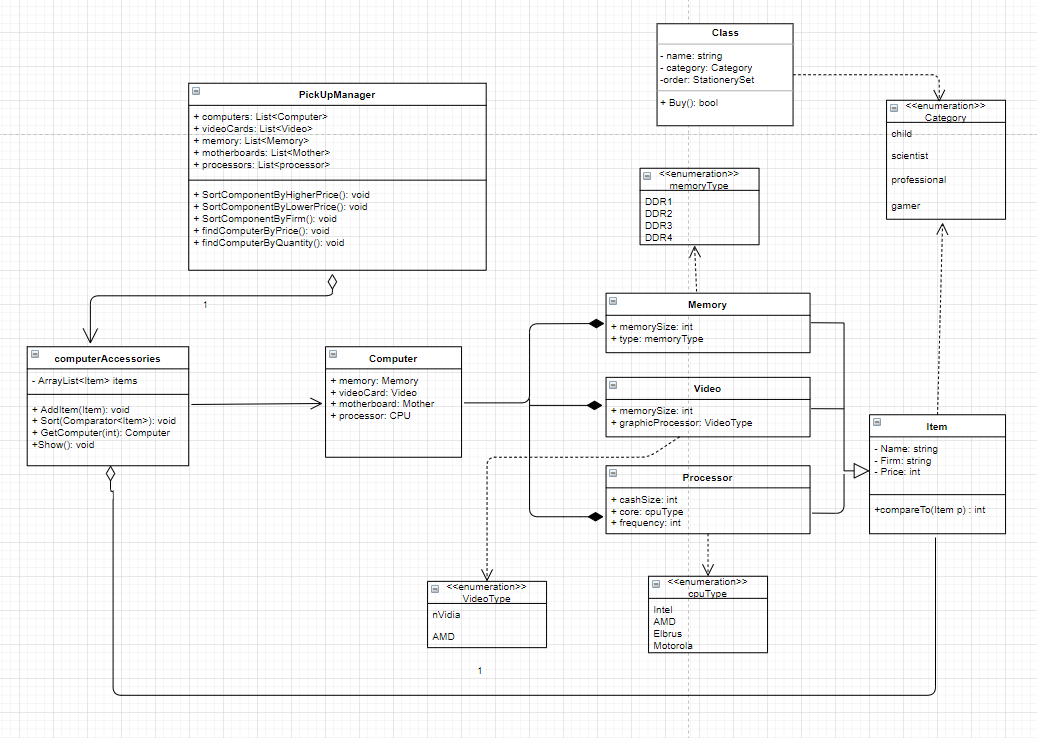
10.Додати ще один, 4-й метод сортування з використанням лямбда-виразів.

*Варіант:*

Комп'ютерна фірма. Реалізувати ієрархію компонентів, які необхідні для бирання комп'ютера. Реалізувати пошук компонентів, які дозволять скласти комп'ютер ціною, меншою за 500$. Вивести компоненти, відсортовані за маркою виробника і їх ціною. Реалізація сортування має передбачати можливість сортувати як за спаданням, так і за зростанням.

**Хід виконання**

1. Діаграма класів.



1. Використання наслідування класів.

public abstract class Item {  
 protected String Name;  
 protected String Firm;  
 protected int price;  
 protected itemCategory cat;  
 protected int quantity;  
  
 public int getPrice() {  
 return price;  
 }  
  
 public String getFirm() {  
 return Firm;  
 }  
  
 public String getName() {  
 return Name;  
 }  
  
 public void setFirm(String firm) {  
 Firm = firm;  
 }  
  
 public void setName(String name) {  
 Name = name;  
 }  
  
 public void setPrice(int price) {  
 this.price = price;  
 }  
  
 public void setQuantity(int quantity) {this.quantity = quantity;}  
  
 public int getQuantity(){return this.quantity;}  
  
 public Item(){  
 this("Item", "Firm,", 100, itemCategory.*none*, 1);  
 }  
  
 public Item(String name, String firm, int price, itemCategory cat, int quantity){  
 this.Name = name;  
 this.cat = cat;  
 this.Firm = firm;  
 this.price = price;  
 this.quantity = quantity;  
 }  
   
 public int compareTo(Item p){  
 int i = Name.compareTo(p.getName());  
 return i;  
 }  
  
 @Override  
 public int hashCode(){  
 return cat.hashCode() + (int)price;  
 }

//import jdk.jfr.Category;  
  
public class Video extends Item {  
 private int memorySize;  
 private VideoType graphicProcessor;  
  
 public Video(String Name,  
 int price,  
 String Firm,  
 int memorySize,  
 VideoType graphicProcessor,  
 itemCategory cat,  
 int quantity)  
 {  
 this.price = price;  
 this.memorySize = memorySize;  
 this.graphicProcessor = graphicProcessor;  
 this.Firm = Firm;  
 this.Name = Name;  
 this.cat = cat;  
 this.quantity = quantity;  
 }  
  
 public int getMemorySize() {  
 return memorySize;  
 }  
  
 public VideoType getGraphicProcessor() {  
 return graphicProcessor;  
 }  
  
 public void setMemorySize(int memorySize) {  
 this.memorySize = memorySize;  
 }  
  
 public void setGraphicProcessor(VideoType graphicProcessor) {  
 this.graphicProcessor = graphicProcessor;  
 }  
  
 @Override  
 public int hashCode(){  
 return super.hashCode() + this.price + graphicProcessor.hashCode();  
 }  
  
 @Override  
 public String toString(){  
 return graphicProcessor.toString() + " " + Firm + " " + Name.toString() + " price - $"+ price + " Videocard for "+cat.toString() +  
 " Specifications: graphcsProcessor - " + graphicProcessor.toString() + ", memory size - " + memorySize + " MB\n";  
 }  
}

import jdk.jfr.Category;  
  
public class Memory extends Item{  
 private int memorySize;  
 private memoryType type;  
  
 public int getMemorySize() {  
 return memorySize;  
 }  
  
 public memoryType getType() {  
 return type;  
 }  
  
 public void setMemorySize(int memorySize) {  
 this.memorySize = memorySize;  
 }  
  
 public void setType(memoryType type) {  
 this.type = type;  
 }  
  
 public Memory()  
 {  
 this(100, "firm", 1000, memoryType.*none*, 1);  
 }  
  
 public Memory(int price,  
 String Firm,  
 int memorySize,  
 memoryType type,  
 int quantity)  
 {  
 this.price = price;  
 this.memorySize = memorySize;  
 this.type = type;  
 this.Firm = Firm;  
 this.quantity = quantity;  
 }  
  
 @Override  
 public int hashCode(){  
 return super.hashCode() + memorySize + type.hashCode();  
 }  
  
 @Override  
 public String toString(){  
 return type.toString()+Firm + " price - $"+ price +  
 " Specifications: type of memory - " + type.toString() + ", memory size - " + memorySize + "MB\n";  
 }  
}

public class CPU extends Item{  
 private int cashSize;  
 private cpuType coreType;  
 private int frequency;  
  
 public void setCashSize(int cashSize) {  
 this.cashSize = cashSize;  
 }  
  
 public void setCoreType(cpuType coreType) {  
 this.coreType = coreType;  
 }  
  
 public void setFrequency(int frequency) {  
 this.frequency = frequency;  
 }  
  
 public int getCashSize() {  
 return cashSize;  
 }  
  
 public int getFrequency() {  
 return frequency;  
 }  
  
 public cpuType getCoreType() {  
 return coreType;  
 }  
  
 public CPU()  
 {  
 this(cpuType.*none*, 100, 10, 100, itemCategory.*none*, 1);  
 }  
  
  
 public CPU(cpuType coreType,  
 int frequency,  
 int cashSize,  
 int price,  
 itemCategory cat,  
 int quantity)  
 {  
 this.coreType = coreType;  
 this.frequency = frequency;  
 this.cashSize = cashSize;  
 this.price = price;  
 this.cat = cat;  
 this.quantity = quantity;  
 }  
  
  
 @Override  
 public int hashCode(){  
 return super.hashCode() + cashSize + coreType.hashCode();  
 }  
  
 @Override  
 public String toString(){  
 String s = Firm + " " + Name.toString() + " price - " + price +  
 " Specifications: Core - " + coreType.toString() + ", cash size - " + cashSize +  
 "MB frequency - " + frequency + "Hz\n";  
 return s;  
 }  
  
}

public class Computer {  
 Memory memory;  
 Video videoCard;  
 CPU processor;  
  
 public Computer(Memory memory,  
 Video videoCard,  
 CPU processor)  
 {  
 this.memory = memory;  
 this.processor = processor;  
 this.videoCard = videoCard;  
 }  
  
 public CPU getProcessor() {  
 return processor;  
 }  
  
 public Memory getMemory() {  
 return memory;  
 }  
  
 public Video getVideoCard() { return videoCard; }  
  
 public void setMemory(Memory m) { this.memory = m; }  
  
 public void setProcessor(CPU c) { this.processor = c; }  
  
 public void setVideoCard(Video v) { this.videoCard = v; }  
  
 @Override  
 public String toString(){  
 if(memory == null || processor == null || videoCard == null){  
 String s = " We can`t pick up a computer for you\n";  
 return s;  
 }  
 else{  
 return " We picked up a computer for you \n technical characteristic is \n" + memory.toString() + "\n" +  
 processor.toString() + "\n" + videoCard.toString();  
 }  
 }  
}

3) Реалізація менеджеру для сортування та пошуку

import jdk.jfr.Category;  
  
import java.util.ArrayList;  
import java.util.Collections;  
import java.util.Comparator;  
  
  
public class PickUpManager {  
  
 private static SortType *sortType*;  
 private static PickUpManager *instance*;  
 private computerAccessories accessories;  
  
 public PickUpManager()  
 {  
 *sortType* = SortType.*increasing*;  
 accessories = new computerAccessories();  
 accessories.AddItem(new Video("geforse 1060", 200, "asus", 3000, VideoType.*nVidia*, itemCategory.*gamer*, 3));  
 accessories.AddItem(new Video("geforse 1020", 100, "asrok", 1000, VideoType.*nVidia*, itemCategory.*child*, 1));  
 accessories.AddItem(new Video("hd6670", 50, "palit", 500, VideoType.*AMD*, itemCategory.*none*, 2));  
  
 accessories.AddItem(new CPU(cpuType.*AMD*, 2000, 10, 400, itemCategory.*none*, 5));  
 accessories.AddItem(new CPU(cpuType.*Intel*, 3000, 60, 800, itemCategory.*gamer*, 2));  
 accessories.AddItem(new CPU(cpuType.*Motorola*, 1000, 5, 100, itemCategory.*none*, 6));  
  
 accessories.AddItem(new Memory(50, "Kingston", 8000, memoryType.*DDR4*, 4));  
 accessories.AddItem(new Memory(30, "SiliconPower", 4000, memoryType.*SODIM\_DDR3*, 5));  
 accessories.AddItem(new Memory(20, "Crusial", 2000, memoryType.*DDR2*, 3));  
  
 }  
  
  
 public static PickUpManager getInstance() {  
 if(*instance* == null)  
 *instance* = new PickUpManager();  
 return *instance*;  
 }  
  
 public static void setSortType(SortType type){  
 PickUpManager.*sortType* = type;  
 }  
  
 //static inner class  
 public static class accessoriesComparatorByName implements Comparator<Item> {  
 public int compare(Item o1, Item o2) {  
 // descending order (ascending order would be: o1.getGrade()-o2.getGrade())  
 int result = o1.getName().compareTo(o2.getName());  
 if(*sortType* == SortType.*increasing*)  
 return result;  
 else  
 return -result;  
 }  
 }  
  
 //inner class  
 class accessoriesComparatorByPrice implements Comparator<Item> {  
 public int compare(Item o1, Item o2){  
 int mult = 1;  
 if(*sortType* == SortType.*decreasing*)  
 mult = -1;  
 double sub = o1.getPrice() - o2.getPrice();  
 if(sub > 0)  
 return 1 \* mult;  
 if(sub < 0)  
 return -1 \* mult;  
 return 0;  
 }  
 }  
  
 public void SortAccessoriesByName(ArrayList<Item> a){  
 accessories.Sort(a, new accessoriesComparatorByName());  
 }  
  
 public void SortAccessoriesByPrice(ArrayList<Item> a){  
 accessories.Sort(a, new accessoriesComparatorByPrice());  
 }  
  
  
 public void SortAccessoriesByName(){  
 accessories.Sort(new accessoriesComparatorByName());  
 }  
  
 public void SortAccessoriesByPrice(){  
 accessories.Sort(new accessoriesComparatorByPrice());  
 }  
  
 //anonymous class  
 public void SortAccessoriesByFirm(ArrayList<Item> a){  
 accessories.Sort(a, new Comparator<Item>(){  
 public int compare(Item o1, Item o2) {  
 int mult = 1;  
 if (*sortType* == SortType.*decreasing*)  
 mult = -1;  
  
 var i = (o1.getFirm().compareTo(o2.getFirm())) \* mult;  
 return i;  
 }  
 });  
 }  
  
 public void SortAccessoriesByQuantity(ArrayList<Item> a){  
 accessories.Sort(a, (Item o1, Item o2) -> {  
 int mult = 1;  
 if(*sortType* == SortType.*decreasing*)  
 mult = -1;  
 return (o1.getQuantity() - o2.getQuantity()) \* mult;  
 });  
 }  
  
 //anonymous class  
 public void SortAccessoriesByFirm(){  
 accessories.Sort(new Comparator<Item>(){  
 public int compare(Item o1, Item o2) {  
 int mult = 1;  
 if (*sortType* == SortType.*decreasing*)  
 mult = -1;  
  
 var i = (o1.getFirm().compareTo(o2.getFirm())) \* mult;  
 return i;  
 }  
 });  
 }  
  
 public void SortAccessoriesByQuantity(){  
 accessories.Sort((Item o1, Item o2) -> {  
 int mult = 1;  
 if(*sortType* == SortType.*decreasing*)  
 mult = -1;  
 return (o1.getQuantity() - o2.getQuantity()) \* mult;  
 });  
 }  
  
  
 public Computer HandleOrder(){  
 return accessories.GetComputer(500);  
 }  
  
 public Computer HandleOrder(int price){  
 return accessories.GetComputer(price);  
 }  
  
 @Override  
 public String toString(){  
 String s = accessories.Show();  
 return s;  
 }  
  
}

import java.util.ArrayList;  
import java.util.Collections;  
import java.util.Comparator;  
  
public class computerAccessories {  
 private ArrayList<Item> items;  
  
 public computerAccessories(){  
 items = new ArrayList<>();  
 }  
  
 public void AddItem(Item i) {  
 items.add(i);  
 }  
  
 public void Sort(ArrayList<Item> accessories, Comparator<Item> compare)  
 {  
 Collections.*sort*(accessories, compare);  
 }  
  
 public void Sort(Comparator<Item> compare)  
 {  
 Collections.*sort*(items, compare);  
 }  
  
 public Computer GetComputer(int price){  
  
 Memory m = null;  
 Video v = null;  
 CPU c = null;  
  
 double c\_price = price / 2;  
 double v\_price = ((price / 2) \* 0.7);  
 double m\_price = price / 2 - v\_price;  
  
 for(var item : items) {  
 if(item instanceof Memory)  
 {  
 if(item.getPrice() < m\_price){  
 if(m == null){  
 m = (Memory) item;  
 }  
 }  
 }  
 if(item instanceof Video)  
 {  
 if(item.getPrice() < v\_price){  
 if(v == null){  
 v = (Video) item;  
 }  
 }  
  
 }  
 if(item instanceof CPU)  
 {  
 if(item.getPrice() < c\_price){  
 if(c == null){  
 c = (CPU) item;  
 }  
 }  
  
 }  
 }  
 Computer comp = new Computer(m, v, c);  
  
 return comp;  
 }  
  
 public String Show() {  
 String s = new String();  
  
 for (var item: items){  
 s += item.toString();  
 s += "\n";  
 }  
  
 return s;  
  
 }  
  
}

**Результати**

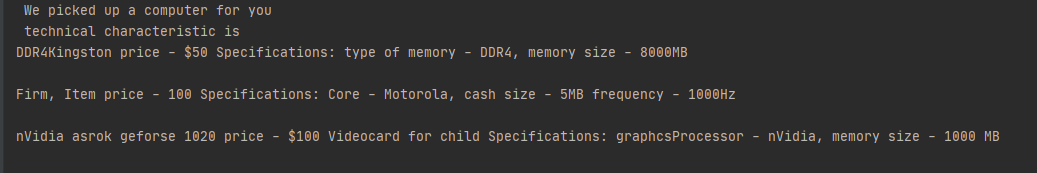


Рис. 1. Підбір комп'ютера з ціною меншою за $500

**Висновки**

В ході виконання лабораторної роботи було опрацьовано теоретичний матеріал стосовно основних конструкцій програмування мови Java, парадигм ООП. Реалізовано прототип комп’ютерної фірми, який може підбирати комп’ютери, а також показувати товари, які є в наявності і сортувати їх за кількістю, фірмою, назвою, ціною. А також було здобуто навички у роботі з середовищем програмування Intellij IDEA, розглянуто типи даних, змінні та масиви мови програмування Java.