**Лабораторная работа №2**

**Коллекции**

Написать и отладить три программы. Первая программа демонстрирует использование контейнерных классов для хранения встроенных типов данных.

Вторая программа демонстрирует использование контейнерных классов для хранения пользовательских типов данных.

Третья программа демонстрирует использование алгоритмов работы с коллекциями.

В программе № 1 выполнить следующее:

1. Создать объект-контейнер в соответствии с вариантом задания (тип 1) и заполнить его данными, тип которых определяется вариантом задания.
2. Просмотреть контейнер.
3. Изменить контейнер, удалив из него одни элементы и заменив другие.
4. Просмотреть контейнер, используя для доступа к его элементам итераторы.
5. Создать второй контейнер этого же класса и заполнить его данными того же типа, что и первый контейнер.
6. Изменить первый контейнер, удалив из него n элементов после заданного и добавив затем в него все элементы из второго контейнера.
7. Просмотреть первый и второй контейнеры.

В программе № 2 выполнить то же самое, но для данных пользовательского типа.

В программе № 3 выполнить следующее:

1. Создать контейнер, содержащий объекты пользовательского типа. Тип контейнера выбирается в соответствии с вариантом задания (тип1).
2. Отсортировать его по возрастанию элементов.
3. Просмотреть контейнер.
4. Используя подходящий алгоритм, найти в контейнере элемент, удовлетворяющий заданному условию.
5. Переместить элементы, удовлетворяющие заданному условию в другой (предварительно пустой) контейнер. Тип второго контейнера определяется вариантом задания (тип2).
6. Просмотреть второй контейнер.
7. Отсортировать первый и второй контейнеры по убыванию элементов.
8. Просмотреть их.
9. Получить третий контейнер путем слияния первых двух. Тип третьего контейнера определить самостоятельно.
10. Просмотреть третий контейнер.

Вариант 9:

1 контейнер: LinkedList

2 контейнер: TreeMap

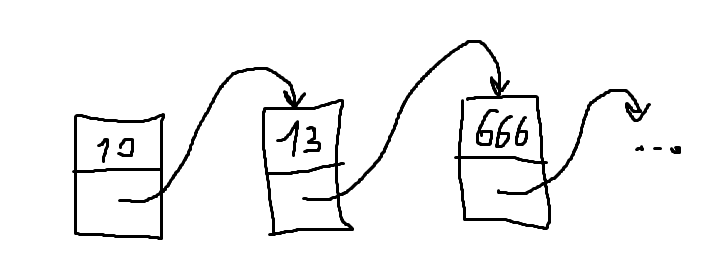
Встроенный тип данных: int

Пользовательский тип данных: Квитанция

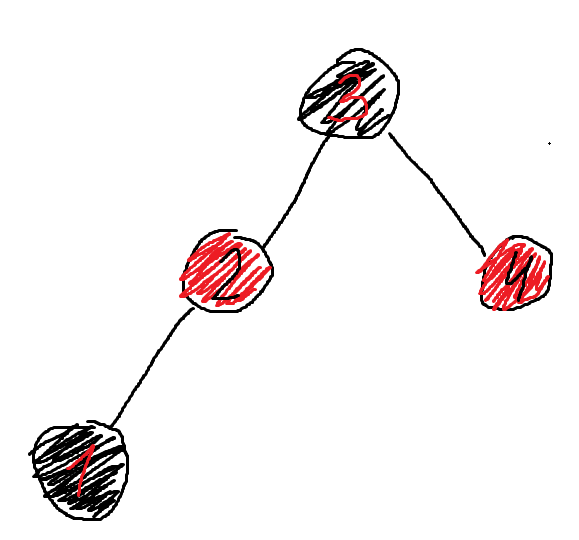
* номер – int
* дата – int
* сумма – float

**Описание контейнеров**

LinkedList – связный список. Это структура данных, состоящая из узлов, каждый из которых содержит значение и указатель на следующий узел. Каждый элемент (узел) знает, где находится следующий, но не знает, кто был до него.



TreeMap – это одна из реализаций интерфейса Map, основанная на красно-чёрном дереве. Она хранит пары ключ–значение и автоматически сортирует ключи по их естественному порядку или по компаратору, если он был передан при создании.



**Текст программы**

1 программа:

import java.util.\*;  
  
public class Main {  
 public static void main(String[] args) {  
 *// 1. Создать контейнер LinkedList и заполнить его данными типа int* LinkedList<Integer> list1 = new LinkedList<>();  
 for (int i = 1; i <= 10; i++) {  
 list1.add(i);  
 }  
  
 *// 2. Просмотреть контейнер* System.*out*.println("Первый контейнер:");  
 System.*out*.println(list1);  
  
 *// 3. Удалить из контейнера одни элементы и заменить другие* list1.removeIf(n -> n % 2 == 0); *// удалить чётные* for (int i = 0; i < list1.size(); i++) {  
 if (list1.get(i) == 5) {  
 list1.set(i, 55); *// заменить 5 на 55* }  
 }  
  
 *// 4. Просмотреть контейнер с использованием итераторов* System.*out*.println("Первый контейнер (через итератор):");  
 Iterator<Integer> it = list1.iterator();  
 while (it.hasNext()) {  
 System.*out*.print(it.next() + " ");  
 }  
 System.*out*.println();  
  
 *// 5. Создать второй контейнер и заполнить его данными типа int* LinkedList<Integer> list2 = new LinkedList<>();  
 for (int i = 100; i <= 105; i++) {  
 list2.add(i);  
 }  
  
 *// 6. Удалить из первого контейнера n элементов после заданного и добавить элементы из второго* int startIndex = 1;  
 int n = 2;  
  
 int removeStart = Math.*min*(startIndex + 1, list1.size());  
 for (int i = 0; i < n && removeStart < list1.size(); i++) {  
 list1.remove(removeStart);  
 }  
  
 list1.addAll(list2);  
  
 *// 7. Просмотреть оба контейнера* System.*out*.println("Первый контейнер после изменений:");  
 System.*out*.println(list1);  
  
 System.*out*.println("Второй контейнер:");  
 System.*out*.println(list2);  
 }  
}

2 программа:

public class Receipt {  
 int number;  
 int date; *// формат: ГГГГММДД* float sum;  
  
 public Receipt(int number, int date, float sum) {  
 this.number = number;  
 this.date = date;  
 this.sum = sum;  
 }  
  
 @Override  
 public String toString() {  
 return "Квитанция №" + number + ", дата: " + date + " сумма: " + sum;  
 }  
}

import java.util.\*;  
  
public class Main {  
 public static void main(String[] args) {  
 *// 1. Создать контейнер LinkedList и заполнить его данными типа Receipt* LinkedList<Receipt> list1 = new LinkedList<>();  
 list1.add(new Receipt(1, 20250401, 1200.50f));  
 list1.add(new Receipt(2, 20250402, 800.00f));  
 list1.add(new Receipt(3, 20250403, 950.75f));  
 list1.add(new Receipt(4, 20250404, 1100.25f));  
 list1.add(new Receipt(5, 20250405, 300.00f));  
  
 *// 2. Просмотреть контейнер* System.*out*.println("Первый контейнер:");  
 for (Receipt k : list1) {  
 System.*out*.println(k);  
 }  
  
 *// 3. Удалить из контейнера одни элементы и заменить другие* list1.removeIf(k -> k.sum < 900); *// удалить квитанции с суммой < 900* for (int i = 0; i < list1.size(); i++) {  
 if (list1.get(i).number == 3) {  
 list1.set(i, new Receipt(33, 20250410, 3333.33f)); *// замена по номеру* }  
 }  
  
 *// 4. Просмотреть контейнер с использованием итераторов* System.*out*.println("\nПервый контейнер (через итератор):");  
 Iterator<Receipt> it = list1.iterator();  
 while (it.hasNext()) {  
 System.*out*.println(it.next());  
 }  
  
 *// 5. Создать второй контейнер и заполнить его данными типа Receipt* LinkedList<Receipt> list2 = new LinkedList<>();  
 list2.add(new Receipt(10, 20250406, 1500.00f));  
 list2.add(new Receipt(11, 20250407, 1750.50f));  
 list2.add(new Receipt(12, 20250408, 1800.00f));  
  
 *// 6. Удалить из первого контейнера n элементов после заданного и добавить элементы из второго* int startIndex = 0; *// после первого элемента* int n = 1; *// количество удаляемых элементов* int removeStart = Math.*min*(startIndex + 1, list1.size());  
 for (int i = 0; i < n && removeStart < list1.size(); i++) {  
 list1.remove(removeStart);  
 }  
  
 list1.addAll(list2);  
  
 *// 7. Просмотреть оба контейнера* System.*out*.println("\nПервый контейнер после изменений:");  
 for (Receipt k : list1) {  
 System.*out*.println(k);  
 }  
  
 System.*out*.println("\nВторой контейнер:");  
 for (Receipt k : list2) {  
 System.*out*.println(k);  
 }  
 }  
}

3 программа:

public class Receipt {  
 int number;  
 int date; *// формат: ГГГГММДД* float sum;  
  
 public Receipt(int number, int date, float sum) {  
 this.number = number;  
 this.date = date;  
 this.sum = sum;  
 }  
  
 public float getSum() {  
 return sum;  
 }  
  
 @Override  
 public String toString() {  
   
 return "Квитанция №" + number + ", дата: " + date + " сумма: " + sum;  
 }  
}

import java.util.\*;  
  
public class Main {  
 public static void main(String[] args) {  
 *// 1. Создание LinkedList и заполнение* LinkedList<Receipt> list = new LinkedList<>();  
 list.add(new Receipt(1, 20250401, 1200.50f));  
 list.add(new Receipt(2, 20250402, 800.00f));  
 list.add(new Receipt(3, 20250403, 950.75f));  
 list.add(new Receipt(4, 20250404, 1800.25f));  
 list.add(new Receipt(5, 20250405, 600.00f));  
  
 *// 2. Сортировка по возрастанию суммы* list.sort(Comparator.*comparingDouble*(Receipt::getSum));  
  
 *// 3. Просмотр контейнера* System.*out*.println("Сортировка по возрастанию (LinkedList):");  
 for (Receipt k : list) {  
 System.*out*.println(k);  
 }

*// 4. Поиск элемента с суммой > 1000*Receipt found = null;  
for (Receipt k : list) {  
 if (k.sum > 1000) {  
 found = k;  
 break;  
 }  
}  
  
System.*out*.println("\nПервый элемент с суммой > 1000:");  
if (found != null) {  
 System.*out*.println(found);  
} else {  
 System.*out*.println("Не найдено");  
}

*// 5. Переместить элементы с суммой < 1000 в TreeMap* TreeMap<Integer, Receipt> map = new TreeMap<>();  
 Iterator<Receipt> it = list.iterator();  
 while (it.hasNext()) {  
 Receipt k = it.next();  
 if (k.sum < 1000) {  
 map.put(k.number, k);  
 it.remove();  
 }  
 }  
  
 *// 6. Просмотр второго контейнера* System.*out*.println("\nTreeMap с квитанциями < 1000:");  
 for (Map.Entry<Integer, Receipt> entry : map.entrySet()) {  
 System.*out*.println(entry.getValue());  
 }  
  
 *// 7. Сортировка обоих контейнеров по убыванию* list.sort((a, b) -> Float.*compare*(b.sum, a.sum));  
  
 TreeMap<Integer, Receipt> sortedMap = new TreeMap<>((a, b) -> b - a);  
 sortedMap.putAll(map);  
  
 *// 8. Просмотр после сортировки по убыванию* System.*out*.println("\nLinkedList по убыванию:");  
 for (Receipt k : list) {  
 System.*out*.println(k);  
 }  
  
 System.*out*.println("\nTreeMap по убыванию ключей:");  
 for (Map.Entry<Integer, Receipt> entry : sortedMap.entrySet()) {  
 System.*out*.println(entry.getValue());  
 }  
  
 *// 9. Слияние в третий контейнер (ArrayList)* ArrayList<Receipt> merged = new ArrayList<>();  
 merged.addAll(list);  
 merged.addAll(sortedMap.values());  
  
 *// 10. Просмотр третьего контейнера* System.*out*.println("\nИтоговый контейнер (ArrayList):");  
 for (Receipt k : merged) {  
 System.*out*.println(k);  
 }  
 }  
}

**Описание программы**

1 программа:

1. Программа создает контейнер **list1** типа **LinkedList** для хранения элементов типа **Integer**. Далее она заполняет его числами от **1** до **10**
2. Программа выводит на экран контейнер **list1**
3. Программа удаляет из контейнера **list1** все четные элементы. Затем программа проходит по всем элементам контейнера и ищет среди них равные **5**. Если таковой находится – он заменяется на **55**
4. Инициализируется итератор **it**. С его помощью выводится содержимое контейнера **list1**
5. Программа создает контейнер **list2** типа **LinkedList** для хранения элементов типа **Integer**. Далее она заполняет его числами от **100** до **105**
6. Программа удаляет из 1 контейнера n элементов после заданного элемента **startIndex**. По умолчанию **n = 2**, а **startIndex = 1**. Затем программа добавляет в **list1** элементы из **list2**
7. Программа выводит на экран контейнеры **list1** и **list2**

2 программа:

Класс **Receipt** содержит поля:

* **number** – номер квитанции
* **date** – дата квитанции
* **sum** – сумма квитанции

Класс **Receipt** содержит методы

* **Receipt()** – конструктор
* **toString()** – перегрузка метода для вывода информации о квитанции

1. Программа создает контейнер **list1** типа **LinkedList** для хранения элементов типа **Receipt**. Далее она создает 5 объектов класса **Receipt** и добавляет их в **list1**
2. Программа выводит на экран контейнер **list1**
3. Программа удаляет из контейнера **list1** все квитанции с суммой менее **900**. Затем программа проходит по всем элементам контейнера и ищет среди них квитанции с номером 3. Если таковая находится – ее номер заменяется на **33**, дата на **20250410**, а сумма – на **3333,33**
4. Инициализируется итератор **it**. С его помощью выводится содержимое контейнера **list1**
5. Программа создает контейнер **list2** типа **LinkedList** для хранения элементов типа **Receipt**. Далее она создает 3 объекта класса **Receipt** и добавляет их в **list2**
6. Программа удаляет из 1 контейнера n элементов после заданного элемента **startIndex**. По умолчанию **n = 1**, а **startIndex = 0**. Затем программа добавляет в **list1** элементы из **list2**
7. Программа выводит на экран контейнеры **list1** и **list2**

3 программа:

Класс **Receipt** содержит поля:

* **number** – номер квитанции
* **date** – дата квитанции
* **sum** – сумма квитанции

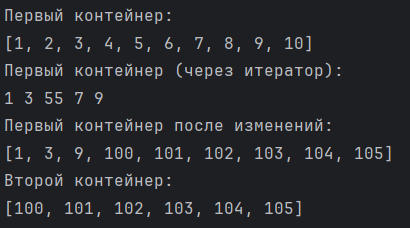
Класс **Receipt** содержит методы

* **Receipt()** – конструктор
* **getSum()** – геттер для суммы
* **toString()** – перегрузка метода для вывода информации о квитанции

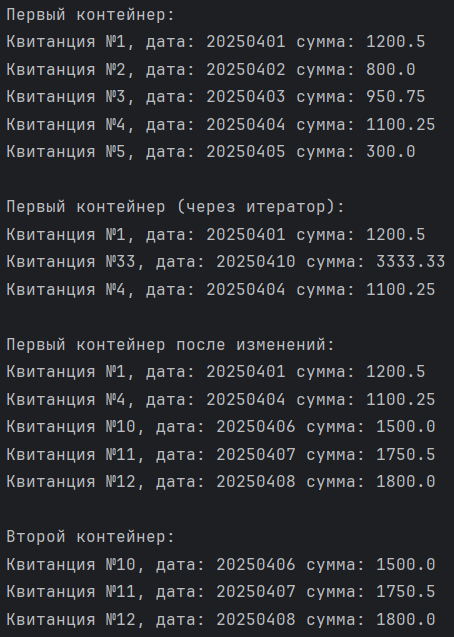
1. Программа создает контейнер **list** типа **LinkedList** для хранения элементов типа **Receipt**. Далее она создает 5 объектов класса **Receipt** и добавляет их в **list**
2. Создается компаратор, который сравнивает элементы списка **list** по возвращаемому значению из метода **getSum()** – сумме квитанции. Компаратор используется для сортировки элементов списка по возрастанию
3. Программа выводит на экран контейнер **list**
4. Программа инициализирует объект **found** класса **Receipt** с начальным значением **null**. Затем программа проходит по всем элементам контейнера **list** и проверяет – сумма квитанции больше **1000** или нет. Если да, то эта квитанция сохраняется в **found**. После этого цикл прекращается. Если квитанция была найдена – она выводится на экран. Если нет – выводится сообщение *«Не найдено»*
5. Программа создает контейнер **map** типа **TreeMap** для хранения элементов типа **Receipt**. Инициализируется итератор **it**. С его помощью элементы **list** с суммой меньше **1000** записываются в map и удаляются из **list**
6. Программа выводит на экран контейнер **map**
7. Создается компаратор, который сравнивает элементы списка **list** по сумме. Компаратор используется для сортировки элементов списка по убыванию. Создается компаратор, который сравнивает элементы **map** по ключам. Компаратор используется для сортировки элементов списка по убыванию. Отсортированные элементы записываются в контейнер **sortedMap**
8. Программа выводит на экран контейнеры **list** и **sortedMap**
9. Программа создает контейнер merged типа **ArrayList**. Она добавляет в него элементы контейнеров **list** и **sortedMap**
10. Программа выводит на экран контейнер **merged**

**Результаты тестов**

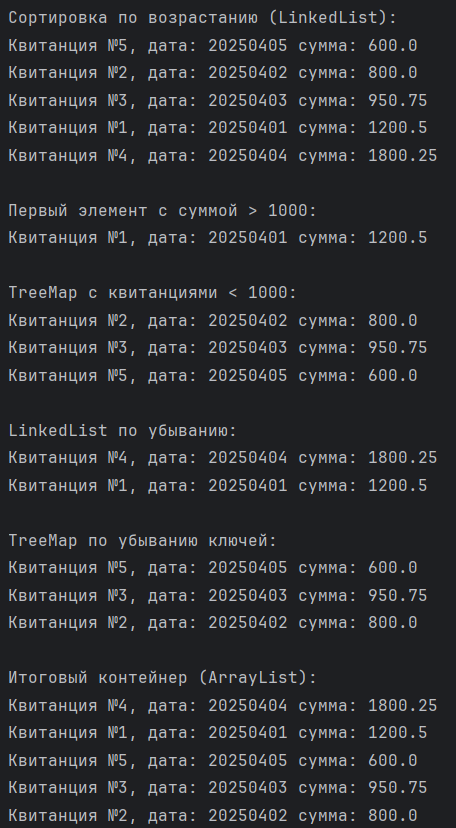
1 программа:



2 программа:



3 программа:



**Вывод**

В ходе выполнения лабораторной работы были созданы контейнеры типов LinkedList и TreeMap. Над ними были выполнены различные операции, такие как добавление, изменение, удаление и поиск элементов, а также сортировка. Были написаны протестированы 3 программы, в которых данные контейнеры хранят данные как встроенных типов, так и пользовательского типа Receipt – квитанция.