Міністерство освіти і науки України

Харківський національний університет ім. В. Н. Каразіна

Факультет комп’ютерних наук

Кафедра безпеки інформаційних систем і технологій

Лабораторна робота №8

*з навчальної дисципліни*

«Спеціалізовані мови програмування та проектування електронних елементів і систем»

Виконав:

Студент групи КБ-41

Кравченко Є.М.

Перевірив:

Олійников Р.В.

Харків – 2020 р.

1.Создать обобщенный класс (generic), объединяющий три объекта одного класса. Класс объектов, принимаемых на входе, и сам созданный обобщенный класс, реализуют интерфейс Comparable<T>. Метод compareTo созданного класса возвращает среднее арифметическое результатов сравнения всех трех вложенных объектов. Создать два объекта для разработанного обобщенного класса для хранения трех объектов класса Long, распечатать результат сравнения двух объектов между собой.

**private static void** task1() {  
 Task1<Long> task1\_1 = **new** Task1<>(1L, 2L, 3L);  
 Task1<Long> task1\_2 = **new** Task1<>(1L, 2L, 4L);  
 *//System.out.println(task1\_1.compareTo(task1\_2));*}

**public class** Task1<T **extends** Comparable<T>> **implements** Comparable<T> {  
 **private** T **a**;  
 **private** T **b**;  
 **private** T **c**;  
  
 **public** Task1(T a, T b, T c) {  
 **this**.**a** = a;  
 **this**.**b** = b;  
 **this**.**c** = c;  
 }  
  
 @Override  
 **public int** compareTo(T o) {  
 **return** (**a**.compareTo(o) + **b**.compareTo(o) + **c**.compareTo(o)) / 3;  
 }

}

2.Разработать обобщенный класс MyStack, реализующий функциональность стека (крайний элемент, помещенный в стек, при чтении возвращается первым). Кроме методов push(E elem) и E pop(), помещающих элемент в стек и извлекающих его, реализовать методы копирования уже существующего стека (отдельный метод и через конструктор), выбрасывания исключения при попытке извлечения элемента из пустого стека, проверки количества элементов в стеке, реализация интерфейса Comparable<T> (стек больше, если в нем больше элементов; если одинаково - элементы сравниваются, начиная с крайнего помещенного, результат для стека - compareTo для первого элемента, отличающееся от нуля; в противном случае стеки равны), методы equals, hashCode и toString(). Если стек реализуется через массив, а не связанный список, предусмотреть метод задания размера для внутреннего хранилища.

**public class** MyStack<T **extends** Number> {  
 **private** Node<? **extends** Number> **head**;  
 **private int size** = 0;  
  
 **public** MyStack() {  
  
 }  
  
 **public** MyStack(MyStack<? **extends** Number> stack) {  
 MyStack<T> res = **new** MyStack<>();  
 Node<? **extends** Number> current = stack.**head**;  
 **while** (current != **null**) {  
 res.push(current.**data**);  
 current = current.**next**;  
 }  
 current = res.**head**;  
 **while** (current != **null**) {  
 **this**.push(current.**data**);  
 current = current.**next**;  
 }  
 }  
  
 **public void** push(Number data) {  
 **if** (**head** == **null**) {  
 **head** = **new** Node<>(data, **null**);  
 } **else** {  
 **head** = **new** Node<>(data, **head**);  
 }  
 **size**++;  
 }  
  
 **public** Number pop() {  
 Number result = **head**.**data**;  
 **head** = **head**.**next**;  
 **size**--;  
 **return** result;  
 }  
  
 **public** MyStack<T> copy() {  
 MyStack<T> result = **new** MyStack<>();  
 MyStack<T> res = **new** MyStack<>();  
 Node<? **extends** Number> current = **head**;  
 **while** (current != **null**) {  
 res.push(current.**data**);  
 current = current.**next**;  
 }  
 current = res.**head**;  
 **while** (current != **null**) {  
 result.push(current.**data**);  
 current = current.**next**;  
 }  
 **return** result;  
 }  
  
 **public void** addAll(MyStack<? **extends** Number> stI) {  
 Node<? **extends** Number> current = stI.**head**;  
 **while** (current != **null**) {  
 **this**.push(current.**data**);  
 current = current.**next**;  
 }  
 }  
  
 **public void** moveElementsTo(MyStack<? **extends** Number> stN) {  
 Node<? **extends** Number> current = **head**;  
 **while** (current != **null**) {  
 stN.push(current.**data**);  
 current = current.**next**;  
 }  
 }  
  
 **private static class** Node<N **extends** Number> {  
 N **data**;  
 Node<? **extends** Number> **next**;  
  
 **public** Node(N data, Node<? **extends** Number> next) {  
 **this**.**data** = data;  
 **this**.**next** = next;  
 }  
 }  
  
 @Override  
 **public** String toString() {  
 StringJoiner stringJoiner = **new** StringJoiner(**", "**, MyStack.**class**.getSimpleName() + **"["**, **"]"**);  
 Node<? **extends** Number> current = **head**;  
 List<Number> list = **new** ArrayList<>();  
 **while** (current != **null**) {  
 list.add(current.**data**);  
 current = current.**next**;  
 }  
 **for** (**int** i = list.size() - 1; i >= 0; i--) {  
 stringJoiner.add(String.*valueOf*(list.get(i)));  
 }  
 **return** stringJoiner.toString();  
 }  
  
 @Override  
 **public boolean** equals(Object o) {  
 **if** (**this** == o) **return true**;  
 **if** (o == **null** || getClass() != o.getClass()) **return false**;  
 MyStack<T> myStack = (MyStack<T>) o;  
 **if** (**size** != myStack.size()) **return false**;  
 Node<? **extends** Number> current = **head**;  
 Node<? **extends** Number> myCurrent = myStack.**head**;  
 **while** (current != **null**) {  
 **if** (current.**data** != myCurrent.**data**) {  
 **return false**;  
 }  
 current = current.**next**;  
 myCurrent = myCurrent.**next**;  
 }  
 **return true**;  
 }  
  
 **private int** size() {  
 **return size**;  
 }  
  
 @Override  
 **public int** hashCode() {  
 **return** Objects.*hash*(**head**, **size**);  
 }  
}

**private static void** task2() {  
 MyStack<Integer> stack = **new** MyStack<>();  
 stack.push(1);  
 stack.push(2);  
 stack.push(3);  
 System.***out***.println(stack);  
 System.***out***.println(stack.pop());  
 System.***out***.println(stack);  
 System.***out***.println(stack.copy());  
 System.***out***.println(**new** MyStack<>(stack));  
 System.***out***.println(stack == stack.copy());  
 System.***out***.println(stack.equals(stack.copy()));  
}

3.Расширить реализацию MyStack для возможности использования иерархии классов (в один и тот же объект стека можно поместить и Integer, и Long, и Double). Убрать реализацию интерфейса Comparable<>.

Разработать методы, перемещающие все элементы:

в текущий из другого стека (этот стек может содержать элементы класса, унаследованного от класса элементов текущего стека);

из текущего - в другой (этот стек может содержать элементы суперкласса относительно класса элементов текущего стека).

**private static void** task3() {  
 MyStack<Number> stN = **new** MyStack<>();  
 MyStack<Integer> stI = **new** MyStack<>();  
 MyStack<Double> stD = **new** MyStack<>();  
  
 **for**(**int** i = 0; i < 3; i++) {  
 stI.push( **new** Integer( i ) );  
 stD.push( **new** Double( i + 3 ) );  
 }  
  
 System.***out***.println( **"Integer stack: "** + stI.toString() );  
 System.***out***.println( **"Double stack: "** + stD.toString() );  
  
 stN.addAll( stI );  
 stD.moveElementsTo( stN );  
  
 System.***out***.println( **"Number stack: "** + stN.toString() );  
}