МИНИСТЕРСТВО ТРАНСПОРТА РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «РОССИЙСКИЙ УНИВЕРСИТЕТ ТРАНСПОРТА (МИИТ)»

Кафедра УТБиИС

**Отчет**

По курсовой работе по теме:

«Структуры данных»

Дисциплина:

«Структуры и алгоритмы обработки данных»

Выполнил:

Студент группы УВП-311,

Новопашенный М. В.

Проверил:

Заманов Е.А.

Москва 2020 г.

Содержание:

1. Описание курсовой работы
2. Листинг кода
3. Вывод
4. **Описание курсовой работы**

В курсовой работе я реализовал структуру данных Двусвязный список (DoublyLinkedList). В качестве рода деятельности я выбрал завод запчастей для автомобилей. В курсовой работе реализован отправку детали на изготовление, проверку на исправность, отправку на конвейер исправных деталей.. В изготовительный станок вводится название детали и ее размер. Некоторые детали после изготовления оказываются неисправными и на конвейер в склад отправляются только исправные детали.

1. **Листинг кода программы**

**public class** DoublyLinkedList<T> {  
 **private** Node<Detail> **head**;  
 **private** Node<Detail> **last**;  
 **private int size**;  
  
 **public** DoublyLinkedList() {  
 }  
  
  
 **public void** addElement(Detail element) {  
 Node<Detail> node = **new** Node<Detail>(element);  
 **if** (**size** == 0) {  
 **head** = node;  
 } **else** {  
 **last**.setNext(node);  
 node.setPrevious(**last**);  
 }  
 **last** = node;  
  
 **size**++;  
 }  
  
  
 **public** Node<Detail> getNode(**int** index) {  
 Node<Detail> current = **null**;  
 **if** (index >= 0 || index < **size**) {  
 current = **head**;  
 **int** k = 0;  
 **while** (k < index) {  
 current = current.getNext();  
 k++;  
 }  
 }  
 **return** current;  
 }  
  
 **private** Detail getElement(**int** index) {  
 **return** getNode(index).getElement();  
 }  
  
  
 **public void** insertBefore(Detail element, **int** index) {  
  
 Node<Detail> node = **null**;  
  
 **if** (index == **size**) {  
 addElement(element);  
 **return**;  
 }  
 node = **new** Node<Detail>(element);  
  
 **if** (index == 0) {  
 node.setNext(**head**);  
 **head**.setPrevious(node);  
 **head** = node;  
 **size**++;  
 **return**;  
 }  
  
 Node<Detail> current = getNode(index);  
  
 current.getPrevious().setNext(node);  
 node.setPrevious(current.getPrevious());  
  
 node.setNext(current);  
 current.setPrevious(node);  
  
 **size**++;  
 }  
  
 **public void** removeAtIndex(**int** index) {  
 Node<Detail> node = getNode(index);  
  
 **if** (**head** == node) {  
 **if** (**head**.getNext() != **null**) {  
 **head**.getNext().setPrevious(**null**);  
 **head** = **head**.getNext();  
 } **else** {  
 **head** = **null**;  
 }  
 } **else if** (**last** == node) {  
 **if** (**last**.getPrevious() != **null**) {  
 **last**.getPrevious().setNext(**null**);  
 **last** = **last**.getPrevious();  
 } **else** {  
 **last** = **null**;  
 }  
 } **else** {  
  
 node.getPrevious().setNext(node.getNext());  
 node.getNext().setPrevious(node.getPrevious());  
 }  
 **size**--;  
 }  
  
 **private void** setElement(Detail val, **int** index) {  
 getNode(index).setElement(val);  
 }  
  
 @Override  
 **public** String toString() {  
 Node<Detail> current = **head**;  
 StringBuilder s = **new** StringBuilder();  
 **while** (current != **null**) {  
 s.append(current.getElement().toString()).append(**", "**);  
 current = current.getNext();  
 }  
  
 **return** String.*format*(**"Количество деталей: %d , Детали:\n%s"**, **size**, s.toString());  
 }  
  
 **public** Node<Detail> getHead() {  
 **return head**;  
 }  
  
 **public** Node<Detail> getLast() {  
 **return last**;  
 }  
  
 **public int** getSize() {  
 **return size**;  
 }  
}

**public class** Node<T> {  
 **private** Node<Detail> **previous**;  
 **private** Node<Detail> **next**;  
 **private** Detail **element**;  
  
 **public** Node(Detail element) {  
 **this**.**element** = element;  
 }  
  
 **public** Node<Detail> getPrevious() {  
 **return previous**;  
 }  
  
 **public void** setPrevious(Node<Detail> previous) {  
 **this**.**previous** = previous;  
 }  
  
 **public** Node<Detail> getNext() {  
 **return next**;  
 }  
  
 **public void** setNext(Node<Detail> next) {  
 **this**.**next** = next;  
 }  
  
 **public** Detail getElement() {  
 **return element**;  
 }  
  
 **public void** setElement(Detail element) {  
 **this**.**element** = element;  
 }  
}

**public class** Detail {  
 **private final** String **title**;  
 **private final int size**;  
 **private int index**;  
 **private boolean isBroken** = **false**;  
  
 **public void** setIndex(**int** index) {  
 **this**.**index** = index;  
 }  
  
 **public** Detail(String title, **int** size) {  
 **this**.**title** = title;  
 **this**.**size** = size;  
 }  
  
 **public int** getIndex() {  
 **return index**;  
 }  
  
  
 **public** String getTitle() {  
 **return title**;  
 }  
  
 **public int** getSize() {  
 **return size**;  
 }  
  
 **public boolean** isBroken() {  
 **return isBroken**;  
 }  
  
 **public void** setBroken(**boolean** broken) {  
 **isBroken** = broken;  
 }  
  
 @Override  
 **public** String toString() {  
 **return "Title - "** + **title** +  
 **", Size - "** + **size** +  
 **", Broken - "** + **isBroken** + **"\n"**;  
 }  
  
}

**public class** Factory **implements** Iterable<Detail> {  
 **private final** String **name**;  
 **private final int rating**;  
 DoublyLinkedList<Detail> **details**;  
 **int i**=0;  
  
 **public** Factory(String name, **int** rating) {  
 **this**.**name** = name;  
 **this**.**rating** = rating;  
 **details**=**new** DoublyLinkedList<Detail>();  
 }  
  
 **public void** addDetailToQueue(Detail detail) {  
 detail.setIndex(detail.getIndex()+**i**);  
 **details**.addElement(detail);  
 **i**++;  
 System.***out***.printf(**"Деталь %s с размером %d отправлена на производство%n"**, detail.getTitle(), detail.getSize());  
 }  
  
 **public void** checkAfterManufacture(Detail detail) {  
 Random random=**new** Random();  
 detail.setBroken(random.nextBoolean());  
  
 System.***out***.printf(**"Деталь %s с размером %d изготовлена. Поломка - %s\n"**, detail.getTitle(), detail.getSize(), detail.isBroken());  
 **if**(detail.isBroken()){  
 **details**.removeAtIndex(detail.getIndex());  
 **i**--;  
 }  
 }  
  
  
  
  
 @Override  
 **public** Iterator<Detail> iterator() {  
 DoublyLinkedList<Detail> list = **details**;  
 **final int**[] i = {-1};  
 **return new** Iterator<Detail>() {  
 @Override  
 **public boolean** hasNext() {  
 **return** list.getNode(i[0]).getNext() != **null**;  
 }  
  
 @Override  
 **public** Detail next() {  
 Detail element = list.getNode(i[0]).getNext().getElement();  
 i[0]++;  
 **return** element;  
 }  
 };  
 }  
  
  
}

**public class** Main {  
  
 **public static void** main(String[] args) {  
 Detail detail1 = **new** Detail(**"motor"**, 10);  
 Detail detail2 = **new** Detail(**"wheel"**, 2);  
 Detail detail3 = **new** Detail(**"gear"**, 5);  
 Detail detail4 = **new** Detail(**"switcher"**, 1);  
 Detail detail5 = **new** Detail(**"stick"**, 3);  
 Detail detail6 = **new** Detail(**"bumper"**, 6);  
 Detail detail7 = **new** Detail(**"fender"**, 3);  
 Factory factory = **new** Factory(**"Tesla motors"**, 10);  
  
 factory.addDetailToQueue(detail1);  
 factory.checkAfterManufacture(detail1);  
  
 factory.addDetailToQueue(detail2);  
 factory.checkAfterManufacture(detail2);  
  
 factory.addDetailToQueue(detail3);  
 factory.checkAfterManufacture(detail3);  
  
 factory.addDetailToQueue(detail4);  
 factory.checkAfterManufacture(detail4);  
  
 factory.addDetailToQueue(detail5);  
 factory.checkAfterManufacture(detail5);  
  
 factory.addDetailToQueue(detail6);  
 factory.checkAfterManufacture(detail6);  
  
 factory.addDetailToQueue(detail7);  
 factory.checkAfterManufacture(detail7);  
  
 System.***out***.println(factory.**details**);  
  
 }  
}

Скачать код можно по ссылке: <https://github.com/comougi/conveyor-comparator>

1. **Вывод:**

В результате выполнения курсового проекта, я работал с двусвязным списком. У этой структуры данных есть преимущества и недостатки. Я считаю, что основным преимуществом двусвязного списка является быстрое внесение изменений в середину списка. Динамический массив теоретически мог вызывать смещение каждого элемента, а связанный список сохраняет каждый другой объект на своём месте. Тем не менее двусвязный список имеет ряд недостатков. Один из них это то, что элементы в памяти соединены в цепочку. Если нам нужен определенный элемент, то мы идем по цепочке от первого элемента к искомому. Это нагружает память.