**Министерство науки и высшего образования Российской Федерации**

**федеральное государственное бюджетное образовательное учреждение высшего образования «Казанский национальный исследовательский технический университет им. А.Н.Туполева-КАИ»**

«Сomputer-центр»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

по дисциплине

«Структуры и алгоритмы обработки данных»

Выполнил:

студент гр. №4280

Салимов И.Г.

Проверил(а): Шакирзянов Р.М.

Казань 2021

**Содержание:**

[Постановка задачи 3](#_Toc62723602)

[Теоретическое описание используемых структур данных с алгоритмами реализации основных операций 5](#_Toc62723603)

[Описание основных понятий и механизмов ООП 9](#_Toc62723604)

[Описание всех разработанных классов (объектная модель) 11](#_Toc62723605)

[Описание демонстрационного модуля с характеристикой использованных стандартных компонентов 18](#_Toc62723606)

[Листинг программы 23](#_Toc62723607)

[Список использованной литературы 26](#_Toc62723608)

Постановка задачи

Цель работы: получение навыков разработки объектных программ, включая создание набора собственных взаимосвязанных классов для объектной реализации специализированного контейнера. Контейнер предназначен для хранения и обработки данных некоторой информационной задачи. Контейнер представляет собой двухуровневую структуру данных, в которой уровни реализуются разными способами – один статически на базе массива (непрерывная реализация), другой – динамически с использованием адресных связей (связная реализация).

Исходные данные:

Объектная реализация контейнера на основе комбинированной структуры «Двунаправленный список массивов-стеков».

Задача «Производственное предприятие»

* Информационные объекты: оборудование цеха предприятия (свойства – НаименованиеОборудования, ГодВыпуска)
* Оборудование объединяется в рамках объекта Цех (свойство – НомерЦеха)
* Цеха объединяются в рамках объекта-контейнера Предприятие (свойство – Название)

Требования

1. Полная объектная реализация с определением классов для всех элементов реализуемой структуры: информационные объекты (обязательно!), объекты-элементы списка (динамическая реализация), объекты-списки, объект-контейнер
2. Имена классов, свойств и методов должны носить содержательный смысл и соответствовать информационной задаче
3. Соблюдение принципа инкапсуляции – использование в классах только закрытых свойств и реализация необходимого набора методов доступа
4. Реализация в классах всех необходимых методов: конструкторы, методы доступа к свойствам, методы добавления и удаления на каждом из двух уровней, метод поиска (при необходимости)
5. Возможность сохранения всей структуры во внешнем файле с обратной загрузкой
6. Наличие модуля, демонстрирующего все возможности созданной библиотеки классов и обладающего удобным оконным пользовательским интерфейсом
7. Язык и среда разработки – по выбору: Delphi, Java, C++, С#

Теоретическое описание используемых структур данных с алгоритмами реализации основных операций

Двунаправленный список. Здесь ссылки в каждом узле указывают на предыдущий и на последующий узел в списке. Как и односвязный список, двусвязный допускает только последовательный доступ к элементам, но при этом дает возможность перемещения в обе стороны. В этом списке проще производить удаление и перестановку элементов, так как легко доступны адреса тех элементов списка, указатели которых направлены на изменяемый элемент. Стандартный набор операций со списком включает:

* добавление нового элемента после заданного или перед заданным элементом с проверкой возможности добавления элемента
* удаление заданного элемента
* проход по списку от первого элемента к последнему и наоборот с выполнением заданных действий
* поиск в списке заданного элемента

Сам двунаправленный список состоит из хранимый сущности – в моем случае это Company под название Data, ссылки на первый элемент – Head, последнего элемента – Tail и количества элементов в самом списке – Count.

**Создание пустого списка** включает:

* выделение памяти под сам объект;
* ссылки остаются пустыми, т.е. **null**

**Поиск заданного элемента** включает:

* установку вспомогательного указателя на «голову» списка
* проход по всей коллекции до указанного, переходя на ссылку последующего элемента
* при нахождении соответствия немедленный выход и возвращение ссылки найденного элемента

**Обратный поиск элемента** включает:

* установку вспомогательного указателя на «хвост» списка
* проход по всей коллекции до заданного элемента, переходя на предыдущий элемент
* при нахождении заданного элемента немедленный выход и возвращение ссылки найденного элемента

**Удаление заданного элемента** включает:

* поиск удаляемого элемента по заданным параметрам
* переопределение ссылок на удаляемый элемент у предыдущего и последующего элемента на друг друга, для удаления ссылок на удаляемый элемент и добавления ссылок для исключения разрыва между элементами
* уменьшение переменной, хранящей число количества элементов в списке

**Добавление** нового элемента после заданного включает:

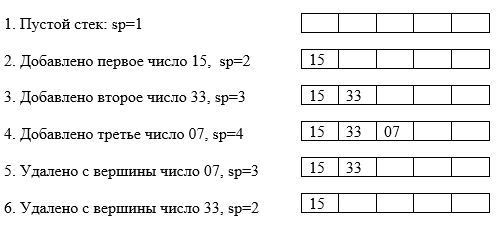
* поиск элемента по заданным параметрам
* переопределение ссылок у найденного и последующего (если он есть) элементов на следующий и предыдущий соответственно на новый элемент
* увеличение переменной, хранящей число количества элементов в списке

**Добавление** нового элемента перед заданным включает:

* обратный поиск заданного элемента
* переопределение ссылок у найденного и предыдущего (если он есть) элементов на предыдущий и следующий соответственно на новый элемент
* увеличение переменной, хранящей число количества элементов в списке

Стек – это класс представляющий собой список элементов, в данном случае на основе массива, организованных по принципу LIFO (last in – last out, «последним пришел – первым вышел»). Для примера можно взять стопку тарелок: чтобы взять самую нижнею, нужно сначала взять все, что над ней.

Для реализации статического стека нужно объявить массив элементов, переменную, хранящую число элементов в этом стеке, и сделать методы для добавления в конец списки и удаления из конца списка. Так же можно сделать метод для получения последнего элемента.



Само **добавление элемента** в стек выполняется следующим образом:

* проверить возможность добавления (в массиве есть свободные ячейки?)
* добавить элемент в массив по переменной количества элементов
* увеличить переменную количества элементов на один

**Удаление элемента** из стека:

* проверить возможность удаления (в очереди есть элементы?)
* извлечь последний элемент из стека
* уменьшить переменную количества элементов в стеке

Описание основных понятий и механизмов ООП

**Объект** – это структурированная переменная, содержащая всю информацию о некотором физическом предмете или реализуемом в программе понятии.

**Класс** – это описание множества объектов программирования (объектов) и выполняемых над ними действий.

**Метод** – это функция, получающая в качестве обязательного параметра указатель на объект и выполняющие определенные действия с данными объекта программирования.

**Инкапсуляция** – это механизм, который объединяет данные и код, манипулирующий с этими данными, а также защищает и то, и другое от внешнего вмешательства или неправильного использования. В ООП код и данные могут быть объединены вместе (в так называемый «черный ящик») при создании объекта.

Внутри объекта коды и данные могут быть закрытыми или открытыми.

Закрытые коды или данные доступны только для других частей того же самого объекта и, соответственно, недоступны для тех частей программы, которые существуют вне объекта.

Открытые коды и данные, напротив, доступны для всех частей программы, в том числе и для других частей того же самого объекта.

**Наследование**. Новый, или производный класс может быть определен на основе уже имеющегося, или базового класса.

При этом новый класс сохраняет все свойства старого: данные объекта базового класса включаются в данные объекта производного, а методы базового класса могут быть вызваны для объекта производного класса,

причем они будут выполняться над данными включенного в него объекта базового класса

Иначе говоря, новый класс наследует как данные старого класса, так и методы их обработки.

Если объект наследует свои свойства от одного родителя, то говорят об одиночном наследовании. Если объект наследует данные и методы от нескольких базовых классов, то говорят о множественном наследовании.

**Полиморфизм** – это свойство, которое позволяет один и тот же идентификатор (одно и то же имя) использовать для решения двух и более схожих, но технически разных задач.

Целью полиморфизма, применительно к ООП, является использование одного имени для задания действий, общих для ряда классов объектов. Такой полиморфизм основывается на возможности включения в данные объекта также и информации о методах их обработки (в виде указателей на функции).

Будучи доступным в некоторой точке программы, объект, даже при отсутствии полной информации о его типе, всегда может корректно вызвать свойственные ему методы.

Описание всех разработанных классов (объектная модель)

Классы для информационных объектов:

Company - класс предприятие.

Приватные свойства:

1. static String DefaultNameCompany – статическая переменная «Default» для присваивания дефолтного названия компании
2. String Name – название компании

Методы:

1. Company() – конструктор класса, в котором наименованию предприятия присваивается дефолтное значение, описанное выше
2. Company(String name) – конструктор класса, в котором можно присвоить произвольное значение переменной Name
3. String getName() – геттер для приватного свойства Name
4. Void setName(String name) – сеттер для приватного свойства Name

CompanyExport – вспомогательный класс для выгрузки предприятия в файл

Приватные свойства:

1. Company Data – само предприятие
2. WorkShopStackEquipments Head – «голова» списка
3. WorkShopStackEquipments Tail – «хвост» списка
4. ArrayList<WorkShopStackEquipments> WorkShops – список цехов
5. Int Count – количество цехов

Методы:

1. CompanyExport() – стандартный конструктор
2. CompanyExport(DoublyListWorkShops company) – конструктор для заполнения приватных свойств из передаваемого значения для выгрузки в файл

А также геттеры и сеттеры для приватных свойств

DoublyItemWorkShop<T> – класс элемента двусвязного списка

Приватные свойства:

1. T Data – хранимые данные, реализовано через обобщения для гибкости
2. WorkShop workshop – объект цеха
3. DoublyItemWorkShop<T> Next – ссылка на следующий элемент
4. DoublyItemWorkShop<T> Previous – ссылка на предыдущий элемент

Методы:

1. DoublyItemWorkShop() – стандартный конструктор
2. DoublyItemWorkShop(T data) – конструктор, присваивающий свойству Data, передаваемое значение
3. DoublyItemWorkShop(WorkShop workshop, T data) – конструктор, присваивающий значение свойствам workShop и Data
4. Int getWorkShopNumber() – геттер для получения номера цеха
5. Void setWorkShopNumber(int number) – сеттер для установки номера цеха

А также геттеры и сеттеры для приватных свойств.

Equipment – класс оборудования

Приватные свойства:

1. String Name – наименование оборудования
2. Int YearOfIssue – год выпуска оборудования

Методы:

1. Equipment() – стандартный конструктор
2. Equipment(String name, int yearOfIssue) – конструктор для присваивания свойствам Name и YearOfIssue

А также геттеры и сеттеры для приватных свойств.

WorkShop – класс цеха

Приватные свойства:

1. Int Number – номер цеха

Методы:

1. WorkShop() – стандартный конструктор
2. WorkShop(int number) – конструктор для присваивания свойства Number
3. Int getNumber() – геттер для свойства Number
4. Void setNumber(int number) – сеттер для свойства Number

WorkShopStackEquipments – вспомогательный класс для выгрузки цеха и оборудования в файл

Приватные свойства:

1. WorkShop workshop – цех
2. StackEquipment equipments – оборудования в данном цеху

Методы:

1. WorkShopStackEquipments() – стандартный конструктор
2. WorkShopStackEquipments(WorkShop workshop, StackEquipment equipments) – конструктор для присваивания значений свойствам workshop и equipments соответственно
3. WorkShop getWorkShop() – геттер для свойтва workshop
4. Void setWorkShop(WorkShop workShop) – сеттер для свойтва workshop
5. StackEquipments getEquipments() – геттер для свойтва equipments
6. Void setEquipments(StackEquipments equipments) – сеттер для свойства equipments

DoublyListWorkShops – класс двусвязного списка

Приватные методы:

1. Company Data – объект предприятия
2. DoublyItemWorkShop<StackEquipments> Head – «голова» списка
3. DoublyItemWorkShop<StackEquipments> Tail – «хвост» списка
4. Int Count – количество элементов в списке

Методы:

1. DoublyListWorkShops() – стандартный конструктор
2. DoublyListWorkShops(CompanyExport companyExport) – конструктор для извлечения из файла данных
3. DoublyItemWorkShop<StackEquipments> Find(int numberWorkShop) – поиск заданного элемента
4. DoublyItemWorkShop<StackEquipments> FindReverse – обратный поиск заданного элемента
5. Void Add(int numberWorkShop, in lengthWorkShop) – добавление нового элемента в конец списка
6. Void AddAfter(int numberWorkShop, int lengthWorkShop, int numberFindWorkShop) – добавление нового элемента после заданного
7. Void AddBefore(int numberWorkShop, int lengthWorkShop, int numberFindWorkShop) – добавление нового элемента перед заданным
8. private DoublyItemWorkShop<StackEquipments> InitDoublyItemWorkShop(int numberWorkShop, int lengthWorkShop) – приватный метод для создания экземпляра класса элемента списка
9. void AddInWorkShop(int numberWorkShop, Equipment equipment) – добавление оборудования в заданный цех
10. DoublyItemWorkShop<StackEquipments> Remove(int numberWorkShop) – удаление заданного цеха
11. Equipment RemoveInWorkShop(int numberWorkShop) – удаление оборудования из заданного цеха

А также геттеры и сеттеры для приватных свойств.

StackEquipments – класс стека оборудования на основе массива

Приватные свойства:

1. Static int DefaultSize – дефолтная размерность стека (10)
2. Equipment[] items – массив оборудований
3. Int count – количество элементов в массиве

Методы:

1. StackEquipments() – стандартный конструктор, который задает дефолтный размер массиву
2. StackEquipments(int length) – конструктор, при помощи которого можно задать размер массива
3. Boolean IsEmpty() – проверяет не пустой ли массив
4. Boolean IsFull() – проверяет не полный ли массив
5. Void Push(Equipment equipment) – добавляет новый элемент в конец массива
6. Equipment Pop() – удаляет последний элемент в массиве
7. Equipment Peek() – возвращает последний элемент массива
8. ArrayList<Equipment> GetList() – возвращает массив в виде списка

А также геттеры и сеттеры для приватных свойств.

MainService – класс для взаимодействия с пользователем

Приватные свойства:

1. Static String FileName – путь к файлу для выгрузки

Публичные методы:

1. Void Main() – главная функция для работы, здесь обрабатывается выбор пользователя

Приватные методы:

1. void AddWorkShopAfter(DoublyListWorkShops company) – добавление цеха после заданного
2. void AddWorkShopBefore(DoublyListWorkShops company) – добавление цеха до заданного
3. DoublyListWorkShops LoadFromFile() – чтение из файла
4. void SaveInFile(DoublyListWorkShops company) – сохранение в файл
5. void RenameCompany(DoublyListWorkShops company) – Переименование предприятия
6. void RemoveEquipment(DoublyListWorkShops company) – удаление оборудования из заданного цеха
7. void AddEquipment(DoublyListWorkShops company) – добавление оборудования в заданный цех
8. void RemoveWorkShop(DoublyListWorkShops company) – удаление заданного цеха
9. void AddWorkShop(DoublyListWorkShops company) – добавление нового цеха
10. void WriteCompany(DoublyListWorkShops company) – вывод всей структуры предприятия в консоль
11. int GetAnswerInt() – получение ввода пользователя с консоли типа int
12. String GetAnswerString() – получения ввода пользователя с консоли типа String
13. void WriteMenu() – вывод возможных действий пользователя на экран
14. DoublyListWorkShops InitCompany() – инициализация предприятия

Program – класс для точки входа в приложение

Публичные методы:

1. Static void main(String args[]) – точка входа в приложение

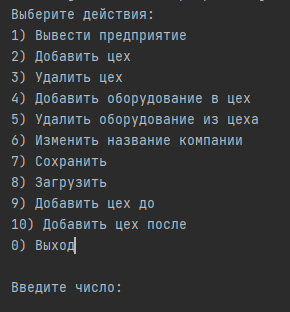
Приватные методы:

1. Static void RunMainService() – метод для запуска главной функции для работы

Описание демонстрационного модуля с характеристикой использованных стандартных компонентов

Демонстрационный модуль реализован в виде консольной программы, которая позволяет пользователю взаимодействовать со структурой.

После запуска программы выводится данное окно с меню для взаимодействия:

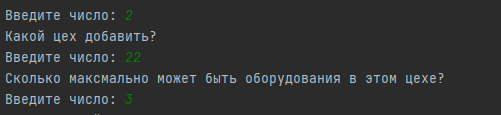


Можно вывести структуру предприятия для этого нужно ввести «1»:



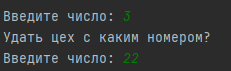
Изначально предприятие не имеет ни цехов, ни оборудования.

Чтобы добавить цех нужно ввести «2»:



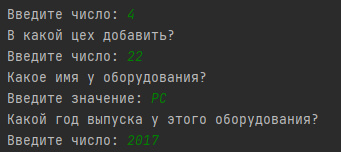
После этого нужно ввести номер цеха, а затем количество оборудования, которое максимально может быть в этом цеху.

Для удаления цеха нужно ввести «3»:



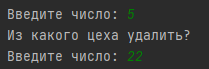
Здесь нужно ввести номер цеха, который вы хотите удалить.

Для добавления оборудования в цех нужно ввести «4»:



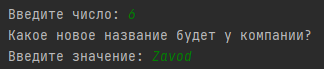
Далее ввести номер цеха, в который вы хотите добавить новое оборудование, наименование оборудования и его год выпуска.

Для удаления оборудования из цеха нужно ввести «5»:



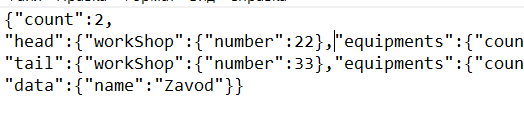
Здесь нужно ввести номер цеха, из которого вы хотите удалить оборудование.

Для того, чтобы изменить наименование предприятия нужно ввести «6»:

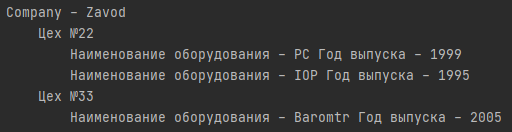


Далее ввести новое наименование для предприятия.

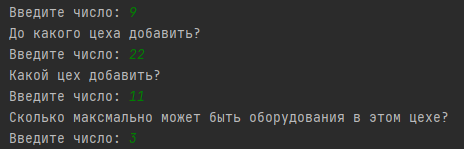
Для сохранения в файл нужно ввести «7». Никаких больше дополнительных действий не нужно. Данные сохранились в файл в виде json:



Для выгрузки из файла нужно ввести «8». Данные из файла загружаются в оперативную память и далее можно вывести их на экран:

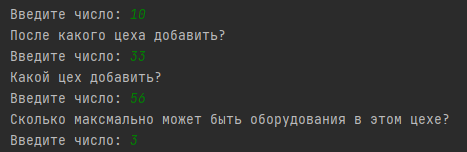


Для добавления нового цеха до существующего нужно ввести «9»:



Далее нужно ввести номер цеха, до которого нужно добавить, номер нового цеха и его размер.

Для того, чтобы добавить новый цех после существующего нужно ввести «10»:



После этого нужно ввести номер цеха, после которого нужно добавить новый цех, номер нового цеха и его размер.

Для выхода из программы нужно ввести «0».

Листинг программы

Program.java – главный файл, использующийся как точка входа:

package main;

import main.Services.MainService;

public class Program {

public static void main(String args[]){

RunMainService();

}

private static void RunMainService(){

var service = new MainService();

service.Main();

}

}

MainService.java – сервис для взаимодействия с пользователем:

package main.Services;

import com.fasterxml.jackson.core.JsonGenerationException;

import com.fasterxml.jackson.databind.DeserializationFeature;

import com.fasterxml.jackson.databind.ObjectMapper;

import resources.Collections.DoublyListWorkShops;

import resources.Collections.StackEquipments;

import resources.Entities.\*;

import java.io.\*;

import java.util.Scanner;

public class MainService {

private static String FileName = "D://MyDevelop//Java//outObject.txt";

public void Main(){

var company = InitCompany();

boolean exit = false;

while (true){

WriteMenu();

var answer = GetAnswerInt();

switch (answer){

case 1:

WriteCompany(company);

break;

case 2:

AddWorkShop(company);

break;

case 3:

RemoveWorkShop(company);

break;

case 4:

AddEquipment(company);

break;

case 5:

RemoveEquipment(company);

break;

case 6:

RenameCompany(company);

break;

case 7:

SaveInFile(company);

break;

case 8:

var companyNew = LoadFromFile();

if(companyNew != null)

company = companyNew;

break;

case 9:

AddWorkShopBefore(company);

break;

case 10:

AddWorkShopAfter(company);

break;

case 0:

exit = true;

break;

default:

System.out.println("Такого значения нет!");

break;

}

if(exit)

break;

}

}

private void AddWorkShopAfter(DoublyListWorkShops company){

System.out.println("После какого цеха добавить?");

var numberFindWorkShop = GetAnswerInt();

System.out.println("Какой цех добавить?");

var numberWorkShop = GetAnswerInt();

System.out.println("Сколько максмально может быть оборудования в этом цехе?");

var lengthWorkShop = GetAnswerInt();

if(lengthWorkShop < 1){

System.out.println("Минимальное количество оборудования в цехе - 1!");

return;

}

company.AddAfter(numberWorkShop, lengthWorkShop, numberFindWorkShop);

}

private void AddWorkShopBefore(DoublyListWorkShops company){

System.out.println("До какого цеха добавить?");

var numberFindWorkShop = GetAnswerInt();

System.out.println("Какой цех добавить?");

var numberWorkShop = GetAnswerInt();

System.out.println("Сколько максмально может быть оборудования в этом цехе?");

var lengthWorkShop = GetAnswerInt();

if(lengthWorkShop < 1){

System.out.println("Минимальное количество оборудования в цехе - 1!");

return;

}

company.AddBefore(numberWorkShop, lengthWorkShop, numberFindWorkShop);

}

private DoublyListWorkShops LoadFromFile(){

try {

var objectMapper = new ObjectMapper();

var companyExport = objectMapper.readValue(new File(FileName), CompanyExport.class);

var company = new DoublyListWorkShops(companyExport);

return company;

}

catch (JsonGenerationException e) {

e.printStackTrace();

}

catch (Exception ex){

System.out.println(ex.getMessage());

}

return null;

}

private void SaveInFile(DoublyListWorkShops company){

try {

CompanyExport companyExport = new CompanyExport(company);

var objectMapper = new ObjectMapper();

objectMapper.writeValue(new File(FileName), companyExport);

}

catch (JsonGenerationException e) {

e.printStackTrace();

}

catch (Exception ex){

System.out.println(ex.getMessage());

}

}

private void RenameCompany(DoublyListWorkShops company){

System.out.println("Какое новое название будет у компании?");

var companyName = GetAnswerString();

if(companyName == null)

return;

company.getCompany().setName(companyName);

}

private void RemoveEquipment(DoublyListWorkShops company){

System.out.println("Из какого цеха удалить?");

var numberWorkShop = GetAnswerInt();

company.RemoveInWorkShop(numberWorkShop);

}

private void AddEquipment(DoublyListWorkShops company){

System.out.println("В какой цех добавить?");

var numberWorkShop = GetAnswerInt();

System.out.println("Какое имя у оборудования?");

var equipmentName = GetAnswerString();

if(equipmentName == null)

return;

System.out.println("Какой год выпуска у этого оборудования?");

var equipmentYear = GetAnswerInt();

var length = (int) (Math.log10(equipmentYear) + 1);

if(length != 4){

System.out.println("Год должен состоять из четырех цифр!");

return;

}

var equipment = new Equipment(equipmentName, equipmentYear);

company.AddInWorkShop(numberWorkShop, equipment);

}

private void RemoveWorkShop(DoublyListWorkShops company){

System.out.println("Удать цех с каким номером?");

var answer = GetAnswerInt();

company.Remove(answer);

}

private void AddWorkShop(DoublyListWorkShops company){

System.out.println("Какой цех добавить?");

var numberWorkShop = GetAnswerInt();

System.out.println("Сколько максмально может быть оборудования в этом цехе?");

var lengthWorkShop = GetAnswerInt();

if(lengthWorkShop < 1){

System.out.println("Минимальное количество оборудования в цехе - 1!");

return;

}

company.Add(numberWorkShop, lengthWorkShop);

}

private void WriteCompany(DoublyListWorkShops company){

System.out.println("Company - " + company.getCompany().getName());

var currentWorkShop = company.getHead();

while (currentWorkShop != null){

System.out.println("\tЦех №" + currentWorkShop.getWorkShopNumber());

if(currentWorkShop.getData() != null && !currentWorkShop.getData().IsEmpty()) {

var equipments = currentWorkShop.getData().getEquipments();

for (var i = 0; i < equipments.length && equipments[i] != null; i++) {

System.out.println("\t\tНаименование оборудования - " + equipments[i].getName() +

" Год выпуска - " + equipments[i].getYearOfIssue());

}

}

currentWorkShop = currentWorkShop.getNext();

}

}

private int GetAnswerInt(){

var console = new Scanner(System.in);

System.out.print("Введите число: ");

var answer = console.nextInt();

return answer;

}

private String GetAnswerString(){

var console = new Scanner(System.in);

System.out.print("Введите значение: ");

var answer = console.nextLine();

if(answer == null || "".equals(answer.trim())){

System.out.println("Значение не может быть пустым!");

return null;

}

return answer;

}

private void WriteMenu(){

System.out.println("Выберите действия:\n" +

"1) Вывести предприятие\n" +

"2) Добавить цех\n" +

"3) Удалить цех\n" +

"4) Добавить оборудование в цех\n" +

"5) Удалить оборудование из цеха\n" +

"6) Изменить название компании\n" +

"7) Сохранить\n" +

"8) Загрузить\n" +

"9) Добавить цех до\n" +

"10) Добавить цех после\n" +

"0) Выход\n");

}

private DoublyListWorkShops InitCompany(){

var company = new DoublyListWorkShops();

company.setData(new Company());

return company;

}

}

Company.java – сущность «Предприятие»:

package resources.Entities;

public class Company {

private static String DefaultNameCompany = "Default";

private String Name;

public Company(){

Name = DefaultNameCompany;

}

public Company(String name){

Name = name;

}

public String getName(){

return Name;

}

public void setName(String name){

Name = name;

}

}

WorkShop.java – сущность «Цех»:

package resources.Entities;

public class WorkShop {

private int Number;

public WorkShop(){

}

public WorkShop(int number){

Number = number;

}

public int getNumber(){

return Number;

}

public void setNumber(int number){

Number = number;

}

}

Equipment.java – сущность «Оборудование»:

package resources.Entities;

public class Equipment {

private String Name;

private int YearOfIssue;

public Equipment(){

}

public Equipment(String name, int yearOfIssue){

Name = name;

YearOfIssue = yearOfIssue;

}

public String getName(){

return Name;

}

public void setName(String name){

Name = name;

}

public int getYearOfIssue(){

return YearOfIssue;

}

public void setYearOfIssue(int yearOfIssue){

YearOfIssue = yearOfIssue;

}

}

Список использованной литературы

1. Козин А. Н. Структуры и алгоритмы обработки данных. Учебно-методическое пособие. Издательство: Татарский институт содействия бизнесу, 2003.
2. Козин А. Н. Объектно-ориентированное программирование. Учебно-методическое пособие. Издательство: Академия Управления «ТИСБИ», 2006.