ПРАКТИЧЕСКАЯ РАБОТА № 2. ИССЛЕДОВАНИЕ

ВОЗМОЖНОСТЕЙ JAVA COLLECTION FRAMEWORK (JCF).

КАРТЫ ОТОБРАЖЕНИЙ

**1. Цель работы**

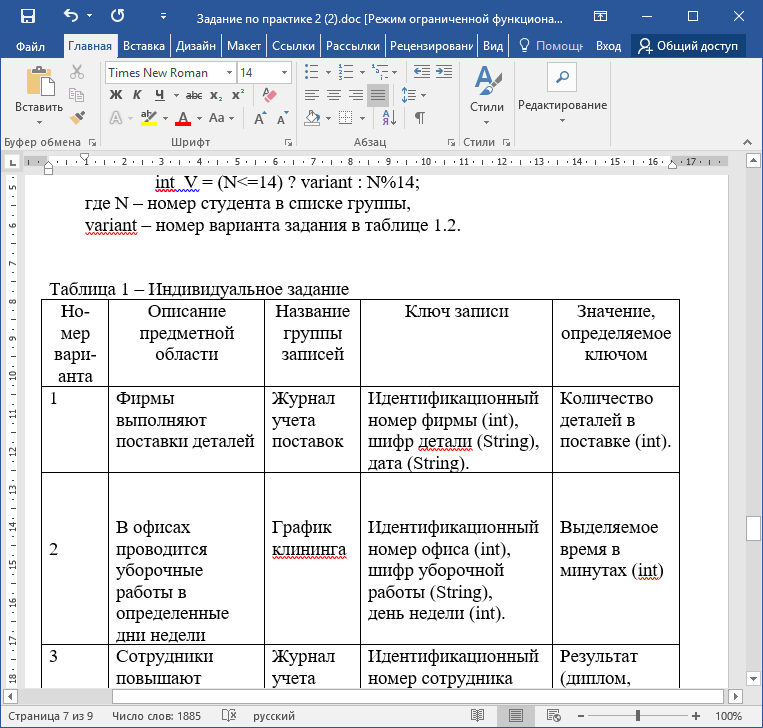
Целью данной работы является получение навыков использования набора классов и интерфейсов JAVA COLLECTION FRAMEWORK для создания карт отображений объектов-ключей в объекты-значения.

**2. Постановка задачи**

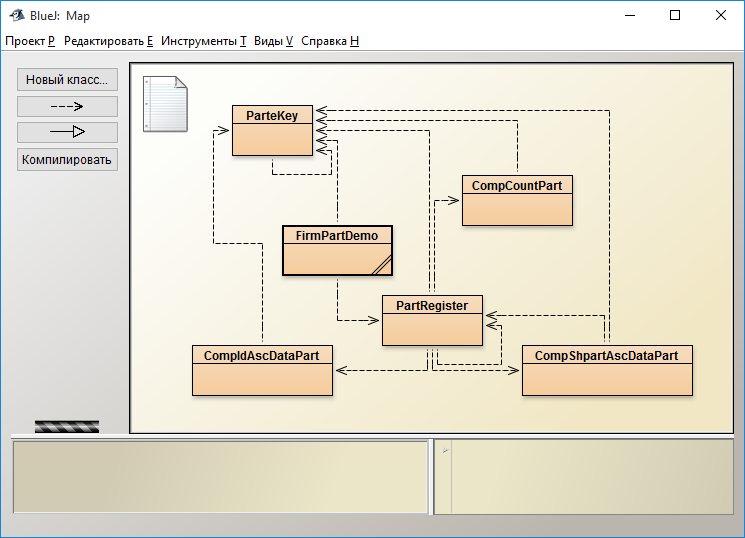
1) Разработать программу, реализующую взаимодействие объектов, образующих иерархическую структуру данных (группу объектов с заданными свойствами и функциями). Для создания объектов, определенных вариантом задания разработать соответствующие классы. Для манипулирования группой объектов использовать возможности интерфейса Map.

3) Сравнить возможности коллекций (списков и множеств) и карт отображений.

Индивидуальное задание (вариант №1): представлено в таблице 1.



**3. Структура проекта**



**4. Описание классов проекта (полей и методов, инкапсулированных в классах)**

Класс ParKey описывает ключ записи о фирме, содержит private-поля (idFirm, Shpart, data), конструкторы для создания объектов, методы- геттеры и сеттеры для обращения к private-полям, переопределяет методы toString(), equals() и hashCode() суперкласса Object, реализует метод CompareTo() интерфейса Comporable (для задания естественного порядка сортировки объектов).

Класс ParRegister() описывает сущность «Поставочная ведомость», т.е группу (карту) фирм. В статической области класса определена и инициализирована ссылка на список возможного количества деталей.

Класс содержит следующие переменные экземпляра (нестатические поля):

name – название ведомости (группы фирм);

register – интерфейсная ссылка на объект типа Мap – карту отображений ключей фирм, включенных в данную ведомость, на соответствующие значения полученных деталей.

Класс имеет несколько конструкторов, в том числе конструктор, позволяющий задать компаратор для карты register. Карты отображений во всех конструкторах создаются на основе класса TreeMap.

Другие методы класса:

setRegisterName – сеттер для поля названия ведомости;

getRegisterName(), getRegister() – геттеры для полей;

toString() – возвращает строку описания объекта;

***запрос на вставку данных:***

addPar() – добавляет деталь (ключ и значение количества) в ведомость;

***запросы на удаление данных:***

delPar (ParKey ek) – исключает запись о детале, заданную ключом ek из ведомости;

delPar (int idFirm) – исключает все записи о деталях заданной детали;

***запрос на обновление данных:***

updatePar () – меняет на заданное количество деталей с заданным ключом;

***запросы на выборку данных:***

size() – вычисляет размер ведомости (количество записей о фирмах в ней);

selectSumCountFirm() – вычисляет суммарное количество деталей;

**selectPartData()–** возвращает подведомость (выборку деталей по заданной фирме);

**selectPartDataData () –** возвращает подведомость (выборку деталей для заданной фирмы в заданной дате);

**selectShpartData () –** возвращает подведомость по заданной детале;

***запросы на сортировку данных:***

**sortIdAscDataDesc () –** возвращает ведомость, являющуюся результатом сортировки основной ведомости по возрастанию номера ID фирмы;

**sortShpartAscDataDesc() –** возвращает ведомость, являющуюся результатом сортировки основной ведомости по возрастанию шифра детали (лексикографически);

**sortCountDesc () –** возвращает ведомость, являющуюся результатом сортировки основной ведомости по количеству;

***запрос на вывод данных:***

putPartRegisterr() – выводит основную ведомость (текущий объект) в окно терминала.

Названия методов, возвращающих ссылки на ведомости (объекты определяемого класса PartRegister) выделены полужирным шрифтом. Можно сказать, что эти методы возвращают ссылки на некоторые подмножества основной ведомости (текущего объекта), которые, в свою очередь, являются ведомостями и, следовательно, могут выполнять все вышеперечисленные методы, определенные в описываемом классе. Мы наблюдаем рекурсию при определении данных.

Кроме описанных выше классов, проект содержит три класса, реализующих интерфейс Comparator: CompIdAscDataDesc, CompShpartAscDataDesc, CompCountDesc. Принцип их действия описан в предыдущей лабораторной работе. Наиболее интересен класс CompCountDesc, с помощью которого может быть реализована сортировка не по ключам (как это принято в TreeMap), а по значениям.

Следует обратить внимание на следующие моменты:

1) Для получения значений в методе compare, мы все равно используем ключи, а также ссылку на карту, являющуюся предметом сортировки, передаваемую в конструктор компаратора (конструктор с параметром);

2) Значения (оценки) в карте не уникальны, в отличие от ключей, а результат, возвращенный методом compare используется системой для принятия решения о добавлении (или запрете добавления) объекта в набор TreeMap. Если при сравнении добавляемого объекта с объектами, добавленными в карту ранее, метод compare на одинаковых значениях оценки будет возвращать 0 (признак равенства объектов), то добавляемый объект не будет добавлен в сортируемый набор. Поэтому возможность возвращения нуля методом compare компаратора CompCountDesc закрыта. Проанализируйте код метода, измените его так, чтобы он при равных значениях оценок сравниваемых объектов возвращал 0, запустите проект и посмотрите, как изменился вывод программы.

Класс FirmPartDemo включает метод main, в котором осуществляется демонстрирование основных возможностей, предусмотренных в проекте для работы с поставочными ведомостями.

**5. Текст программы с комментариями**

import java.io.\*;

**public class FirmPartDemo{**

**public static void main (String[ ] args){**

PartRegister reg=new PartRegister ("Ведомость");

// в массив args при запуске метода main (String[ ] args)

// ввели 10 фирм

int j = 0; // индекс массива args

// добавление фирм

for (int i = 0; i < 10; i++) {

int id = Integer.valueOf(args[j]);

j++; // получен int-эквивалент из String

String Shpart = args[j];

j++;

String data = args[j];

j++;

// получаем int-эквивалент из String

int count = Integer.valueOf(args[j]);

j++;

// добавить фирм

reg.addPar(new ParteKey(id, Shpart, data),count);

}

System.out.println(

String.format("Начальное состояние ведомости %s:",reg));

reg.putPartRegister();

System.out.println("\n----------------1.Запрос на добавление записи о детале.----------------");

//попытка добавить деталь с ключом, который уже присутствует в ведомости

ParteKey ek = new ParteKey(1432,"Процессор","12.03.16");

if (!reg.addPar(ek,63))

System.out.println(

String.format("Запись %s\n\t нарушает правило уникальности ключа",ek));

System.out.println("\n----------------2.Запрос на обновление записи о детале.----------------");

//Изменить количество сущ детали/ прерасчет

ek = new ParteKey(1432,"Процессор","12.03.16");

reg.updatePar(ek,342);

//Выводим ведомость после обновления записи

System.out.println(

String.format("После обновления количества в ведомости:\n\t %s",ek));

reg.putPartRegister();

//Запросы на выборку данных

System.out.println("\n----------------3.Запросы на выборку записей о деталях.----------------");

// Выводим выборку из карты деталей для заданной фирмы,

// упорядоченную по дате и количеству

reg.selectPartData(653).putPartRegister();

reg.selectPartData(756).putPartRegister();

// Выводим суммарное колличество деталей для заданной фирмы

System.out.println(

String.format("Сумма количества всех деталей: %5d",reg.selectSumCountDis(1)));

// Выводим выборку из карты деталей для заданной фирмы в заданной дате,

// упорядоченную по колличеству

reg.selectPartDataData(228,"31.12.16").sortCountDesc ().putPartRegister();

reg.selectPartDataData(228,"09.05.16").sortCountDesc ().putPartRegister();

reg.selectShpartData("Процессор").sortIdAscDataDesc().putPartRegister();

// Запросы на удаление записей о деталях

System.out.println("\n----------------4.Запрос на удаление записи о детале.----------------");

//удаляем из ведомости

//запись о хранении деталях фирмы 313 (03.03.13)

ek=new ParteKey(313,"Болты","03.03.13");

reg.delPar(ek);

//Выводим ведомость после удаления записи

System.out.println(

String.format("После удаления записи:\n\t %s",ek));

reg.putPartRegister();

//пытаемся удалить запись, которой нет в ведомости

ek = new ParteKey(222,"Кнопки","31.12.16");

if (!reg.delPar(ek))

System.out.println(

String.format("Попытка удаления несуществующей записи:\n\t %s",ek));

//Удаляем из ведомости все детали фирмы 222

reg.delPar(222);

//Выводим ведомость после удаления записей

System.out.println("После удаления всех записей о деталях фирмы 222");

reg.putPartRegister();

System.out.println("\n Рассмотренная структура программы обработки базы данных (БД):");

System.out.println("код и вызовы запросов встроены в код программы,");

System.out.println("может применяться, если список запросов к БД");

System.out.println("ограничен и заранее известен.");

System.out.println("В противном случае, нужно иcпользовать возможности Java");

System.out.println("по применению встроенного динамического SQL.");

}

}

import java.util.\*;

**public class PartRegister {**

private final static String ParREG\_FORMAT\_STRING =

"\"Журнал учета поставок: %s, количество деталей: %1d\"";

private String name;

private Map <ParteKey, Integer> register;

public PartRegister(){

name="";

register = new TreeMap <ParteKey, Integer>();

}

public PartRegister(String name){

this.name=name;

register = new TreeMap <ParteKey, Integer>();

}

public PartRegister(String name, Comparator comp){

this.name=name;

register = new TreeMap <ParteKey, Integer> (comp);

}

public void setRegisterName(String name)

{this.name=name;}

public String getRegisterName(){return name;}

public Map <ParteKey, Integer> getRegister(){

return register;

}

public String toString(){

return String.format(ParREG\_FORMAT\_STRING,name,size());

}

public boolean addPar(ParteKey ek, int Count){

if (register.containsKey(ek)) return false;

register.put(ek,Count);

return true;

}

public boolean delPar(ParteKey ek){

if (register.containsKey(ek)){register.remove(ek); return true;}

else return false;

}

public boolean delPar(int id){

Set <Map.Entry <ParteKey ,Integer>> setE = register.entrySet();

Iterator <Map.Entry <ParteKey, Integer>> it = setE.iterator();

int i=0;

while (it.hasNext()){

Map.Entry <ParteKey,Integer> keyVal = it.next();

if (keyVal.getKey().getId()==id){

it.remove();

i=i+1;

}

}

if (i==0) return false;

return true;

}

public boolean updatePar(ParteKey ek, int Count){

if (!register.containsKey(ek)) return false;

register.put(ek, Count);

return true;

}

public int size(){

return register.size();

}

public PartRegister selectPartData(int idPar){

PartRegister disExReg = new PartRegister ( String.format("%s: выборка по детале %7d",

name,idPar));

Set <Map.Entry <ParteKey, Integer>> setE = register.entrySet();

for (Map.Entry <ParteKey, Integer> keyVal:setE)

if (keyVal.getKey().getId()==idPar)

disExReg.register.put(keyVal.getKey(),keyVal.getValue());

return disExReg;

}

public PartRegister selectShpartData(String Shpart){

PartRegister disExReg = new PartRegister ( String.format("%s: выборка по шифру %s",

name,Shpart

));

Set <Map.Entry <ParteKey, Integer>> setE = register.entrySet();

for (Map.Entry <ParteKey, Integer> keyVal:setE)

if (keyVal.getKey().getShpart().equals(Shpart))

disExReg.register.put(keyVal.getKey(),keyVal.getValue());

return disExReg;

}

public PartRegister selectPartDataData(int id, String Data){

PartRegister disExReg = new PartRegister (

String.format("%s: выборка по детале %1d и дате %1s",

name,id, Data));

Set <Map.Entry <ParteKey, Integer>> setE = register.entrySet();

for (Map.Entry <ParteKey, Integer> keyVal:setE)

if ((keyVal.getKey().getId()==id)&&(keyVal.getKey().getData()==Data))

disExReg.register.put(keyVal.getKey(),keyVal.getValue());

return disExReg;

}

public int selectSumCountDis(int id){

int sum = 0;

PartRegister disReg = selectPartData(id);

Collection <Integer> v = register.values();

for (int Count: v) sum+=Count;

return sum;

}

public PartRegister sortIdAscDataDesc(){

PartRegister disExReg = new PartRegister(String.format("%s:\n\t сортировка по возрастанию номера id",name),new CompIdAscDataPart());

Set <Map.Entry <ParteKey, Integer>> setE = register.entrySet();

for (Map.Entry <ParteKey, Integer> keyVal:setE)

disExReg.addPar(keyVal.getKey(), keyVal.getValue());

return disExReg;

}

public PartRegister sortShpartAscDataDesc(){

PartRegister disExReg = new PartRegister(String.format(

"%s:\n\t сортировка по возрастанию шифра и убыванию количества",

name),new CompShpartAscDataPart());

Set <Map.Entry <ParteKey, Integer>> setE = register.entrySet();

for (Map.Entry <ParteKey, Integer> keyVal:setE)

disExReg.addPar(keyVal.getKey(), keyVal.getValue());

return disExReg;

}

public PartRegister sortCountDesc (){

PartRegister disExReg = new PartRegister(String.format(

"%s:\n\t сортировка по убыванию количества",

name),new CompCountPart(this.register));

Set <Map.Entry <ParteKey, Integer>> setE = register.entrySet();

for (Map.Entry <ParteKey, Integer> keyVal:setE)

disExReg.addPar(keyVal.getKey(), keyVal.getValue());

return disExReg;

}

public void putPartRegister(){

System.out.println(name);

System.out.printf("%5s%7s%20s%20s%25s\n",

"Number","ID","Шифр детали","Дата","Количество");

int i=1;

Set <Map.Entry <ParteKey, Integer>> setE = register.entrySet();

for (Map.Entry <ParteKey, Integer> keyVal:setE){

ParteKey key=keyVal.getKey();

System.out.printf(" %-7d %-11d %-19s %7s %20s\n",

i, key.getId(), key.getShpart(), key.getData(), keyVal.getValue());

i=i+1;

}

}

}

**public class ParteKey implements Comparable <ParteKey>{**

private final static String PartKey\_FORMAT\_STRING =

"Фирма: %1d | Шифр детали: %1s | Дата: %1s |";

private int id;

private String Shpart;

private String Data;

public ParteKey(){

this.id=0; this.Shpart="";

this.Data="";

}

public ParteKey(int id, String Shpart, String Data){

this.id=id; this.Shpart=Shpart;

this.Data=Data;

}

public int getId(){return id;}

public String getShpart(){return Shpart;}

public String getData(){return Data;}

public void setId(int id){this.id = id;}

public void setShpart(String Shpart){this.Shpart = Shpart;}

public void setData(String Data){this.Data = Data;}

public String toString(){

return String.format(PartKey\_FORMAT\_STRING,id,Shpart,Data);

}

public boolean equals (Object ob){

if (ob==this) return true;

if (ob==null) return false;

if (getClass()!=ob.getClass())return false;

ParteKey ek=(ParteKey)ob;

return (id == ek.id)&&(Shpart.equals(ek.Shpart))

&&(Data==ek.Data);

}

public int hashCode(){

return 7\* (new Integer(id)).hashCode()+

11 \* Shpart.hashCode()+

13 \* (new Integer(Data)).hashCode();

}

public int compareTo(ParteKey ek){

if (id < ek.id) return -1;

if (id > ek.id) return 1;

if (Shpart.compareTo(ek.Shpart)<0) return -1;

if (Shpart.compareTo(ek.Shpart)>0) return 1;

if (Data.compareTo(ek.Data)<0) return -1;

if (Data.compareTo(ek.Data)>0) return 1;

return 0;

}

}

import java.util.\*;

**public class CompIdAscDataPart implements Comparator <ParteKey>{**

public int compare (ParteKey ex1, ParteKey ex2){

int id1 = ex1.getId();

int id2 = ex2.getId();

String dat1 = ex1.getData();

String dat2 = ex2.getData();

if (id1<id2) return -1;

if (id1>id2) return 1;

//номера зачеток совпадают (один и тот же студент)

// if (dat1>dat2) return -1;

// if (dat1<dat2) return 1;

return 0;

}

}

import java.util.\*;

**public class CompCountPart implements Comparator <ParteKey>{**

Map <ParteKey, Integer> registr;

public CompCountPart (Map <ParteKey, Integer> registr){

this.registr=registr;

}

public int compare (ParteKey ex1,ParteKey ex2){

int Count1=registr.get(ex1);

int Count2=registr.get(ex2);

if (Count1>Count2) return -1;

return 1;

}

}

import java.util.\*;

**public class CompShpartAscDataPart implements Comparator <ParteKey>{**

public int compare (ParteKey ex1, ParteKey ex2){

String subj1 = ex1.getShpart();

String subj2 = ex2.getShpart();

String dat1 = ex1.getData();

String dat2 = ex2.getData();

if (subj1.compareTo(subj2)<0)return -1;

if (subj1.compareTo(subj2)>0)return 1;

//if (dat1>dat2) return -1;

//if (dat1<dat2) return 1;

return 0;

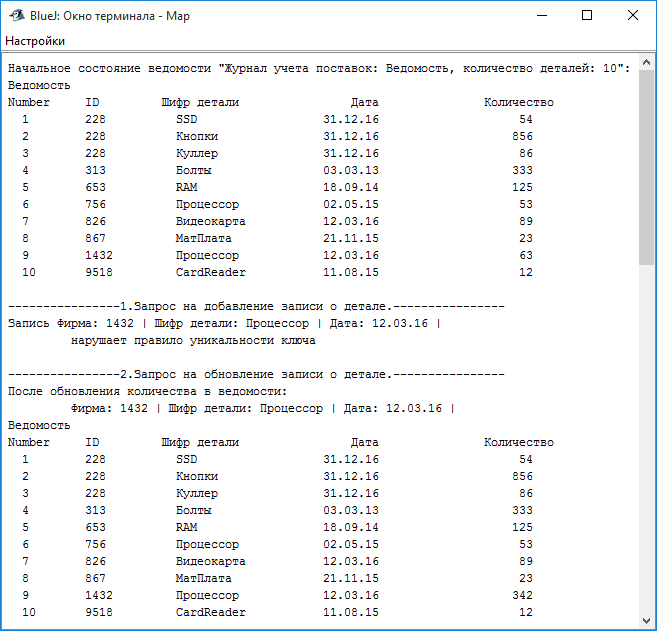
}

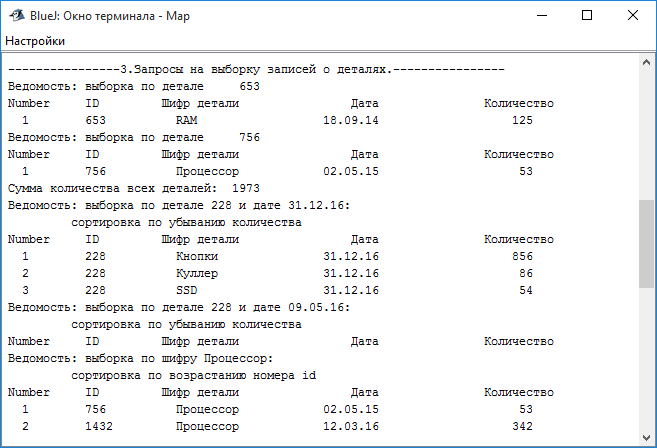
}

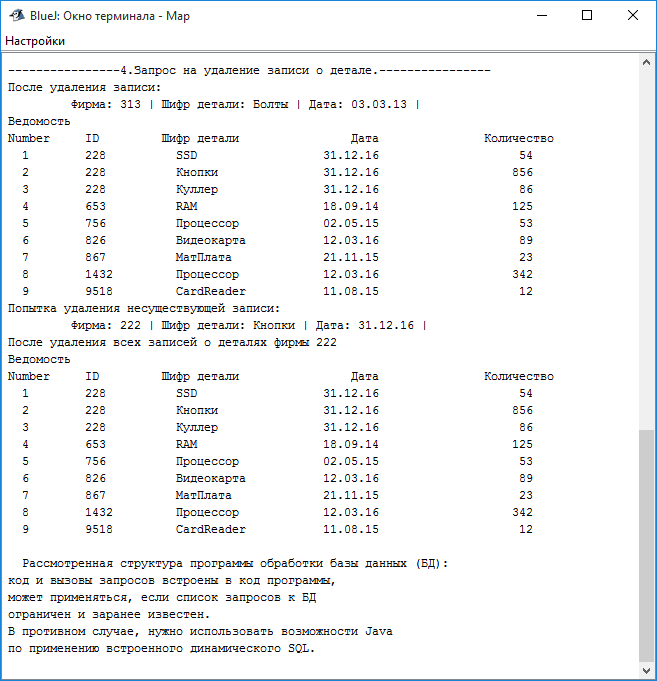
**6. Протокол отладки и тестирования программы (с подробным описанием всех ошибок процесса выполнения, в том числе, вызванных некорректной работой со списками, множествами и картами отображений)**

Ошибки возникают при некорректном заполнении ведомости.

**7. Результаты работы программы**







**8. Сравнительный анализ возможностей инструментария коллекций и карт отображений (по итогам разработки трех программ)**

List - это динамический массив

Set - это массив исключающий дубликаты

Map - это контейнер с парами ключ -> значение (ключи уникальны). Имеет 2 ключевых значения.

**9. Вывод**

В ходе выполнения данной работы были получены навыки использования набора классов и интерфейсов JAVA COLLECTION FRAMEWORK для создания карт отображений объектов-ключей в объекты-значения.

**Заключение**

Изучил средства для работы с группами объектов (коллекциями).

Познакомился с фреймами. Выучил обработку событий, ведение журнала событий. Изучил менеджеры компоновки. Выучил все основные элементы интерфейса: выпадающий список, текстовое поле, прокрутка, выбор кнопок и их группировка. Познакомился со спиннерами и диалогами. Научился работать с панелью цветов. Получил первичные навыки работы с апплетами.

**Библиографический список**

1. Гласс Р. Руководство по надежному программированию. М.: Финансы и статистика, 1982.
2. Зельковиц М., Шоу А. Принципы разработки ПО. М.: Мир, 1982.
3. Керниган Б., Гллоджер Ф. Элементы стиля программирования. М.: Радио и связь, 1984.
4. Липаев В.В. Качество программного обеспечения. М.: Финансы и статистика, 1983.
5. Майерс Г. Надежность программного обеспечения. М.: Финансы и статистика, 1980.