Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**ЛАБОРАТОРНАЯ РАБОТА №2**

**Дисциплина: Платформо-независимое программирование**

Работу выполнил: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ К. А. Корнилов

Направление подготовки: 02.03.02 Фундаментальная информатика и информационные технологии

Преподаватель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_В. И. Шиян

**Тема:** Наследование и реализация полиморфизма Java.

**Цель**:

Наследование и реализация полиморфизма в Java. Изучить структуру проекта. Освоить процесс построения иерархии классов на основе разработанных UML-диаграмм. Изучить синтаксис и возможности переопределенных функций. Разработать программу согласно варианту.

**Задание:**

1. Скомпилировать и выполнить примеры листинга 6;
2. Проверить пример 3.3. без переопределенного метода equals()
3. Разработать заданную иерархию классов приведенным вариантам, обязательным элементом является разработка UML–диаграмм классов. Классы должны быть расположены по пакетам, все классы должны иметь необходимые методы get и set.
4. При разработке конструкторов использовать обращение к суперклассу.
5. В классе main продемонстрировать создание массива разнотипных объектов, установку и вывод характеристик каждого объекта. В массив вносить только уникальные объекты.
6. Для вывода содержимого объектов выполнить перегрузку метода toString, начиная с класса родителя.

**Ход работы:**

1. Ide создадим класс User с перегруженным методом toString.

Код User.java:

package javanew;  
  
  
class User {  
 public User(String name,String surname,int birthYeari){  
 this.name=name;  
 this.surname=surname;  
 this.birthYear=birthYeari;  
 }  
 private String name;  
 private String surname;  
 private int birthYear;  
  
 public String getName(){  
 return this.name;  
 }  
 public String getSurname(){  
 return this.surname;  
 }  
 public int getBirthYear(){  
 return this.birthYear;  
 }  
 public void setName(String n){  
 this.name=n;  
 }  
 public void setSurname(String s){  
 this.surname=s;  
 }  
 public void setBirthYear(int y){  
 this.birthYear=y;  
 }  
  
 @Override  
 public String toString(){  
 return this.name+" "+this.surname+" "+getBirthYear() +"г.";  
 }  
}

Результат вывода работы программы с данным классом представлен на рисунке 1.

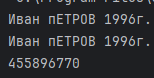


Рисунок 1 – Результат работы листинга 6.

1. Проверил пример 3.3. без переопределенного метода equals()

Код программы:

package javanew;  
  
public class MyDate {  
 private int day,month,year;  
 public MyDate(int d,int m,int y){  
 day=d;  
 month=m;  
 year=y;  
 }  
}

MyDate date1=new MyDate(7,11,2014);  
MyDate date2=new MyDate(7,11,2014);  
if( date1 == date2) System.*out*.println("date 1 is identical date 2");  
else System.*out*.println("date 1 is not identical date 2");  
  
if( date1.equals(date2)) System.*out*.println("date 1 is equal date 2");  
else System.*out*.println("date 1 is not equal date 2");  
  
System.*out*.println("set date2=date1");  
date2=date1;  
  
if( date1 == date2) System.*out*.println("date 1 is identical date 2");  
else System.*out*.println("date 1 is not identical date 2");

Результат выполнения программы представлен на рисунке 2:

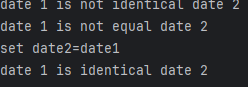


Рисунок 2 – Команда компиляции классов.

Согласно данному результату можно сделать вывод,что без перезаписи метода equals() будет происходить сравнения ссылок на объекты, а не самого содержания объектов.

1. На основании заданий была составлена и написана иерархия классов.

Диаграмма классов полученной иерархии представлена на рисунке 3. Код создания классов представлен в приложении.

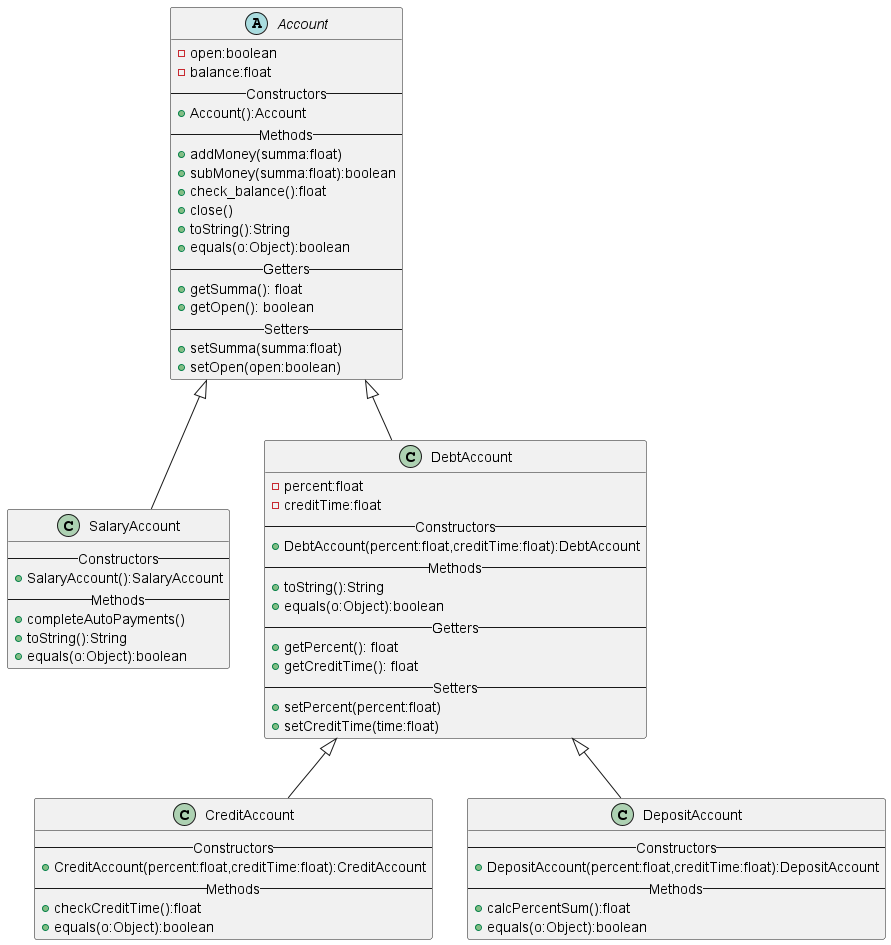


Рисунок 3 – Диаграмма классов иерархии.

1. Был создан массив разнотипных объектов для отображения свойства полиморфизма.

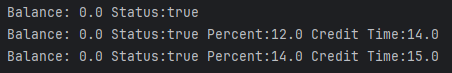


Рисунок 4 – Результат демонстрации полиморфизма.

Код:

Account[] accounts = new Account[3];  
accounts[0]=new SalaryAccount();  
accounts[1]=new CreditAccount(12,14);  
accounts[2]=new DepositAccount(14,15);  
for(Account p:accounts){  
 System.*out*.println(p);  
}

1. Был создан массив, в который программно добавлялись только уникальные элементы.

Код:

public class JavaNew {  
 public static void main(String[] args) {  
 Account[] accounts = new Account[10];  
 accounts[0]=new SalaryAccount();  
 accounts[1]=new CreditAccount(12,14);  
 accounts[2]=new DepositAccount(13,15);  
 accounts[3]=new DepositAccount(13,15);  
 accounts[4]=new DepositAccount(14,15);  
 accounts[5]=new CreditAccount(11,14);  
 accounts[6]=new CreditAccount(0,14);  
 accounts[7]=new CreditAccount(12,14);  
 accounts[8]=new SalaryAccount();  
 accounts[9]=new SalaryAccount();  
 ArrayList<Account> acc = new ArrayList<>();  
 for(Account p:accounts){  
 System.*out*.println("``````");  
 System.*out*.println(p);  
 boolean res=true;  
 for(Account m:acc){  
 if(p.equals(m)){res=false;}  
 System.*out*.println(p+" "+m+" "+res);  
 }  
 if(res) {  
 acc.add(p);  
 }  
 }  
 System.*out*.println("`````````");  
 for(Account p:acc){  
 System.*out*.println(p);  
 }  
 }  
}

Результат выполнения представлен на рисунке 5.

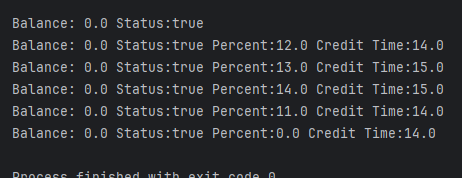


Рисунок 5 – Массив уникальных объектов.

**Вывод:** Был освоен процесс построения иерархии классов на основе разработанных UML-диаграмм. Был изучен синтаксис и возможности переопределения функций.

**ПРИЛОЖЕНИЕ**

**Полный код классов:**

package javanew;  
  
abstract public class Account {  
 private float balance;  
 private boolean open;  
 public Account(){  
 this.balance = 0;  
 this.open = true;  
 }  
 public float getSumma(){  
 return this.balance;  
 }  
 public void setSumma(float summa){  
 this.balance=summa;  
 }  
 public void addMoney(float summa){  
 this.balance=this.balance+summa;  
 }  
 public boolean subMoney(float summa){  
 if(this.balance-summa>0){  
 this.balance=this.balance-summa;  
 return true;  
 }  
 return false;  
 }  
 public float check\_balance(){  
 if(this.open)return this.getSumma();  
 return 0;  
 }  
 public boolean getOpen(){  
 return this.open;  
 }  
 public void setOpen(boolean open){  
 this.open=open;  
 }  
 public void close(){  
 this.open=!this.open;  
 }  
  
 @Override  
 public String toString(){  
 return "Balance: "+this.getSumma()+" Status:"+this.getOpen();  
 }  
 @Override  
 public boolean equals(Object o){  
 boolean result = false;  
 if(o!=null && o instanceof Account){  
 Account d = (Account) o;  
 if((this.balance==d.getSumma())&&(this.open==d.getOpen())){  
 result=true;  
 }  
 }  
 return result;  
 }  
}

package javanew;  
  
public class CreditAccount extends DebtAccount {  
 public CreditAccount(float percent,float creditTime){  
 super(percent,creditTime);  
 }  
 public float checkCreditTime(){  
 return this.getCreditTime();  
 }  
  
 @Override  
 public boolean equals(Object o){  
 return (o instanceof CreditAccount) && super.equals(o);  
 }  
}

package javanew;  
  
public class DebtAccount extends Account{  
  
 private float percent;  
 private float creditTime;  
  
 public DebtAccount(float percent,float creditTime){  
 super();  
 this.percent=percent;  
 this.creditTime=creditTime;  
 }  
  
 public void setPercent(float percent){  
 this.percent=percent;  
 }  
 public float getPercent(){  
 return this.percent;  
 }  
  
 public void setCreditTime(float time){  
 this.creditTime=time;  
 }  
 public float getCreditTime(){  
 return this.creditTime;  
 }  
 @Override  
 public String toString(){  
 return super.toString()+" Percent:"+this.percent+" Credit Time:"+this.creditTime;  
 }  
 @Override  
 public boolean equals(Object o){  
 boolean result = false;  
 if(o!=null && o instanceof DebtAccount){  
 DebtAccount d = (DebtAccount) o;  
 if((this.getSumma()==d.getSumma())&&(this.getOpen()==d.getOpen())&&(this.getPercent()==d.getPercent())&&(this.getCreditTime()==d.getCreditTime())){  
 result=true;  
 }  
 }  
 return result;  
 }  
  
}

package javanew;  
  
public class DepositAccount extends DebtAccount{  
 public DepositAccount(float percent,float creditTime){  
 super(percent,creditTime);  
 }  
 public float calcPercentSum(){  
 return this.getPercent();  
 }  
  
 @Override  
 public boolean equals(Object o){  
 return (o instanceof DepositAccount) && super.equals(o);  
 }  
}