Федеральное государственное автономное образовательное учреждение высшего профессионального образования

«Университет ИТМО»

**ОТЧЕТ**

по лабораторной работе

# «Читаем и вспоминаем из C#»

Выполнил:

Студент гр.K33204

Мороз Илья Олегович

Проверил:

Тьютор

Дубаков Анатолий Алексеевич

Санкт-Петербург

2020 г.

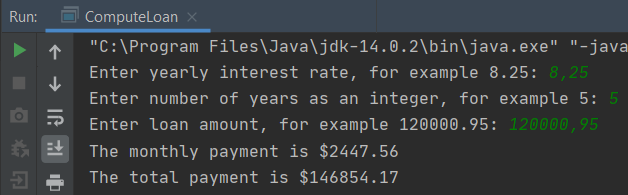
**Переход от процедурного программирования к ООП**

**Подсчёт loan с процедурным и объектно-ориентированным подходами.**

Для начала рассмотрим процедурный подход:

import java.util.Scanner;  
  
class ComputeLoan {  
 public static void main(String[] args) {  
 // Create a Scanner  
 Scanner input = new Scanner(System.*in*);  
 // Enter yearly interest rate  
 System.*out*.print("Enter yearly interest rate, for example 8.25: ");  
 double annualInterestRate = input.nextDouble();  
 // Obtain monthly interest rate  
 double monthlyInterestRate = annualInterestRate / 1200;  
 // Enter number of years  
 System.*out*.print("Enter number of years as an integer, for example 5: ");  
 int numberOfYears = input.nextInt();  
 // Enter loan amount  
 System.*out*.print("Enter loan amount, for example 120000.95: ");  
 double loanAmount = input.nextDouble();  
 // Calculate payment  
 double monthlyPayment = loanAmount \* monthlyInterestRate / (1  
 - 1 / Math.*pow*(1 + monthlyInterestRate, numberOfYears \* 12));  
 double totalPayment = monthlyPayment \* numberOfYears \* 12;  
 // Display results  
 System.*out*.println("The monthly payment is $" + (int) (monthlyPayment \* 100) / 100.0);  
 System.*out*.println("The total payment is $" + (int) (totalPayment \* 100) / 100.0);  
 }  
}

Как мы можем заметить, весь код находится в методе main, который является исполняемым по умолчанию. Следовательно, мы никак не можем использовать этот подсчёт, например, в классе Person, чтобы считать его loan.



Теперь рассмотрим решение с подходом ООП.

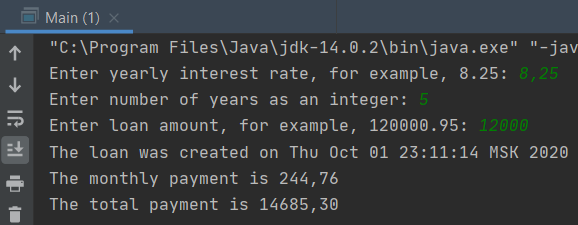
Для такого подхода нам понадобится реализовать методы для расчёта loan в отдельном классе. Класс Loan в таком случае может выглядеть вот так:

public class Loan {  
 private double annualInterestRate;  
 private int numberOfYears;  
 private double loanAmount;  
 private java.util.Date loanDate;  
  
 */\*\*  
 \* Default constructor  
 \*/* public Loan() {  
 this(2.5, 1, 1000);  
 }  
  
 */\*\*  
 \* Construct a loan with specified annual interest rate,  
 \* number of years and loan amount  
 \*/* public Loan(double annualInterestRate, int numberOfYears,  
 double loanAmount) {  
 this.annualInterestRate = annualInterestRate;  
 this.numberOfYears = numberOfYears;  
 this.loanAmount = loanAmount;  
 loanDate = new java.util.Date();  
 }  
  
 */\*\*  
 \* Return annualInterestRate  
 \*/* public double getAnnualInterestRate() {  
 return annualInterestRate;  
 }  
  
 */\*\*  
 \* Set a new annualInterestRate  
 \*/* public void setAnnualInterestRate(double annualInterestRate) {  
 this.annualInterestRate = annualInterestRate;  
 }  
  
 */\*\*  
 \* Return numberOfYears  
 \*/* public int getNumberOfYears() {  
 return numberOfYears;  
 }  
  
 */\*\*  
 \* Set a new numberOfYears  
 \*/* public void setNumberOfYears(int numberOfYears) {  
 this.numberOfYears = numberOfYears;  
 }  
  
 */\*\*  
 \* Return loanAmount  
 \*/* public double getLoanAmount() {  
 return loanAmount;  
 }  
  
 */\*\*  
 \* Set a newloanAmount  
 \*/* public void setLoanAmount(double loanAmount) {  
 this.loanAmount = loanAmount;  
 }  
  
 */\*\*  
 \* Find monthly payment  
 \*/* public double getMonthlyPayment() {  
 double monthlyInterestRate = annualInterestRate / 1200;  
 double monthlyPayment = loanAmount \* monthlyInterestRate / (1 -  
 (Math.*pow*(1 / (1 + monthlyInterestRate), numberOfYears \* 12)));  
 return monthlyPayment;  
 }  
  
 */\*\*  
 \* Find total payment  
 \*/* public double getTotalPayment() {  
 double totalPayment = getMonthlyPayment() \* numberOfYears \* 12;  
 return totalPayment;  
 }  
  
 */\*\*  
 \* Return loan date  
 \*/* public java.util.Date getLoanDate() {  
 return loanDate;  
 }  
}

Теперь импортируем и используем его в методе main:

import java.util.Scanner;  
  
public class Main {  
 */\*\*  
 \* Main method  
 \*/* public static void main(String[] args) {  
 // Create a Scanner  
 Scanner input = new Scanner(System.*in*);  
  
 // Enter yearly interest rate  
 System.*out*.print(  
 "Enter yearly interest rate, for example, 8.25: ");  
 double annualInterestRate = input.nextDouble();  
  
 // Enter number of years  
 System.*out*.print("Enter number of years as an integer: ");  
 int numberOfYears = input.nextInt();  
  
 // Enter loan amount  
 System.*out*.print("Enter loan amount, for example, 120000.95: ");  
 double loanAmount = input.nextDouble();  
  
 // Create Loan object  
 Loan loan =  
 new Loan(annualInterestRate, numberOfYears, loanAmount);  
  
 // Display loan date, monthly payment, and total payment  
 System.*out*.printf("The loan was created on %s\n" +  
 "The monthly payment is %.2f\nThe total payment is %.2f\n",  
 loan.getLoanDate().toString(), loan.getMonthlyPayment(),  
 loan.getTotalPayment());  
 }  
}

Как мы можем заметить, теперь мы используем объект класса Loan для расчётов. Это позволяет переиспользовать наш класс в других классах, а также сократить повторяющийся код.

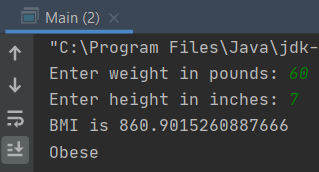


**Подсчёт индекса массы тела**

Рассмотрим пример с подсчётом индекса массы тела. Для начала снова рассмотрим пример процедурного программирования.

import java.util.Scanner;  
  
public class Main {  
 public static void main(String[] args) {  
 Scanner input = new Scanner(System.*in*);  
  
 // Prompt the user to enter weight in pounds  
 System.*out*.print("Enter weight in pounds: ");  
 double weight = input.nextDouble();  
  
 // Prompt the user to enter height in inches  
 System.*out*.print("Enter height in inches: ");  
 double height = input.nextDouble();  
  
 final double KILOGRAMS\_PER\_POUND = 0.45359237; // Constant  
 final double METERS\_PER\_INCH = 0.0254; // Constant   
  
 // Compute BMI  
 double weightInKilograms = weight \* KILOGRAMS\_PER\_POUND;  
 double heightInMeters = height \* METERS\_PER\_INCH;  
 double bmi = weightInKilograms /  
 (heightInMeters \* heightInMeters);  
  
 // Display result  
 System.*out*.println("BMI is " + bmi);  
 if (bmi < 18.5)  
 System.*out*.println("Underweight");  
 else if (bmi < 25)  
 System.*out*.println("Normal");  
 else if (bmi < 30)  
 System.*out*.println("Overweight");  
 else  
 System.*out*.println("Obese");  
 }  
}

Как мы можем заметить, это схоже с предыдущим примером. Весь расчёт находится внутри метода main, что не позволяет переиспользовать код.



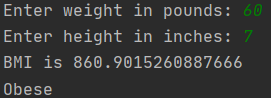
Теперь посмотрим на вариант с использованием ООП. Класс BMI будет выглядеть следующим образом:

public class BMI {  
 private String name;  
 private int age;  
 private double weight; // in pounds  
 private double height; // in inches  
 public static final double *KILOGRAMS\_PER\_POUND* = 0.45359237;  
 public static final double *METERS\_PER\_INCH* = 0.0254;  
  
 public BMI(String name, int age, double weight, double height) {  
 this.name = name;  
 this.age = age;  
 this.weight = weight;  
 this.height = height;  
 }  
  
 public BMI(String name, double weight, double height) {  
 this(name, 20, weight, height);  
 }  
  
 public double getBMI() {  
 double bmi = weight \* *KILOGRAMS\_PER\_POUND* /  
 ((height \* *METERS\_PER\_INCH*) \* (height \* *METERS\_PER\_INCH*));  
 return Math.*round*(bmi \* 100) / 100.0;  
 }  
  
 public String getStatus() {  
 double bmi = getBMI();  
 if (bmi < 18.5)  
 return "Underweight";  
 else if (bmi < 25)  
 return "Normal";  
 else if (bmi < 30)  
 return "Overweight";  
 else  
 return "Obese";  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public int getAge() {  
 return age;  
 }  
  
 public double getWeight() {  
 return weight;  
 }  
  
 public double getHeight() {  
 return height;  
 }  
}

Теперь реализуем метод main для работы с этим классом

import java.util.Scanner;  
  
public class Main {  
 public static void main(String[] args) {  
 Scanner input = new Scanner(System.*in*);  
  
 // Prompt the user to enter weight in pounds  
 System.*out*.print("Enter weight in pounds: ");  
 double weight = input.nextDouble();  
  
 // Prompt the user to enter height in inches  
 System.*out*.print("Enter height in inches: ");  
 double height = input.nextDouble();  
  
 BMI bmi = new BMI("Ilya", 19, weight, height);  
  
 double result = bmi.getBMI();  
  
 // Display result  
 System.*out*.println("BMI is " + result);  
 if (result < 18.5)  
 System.*out*.println("Underweight");  
 else if (result < 25)  
 System.*out*.println("Normal");  
 else if (result < 30)  
 System.*out*.println("Overweight");  
 else  
 System.*out*.println("Obese");  
 }  
}

Как мы можем заметить, теперь мы можем производить расчёт с помощью объекта класса BMI.



**Ассоциация на примере студентов и курса**

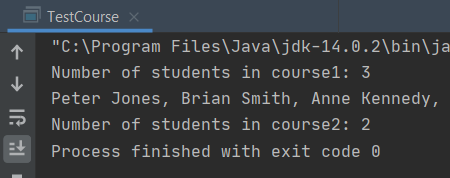
Рассмотрим ассоциацию на примере студентов и курса. У нас будет класс Course со следующим функционалом:

public class Course {  
 private String courseName;  
 private String[] students = new String[100];  
 private int numberOfStudents;  
  
 public Course(String courseName) {  
 this.courseName = courseName;  
 }  
  
 public void addStudent(String student) {  
 students[numberOfStudents] = student;  
 numberOfStudents++;  
 }  
  
 public String[] getStudents() {  
 return students;  
 }  
  
 public int getNumberOfStudents() {  
 return numberOfStudents;  
 }  
  
 public String getCourseName() {  
 return courseName;  
 }  
  
 public void dropStudent(String student) {  
 for (int i = 0; i < this.numberOfStudents; i++) {  
 if (this.students[i].equals(student)) {  
 String[] copyArray = new String[this.numberOfStudents - 1];  
 System.*arraycopy*(this.students, 0, copyArray, 0, i);  
 System.*arraycopy*(this.students, i + 1, copyArray, i, this.numberOfStudents - i - 1);  
 break;  
 }  
 }  
 }  
}

Теперь посмотрим на main с использованием этого класса

public class TestCourse {  
 public static void main(String[] args) {  
 Course course1 = new Course("Data Structures");  
 Course course2 = new Course("Database Systems");  
  
 course1.addStudent("Peter Jones");  
 course1.addStudent("Brian Smith");  
 course1.addStudent("Anne Kennedy");  
  
 course2.addStudent("Peter Jones");  
 course2.addStudent("Steve Smith");  
  
 System.*out*.println("Number of students in course1: "  
 + course1.getNumberOfStudents());  
 String[] students = course1.getStudents();  
 for (int i = 0; i < course1.getNumberOfStudents(); i++)  
 System.*out*.print(students[i] + ", ");  
  
 System.*out*.println();  
 System.*out*.print("Number of students in course2: "  
 + course2.getNumberOfStudents());  
 }  
}

И на результат работы программы



**Реализация стека с использованием ООП**

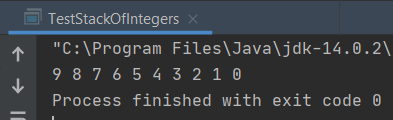
Реализуем класс Стек. ООП позволит использовать данный класс в дальнейшем.

public class StackOfIntegers {  
 private int[] elements;  
 private int size;  
 public static final int *DEFAULT\_CAPACITY* = 16;  
  
 */\*\*  
 \* Construct a stack with the default capacity 16  
 \*/* public StackOfIntegers() {  
 this(*DEFAULT\_CAPACITY*);  
 }  
  
 */\*\*  
 \* Construct a stack with the specified maximum capacity  
 \*/* public StackOfIntegers(int capacity) {  
 elements = new int[capacity];  
 }  
  
 */\*\*  
 \* Push a new integer into the top of the stack  
 \*/* public void push(int value) {  
 if (size >= elements.length) {  
 int[] temp = new int[elements.length \* 2];  
 System.*arraycopy*(elements, 0, temp, 0, elements.length);  
 elements = temp;  
 }  
 elements[size++] = value;  
 }  
  
 */\*\*  
 \* Return and remove the top element from the stack  
 \*/* public int pop() {  
 return elements[--size];  
 }  
  
 */\*\*  
 \* Return the top element from the stack  
 \*/* public int peek() {  
 return elements[size - 1];  
 }  
  
 */\*\*  
 \* Test whether the stack is empty  
 \*/* public boolean empty() {  
 return size == 0;  
 }  
  
 */\*\*  
 \* Return the number of elements in the stack  
 \*/* public int getSize() {  
 return size;  
 }  
}

Проверим работу этого класса в методе main.

public class TestStackOfIntegers {  
 public static void main(String[] args) {  
 StackOfIntegers stack = new StackOfIntegers();  
  
 for (int i = 0; i < 10; i++)  
 stack.push(i);  
  
 while (!stack.empty())  
 System.*out*.print(stack.pop() + " ");  
 }  
}

Вот такой результат мы получили в консоли. Следовательно, всё работает правильно.



**Ассоциация**

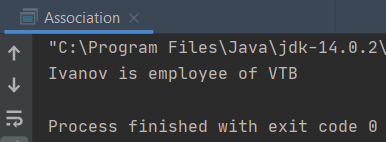
Рассмотрим ассоциацию на примере программы с классами Bank и Employee. Данные классы будут связаны только благодаря объектам этих классов. В одном банке может работать много работников.

class Bank {  
 private String name;  
  
 // имя банка  
 Bank(String name) {  
 this.name = name;  
 }  
  
 public String getBankName() {  
 return this.name;  
 }  
}

class Employee {  
 private String name;  
  
 // имя employee  
 Employee(String name) {  
 this.name = name;  
 }  
  
 public String getEmployeeName() {  
 return this.name;  
 }  
}

class Association {  
 public static void main(String[] args) {  
 Bank bank = new Bank("VTB");  
 Employee emp = new Employee("Ivanov");  
  
 System.*out*.println(emp.getEmployeeName() +  
 " is employee of " + bank.getBankName());  
 }  
}

Запустим программу и посмотрим на вывод



**Агрегация**

Рассмотрим агрегацию на примере программы со студентами и институтом.

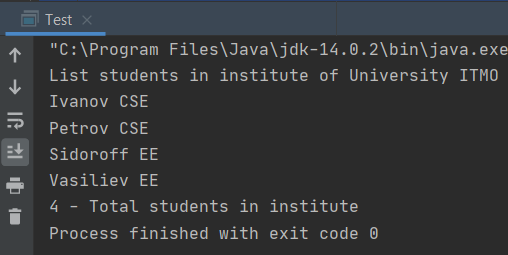
import java.util.\*;  
  
// student class  
class Student {  
 String name;  
 int id;  
 String dept;  
  
 Student(String name, int id, String dept) {  
  
 this.name = name;  
 this.id = id;  
 this.dept = dept;  
  
 }  
}

import java.util.ArrayList;  
  
class Department {  
 String name;  
 private ArrayList<Student> students;  
  
 Department(String name, ArrayList<Student> students) {  
  
 this.name = name;  
 this.students = students;  
  
 }  
  
 public ArrayList<Student> getStudents() {  
 return students;  
 }  
}

import java.util.ArrayList;  
  
class Institute {  
  
 String instituteName;  
 private ArrayList<Department> departments;  
  
 Institute(String instituteName, ArrayList<Department> departments) {  
 this.instituteName = instituteName;  
 this.departments = (ArrayList<Department>) departments;  
 }  
  
 // общее количество студентов всех департаментов  
 // в данном институте  
 public int getTotalStudentsInInstitute() {  
 int noOfStudents = 0;  
 ArrayList<Student> students;  
 for (Department dept : departments) {  
 students = dept.getStudents();  
 for (Student s : students) {  
 System.*out*.println(s.name + " " + s.dept);  
 noOfStudents++;  
 }  
 }  
 return noOfStudents;  
 }  
  
}

import java.util.ArrayList;  
  
class Test {  
 public static void main(String[] args) {  
 Student s1 = new Student("Ivanov", 1, "CSE");  
 Student s2 = new Student("Petrov", 2, "CSE");  
 Student s3 = new Student("Sidoroff", 1, "EE");  
 Student s4 = new Student("Vasiliev", 2, "EE");  
  
 // Создание списка ArrayList студентов CSE .  
 ArrayList<Student> cse\_students = new ArrayList<Student>();  
 cse\_students.add(s1);  
 cse\_students.add(s2);  
  
 // Создание списка ArrayList студентов EE  
 ArrayList<Student> ee\_students = new ArrayList<Student>();  
 ee\_students.add(s3);  
 ee\_students.add(s4);  
  
 Department CSE = new Department("CSE", cse\_students);  
 Department EE = new Department("EE", ee\_students);  
  
 ArrayList<Department> departments = new ArrayList<Department>();  
 departments.add(CSE);  
 departments.add(EE);  
  
 // creating an instance of Institute.  
 Institute institute = new Institute("University ITMO", departments);  
 System.*out*.println("List students in institute of " + institute.instituteName);  
 System.*out*.print(institute.getTotalStudentsInInstitute());  
 System.*out*.print(" - Total students in institute");  
  
 }  
}

Посмотрим на вывод в консоли и удостоверимся, что программа работает.



**Агрегация**

Рассмотрим агрегацию на примере программы, которая описывает книги и библиотеку.

// Класс Book  
class Book {  
  
 public String title;  
 public String author;  
  
 Book(String title, String author) {  
  
 this.title = title;  
 this.author = author;  
 }  
}

import java.util.ArrayList;  
  
class Library {  
  
 // ссылка на список книг в библиотеке.  
 private final ArrayList<Book> books;  
  
 Library(ArrayList<Book> books) // конструктор  
 {  
 this.books = books;  
 }  
  
 public ArrayList<Book> getTotalBooksInLibrary() {  
  
 return books;  
 }  
  
}

import java.util.ArrayList;  
import java.util.List;  
  
class TestComposition {  
 public static void main(String[] args) {  
  
 // Создание объектов класса Book  
 Book b1 = new Book("EffectiveJ Java", "Joshua Bloch");  
 Book b2 = new Book("Thinking in Java", "Bruce Eckel");  
 Book b3 = new Book("Java: The Complete Reference", "Herbert Schildt");  
  
 // Создание списка, который содержит  
 // книги.  
 ArrayList<Book> books = new ArrayList<>();  
 books.add(b1);  
 books.add(b2);  
 books.add(b3);  
  
 Library library = new Library(books);  
  
 List<Book> bks = library.getTotalBooksInLibrary();  
 for (Book bk : bks) {  
  
 System.*out*.println("Title : " + bk.title + " and "  
 + " Author : " + bk.author);  
 }  
 }  
}

Проверим работоспособность программы в консоли.

