2. ОБЪЕКТЫ

Цель – научиться проектировать и реализовывать приложения, ориентированные на использование объектов.

Задачи*:*

– научиться создавать объекты с использованием конструкторов различной сигнатуры;

– научиться реализовывать этап объектного проектирования;

– научиться решать задачи автоматизации процессов предметной области с помощью приложений, функционирующих в рамках ООП-парадигмы.

Формируемые компетенции: ПК-11; ПК-12; ПК-30.

Теоретическая часть

Создание объектов. Конструкторы. В предыдущей работе создание экземпляра типа (объекта) на базе определения типа (класса) уже было продемонстрировано (например, рис. 1.3). Создание объекта происходит с использованием ключевого слова new. На рис. 2.1 представлено определение класса, на рис. 2.2 создание объекта и вызов его метода.

```
1 package lesson2_Example;
2
3 public class Example {
4     public void RepeatAfterMe(String str){
5         System.out.println(str);
6     }
7 }
```

Рис. 2.1. Определение класса Example

```
1 package lesson2_Example;
2
3 public class Starter_Examole {
4
5     public static void main(String[] args) {
6         Example ex = new Example();
7         ex.RepeatAfterMe("Say Hello!");
8     }
9 }
```

Рис. 2.2. Реализация типа Example

Пример простой, но необходимо обратить внимание на строку 6 рис. 2.2, в которой производится создание объекта. Переменная ex содержит только адрес области памяти, в которой размещен экземпляр типа, построенный с использованием шаблона Example. Объект был создан с использованием конструкции:

Имя_Класса имя_переменной
= new *Имя_Конструктора*([параметры]);

После зарезервированного слова new следует имя конструктора класса – специального метода, который не вызывается явно, а используется только при создании объекта совместно с new. На рис. 2.1 в определении класса метод с такой сигнатурой отсутствует – в языках Java и C# используется конструктор без параметров, присутствующий в классе по умолчанию. Хотя бы один конструктор должен присутствовать в каждом классе (явный или по умолчанию). Конструкторов в классе может быть несколько (определяется соображениями целесообразности). Несмотря на специальное назначение методов, для них действуют такие же правила перегрузки, как и для остальных методов класса. На рис. 2.3 представлено определение класса с несколькими конструкторами. Конструктор без параметров закомментирован. Конструктор с единственным параметром введен для инициализации закрытой переменной fio. Конструктор с тремя параметрами осуществляет инициализацию трех закрытых полей: поле fio будет проинициализировано с помощью вызова конструктора этого же класса с другой сигнатурой; остальным полям значения будут присвоены явно.

```
1  package lesson2_Example;
2
3  public class Студент {
4      private String fio;
5      private String группа;
6      private float балл_по_математике;
7      private float балл_по_физике;
8      private float балл_по_программированию;
9      // Конструктор без параметров (не используется)
10     //Студент(){
11     //}
12     // Конструктор с параметром (ФИО студента)
13     Студент(String pFio){
14         fio = pFio;
15     }
16     // Конструктор с параметрами (ФИО, оценки по физиек и математике)
17     Студент(String pFio, float bMath, float bPhys)
18     {
19         this(pFio);
20         балл_по_математике = bMath;
21         this.балл_по_физике = bPhys;
22
23     }
24     // Кнструктор с параметрами (ФИО и все оценки)
25     Студент(String pFio, float bMath,
26             float bPhys, float bProg){
27         this(pFio, bMath, bPhys);
28         балл_по_программированию = bProg;
29     }
30 }
```

Рис. 2.3. Определение класса «Студент»

Следует обратить внимание на использование зарезервированного слова this. В одном случае (строка 19 на рис. 2.3) this используется для вызова конструктора; в другом (строка 21) – для получения ссылки на текущий объект и, через ссылку, обращение к полю данного объекта. Второй вариант использования this применяют программисты если необходимо подчеркнуть принадлежность поля этому классу. На рис. 2.4 показано создание нескольких экземпляров класса «Студент». Обратите внимание, что конструктор по умолчанию становится недоступным после определения хотя бы одного явного конструктора. Если он необходим, то его нужно реализовать явно.

```
1  package lesson2_Example;
2
3  public class Starter_Examole {
4
5      public static void main(String[] args) {
6          Студент студент1 = new Студент();
7          Студент студент2 = new Студент("Никитин Н.К.");
8          Студент студент3 = new Студент("Иванова Е.Е.",
9                  89.11f, 97.00f);
10         Студент студент4 = new Студент("Климов А.С.",
11                 91.30f, 11.70f, 56.50f);
12     }
13 }
```

Рис. 2.4. Использование класса «Студент»

Для сравнения на рис. 2.5 представлено определение класса на языке C#. Создание объектов синтаксически не отличается.

Сравните определения классов на рис. 2.3 и рис. 2.5. Подумайте, почему в Java перед конструкторами не указан модификатор доступа public.

При программировании классов на языках C# или Java следует понимать отличие объектов Java (C#) от ссылок C++: в C++ такая переменная не может иметь значение null; объектные переменные Java – это указатели на объекты. Все объекты в языке Java располагаются в динамической памяти. Если объект содержит другую объектную переменную, она представляет собой всего лишь указатель на другой объект, расположенный в области памяти под названием «куча» (heap).

Декомпозиция. Проектирование классов. Современные корпоративные информационные системы предоставляют пользователю большое количество сервисов и манипулируют большим количеством разнообразной информации. В разработке таких сложных систем участвует большое количество программистов-разработчиков. Информационные системы состоят из большого количества подсистем, модулей, которые взаимодействуют между собой через ограниченное количество интерфейсов (методов). Использование в таких системах пронципов объектно-ориентированного программирования позволяет сделать приемлемыми сроки проектирования, разработки, модификации. Один из таких принципов – декомпозиция.

```
class Студент
{
    private String fio;
    private String группа;
    private float балл_по_математике;
    private float балл_по_физике;
    private float балл_по_программированию;

    // Конструктор с параметром (ФИО студента)
    1 reference
    public Студент(String pFio)
    {
        fio = pFio;
    }
    // Конструктор с параметрами (ФИО, оценки по физике и математике)
    1 reference
    public Студент(String pFio, float bMath, float bPhys)
        :this(pFio)
    {
        балл_по_математике = bMath;
        this.балл_по_физике = bPhys;

    }
    // Конструктор с параметрами (ФИО и все оценки)
    0 references
    public Студент(String pFio, float bMath,
            float bPhys, float bProg)
        :this(pFio, bMath, bPhys)
    {
        балл_по_программированию = bProg;
    }
}
```

Рис. 2.5. Определение класса «Студент» на языке C#

Декомпозиция – это определение физических и логических сущностей, а также принципов их взаимодействия на основе анализа предметной области создаваемой системы.

Все системы в ООП-парадигме состоят из классов. При проектировании приложения следует придерживаться следующих принципов; их использование позволяет сделать код более профессиональным и качественным и обеспечить простое сопровождение, поддержку и модификацию.

1. Имя класса должно отображать его роль, назначение.

2. Разрабатывать класс нужно таким образом, чтобы внесение в него изменений происходило относительно просто.

3. Назначение класса должно быть простым.

4. Код конструктора должен только инициализировать состояние объекта, в нем не должно быть длительных операций. Необходимо избегать вызовов других методов в классе (за исключение других конструкторов и методов final).

5. Необходимо использовать инкапсуляцию нестатических и неконстантных полей.

6. Необходимо избегать слишком длинных методов.

7. Необходимо избегать длинных списков аргументов (возможно следует объединять группы аргументов в новый тип).

8. Если метод используется только другими методами данного класса, то следует объявлять его как private.

Методика и порядок выполнения работы

Для закрепления теоретического материала и выполнения задач лабораторной работы рассмотрим пример решения задачи.

Задача. Разработать приложение для автоматизации рабочего места сотрудника библиотеки.

Задача является учебной и необходимо выделить ряд объектов предметной области; определить действия, которые они реализуют; установить интерфейсы их взаимодействия.

Решение. Представим предметную область совокупностью следующих классов: «Книга» (рис. 2.6), «Читатель» (рис. 2.8), «Библиотека» (рис. 2.9). Затем реализовать перечень необходимых операций (методов) для изменения состояния объектов.

```
1 package lesson2;
2
3 public class Книга {
4     String название;
5     String автор;
6     String код;
7     Жанр жанр;
8     Состояние состояние;
9 }
```

Рис. 2.6. Определение класса «Книга»

Дополнительно определим два перечня: «Состояние» и «Жанр» (рис. 2.7). Наравне с классами в формировании структуры программы принимают участие и другие типы: перечисления, структуры, интерфейсы, события.

```
1  package lesson2;
2
3  public enum Состояние {
4      НаРуках,
5      Доступна
6  }
```

а)

```
1  package lesson2;
2
3  public enum Жанр {
4      Детектив,
5      Триллкр,
6      Образование,
7      Информатика,
8      Физика,
9      Психология
10 }
```

б)

Рис. 2.7. Вспомогательные типы-перечисления: а – описывает состояние объекта; б – описывает жанровую принадлежность книги

Обоснованное использование перечней также является хорошим признаком качества программного кода. В данном случае программист достигает консолидации всей информации о жанрах книг и всех возможных состояниях каждой книги всего в двух структурах данных.

```
1  package lesson2;
2
3  public class Читатель {
4      String фио;
5      String билет;
6      public Читатель(String фиоНовый,
7              String билетНовый) {
8          фио = фиоНовый;
9          билет = билетНовый;
10     }
11 }
```

Рис. 2.8. Определение класса «Читатель»

В классе «Библиотека» инкапсулированы методы, выполняющие основные функции автоматизации поиска и учета книг.

```
1  package lesson2;
2
3  import java.util.Hashtable;
9
10 public class Библиотека {
11
12     Hashtable<Книга, Vector<Читатель>> книги =
13             new Hashtable<Книга, Vector<Читатель>>();
14
15     private Библиотека() {  }
16
17     public static Библиотека Инициализация(){
18         Библиотека res = new Библиотека();
19         Книга новый = new Книга();
20         новый.автор = "Коротков А.Е.";
21         новый.жанр = Жанр.Образование;
22         новый.код = "001";
23         новый.название = "Программирование C#";
24         новый.состояние = Состояние.Доступна;
25         res.книги.put(
26                 новый,
27                 new Vector<>());
28         новый = new Книга();
29         новый.автор = "Тодеев А.А.";
30         новый.жанр = Жанр.Образование;
31         новый.код = "002";
32         новый.название = "Java. Учебник";
33         новый.состояние = Состояние.Доступна;
34         res.книги.put(
35                 новый,
36                 new Vector<>());
37
38         return res;
39     }
```

```
41    public void ВыдачаКниги(String кодКниги, Читатель читатель) {
42        // Ищем книгу с переданным кодом
43        Optional<Entry<Книга, Vector<Читатель>>>  find =
44                книги.entrySet().stream()
45                .filter(e->e.getKey().код.equals(кодКниги)).findFirst();
46
47        // Если книга нашлась
48        if(find.isPresent()){
49            // проверяем доступность
50            if(find.get().getKey().состояние == Состояние.Доступна){
51                find.get().getKey().состояние = Состояние.НаРуках;
52                find.get().getValue().add(читатель);
53                System.out.println("Выдача книги зафиксирована.");
54            }
55            else
56                System.out.println("Книга на руках!!!");
57        }
58          else
59              System.out.println("Книги с таким кодом нет!!!");
60    }
61
62
63    public void ВозвратКниги(Читатель читатель) {
64        System.out.println("Функция не реализована!!!");
65    }
66
67    public void Каталог(){
68        книги.entrySet().stream().forEach(e->{
69            Книга tek = e.getKey();
70            System.out.println(tek.автор + "\n" +
71            tek.название + "\n" + tek.код + "\n");
72        });
73    }
74 }
```

Рис. 2.9. Определение класса «Библиотека»

Класс «Библиотека» содержит несколько методов, назначение которых не нуждается в пояснении. Но исходные коды методов Каталог() и ВыдачаКниги(…) содержат интересный пример использования лямбда-выражений, которые в данном случае заменяют громоздкие конструкции циклов. Для сравнения в приложении 1 представлен исходный код класса «Библиотека», реализованного на языке C# и описывающего аналогичные методы.

```
1  package lesson2;
2
3  import java.util.Scanner;;
4
5  public class Starter {
6
7      public static void main(String[] args) {
8          Scanner sc = new Scanner(System.in);
9          Библиотека мояБиблиотека = Библиотека.Инициализация();
10         System.out.println("Библиотека готова к работе!");
11         System.out.println("Идентификация читателя:");
12         System.out.print("ФИО > ");
13         String фиоЧ = sc.nextLine();
14         System.out.print("Читательский билет > ");
15         String билетЧ = sc.next();
16         Читатель клиент = new Читатель(фиоЧ,билетЧ);
17         int операция = 0;
18         do{
19             System.out.print("Уважаемый " + клиент.фио +
20                     ", Выберите действие:\n1 - Вывод каталога\n" +
21                     "2 - Запрос книги\n3 - Сдача книги\n" +
22                     "0 - выход\n Ваш выбор>");
23             операция = sc.nextInt();
24             switch(операция){
25             case 1: мояБиблиотека.Каталог(); break;
26             case 2:
27                 // какую книгу брать?
28                 System.out.print("Код книги ?> ");
29                 String код = sc.next();
30                 мояБиблиотека.ВыдачаКниги(код, клиент);
31                 break;
32             case 3:
33                 мояБиблиотека.ВозвратКниги(клиент);
34                 break;
35             default:break;
36             }
37         } while (операция != 0);
38
39         sc.close();
40         System.out.println("Библиотека. Работа завершена!");
41     }
42 }
```

Рис. 2.10. Основная программа

На рис. 2.10 представлен код метода main(). Следует учитывать, что при практическом построении корпоративных приложений в рамках ООП-парадигмы такой большой объем кода в функции main является недопустимым. В данной функции должны только создаваться объекты (небольшое количество) вызываться их методы, запускающие задачи на выполнение. Постарайтесь исправить данный недостаток при реализации индивидуального задания.

На рис. 2.11 показан фрагмент консольного вывода приложения.

```
Библиотека готова к работе!
Идентификация читателя:
ФИО > Новиков И.Е.
Читательский билет > 999-7
Уважаемый Новиков И.Е., Выберите действие:
1 - Вывод каталога
2 - Запрос книги
3 - Сдача книги
0 - выход
 Ваш выбор>1
Коротков А.Е.
Программирование C#
001

Тодеев А.А.
Java. Учебник
002

Уважаемый Новиков И.Е., Выберите действие:
1 - Вывод каталога
2 - Запрос книги
3 - Сдача книги
0 - выход
 Ваш выбор>0
Библиотека. Работа завершена!
```

Рис. 2.11. Консольный вывод программы

Внимательно изучите код полученной программы, затем выполните индивидуальное задание.

Индивидуальное задание

В каждом приложении необходимо выделить классы, необходимые для описания предметной области; разработать структуру каждого класса (необходимые методы и поля). После этого реализуйте приложение с использованием разработанных типов данных. Всего необходимо реализовать не менее 3–4 классов в рамках автоматизации задачи.

Вариант	Описание предметной области
1	Учет поступления продуктов питания на склад
2	Учет взаиморасчетов с клиентами
3	Система поддержания товарно-материальных запасов

4	Система управления взаимоотношениями с клиентами
5	Учет сетевого и компьютерного оборудования
6	Электронный журнал учебного заведения
7	Учет поступления налоговых платежей от контрагентов
8	Система анализа сетевых запросов
9	Система управления роботом-манипулятором
10	Система планирования работ в коллективе исполнителя
11	Учет товаров в разрезе товарных групп
12	Система учета платежей за услуги провайдера
13	Система анализа банковских счетов клиентов
14	Пропускная система на предприятии
15	Учет текущей аттестации студентов
16	Система учета фактов правонарушений на дорогах
17	Билетная система кинотеатра
18	Система заказов интернет-магазина
19	Электронная очередь частной клиники
20	Учет автомобильных комплектующих
21	Система анализа кредитоспособности клиентов банка
22	Учет поступления комплектующих в сборочный цех
23	Анализ эффективности менеджеров
24	Учет финансовых операций предприятия
25	Система поддержания запасов стройматериалов

Содержание отчета и его форма

Отчет по лабораторной работе должен содержать следующие данные.

1. Номер и название лабораторной работы; задачи.

3. Ответы на контрольные вопросы.

4. Экранные формы (консольный вывод) и листинг программного кода с комментариями, показывающие порядок выполнения лабораторной работы, и результаты, полученные в ходе её выполнения.

Отчет о выполнении лабораторной работы, подписанный студентом, сдается преподавателю.

Контрольные вопросы

1. В каких случаях при создании объекта не производится вызова конструктора?

2. Сравните синтаксис определения и вызова конструктора класса на языках C# и Java? Существуют ли синтаксические различия?

3. Что такое конструктор по умолчанию? Когда конструктор по умолчанию становится недоступным?

4. Какие типы данных кроме классов существуют в Java?

5. Сколько конструкторов может иметь класс? Что такое «сигнатура метода»? Чем отличаются конструкторы одного и того же класса?

6. Поясните принципы выделения отдельных сущностей предметной области в классы. Опишите рекомендации, которыми нужно при этом руководствоваться.

Литература: 1, 7.