Тема 4
Работа с объектами

Объявление класса

- В Java не допускается наличие кода вне классов, т. е. нет нелокальных переменных, которые бы не являлись полями, и нет функций в смысле C++.
- Любая программа начинается с объявления класса. Для этого используется следующий синтаксис:

```
[ специф_доступа ] class имя_класса
{
   [ специф_доступа ] тип_данных имя_поля [ = начальн_знач ];
     . . .
   [ специф_доступа ] тип_данных имя_метода(список_аргументов)
     {
          тело_метода
     }
     . . .
}
```

- Порядок объявления полей и методов в классе может быть произвольным. Объявление класса и его реализация размещаются в одном и том же файле.

Конструктор

- Конструктор – это блок инструкций, вызываемый при создании объекта данного класса.

- Конструктор в Java имеет имя, совпадающее с именем класса, и не имеет возвращаемого значения.

- Если конструктор класса не объявлен явно, то создаётся пустой конструктор, не имеющий аргументов (конструктор по умолчанию).

Перегрузка операторов

- Если в классе определено два метода с одним и тем же именем, то такие методы называются *перегруженными*.
- Выбор необходимого перегруженного метода осуществляется компилятором на основании количества и типов аргументов, передаваемых методу.
- Не допускается определение двух методов с одинаковыми сигнатурами, т.е. совокупностью имени метода и типов его аргументов, так как в этом случае компилятор не сможет определить, какой из методов следует вызывать.

Объявление и создание объекта

Конструкция вида

```
MyClass myObject;
```

объявляет объектную ссылку типа MyClass с именем myObject и не создаёт сам объект.

Собственно создание объекта осуществляется с помощью операции new:

```
myObject = new MyClass();
```

Здесь создаётся объект типа MyClass (при создании вызывается конструктор без аргументов), и ссылка на него записывается в переменную myObject.

Возможно одновременное объявление объектной ссылки и её инициализация:

```
MyClass myObject = new MyClass();
```

Обращение к полям и методам

- Когда объект создан, доступ к его полям и методам осуществляется посредством операции . (точка).
- При обращении к полю или методу объекта из метода, вызванного на этом же объекте, имя объекта может опускаться либо заменяться ключевым словом this.
- Ключевое слово this используется также для вызова конструктора класса из другого конструктора. В последнем случае подобный вызов должен быть первым оператором конструктора.

Статические поля и методы

- Для объявления статических полей и методов используется спецификатор static, например:

```
static int field = 10;
```

- Статические методы не могут (без явного указания объекта) обращаться к нестатическим полям и нестатическим методам класса.
- Вызов статических полей и методов может осуществляться без указания экземпляра класса. При таком обращении вместо имени экземпляра используется имя самого класса. Например:

```
double x = Math.sqrt(2.0);
```

Метод main(), который является точкой входа в Java-программу, всегда должен объявляться статическим.

- Управление доступом осуществляется путём установки определённого **спецификатора доступа** в описании классов, полей и методов.
- В Java существует два спецификатора доступа для классов и четыре — для полей и методов. Уровни доступа обозначаются своими спецификаторами, кроме уровня доступа «по умолчанию», которому никакого специального спецификатора не соответствует.
- **Главное правило управления доступом**: при проектировании для всех элементов программы следует задавать максимально ограниченный уровень доступа, позволяющий этим элементам правильно функционировать. Чем меньше классы знают друг о друге, тем больше шансов они имеют быть использованными повторно.

Уровни доступа вместе с областями видимости соответствующих элементов программы приведены в таблицах:

Область видимости		private	по умолч.	protected	public
тот же пакет	тот же класс	+	+	+	+
	другой класс	–	+	+	+
другой пакет	субкласс	–	–	+	+
	не субкласс	–	–	–	+

Таблица 1. Уровни доступа полей и методов

Область видимости	по умолч.	public
тот же пакет	+	+
другой пакет	–	+

Таблица 2. Уровни доступа классов и интерфейсов

Private

Для полей классов используется доступ private. Для чтения и изменения их значений извне используются специальные методы (setter'ы и getter'ы), имеющие специф[illegible]

```
class MyClass
{
    private int value;
    public void setValue(int value) { this.value = value; }
    public int getValue() { return value; }
}
```

Благодаря этому подходу всегда можно сделать некое поле полем только для чтения, удалив или просто не реализовав соответствующий setter, или ограничить возможные значения для поля, вставив соответствующую проверку.

Public

- Методы класса, имеющие спецификатор доступа public, образуют интерфейс к данным этого класса.
- Только через эти методы внешние классы смогут обращаться к данным, инкапсулированным классом, и только они определяют возможные способы обработки этих данных.
- При наличии наследования спецификаторы доступа элементов private и «по умолчанию» часто заменяются спецификатором protected, чтобы позволить наследникам класса иметь доступ к соответствующим элементам классов.

Наследование

- *Наследование* — это механизм построения новых классов на основе уже существующих. При этом наследники класса (субклассы, или подклассы) получают свойства и функциональность класса-родителя (суперкласса) и имеют возможность изменять и расширять их.
- Механизм наследования используется для создания более частных, специализированных классов по сравнению с суперклассом.
- Для того чтобы объявить класс наследником некоторого другого класса, используется ключевое слово **extends** с последующим именем наследуемого класса, добавляемое в объявление класса:

[специф_доступа] class имя_субкласса extends имя_суперкласса

- В Java допускается **наследование только от одного класса**.
- Субкласс **наследует все поля и методы** суперкласса, однако те из них, которые объявлены со спецификаторами доступа private и по умолчанию, оказываются недоступными для субкласса.
- Субкласс может **переопределять поля и методы** суперкласса. В этом случае доступ к родительским полям и методам закрывается и может осуществляться только из методов субкласса посредством ключевого слова super, синтаксис которого совпадает с синтаксисом явного обращения к элементам текущего экземпляра через this. Например:

```
public void Method() {
    super.Method();
// . . . }
```

- При переопределении полей и методов возможно **изменение спецификатора доступа** на более широкий (private на любой другой; по умолчанию на любой другой, кроме private и т. д.).

- Конструкторы не наследуются, однако конструкторы субклассов **обязательно вызывают конструкторы** своих суперклассов. Последнее может происходить явно (для этого первым оператором конструктора должен быть вызов конструктора суперкласса через super) или неявно (в этом случае происходит вызов конструктора по умолчанию). Вызов конструкторов родительских классов происходит сверху вниз по дереву наследования.

- Если в описании класса суперкласс для него не определён, то по умолчанию производится наследование от **библиотечного класса Object**. Этот класс представляет собой корень иерархии наследования в Java и содержит методы, необходимые для корректного функционирования JVM, а также методы, выполняющие такие распространённые операции, как преобразование к строке, сравнение объектов на равенство и т. д.

- Если метод объявлен со **спецификатором final**, то его переопределение запрещается. Если же со спецификатором final объявлен класс, то запрещается наследование от такого класса.

Полиморфизм

- **Полиморфизм** — это механизм, который позволяет обращаться к объектам, унаследованным от данного класса, как к объектам этого класса.
- При обращении к объектам субклассов они реагируют способом, определённым в субклассе, а не в родительском классе. Методы, определённые в суперклассе, образуют единый интерфейс доступа к данным всех субклассов.
- Полиморфизм является одним из самых мощных средств ООП, поскольку позволяет **единообразно обрабатывать наборы объектов**, принадлежащих различным классам, причём выбор реализации той или иной операции осуществляется автоматически на основании типа данных объекта.

Реализация полиморфизма в Java

(1) Ссылка на объект класса может быть приведена к ссылке на объект любого из классов, находящихся выше данного в иерархии наследования. Это преобразование является расширяющим и может производиться автоматически. Обратное преобразование также может быть произведено, но оно выполняется всегда явно.

(2) Если некоторые из методов суперкласса были переопределены в субклассе, то при обращении к этим методам даже посредством ссылки на объект суперкласса будет происходить вызов именно переопределённых методов.

Рассмотрим пример.

Пусть есть базовый класс «геометрическая фигура». Он содержит метод area(), предназначенный для вычисления площади фигуры. Каждый из субклассов класса «фигура» переопределяет этот метод таким образом, чтобы вычислялась площадь соответствующей фигуры:

```
/** Абстрактный класс "Геометрическая фигура" */
abstract class Figure
{
    /** Метод вычисления площади фигуры */
    abstract public double area();
}
/** Класс "Прямоугольник" */
class Rectangle extends Figure
{
    private double a, b;
    public Rectangle(double a, double b) { this.a = a; this.b = b; }
    public double area() { return a * b; }
}
/** Класс "круг" */
class Circle extends Figure
{
    private double r;
    public Circle(double r) { this.r = r; }
    public double area() { return Math.PI * r * r; }
}
/** Класс "треугольник" */
class Triangle extends Figure
{
    private double a, b, c;
    public Triangle(double a, double b, double c)
    {
        this.a = a; this.b = b; this.c = c;
    }
    public double area()
    {
        double p = (a + b + c) / 2;
        return Math.sqrt(p * (p - a) * (p - b) * (p - c));
    }
}
```

Ссылки на создаваемые экземпляры классов конкретных фигур сохраняются в массиве ссылок, имеющих тип «фигура», а затем единообразно обрабатываются: для каждой фигуры выводится её площадь. При этом никаких проверок типов фигур не требуется, для каждой фигуры автоматически вызывается нужный метод:

```
public class TestObject2 {
    public static void main(String[] args) {
        Figure f[] = new Figure[3]; // массив объектных ссылок
        f[0] = new Rectangle(2, 3);
        f[1] = new Triangle(3, 4, 5);
        f[2] = new Circle(1);
        for(int i = 0; i < f.length; ++i)
            System.out.println(f[i].area());
    }
}
```

Рекомендации по использованию наследования и полиморфизма

- При разработке иерархии классов следует строить дерево наследования в соответствии с **принципом**: субкласс является более частным, специализированным классом по отношению к суперклассу.
- Несоблюдение этого принципа приводит к серьёзным **ошибкам проектирования**, проявляющимся в сильной связности классов, неочевидности взаимоотношений между ними и затруднении проектирования на последующих этапах.
- Наследование поддерживает полиморфизм «генетически» общих методов.
- Полиморфизм же методов, имеющий другую природу, должен быть реализован посредством интерфейсов.

Задание

1) Создать класс «Служащий» с полями «Имя», «Дата рождения», «Зарплата» и методом «Повышение зарплаты», который увеличивает зарплату данного служащего на процент, передаваемый в метод. Кроме того, в классе должны быть предусмотрены методы, позволяющие получать значения полей («геттеры»).

2) Создать класс «Начальник», который будет дочерним классом класса «Служащий». Предусмотреть в нем дополнительное поле «Повышающий коэффициент» и переопределить в нем метод «Повышение зарплаты» с учетом этого коэффициента.

3) В основной программе создать массив сотрудников, среди которых будут как простые служащие, так и начальники. Поднять зарплату всем сотрудникам. Вывести на экран актуальную информацию о сотрудниках.