ПРАКТИЧЕСКАЯ РАБОТА №1

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ В JAVA

ЦЕЛЬ ПРАКТИЧЕСКОЙ РАБОТЫ:

Цель данной практической работы – освоить на практике работу с классами на Java.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ:

1. **Понятие класса**

В Java, класс является определением объектов одного и того же вида. Другими словами, класс это тип данных создаваемый программистом для решения задач. Он представляет из себя шаблон , или прототип, который определяет и описывает статические свойства и динамическое поведение, общие для всех объектов одного и того же вида.

Экземпляр класса - реализация конкретного элемента класса. Другими словами ,экземпляр экземпляра класса . Все экземпляры класса имеют аналогичные свойства , как описано в определении класса . Например, вы можете определить класс с именем "Студент " и создать три экземпляра класса "Студент ": " Петр", " Павел" и " Полина ". Термин " Объект " обычно относится к экземпляру класса . Но он часто используется свободно, которые могут относиться к классу или экземпляру.

Графически представляем Класс как трехкомпонентный контейнер-бокс с отсеками инкапсуляции данных и операций (методами)

Графически можно изобразить Класс как три отсека в боксе, в общем как на рисунке 1.

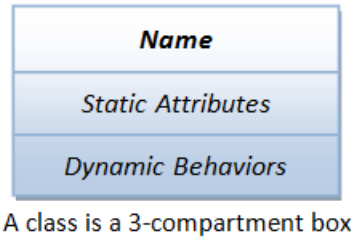

Рис.1 Представление класса

Имя (или сущность) : определяет класс.

Переменные (или атрибуты, состояние, поля данных класса): содержит статические атрибуты класса, или описывают свойства класса (сущности предметной области).

Методы (или поведение, функции, работа с данными): описывают динамическое поведение класса. Другими словами, класс инкапсулирует статические свойства (данные) и динамические модели поведения (операции, которые работают с данными) в одном месте (“контейнере” или “боксе”).

На рисунке 1.2 показано несколько примеров классов:

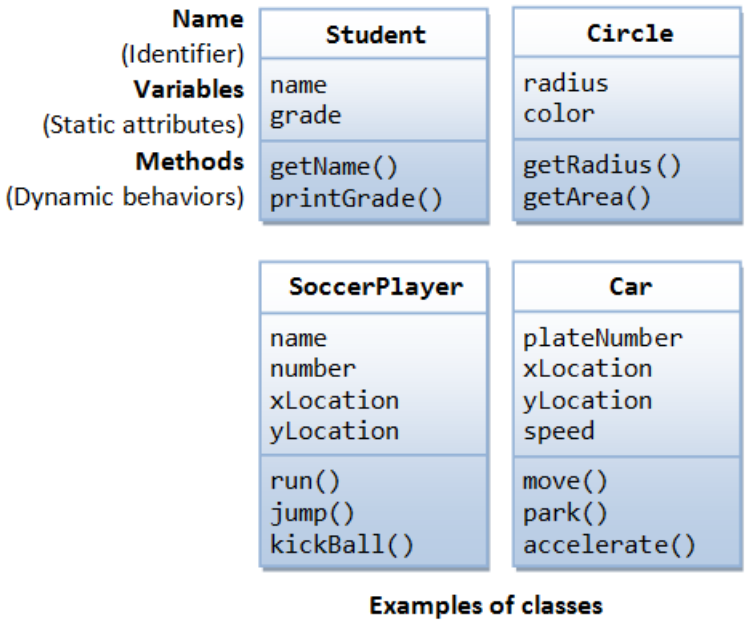

Рис. 1.2 Примеры классов

На рисунке 1.3 показаны два экземпляра класса типа Student "paul" и "peter" .

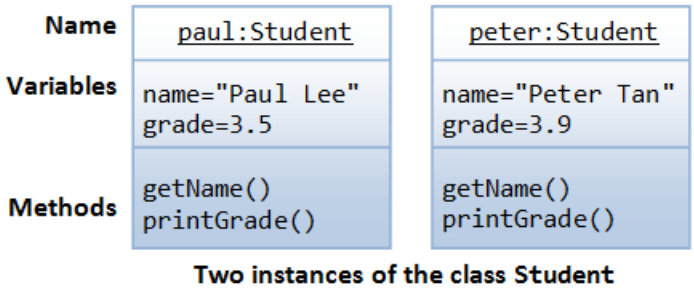

Рис. 1.3 Экземпляры класса Student

Unified Modeling Language (UML) класс и экземпляр диаграмм:

Приведенные выше диаграммы классов описаны в соответствии с UML нотацией. Класс представляется в этой нотации как прямоугольник, разделенный на три отсека, один содержит название , два вторых переменные (поля данных класса) и методы , соответственно. Имя класса выделено жирным шрифтом и находится посредине. Экземпляр (объект класса) также представляется в виде прямоугольника, разделенного на три отсека, в первом отсеке, надпись с именем экземпляра, показанной в instanceName:Classname и выделенная подчеркиванием (название_экземпляра : имя_класса).

Кратко подведем итоги:

1) Класс определенный программистом, абстрактный тип данных, повторно-используемый программный объект, который имитирует реальные сущности предметной области. Класс контейнер-бокс, который состоит из 3 – отсеков и содержит имя , переменные и методы .

2) Класс инкапсулирует структуры данных (в переменных) и алгоритмы (методы).

3) Значения переменных составляют его состояние. Методы создает свои модели поведения.

Экземпляр это представление (или реализация) конкретного представителя класса.

2. Определение класса

В Java , мы используем класс как ключевое слово, например чтобы определить класс.

Для примера:

```
 1 public class Circle {         // class name
 2    double radius;             // variables
 3    String color;
 4
 5    double getRadius() {...}   // methods
 6    double getArea() {...}
 7 }
 8 public class SoccerPlayer {   // class name
 9    int number;                // variables
10    String name;
11    int x, y;
12
13    void run() {...}           // methods
14    void kickBall() {...}
15 }
```

Синтаксис определения класса в Java:

```
1 [AccessControlModifier] class ClassName {
2    // class body contains definition of variables and methods
3    ...
4 }
```

Давайте разъясним, что такое контроль доступа или спецификатор доступа, например, public и private, позже.

Конвенция кода для класса (Class Naming Convention):

Имя класса должно быть всегда существительным или словосочетание из нескольких слов. Все слова должны с Прописной буквы (верблюжья нотация). Используйте существительное в единственном числе для имени класса. Выберите значимое и самодостаточное имя для класса. Для примера, SoccerPlayer, HttpProxyServer, FileInputStream, PrintStream and SocketFactory.

3. Создание экземпляров класса

Чтобы создать экземпляр класса, вы должны:

Объявить идентификатор экземпляра (имя экземпляра) конкретного класса.

Построить экземпляр (то есть, при выделении памяти для экземпляра и инициализации экземпляра) с помощью оператор "new".

Для примера , предположим, что у нас есть класс с именем Circle, мы можем создавать экземпляры Circle, следующим образом:

```
1 // Declare 3 instances of the class Circle, c1, c2, and c3
2 Circle c1, c2, c3;
3 // Allocate and construct the instances via new operator
4 c1 = new Circle();
5 c2 = new Circle(2.0);
6 c3 = new Circle(3.0, "red");
7 // You can declare and construct in the same statement
8 Circle c4 = new Circle();
```

4. Операция точка “.”

Переменные и методы, принадлежащие к классу формально называется переменные-поля данных класса и методы класса. Для ссылки на переменную-поле данных класса или метод, вы должны:

- Сначала создать экземпляр класса, который вам нужен;
- Затем, использовать оператор точка “.” чтобы сослаться на элемент класса (переменную-поле данных или метод класса)

Например, предположим, что у нас есть класс с именем Circle, с двумя переменными (радиус и цвет) и двумя методами (getRadius () и GetArea ()). Мы создали три экземпляра класса Circle, а именно, C1, C2 и C3 . Чтобы вызвать метод GetArea (), вы должны сначала определить к какой именно сущности вы обращаетесь, об этом собственно говорит говорит c2 , а затем использовать оператор точка , в виде c2.getArea (), для вызова метода GetArea () экземпляра c2.

Например,

```
1 // Declare and construct instances c1 and c2 of the class Circle
2 Circle c1 = new Circle ();
3 Circle c2 = new Circle ();
4 // Invoke member methods for the instance c1 via dot operator
5 System.out.println(c1.getArea());
6 System.out.println(c1.getRadius());
7 // Reference member variables for instance c2 via dot operator
8 c2.radius = 5.0;
9 c2.color = "blue"
```

Вызов метода GetArea () без указания экземпляра не имеет смысла , так как радиус неизвестно какого объекта (может быть много окружностей, у каждой из которых п свой собственный радиус) .

В общем, полагают, есть класс, называемый AClass с переменной-полем данных под названием aVariable и способом доступа методом aMethod ( ). Экземпляр называется anInstance и строится для AClass.

Вы можете использовать:

anInstance.aVariable и anInstance.aMethod ().

5. Переменные - поля данных класса

Переменная-поле данных имеет имя (или идентификатор) и тип ; и имеет значение определенного типа (с таблицей на предыдущей главе).

Переменная-поле данных может также быть экземпляром определенного класса (которые будут

обсуждаться позже).

Конвенция об именах переменных: имя переменной должно быть существительным или словосочетанием из нескольких слов. Первое слово в нижнем регистре , а остальные слова пишутся с Прописной букы (двугорбая нотация), например , размер шрифта, roomNumber, Xmax , Ymin и xTopLeft .

Обратите внимание, что имя переменной начинается с буквы в нижнем регистре, в то время как имя класса начинается с заглавной буквы.

Формальный синтаксис для определения переменной в Java:

*[AccessControlModifier] type variableName [= initialValue]**;***

*[AccessControlModifier] type variableName-1 [= initialValue-1] [**,** type variableName-2 [= initialValue-2]] ...* ***;***

Например:

```
1  private double radius;
2  public int length = 1, width =
   1;
```

6. Методы класса

Метод (способ как описано в предыдущем разделе):

- принимает параметры из вызова (как в функции);
- выполняет операции, описанные в теле метода, и;
- возвращает часть результата (или void) в точку вызова.

Формальный синтаксис объявления метода в Java:

```
1 [AccessControlModifier] returnType methodName ([argumentList]) {
2    // method body or implementation
3    ......
4 }
```

Например:

```
1 public double getArea() {
2    return radius*radius*Math.PI;
3 }
```

Конвенция записи имен методов :

Имя метода должно быть глаголом или начинаться глаголом в виде фразы из нескольких слов. Первое слово в нижнем регистре, а остальные слова начинаются с Прописной буквы (двугорбая запись). Например, getRadius ( ) , getParameterValues ( ) .

Обратите внимание, что имя переменной существительное (обозначающий статический атрибут) , в то время как имя метода - глагол (обозначает действие) . Они имеют те же наименования. Тем не менее, вы можете легко отличить их от контекста. Методы могут принимать аргументы в скобках (возможно, нулевой аргумент, в пустых скобках), ноне поля данных. При записи, методы обозначаются парой круглых скобок, например , Println ( ) , GetArea ( ).

7. Теперь соберем все вместе: Пример ООП

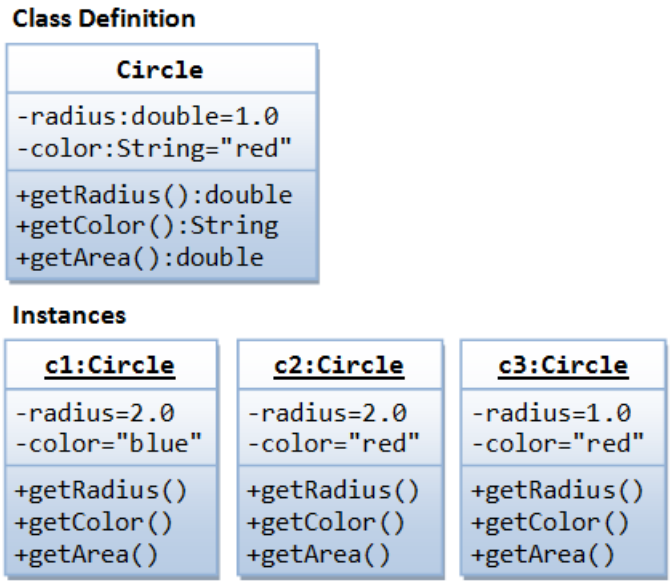

Диаграмма класса и экземпляров класса на рисунке выше

Класс называется Circle и должен быть определен , как показано на диаграмме классов .

Он содержит две переменные: radius (типа double) и color (типа String); и три метода: getRadius ( ) , GetColor ( ) , и GetArea ( ) .

Три экземпляра Circle называются C1 , C2, C3 , и должны быть построены с учетом их соответствующих элементов данных и методов, как показано на схемах UML для экземпляров класса.

Исходные коды для Circle.java:

Circle.java

```
 1 // Define the Circle class
 2 public class Circle {    // Save as "Circle.java"
 3    // Private variables
 4    private double radius;
 5    private String color;
 6
 7    // Constructors (overloaded)
 8    public Circle() {                   // 1st Constructor
 9       radius = 1.0;
10       color = "red";
11    }
12    public Circle(double r) {           // 2nd Constructor
13       radius = r;
14       color = "red";
15    }
16    public Circle(double r, String c) { // 3rd Constructor
17       radius = r;
18       color = c;
19    }
20
```

```
21    // Public methods
22    public double getRadius() {
23       return radius;
24    }
25    public String getColor() {
26       return color;
27    }
28    public double getArea() {
29       return radius*radius*Math.PI;
30    }
31 }
```

Компиляция "Circle.java" в "Circle.class".

Обратите внимание, что в классе Circle нет метода Main () . Следовательно, это не будет программой на Java, и вы не можете запустить класс Circle сам по себе. Класс Circle нужен, чтобы быть строительным блоком и использоваться в других программах.

Дополним нашу программу еще одним классом, который будет демонстрировать работу с нашим классом. Мы напишем TestCircle, в котором будем использовать Circle класс. Класс TestCircle в котором есть метод main() и мы можем теперь запустить программу.

TestCircle.java

```
public class TestCircle {     // Save as "TestCircle.java"
   public static void main(String[] args) {   // Execution entry
1
2  point
3        // Construct an instance of the Circle class called c1
4        Circle c1 = new Circle(2.0, "blue");     // Use 3rd constructor
5        System.out.println("Radius is " + c1.getRadius()  // use dot
6  operator to invoke member methods
7           + " Color is " + c1.getColor()
8           + " Area is " + c1.getArea());
9
10       // Construct another instance of the Circle class called c2
11       Circle c2 = new Circle(2.0);          // Use 2nd constructor
12       System.out.println("Radius is " + c2.getRadius()
13          + " Color is " + c2.getColor()
14          + " Area is " + c2.getArea());
15
16       // Construct yet another instance of the Circle class called
17 c3
18       Circle c3 = new Circle();             // Use 1st constructor
19       System.out.println("Radius is " + c3.getRadius()
20          + " Color is " + c3.getColor()
21          + " Area is " + c3.getArea());
   }
}
```

Запустим TestCircle и увидим результат:

Radius is 2.0 Color is blue Area is 12.566370614359172

Radius is 2.0 Color is red Area is 12.566370614359172

Radius is 1.0 Color is red Area is 3.141592653589793

8. Конструкторы

Конструктор - специальный метод класса, который имеет то же имя, что используется в качестве имени класса. В приведенном выше класса Circle, мы определим три перегруженных версии конструктора " Circle(...) " . Конструктор используется для создания и инициализации всех переменных-полей данных класса. Чтобы создать новый экземпляр класса, вы должны использовать специальный оператор "new" с последующим вызовом одного из конструкторов.

Например,

```
1 Circle c1 = new Circle();
2 Circle c2 = new Circle(2.0);
3 Circle c3 = new Circle(3.0, "red");
```

Конструктор отличается от обычного метода в следующих аспектах:

- название метода-конструктора совпадает с именем класса, а имя класса по конвенции, начинается с заглавной буквы .
- Конструктор не имеет возвращаемого значения типа (или неявно не возвращает). Таким образом, нет объявления типа возвращаемого значения при объявлении.
- Конструктор может быть вызван только через оператор «new». Он может быть использован только один раз, чтобы инициализировать построенный экземпляр. Вы не можете впоследствии вызвать конструктор в теле программы подобно обычным методам (функциям).
- Конструкторы не наследуется (будет объяснено позже).

Конструктор без параметров называется конструктором по умолчанию, который инициализирует переменные-поля данных через их значения по умолчанию. Например, Circle( ) в приведенном выше примере.

9. Перегрузка методов

Перегрузка методов означает, что несколько методов могут иметь то же самое имя метод, но сами методы могут иметь различные реализации (версии) . Тем не менее, различные реализации должны быть различимы по их списком аргументов (либо количество аргументов , или типа аргументов , или их порядок).

Пример: метод average( ) имеет 3 версии с различными списками аргументов. При вызове может использоваться соответствующий выбору вариант, в соответствии с аргументами.

```
1 public class TestMethodOverloading {
2    public static int average(int n1, int n2) {          // A
3       return (n1+n2)/2;
4    }
5
6    public static double average(double n1, double n2) { // B
7       return (n1+n2)/2;
8    }
9
```

```
10    public static int average(int n1, int n2, int n3) {   // C
11       return (n1+n2+n3)/3;
12    }
13
14    public static void main(String[] args) {
15       System.out.println(average(1, 2));      // Use A
16       System.out.println(average(1.0, 2.0)); // Use B
17       System.out.println(average(1, 2, 3));   // Use C
18       System.out.println(average(1.0, 2));    // Use B - int 2
19 implicitly casted to double 2.0
20       // average(1, 2, 3, 4); // Compilation Error - No matching
21 method
22    }
   }
```

Перегрузка конструктора класса Circle

Приведенный выше класс Circle имеет три версии конструктора, которые отличаются списком их параметров, следовательно:

```
1 Circle()
2 Circle(double r)
3 Circle(double r, String c)
```

В зависимости от фактического списка аргументов , используемых при вызове метода, будет вызван соответствующий конструктор. Если ваш список аргументов не соответствует ни одному из определенных методов, вы получите ошибку компиляции.

10. Public или Private – модификаторы контроля доступа

Контроль за доступом осуществляется с помощью модификатора, он может быть использован для управления видимостью класса или переменных –полей или методов внутри класса. Мы начнем со следующих двух модификаторов управления доступом:

public : класс / переменная / метод доступным и для всех других объектов в системе.

private : класс / переменная / метод доступным и в пределах только этого класса.

Например, в приведенном выше определении Circle, radius переменная-поле данных класса объявлена private . В результате ,radius доступен внутри класса Circle , но не внутри класса TestCircle . Другими словами, вы не можете использовать "*c1.radius*" по отношению к радиусу C1 в классе TestCircle . Попробуйте вставить текст "*System.out.println (c1.radius) ;*" в TestCircle и понаблюдать за сообщением об ошибке.

Попробуйте изменить radius на public, и повторно запустить пример.

С другой стороны, метод getRadius ( ) определяется как public в классе Circle . Таким образом, он может быть вызван в классе TestCircle.

В нотации UML на диаграмме классов обозначается: общественные (public) элементы обозначены со знаком " + " , в то время как частные (private) элементы со знаком " - ".

11. Информация по сокрытию реализации и инкапсуляции

Класс инкапсулирует имя, статические атрибуты и динамического поведение в "3-части бокса - коробки". После того, как класс определен, вы можете запечатать "крышку" и поставить "коробку" на полку для использования другими и использовать самим. Любой желающий может снять "крышку" и использовать это в своем приложении. Это не может быть сделано в традиционном процедурном-ориентированного языка как C, так как статические атрибуты (или переменные) разбросаны на протяжении всей программы и файлов заголовков. Вы не можете "вырезать" куски из части программы на C, подключить в другую программу и ожидает, что запуск программы произойдет без серьезных изменений.

Переменные-поля данных класса, как правило, скрыты от внешнего слоя (то есть, другие классы), с контролем через private доступ модификатора. Доступ к переменным полям класса предоставляются через методы public, например, getRadius () и GetColor ().

Это соответствует принципу сокрытия информации. То есть, объекты могут общаться друг с другом, используя хорошо определенные интерфейсы (публичные методы). Объектам не позволено знать детали реализации других объектов. Детали реализации скрыты или инкапсулированы внутри класса. Сокрытие информации облегчает повторное использование класса.

Основное правило: Не делайте никаких переменных public, если у вас на то нет веских оснований.

ЗАДАНИЯ

Необходимо реализовать простейший класс на языке программирования Java. Добавить метод ToString. Создать класс-тестер для вывода информации об объекте.

Упражнение 1.

Реализуйте простейший класс «Мяч»

Упражнение 2.

Реализуйте простейший класс «Книга»

ПРИМЕР ВЫПОЛНЕНИЯ ЗАДАНИЯ

Задание: Разработка и реализация класса Dog, который содержит данные экземпляра, которые представляют имя собаки и возраст . Определить конструктор собаки, чтобы принять и инициализировать данные экземпляр. Включите в класс методы получения и записи данных в поля имени и возраста (геттеры и сеттеры). Включите в класс метод для вычисления возраста собаки, метод возвращает возраст собаки в "человеческих" летах (если возраст собаки умножить на семь, то получим возраст человека). Включите метод ToString(), который возвращает строчку в одну описание объекта-собаки. Назначение класса Driver или по другому Tester (можно также использовать для этого класса название - ПитомникСобак, но на англ. языке) демонстрировать работу программы, следовательно в этот класс необходимо поместить основной метод main(), внутри которого создать несколько объектов класса Dog и продемонстрировать работу разработанных в классе методов.

Код программы:

Dog.java

```
import java.lang.*;
public class Dog {
    private String name;
```

```
    private int age;

    public Dog(String n, int a){
        name = n;
        age = a;
    }
    public Dog(String n){
        name = n;
        age = 0;
    }
    public Dog(){
        name = "Pup";
        age = 0;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getName(String name){
        return name;
    }

    public int getAge() {
        return age;
    }

    public String toString(){
        return this.name+", age "+this.age;
    }

    public void intoHumanAge(){
        System.out.println(name+"'s age in human years is "+age*7+"
years");
    }
}
```

Nursery.java

```
import java.lang.*;
public class Nursery {
    public static void main(String[] args) {
        Dog d1 = new Dog("Mike", 2);
        Dog d2 = new Dog("Helen", 7);
        Dog d3 = new Dog("Bob");
        d3.setAge(1);
        System.out.println(d1);
        d1.intoHumanAge();
        d2.intoHumanAge();
        d3.intoHumanAge();
    }
}
```