Индивидуальная работа № 1. КЛАССЫ И ОБЪЕКТЫ В JAVA

Цель работы. Получить практические навыки разработки программ использованием объектно-ориентированного подхода на языке Java, создавать классы и объекты.

3.1. Теоретический материал

3.1.1 Концепции объектно-ориентированного программирования

Что такое объект?

Языки структурного программирования, такие как Си и Pascal, следуют совсем иной парадигме программирования, чем объектно-ориентированные языки. Парадигма структурного программирования ориентирована на данные, что означает, что сначала создаются структуры данных, а затем пишутся команды для работы с этими данными. В объектно-ориентированных языках, таких как Java, данные и команды программы скомбинированы в *объекты.*

Объект представляет собой автономный модуль со своими атрибутами и поведением. Вместо структуры данных с полями (атрибуты), которая отражается на всей логике программы, влияющей на ее поведение, в объектно-ориентированном языке данные и логика программы объединены. Эта комбинация может быть реализована на совершенно разных уровнях детализации, от самых мелких объектов, до самых крупных.

Родительские и дочерние объекты

Родительский объект служит в качестве структурной основы для получения более сложных дочерних объектов. Дочерний объект повторяет родительский, но является более специализированным. Объектно-ориентированная парадигма позволяет многократно использовать общие атрибуты и поведение родительского объекта, добавляя к ним новые атрибуты и поведение дочерних объектов.

Связь между объектами и координация

Объекты общаются друг с другом, отправляя сообщения (на языке Java - вызовы методов). Кроме того, в объектно-ориентированных приложениях программа координирует взаимодействие между объектами для решения задачи в контексте данной предметной области.

Краткие сведения об объектах

Хорошо написанный объект:

- имеет четкие границы;
- имеет конечный набор действий;
- "знает" только о своих данных и любых других объектах, которые нужны для его деятельности.

По сути, объект - это дискретный модуль, который обладает только необходимыми зависимостями от других объектов для решения собственных задач.

Теперь посмотрим, как выглядит объект.

Объект Person

Начнем с примера, основанного на общем сценарии разработки приложений: физического лица, представленного объектом *Person.*

Возвращаясь к определению объекта, вспомним, что объект состоит из двух основных элементов: атрибутов и поведения. Посмотрим, как это относится к объекту *Person*.

Атрибуты (Свойства)

Какие атрибуты может иметь физическое лицо? Вот самые распространенные из них:

- имя,
- возраст,
- рост,
- вес,
- цвет глаз,
- пол.

Можно придумать еще (дополнительные атрибуты всегда можно добавить), но для начала этого достаточно.

Поведение

Физическое лицо может делать все что угодно, но поведение объекта обычно относится к контексту какого-то приложения. Например, в контексте бизнес-приложения можно задать своему объекту *Person* вопрос: "Сколько вам лет?" В ответ на это *Person* сообщит значение своего атрибута "возраст".

Внутри объекта *Person* может быть скрыта и более сложная логика, но пока предположим, что *Person* умеет отвечать на вопросы:

- Как вас зовут?
- Сколько вам лет?
- Ваш рост?
- Ваш вес?
- Какого цвета ваши глаза?
- Какого вы пола?

Состояние и строка

Состояние – важное понятие в ООП. В каждый момент времени состояние объекта представлено значением его атрибутов.

В случае объекта Person его состояние определяется такими атрибутами, как имя, возраст, рост и вес.

3.1.2 Структура объекта Java

Объект представляет собой дискретный модуль со своими атрибутами и поведением. Это означает, что он имеет четкие границы и состояние и может выполнять разнообразные действия, если к нему правильно обратиться. В каждом объектно-ориентированном языке есть правила определения объектов.

В языке Java объекты определяются, как показано в листинге 4.1.

Листинг 3.1. Определение объекта

```
package packageName;

import ClassNameToImport;
accessSpecifier class ClassName {
  accessSpecifier dataType variableName [= initialValue];
  accessSpecifier ClassName([argumentList]) {
```

```
    constructorStatement(s)
  }
  accessSpecifier returnType methodName([argumentList]) {
    methodStatement(s)
  }
  // Это комментарий
  /* Это тоже комментарий */
}
```

где:

– *packageName* – наименование пакета в котором находится класс;

– *ClassNameToImport* – наименование импортируемого класса или пакета;

– *accessSpecifier* **–** модификатор доступа;

– *ClassName* – наименование класса;

– *dataType* – тип атрибута;

– *variableName* – наименование атрибута;

– *initialValue* – начальная инициализация объекта;

– *argumentList* – список передаваемых аргументов;

– *constructorStatement* – операторы конструктора;

– *returnType* – возвращаемый тип;

– *methodName* – наименование метода;

– *methodStatement* – операторы метоа.

В листинге 3.1 содержатся конструкции разного типа. Конструкции, показанные жирным шрифтом – литералы (зарезервированные слова); в любом определении объекта они должны быть точно такими же, как здесь.

Упаковка объектов

Язык Java позволяет выбирать имена объектов, такие как Account, Person или LizardMan. Время от времени может получаться так, что одно и то же имя должно выражать два разных понятия. Это называется *конфликтом имен* и происходит довольно часто. Для разрешения таких конфликтов в языке Java используются *пакеты*.

Пакет Java – это механизм организации пространства имен: ограниченная область, в которой имена уникальны, но вне которой они могут не существовать. Чтобы определить конструкцию как уникальную, нужно полностью описать ее, указав ее пространство имен.

Пакеты предоставляют также хороший способ для построения более сложных приложений из дискретных единиц функциональности.

Определение пакета

Для определения пакета используется ключевое слово **package**, за которым следует формальное имя пакета, заканчивающееся точкой с запятой (***packageName***). Часто имена пакетов разделяются точками и следуют такой схеме *де-факто*:

package *orgType.orgName.appName.compName;*

– Это определение пакета расшифровывается так:

– ***orgType*** – это тип организации, такой как com, org или net.

– ***OrgName –*** доменное имя организации, такое как makotogroup, sun или ibm;

– ***AppName*** – сокращенное имя приложения;

– ***compName*** – имя компонента.

Язык Java не вынуждает следовать этому соглашению о пакетах. На самом деле, определять пакет вообще не обязательно, но тогда все объекты должны иметь уникальные имена классов и будут находиться в пакете по умолчанию.

Операторы импорта

Следующим в определении объекта (возвращаясь к листингу 3.1) идет *оператор import*. Он сообщает компилятору Java, где найти классы, на которые ссылается код. Любой нетривиальный объект использует другие объекты для выполнения тех или иных функций, и оператор импорта позволяет сообщить о них компилятору Java.

Оператор импорта обычно выглядит так:

```
import ClassNameToImport;
```

За ключевым словом **import** следуют класс, который нужно импортировать, и точка с запятой. Имя класса должно быть полным, то есть включать свой пакет.

Чтобы импортировать все классы из пакета, после имени пакета можно поместить .*. Например, следующий оператор импортирует каждый класс пакета **com.makotogroup:**

```
import com.makotogroup.*;
```

Импорт всего пакета может сделать код менее читабельным, поэтому рекомендуется импортировать только нужные классы.

Eclipse упрощает импорт

При написании кода в редакторе Eclipse можно ввести имя класса, а затем нажать Ctrl+Space. Eclipse определяет, какие классы нужно импортировать, и добавляет их автоматически. Если Eclipse находит два класса с одним и тем же именем, он выводит диалоговое окно с запросом, какой именно класс вы хотите добавить.

3.1.3. Объявление класса

Чтобы определить объект в языке Java, нужно объявить класс. Класс можно представить в качестве шаблона объектов, как формочка для печенья. Класс определяет базовую структуру объекта и во время выполнения программы создает экземпляр этого объекта.

Листинг 3.1 содержит следующее объявление класса:

```
accessSpecifier class ClassName {
  accessSpecifier dataType variableName [= initialValue];
  accessSpecifier ClassName([argumentList]) {
    constructorStatement(s)
  }
  accessSpecifier returnType methodName([argumentList]) {
    methodStatement(s)
  }
}
```

3.1.4 Соглашения об именах классов

В принципе классы можно называть как угодно, но по устоявшемуся соглашению используют *ВерблюжийСтиль*(CamelCase): начинают с прописной буквы, и первую букву каждого слова тоже делают прописной, а все остальные буквы – строчными. Имена классов должны содержать только буквы и цифры. Соблюдение этих принципов гарантирует, что ваш код будет понятен другим разработчикам, следующим тем же правилам.

У класса могут быть члены двух типов: *свойства* и *методы*.

Свойства

Значения переменных данного класса различны в разных экземплярах этого класса и определяют его состояние. Эти значения часто называют *свойства экземпляра*. Определение свойства содержит следующие элементы:

- ***accessSpecifier*;**
- ***dataType*;**
- ***variableName*;**
- а также может содержать ***initialValue*.**

Возможные значения ***accessSpecifier*:**

- **public:** переменную видит любой объект любого пакета;
- **protected:** переменную видит любой объект, определенный в том же пакете, или подкласс (определенный в любом пакете);
- без указателя (его еще называют доступом *для своих* или *внутри пакетным* доступом): переменную видят только объекты, класс которых определен в том же пакете;
- private: переменную видит только класс, ее содержащий.

Тип (***dataType*)** переменной зависит от того, что представляет собой переменная – это может быть простой тип или тип другого класса.

Имя переменной (***variableName***) зависит от вас, но по общему соглашению, в именах переменных используется *ВербюжийСтиль*, за исключением того, что начинаются они со строчной буквы. (Этот стиль иногда называют *lowerCamelCase*.)

Пример: определение класса *Person*

Прежде чем перейти к методам, приведем пример, суммирующий все, что вы усвоили до сих пор. Листинг 3.2 представляет собой определение класса Person.

Листинг 3.2. Определение класса *Person*

```
package com.makotogroup.intro;
public class Person {
  private String name;
  private int age;
  private int height;
  private int weight;
  private String eyeColor;
  private boolean gender;
}
```

Методы

Методы класса определяют его поведение. Иногда такое поведение – не более чем возврат (геттер, getter) текущего значения атрибута. В других случаях поведение может быть довольно сложным.

Существуют две категории методов: ***конструкторы*** и все прочие методы. Метод-конструктор используется только для создания экземпляра класса. Другие методы могут использоваться практически для любого поведения программы.

Возвращаясь к листингу 3.1, он иллюстрирует способ определения структуры метода, который включает в себя такие вещи, как:

***accessSpecifier*;**
***returnType*;**
***methodName*;**
***argumentList*.**

Сочетание этих структурных элементов в определении метода называется его *сигнатурой*.

Рассмотрим оба типа методов подробнее, начиная с конструкторов.

Методы-конструкторы

Конструкторы позволяют указать, как создавать экземпляр класса. В листинге 1 был показан синтаксис декларации конструктора в абстрактной форме; он еще раз приведен в листинге 3.3.

Листинг 3.3 Синтаксис декларации конструктора

```
accessSpecifier ClassName([argumentList]) {
  constructorStatement(s)
}
```

Если конструктор отсутствует, компилятор создаст конструктор по умолчанию (или *без аргументов*). Но он не станет его генерировать, если указать другой конструктор, отличный от конструктора без аргументов.

accessSpecifier для конструктора тот же, что и для переменных. Имя конструктора должно совпадать с именем класса. Так что если мы назвали свой класс *Person*, то и имя конструктора должно быть ***Person*.**

Для любого конструктора, кроме конструктора по умолчанию, создается список аргументов ***argumentList***, который содержит один или более аргументов:

```
argumentType argumentName
```

Аргументы в списке ***argumentList*** разделены запятыми, и два аргумента не могут носить одно и то же **имя.*argumentType* —** это либо простой тип, либо тип класса (так же, как в случае с типами переменных).

Определение класса с помощью конструктора

Теперь посмотрим, что происходит, когда появляется возможность создания объекта ***Person*** двумя способами: с использованием конструктора без аргументов и с инициализацией неполного списка атрибутов.

В листинге 3.4 показано, как создавать конструкторы и как использовать argumentList.

Листинг 3.4. Определение класса Person **с помощью конструктора**

```
package com.makotogroup.intro;
public class Person {
  private String name;
  private int age;
  private int height;
  private int weight;
  private String eyeColor;
  private boolean gender;
  public Person() {
    // Делать больше нечего...
  }
public Person(String name, int age, int height, String eyeColor, boolean gender)
{
    this.name = name;
    this.age = age;
    this.height = height;
    this.weight = weight;
    this.eyeColor = eyeColor;
    this.gender = gender;
  }
}
```

Обратите внимание на использование ключевого слова **this** при присвоении значений переменным в листинге 3.4. В Java оно означает **"this object"** (этот объект) и служит для обращения к двум переменным с одинаковыми именами (как в данном случае, когда **age** – это и параметр конструктора, и переменная класса), а также помогает компилятору при неоднозначности ссылки.

У любого класса есть **конструктор по умолчанию**, например:

public Person() { }

Но при указании нового конструктора, конструктор по умолчанию становится не доступным до тех пор, пока его явно не указать.

Другие методы

Конструктор – это метод особого рода с особой функцией. Точно так же методы многих других видов выполняют конкретные обязанности в Java-программах.

В листинге 3.1 был показан синтаксис объявления метода:

```
accessSpecifier returnType methodName([argumentList]) {
  methodStatement(s)
}
```

Другие методы, за несколькими исключениями, выглядят так же, как конструкторы. Во-первых, другие методы можно называть, как вам заблагорассудится, но существуют следующие соглашения:

- начинайте со строчной буквы;
- избегайте цифр без крайней необходимости;
- используйте только буквы.

Другие методы объекта Person

Можно посмотреть в листинге 3.5, что произойдет при добавлении к объекту Person еще нескольких методов.

Листинг 3.5. *Person* **с несколькими новыми методами**

```
package com.makotogroup.intro;

public class Person {
  private String name;
  private int age;
  private int weight;
  private boolean gender;

  public String getName() { return name; }
  public void setName(String value) { name = value; }
  // Другие комбинации геттеров/сеттеров...
}
```

Обратите внимание на комментарий в листинге 5 о "комбинациях геттеров/сеттеров". ***Геттер*** – это метод для получения значения атрибута, а ***сеттер*** – для изменения этого значения. В листинге 3.5 показана одна комбинацию геттер/сеттер (для атрибута Name), но аналогичным образом можно определить и другие.

Методы экземпляра и статические методы

Существуют два основных типа методов (кроме конструкторов): *методы экземпляра (обычные методы)* и *методы класса (статические методы)*. Поведение метода экземпляра зависит от состояния конкретного экземпляра объекта. Статические методы иногда еще называют *методами класса*, так как их поведение не зависит от состояния какого-либо одного объекта. Поведение статического метода определяется на уровне класса.

Статические методы используются в основном для удобства, их можно представить как способ создания глобальных методов с сохранением при этом самого кода сгруппированным с классом, которому они нужны.

Метод toString()

Этот метод служит для представления объекта в виде строки. Это требуется, например, если необходимо вывести объект на экран.

Самое главное знать, что метод **toString()** есть у всех объектов и все объекты используют этот метод при работе со строками. Этот метод является методом класса Object. В случаи, когда от объекта требуется результат типа **String**, например: ***System.out.println(new Object()),*** этот метод вызывается автоматически. Он возвращает представление объекта в виде строки и по-умолчанию состоит из двух составляющих разделенных собачкой. Эти составляющие: **имя_класса_объекта** и **хэш_кода**. Пример: **java.lang.Integer;@24d200d8**.

Но это метод можно переопределить в своем классе и вывести более подробную информацию (листинг 3.6).

```
package com.makotogroup.intro;
public class Person {
```

```
private String name;
private int age;
private int weight;
private boolean gender;
public Person(String name, int age, int weight, boolean gender) {
        this.name = name;
        this.age = age;
        this.weight = weight;
        this.gender = gender;
}
public String getName() {
        return name;
}
public void setName(String name) {
        this.name = name;
}
public int getAge() {
        return age;
}
public void setAge(int age) {
        this.age = age;
}
public int getWeight() {
        return weight;
}
public void setWeight(int weight) {
        this.weight = weight;
}
public boolean getGender() {
        return gender;
}
public void setGender(boolean gender) {
        this.gender = gender;
}
public String toString()
{
   return "My name is " + name + " I'm " + age + " years old.";
}
}

package com.makotogroup.intro;
public class TestPerson {
public static void main(String[] args) {
        Person person = new Person("Иван", 20, 175, true);
        System.out.println(person.toString());
        System.out.println(person);
}
}
```

Оператор *new* создает экземпляр (объект) указанного класса и возвращает ссылку на вновь созданный объект. Ниже приведен пример создания и присваивание переменной *person* экземпляра класса *Person.*

Person person = new Person("Иван", 20, 175, true);

Где:

- ***Person("Иван", 20, 175, true)*** – является конструктором.

Можно создать несколько ссылок на один и тот же объект. Например:

Person person1 = new Person("Иван", 20, 175, true);

Person person2 = person1;