Контрольная работа - 4 семестр

Классы и объекты

Java является объектно-ориентированным языком, поэтому такие понятия как "класс" и "объект" играют в нем ключевую роль. Любую программу на Java можно представить как набор взаимодействующих между собой объектов.

Шаблоном или описанием объекта является класс, а объект представляет экземпляр этого класса. Можно еще провести следующую аналогию. У нас у всех есть некоторое представление о человеке - наличие двух рук, двух ног, головы, туловища и т.д. Есть некоторый шаблон - этот шаблон можно назвать классом. Реально же существующий человек (фактически экземпляр данного класса) является объектом этого класса.

Класс определяется с помощью ключевого слова class:

```
class Person{

}
```

В данном случае класс называется Person. После названия класса идут фигурные скобки, между которыми помещается тело класса - то есть его поля и методы.

Любой объект может обладать двумя основными характеристиками: состояние - некоторые данные, которые хранит объект, и поведение - действия, которые может совершать объект.

Для хранения состояния объекта в классе применяются поля или переменные класса. Для определения поведения объекта в классе применяются методы. Например, класс Person, который представляет человека, мог бы иметь следующее определение:

```
class Person{

String name; // имя
int age; // возраст
void displayInfo(){
System.out.printf("Name: %s \tAge: %d\n", name, age);
}
}
```

В классе Person определены два поля: name представляет имя человека, а age - его возраст. И также определен метод displayInfo, который ничего не возвращает и просто выводит эти данные на консоль.

Теперь используем данный класс. Для этого определим следующую программу:

```
public class Program{

public static void main(String[] args) {

Person tom;
}
}
class Person{

String name; // имя
int age; // возраст
void displayInfo(){
System.out.printf("Name: %s \tAge: %d\n", name, age);
}
}
```

Как правило, классы определяются в разных файлах. В данном случае для простоты мы определяем два класса в одном файле. Стоит отметить, что в этом случае только один класс может иметь модификатор public (в данном случае это класс Program), а сам файл кода должен называться по имени этого класса, то есть в данном случае файл должен называться Program.java.

Класс представляет новый тип, поэтому мы можем определять переменные, которые представляют данный тип. Так, здесь в методе main определена переменная tom, которая представляет класс Person. Но пока эта переменная не указывает ни на какой объект и по умолчанию она имеет значение null. По большому счету мы ее пока не можем использовать, поэтому вначале необходимо создать объект класса Person.

Конструкторы

Кроме обычных методов классы могут определять специальные методы, которые называются конструкторами. Конструкторы вызываются при создании нового объекта данного класса. Конструкторы выполняют инициализацию объекта.

Если в классе не определено ни одного конструктора, то для этого класса автоматически создается конструктор без параметров.

Выше определенный класс Person не имеет никаких конструкторов. Поэтому для него автоматически создается конструктор по умолчанию, который мы можем использовать для создания объекта Person. В частности, создадим один объект:

```
public class Program{

public static void main(String[] args) {

Person tom = new Person(); // создание объекта
tom.displayInfo();

// изменяем имя и возраст
tom.name = "Tom";
tom.age = 34;
tom.displayInfo();
}
}
class Person{

String name; // имя
int age; // возраст
void displayInfo(){
System.out.printf("Name: %s \tAge: %d\n", name, age);
}
}
```

Для создания объекта Person используется выражение new Person(). Оператор new выделяет память для объекта Person. И затем вызывается конструктор по умолчанию, который не принимает никаких параметров. В итоге после выполнения данного выражения в памяти будет выделен участок, где будут храниться все данные объекта Person. А переменная tom получит ссылку на созданный объект.

Если конструктор не инициализирует значения переменных объекта, то они получают значения по умолчанию. Для переменных числовых типов это число 0, а для типа string и классов - это значение null (то есть фактически отсутствие значения).

После создания объекта мы можем обратиться к переменным объекта Person через переменную tom и установить или получить их значения, например, tom.name = "Tom".

В итоге мы увидим на консоли:

```
Name: null Age: 0
Name: Tom Age: 34
```

Если необходимо, чтобы при создании объекта производилась какая-то логика, например, чтобы поля класса получали какие-то определенные значения, то можно определить в классе свои конструкторы. Например:

```
public class Program{

public static void main(String[] args) {

Person bob = new Person(); // вызов первого конструктора без параметров
bob.displayInfo();

Person tom = new Person("Tom"); // вызов второго конструктора с одним параметром
tom.displayInfo();

Person sam = new Person("Sam", 25); // вызов третьего конструктора с двумя параметрами
sam.displayInfo();
}
}
class Person{

String name; // имя
int age; // возраст
Person()
{
```

```
name = "Undefined";
age = 18;
}
Person(String n)
{
name = n;
age = 18;
}
Person(String n, int a)
{
name = n;
age = a;
}
void displayInfo(){
System.out.printf("Name: %s \tAge: %d\n", name, age);
}
}
```

Теперь в классе определено три коструктора, каждый из которых принимает различное количество параметров и устанавливает значения полей класса.

Консольный вывод программы:

```
Name: Undefined Age: 18
Name: Tom Age: 18
Name: Sam Age: 25
```

Ключевое слово this

Ключевое слово this представляет ссылку на текущий экземпляр класса. Через это ключевое слово мы можем обращаться к переменным, методам объекта, а также вызывать его конструкторы. Например:

```
public class Program{

public static void main(String[] args) {

Person undef = new Person();
undef.displayInfo();

Person tom = new Person("Tom");
tom.displayInfo();

Person sam = new Person("Sam", 25);
sam.displayInfo();
}
}
class Person{

String name; // имя
int age; // возраст
Person()
{
this("Undefined", 18);
}
Person(String name)
{
this(name, 18);
}
Person(String name, int age)
{
this.name = name;
this.age = age;
}
void displayInfo(){
System.out.printf("Name: %s \tAge: %d\n", name, age);
}
}
```

В третьем конструкторе параметры называются так же, как и поля класса. И чтобы разграничить поля и параметры, применяется ключевое слово this:

this.name = name;
Так, в данном случае указываем, что значение параметра name присваивается полю name.

Кроме того, у нас три конструктора, которые выполняют идентичные действия: устанавливают поля name и age. Чтобы избежать повторов, с помощью this можно вызвать один из конструкторов класса и передать для его параметров необходимые значения:

```
Person(String name)

{
this(name, 18);
}
```

В итоге результат программы будет тот же, что и в предыдущем примере.

Инициализаторы

Кроме конструктора начальную инициализацию объекта вполне можно было проводить с помощью инициализатора объекта. Инициализатор выполняется до любого конструктора. То есть в инициализатор мы можем поместить код, общий для всех конструкторов:

```
public class Program{

public static void main(String[] args) {

Person undef = new Person();
undef.displayInfo();

Person tom = new Person("Tom");
tom.displayInfo();
}
}
class Person{

String name; // имя
int age; // возраст

/*начало блока инициализатора*/
{
name = "Undefined";
age = 18;
}
/*конец блока инициализатора*/
Person(){

}
Person(String name){

this.name = name;
}
Person(String name, int age){

this.name = name;
this.age = age;
}
void displayInfo(){
System.out.printf("Name: %s \tAge: %d\n", name, age);
}
}
```

Консольный вывод:

```
Name: Undefined Age: 18
Name: Tom Age: 18
```

Интерфейсы

◄ Объявления

Перейти на...