

КЛАССЫ И ОБЪЕКТЫ

Java - объектно-ориентированный язык, в котором все структуры данных, кроме примитивов, - объекты (ссылочные типы).

Программы строятся на взаимодействии объектов, объекты могут взаимодействовать между собой с помощью методов.

Объекты создаются на основе классов, в которых перечисляются методы (возможности объектов) и свойства (характеристики объектов).

Поэтому создание объекта всегда начинается с написания класса.

Обычно каждый класс хранится в отдельном исходном файле (*.java), для их удобной организации и разрешения конфликтов имен используются пакеты.

Пакет — это каталог с классами.

Пакеты часто именуются как доменное имя компании наоборот (начиная с корневого домена). Например, класс `BattleUnit` лежит в пакете `com.ifmo.battle` или класс `Arrays` лежит в пакете `java.util`.

Таким образом, **полное имя класса** будет содержать название пакета и имя класса. Например, `com.ifmo.battle.BattleUnit` или `java.util.Arrays`. Это позволит использовать в программе множество классов с именем `BattleUnit`, но в разных пакетах, например, `org.nicegame.unit.BattleUnit` и `com.ifmo.battle.BattleUnit`.

Но пользоваться такими длинными именами неудобно, поэтому используют ключевое слово `import`, которое позволяет вынести названия пакета из имени класса.

Имя класса и import

```
1 // полное имя класса
2 com.itmo.battle.BattleUnit u = new com.itmo.battle.BattleUnit(9, 3);
3 java.util.Scanner sc = new java.util.Scanner(System.in);
4
5 // имя класса с использованием import
6 import com.itmo.battle.BattleUnit;
7 import java.util.Scanner;
8 ...
9 BattleUnit u = new BattleUnit(9, 3);
10 Scanner sc = new Scanner(System.in);
```

Но прежде, чем создать объект BattleUnit необходимо создать шаблон, по которому он будет создан, перечислить его характеристики и возможности. Для этого необходимо создать класс.

```
1 public class BattleUnit {  
2 }
```

Теперь можно создавать объекты

```
1 /* типДанных имяПеременной = new вызов_конструктора */  
2 BattleUnit unit = new BattleUnit();  
3 /* был создан объект типа BattleUnit (экземпляр класса BattleUnit),  
4 ссылка на него присвоена переменной unit */
```

Свойсва (поля, атрибуты) - характеристики будущих объектов, перечисленные в классе

```
1 public class BattleUnit {  
2     int health; // свойсво health типа int, значение по умолчанию 0  
3     int attack; // свойсво attack типа int, значение по умолчанию 0  
4 }
```

Доступ к свойсвам осуществляется по имени объекта через точку

```
1 BattleUnit unit = new BattleUnit();  
2 unit.health = 10; // свойству health объекта unit присвоили значение 10  
3 unit.attack = 5; // свойству attack объекта unit присвоили значение 5  
4 // вывели в консоль значение свойства health объекта unit  
5 System.out.println(unit.health); // 10
```

Создание объектов. Обращение к свойствам

```
1 BattleUnit knight = new BattleUnit();  
2 knight.health = 10;  
3 knight.attack = 5;  
4  
5 BattleUnit infantry = new BattleUnit();  
6 infantry.health = 15;  
7 infantry.attack = 3;
```

knight и infantry - объекты, которые созданы на основе одного класса и поэтому обладают одинаковыми свойствами: health и attack.

Но это разные объекты (разные юниты), поэтому значения свойств могут быть разными и не зависят друг от друга.

Методы (последовательность действий, инструкций) - возможности будущих объектов, перечисленные в классе

```
1  /* синтаксис */
2  /* метод выполняет инструкции и ничего не возвращает (void) */
3  void имяМетода(типДанных имяАргумента и тд) {
4      тело метода (инструкции)
5  }
6  /* метод выполняет инструкции и возвращает значение того типа данных,
7     который указан перед названием метода */
8  типДанных имяМетода(типДанных имяАргумента и тд) {
9      тело метода (инструкции)
10     return значение;
11 }
```

Добавим объектам BattleUnit возможность совершать действия

```
1  public class BattleUnit {
2      // возможность атаковать противника
3      void attackEnemy(BattleUnit enemy) {
4          // ссылка на объект противника будет передана в метод (в enemy)
5          // уменьшим количество здоровья противника
6          // на количество attack текущего объекта
7          enemy.health -= attack;
8      }
9      // возможность сообщить, о том, жив юнит или нет
10     boolean isAlive() {
11         // метод вернет true или false,
12         // в зависимости от значения health текущего объекта
13         return health > 0;
14     }
15 }
```

Добавим объектам BattleUnit возможность устанавливать значения свойств и проверять устанавливаемые значения

```
1 public class BattleUnit {
2     // при вызове метода необходимо будет передать значение здоровья,
3     // оно скопируется в переменную health
4     void setHealth(int health){
5         // this - ссылка на текущий (вызывающий метод) объект
6         // свойству health текущего объекта ( this.health )
7         // будет установлено переданное значение ( = health )
8         if (health < 3) { // проверка переданных данных
9             throw new IllegalArgumentException("Маленькое значение");
10        }
11        this.health = health; // установка значения свойства health
12    }
13    /* аналогично для свойства attack */
14 }
```

Вызов методов осуществляется по имени объекта через точку, после которой нужно указать имя метода и круглые скобки. При необходимости передать данные. При вызове метода выполняются инструкции, перечисленные в теле метода.

```
1 BattleUnit knight = new BattleUnit();
2 knight.setHealth(10);
3 knight.setAttack(5);
4
5 BattleUnit infantry = new BattleUnit();
6 infantry.setHealth(15);
7 infantry.setAttack(3);
8
9 knight.attackEnemy(infantry);
10 System.out.println(infantry.isAlive());
```

Перегрузка методов в классе можно объявить несколько методов с одинаковым именем, но с разными типами и/или количеством параметров.

```
1 public class BattleUnit {
2     /* свойства */
3     void attackEnemy(BattleUnit enemy) {
4         enemy.health -= attack;
5     }
6     void attackEnemy(BattleUnit... enemies) {
7         // три точки в BattleUnit... enemies означает,
8         // что enemies - массив
9         for (BattleUnit unit: enemies){
10             // можно вызвать один метод из другого
11             attackEnemy(unit);
12         }
13     }
14 }
```

Методы, которые позволяют установить значения свойств объекта (**сеттеры**) помогают обезопасить объект от установки невалидных данных. Но они становятся бессмысленными, если доступ к свойствам объекта возможен из вне, например, `unit.attack = -13`; Чтобы ограничить доступ к свойствам объекта используются **модификаторы доступа**.

Модификаторы доступа также используются с методами и классами.

Модификаторы доступа

- **public** - доступ возможен из любой точки программы
- **package private** - доступ возможен внутри пакета
- **protected** - доступ возможен из дочернего класса и внутри пакета
- **private** - доступ возможен только внутри класса

```
1 // ограничим доступ к свойствам
2 public class BattleUnit {
3     private int health; // свойство health типа int, значение по умолчанию
4     private int attack; // свойство attack типа int, значение по умолчанию
5 }
6 // теперь обращение к свойствам вне класса приведет к ошибке
7 knight.health = -100; // ошибка, нет доступа
8 // установить значение можно только с проверкой через сеттер
9 knight.setHealth(10);
```

После того, как доступ к свойствам был ограничен, мы столкнулись с проблемой: их нельзя прочитать

```
1 System.out.println(knight.health); // ошибка, нет доступа
```

Объектам нужно добавить возможность сообщить о значении свойств. Необходимо добавить методы, которые возвращают значения свойств - **геттеры**.

```
1 public class BattleUnit {  
2     int getHealth() {  
3         return health; // вернет значения свойства health  
4     }  
5     /* аналогично для свойства attack */  
6 }
```



```
1 public class BattleUnit {
2     private int health;
3     private int attack;
4     public void setHealth(int health) {
5         if (health <= 3) {
6             throw new IllegalArgumentException("health д.б. больше 3");
7         }
8         this.health = health;
9     }
10    /* аналогично для свойства attack */
11    public int getHealth() {
12        return health;
13    }
14    /* аналогично для свойства attack */
15    void attackEnemy(BattleUnit enemy) {
16        Objects.requireNonNull(enemy, "enemy не должен быть null");
17        enemy.health -= attack;
18    }
19    boolean isAlive() {
20        return health > 0;
21    }
22 }
```

В каждом классе есть как минимум один **конструктор**, который и вызывается при создании объекта. Роль конструктора — начальная инициализация объекта.

```
1 // у класса BattleUnit конструктор определен неявно,  
2 // и при компиляции класс развернется в такую структуру:  
3 public class BattleUnit {  
4     private int health;  
5     private int attack;  
6  
7     public BattleUnit() {  
8         // конструктор  
9     }  
10  
11     /* методы */  
12 }
```

Если для создания объекта вызывается конструктор без параметров, то значения свойств принимают значения по умолчанию.

- свойства типа `byte/short/int/long` - 0
- свойства типа `double/float` - 0.0
- свойства типа `boolean` - false
- свойства типа `char` - `'\u0000'`
- свойства `ССЫЛОЧНЫХ ТИПОВ` - null

```
1 // Например,
2 public class BattleUnit {
3     private int health = 4;
4     private int attack;
5
6     /* методы */
7 }
8 BattleUnit knight = new BattleUnit();
9 // значение health и attack принимают значения по умолчанию
10 // значение attack объекта knight равно 0,
11 // значение health объекта knight равно 4
12 BattleUnit infantry = new BattleUnit();
13 // значение health и attack принимают значения по умолчанию
14 // значение attack объекта infantry равно 0,
15 // значение health объекта infantry равно 4
16
17 // позже значения свойств можно задать/изменить через сеттеры
```

Перепишем класс таким образом, чтобы юнит сразу создавался с указанными здоровьем и атакой, т.е. у нас будет возможность **установить эти значения на этапе создания объекта** (при вызове конструктора). Добавим **конструктор с параметрами**.

```
1 public class BattleUnit {
2     private int health;
3     private int attack;
4
5     public BattleUnit(int health, int attack) {
6         this.health = health;
7         this.attack = attack;
8     }
9     /* методы */
10 }
11 BattleUnit knight = new BattleUnit(10, 5);
12 // значение attack объекта knight равно 5, health равно 10
```

Если в классе появляется хотя бы один конструктор с параметрами, то конструктор без параметров становится недоступен. При необходимости использовать оба конструктора, конструктор без параметров нужно указать явно.

```
1 public class BattleUnit {
2     private int health;
3     private int attack;
4
5     public BattleUnit(int health, int attack) {
6         this.health = health;
7         this.attack = attack;
8     }
9     public BattleUnit() {}
10 }
11 BattleUnit knight = new BattleUnit(10, 5);
12 BattleUnit infantry = new BattleUnit();
```

В классе может быть объявлено **несколько конструкторов с параметрами**, они должны отличаться набором аргументов.

```
1 public class BattleUnit {
2     private int health;
3     private int attack;
4
5     public BattleUnit(int health, int attack) {
6         this.health = health;
7         this.attack = attack;
8     }
9     public BattleUnit(int health) {
10        this.health = health;
11    }
12 }
13 BattleUnit knight = new BattleUnit(10, 5);
14 BattleUnit infantry = new BattleUnit(15);
```

Нельзя забывать про проверки входящих данных в конструкторах.

```
1 public class BattleUnit {
2     private int health;
3     private int attack;
4
5     public BattleUnit(int health, int attack) {
6         // можно написать проверку в конструкторе или вызвать сеттер
7         setHealth(health); // вызов метода setHealth
8         setAttack(attack); // вызов метода setAttack
9     }
10    public void setHealth(int health){
11        if (health <= 3) {
12            throw new IllegalArgumentException("задайте health больше
13        }
14        this.health = health;
15    }
16    /* аналогично для свойства attack */
17 }
```


В конструкторах можно вызывать не только методы, но и другие конструкторы.

```
1 public class BattleUnit {
2     private int health;
3     private int attack;
4
5     public BattleUnit(int health, int attack) {
6         setHealth(health); // вызов метода setHealth
7         setAttack(attack); // вызов метода setAttack
8     }
9     public BattleUnit(int health) {
10        // вызвали конструктор BattleUnit(int health, int attack)
11        this(health, 6);
12        // остальные инструкции конструктора
13    }
14    /* методы */
15 }
```

```
1 public class BattleUnit {
2     private int health;
3     private int attack = 4;
4
5     public BattleUnit() {}
6     public BattleUnit(int health, int attack) {
7         setHealth(health);
8         setAttack(attack);
9     }
10    public BattleUnit(int health) {
11        this(health, 6);
12    }
13    public void setHealth(int health){
14        if (health <= 3) throw new IllegalArgumentException("задайте health больше 3");
15        this.health = health;
16    }
17    /* аналогично для свойства attack */
18    public int getHealth(){
19        return health;
20    }
21    /* аналогично для свойства attack */
22    public void attackEnemy(BattleUnit enemy) {
23        Objects.requireNonNull(enemy, "enemy не должен быть null");
24        enemy.health -= attack;
25    }
26    public void attackEnemy(BattleUnit... enemies) {
27        Objects.requireNonNull(enemies, "enemies не должен быть null");
28        for (BattleUnit unit: enemies) attackEnemy(unit);
29    }
30    public boolean isAlive() {
31        return health > 0;
32    }
33 }
```

