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

```java
// полное имя класса
com.itmo.battle.BattleUnit u = new com.itmo.battle.BattleUnit(9, 3);
java.util.Scanner sc = new java.util.Scanner(System.in);

// имя класса с использованием import
import com.itmo.battle.BattleUnit;
import java.util.Scanner;
...
BattleUnit u = new BattleUnit(9, 3);
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 1
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 свойсвами: 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 дейсвия

```
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 установленно 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 свойсва 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; // свойсво health типа int, значение по умолча
4     private int attack; // свойсво attack типа int, значение по умолча
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

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

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

```
public class BattleUnit {
    int getHealth(){
        return health; // вернет значения свойства health
    }
    /* аналогично для свойсва attack */
}
```

```java
public class BattleUnit {
    private int health;
    private int attack;
    public void setHealth(int health){
        if (health <= 3) {
          throw new IllegalArgumentException("health д.б. больше 3");
        }
        this.health = health;
    }
    /* аналогично для свойсва attack */
    public int getHealth(){
        return health;
    }
    /* аналогично для свойсва attack */
    void attackEnemy(BattleUnit enemy) {
        Objects.requireNonNull(enemy, "enemy не должен быть null");
        enemy.health -= attack;
    }
    boolean isAlive() {
        return health > 0;
    }
}
```

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

```
// у класса BattleUnit конструктор определен неявно,
// и при компиляции класс развернется в такую структуру:
public class BattleUnit {
    private int health;
    private int attack;

    public BattleUnit(){
        // конструктор
    }

    /* методы */
}
```

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

```
// Например,
public class BattleUnit {
    private int health = 4;
    private int attack;

    /* методы */
}
BattleUnit knight = new BattleUnit();
// значение health и attack принимают значения по умолчанию
// значение attack объекта knight равно 0,
// значение health объекта knight равно 4
BattleUnit infantry = new BattleUnit();
// значение health и attack принимают значения по умолчанию
// значение attack объекта infantry равно 0,
// значение health объекта infantry равно 4

// 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 конструкторв 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 свойсва 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 конструтора
13     }
14     /* методы */
15 }
```

```java
public class BattleUnit {
   private int health;
   private int attack = 4;

   public BattleUnit(int health) {}
   public BattleUnit(int health, int attack) {
       setHealth(health);
       setAttack(attack);
   }
   public BattleUnit(int health) {
       this(health, 6);
   }
   public void setHealth(int health){
        if (health <= 3) throw new IllegalArgumentException("задайте health больше 3");
        this.health = health;
    }
   /* аналогично для свойсва attack */
   public void getHealth(){
        return health;
   }
   /* аналогично для свойсва attack */
   public void attackEnemy(BattleUnit enemy) {
         Objects.requireNonNull(enemy, "enemy не должен быть null");
         enemy.health -= attack;
   }
   public void attackEnemy(BattleUnit... enemies) {
        Objects.requireNonNull(enemies, "enemies не должен быть null");
        for (BattleUnit unit: enemies) attackEnemy(unit);
   }
   public boolean isAlive() {
        return health > 0;
   }
}
```