

ЗАНЯТИЕ 1.4

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ

Тема: Создание объектно-ориентированного приложения. Изучение наследования и инкапсуляции.

Для выполнения практического задания необходимо **обязательно** ознакомиться с лекцией 1.3.

Рассмотрим создание Java-приложения с применением объектно-ориентированного подхода для работы с сущностью Person (человек). Создайте новый проект в Apache Netbeans (File -> New Project). Назначьте ему имя Practical2.

Далее добавьте новый класс в созданное приложение (правой кнопкой мыши нажать на имя проекта, выбрать New -> Java Class):

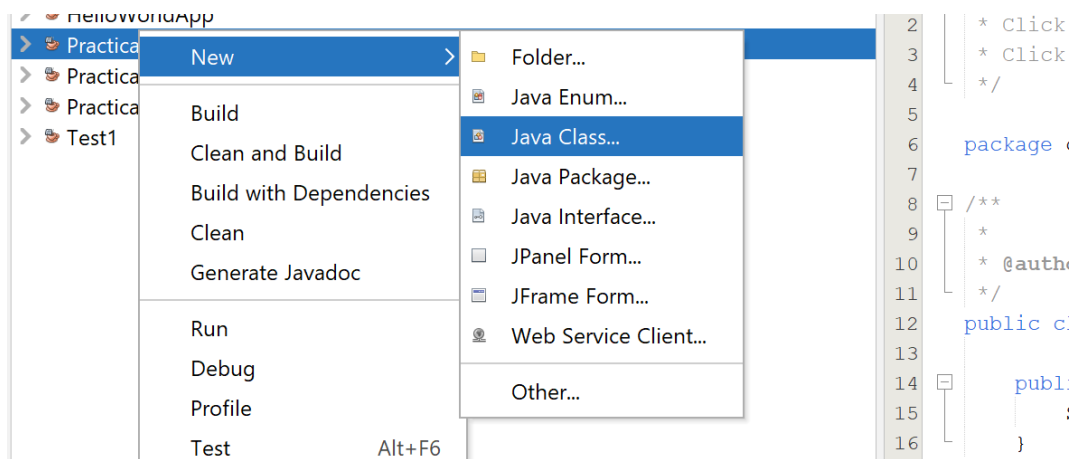


Рисунок 1

Укажите имя для этого класса – Person (названия классов должны начинаться с заглавной буквы). Подразумевается, что данный класс будет описывать сущность «Человек».

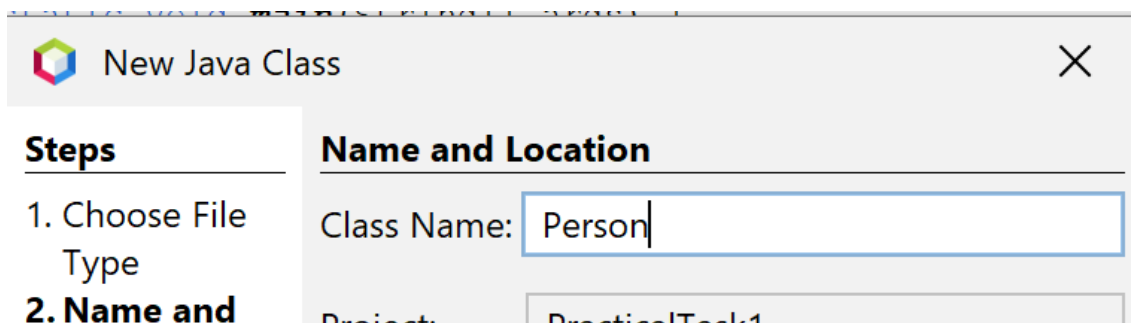


Рисунок 2

После этого в нашем проекте появится новый файл Person.java. Теперь в вашем проекте **два класса** – Practical2 и Person. Пока класс Person пустой:

```

*
* @author User
*/
public class Person {

}

```

Рисунок 3

Класс – это шаблон, в котором описаны общие черты и действия похожих друг на друга объектов. Общую структуру класса можно представить следующим образом:

```

модификатор_доступа class Имя_класса{
    // Переменные класса
    модификатор_доступа тип_переменной имя_переменной;
    // другие переменные класса

    //Конструкторы класса

    // Методы класса
    модификатор_доступа тип_возвращаемого значения имя_метода(аргументы){
        // Тело метода
    }
    // другие методы класса
}

```

Рисунок 4

Переменные класса описывают свойства класса, его основные характеристики. Какие основные свойства можно выделить у сущности «человек»? Наиболее подходящие: имя и возраст. Как правило, переменные класса имеют приватный модификатор доступа (private), чтобы к ним не было прямого доступа из других классов (соблюдение принципа ООП – инкапсуляции).

```

7  /**
8   *
9   * @author User
10  */
11  public class Person {
12      private String name;
13      private int age;
14
15  }
16

```

Рисунок 5

Чем отличаются переменные класса от локальных переменных метода? Локальные переменные «существуют» только в рамках этого метода, а также не имеют модификатор доступа. Например:

```
private void someMethod(){
    int x = 1;
    int y = 2;
    int z = (x + y) * 2;
    // Еще какой-то код
}
```

Локальные переменные
метода someMethod()

Рисунок 6

Методы класса описывают поведение, т.е. «что умеет делать данная сущность». Какие действия выполняет человек? Для примера, добавим два метода (подумайте, какие можно добавить еще). Чтобы методы можно было вызывать из других классов, используем для них модификаторы `public`.

```
public class Person {
    private String name;
    private int age;

    public void doSleep(int hours) {
        System.out.println("Person sleep " + hours + " hours");
    }

    public void doEat(String[] foods) {
        //здесь какой-то код
    }
}
```

Рисунок 7

Тип **void** означает, что метод ничего не возвращает. Названия методов и переменных должны удовлетворять нотации «lowerCamelCase» (см. лекцию 1.3).

Как вызвать метод?

Вызвать метод необходимо по его имени.

Если вы вызываете метод из этого же класса, то можно просто написать:

***someMethod();** // вызов метода с именем someMethod, у которого нет аргументов*

Если вы вызываете метод другого класса, то необходимо обратиться к нему через ссылочную переменную (ссылку) этого класса:

<имя_ссылочной_переменной>.someMethod();

В главном классе программы, в методе **main** создайте объект класса **Person** и вызовите для этого объекта метод **doSleep** (доступ к публичным элементам класса – методам или переменным осуществляется через точку). В качестве аргумента для примера передайте в метод значение 8. Запустите программу:

```
public class Practical2 {  
    public static void main(String[] args) {  
        System.out.println("Hello World!");  
        Person pers = new Person();  
        pers.doSleep( hours: 8);  
    }  
}
```



```
cd C:\Users\User\Documents\NetBeansProjects\PR1_4; "JAVA_HOME=  
Running NetBeans Compile On Save execution. Phase execution is  
Scanning for projects...  
  
-----< com.mirea.kt:Practical2 >-----  
Building Practical2 1.0  
-----[ jar ]-----  
--- exec-maven-plugin:3.0.0:exec (default-cli) @ Practical2 ---  
Hello World!  
Person sleep 8 hours  
  
BUILD SUCCESS
```

Рисунок 8

Примечание: не нужно писать слово «hours»! Это подписывает сама IDE, чтобы программисту было нагляднее, какие переменные используются в качестве аргументов метода.

Рассмотрим подробнее процесс создания объекта класса **Person** (создание экземпляра класса **Person**):

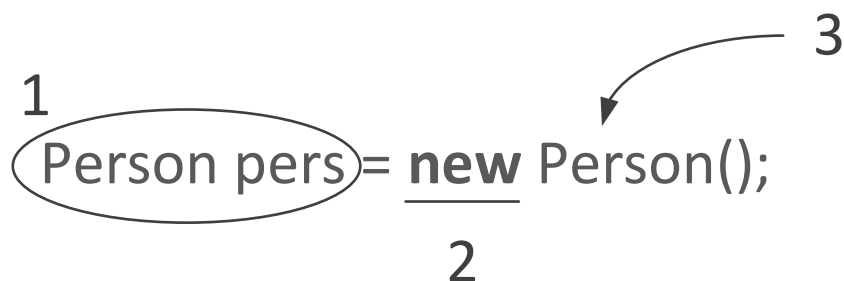


Рисунок 9

На **первом** этапе создается **ссылочная** переменная. Тип у этой ссылочной переменной – **Person**. Имя ссылочной переменной может быть любое (но должно отражать ее смысл и удовлетворять нотации «lowerCamelCase»). В данном случае имя – **pers**.

На **втором** этапе используется ключевое слово **new**.

На **третьем** этапе вызывается **конструктор** класса **Person**.

Экземпляр класса **Person** создан!

Откуда взялся конструктор класса **Person**? Можно заметить, что в классе **Person** мы его не создавали.

В случае, если программист не создал ни одного конструктора класса, компилятор неявно создает его (конструктор по умолчанию - без аргументов).

Вот так выглядит конструктор по умолчанию, неявно создаваемый компилятором:

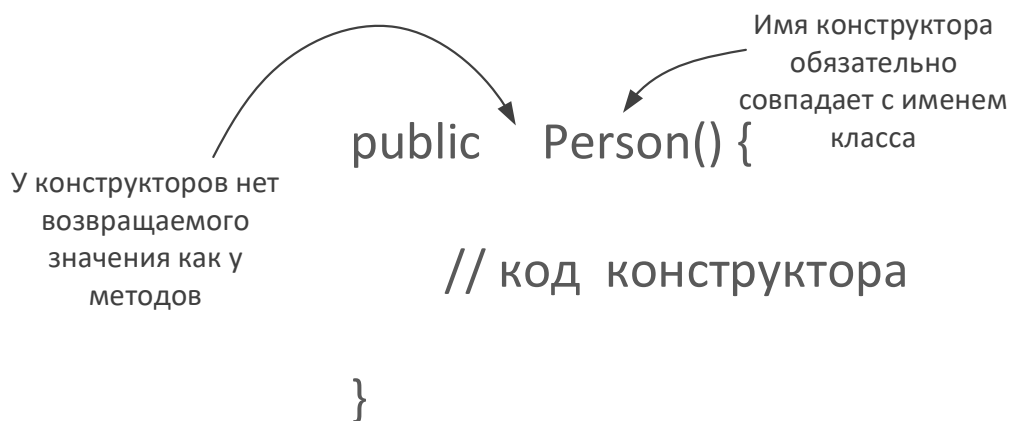


Рисунок 10

Как используется конструктор? Как правило, конструктор используется для инициализации состояния объекта (то есть для присваивания значений переменным класса). Ведь когда мы создали объект с помощью конструктора, представленного на рисунке 8, значения переменных остаются значениями по умолчанию (см. таблицу ниже) в зависимости от их типа.

Тип	Значение по умолчанию
boolean	false
byte	(byte) 0
short	(short) 0
int	0
long	0L
char	\u0000
float	0.0F
double	0.0D
object reference	null

Рисунок 11

Создайте свой конструктор для инициализации переменных класса Person. Это можно быстро выполнить с помощью средств IDE Netbeans. В классе Person, внутри класса перейти в меню, нажав правую кнопку мыши. В появившемся меню необходимо выбрать Insert Code.

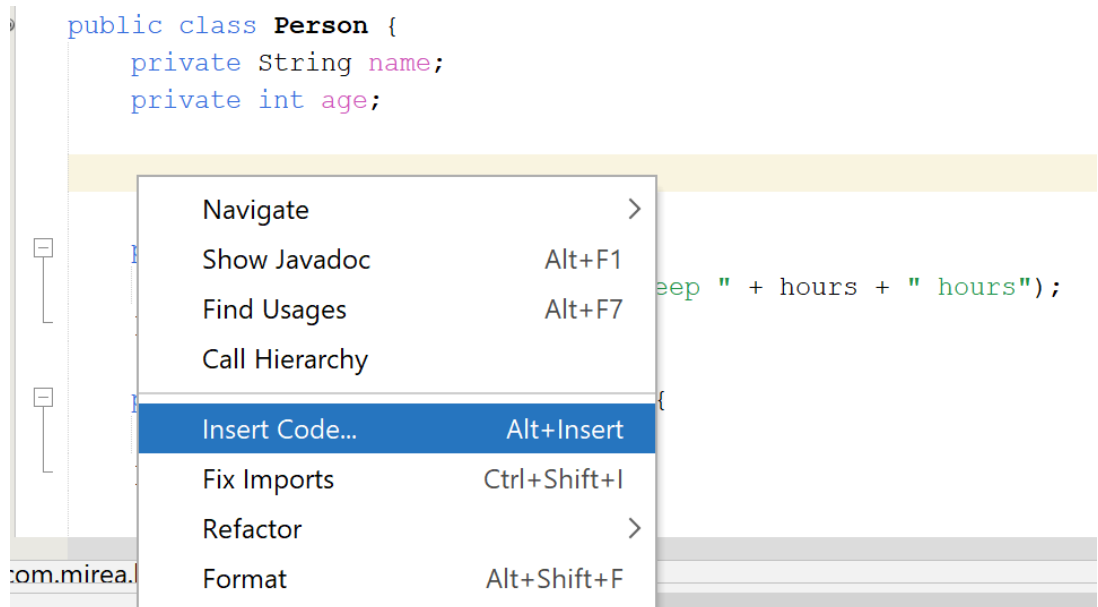


Рисунок 12

Далее необходимо выбрать Constructor:

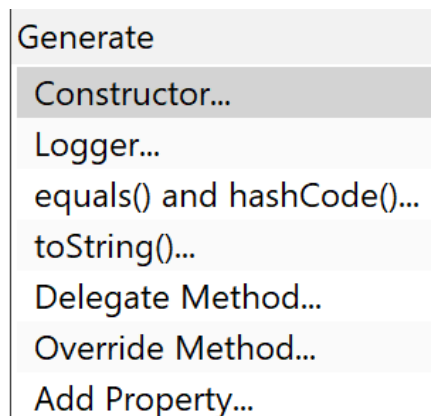


Рисунок 13

Далее предлагается выбрать названия переменных, которые будут инициализованы с помощью конструктора. Сначала создайте конструктор, который будет инициализировать обе переменных:

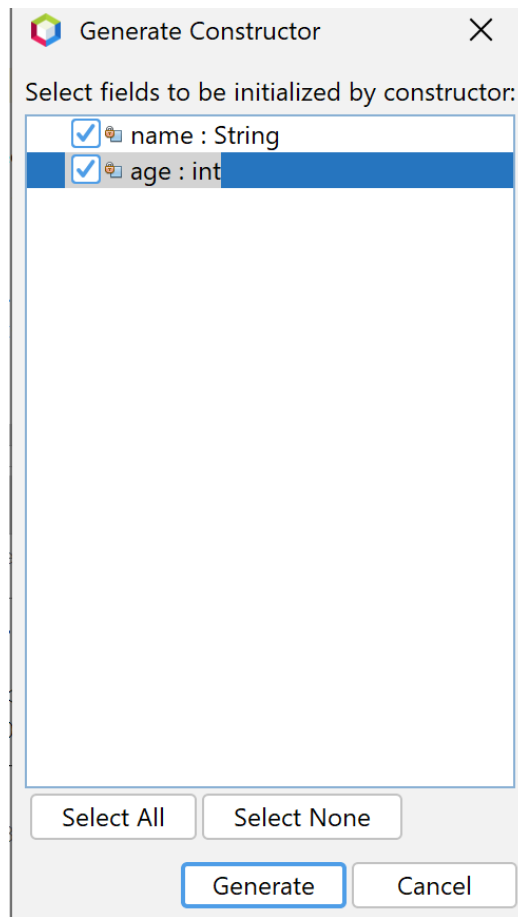


Рисунок 14

Таким же способ сгенерируйте код конструктора для инициализации только одной переменной **name**. В результате в классе **Person** будет **два** конструктора (это позволяет механизм **перегрузки**).

```

^/
public class Person {
    private String name;
    private int age;

    public Person(String name) {
        this.name = name;
    }

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

    public void doSleep(int hours) {
        System.out.println("Person sleep " + hours + " hours");
    }

    public void doEat(String[] foods) {
        //здесь какой-то код
    }
}

```

Рисунок 15

Ключевое слово **this** используется для обращения к элементам внутри класса (это позволяет нам использовать в списке аргументов конструктора такие же имена переменных).

После того как были описаны конструкторы, компилятор больше не создает конструктор по умолчанию!

Поэтому поменяем конструктор в нашем главном классе и передадим в конструктор значения для переменных **name** и **age**.

```

public class Practical2 {

    public static void main(String[] args) {
        System.out.println("Hello World!");
        String str = "Ivan Ivanov";
        Person pers = new Person(name:str, age:20);
        pers.doSleep(hours:8);
    }
}

```

Рисунок 16

Теперь у созданного экземпляра класса Person есть имя и возраст! Это можно проверить, запустив приложение в режиме отладки:

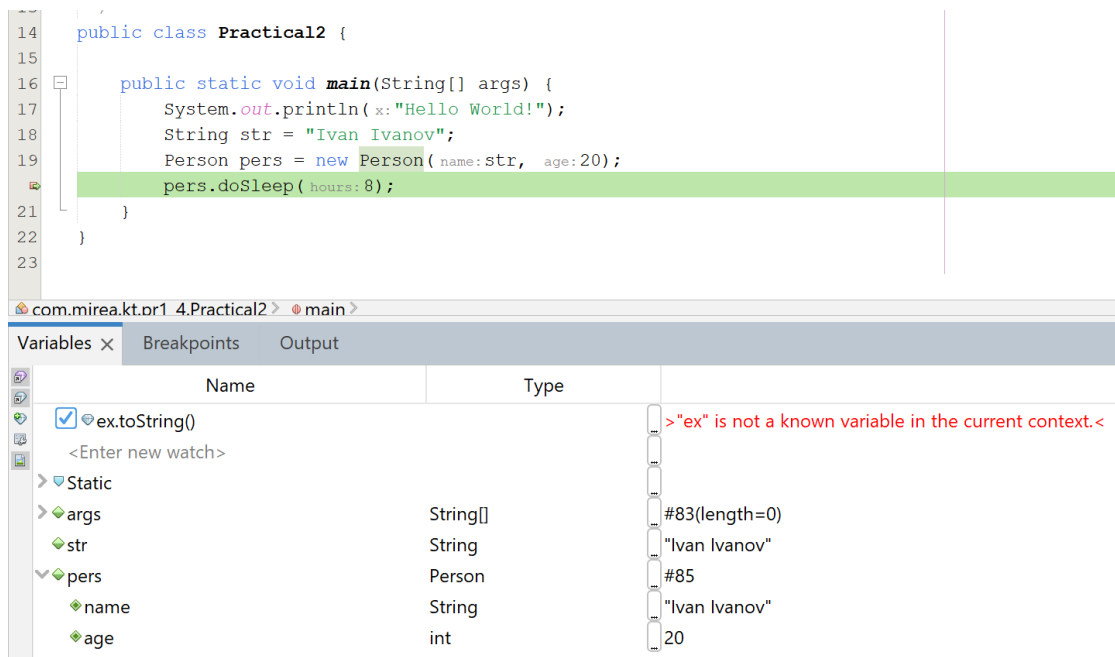


Рисунок 17

Переменная **age** имеет тип **int**, а это значит, что данной переменной можно присвоить любое целочисленное значение, в том числе и отрицательное. Но это противоречит смыслу класса **Person**. Значит данные переменной необходимо «защитить», скрыв к ней прямой доступ. Для этого в ООП существует принцип инкапсуляции.

Для реализации этого принципа необходимо создать в классе **Person** специальные методы – сеттеры (для инициализации значений переменных) и геттеры (для получения значения переменных). Эти методы публичные и имеют доступ к приватным переменным. Создать эти методы можно вручную или с помощью средств IDE (меню класса -> Insert Code -> Getters and Setters).

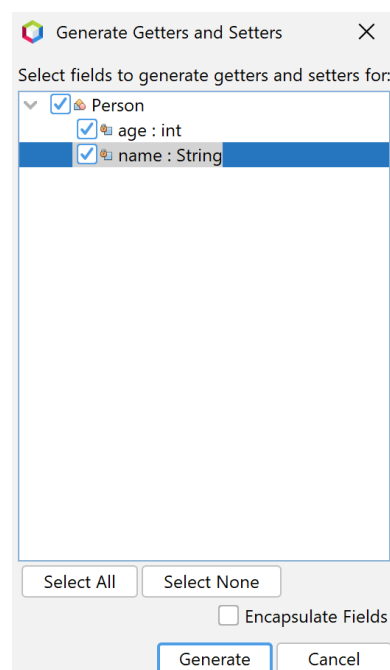
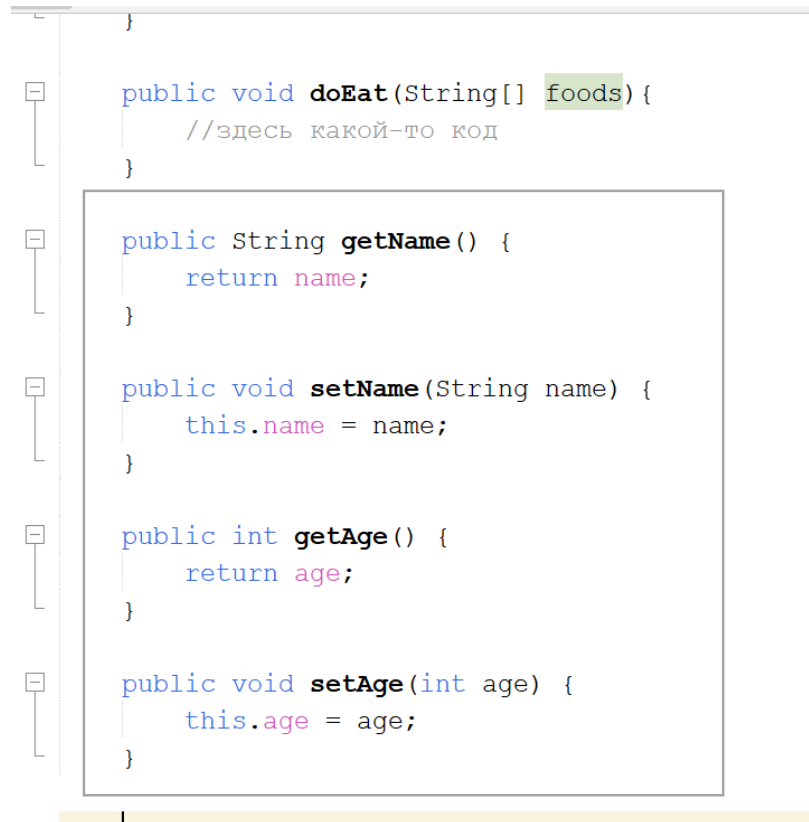


Рисунок 18

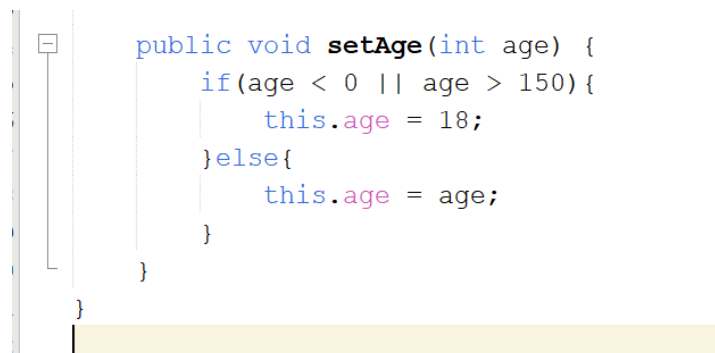
В результате в классе `Person` будут созданы методы:



```
    }  
  
    public void doEat(String[] foods) {  
        //здесь какой-то код  
    }  
  
    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;  
    }  
}
```

Рисунок 19

Как уже было описано ранее, сеттеры (как конструкторы) используются для инициализации значений переменных класса. Реализуем в методе `setAge` защиту от инициализации переменной `age` отрицательными значениями.



```
    public void setAge(int age) {  
        if(age < 0 || age > 150) {  
            this.age = 18;  
        }else{  
            this.age = age;  
        }  
    }  
}
```

Рисунок 20

Теперь, если кто-то извне попытается некорректно инициализировать переменную нашего класса, сеттер обработает такое значение. Например:

```

public class Practical2 {

    public static void main(String[] args) {
        System.out.println(x:"Hello World!");
        String str = "Ivan Ivanov";
        Person pers = new Person( name:str);
        pers.setAge( age:-1); // здесь мы попытались назначить нашей переменной класса отрицательное значение
        pers.doSleep( hours:8);
    }
}

```

Рисунок 21

Инкапсуляция работает!

Теперь рассмотрим еще один принцип ООП – наследование. Для этого создадим в этом же проекте еще два класса – **Student** (студент) и **Worker** (рабочий).

Очевидно, что оба этих класса наследуются от класса **Person**, а значит являются его потомками. Пометим это в созданных классах с помощью ключевого слова **extends**.

Для класса **Student**:

```

public class Student extends Person {

```

И для класса **Worker**:

```

public class Worker extends Person{

```

Для двух новых классов необходимо также создать отличительные переменные и методы. Для сущности «студент» для примера создадим три переменных (можно добавить больше), для сущности «рабочий» - две:

```

public class Student extends Person {

    private String group; // название группы
    private int number; // порядковый номер
    boolean isFullTime; // тип обучения – очно или заочно

```

Рисунок 22

```

public class Worker extends Person{

    private String spec; // специальность
    private String workPlace; // место работы

```

Рисунок 23

Так как `Student` и `Worker` являются потомками класса `Person`, они наследуют его поведение (то есть все его публичные методы), а значит уже «умеют» спать и есть. Необходимо определить специфичное поведение для `Student` и `Worker`, поэтому добавим для каждого класса, например, по одному методу.

```
public class Student extends Person {  
  
    private String group; // название группы  
    private int number; // порядковый номер  
    boolean isFullTime; // тип обучения - очно или заочно  
  
    public void goToUniversity(String univerName) {  
        System.out.println("Student go to " + univerName);  
    }  
}
```

Рисунок 24

```
public class Worker extends Person {  
  
    private String spec; // специальность  
    private String workPlace; // место работы  
  
    public String work(String[] todoList) {  
        // рабочий выполняет список дел  
        return "какой-то результат";  
    }  
}
```

Рисунок 25

Так же создадим конструкторы для каждого класса средствами IDE.


```
public class Student extends Person {  
  
    private String group; // название группы  
    private int number; // порядковый номер  
    boolean isFullTime; // тип обучения - очно или заочно  
  
    public Student(String group, int number, boolean isFullTime, String name) {  
        super(name);  
        this.group = group;  
        this.number = number;  
        this.isFullTime = isFullTime;  
    }  
  
    public void goToUniversity(String univerName) {  
        System.out.println("Student go to " + univerName);  
    }  
}
```

Рисунок 26

Следует обратить внимание на строку **super(name)** в конструкторе дочернего класса. Это означает вызов конструктора суперкласса (то есть родительского класса). Зачем он там?

Когда создается объект дочернего класса, сначала должен быть создан объект родительского класса (ведь потомок не может быть создан раньше родителя!). В данном случае вызывается конструктор класса `Person` с одним аргументом (можно вызвать другой – выбор за разработчиком), а уже потом инициализируются переменные класса `Student`.

Создадим объект класса `Student` в нашем главном классе, инициализируем все переменные (в том числе те, которые относятся к родительскому классу – ведь у «студента» тоже есть имя и возраст; наследование позволяет использовать родительские методы – `setAge` и `setName`). А затем вызовем метод **goToUniversity** класса `Student`. В качестве аргумента передадим строковое значение «MIREA».



```
public class Practical2 {  
    public static void main(String[] args) {  
        System.out.println("Hello World!");  
        String str = "Ivan Ivanov";  
        //Person pers = new Person(str);  
        //pers.setAge(-1);  
        //pers.doSleep(8);  
        Student firstStudent = new Student(group: "ПИО-00-21", number: 1, isFullTime: true, name: str);  
        firstStudent.setAge(19); // так как Student наследуется от Person  
                                // ссылочная переменная типа Student может выполнять методы родителя  
        firstStudent.goToUniversity(univerName: "MIREA");  
    }  
}
```

om.mirea.kt.pr1 4.Practical2 >

Debugger Console X Run (Practical2) X

```
cd C:\Users\User\Documents\NetBeansProjects\PR1_4; "JAVA_HOME=C:\Program Files\Java\jdk1.8.0_241" cmd /c  
Running NetBeans Compile On Save execution. Phase execution is skipped and output directories of dependency  
Scanning for projects...  
  
-----< com.mirea.kt:Practical2 >-----  
Building Practical2 1.0  
-----[ jar ]-----  
--- exec-maven-plugin:3.0.0:exec (default-cli) @ Practical2 ---  
Hello World!  
Student go to MIREA  
-----  
BUILD SUCCESS
```

Рисунок 27

В итоге мы создали объектно-ориентированное приложение, которое имеет два уровня в иерархии наследования.

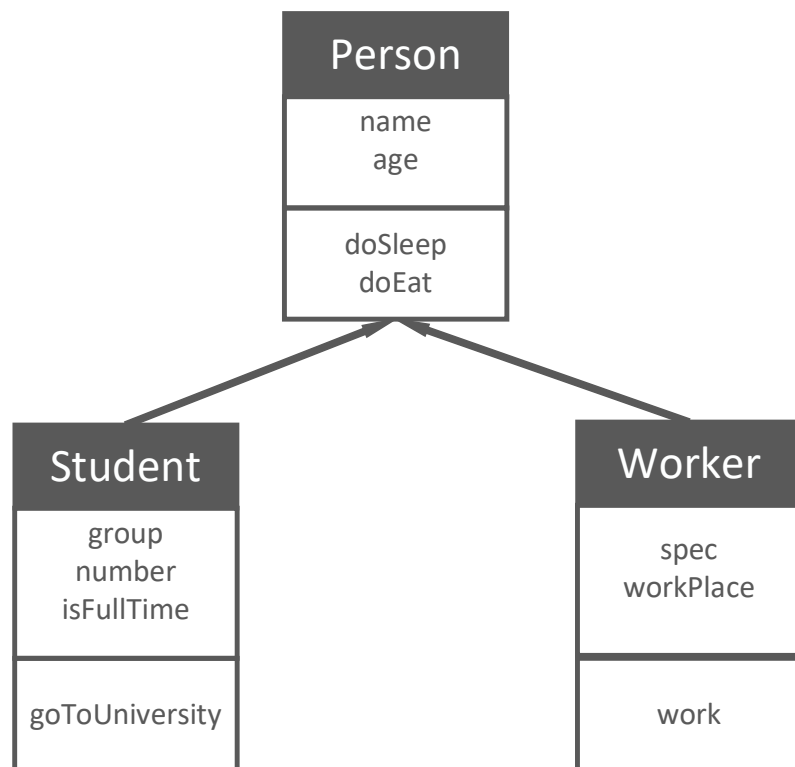


Рисунок 28