Java

Введение в классы

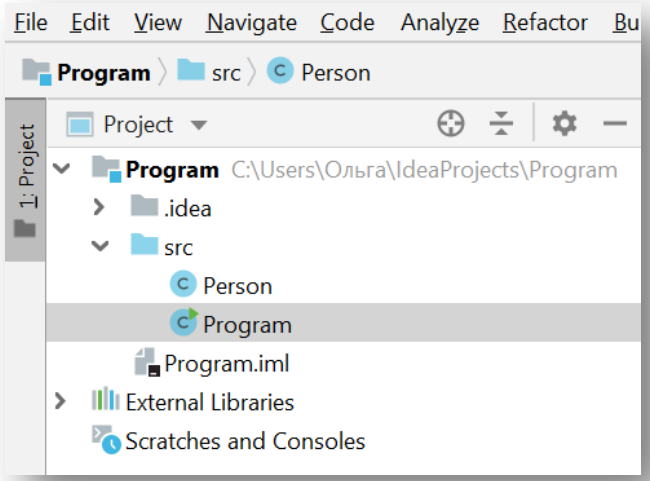
Лекция #4

Пустовалова О.Г.
доцент. каф. мат.мод.
ИММиКН ЮФУ

Содержание

ПРОЕКТ INTELLIJ IDEA

Пример проекта IntelliJ Idea



Program.java Person.java

```java
public class Program{

    public static void main(String[] args) {

        Person kate = new Person( name: "Kate",  age: 32);
        kate.displayInfo();
    }
}
```

Program.java Person.java

```java
class Person{

    String name;
    int age;

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

Пример проекта IntelliJ Idea

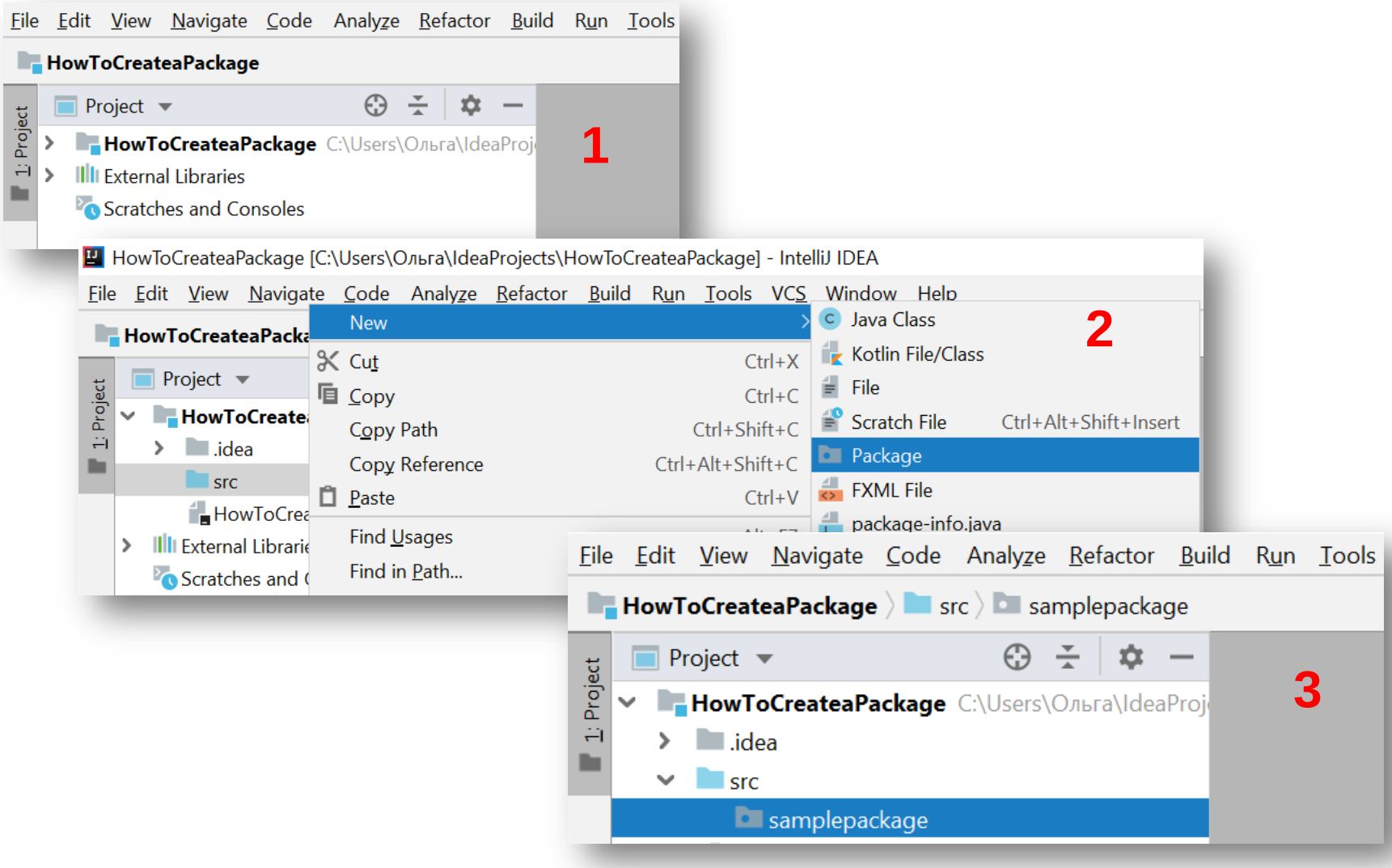
```
class Person{

    String name;
    int age;

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

Пример проекта IntelliJ Idea

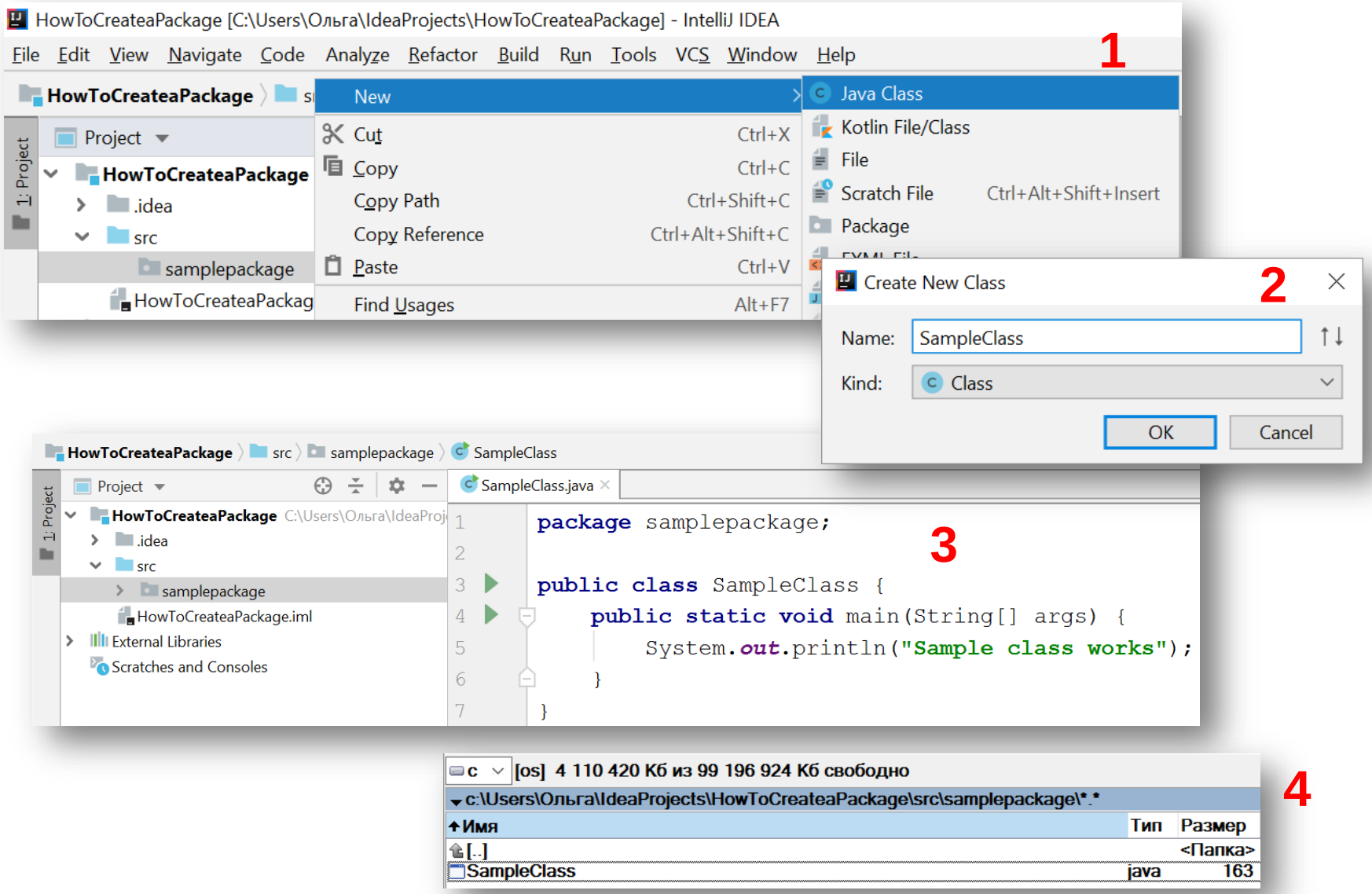
```
public class Program{

    public static void main(String[] args) {

        Person kate = new Person("Kate", 32);
        kate.displayInfo();
    }
}
```

КАК СОЗДАТЬ ПАКЕТ

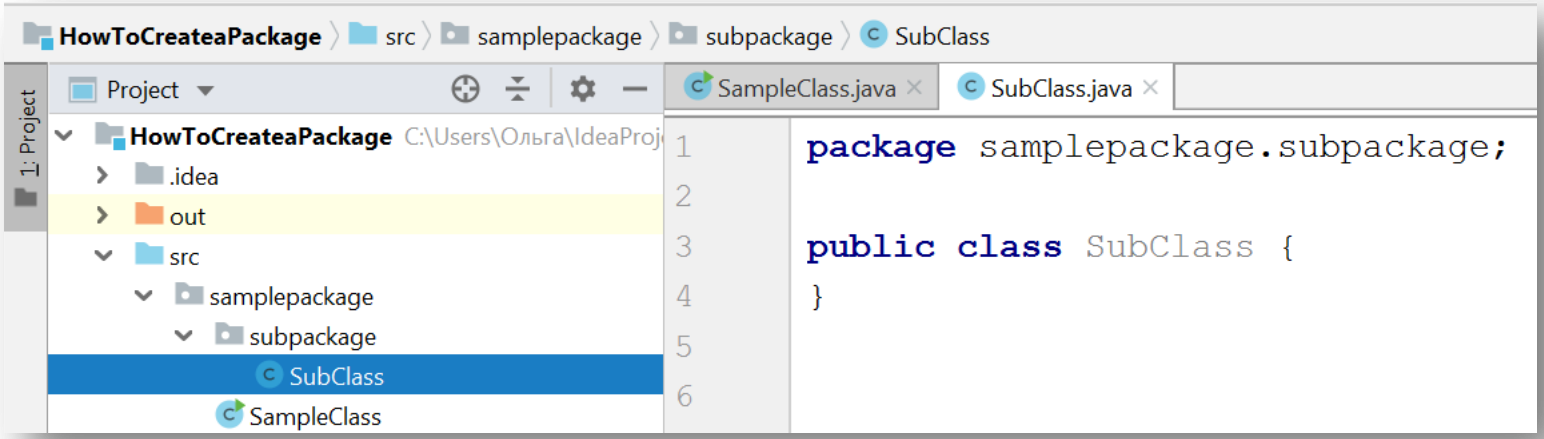
Как создать пакет

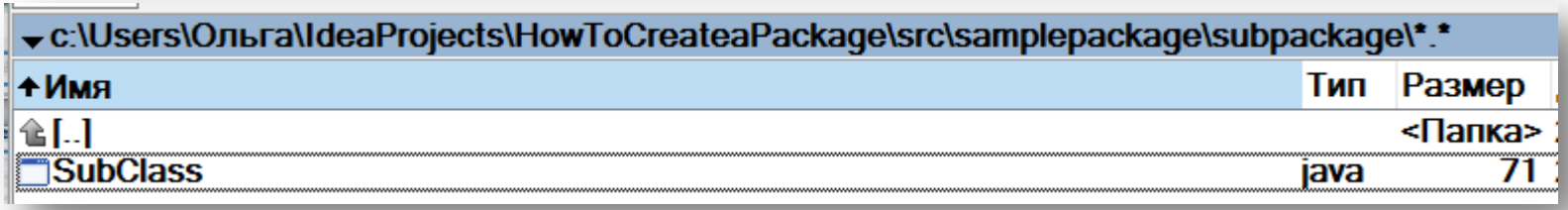


Как создать пакет. В пакете создаем класс



Как создать пакет. В пакете еще один под-пакет





При необходимости можно создать иерархию пакетов

Для чего создавать пакеты

- Для больших проектов в Java классы объединяются в пакеты.
- Пакеты позволяют организовать классы **логические наборы**.
- По умолчанию java уже имеет ряд встроенных пакетов, например, **java.lang**, **java.util**, **java.io** и т.д.
- Пакеты могут иметь вложенные пакеты.

Для чего создавать пакеты

- Организация классов в виде пакетов позволяет избежать **конфликта имен между классами**.
- Бывают ситуации, когда разработчики называют свои классы одинаковыми именами. Принадлежность к пакету позволяет **гарантировать однозначность имен**.
- Названия пакетов соответствуют физической структуре проекта, то есть организации каталогов, в которых находятся файлы с исходным кодом.

- Классы необязательно определять в пакеты. Если для класса пакет не определен, то считается, что данный класс находится в пакете по умолчанию, который не имеет имени.

Пакеты

Пакеты служат в основном для обеспечения **однозначности** имен классов.

Чтобы обеспечить абсолютную однозначность имени пакета, рекомендуется использовать доменное имя компании в Интернете, записанное в обратном порядке.

ru.sfedu.mmcs

Пакеты

- ✓ С точки зрения компилятора между вложенными пакетами отсутствует какая-либо связь.
- ✓ Например, пакеты **java.util** и **java.util.jar** вообще не связаны друг с другом.
- ✓ Каждый из них представляет собой независимую коллекцию классов.

Пакеты. Импорт классов

В классе могут использоваться все классы из собственного пакета и все *открытые* классы из других пакетов.

Доступ к классам из других пакетов можно получить двумя способами.

1. Перед именем каждого класса можно указать полное имя пакета:

```
java.time.LocalDate today = java.time.LocalDate.now() ;
```

2. Импортировать

```
// все классы из пакета time
import java.time.*;

// импортирование отдельного класса
import java.time.LocalDate;
```

Пакеты. Импорт классов

```
public class SampleClass {

    public static void main(String[] args) {

        java.time.LocalDate today = java.time.LocalDate.now() ;

        System.out.println(" today = "+today);
    }
}
```

today = 2019-04-24

```
package samplepackage;

// все классы из пакета time
import java.time.*;


public class SampleClass {
    public static void main(String[] args) {
        LocalDate today = LocalDate.now() ;

        System.out.println(" today = "+today);
    }}
```

ПЕРЕДАЧА ПАРАМЕТРОВ

Инициализация полей по умолчанию

Если значение поля в конструкторе явно не задано, то ему автоматически присваивается значение по умолчанию:

- ✓ числам — нули;
- ✓ логическим переменным — логическое значение false;
- ✓ ссылкам на объект — пустое значение null.

Если поля инициализируются неявно, программа становится менее понятной.

Конструктор без аргументов

```
public Employee()
{
   name ="";
   salary = 0;
}
```

Если в классе совсем не определены конструкторы, то автоматически создается конструктор без аргументов.

В этом конструкторе всем полям экземпляра присваиваются их значения, предусмотренные по умолчанию.

Передача параметров в метод. Общий подход

Во многих языках программирования (в частности, C++ и Pascal) предусмотрены два способа передачи параметров:

- ✓ по значению
- ✓ по ссылке.

Передача параметров в метод по значению

Для того, чтобы можно было изменить переданное значение в качестве параметра нужно использовать объект.

Передача объекта в метод. Пример. Часть 1. Employee

```
class Employee {
   // к данным полям имеют доступ только методы самого класса
   private String name;
   private double salary;
   // конструктор
   public Employee(String n, double s) {
      name = n;
      salary = s;     }
   // метод
   public String getNameO() {
      return name;
   }
   public double getSalary() {
      return salary;
   }
   public void raiseSalary(double byPercent) {
      double raise = salary * byPercent / 100;
      salary += raise;
   }
    public static void tripleSalary(Employee x) {
        x.raiseSalary(200);
    }
}
```

Передача объекта в метод. Пример. Часть 2. Employee

```
public class EmployeeTest {

    public static void main(String[] args) {
// заполнить массив staff тремя объектами типа Employee
        Employee[] staff = new Employee[3];

        staff[0] = new Employee("Carl ", 10000);
        staff[1] = new Employee("Harry ", 500);
        staff[2] = new Employee("Tony ", 400);

// поднять всем работникам зарплату на 5%
        for (Employee e : staff)
            e.raiseSalary(5);

        staff[0].tripleSalary(staff[0]);
// вывести данные обо всех объектах типа Employee
        for (Employee e : staff)
                System.out.println(e.getNameO() + " - " + e.getSalary());
    }
}
```

```
Carl    - 31500.0
Harry   - 525.0
Tony    - 420.0
```

НАСЛЕДОВАНИЕ

Наследование

- Классы, суперклассы и подклассы
- Глобальный суперкласс Object
- Обобщенные списочные массивы
- Объектные оболочки и автоупаковка
- Методы с переменным числом параметров
- Классы перечислений
- Рефлексия
- Рекомендации по применению наследования

Наследование. Классы, суперклассы и подклассы

```
class Manager extends Employee
{
   // Дополнительные методы и поля
}
```

Для обозначения наследования в Java служит ключевое слово

extends

Любое наследование в Java является открытым, т.е. в этом языке нет аналога закрытому и защищенному наследованию, допускаемому в C++.

Наследование. Классы, суперклассы и подклассы

Ключевое слово **extends** означает, что на основе существующего класса создается новый класс.

Существующий класс называется ***суперклассом***, *базовым* или *родительским*, а вновь создаваемый — ***подклассом***, *производным* или *порожденным*.

В среде программирующих на Java наиболее широко распространены термины ***суперкласс*** и ***подкласс***, хотя некоторые из них предпочитают пользоваться терминами ***родительский*** и ***порожденный***, более тесно связанными с понятием наследования.

Наследование. Переопределение методов подкласса

```
public double getSalary()



{
      double baseSalary = super.getSalary();

   return baseSalary + bonus;
}
```

Вызов метода суперкласса должен осуществляться с помощью ключевого слова super

Пояснение. Если в переопределяемом методе используется метод суперкласса, то он должен вызываться вместе с ключевым словом **super**.

В подкласс можно *вводить* поля, а также *вводить* и *переопределять* методы из суперкласса.

Наследование. Конструкторы подклассов

```
public Manager(String n, double s)
{
   super(n, s);

    bonus = 0;
}
```

Вызов конструктора суперкласса **Employee** с параметрами **n**, **s**.

Конструктор класса **Manager** не имеет доступа к закрытым полям класса **Employee,** поэтому он должен инициализировать их, вызывая другой конструктор с помощью ключевого слова **super.**

Вызов, содержащий обращение **super,** должен быть первым оператором в конструкторе подкласса.

Наследование. Пример Employee. Часть 1

```
class Employee {
    // к данным полям имеют доступ только методы самого класса
    private String name;
    private double salary;
    // конструктор
    public Employee(String n, double s) {
        name = n;
        salary = s;
    }
    // метод
    public String getName() {
        return name;
    }
    public double getSalary() {
        return salary;
    }}
```

Наследование. Пример Employee. Часть 2

```
class Manager extends Employee
{
   private double bonus;
   public Manager(String n, double s)
   {
      super(n, s);
      bonus = 0;
   }
   public double setBonus(double b){
      bonus=b;
     return bonus;
         }
   public double getSalary(){
      double baseSalary = super.getSalary();
      return baseSalary + bonus;
   }}
```

Наследование. Пример Employee. Часть 3

```
public class EmployeeTest {
    public static void main(String[] args) {

        Employee[] staff = new Employee[3];
        Manager boss = new Manager("Carl", 10000);
        boss.setBonus(5000);

        staff[0] = boss;
        staff[1] = new Employee("Harry", 500);
        staff[2] = new Employee("Tony", 400);

        for (Employee e : staff)
            System.out.println(e.getName() + " - "
                    + e.getSalary());
    }
}
```

```
Carl - 15000.0
Harry - 500.0
Tony - 400.0
```

Задания

Для всех заданий создать класс Test, в котором проверить работу дочерних классов.

1. Создать базовый класс Animal с полями: private boolean vegetarian; private String eats; private int noOfLegs;. На его основе создать подкласс Cat с полями суперкласса и private String color.

2. Создать базовый класс Building. На его основе создать подклассы: House, School, Library, Supermarket.

3. Создать базовый класс Horse. На его основе создать подклассы: Zebra, Donkey, Pegasus.

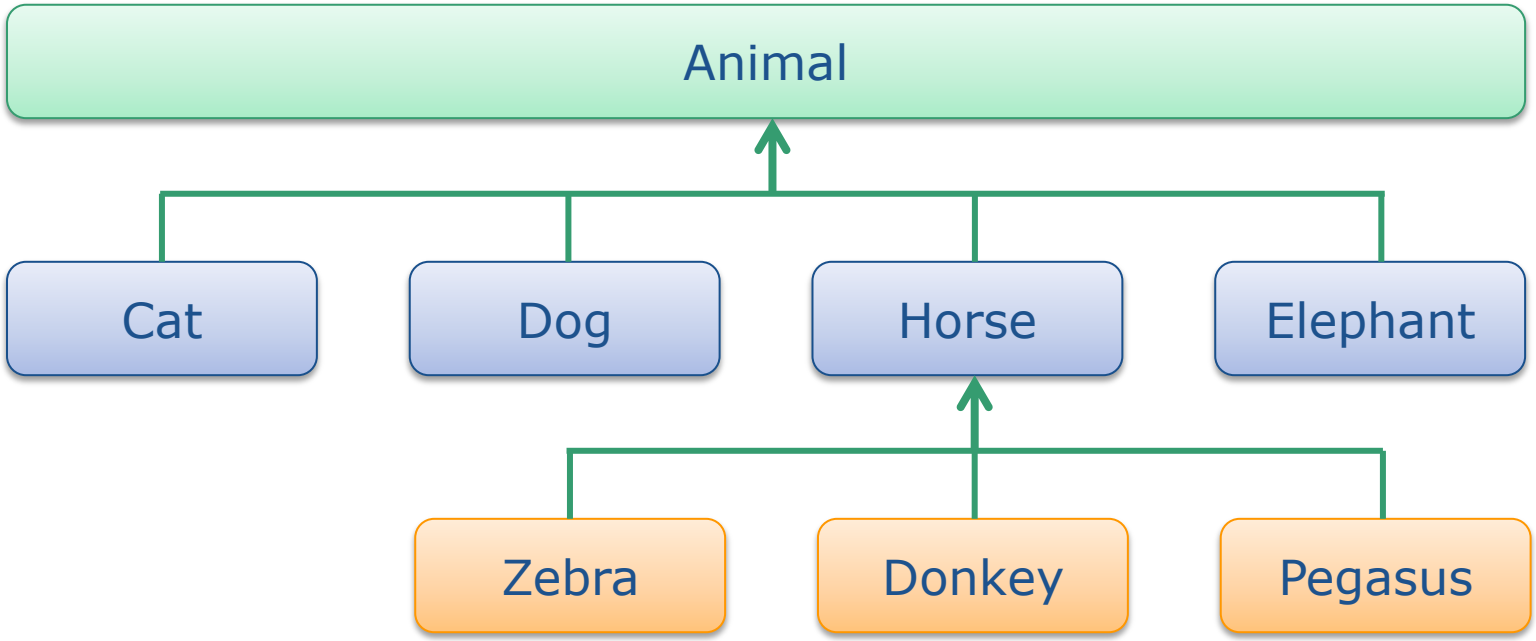
4. Создать базовый класс Food. На его основе создать подклассы: Starter, Soup, Salad, MainDish.

5. Создать базовый класс FootWear. На его основе создать подклассы: Shoes, Trainers, Boots, HighShoes, Sandals, Slippers.

ИЕРАРХИИ НАСЛЕДОВАНИЯ

Иерархии наследования

- ✓ Наследование не обязательно ограничивается одним уровнем классов.
- ✓ Совокупность всех классов, производных от общего суперкласса, называется иерархией наследования.
- ✓ Путь от конкретного класса к его потомкам в иерархии называется **цепочкой наследования**.



РЕКОМЕНДАЦИИ ПО РАЗРАБОТКЕ КЛАССОВ

Рекомендации по разработке классов

1. Всегда храните данные в переменных, объявленных как private.

- Первое и главное требование: всеми средствами избегайте нарушения инкапсуляции.
- Иногда приходится писать методы доступа к полю или модифицирующие методы, но предоставлять доступ к полям не следует

Рекомендации по разработке классов

2. Всегда инициализируйте данные.

- В языке Java локальные переменные не инициализируются, но поля в объектах инициализируются.
- Не полагайтесь на действия по умолчанию, инициализируйте переменные явным образом с помощью конструкторов.

Рекомендации по разработке классов

3. Не употребляйте в классе слишком много простых типов.

- Несколько связанных между собой полей простых типов следует объединять в новый класс. Такие классы проще для понимания, а кроме того, их легче видоизменить.

Четыре поля из класса Customer нужно объединить в новый класс Address:

```
private String street;
private String city;
private String state;
private int zip;
```

Рекомендации по разработке классов

4. Не для всех полей нужно создавать методы доступа и модификации.

- существуют поля, которые после создания объекта совсем не изменяются. К их числу относится, в частности, массив сокращенных названий штатов США в классе Address.

Рекомендации по разработке классов

5. Разбивайте на части слишком крупные классы.

- Если есть очевидная возможность разделить один сложный класс на два класса попроще, то воспользуйтесь ею.

Рекомендации по разработке классов

6. Выбирайте для классов и методов осмысленные имена, ясно указывающие на их назначение.

- Удобно принять следующие условные обозначения: имя класса должно быть именем существительным (**Order**) или именем существительным, которому предшествует имя прилагательное (**RushOrder**) или деепричастие (**BillingAddress**).

- Как правило, методы доступа должны начинаться словом **get**, представленным строчными буквами (**getSalary**), а модифицирующие методы — словом **set**, также представленным строчными буквами (**setSalary**).

Рекомендации по разработке классов

7. Отдавайте предпочтение неизменяемым классам.

- Трудность модификации состоит в том, что она может происходить параллельно, когда в нескольких потоках исполнения предпринимается одновременная попытка обновить объект. Результаты такого обновления непредсказуемы.

- Если же классы являются неизменяемыми, то их объекты можно благополучно разделять среди нескольких потоков исполнения.

Спасибо за внимание!