Java

НАСЛЕДОВАНИЕ

Лекция #11

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

Содержание

НАСЛЕДОВАНИЕ

Наследование

Принцип наследования состоит в том, что новые классы можно создавать из уже существующих классов.

При наследовании методы и поля существующего класса используются повторно (наследуются) вновь создаваемым классом, причем для адаптации нового класса к новым условиям в него добавляют дополнительные поля и методы.

Руководитель является работником

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());
    }
}
```

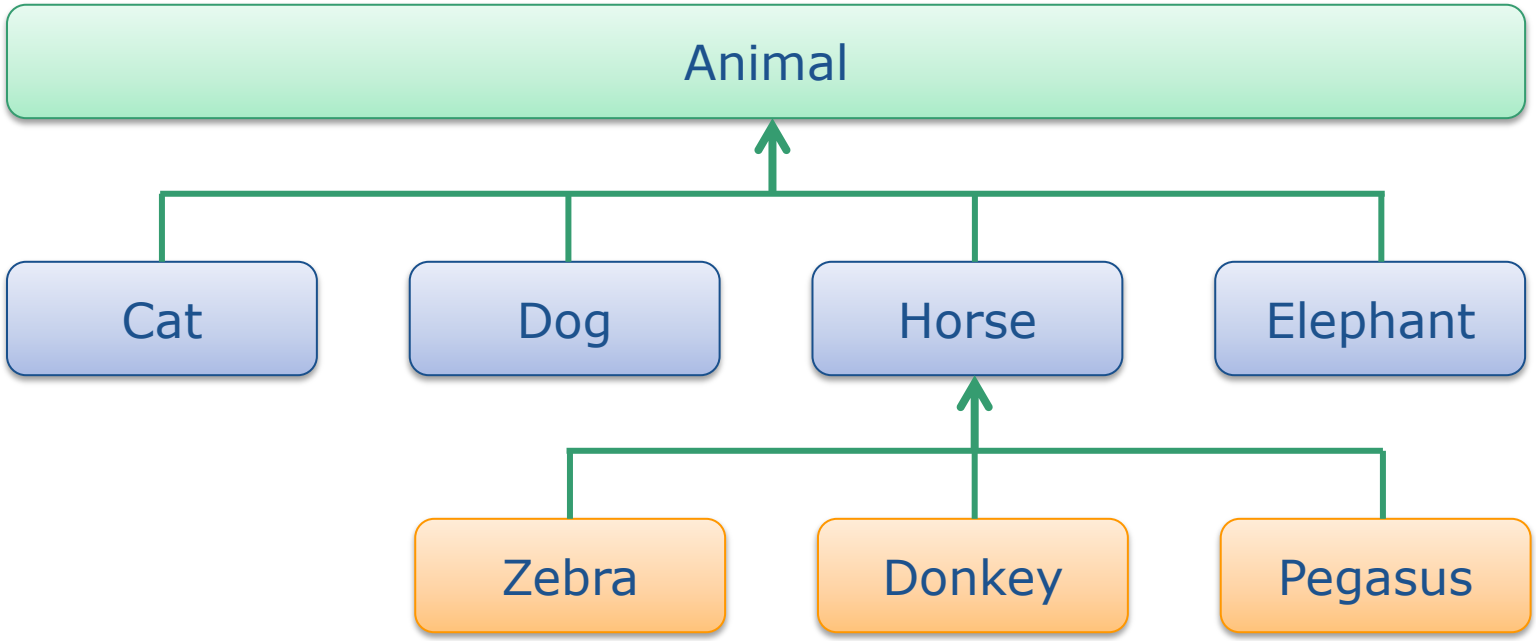
```
Carl - 15000.0
Harry - 500.0
Tony - 400.0
```

вызывается именно тот метод **getSalary**(), который необходим в данном случае

ИЕРАРХИИ НАСЛЕДОВАНИЯ

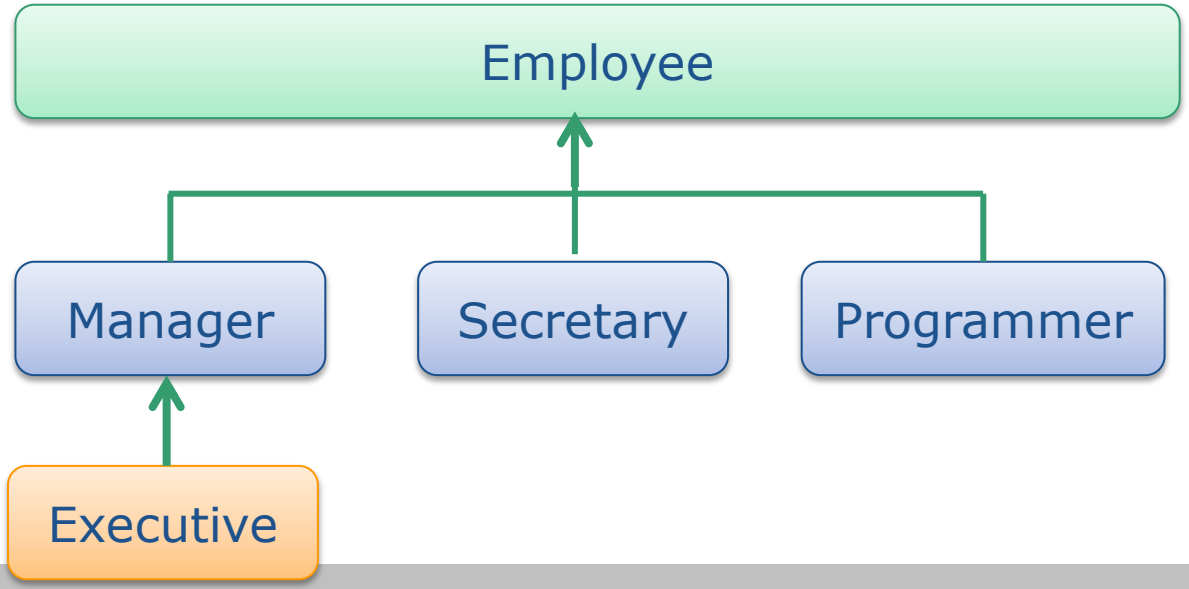
Иерархии наследования

- ✓ 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**.



Иерархии наследования

Обычно для класса существует несколько цепочек наследования. На основе класса **Employee** для работников можно, например, создать подкласс **Programmer** для программистов или класс **Secretary** для секретарей, причем они будут совершенно независимыми как от класса **Manager** для руководителей, так и друг от друга.



ПОЛИМОРФИЗМ

Полиморфизм

Существует простое правило, позволяющее определить, стоит ли в конкретной ситуации применять наследование или нет.

Если между объектами существует отношение "**является**", то каждый объект подкласса *является* объектом суперкласса.

Например, каждый руководитель является работником.

```
Manager boss = new manager(...);
Employee[] staff = new Employee[3];
staff[0] = boss;

boss.setBonus(5000); // Допустимо!
staff[0].setBonus(5000); // ОШИБКА!
// не все работники являются руководителями
```

ПРЕДОТВРАЩЕНИЕ НАСЛЕДОВАНИЯ

Предотвращение наследования

Иногда наследование оказывается нежелательным. Классы, которые нельзя расширить, называются *конечными.*

Для указания на это в определении класса используется модификатор доступа **final.**

Допустим, требуется предотвратить создание подклассов, производных от класса **Executive.**

В таком случае класс **Executive** определяется следующим образом:

```
final class Executive extends Manager{
    ...
}
```

Конечный метод класса final

Отдельный метод класса также может быть конечным.

Такой метод не может быть переопределен в подклассах.

```
class Employee
{
    ...
    public final String getName()
    {
        return name;
    }
        ...
}
```

С помощью модификатора доступа **final** могут быть также описаны поля, являющиеся константами. После создания объекта значение такого поля нельзя изменить.

Но если **класс** объявлен как **final**, то конечными автоматически становятся только его **методы**, но не поля.

Когда нужно создавать конечный класс

Существует единственный аргумент в пользу указания ключевого слова **final** при объявлении метода или класса: гарантия неизменности семантики в подклассе.

Например, класс **String** – конечный. Создавать подклассы, производные от этого класса, запрещено.
Следовательно, если имеется переменная типа **String,** то можно не сомневаться, что она ссылается именно на символьную строку, а не на что-нибудь другое.

Приведение типов

Случается, что ссылку на объект требуется привести к типу другого класса.

Для такого приведения типов служат те же самые синтаксические конструкции, что и для числовых выражений.

```
Manager boss = (Manager) staff[0];
```

перед приведением типов следует непременно проверить его корректность:

```
if (staff[1] instanceof Manager)
        {
        boss = (Manager) staff[1];
        ...
        }
```

Оператор **instanceof** нужен, чтобы проверить, был ли объект, на который ссылается переменная X, создан на основе какого-либо класса Y.

Правила приведения типов

• Приведение типов можно выполнять только в иерархии наследования.

• Чтобы проверить корректность приведения суперкласса к подклассу, следует выполнить операцию **instanceof**.

Приведение типов целесообразно лишь в том случае, когда для объектов, представляющих руководителей, требуется вызвать особый метод, имеющийся только в классе **Manager,** например метод **setBonus** ().

Правила наследования. 1. Наследуем только один класс

Правило 1. Наследуем только один класс. Java не поддерживает наследование нескольких классов. Один класс - один родитель.

Обратите внимание - нельзя наследовать самого себя!

Правила наследования. 2. Наследуется все кроме private

Правило 2. Наследуется все кроме приватных переменных и методов.

На самом деле, все методы и переменные, помеченные модификатором **private**, не доступны классу-наследнику.

Правила наследования. 3. @Override

Правило 3. Переделать метод класса-родителя.

Представим, что мы наследуем класс, но нам нравится не все, что мы унаследовали. Допустим, мы хотим, чтобы определенный - метод работал не так, как в родителе.

Для того, чтобы переопределить метод класса-родителя, пишем над ним **@Override**

Правила наследования. 4. super – методы родителя

Правило 4. Вызываем методы родителя через ключевое слово super.

Правила наследования. 5. final – запрещение наследования

Правило 5. Запрещаем наследование. Если Вы не хотите, чтобы кто-то наследовал Ваш класс, поставьте перед ним модификатор **final**.

МОДИФИКАТОРЫ ДОСТУПА

Модификаторы доступа

1. Модификатор доступа **private** — ограничивает область действия классом.

2. Модификатор доступа public — не ограничивает область действия.

3. Модификатор доступа **protected** — ограничивает область действия пакетом и всеми подклассами.

4. Модификатор доступа *отсутствует* — область действия ограничивается пакетом по умолчанию.

Спасибо за внимание!

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.