Plan

2023-10-02

1. Inheritance
2. Overriding methods in subclasses
3. type casting
4. Typecasting (20 minutes)
5. Practice problems in class (30 minutes)
    - Task 1: Create a superclass `Vehicle` and subclasses `Car` and `Bicycle`. Implement methods and fields. Show inheritance and type casting.
    - Task 2: Create an array of objects of different classes and demonstrate working with type casting.

---

1. Наследование
2. Переопределение методов в подклассах
3. приведение типов
4. Приведение типов (20 минут)
5. Практические задачи в классе (30 минут)

- Задача 1: Создать суперкласс `Vehicle` и подклассы `Car` и `Bicycle`. Реализовать методы и поля. Показать наследование и приведение типов.
- Задача 2: Создать массив объектов разных классов и продемонстрировать работу с приведением типов.

Theory

▶ **English**

▼ **На русском**

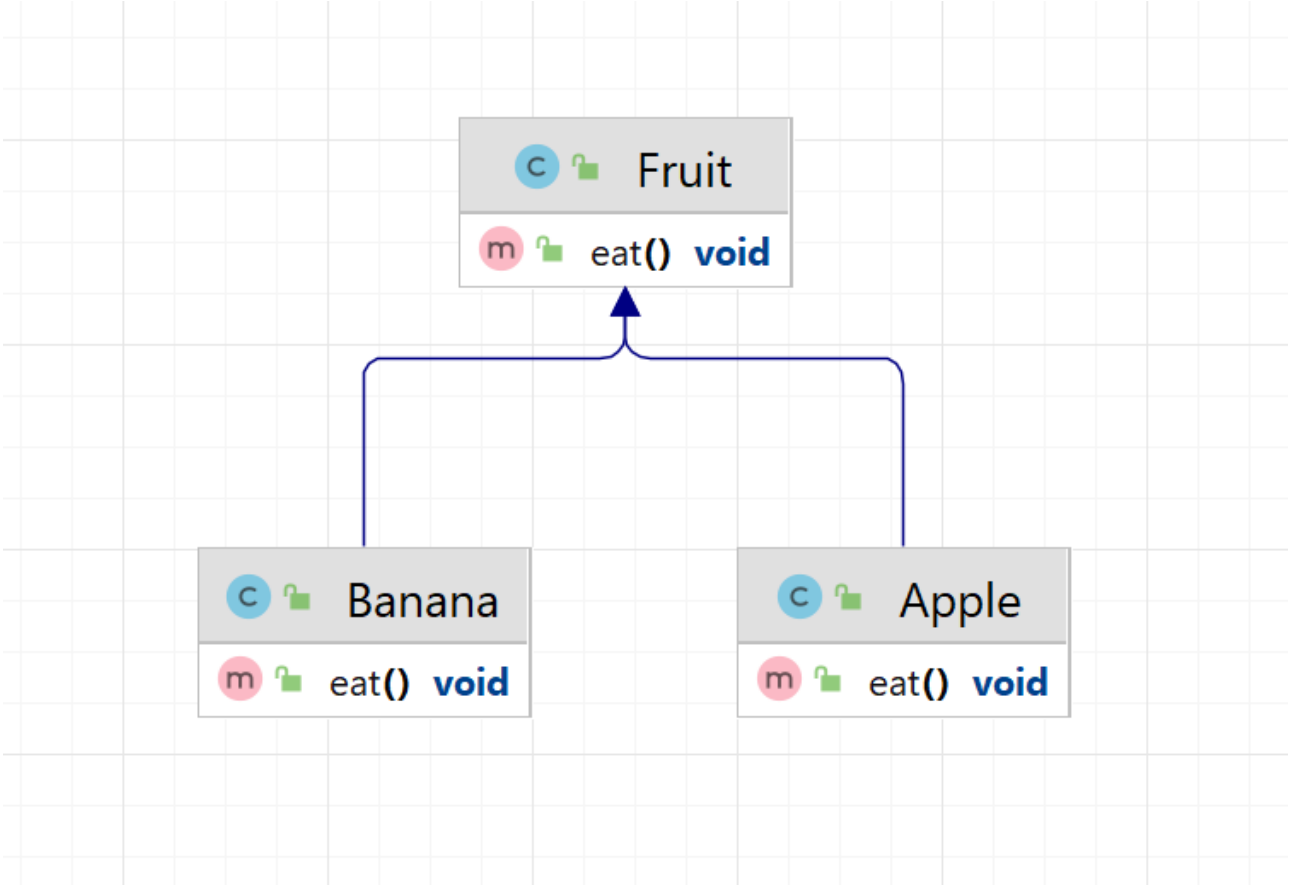
Наследование:

Наследование - это как наследование в реальной жизни, например, когда ребенок наследует некоторые черты от своих родителей. В программировании, это означает, что один класс может наследовать свойства и методы другого класса. Например, у нас есть класс "Фрукт", и мы создаем подкласс "Яблоко". Яблоко автоматически наследует свойства фрукта, такие как цвет и вес.

Пример:

```
class Fruit {
    String color;
    double weight;
}

class Apple extends Fruit {
    // Мы автоматически наследуем цвет и вес из класса Fruit
    String variety;
}
```

```
class Banana extends Fruit {
    // Мы автоматически наследуем цвет и вес из класса Fruit
    String variety;
}
```



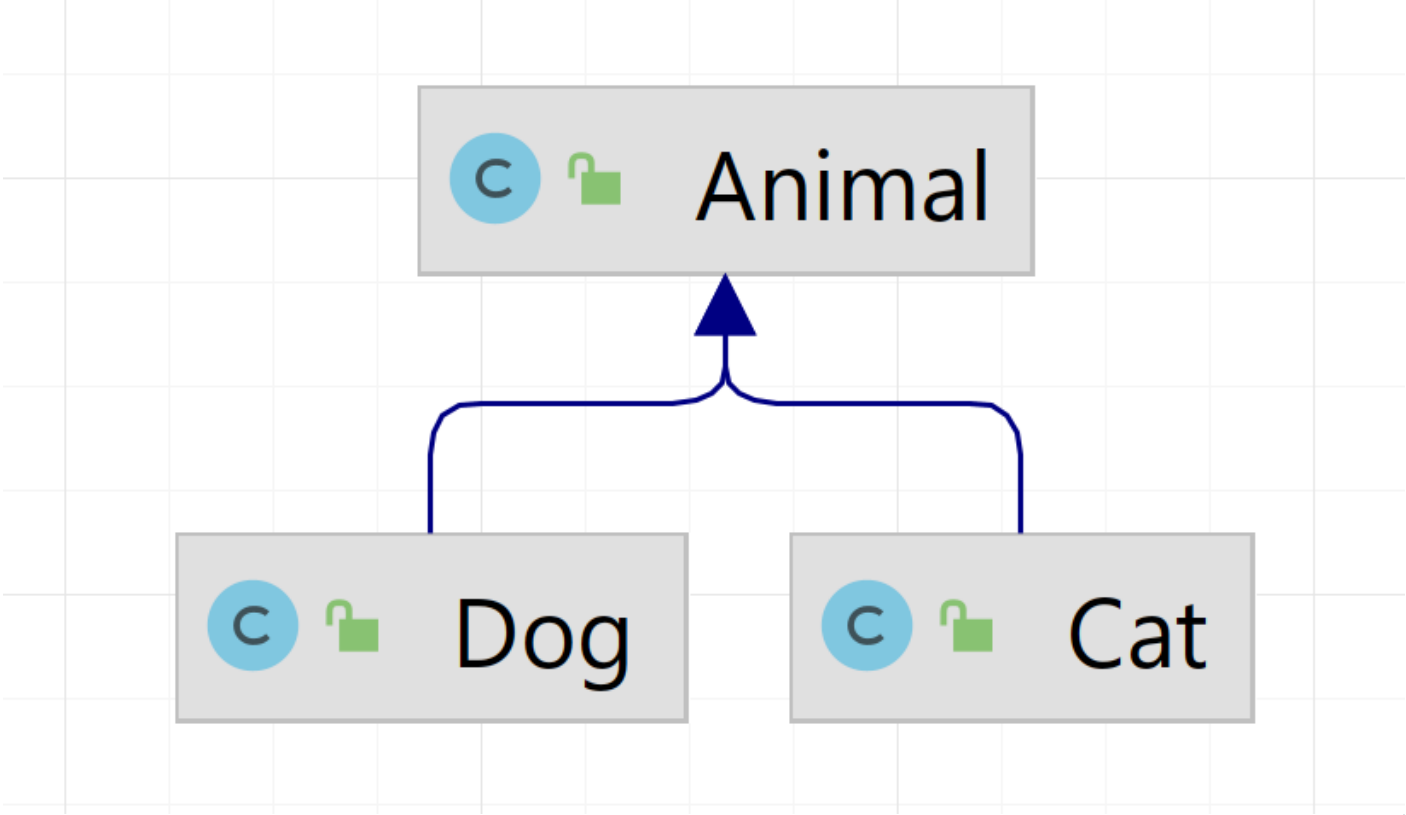
Переопределение методов в подклассах:

Переопределение методов - это как изменение правил игры. Если у нас есть метод в суперклассе, мы можем переопределить его в подклассе, чтобы сделать его специфичным для этого подкласса. Это позволяет нам использовать одинаковое имя метода, но с разными действиями для разных классов.

Представьте что у нас есть Животные, все животные могут воспроизводить звук. Но каждое животное делает это по разному. К примеру кошка делает "мяу-мяу", собачка "гав-гав".

Animal - родительский класс для всех животных. **Dog** - подкласс, представляющий собаку. **Cat** - подкласс, представляющий кошку. **makeSound()** - метод который выводит звук, издаваемый животным

Пример:




```java
public class Animal { // родительский класс для всех животных

    void makeSound() {
        System.out.println("Звук животного, не понятно какой"); // выводит звук, издаваем
    }
}

public class Dog extends Animal { // подкласс, представляющий собаку

}

public class Cat extends Animal { // подкласс, представляющий кошку

}
```

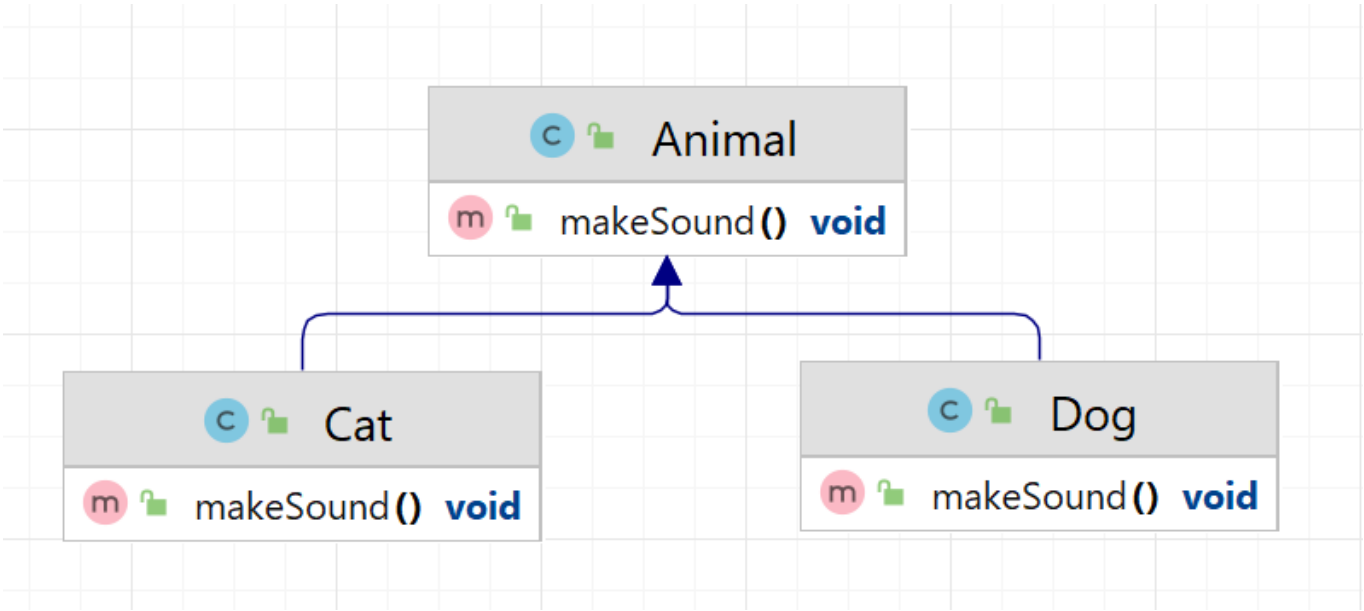
```java
public class Main {

    public static void main(String[] args) {
        Dog dog = new Dog();
        dog.makeSound(); //  Звук животного, не понятно какой

        Cat dog = new Cat();
        Cat.makeSound(); //  Звук животного, не понятно какой
    }
}
```

В этом примере собака и кошка будут уметь воспроизводить звук, но только так как это опрделено в родительском классе. Т.к. каждый раз когда мы возовим метод `makeSound()` не важно у какого типа, всегда

будет выведенно в конось "Звук животного, не понятно какой".

Но мы сказали что кошка должна воспроизводить "мяу-мяу", а собачка "гав-гав". Т.е. вопроизводить звук `makeSound()` но уже по своему.

```
public class Animal {

    public void makeSound() {
        System.out.println("Звук животного, не понятно какой");
    }
}

public class Dog extends Animal {

    @Override // переопределили метод супер класса
    public void makeSound() {
        System.out.println("Гав-гав!");  // указали поведение метода для класса Dog
    }
}

public class Cat extends Animal {

    @Override // переопределили метод супер класса
    public void makeSound() {
        System.out.println("Мяу-мяу!"); // указали поведение метода для класса Cat
    }
}
```



Этот код переопределяет в каждом классе наследнике родительский метод `makeSound()`. Что позволяет животным воспроизводить верные звуки.

```
public class Main {

    public static void main(String[] args) {
        Dog dog = new Dog();
```

```
        dog.makeSound(); //  Гав-гав!

        Cat dog = new Cat();
        Cat.makeSound(); //  Мяу-мяу!
    }
}
```

Приведение типов:

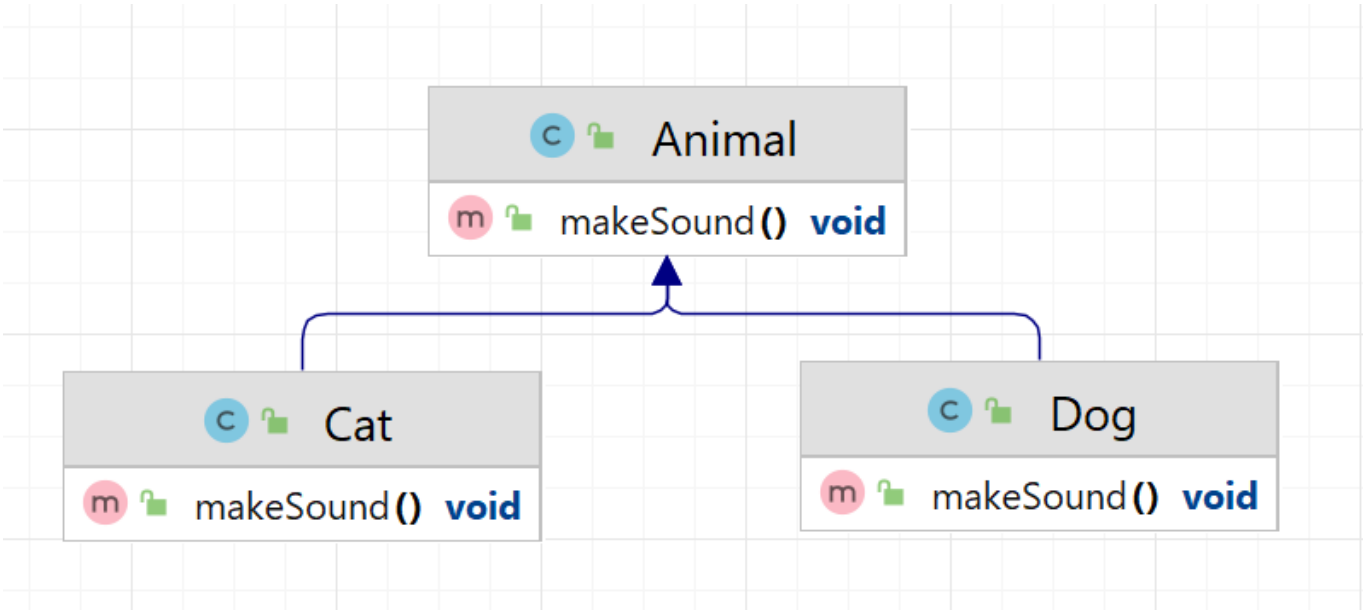
Приведение типов - это как превращение одной вещи в другую. В Java, это может быть нужно, когда у нас есть объект одного класса, но мы хотим его использовать как объект другого класса. Это может быть явным или неявным.

Давайте рассмотрим пример с собачками и кошками, чтобы лучше понять приведение типов.

У нас есть иерархия классов:

1. `Animal` - родительский класс для всех животных.
2. `Dog` - подкласс, представляющий собаку.
3. `Cat` - подкласс, представляющий кошку.

Каждый класс имеет метод `makeSound()`, который выводит звук, издаваемый животным.



Теперь давайте рассмотрим приведение типов:

```
Animal animal1=new Dog(); // Неявное приведение типа, собака становится животным
        Animal animal2=new Cat(); // Неявное приведение типа, кошка становится животным

        Dog dog=(Dog)animal1; // Явное приведение типа, животное становится собакой
        Cat cat=(Cat)animal2; // Явное приведение типа, животное становится кошкой
```

В этом примере:

1. `animal1` и `animal2` - это переменные типа `Animal`, но они содержат объекты `Dog` и `Cat`. Это неявное приведение типов, так как мы "сужаем" типы.

2. `dog` и `cat` - это переменные типа `Dog` и `Cat`, но мы явно указываем типы для приведения. Мы "расширяем" типы обратно к подклассам.

Теперь мы можем вызвать метод `makeSound()` для каждой переменной:

```
animal1.makeSound(); // Выведет: Гав-гав!
        animal2.makeSound(); // Выведет: Мяу-мяу!
        dog.makeSound();     // Выведет: Гав-гав!
        cat.makeSound();     // Выведет: Мяу-мяу!
```

Приведение типов позволяет нам работать с объектами разных классов в иерархии и использовать их специфические методы, когда это необходимо.

Восходящее и нисходящее приведение типов:

1. **Восходящее приведение (Upcasting)**:
   - Это приведение объекта к его суперклассу или интерфейсу.
   - Происходит автоматически (неявно).
   - Это безопасное приведение, так как объект всегда может быть рассмотрен как объект его суперкласса.

Пример восходящего приведения:

```
Cat cat=new Cat();
        Animal animal=cat; // Восходящее приведение, cat автоматически становится объектом
```

2. **Нисходящее приведение (Downcasting)**:
   - Это приведение объекта к его подклассу после восходящего приведения.
   - Происходит с использованием оператора приведения типа `(Type)` и может вызвать `ClassCastException`, если объект невозможно привести к указанному типу.
   - Это более рискованное приведение, и оно требует проверки типа перед выполнением приведения.

Пример нисходящего приведения:

```
Animal animal=new Cat(); // Восходящее приведение
        Cat cat=(Cat)animal;    // Нисходящее приведение, требует явного приведения типа
```

Таким образом, восходящее приведение обычно безопасно и выполняется автоматически, а нисходящее приведение может вызвать ошибку и требует явного приведения типа и проверки типа перед его выполнением.

Ошибки при приведении типов:

Ошибки при приведении типов и различия между восходящим и нисходящим приведением типов - это важные аспекты работы с приведением типов в Java. Давайте подробнее разберемся в этих концепциях.

1. **ClassCastException**:
   - Это исключение возникает при попытке выполнить недопустимое приведение типов.
   - Например, если у нас есть объект `Animal`, который на самом деле является объектом `Cat`, и мы пытаемся явно привести его к типу `Dog`, это вызовет `ClassCastException`.

Пример:

```
Animal animal=new Cat();
        Dog dog=(Dog)animal; // Вызовет ClassCastException, так как animal на самом деле к
```

Чтобы избежать таких ошибок, всегда следует выполнять проверку типов с использованием оператора `instanceof` перед приведением типа.

Пример проверки типа:

```
Animal animal=new Cat();
        if(animal instanceof Dog){
        Dog dog=(Dog)animal; // Этот код выполнится только, если animal является объектом
        }
```

Homework

▶ **English**

▼ **На русском**

Задача 1

Посмотрите вокруг себя. Напишите три класса, описывающие какие-то сущности вокруг Вас.

Например, опишите студента, стол, клавиатуру и т.д.

Вы можете выбрать свои сущности(объекты) для описания.

Задача 2

Иерархия классов "**Сотрудники компании**"

Создать:

- Базовый класс Сотрудник (**Employee**):
  - Приватные атрибуты: имя (name), зарплата (salary).
  - Конструктор для инициализации имени и зарплаты.
  - Геттеры и сеттеры для имени и зарплаты.
  - Метод **работать()** (`work()`), который выводит сообщение "***Я сотрудник, я работаю***".
- Класс Менеджер (**Manager**) наследуется от **Сотрудник**:
  - Дополнительный приватный атрибут: количество подчиненных (numSubordinates).
  - Конструктор для инициализации всех атрибутов.
  - Геттеры и сеттеры для нового атрибута.
  - Переопределенный метод **работать()**, который выводит сообщение "***Я менеджер, я управляю командой***".
- Класс Инженер (**Engineer**) наследуется от **Сотрудник**:
  - Дополнительный приватный атрибут: специализация (specialization).
  - Конструктор для инициализации всех атрибутов. Геттеры и сеттеры для нового атрибута.
  - Переопределенный метод работать(), который выводит сообщение "***Я разрабатываю***".

- Класс Интерн (**Intern**) наследуется от **Сотрудник**:
  - Дополнительный приватный атрибут: университет (university).
  - Конструктор для инициализации всех атрибутов. Геттеры и сеттеры для нового атрибута.
  - Переопределенный метод **работать()**, который выводит сообщение "***Я учусь и работаю***".

Задачи:

1. Создать несколько экземпляров каждого класса и вызвать метод **работать()** для каждого из них.
2. Создать массив или список Сотрудников и добавить в него объекты всех классов. Пройтись по этому массиву в цикле, вызывая метод работать() для каждого объекта.
3. Продемонстрировать принципы инкапсуляции, создав геттеры и сеттеры для всех атрибутов и делая атрибуты приватными.

Задача 3 (Опциональная)

Иерархия классов "**Музыкальные инструменты**"

- Базовый класс Музыкальный Инструмент (MusicalInstrument):
  - Приватные атрибуты: название (name), тип (type), цена (price).
  - Конструктор для инициализации атрибутов.
  - Геттеры и сеттеры для всех атрибутов.
  - Метод играть() (play), который выводит сообщение "Играет [название инструмента]".
- Класс Струнный Инструмент (StringInstrument) наследуется от Музыкальный Инструмент:
  - Дополнительный атрибут: количество струн (numberOfStrings).
  - Конструктор для инициализации всех атрибутов.
  - Геттер и сеттер для нового атрибута. Переопределенный метод играть(), который выводит " Играет [название инструмента] с [количество струн] струнами".
- Класс Духовой Инструмент (WindInstrument) наследуется от Музыкальный Инструмент:
  - Дополнительный атрибут: материал (material), например, "дерево" или "металл".
  - Конструктор для инициализации всех атрибутов.
  - Геттер и сеттер для нового атрибута.
  - Переопределенный метод играть(), который выводит "Играет [название инструмента] из [материал]".

Задачи:

1. Создать несколько экземпляров каждого класса и вызвать метод играть() для каждого из них.
2. Создать массив или список объектов типа Музыкальный Инструмент и добавьте в него объекты всех производных классов. Пройтись по этому массиву в цикле, вызывая метод играть() для каждого объекта.
3. Используйте геттеры и сеттеры для всех атрибутов классов для демонстрации принципов инкапсуляции.
4. добавьте метод стоимостьАренды() (rentalCost) в базовый класс, который будет рассчитывать стоимость аренды инструмента на основе его цены. Переопределите этот метод в производных классах, если необходимо.

Не забывайте:

- Переменные: camelCase, существительные (totalSum).
- Методы: camelCase, глаголы (calculateSum).
- Булевы: начинаются с is, has, can (isReady).

Code

code/Lesson_20/src/animal/Animal.java

```java
package animal;

/**
 * @author Andrej Reutow
 * created on 02.10.2023
 */
public class Animal {

    public void makeSound() {
        System.out.println("Животное издает звук, не понятно какой именно!");
    }
}
```

code/Lesson_20/src/animal/Cat.java

```java
package animal;

/**
 * @author Andrej Reutow
 * created on 02.10.2023
 */
public class Cat extends Animal {

    @Override
    public void makeSound() {
        System.out.println("Мяу-мяу!"); // указали поведение метода для класса Cat
    }

    /**
     * Метод, который позволяет кошке лазить по деревьям.
     */
    public void climbTree() {
        System.out.println("Кошка лазит по деревьям.");
    }

    /**
     * Метод, представляющий охотничьи навыки кошки.
     */
    public void hunt() {
        System.out.println("Кошка идет на охоту.");
    }
}
```

code/Lesson_20/src/animal/Dog.java

```java
package animal;

/**
 * @author Andrej Reutow
 * created on 02.10.2023
 */
public class Dog extends Animal {

    @Override // переопределили метод супер класса
    public void makeSound() { // переопределили метод супер класса
        System.out.println("Гав-гав!");  // указали поведение метода для класса Dog
    }

    /**
     * Метод, который позволяет собаке вернуть мяч после его броска
     */
    public void fetchBall() {
        System.out.println("Собака принесла мяч");
    }

    /**
     * Метод, который представляет действие охраны дома
     */
    public void guardHouse() {
        System.out.println("Собака охраняет дом");
    }
}
```

code/Lesson_20/src/animal/AnimalApl.java

```java
package animal;

/**
 * @author Andrej Reutow
 * created on 02.10.2023
 */
public class AnimalApl {

    public static void main(String[] args) {
        Animal animal = new Animal();
        Dog dog = new Dog();
        Cat cat = new Cat();


        animal.makeSound(); // Вывод: Животное издает звук, не понятно какой именно!
        // т.к. тип и фактический тип Animal берется реализация метода makeSound() из кла

        dog.makeSound(); // Вывод: Гав-гав!
        // т.к. тип и фактический тип Dog берется реализация метода makeSound() из классе
```

```
            cat.makeSound();// Вывод: Мяу-мяу!
            // т.к. тип и фактический тип Cat берется реализация метода makeSound() из классе
        }
    }
```

code/Lesson_20/src/animal/AnimalApl2.java

```
package animal;

/**
 * @author Andrej Reutow
 * created on 02.10.2023
 */
public class AnimalApl2 {

    public static void main(String[] args) {

        // создаем объекты разных типов:  Animal, Dog, Cat
        // обратите внимание, что animalDog, animalCat объявлены как тип Animal
        // animalDog,animalCat, на самом деле могут содержать объекты подклассов (Dog, Cat
        Animal animal = new Animal();
        Animal animalDog = new Dog();
        Animal animalCat = new Cat();


        animal.makeSound(); // Животное издает звук, не понятно какой именно!
        animalDog.makeSound(); // Гав-гав!
        animalCat.makeSound(); // Мяу-мяу!

        // Приведение типов (casting)
        // Нисходящее приведение (Downcasting):
        // animalDog объявлен как тип Animal
        // Мы выполняем нисходящее приведение объекта animalDog к типу Dog, чтобы вызвать
        // не безопастное нисходящее приведение типа, т.к. animalDog (в теории) моежен хра
        Dog dog = (Dog) animalDog; // Нисходящее приведение (Downcasting)

        dog.makeSound();
        dog.fetchBall();
        dog.guardHouse();

        // animalDog объявлен как тип Animal
        // Мы выполняем нисходящее приведение объекта animalDog к типу Dog, чтобы вызвать
        // безопастным нисходящим приведение типа, проверям является ли animalDog Cat. Ис
        if (animalDog instanceof Cat) {
            Cat cat = (Cat) animalDog;
            cat.makeSound();
            cat.climbTree();
            cat.hunt();
        } else {
            System.out.println("animalDog не может быть преобразован в Cat");
```

```
        }

        if (animalCat instanceof Cat) {
            Cat cat = (Cat) animalCat;
            cat.makeSound();
            cat.climbTree();
            cat.hunt();
        } else {
            System.out.println("animalDog не может быть преобразован в Cat");
        }


        //  Восходящее приведение (Upcasting)
        Cat catTom = new Cat();
        Dog dogBarbos = new Dog();

        Animal animalDogBarbos = dogBarbos;
        Animal animalCatTom = catTom;
        animalDogBarbos.makeSound();

        dogBarbos.fetchBall();


        System.out.println("########\n");
        Animal[] animals = new Animal[4];
        animals[0] = animalDog;
        animals[1] = animalCat;
        animals[2] = dog;
        animals[3] = animal;

        for (int i = 0; i < animals.length; i++) {
            Animal currentValue = animals[i];
            doSomth(currentValue);
        }
    }

    private static void doSomth(Animal animal) {
        animal.makeSound();
        if (animal instanceof Cat) {
            Cat cat = (Cat) animal;
            cat.hunt();
            cat.climbTree();
        } else if (animal instanceof Dog) {
            Dog dog = (Dog) animal;
            dog.fetchBall();
            dog.guardHouse();
        }
    }
}
```

code/Lesson_20/src/animal2/Animal.java

```java
package animal2;

/**
 * @author Andrej Reutow
 * created on 02.10.2023
 */
public class Animal {

    private String name;
    private String color;

    public Animal(String name, String color) {
        this.name = name;
        this.color = color;
    }

    public void makeSound() {
        System.out.println("Животное издает звук, не понятно какой именно!");
    }

    /**
     * Выводит данные животного
     *
     * @return Выводит данные животного
     */
    public String getDetails() {
        String result = "Name: " + this.name + " color: " + this.color;
        return result;
    }

    public String getName() {
        return this.name;
    }

    public String getColor() {
        return this.color;
    }
}
```

code/Lesson_20/src/animal2/Cat.java

```java
package animal2;

/**
 * @author Andrej Reutow
 * created on 02.10.2023
 */
public class Cat extends Animal {
```

```
    public Cat(String name, String color) {
        super(name, color);
    }

    @Override
    public void makeSound() {
        System.out.println("Мяу-мяу!"); // указали поведение метода для класса Cat
    }

    @Override
    public String getDetails() {
        // Я кошка, Name: Том color: Серый
//        String details = super.getDetails();    //details = Name: Том color: Серый
//        String result = "Я кошка, " + details;  // result = Я кошка, Name: Том color: Се
        String result = "Я кошка, " + "Name: " + super.getName() + " color: " + super.get(
        return result;
    }

    @Override
    public String getName() {
        return "tram-pam-pam";
    }

    /**
     * Метод, который позволяет кошке лазить по деревьям.
     */
    public void climbTree() {
        System.out.println("Кошка лазит по деревьям.");
    }

    /**
     * Метод, представляющий охотничьи навыки кошки.
     */
    public void hunt() {
        System.out.println("Кошка идет на охоту.");
    }
}
```

code/Lesson_20/src/animal2/Dog.java

```
package animal2;

/**
 * @author Andrej Reutow
 * created on 02.10.2023
 */
public class Dog extends Animal {

    public Dog(String name, String color) {
        super(name, color);
```

```
    }

    @Override // переопределили метод супер класса
    public void makeSound() { // переопределили метод супер класса
        System.out.println("Гав-гав!");  // указали поведение метода для класса Dog
    }

    /**
     * Метод, который позволяет собаке вернуть мяч после его броска
     */
    public void fetchBall() {
        System.out.println("Собака принесла мяч");
    }

    /**
     * Метод, который представляет действие охраны дома
     */
    public void guardHouse() {
        System.out.println("Собака охраняет дом");
    }
}
```

code/Lesson_20/src/animal2/AnimalApl3.java

```
package animal2;

/**
 * @author Andrej Reutow
 * created on 02.10.2023
 */
public class AnimalApl3 {

    public static void main(String[] args) {
        Animal animal = new Animal("Без имени", "прозрачный");
        Animal animalDog = new Dog("Снежок", "Черный");
        Animal animalCat = new Cat("Том", "Серый");

        System.out.println("animal: " + animal.getName());
        System.out.println("animalDog: " + animalDog.getName());
        System.out.println("animalCat: " + animalCat.getName());

        System.out.println();
        System.out.println("animal: " + animal.getDetails());
        System.out.println("animalDog: " + animalDog.getDetails());
        System.out.println("animalCat: " + animalCat.getDetails());

        if (animalCat instanceof Cat) {
            Cat cat = (Cat) animalCat;
            cat.hunt();
```

```
            String name = ((Cat) animalCat).getName();
        }
    }
}
```

code/ClassWork_20/src/zoo2/Main.java

```
package zoo2;

public class Main {
    public static void main(String[] args) {
        // Восходящее приведение (Upcasting)
        Dog dog = new Dog(); // Создаем объект типа Dog
        Animal animal = dog; // Восходящее приведение, объект типа Dog становится объектом

        // Вызываем методы суперкласса Animal
        animal.makeSound(); // Выведет: Звук животного

        // Нисходящее приведение (Downcasting)
        Animal animal2 = new Cat(); // Создаем объект типа Cat и приводим его к типу Anima

        // В этой части кода мы начнем использовать нисходящее приведение,
        // чтобы вернуться к типам Dog и Cat.

        // Это потенциально опасно, так как animal2 на самом деле содержит объект типа Cat
        // и нельзя быть уверенным, что он может быть безопасно приведен к типу Dog.
        // Мы должны проверить, является ли он экземпляром Dog, прежде чем выполнять нисхо
        if (animal2 instanceof Dog) {
            Dog anotherDog = (Dog) animal2; // Нисходящее приведение

            // Теперь мы можем вызвать метод makeSound() специфичный для Dog.
            // Обратите внимание, что makeSound() переопределен в классе Dog.
            anotherDog.makeSound(); // Выведет: Гав-гав!
        } else {
            System.out.println("Этот объект не является экземпляром Dog.");
        }

        // Попробуем еще одно нисходящее приведение
        // В этом случае, объект animal2 на самом деле является экземпляром Cat,
        // и мы безопасно можем привести его к типу Cat.
        if (animal2 instanceof Cat) {
            Cat cat = (Cat) animal2; // Нисходящее приведение

            // Теперь мы можем вызвать метод makeSound() специфичный для Cat.
            // Обратите внимание, что makeSound() переопределен в классе Cat.
            cat.makeSound(); // Выведет: Мяу-мяу!
        } else {
            System.out.println("Этот объект не является экземпляром Cat.");
        }
```

```
    }
}
```

code/ClassWork_20/src/zoo2/Animal.java

```
package zoo2;

/**
 * @author Andrej Reutow
 * created on 01.10.2023
 */
public class Animal {
    public void makeSound() {
        System.out.println("Звук животного");
    }
}
```

code/ClassWork_20/src/zoo2/Cat.java

```
package zoo2;

public class Cat extends Animal {

    @Override
    public void makeSound() {
        System.out.println("Мяу-мяу!");
    }

    /**
     * Метод, который позволяет кошке лазить по деревьям. Это типичное действие для кошек
     */
    public void climbTree() {
        System.out.println("Кошка лазит по деревьям.");
    }

    /**
     * Метод, представляющий охотничьи навыки кошки. Кошки часто охотятся на мелких животн
     */
    public void hunt() {
        System.out.println("Кошка идет на охоту.");
    }
}
```

code/ClassWork_20/src/zoo2/Dog.java

```
package zoo2;

public class Dog extends Animal {
```

```
    @Override
    public void makeSound() {
        System.out.println("Гав-гав!");
    }

    /**
     * Метод, который позволяет собаке вернуть мяч после его броска. Это типичное действие
     */
    void fetchBall() {
        System.out.println("Собака принесла мяч.");
    }

    /**
     * Метод, который представляет действие охраны дома. Собаки часто используются в каче
     */
    void guardHouse() {
        System.out.println("Собака охраняет дом.");
    }
}
```