**Lab1**

**Задание:**

**Создайте цепочку наследования, состоящую из одного родительского класса и трёх классов-наследников. Кроме того, нужно создать два интерфейса, каждый должен содержать минимум одно объявление метода. Два из созданных классов должны реализовывать хотя бы один интерфейс, а один класс должен реализовывать оба интерфейса.  
  
Классы должны переопределять методы toString и equals.  
  
Необходимо загрузить отчет, код должен быть представлен на паре**

**В методе `main` создайте массивы объектов типа родительского класса и массивы объектов, относящихся к интерфейсам, и продемонстрируйте работу с экземплярами всех классов через эти массивы.**

В коде оставил ко всему комментарии, поэтому описываю общую структуру и реализацию.

**Основная идея**: Создание абстрактного класса Animal и подклассов Cat, Dog, Racoon на его основе.

Родительский класс описывает структуру дочерних и 2 общих метода equals & isUnic, так как данные методы во всех классах реализовываются одинаково.

Equals проверяет если объект, переданный в качестве аргумента, принадлежит тому же классу, что и вызывающий данный метод объект, либо если является его подклассом.  
isUnic – проверяет уникален ли элемент в списке (если он встречается 1 раз или более).

Так как по условию классы должны переопределять методы equals & toString: первый переопределил в суперклассе, второй в каждом отдельно для наглядной демонстрации.

Из соображений совершенствования кода можно было создать в переопределить в родительском классе, но для наглядности оставил в каждом.

**Алгоритм:**

-Создание 2 интерфейсов, которые должен будет унаследовать родительский класс

-Создание абстрактного родительского класса и переопределение необходимых методов, описание общих методов/абстрактных

-Создание дочерних классов и переопределение заданных методов

-Создание массивов объектов и проверка корректности работы программы

**Листинг кода:**

Интерфейсы:

Movable:  
  
public interface Movable {  
 void move();  
}

Soundable:

public interface Soundable {  
 void voice();  
}

Суперкласс Animal:

//Абстрактный родительский класс, который обязует дочерние реализоть/переопределить в себе методы toString() & equals(),  
// и реализовать интерфейсы Movable & Soundable  
public abstract class Animal implements Movable, Soundable{  
  
 // toString объявляю абстрактным, так как данный метод должен быть переопределен в каждом классе (so i want)  
 @Override  
 public abstract String toString();  
  
 // equals должен будет просто проверять, относятли ли объекты к одному классу или нет  
 @Override  
 public boolean equals(Object object){  
 if (this == object) return true; //если вдруг это один и тот же объект  
 return(this.getClass() == object.getClass()); // если разные объекты, но экземпляры одного класса  
 }  
  
 // Для демонстрации работы метода equals: проверяет, если в списке только один экземпляр определнного класса.  
 public String isUnic(Animal object, Animal[] animals){  
 int count = 0;  
 for (Animal animal: animals){  
 if (object.getClass().isInstance(animal)) count++;  
 }  
 return count >1 ? "There are many " + object + "'s in list!"  
 : "There is the only one " + object + " in list!";  
 }  
}

Дочерние классы:

// Переопределяю методы родительского класса  
class Cat extends Animal {  
  
 @Override  
 public void move() {  
 System.*out*.println("The cat moves carefully");  
 }  
  
 @Override  
 public void voice() {  
 System.*out*.println("Meow");  
 }  
  
 @Override  
 public String toString() {  
 return "Cat";  
 }  
}

// Переопределяю методы родительского класса  
class Dog extends Animal{  
  
 @Override  
 public void voice() {  
 System.*out*.println("Gav");  
 }  
  
 @Override  
 public void move() {  
 System.*out*.println("The dog moves rapidly");  
 }  
  
 @Override  
 public String toString(){  
 return "Dog";  
 }  
  
}

public class Racoon extends Animal{  
  
 @Override  
 public void voice() {  
 System.*out*.println("I dkn what he says");  
 }  
  
 @Override  
 public void move() {  
 System.*out*.println("Racoon is lazy and do not moves. He sleeps.");  
 }  
  
 @Override  
 public String toString() {  
 return "Racoon";  
 }  
  
}

Main:

public class Main {  
 public static void main(String[] args) {  
 Animal[] animals = new Animal[4];  
 animals[0] = new Cat();  
 animals[1] = new Dog();  
 animals[2] = new Racoon();  
 animals[3] = new Cat();

// Демонтрация работы методов объектов всех классов  
 for (Animal animal:animals){  
 System.*out*.println(animal);  
 animal.move();  
 animal.voice();  
 System.*out*.println(animal.isUnic(animal, animals));  
 System.*out*.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");  
 }  
 // Объекты интерфейсов создавать напрямую нельзя, но можно создать ссылку на такой объект  
 // Так как у меня все классы дочерние реализуют оба интерфейса, могу создать ссылку на объект любого дочернего Animal класса  
  
 Soundable[] soundables = new Soundable[]{new Cat()};  
 Movable[] movables = new Movable[]{new Racoon()};  
  
 // Демонстрация работы методов даннных объектов  
 System.*out*.println(soundables[0]);  
 soundables[0].voice();  
 System.*out*.println(movables[0]);  
 movables[0].move();  
 Soundable a = soundables[0];  
  
 }  
}

Результат при запуске программы:  
Cat

The cat moves carefully

Meow

There are many Cat's in list!

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Dog

The dog moves rapidly

Gav

There is the only one Dog in list!

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Racoon

Racoon is lazy and do not moves. He sleeps.

I dkn what he says

There is the only one Racoon in list!

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Cat

The cat moves carefully

Meow

There are many Cat's in list!

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Cat

Meow

Racoon

Racoon is lazy and do not moves. He sleeps.

Process finished with exit code 0