# Лабораторная работа № 1

**Цель:**

Изучить раздел рекомендаций при проектировании иерархии книги Блинова. Научиться работать с наследованием и полиморфизмом.

**Постановка задачи:**

Написать приложение, используя наследование, полиморфизм, если это логически обосновано. Создать объект класса “Щенок”, используя классы “Животное”, “Собака”. Методы: вывести на консоль имя, подать голос, прыгать, бегать, кусать. Аргументировать принадлежность классу каждого метода и корректно переопределить для каждого класса методы “equals()”, “hashCode()”, “toString()”. Разработать UML-диаграмму с помощью среды.

**Решение задачи:**

При создании приложения учитывается логическая обоснованность наследования. Класс “Animal” содержит в себе такое поле, как “Type type” (enumeration), включающее 5 типов животных (“Mammal”, “Fish”, “Reptile”, “Amphibian”, “Bird”), для которых в методах “jump()” и “run()” определены некоторые свойства: если объект класса принадлежит типу “Fish” (“Рыба”), то он не может ни бегать, ни прыгать, а “Mammal” (“Млекопитающее”), наоборот, обладает и тем, и другим навыком и т.п.

```
public enum Type {MAMMAL, FISH, REPTILE, AMPHIBIAN, BIRD}
private Type type;
```

*Рис.1. Поле “Type type” класса “Animal”.*

```
public String jump(){
    if(type == Type.MAMMAL || type == Type.BIRD){
        return "Skill " + Skill.EXISTS.toString().toLowerCase();
    }
    else
        return "Skill is " + Skill.ABSENT.toString().toLowerCase();
}
public String run(){
    if(type == Type.MAMMAL || type == Type.REPTILE || type == Type.AMPHIBIAN){
        return "Skill " + Skill.EXISTS.toString().toLowerCase();
    }
    else
        return "Skill is " + Skill.ABSENT.toString().toLowerCase();
}
```

*Рис.2. Методы “jump()” и “run()”.*

В конструкторе с параметрами класса “Dog” (наследуется от класса “Animal”) прописывается метод “super()” с целью вызова конструктора абстрактного класса, изначально указывается, что тип животного - млекопитающее.

```
public Dog (String breed, Maturity maturity, Skill skill) throws BadInputException {
    super(Type.MAMMAL);
    if(breed == null)
        throw new BadInputException("Breed cannot be null!");
    if(maturity == null)
        throw new BadInputException("Maturity cannot be null!");
    if(skill == null)
        throw new BadInputException("Skill cannot be null!");

    this.breed = breed;
    this.maturity = maturity;
    this.skill = skill;
}
```

*Рис.3.Конструктор с параметрами класса “Dog”.*

Требуемый метод “vote()” (“подать голос”) реализуется в классе “Dog”. Существование или отсутствие навыка указывается при создании объекта.

```
public String vote(){
    if(skill == Skill.ABSENT){
        return "<Skill is " + Skill.ABSENT.toString().toLowerCase() + ">";
    }
    else
        return "Skill " + Skill.EXISTS.toString().toLowerCase() + ">";
}
```

*Рис.4.Метод “vote()” класса “Dog”.*

Метод “getName()” (вывод имени объекта на консоль) определяем непосредственно в классе “Puppy”, который является наследником класса “Dog”, причем создание объекта данного класса возможно только при условии, что поле “Maturity maturity”(enumeration) принимает значение “Maturity.PUPPY”.

```
public String getName(){
    return name;
}
```

*Рис.5. Метод “getName()” класса “Puppy”.*

```
public String bite(){
    if(bites == How.CAREFULLY) {
        return "Bites carefully. Good boy:)";
    }
    if(bites == How.PAINFULLY){
        return "Bites painfully. Don't run up!";
    }
    else
        return "Doesn't bites. Toothless puppy.";
}
```

*Рис.6. Метод “bite()” (“кусать”), отражающий способность щенка (объекта) кусаться.*

Согласно условию следует переопределить для каждого класса методы “toString()”, “hashCode()” и “equals()”. Во всех случаях используется “super()” в качестве отсылки к параметрам класса-предка.

```
@Override
public String toString() {
    StringBuilder sb = new StringBuilder(" | Name: ");
    sb.append(name).append(" | Bites: ").append(bite()).
            append(super.toString());
    return sb.toString();
}

@Override
public int hashCode() {
    return Objects.hash(super.hashCode(), name, bites);
}

@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (!(o instanceof Puppy)) return false;
    if (!super.equals(o)) return false;
    Puppy puppy = (Puppy) o;
    return name.equals(puppy.name) && bites.equals(puppy.bites);
}
```

*Рис.7. Переопределенные в классе “Puppy” методы “toString()”, “hashCode()” и “equals()”.*

**Результат:**

Результат работы программы есть не что иное, как тестирование всех методов для объектов класса “Puppy”.

```
    TESTING METHODS
***toString***
 | Name: Keks | Bites: Bites carefully. Good boy:) | Breed: Pug | Maturity: PUPPY || Vote: <Skill is absent>

 | Type of animal: MAMMAL | Jump: Skill exists | Run: Skill exists

 ***jump***
 Skill exists
 ***run***
 Skill exists
 ***getName***
 Keks
 ***bite***
 Doesn't bites. Toothless puppy.
 Bites painfully. Don't run up!
 Bites carefully. Good boy:)
```

*Рис.8. Результат лабораторной работы № 1.*

UML-диаграмма дает возможность наглядно рассмотреть наследование в программе.

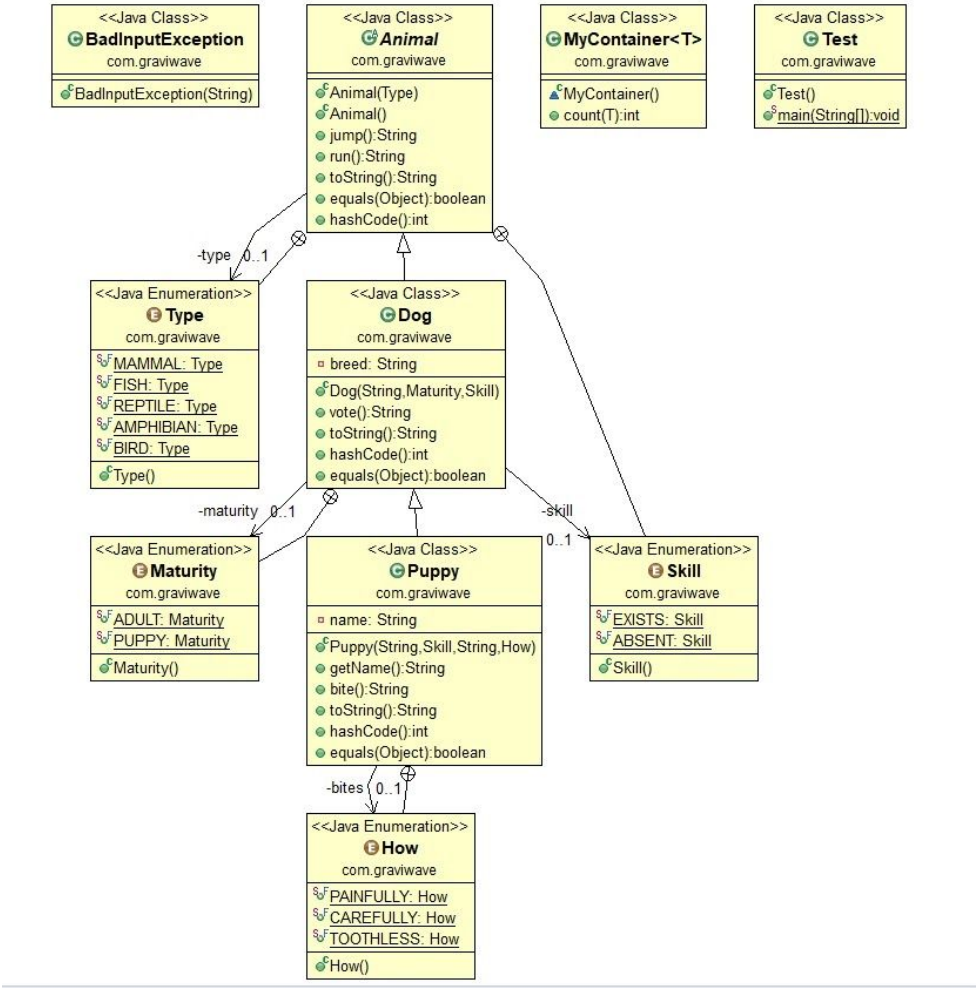


*Рис.9. UML-диаграмма.*