Пример 1: Программа "Forum"

Цель примера

Продемонстрировать процесс создания простейшей программы с использованием принципа наследования.

Введение

Наследование является одним из ключевых аспектов объектно-ориентированного программирования. С помощью наследования можно расширить функционал уже имеющихся классов или изменить, а также уменьшить дублирование кода.

Один класс может наследовать характеристики другого, а именно его методы и переменные.

‼️ **Примечание**: Класс может наследовать только те элементы, которые не помечены модификатором доступа `private` .

Практическое руководство

Рассмотрим программу, которая демонстрирует наследование на примере форума. На форуме есть различные типы учетных записей:

- Пользователь `User` – может читать сообщения, писать сообщения;
- Модератор `Moderator` – может удалять сообщения и все доступные возможности пользователя;
- Администратор `Administrator` – может блокировать пользователя и все доступные возможности модератора;

Шаг 1.

Создадим класс `User` с произвольной логикой каждого из методов:

```
public class User {

    public void readPost() {
        System.out.println("Reading post");
    }

    public void writePost() {
        System.out.println("Writing post");
    }

}
```

Шаг 2.

Создадим класс `Moderator` с произвольной логикой, который также выполняет те же действия, что и пользователь:

```
public class Moderator {

    public void readPost() {
        System.out.println("Reading post");
    }

    public void writePost() {
        System.out.println("Writing post");
    }

    public void deletePost() {
        System.out.println("Deleting post");
    }

}
```

Шаг 3.

Создадим класс `Administrator` с произвольной логикой, который также выполняет те же действия, что модератор и пользователь:

```
public class Administrator {

    public void readPost() {
        System.out.println("Reading post");
    }

    public void writePost() {
        System.out.println("Writing post");
    }

    public void deletePost() {
        System.out.println("Deleting post");
    }

    public void blockUser() {
        System.out.println("Blocking user");
    }

}
```

Шаг 4.

Продемонстрируем работу методов каждого из классов в классе `Main`:

```
public class Main {

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

```
        User user = new User();
        Moderator moderator = new Moderator();
        Administrator administrator = new Administrator();

        user.readPost();
        user.writePost();

        System.out.println();

        moderator.readPost();
        moderator.writePost();
        moderator.deletePost();

        System.out.println();

        administrator.readPost();
        administrator.writePost();
        administrator.deletePost();
        administrator.blockUser();

    }

}
```

Результат работы программы:

> Reading post Writing post
>
> Reading post Writing post Deleting post
>
> Reading post Writing post Deleting post Blocking user

Видно, что основные условия программы выполняются, однако сейчас в коде есть ряд недостатков: логика работы методов одинаковая, независимо от того, какой из классов ее выполняет, следовательно это являются дублированием кода. При необходимости изменить логику одного из методов, например записи поста, то ее нужно менять во всех классах.

Чтобы уменьшить дублирование кода, существующий функционал дополняют используя механизм наследования.

Шаг 5.

Изменим код таким образом, чтобы логика `Moderator` наследовалась от класса `User`:

```
public class Moderator extends User {

    public void deletePost() {
        System.out.println("Deleting post");
    }

}
```

Результат работы программы:

> Reading post Writing post
>
> Reading post Writing post Deleting post
>
> Reading post Writing post Deleting post Blocking user

Обратите внимание, что логика работы не изменилась, но количество кода стало меньше.

Шаг 6.

Наследуем логику класса `Moderator` в классе `Administrator` :

```
public class Administrator extends Moderator {

    public void blockUser() {
        System.out.println("Blocking user");
    }

}
```

Результат работы программы:

> Reading post Writing post
>
> Reading post Writing post Deleting post
>
> Reading post Writing post Deleting post Blocking user

Логика работы осталась прежней.

Шаг 7.

Предположим класс `Administrator` пишет пост с другой логикой. В таком случае поведение метода можно переопределить:

```
public class Administrator extends Moderator {

    @Override
    public void writePost() {
        System.out.println("Administrator: Writing post");
    }

    public void blockUser() {
        System.out.println("Blocking user");
    }

}
```

Результат работы программы:

> Reading post Writing post
>
> Reading post Writing post Deleting post
>
> Reading post Administrator: Writing post Deleting post Blocking user

Видно, что логика работы изменилась только в классе `Administrator` . При необходимости, переопреоедять можно методы в любой точке иерархии, однако в таком случае изменения будут влиять на все классы – наследники.

Рекомендации:

- Запустить программу и сравнить результаты;
- Попробовать изменить логику работы в методе класса `User` и сравнить результаты;
- Попробовать изменить логику работы в методе класса `Moderator` и сравнить результаты;
- Переопределить методы в классах `Moderator` и `Administrator` и сравнить результаты;
- Дополнить произвольную логику в наследуемый метод, не изменяя поведение класса – родителя, используя `super`, например `super.readPost()`;