Наследование и полиморфизм

Лекция 3: Продвинутые концепции ООП

Преподаватель: [Ваше имя]

Группа: 203

Семестр: Осенний 2024

План лекции

1. **Наследование в Java**
2. **Переопределение методов**
3. **Абстрактные классы**
4. **Полиморфизм**
5. **Практический пример: Иерархия юнитов**

Наследование в Java

Ключевые концепции:

- **Наследование** — создание нового класса на основе существующего
- **Базовый класс** (родительский, суперкласс) — класс, от которого наследуются
- **Производный класс** (дочерний, подкласс) — класс, который наследует

Синтаксис:

```
public class Подкласс extends Суперкласс {
    // новые поля и методы
    // переопределенные методы
}
```

Пример наследования

```java
// Базовый класс
public class Unit {
    protected String name;
    protected int health;
    protected int maxHealth;
    protected Position position;

    public Unit(String name, int health) {
        this.name = name;
        this.health = health;
        this.maxHealth = health;
    }

    public void takeDamage(int damage) {
        this.health -= damage;
        if (this.health < 0) this.health = 0;
    }

    public boolean isAlive() {
        return this.health > 0;
    }
}

// Производный класс
public class Warrior extends Unit {
    private int armor;

    public Warrior(String name, int health, int armor) {
        super(name, health);  // вызов конструктора суперкласса
        this.armor = armor;
    }
}
```

Ключевое слово `super`

Использование:

- **Вызов конструктора** суперкласса
- **Обращение к методам** суперкласса
- **Обращение к полям** суперкласса

```
public class Archer extends Unit {
    private int range;

    public Archer(String name, int health, int range) {
        super(name, health);  // вызов конструктора Unit
        this.range = range;
    }

    @Override
    public void takeDamage(int damage) {
        // Сначала вызываем метод суперкласса
```

Переопределение методов

Правила переопределения:

- **Сигнатура метода** должна совпадать
- **Возвращаемый тип** должен быть совместим
- **Модификатор доступа** не может быть более строгим
- **Исключения** не могут быть более широкими

Аннотация `@Override`:

```
@Override
public void takeDamage(int damage) {
    // новая реализация
}
```

Примеры переопределения

```
public class Unit {
    public void performAction() {
        System.out.println("Unit performs basic action");
    }

    public String getDescription() {
        return "Basic unit";
    }
}

public class Warrior extends Unit {
    @Override
    public void performAction() {
        System.out.println("Warrior attacks with sword!");
    }

    @Override
    public String getDescription() {
        return "Strong melee fighter";
    }
}

public class Mage extends Unit {
    @Override
    public void performAction() {
        System.out.println("Mage casts a spell!");
    }

    @Override
    public String getDescription() {
        return "Powerful magic user";
    }
}
```

Абстрактные классы

Особенности:

- **Нельзя создать экземпляр** абстрактного класса
- **Может содержать абстрактные методы** (без реализации)
- **Может содержать обычные методы** с реализацией
- **Наследуется** как обычный класс

Синтаксис:

```
public abstract class AbstractUnit {
    protected String name;

    // Абстрактный метод (без реализации)
    public abstract void performAction();
```

Абстрактные методы

Характеристики:

- **Не имеют тела** (реализации)
- **Должны быть реализованы** в подклассах
- **Могут быть только в абстрактных классах**
- **Подкласс должен реализовать ВСЕ** абстрактные методы

```
public abstract class Unit {
    protected String name;
    protected int health;

    // Абстрактные методы
    public abstract void performAction();
    public abstract String getUnitType();
    public abstract int getAttackRange();
```

Полиморфизм

Определение:

Полиморфизм — способность объектов с одинаковым интерфейсом иметь различную реализацию.

Типы полиморфизма:

1. **Полиморфизм времени выполнения** (переопределение методов)
2. **Полиморфизм времени компиляции** (перегрузка методов)

Полиморфизм времени выполнения

```
public class Game {
    public static void main(String[] args) {
        // Создаем массив разных типов юнитов
        Unit[] units = new Unit[3];
        units[0] = new Warrior("Aragorn", 200);
        units[1] = new Archer("Legolas", 150);
        units[2] = new Mage("Gandalf", 120);

        // Полиморфный вызов методов
        for (Unit unit : units) {
            unit.performAction();  // Вызывается разная реализация
            System.out.println(unit.getDescription());
        }
    }
}
```

Вывод:

Практический пример: Иерархия юнитов

```
// Абстрактный базовый класс
public abstract class Unit {
    protected String name;
    protected int health;
    protected int maxHealth;
    protected Position position;

    public abstract void performAction();
    public abstract String getUnitType();
    public abstract int getAttackRange();

    public void moveTo(Position newPosition) {
        this.position = newPosition;
    }

    public boolean canAttack(Unit target) {
        int distance = this.position.getDistanceTo(target.position);
        return distance <= this.getAttackRange();
    }
}
```

Конкретные классы юнитов

```
public class Warrior extends Unit {
    private int armor;

    public Warrior(String name, int health, int armor) {
        this.name = name;
        this.health = health;
        this.maxHealth = health;
        this.armor = armor;
    }

    @Override
    public void performAction() {
        System.out.println(name + " attacks with sword!");
    }

    @Override
    public String getUnitType() {
        return "Warrior";
    }

    @Override
    public int getAttackRange() {
        return 1;  // Ближний бой
    }
}
```

Использование полиморфизма

```
public class Army {
    private List<Unit> units = new ArrayList<>();

    public void addUnit(Unit unit) {
        units.add(unit);
    }

    public void performAllActions() {
        for (Unit unit : units) {
            unit.performAction();  // Полиморфный вызов
        }
    }

    public List<Unit> getUnitsByType(String type) {
        return units.stream()
                    .filter(unit -> unit.getUnitType().equals(type))
                    .collect(Collectors.toList());
    }
}
```

Преимущества наследования и полиморфизма

Наследование:

- ✅ Переиспользование кода
- ✅ Иерархическая организация
- ✅ Расширяемость системы

Полиморфизм:

- ✅ Гибкость кода
- ✅ Единый интерфейс
- ✅ Легкость добавления новых типов

Домашнее задание

Задача 1:

Создать абстрактный класс `Building` с методами:

- `abstract void produce()`
- `abstract String getType()`

Задача 2:

Реализовать классы `Barracks`, `MageTower`, `Farm`, наследующие от `Building`

Задача 3:

Создать класс `City` с полиморфным методом `produceAll()`

Что дальше?

На следующей лекции:

- **Интерфейсы**
- **Множественное наследование**
- **Default методы**

Подготовка:

- Изучить главу 5-6 из учебника
- Выполнить домашнее задание
- Подготовить вопросы по текущей теме

Вопросы?

Контакты:

- **Email:** [ваш.email@university.edu]
- **Telegram:** [@username]
- **Офис:** [номер кабинета]

Следующая лекция: Интерфейсы и абстрактные классы