Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования «Алтайский государственный технический университет им. И. И. Ползунова»

Факультет информационных технологий Кафедра прикладной математики

Отчет защищен с оценкой _____

Преподаватель _____
(подпись)
«___» _____2025 г.

Отчет
По лабораторной работе №4

«Программирование на языке с автоматическим управлением памятью»

по дисциплине «Программирование – 3 семестр»

Студент группы ПИ-42 Сычев Я. Р.

Преподаватели: Троицкий Виктор Сергеевич, Рахманин Данила Сергеевич

Барнаул 2025

Main.java

```
import java.util.ArrayList;
import java.util.List;

public class Main {
    // Константы для демонстрации
    private static final int CASTLE_INITIAL_HEALTH = 200;
    private static final int METEOR1_HEALTH = 80;
    private static final int METEOR1_SPEED = 2;
    private static final int METEOR1_SIZE = 4;
    private static final int METEOR1_DAMAGE = 20;
    private static final int METEOR2_HEALTH = 70;
    private static final int METEOR2_SPEED = 3;
    private static final int METEOR2_SIZE = 3;
    private static final int METEOR2_DAMAGE = 15;
    private static final int BOSS_HEALTH = 300;
    private static final int BOSS_SPEED = 1;
    private static final int BOSS_PHASE = 1;
    private static final int BOSS_SIZE = 12;
    private static final int BOSS_DAMAGE = 40;
    private static final int PLAYER_POSITION_X = 100;
    private static final int PLAYER_POSITION_Y = 150;
    private static final int PROJECTILE1_POSITION_X = 50;
    private static final int PROJECTILE1_POSITION_Y = 50;
    private static final int PROJECTILE2_POSITION_X = 60;
    private static final int PROJECTILE2_POSITION_Y = 60;
    private static final int METEOR_DAMAGE_AMOUNT = 15;
    private static final int METEOR2_DAMAGE_AMOUNT = 20;
    private static final int CASTLE_DAMAGE_AMOUNT = 25;
    private static final int BOSS_DAMAGE_THRESHOLD = 200;
    private static final int DYNAMIC_METEORS_COUNT = 2;
    private static final int PROJECTILE_ARRAY_SIZE = 2;

    public static void main(String[] args) {
        System.out.println("ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ИГРЫ 'FLYING DANGER' НА
JAVA\n");
        System.out.println("ХРОНОЛОГИЧЕСКАЯ ПОСЛЕДОВАТЕЛЬНОСТЬ ДЕЙСТВИЙ:");

        // 1. Создание основных объектов
        System.out.println("\n1. СОЗДАНИЕ ОСНОВНЫХ ОБЪЕКТОВ:");
        GameManager gameManager = new GameManager();

        // 2. Создание игровых объектов
        System.out.println("\n2. СОЗДАНИЕ ИГРОВЫХ ОБЪЕКТОВ:");
        Castle castle = new Castle(CASTLE_INITIAL_HEALTH);
        Player player = new Player(gameManager);

        // 3. Старт игры
        System.out.println("\n3. ЗАПУСК ИГРОВОГО ПРОЦЕССА:");
        gameManager.run();

        // 4. Создание врагов
        System.out.println("\n4. СОЗДАНИЕ ВРАГОВ:");
        Meteor meteor1 = new Meteor(METEOR1_HEALTH, METEOR1_SPEED,
METEOR1_SIZE, METEOR1_DAMAGE, gameManager);
        Meteor meteor2 = new Meteor(METEOR2_HEALTH, METEOR2_SPEED,
METEOR2_SIZE, METEOR2_DAMAGE, gameManager);
        Boss boss = new Boss(BOSS_HEALTH, BOSS_SPEED, BOSS_PHASE, BOSS_SIZE,
BOSS_DAMAGE, gameManager);

        // 5. Добавление врагов в менеджер
        System.out.println("\n5. ДОБАВЛЕНИЕ ВРАГОВ В ИГРУ:");
        gameManager.addEnemy(meteor1);
```

```
        gameManager.addEnemy(meteor2);
        gameManager.addEnemy(boss);

        // 6. Действия игрока
        System.out.println("\n6. ДЕЙСТВИЯ ИГРОКА:");
        player.move(PLAYER_POSITION_X + "," + PLAYER_POSITION_Y);
        player.shoot();
        player.update();

        // 7. Движение врагов
        System.out.println("\n7. ДВИЖЕНИЕ ВРАГОВ:");
        meteor1.move();
        meteor2.move();
        boss.move();

        // 8. Создание снарядов
        System.out.println("\n8. СОЗДАНИЕ СНАРЯДОВ:");
        Projectile projectile1 = new Projectile(PROJECTILE1_POSITION_X + ","
+ PROJECTILE1_POSITION_Y, "1,0", 25, player);
        Projectile projectile2 = new Projectile(PROJECTILE2_POSITION_X + ","
+ PROJECTILE2_POSITION_Y, "1,1", 30, player);

        // 9. Обновление снарядов
        System.out.println("\n9. ОБНОВЛЕНИЕ СНАРЯДОВ:");
        projectile1.update();
        projectile2.update();

        // 10. Атаки и получение урона
        System.out.println("\n10. АТАКИ И ПОЛУЧЕНИЕ УРОНА:");
        meteor1.takeDamage(METEOR_DAMAGE_AMOUNT);
        meteor2.takeDamage(METEOR2_DAMAGE_AMOUNT);
        castle.takeDamage(CASTLE_DAMAGE_AMOUNT);

        // 11. Проверка столкновений
        System.out.println("\n11. ПРОВЕРКА СТОЛКНОВЕНИЙ:");
        projectile1.checkCollision(meteor1);
        meteor1.checkCollision("castle");

        // 12. Смена фазы босса
        System.out.println("\n12. СМЕНА ФАЗЫ БОССА:");
        boss.takeDamage(BOSS_DAMAGE_THRESHOLD);

        // 13. Демонстрация полиморфизма
        System.out.println("\n13. ПОЛИМОРФИЗМ:");
        List<Enemy> enemies = new ArrayList<>();
        enemies.add(meteor1);
        enemies.add(meteor2);
        enemies.add(boss);

        for (Enemy enemy : enemies) {
            enemy.update();
            enemy.draw();
        }

        // 14. Динамический массив объектов
        System.out.println("\n14. ДИНАМИЧЕСКИЙ МАССИВ ОБЪЕКТОВ:");
        Projectile[] projectileArray = new Projectile[PROJECTILE_ARRAY_SIZE];
        for (int i = 0; i < PROJECTILE_ARRAY_SIZE; i++) {
            projectileArray[i] = new Projectile();
            projectileArray[i].update();
        }

        // 15. Массив динамических объектов
        System.out.println("\n15. МАССИВ ДИНАМИЧЕСКИХ ОБЪЕКТОВ:");
```

```
        Meteor[] dynamicMeteors = new Meteor[DYNAMIC_METEORS_COUNT];
        for (int i = 0; i < DYNAMIC_METEORS_COUNT; i++) {
            dynamicMeteors[i] = new Meteor(50 + i * 10, 2, 3, 15,
gameManager);
            dynamicMeteors[i].move();
        }

        System.out.println("\nКОНЕЦ ДЕМОНСТРАЦИИ!");

        System.out.println("Для выхода нажмите Enter...");
        try {
            System.in.read();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Castle.java

```
public class Castle {
    private int health;
    private int maxHealth;
    private String position;
    private String sprite;
    private String rect;

    // Константы
    public static final int DEFAULT_HEALTH = 100;
    public static final int DEFAULT_MAX_HEALTH = 100;
    public static final int CASTLE_ATTACK_DAMAGE = 10;

    public Castle() {
        this.health = DEFAULT_HEALTH;
        this.maxHealth = DEFAULT_MAX_HEALTH;
        this.position = "";
        this.sprite = "";
        this.rect = "";
        System.out.println("Объект класса Castle создан");
    }

    public Castle(int maxHealth) {
        this.health = maxHealth;
        this.maxHealth = maxHealth;
        this.position = "";
        this.sprite = "";
        this.rect = "";
        System.out.println("Объект класса Castle создан с максимальным
здоровьем: " + maxHealth);
    }

    public void takeDamage(int amount) {
        health -= amount;
        System.out.println("Метод takeDamage() класса Castle выполнен.
Получено урона: " + amount);
        System.out.println("Осталось здоровья замка: " + health + "/" +
maxHealth);
    }

    public boolean isDestroyed() {
        boolean destroyed = health <= 0;
        System.out.println("Метод isDestroyed() класса Castle выполнен.
Результат: " +
```

```
                (destroyed ? "разрушен" : "цел"));
        return destroyed;
    }

    public void draw() {
        System.out.println("Метод draw() класса Castle выполнен");
    }

    public int getHealth() { return health; }
    public int getMaxHealth() { return maxHealth; }
}
```

Projectile.java

```
public class Projectile {
    private String position;
    private String direction;
    private int speed;
    private int damage;
    private String rect;
    private boolean isActive;
    private Player owner;

    // Константы
    public static final int DEFAULT_SPEED = 10;
    public static final int DEFAULT_DAMAGE = 10;

    public Projectile() {
        this.position = "";
        this.direction = "";
        this.speed = DEFAULT_SPEED;
        this.damage = DEFAULT_DAMAGE;
        this.rect = "";
        this.isActive = true;
        this.owner = null;
        System.out.println("Объект класса Projectile создан");
    }

    public Projectile(String pos, String dir, int dmg) {
        this();
        this.position = pos;
        this.direction = dir;
        this.damage = dmg;
        System.out.println("Объект класса Projectile создан с параметрами");
    }

    public Projectile(String pos, String dir, int dmg, Player player) {
        this(pos, dir, dmg);
        this.owner = player;
        System.out.println("Объект класса Projectile создан с владельцем");
    }

    public void update() {
        System.out.println("Метод update() класса Projectile выполнен");
    }

    public boolean checkCollision(Enemy enemy) {
        System.out.println("Метод checkCollision() класса Projectile
выполнен");
        return false;
    }
```

```
    public void onHit() {
        System.out.println("Метод onHit() класса Projectile выполнен");
        isActive = false;
    }

    public void checkBoundaries() {
        System.out.println("Метод checkBoundaries() класса Projectile
выполнен");
    }

    public void draw() {
        System.out.println("Метод draw() класса Projectile выполнен");
    }

    public boolean isActive() { return isActive; }
    public int getDamage() { return damage; }
}
```

GameManager.java

```
import java.util.ArrayList;
import java.util.List;

public class GameManager {
    private String screen;
    private String clock;
    private String gameState;
    private int score;
    private int castleHealth;
    private Castle castle;
    private Player player;
    private List<Enemy> enemies;

    // Константы
    public static final String DEFAULT_GAME_STATE = "menu";
    public static final int DEFAULT_SCORE = 0;
    public static final int DEFAULT_CASTLE_HEALTH = 100;

    public GameManager() {
        this.screen = "";
        this.clock = "";
        this.gameState = DEFAULT_GAME_STATE;
        this.score = DEFAULT_SCORE;
        this.castleHealth = DEFAULT_CASTLE_HEALTH;
        this.enemies = new ArrayList<>();

        this.castle = new Castle(DEFAULT_CASTLE_HEALTH);
        this.player = new Player();

        System.out.println("Объект класса GameManager создан");
    }

    public void run() {
        System.out.println("Метод run() класса GameManager запущен");
    }

    public void switchState() {
        System.out.println("Метод switchState() класса GameManager
выполнен");
    }

    public void checkWinConditions() {
```

```
        System.out.println("Метод checkWinConditions() класса GameManager
выполнен");
    }

    public void handleEvents() {
        System.out.println("Метод handleEvents() класса GameManager
выполнен");
    }

    public void update() {
        System.out.println("Метод update() класса GameManager выполнен");
        System.out.println("Обновление состояния всех игровых объектов...");
    }

    public void addEnemy(Enemy enemy) {
        enemies.add(enemy);
        System.out.println("Враг добавлен в GameManager");
    }

    public void removeEnemy(Enemy enemy) {
        enemies.remove(enemy);
        System.out.println("Враг удален из GameManager");
    }
}
```

Enemy.java

```
public abstract class Enemy {
    protected int health;
    protected int maxHealth;
    protected int speed;
    protected String position;
    protected String sprite;
    protected String rect;
    protected GameManager gameManager;

    // Константы
    public static final int DEFAULT_HEALTH = 50;
    public static final int DEFAULT_MAX_HEALTH = 50;
    public static final int DEFAULT_SPEED = 2;
    public static final int CASTLE_ATTACK_DAMAGE = 10;

    public Enemy() {
        this.health = DEFAULT_HEALTH;
        this.maxHealth = DEFAULT_MAX_HEALTH;
        this.speed = DEFAULT_SPEED;
        this.position = "";
        this.sprite = "";
        this.rect = "";
        this.gameManager = null;
        System.out.println("Объект класса Enemy создан");
    }

    public Enemy(int hp, int spd) {
        this();
        this.health = hp;
        this.maxHealth = hp;
        this.speed = spd;
        System.out.println("Объект класса Enemy создан с параметрами");
    }

    public Enemy(int hp, int spd, GameManager gm) {
```

```
        this(hp, spd);
        this.gameManager = gm;
        System.out.println("Объект класса Enemy создан с ссылкой на
GameManager");
    }

    public abstract void move();
    public abstract void takeDamage(int amount);
    public abstract boolean checkCollision(String target);
    public abstract void update();
    public abstract void draw();

    public void attackCastle(Castle castle) {
        if (castle != null && isAlive()) {
            System.out.println("Враг атакует замок!");
            castle.takeDamage(CASTLE_ATTACK_DAMAGE);
        }
    }

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

    public void checkBoundaries() {
        System.out.println("Базовый метод checkBoundaries() класса Enemy");
    }

    public int getMaxHealth() { return maxHealth; }
}
```

Meteor.java

```
public class Meteor extends Enemy {
    private int size;
    private int damage;

    // Константы
    public static final int DEFAULT_SIZE = 3;
    public static final int DEFAULT_DAMAGE = 15;

    public Meteor() {
        super();
        this.size = DEFAULT_SIZE;
        this.damage = DEFAULT_DAMAGE;
        System.out.println("Объект класса Meteor создан");
    }

    public Meteor(int hp, int spd, int sz, int dmg) {
        super(hp, spd);
        this.size = sz;
        this.damage = dmg;
        System.out.println("Объект класса Meteor создан с параметрами");
    }

    public Meteor(int hp, int spd, int sz, int dmg, GameManager gm) {
        super(hp, spd, gm);
        this.size = sz;
        this.damage = dmg;
        System.out.println("Объект класса Meteor создан с ссылкой на
GameManager");
    }
```

```
    public void move() {
        System.out.println("Метод move() класса Meteor выполнен");
        System.out.println("Метеор размера " + size + " движется со скоростью
" + speed);
    }

    public void takeDamage(int amount) {
        health -= amount;
        System.out.println("Метод takeDamage() класса Meteor выполнен");
        System.out.println("Получено урона: " + amount + ", осталось
здоровья: " + health);

        if (!isAlive()) {
            System.out.println("Метеор уничтожен!");
        }
    }

    public boolean checkCollision(String target) {
        System.out.println("Метод checkCollision() класса Meteor выполнен с
целью: " + target);
        return true;
    }

    public void update() {
        System.out.println("Метод update() класса Meteor выполнен");

        move();
        checkBoundaries();
    }

    public void draw() {
        System.out.println("Метод draw() класса Meteor выполнен");
        System.out.println("Отрисовка метеора размера " + size + " в позиции
" + position);
    }

    public void applyDamage(int amount) {
        System.out.println("Метод applyDamage() класса Meteor выполнен");
        takeDamage(amount);
    }

    public void checkBoundaries() {
        System.out.println("Метод checkBoundaries() класса Meteor выполнен");
        System.out.println("Проверка границ экрана для метеора в позиции " +
position);

        if (position.contains("out")) {
            System.out.println("Метеор вышел за границы экрана!");
        }
    }

    public int getSize() { return size; }
    public int getDamage() { return damage; }
}
```

Boss.java

```
public class Boss extends Enemy {
    private int phase;
    private int size;
    private int damage;
```

```
    // Константы
    public static final int DEFAULT_PHASE = 1;
    public static final int DEFAULT_SIZE = 10;
    public static final int DEFAULT_DAMAGE = 30;
    public static final double PHASE_CHANGE_THRESHOLD = 0.5;
    public static final int PHASE_SPEED_BONUS = 2;
    public static final int PHASE_DAMAGE_BONUS = 10;

    public Boss() {
        super();
        this.phase = DEFAULT_PHASE;
        this.size = DEFAULT_SIZE;
        this.damage = DEFAULT_DAMAGE;
        System.out.println("Объект класса Boss создан");
    }

    public Boss(int hp, int spd, int ph, int sz, int dmg) {
        super(hp, spd);
        this.phase = ph;
        this.size = sz;
        this.damage = dmg;
        System.out.println("Объект класса Boss создан с параметрами");
    }

    public Boss(int hp, int spd, int ph, int sz, int dmg, GameManager gm) {
        super(hp, spd, gm);
        this.phase = ph;
        this.size = sz;
        this.damage = dmg;
        System.out.println("Объект класса Boss создан с ссылкой на
GameManager");
    }

    public void move() {
        System.out.println("Метод move() класса Boss выполнен");
        System.out.println("Босс фазы " + phase + " движется к цели");
    }

    public void takeDamage(int amount) {
        health -= amount;
        System.out.println("Метод takeDamage() класса Boss выполнен");
        System.out.println("Получено урона: " + amount + ", осталось
здоровья: " + health);

        if (health < maxHealth * PHASE_CHANGE_THRESHOLD && phase == 1) {
            changePhase();
        }

        if (!isAlive()) {
            System.out.println("Босс побежден!");
        }
    }

    public boolean checkCollision(String target) {
        System.out.println("Метод checkCollision() класса Boss выполнен с
целью: " + target);
        return true;
    }

    public void update() {
        System.out.println("Метод update() класса Boss выполнен");

        move();
```

```
        if (phase > 1) {
            System.out.println("Босс использует специальную атаку фазы " +
phase);
        }
    }

    public void draw() {
        System.out.println("Метод draw() класса Boss выполнен");
        System.out.println("Отрисовка босса размера " + size + " в фазе " +
phase);
    }

    public void applyDamage(int amount) {
        System.out.println("Метод applyDamage() класса Boss выполнен");
        takeDamage(amount);
    }

    public void changePhase() {
        phase++;
        System.out.println("Босс перешел в фазу " + phase + "!");
        System.out.println("Скорость и урон увеличены!");

        speed += PHASE_SPEED_BONUS;
        damage += PHASE_DAMAGE_BONUS;
    }

    public int getPhase() { return phase; }
    public int getSize() { return size; }
    public int getDamage() { return damage; }
}
```

Файлы для сборки и запуска

1. Windows:

build.bat

```
@echo off
echo Сборка Flying Danger Java...
if exist bin rmdir /s /q bin
mkdir bin
javac -d bin src/*.java
if %errorlevel% equ 0 (
    echo Сборка завершена успешно!
    dir bin
) else (
    echo Ошибка сборки!
)
echo.
```

run.bat

```
@echo off
echo Запуск Flying Danger Java...
echo ================================
java -cp bin Main
echo ================================
echo.
pause
```

2. Linux (Ubuntu)

built.sh

```
#!/bin/bash
echo "[Linux] Building Java Application..."

# Создаем выходную директорию если её нет
mkdir -p bin

# Компилируем Java классы
javac -d bin -sourcepath src src/game/*.java

echo "[Linux] Build completed!"
echo
```

run.sh

```
#!/bin/bash
echo "[Linux] Running Java Application..."
echo "================================="
java -cp bin game.Main
echo "================================="
echo
```

Результат работы на Windows:

```
D:\Labs\FlyingDangerJava>run.bat
Starting Flying Danger Java...
=================================
ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ИГРЫ 'FLYING DANGER' НА JAVA

ХРОНОЛОГИЧЕСКАЯ ПОСЛЕДОВАТЕЛЬНОСТЬ ДЕЙСТВИЙ:

1. СОЗДАНИЕ ОСНОВНЫХ ОБЪЕКТОВ:
Объект класса Castle создан с максимальным здоровьем: 100
Объект класса Player создан
Объект класса GameManager создан

2. СОЗДАНИЕ ИГРОВЫХ ОБЪЕКТОВ:
Объект класса Castle создан с максимальным здоровьем: 200
Объект класса Player создан
Объект класса Player создан с ссылкой на GameManager

3. ЗАПУСК ИГРОВОГО ПРОЦЕССА:
Метод run() класса GameManager запущен
```

Результат работы на Linux:

```
name@DESKTOP-08HHUQ3:~$ cd /mnt/d/Labs/FlyingDangerJava
name@DESKTOP-08HHUQ3:/mnt/d/Labs/FlyingDangerJava$ javac -d bin src/*.java
name@DESKTOP-08HHUQ3:/mnt/d/Labs/FlyingDangerJava$ java -cp bin Main
ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ИГРЫ 'FLYING DANGER' НА JAVA

ХРОНОЛОГИЧЕСКАЯ ПОСЛЕДОВАТЕЛЬНОСТЬ ДЕЙСТВИЙ:

1. СОЗДАНИЕ ОСНОВНЫХ ОБЪЕКТОВ:
Объект класса Castle создан с максимальным здоровьем: 100
Объект класса Player создан
Объект класса GameManager создан

2. СОЗДАНИЕ ИГРОВЫХ ОБЪЕКТОВ:
Объект класса Castle создан с максимальным здоровьем: 200
Объект класса Player создан
Объект класса Player создан с ссылкой на GameManager

3. ЗАПУСК ИГРОВОГО ПРОЦЕССА:
Метод run() класса GameManager запущен
```