Потоки и файлы

Лекция 7: Работа с данными

Преподаватель: [Ваше имя]

Группа: 203

Семестр: Осенний 2024

План лекции

1. **Потоки ввода-вывода**
2. **Работа с файлами**
3. **Сериализация объектов**
4. **JSON обработка**
5. **Сохранение/загрузка игры**
6. **Практический пример: Система сохранений**

Потоки ввода-вывода

Что такое потоки?

Поток — это последовательность данных, передаваемых между программой и внешним источником.

Типы потоков:

- **InputStream/OutputStream** — байтовые потоки
- **Reader/Writer** — символьные потоки
- **Buffered** — буферизованные потоки
- **Data** — потоки для примитивных типов

Базовые потоки

Чтение файла:

```
try (FileInputStream fis = new FileInputStream("file.txt")) {
    int data;
    while ((data = fis.read()) != -1) {
        System.out.print((char) data);
    }
} catch (IOException e) {
    e.printStackTrace();
}
```

Запись в файл:

```
try (FileOutputStream fos = new FileOutputStream("output.txt")) {
    String text = "Hello, World!";
    fos.write(text.getBytes());
```

Буферизованные потоки

BufferedReader для чтения:

```
try (BufferedReader reader = new BufferedReader(new FileReader("game.txt"))) {
    String line;
    while ((line = reader.readLine()) != null) {
        System.out.println(line);
    }
} catch (IOException e) {
    e.printStackTrace();
}
```

BufferedWriter для записи:

```
try (BufferedWriter writer = new BufferedWriter(new FileWriter("save.txt"))) {
    writer.write("Игрок: Aragorn");
    writer.newLine();
    writer.write("Уровень: 5");
```

Работа с файлами

Класс File:

```
File file = new File("game/save.txt");

// Проверка существования
if (file.exists()) {
    System.out.println("Файл существует");
}

// Создание директорий
File dir = new File("game/saves");
if (!dir.exists()) {
    dir.mkdirs();
}

// Получение информации
System.out.println("Размер: " + file.length() + " байт");
System.out.println("Последнее изменение: " + new Date(file.lastModified()));
```

Сканирование директорий

```
public class FileManager {
    public static void listGameFiles(String directory) {
        File dir = new File(directory);
        if (!dir.isDirectory()) {
            System.out.println("Не является директорией");
            return;
        }

        File[] files = dir.listFiles();
        if (files != null) {
            for (File file : files) {
                if (file.isFile()) {
                    System.out.println("Файл: " + file.getName());
                } else if (file.isDirectory()) {
                    System.out.println("Директория: " + file.getName());
                }
            }
        }
    }

    public static void findSaveFiles(String directory) {
        File dir = new File(directory);
        File[] files = dir.listFiles((d, name) -> name.endsWith(".save"));

        if (files != null) {
            for (File file : files) {
                System.out.println("Сохранение: " + file.getName());
            }
        }
    }
}
```

Сериализация объектов

Что такое сериализация?

Сериализация — процесс преобразования объекта в последовательность байтов для сохранения или передачи.

Требования:

- Класс должен реализовывать `Serializable`
- Все поля должны быть сериализуемыми
- Использовать `transient` для несериализуемых полей

Реализация Serializable

```
public class GameState implements Serializable {
    private static final long serialVersionUID = 1L;

    private String playerName;
    private int level;
    private int experience;
    private List<Unit> units;
    private transient GameEngine engine; // Не сериализуется

    // Конструкторы, геттеры, сеттеры...

    public void saveToFile(String filename) {
        try (ObjectOutputStream oos = new ObjectOutputStream(
                new FileOutputStream(filename))) {
            oos.writeObject(this);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static GameState loadFromFile(String filename) {
        try (ObjectInputStream ois = new ObjectInputStream(
                new FileInputStream(filename))) {
            return (GameState) ois.readObject();
        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
            return null;
        }
    }
}
```

JSON обработка

Что такое JSON?

JSON (JavaScript Object Notation) — текстовый формат для обмена данными.

Преимущества:

- **Читаемость** для человека
- **Кроссплатформенность**
- **Поддержка** во многих языках
- **Легкость** парсинга

Jackson библиотека

Добавление зависимости:

```
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.15.0</version>
</dependency>
```

Сериализация в JSON:

```
ObjectMapper mapper = new ObjectMapper();

// Объект в JSON
GameState gameState = new GameState("Player1", 5, 1000);
String json = mapper.writeValueAsString(gameState);

// JSON в файл
```

Десериализация из JSON

```
ObjectMapper mapper = new ObjectMapper();

// JSON из файла в объект
GameState loadedState = mapper.readValue(new File("save.json"), GameState.class);

// JSON из строки в объект
String jsonString = "{\"playerName\":\"Player1\",\"level\":5,\"experience\":1000}";
GameState state = mapper.readValue(jsonString, GameState.class);

// Обработка ошибок
try {
    GameState state = mapper.readValue(jsonString, GameState.class);
} catch (JsonProcessingException e) {
    System.err.println("Ошибка парсинга JSON: " + e.getMessage());
}
```

Аннотации Jackson

```
public class Unit implements Serializable {
    @JsonProperty("unit_name")
    private String name;

    @JsonProperty("unit_health")
    private int health;

    @JsonIgnore
    private transient Position position; // Не включается в JSON

    @JsonFormat(shape = JsonFormat.Shape.STRING)
    private UnitType type;

    // Конструкторы, геттеры, сеттеры...
}

public enum UnitType {
    @JsonProperty("warrior")
    WARRIOR,

    @JsonProperty("archer")
    ARCHER,

    @JsonProperty("mage")
    MAGE
}
```

Система сохранений игры

```
public class SaveGameManager {
    private static final String SAVE_DIR = "game/saves/";
    private static final String SAVE_EXTENSION = ".save";

    public void saveGame(GameState gameState, String saveName) {
        try {
            // Создание директории если не существует
            File saveDir = new File(SAVE_DIR);
            if (!saveDir.exists()) {
                saveDir.mkdirs();
            }

            // Сохранение в JSON
            String jsonFilename = SAVE_DIR + saveName + ".json";
            ObjectMapper mapper = new ObjectMapper();
            mapper.writerWithDefaultPrettyPrinter()
                .writeValue(new File(jsonFilename), gameState);

            // Сохранение в бинарном формате
            String binFilename = SAVE_DIR + saveName + SAVE_EXTENSION;
            try (ObjectOutputStream oos = new ObjectOutputStream(
                    new FileOutputStream(binFilename))) {
                oos.writeObject(gameState);
            }

            System.out.println("Игра сохранена: " + saveName);

        } catch (IOException e) {
            System.err.println("Ошибка сохранения: " + e.getMessage());
        }
    }
}
```

Загрузка сохранений

```
public class LoadGameManager {
    public GameState loadGame(String saveName) {
        try {
            // Попытка загрузить из JSON
            String jsonFilename = SAVE_DIR + saveName + ".json";
            File jsonFile = new File(jsonFilename);

            if (jsonFile.exists()) {
                ObjectMapper mapper = new ObjectMapper();
                return mapper.readValue(jsonFile, GameState.class);
            }

            // Попытка загрузить из бинарного файла
            String binFilename = SAVE_DIR + saveName + SAVE_EXTENSION;
            File binFile = new File(binFilename);

            if (binFile.exists()) {
                try (ObjectInputStream ois = new ObjectInputStream(
                        new FileInputStream(binFile))) {
                    return (GameState) ois.readObject();
                }
            }

            System.err.println("Сохранение не найдено: " + saveName);
            return null;

        } catch (IOException | ClassNotFoundException e) {
            System.err.println("Ошибка загрузки: " + e.getMessage());
            return null;
        }
    }
}
```

Автосохранение

```
public class AutoSaveManager {
    private static final int AUTO_SAVE_INTERVAL = 5; // каждые 5 ходов
    private int moveCount = 0;
    private GameState currentState;

    public void onMoveCompleted() {
        moveCount++;
        if (moveCount % AUTO_SAVE_INTERVAL == 0) {
            autoSave();
        }
    }

    private void autoSave() {
        try {
            String autoSaveName = "autosave_" + System.currentTimeMillis();
            SaveGameManager saveManager = new SaveGameManager();
            saveManager.saveGame(currentState, autoSaveName);

            // Удаление старых автосохранений
            cleanupOldAutoSaves();

        } catch (Exception e) {
            System.err.println("Ошибка автосохранения: " + e.getMessage());
        }
    }

    private void cleanupOldAutoSaves() {
        File saveDir = new File(SAVE_DIR);
        File[] autoSaves = saveDir.listFiles((dir, name) ->
            name.startsWith("autosave_"));

        if (autoSaves != null && autoSaves.length > 10) {
            // Оставляем только 10 последних автосохранений
            Arrays.sort(autoSaves, Comparator.comparingLong(File::lastModified));
            for (int i = 0; i < autoSaves.length - 10; i++) {
                autoSaves[i].delete();
            }
        }
    }
}
```

Обработка ошибок файлов

```
public class FileErrorHandler {
    public static void handleFileError(String operation, String filename, Exception e) {
        String errorMessage = String.format("Ошибка %s файла %s: %s",
            operation, filename, e.getMessage());

        // Логирование ошибки
        Logger logger = Logger.getLogger(FileErrorHandler.class.getName());
        logger.log(Level.SEVERE, errorMessage, e);

        // Показать пользователю
        showErrorMessage(errorMessage);

        // Попытка восстановления
        if (operation.equals("чтения")) {
            attemptRecovery(filename);
        }
    }

    private static void attemptRecovery(String filename) {
        // Попытка загрузить резервную копию
        String backupName = filename + ".backup";
        if (new File(backupName).exists()) {
            System.out.println("Попытка восстановления из резервной копии...");
            // Логика восстановления
        }
    }
}
```

Лучшие практики работы с файлами

✅ Что делать:

- **Использовать try-with-resources** для автоматического закрытия
- **Проверять существование** файлов перед операциями
- **Обрабатывать исключения** корректно
- **Использовать буферизацию** для больших файлов
- **Создавать резервные копии** важных данных

❌ Чего избегать:

- **Забывать закрывать** потоки
- **Игнорировать** исключения
- **Использовать абсолютные пути** без проверки

Домашнее задание

Задача 1:

Создать систему сохранения/загрузки игрового состояния

Задача 2:

Реализовать автосохранение каждые N ходов

Задача 3:

Создать менеджер сохранений с возможностью просмотра списка

Что дальше?

На следующей лекции:

- **GUI с JavaFX**
- **Создание интерфейса**
- **Обработка событий**
- **Анимации**

Подготовка:

- Изучить главу 13-14 из учебника
- Выполнить домашнее задание
- Подготовить вопросы по текущей теме

Вопросы?

Контакты:

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

Следующая лекция: GUI с JavaFX