Исключения и обработка ошибок

Лекция 6: Надежность программ

Преподаватель: [Ваше имя]

Группа: 203

Семестр: Осенний 2024

План лекции

1. **Что такое исключения?**
2. **Иерархия исключений**
3. **Try-catch блоки**
4. **Checked vs Unchecked исключения**
5. **Создание собственных исключений**
6. **Логирование**
7. **Практический пример: Обработка ошибок в игре**

Что такое исключения?

Определение:

Исключение — это событие, которое происходит во время выполнения программы и нарушает нормальный ход выполнения.

Причины исключений:

- **Ошибки программиста** (null pointer, array bounds)
- **Ошибки пользователя** (неверный ввод)
- **Системные ошибки** (файл не найден, сеть недоступна)
- **Физические ограничения** (недостаточно памяти)

Иерархия исключений

```
Throwable
├── Error (необрабатываемые)
│   ├── OutOfMemoryError
│   ├── StackOverflowError
│   └── VirtualMachineError
└── Exception (обрабатываемые)
    ├── RuntimeException (unchecked)
    │   ├── NullPointerException
    │   ├── ArrayIndexOutOfBoundsException
    │   └── IllegalArgumentException
    └── IOException (checked)
        ├── FileNotFoundException
        └── EOFException
```

Try-catch блоки

Базовый синтаксис:

```
try {
    // Код, который может вызвать исключение
    riskyOperation();
} catch (ExceptionType e) {
    // Обработка исключения
    System.err.println("Ошибка: " + e.getMessage());
} finally {
    // Код, который выполняется всегда
    cleanup();
}
```

Пример:

```
try {
```

Множественные catch блоки

```
try {
    // Код, который может вызвать разные исключения
    File file = new File("nonexistent.txt");
    FileReader reader = new FileReader(file);
    int data = reader.read();
} catch (FileNotFoundException e) {
    System.err.println("Файл не найден: " + e.getMessage());
} catch (IOException e) {
    System.err.println("Ошибка ввода-вывода: " + e.getMessage());
} catch (Exception e) {
    System.err.println("Неизвестная ошибка: " + e.getMessage());
} finally {
    System.out.println("Попытка чтения завершена");
}
```

Try-with-resources (Java 7+)

Автоматическое закрытие ресурсов:

```
// Старый способ
FileReader reader = null;
try {
    reader = new FileReader("file.txt");
    // работа с файлом
} catch (IOException e) {
    e.printStackTrace();
} finally {
    if (reader != null) {
        try {
            reader.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

// Новый способ (try-with-resources)
try (FileReader reader = new FileReader("file.txt")) {
    // работа с файлом
    // файл автоматически закроется
```

Checked vs Unchecked исключения

Checked исключения (проверяемые):

- **Должны быть обработаны** или объявлены в throws
- **Наследуются от Exception** (но не от RuntimeException)
- **Компилятор проверяет** их обработку

Unchecked исключения (непроверяемые):

- **Не обязательны к обработке**
- **Наследуются от RuntimeException**
- **Компилятор не проверяет** их обработку

Примеры Checked исключений

```
public class FileManager {
    // Должны объявить throws или обработать исключение
    public String readFile(String filename) throws IOException {
        try (BufferedReader reader = new BufferedReader(new FileReader(filename))) {
            StringBuilder content = new StringBuilder();
            String line;
            while ((line = reader.readLine()) != null) {
                content.append(line).append("\n");
            }
            return content.toString();
        }
    }

    // Или обработать исключение внутри метода
    public String readFileSafe(String filename) {
        try {
            return readFile(filename);
        } catch (IOException e) {
            System.err.println("Ошибка чтения файла: " + e.getMessage());
            return "";
        }
    }
}
```

Примеры Unchecked исключений

```
public class GameLogic {
    public void moveUnit(Unit unit, Position newPosition) {
        // NullPointerException - unchecked
        if (unit == null) {
            throw new IllegalArgumentException("Юнит не может быть null");
        }

        // IllegalArgumentException - unchecked
        if (newPosition == null) {
            throw new IllegalArgumentException("Позиция не может быть null");
        }

        // ArrayIndexOutOfBoundsException - unchecked
        if (newPosition.getX() < 0 || newPosition.getY() < 0) {
            throw new IllegalArgumentException("Позиция вне игрового поля");
        }

        // Выполнение движения
        unit.setPosition(newPosition);
    }
}
```

Создание собственных исключений

Простое исключение:

```
public class GameException extends Exception {
    public GameException(String message) {
        super(message);
    }

    public GameException(String message, Throwable cause) {
        super(message, cause);
    }
}
```

Специализированные исключения:

```
public class InvalidMoveException extends GameException {
    private Position from;
    private Position to;
```

Использование собственных исключений

```
public class GameEngine {
    public void moveUnit(Unit unit, Position newPosition) throws InvalidMoveException {
        try {
            // Проверка возможности движения
            if (!unit.canMoveTo(newPosition)) {
                throw new InvalidMoveException(
                    "Юнит не может двигаться на позицию " + newPosition,
                    unit.getPosition(),
                    newPosition
                );
            }

            // Выполнение движения
            unit.setPosition(newPosition);

        } catch (Exception e) {
            // Логирование ошибки
            Logger.getLogger(GameEngine.class.getName())
                .log(Level.SEVERE, "Ошибка движения юнита", e);

            // Перебрасывание как InvalidMoveException
            throw new InvalidMoveException(
                "Ошибка при движении юнита: " + e.getMessage(),
                unit.getPosition(),
                newPosition
            );
        }
    }
}
```

Логирование в Java

Что такое логирование?

Логирование — процесс записи информации о работе программы для отладки и мониторинга.

Уровни логирования:

- **SEVERE** — критические ошибки
- **WARNING** — предупреждения
- **INFO** — информационные сообщения
- **CONFIG** — конфигурация
- **FINE** — детальная отладочная информация
- **FINER** — более детальная информация

Настройка логирования

```
import java.util.logging.*;

public class GameLogger {
    private static final Logger logger = Logger.getLogger(GameLogger.class.getName());

    static {
        // Настройка уровня логирования
        logger.setLevel(Level.ALL);

        // Создание обработчика для файла
        try {
            FileHandler fileHandler = new FileHandler("game.log", true);
            fileHandler.setFormatter(new SimpleFormatter());
            logger.addHandler(fileHandler);
        } catch (IOException e) {
            e.printStackTrace();
        }

        // Создание обработчика для консоли
        ConsoleHandler consoleHandler = new ConsoleHandler();
        consoleHandler.setLevel(Level.INFO);
        logger.addHandler(consoleHandler);
    }

    public static void logGameEvent(String event) {
        logger.info("Игровое событие: " + event);
    }

    public static void logError(String error, Throwable e) {
        logger.log(Level.SEVERE, "Ошибка: " + error, e);
    }
}
```

Практический пример: Обработка ошибок в игре

```
public class GameManager {
    private static final Logger logger = Logger.getLogger(GameManager.class.getName());

    public void startGame(String player1Name, String player2Name) {
        try {
            logger.info("Начинаем новую игру: " + player1Name + " vs " + player2Name);

            // Создание игроков
            Player player1 = new Player(player1Name);
            Player player2 = new Player(player2Name);

            // Инициализация игрового поля
            GameBoard board = new GameBoard(10, 10);

            // Размещение начальных юнитов
            placeInitialUnits(board, player1, player2);

            logger.info("Игра успешно инициализирована");

        } catch (GameException e) {
            logger.log(Level.SEVERE, "Ошибка инициализации игры", e);
            throw new RuntimeException("Не удалось запустить игру", e);
        }
    }

    private void placeInitialUnits(GameBoard board, Player p1, Player p2)
            throws InvalidPositionException {
        try {
            // Размещение юнитов игрока 1
            board.placeUnit(new Warrior("Warrior1", 150), new Position(0, 0));
            board.placeUnit(new Archer("Archer1", 100), new Position(1, 0));

            // Размещение юнитов игрока 2
            board.placeUnit(new Warrior("Warrior2", 150), new Position(9, 9));
            board.placeUnit(new Mage("Mage2", 80), new Position(8, 9));

        } catch (InvalidPositionException e) {
            logger.log(Level.WARNING, "Ошибка размещения юнитов: " + e.getMessage());
            throw e;
        }
    }
}
```

Обработка исключений в UI

```
public class GameController {
    private static final Logger logger = Logger.getLogger(GameController.class.getName());

    public void handlePlayerMove(Position from, Position to) {
        try {
            // Выполнение хода
            gameEngine.moveUnit(selectedUnit, to);

            // Обновление UI
            updateGameDisplay();

            // Проверка победы
            if (gameEngine.isGameOver()) {
                showGameOverDialog();
            }

        } catch (InvalidMoveException e) {
            // Показать пользователю понятное сообщение
            showErrorMessage("Невозможно выполнить ход: " + e.getMessage());

            // Логирование для разработчиков
            logger.log(Level.WARNING, "Неверный ход игрока", e);

        } catch (GameException e) {
            // Общая ошибка игры
            showErrorMessage("Ошибка игры: " + e.getMessage());
            logger.log(Level.SEVERE, "Критическая ошибка игры", e);

        } catch (Exception e) {
            // Неожиданная ошибка
            showErrorMessage("Произошла неожиданная ошибка");
            logger.log(Level.SEVERE, "Неожиданная ошибка", e);
        }
    }
}
```

Лучшие практики обработки исключений

✅ Что делать:

- **Обрабатывать исключения** на соответствующем уровне
- **Логировать** все ошибки с контекстом
- **Предоставлять пользователю** понятные сообщения
- **Использовать try-with-resources** для ресурсов
- **Создавать специфичные** исключения

❌ Чего избегать:

- **Игнорировать** исключения (пустые catch блоки)
- **Логировать и перебрасывать** без изменений
- **Показывать пользователю** технические детали

Домашнее задание

Задача 1:

Создать иерархию исключений для игровых ошибок

Задача 2:

Реализовать логирование всех действий в игре

Задача 3:

Создать систему восстановления после ошибок

Что дальше?

На следующей лекции:

- **Потоки и файлы**
- **Работа с файлами**
- **Сериализация**
- **JSON обработка**

Подготовка:

- Изучить главу 11-12 из учебника
- Выполнить домашнее задание
- Подготовить вопросы по текущей теме

Вопросы?

Контакты:

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

Следующая лекция: Потоки и файлы