Коллекции и generics

Лекция 5: Работа с данными в Java

Преподаватель: [Ваше имя]

Группа: 203

Семестр: Осенний 2024

План лекции

1. **Коллекции в Java**
2. **List интерфейс**
3. **Set интерфейс**
4. **Map интерфейс**
5. **Generics**
6. **Stream API**
7. **Практический пример: Игровые коллекции**

Коллекции в Java

Что такое коллекции?

Коллекции — это структуры данных для хранения и управления группами объектов.

Иерархия коллекций:

```
Collection<E>
├── List<E>          (списки)
├── Set<E>           (множества)
└── Queue<E>         (очереди)

Map<K,V>             (отображения)
```

Основные операции:

- **Добавление** элементов

List интерфейс

Особенности:

- **Упорядоченная** последовательность элементов
- **Дублирующиеся** элементы разрешены
- **Индексированный** доступ к элементам
- **Динамический** размер

Основные реализации:

- `ArrayList<E>` — массив с динамическим размером
- `LinkedList<E>` — связанный список
- `Vector<E>` — потокобезопасный список

ArrayList vs LinkedList

ArrayList:

```
ArrayList<Unit> units = new ArrayList<>();
units.add(new Warrior("Aragorn", 150));
units.add(new Archer("Legolas", 100));
units.add(new Mage("Gandalf", 80));

// Быстрый доступ по индексу
Unit firstUnit = units.get(0);
Unit lastUnit = units.get(units.size() - 1);

// Добавление в конец
units.add(new Warrior("Boromir", 120));

// Удаление по индексу
units.remove(1);
```

Set интерфейс

Особенности:

- **Уникальные** элементы
- **Неупорядоченные** (в большинстве реализаций)
- **Быстрый поиск** элементов

Основные реализации:

- `HashSet<E>` — хеш-таблица
- `TreeSet<E>` — сбалансированное дерево
- `LinkedHashSet<E>` — хеш-таблица с сохранением порядка

HashSet и TreeSet

HashSet:

```
HashSet<String> unitTypes = new HashSet<>();
unitTypes.add("Warrior");
unitTypes.add("Archer");
unitTypes.add("Mage");
unitTypes.add("Warrior"); // Дубликат не добавится

// Быстрый поиск
boolean hasWarrior = unitTypes.contains("Warrior");

// Итерация (порядок не гарантирован)
for (String type : unitTypes) {
    System.out.println(type);
}
```

TreeSet:

Map интерфейс

Особенности:

- **Пары ключ-значение**
- **Уникальные ключи**
- **Быстрый поиск по ключу**

Основные реализации:

- `HashMap<K,V>` — хеш-таблица
- `TreeMap<K,V>` — сбалансированное дерево
- `LinkedHashMap<K,V>` — хеш-таблица с сохранением порядка

HashMap примеры

```
HashMap<String, Unit> unitRegistry = new HashMap<>();

// Добавление элементов
unitRegistry.put("Aragorn", new Warrior("Aragorn", 150));
unitRegistry.put("Legolas", new Archer("Legolas", 100));
unitRegistry.put("Gandalf", new Mage("Gandalf", 80));

// Получение элемента
Unit aragorn = unitRegistry.get("Aragorn");

// Проверка существования
if (unitRegistry.containsKey("Gimli")) {
    Unit gimli = unitRegistry.get("Gimli");
}

// Итерация по ключам
for (String name : unitRegistry.keySet()) {
    System.out.println(name + ": " + unitRegistry.get(name));
}

// Итерация по парам
for (Map.Entry<String, Unit> entry : unitRegistry.entrySet()) {
    System.out.println(entry.getKey() + " -> " + entry.getValue());
}
```

Generics в Java

Что такое generics?

Generics — механизм для создания типизированных классов, интерфейсов и методов.

Преимущества:

- **Типобезопасность** на этапе компиляции
- **Устранение** приведения типов
- **Переиспользование** кода для разных типов

Синтаксис generics

Объявление generic класса:

```
public class Container<T> {
    private T item;

    public void setItem(T item) {
        this.item = item;
    }

    public T getItem() {
        return item;
    }
}

// Использование
Container<String> stringContainer = new Container<>();
Container<Integer> intContainer = new Container<>();
```

Ограничения generics

Ограничение типа:

```
// T должен быть подклассом Number
public class NumberContainer<T extends Number> {
    private T number;

    public double getDoubleValue() {
        return number.doubleValue();
    }
}

// Использование
NumberContainer<Integer> intContainer = new NumberContainer<>();
NumberContainer<Double> doubleContainer = new NumberContainer<>();
// NumberContainer<String> stringContainer = new NumberContainer<>(); // Ошибка!
```

Множественные ограничения:

Wildcards в generics

Неограниченный wildcard:

```
// Список любого типа
public void printList(List<?> list) {
    for (Object item : list) {
        System.out.println(item);
    }
}

// Использование
List<String> strings = Arrays.asList("a", "b", "c");
List<Integer> numbers = Arrays.asList(1, 2, 3);
printList(strings);
printList(numbers);
```

Ограниченный wildcard:

Stream API (Java 8+)

Что такое Stream?

Stream — последовательность элементов, поддерживающая различные операции.

Основные операции:

- **Промежуточные** (filter, map, sorted)
- **Терминальные** (collect, forEach, reduce)

Примеры Stream API

Фильтрация и преобразование:

```
List<Unit> units = Arrays.asList(
    new Warrior("Aragorn", 150),
    new Archer("Legolas", 100),
    new Mage("Gandalf", 80),
    new Warrior("Boromir", 120)
);

// Фильтрация живых юнитов
List<Unit> aliveUnits = units.stream()
    .filter(unit -> unit.isAlive())
    .collect(Collectors.toList());

// Получение имен всех юнитов
List<String> unitNames = units.stream()
    .map(Unit::getName)
    .collect(Collectors.toList());

// Подсчет воинов
```

Продвинутые операции Stream

Группировка и статистика:

```
// Группировка по типу юнита
Map<Class<?>, List<Unit>> unitsByType = units.stream()
    .collect(Collectors.groupingBy(Unit::getClass));

// Статистика по здоровью
DoubleSummaryStatistics healthStats = units.stream()
    .mapToDouble(Unit::getHealth)
    .summaryStatistics();

System.out.println("Среднее здоровье: " + healthStats.getAverage());
System.out.println("Максимальное здоровье: " + healthStats.getMax());

// Поиск самого здорового юнита
Optional<Unit> healthiestUnit = units.stream()
    .max(Comparator.comparing(Unit::getHealth));
```

Практический пример: Игровые коллекции

```
public class GameWorld {
    private Map<Position, Unit> unitPositions = new HashMap<>();
    private List<Building> buildings = new ArrayList<>();
    private Set<String> playerNames = new HashSet<>();
    private Queue<GameEvent> eventQueue = new LinkedList<>();

    // Добавление юнита
    public void addUnit(Unit unit, Position position) {
        unitPositions.put(position, unit);
        playerNames.add(unit.getPlayerName());
    }

    // Поиск юнитов в радиусе
    public List<Unit> getUnitsInRadius(Position center, int radius) {
        return unitPositions.entrySet().stream()
            .filter(entry -> entry.getKey().getDistanceTo(center) <= radius)
            .map(Map.Entry::getValue)
            .collect(Collectors.toList());
    }

    // Получение всех юнитов игрока
    public List<Unit> getPlayerUnits(String playerName) {
        return unitPositions.values().stream()
            .filter(unit -> unit.getPlayerName().equals(playerName))
            .collect(Collectors.toList());
    }
}
```

Сравнение производительности коллекций

Время доступа:

Операция	ArrayList	LinkedList	HashSet	HashMap
Получение по индексу	O(1)	O(n)	-	-
Поиск элемента	O(n)	O(n)	O(1)	O(1)
Добавление в конец	O(1)	O(1)	O(1)	O(1)
Удаление элемента	O(n)	O(1)	O(1)	O(1)

Выбор коллекции:

- **ArrayList** — частый доступ по индексу
- **LinkedList** — частые вставки/удаления

Домашнее задание

Задача 1:

Создать класс `UnitManager` с коллекциями для управления юнитами

Задача 2:

Реализовать методы поиска и фильтрации юнитов с использованием Stream API

Задача 3:

Создать generic класс `GameContainer<T>` для хранения игровых объектов

Что дальше?

На следующей лекции:

- **Исключения и обработка ошибок**
- **Try-catch блоки**
- **Checked и unchecked исключения**
- **Логирование**

Подготовка:

- Изучить главу 9-10 из учебника
- Выполнить домашнее задание
- Подготовить вопросы по текущей теме

Вопросы?

Контакты:

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

Следующая лекция: Исключения и обработка ошибок