Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №4

З дисципліни: Технології розроблення програмного забезпечення

Тема: “Паттерн Ітеаратор”

Виконав:

студент групи ІА-14

Рисаков Богдан

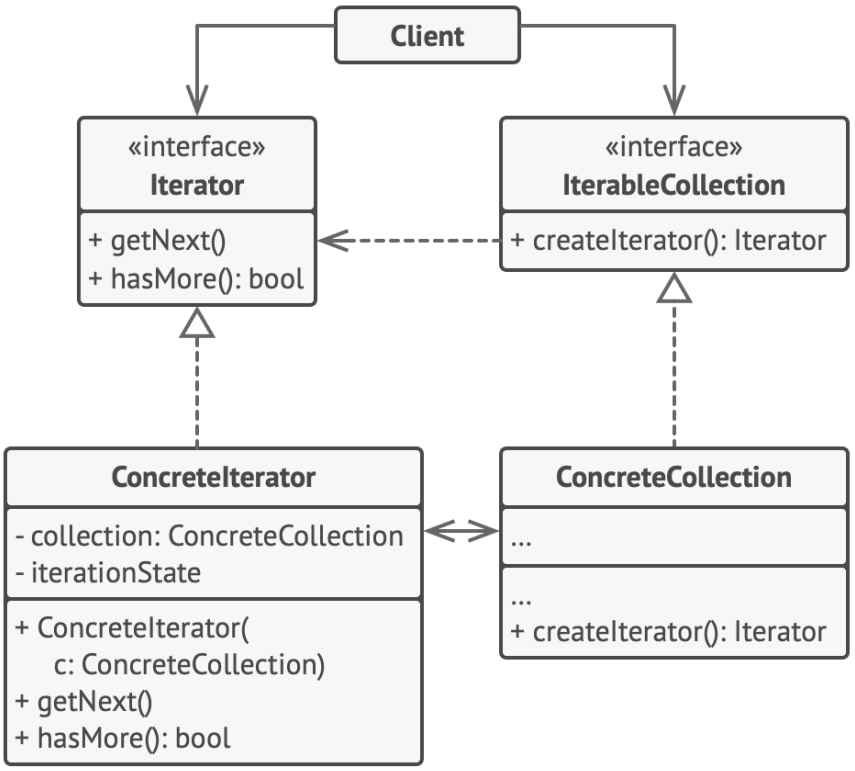
Перевірив:

Мягкий Михайло Юрійович

Київ, 2023

Мета: Реалізувати паттерн Iterator у проекті на тему System Activity Monitor

Ітератор — це поведінковий патерн проектування, що дає змогу послідовно обходити елементи складових об'єктів, не розкриваючи їхньої внутрішньої організації.



Приклад структури реалізації та використання паттерну. Reference

Переваги:

- Спрощує класи зберігання даних.
- Дозволяє реалізувати різні способи обходу структури даних.
- Дозволяє одночасно переміщуватися структурою даних у різних напрямках.

Недоліки:

- Невиправданий, якщо можна обійтися простим циклом.

Для початку обозначим по колекцію чого будемо проходитись:

```
public abstract class Report {

   protected String title;

   protected LocalDateTime timestamp;

   protected Duration duration;



   protected List<DataPoint> data;

   protected String summary;

   protected ReportType reportType;



   public Report(String title, Duration duration, ReportType reportType) {

       this.title = title;

       this.timestamp = LocalDateTime.now();

       this.duration = duration;

       this.data = new ArrayList<>();

       this.reportType = reportType;

   }

}
```

Це відповідно Батьківський клас для усіх звітів що ми будемо робити

Визначимо цілі, для яких ми використовуємо ітератори та коротко опишемо їх:

- DataPointCountIterator: Ітерує через список звітів, повертаючи лише ті звіти, кількість даних у яких перевищує задане значення minDataPointCount.
- DateIterator: Ітерує через список звітів, вибираючи лише ті звіти, дата яких знаходиться у вказаному діапазоні між lastDate і firstDate.
- DurationIterator: Ітерує через список звітів, видаючи лише ті звіти, тривалість яких є меншою або більшою за задану тривалість (за допомогою об'єкта java.time.Duration), в залежності від заданого параметра isDurationLower.

Кожен із цих ітераторів виконує унікальну роль у системі збору та аналізу даних, дозволяючи користувачам ефективно сортувати та вибирати звіти відповідно до їхніх специфічних потреб.

Опишем інтерфейс для ітераторів:

```
public interface ReportIterator {

   boolean hasNext();

   Report getNext();

}
```

Напишемо самі ітератори:
1)

```
public class DataPointCountIterator {

   private final List<Report> reports;

   private int position;

   private final int minDataPointCount;


   public DataPointCountIterator(List<Report> reports, int minDataPointCount) {

       this.reports = reports;

       this.position = 0;

       this.minDataPointCount = minDataPointCount;

   }


   public boolean hasNext() {

       while (position < reports.size()) {

           Report currentReport = reports.get(position);

           if (currentReport.getData().size() > minDataPointCount) {

               return true;

           }

           position++;

       }

       return false;

   }


   public Report getNext() {
```

```
        if (!hasNext()) {

            return null;

        }

        return reports.get(position++);

    }

}
```

2)

```
public class DateIterator implements ReportIterator {

    private final List<Report> reports;

    private int position;

    private final LocalDateTime lastDate;

    private final LocalDateTime firstDate;



    public DateIterator(List<Report> reports, LocalDateTime lastDate, LocalDateTime
firstDate) {

        this.reports = reports;

        this.position = 0;

        this.lastDate = lastDate;

        this.firstDate = firstDate;

    }



    @Override

    public boolean hasNext() {

        while (position < reports.size()) {

            Report currentReport = reports.get(position);

            LocalDateTime reportDate = currentReport.getTimestamp();

            if (reportDate != null && !reportDate.isBefore(lastDate) &&
!reportDate.isAfter(firstDate)) {

                return true;

            }

            position++;
```

```
        }
        return false;
    }

    @Override
    public Report getNext() {
        if (!hasNext()) {
            return null;
        }
        return reports.get(position++);
    }
}
```

3)

```
public class DurationIterator implements ReportIterator {
    private final List<Report> reports;
    private int position;
    private final Duration targetDuration;
    private final boolean isDurationLower;

    public DurationIterator(List<Report> reports, Duration targetDuration, boolean
isDurationLower) {
        this.reports = reports;
        this.position = 0;
        this.targetDuration = targetDuration;
        this.isDurationLower = isDurationLower;
    }

    @Override
```

```
    public boolean hasNext() {

        while (position < reports.size()) {

            Report currentReport = reports.get(position);

            Duration reportDuration = currentReport.getDuration();

            if (reportDuration != null &&

                    (isDurationLower ? reportDuration.compareTo(targetDuration) < 0
: reportDuration.compareTo(targetDuration) > 0)) {

                return true;

            }

            position++;

        }

        return false;

    }



    @Override

    public Report getNext() {

        if (!hasNext()) {

            return null;

        }

        return reports.get(position++);

    }

}
```

(Додатково надам скріни з середовища розробки)

Висновок: Я реалізував паттерн Ітератор

Додаткові скріни:

```java
no usages
public class DateIterator implements ReportIterator {
    4 usages
    private final List<Report> reports;
    5 usages
    private int position;
    2 usages
    private final LocalDateTime lastDate;
    2 usages
    private final LocalDateTime firstDate;

    no usages
    public DateIterator(List<Report> reports, LocalDateTime lastDate, LocalDateTime firstDate) {
        this.reports = reports;
        this.position = 0;
        this.lastDate = lastDate;
        this.firstDate = firstDate;
    }

    2 usages
    @Override
    public boolean hasNext() {
        while (position < reports.size()) {
            Report currentReport = reports.get(position);
            LocalDateTime reportDate = currentReport.getTimestamp();
            if (reportDate != null && !reportDate.isBefore(lastDate) && !reportDate.isAfter(firstDate)) {
                return true;
            }
            position++;
        }
        return false;
    }

    no usages
    @Override
    public Report getNext() {
        if (!hasNext()) {
            return null;
        }
        return reports.get(position++);
    }
}
```

```java
no usages
public class DateIterator implements ReportIterator {   You, 2 minutes ago • Uncommitted changes
    4 usages
    private final List<Report> reports;
    5 usages
    private int position;
    2 usages
    private final LocalDateTime lastDate;
    2 usages
    private final LocalDateTime firstDate;

    no usages
    public DateIterator(List<Report> reports, LocalDateTime lastDate, LocalDateTime firstDate) {
        this.reports = reports;
        this.position = 0;
        this.lastDate = lastDate;
        this.firstDate = firstDate;
    }

    2 usages
    @Override
    public boolean hasNext() {
        while (position < reports.size()) {
            Report currentReport = reports.get(position);
            LocalDateTime reportDate = currentReport.getTimestamp();
            if (reportDate != null && !reportDate.isBefore(lastDate) && !reportDate.isAfter(firstDate)) {
                return true;
            }
            position++;
        }
        return false;
    }

    no usages
    @Override
    public Report getNext() {
        if (!hasNext()) {
            return null;
        }
        return reports.get(position++);
    }
}
```

```java
public class DurationIterator implements ReportIterator {
    4 usages
    private final List<Report> reports;
    5 usages
    private int position;
    3 usages
    private final Duration targetDuration;
    2 usages
    private final boolean isDurationLower;

    no usages
    public DurationIterator(List<Report> reports, Duration targetDuration, boolean isDurationLower) {
        this.reports = reports;
        this.position = 0;
        this.targetDuration = targetDuration;
        this.isDurationLower = isDurationLower;
    }   ajax-rysakov-b, Today • add iterator

    2 usages
    @Override
    public boolean hasNext() {
        while (position < reports.size()) {
            Report currentReport = reports.get(position);
            Duration reportDuration = currentReport.getDuration();
            if (reportDuration != null &&
                    (isDurationLower ? reportDuration.compareTo(targetDuration) < 0 :
                            reportDuration.compareTo(targetDuration) > 0)) {
                return true;
            }
            position++;
        }
        return false;
    }

    no usages
    @Override
    public Report getNext() {
        if (!hasNext()) {
            return null;
        }
        return reports.get(position++);
    }
}
```