Поведенческие паттерны в Java: Iterator и Observer

Дрекалов Никита & Соколов Артем, 5030102/20202

План презентации

Iterator

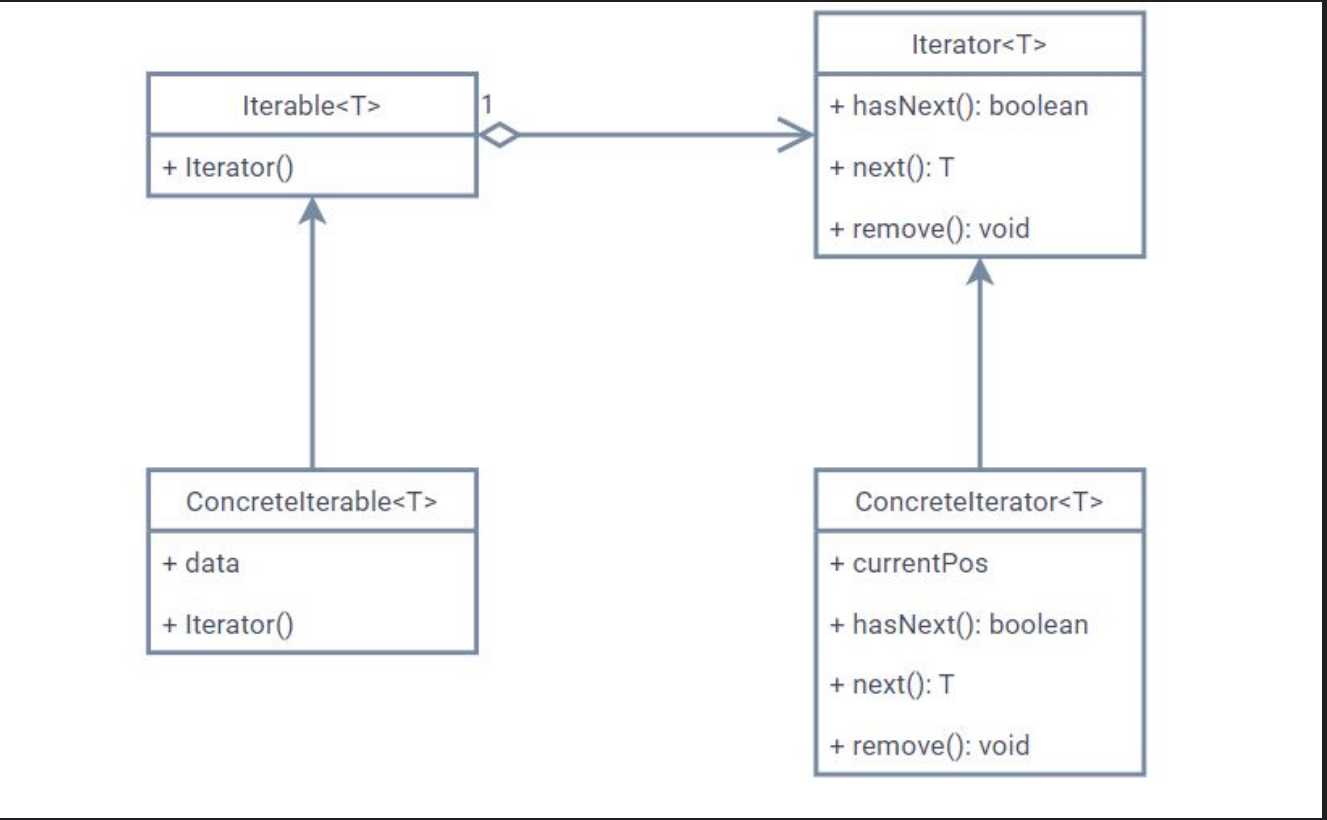
1. Описание и структура
2. Интерфейсы Iterator и Observable
3. Диаграмма последовательности и способы реализации
4. Пример использования паттерна
5. ListIterator
6. Spliterator


РАЗГОВОРЫ
С ИТЕРАТОРАМИ

Iterator

- Основные принципы:
    - Разделение ответственности
    - Инкапсуляция внутренней структуры
    - Работа с коллекциями в унифицированном интерфейсе

Структура


Iterable<T>
+ Iterator()
1
Iterator<T>
+ hasNext(): boolean
+ next(): T
+ remove(): void
ConcreteIterable<T>
+ data
+ Iterator()
ConcreteIterator<T>
+ currentPos
+ hasNext(): boolean
+ next(): T
+ remove(): void

Интерфейсы Iterator и Iterable

```
public interface Iterator<E> {

   E next();
   boolean hasNext();
   void remove();
}
```

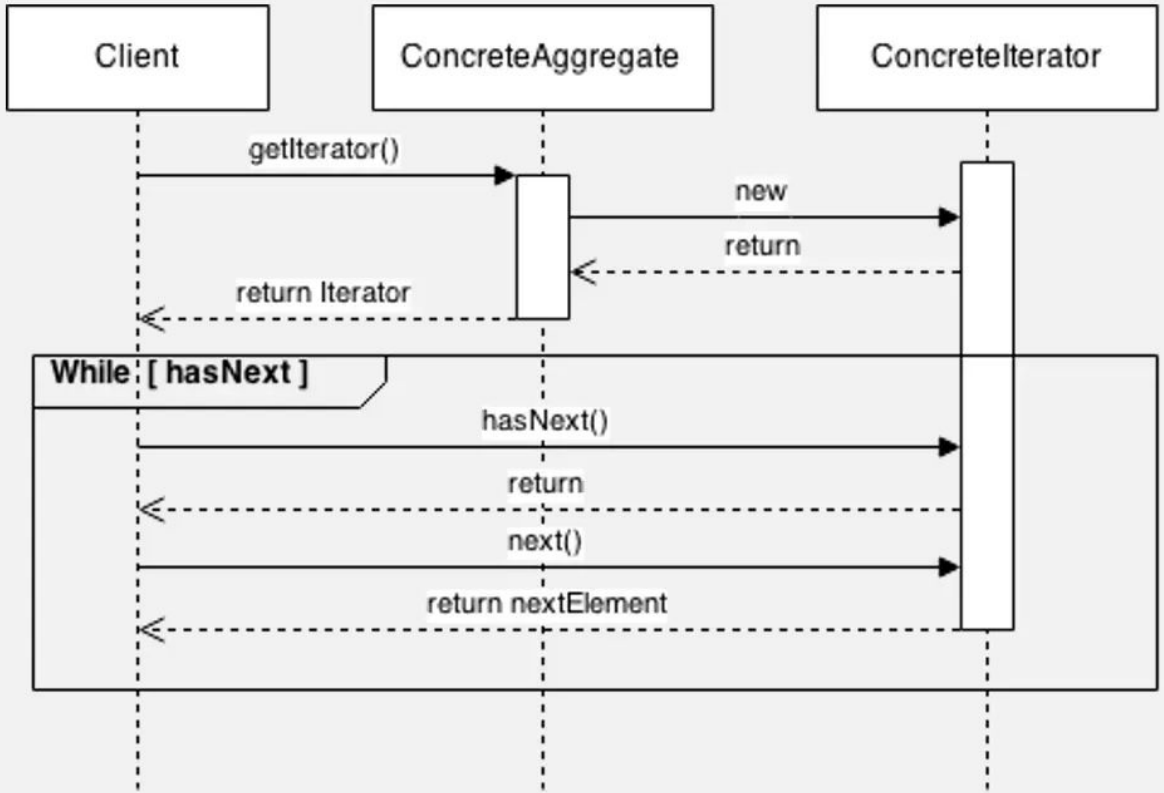
```
public interface Iterable<T> {
   Iterator<T> iterator();

   default void forEach(Consumer<? super T> action)
{...}
   default Spliterator<T> spliterator() {...}
}
```

Способы реализации

Если нужно создать свою коллекцию и итератор, нужно реализовать интерфейсы `Iterable` и `Iterator`

```
import java.util.Iterator;

class CustomCollection implements Iterable<Integer> {
    private int[] data;

    public CustomCollection(int[] data) {
        this.data = data;
    }

    @Override
    public Iterator<Integer> iterator() {
        return new CustomIterator();
    }

    private class CustomIterator implements Iterator<Integer> {
        private int index = 0;

        @Override
        public boolean hasNext() {
            return index < data.length;
        }

        @Override
        public Integer next() {
            return data[index++];
        }
    }
}
```

Iterator pattern – Diagram of sequence



Пример использования паттерна

```java
import java.util.Iterator;

class CustomCollection implements Iterable<Integer> {
    private int[] data;

    public CustomCollection(int[] data) {
        this.data = data;
    }

    @Override
    public Iterator<Integer> iterator() {
        return new CustomIterator();
    }

    private class CustomIterator implements Iterator<Integer> {
        private int index = 0;

        @Override
        public boolean hasNext() {
            return index < data.length;
        }

        @Override
        public Integer next() {
            return data[index++];
        }
    }
}
```

```java
public class Main {
    public static void main(String[] args) {
        int[] numbers = {1, 2, 3, 4, 5};
        var collection = new CustomCollection(numbers);

        // Явное использование итератора
        Iterator<Integer> iter = collection.iterator();
        while (iter.hasNext()) {
            System.out.println(iter.next());
        }

        // Цикл foreach
        for (int number : collection) {
            System.out.println(number);
        }
    }
}
```

```
1
2
3
4
5
1
2
3
4
5

Process finished with exit code 0
```

ListIterator

```
public interface ListIterator<E> extends Iterator<E> {
   boolean hasNext();

   E next();

   boolean hasPrevious();

   E previous();

   int nextIndex();

   int previousIndex();

   void remove();

   void set(E e);

   void add(E e);
}
```

Spliterator

```
public interface Spliterator<T> {
   boolean tryAdvance(Consumer<? super T> action);

   Spliterator<T> trySplit();

   long estimateSize();

   int characteristics();
}
```

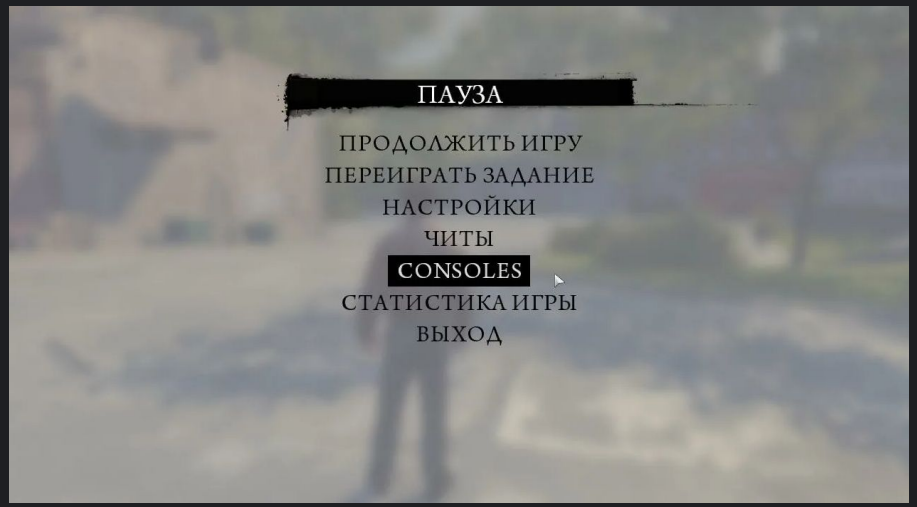
Основные различия

Характеристика	Iterator	ListIterator	Spliterator
Направление обхода	Только вперед	Вперед и назад	Вперёд, параллельно
Поддержка индексов	Нет	Да	Нет
Изменение коллекции	Удаление	Удаление, добавление, замена	Нет напрямую
Параллельная обработка	Нет	Нет	Да
Потоки (Stream API)	Нет	Нет	Да

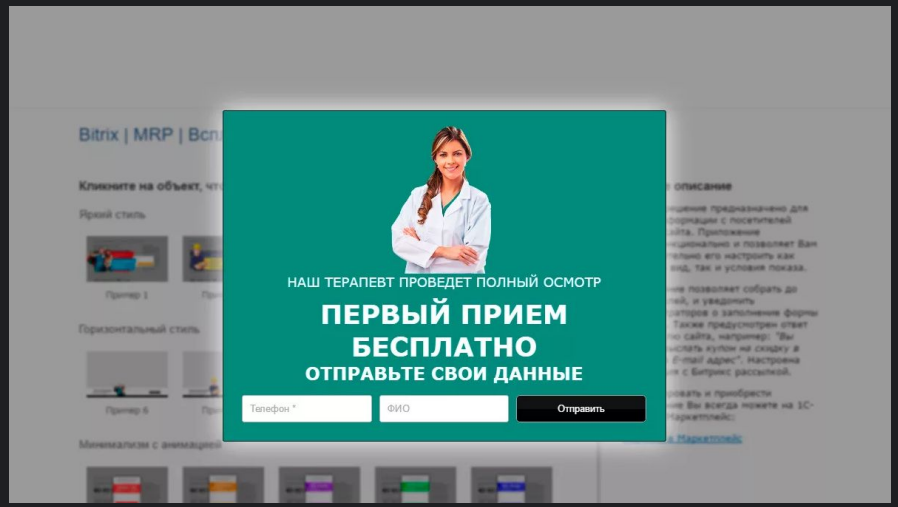
Observer

Состояния программы

Игра на паузе



Предложение регистрации



Исходники примеров:

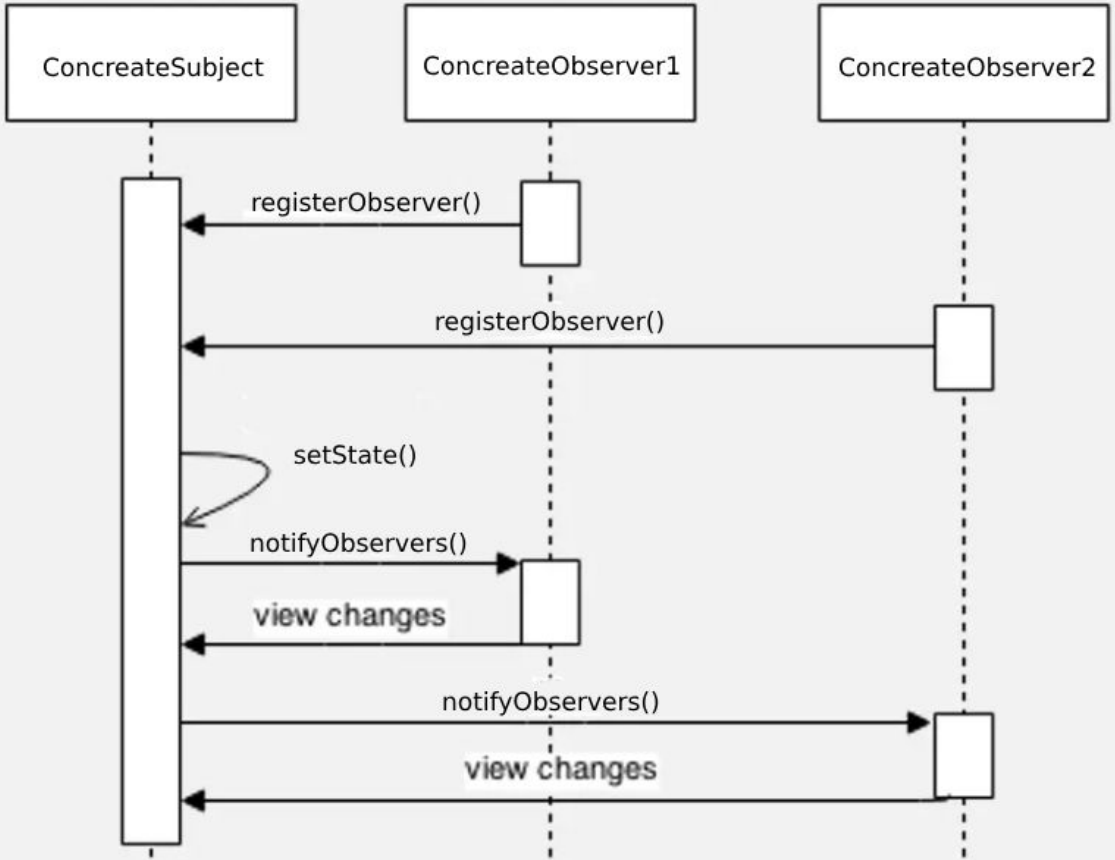
https://github.com/azya0/java2025 Директория “Presentation“

```java
interface Subject {
    void registerObserver(Observer observer);
    void removeObserver(Observer observer);
    void notifyObservers();

    int getState();
    void setState(int state);
}

interface Observer {
    void subscribe(Subject subscriber);
    void update(int state);

    Subject getSubject();
}
```

```java
class ConcreteSubject implements Subject {
    private List<Observer> observers;
    private int state;

    public ConcreteSubject() {
        observers = new ArrayList<>();
    }

    public Observer createObserver() {
        var result = new ConcreteObserver(this);
        registerObserver(result);
        return result;
    }

    @Override
    public void registerObserver(Observer observer) {
        observers.add(observer);
    }

    @Override
    public void removeObserver(Observer observer) {
        observers.remove(observer);
    }

    @Override
    public void notifyObservers() {
        for (Observer observer : observers) {
            observer.update(state);
        }
    }

    @Override
    public void setState(int state) {
        this.state = state;
        notifyObservers();
    }

    @Override
    public int getState() {
        return this.state;
    }
}
```

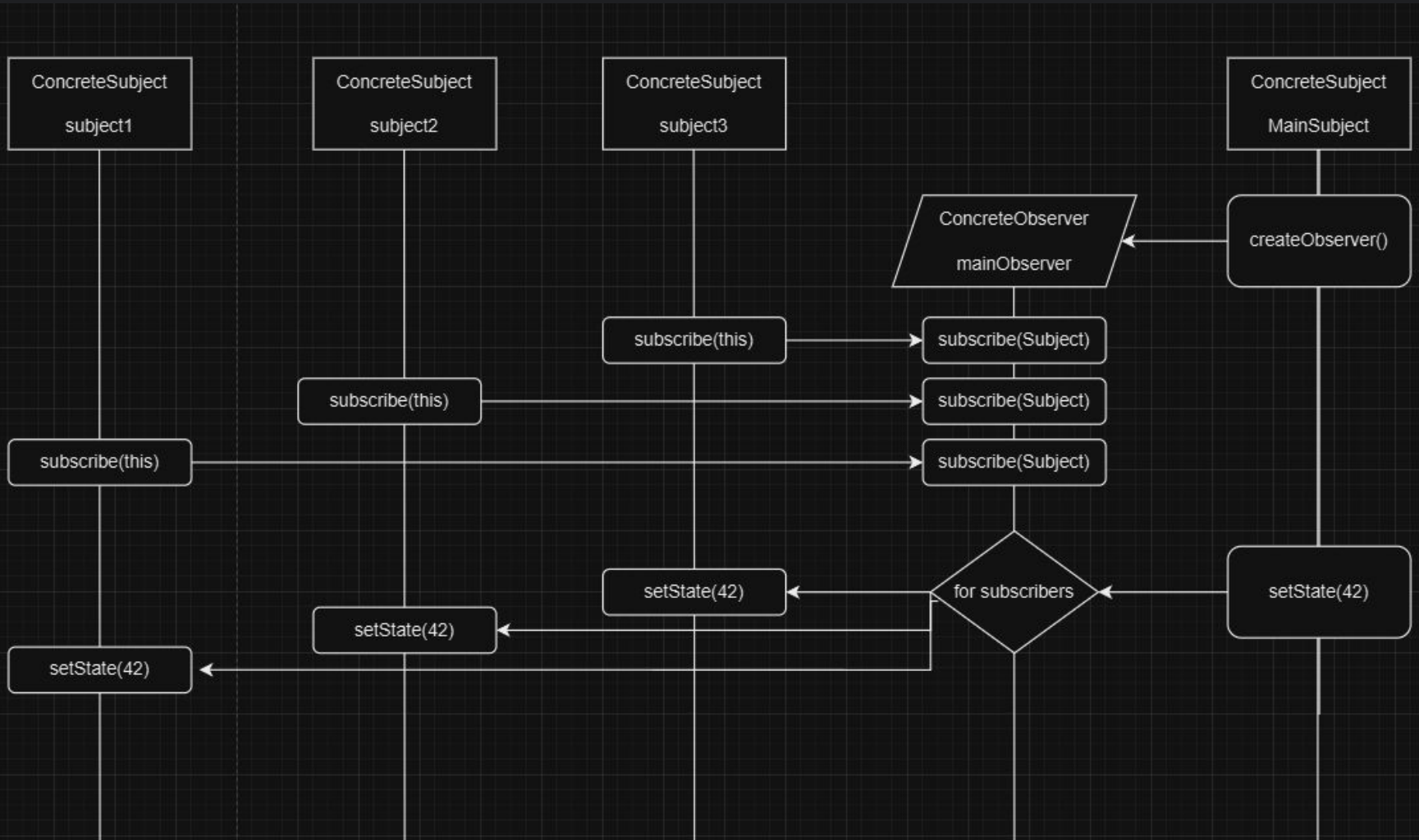
```java
class ConcreteObserver implements Observer {
    private Subject subject;
    private List<Subject> subscribers;


    public ConcreteObserver(Subject subject) {
        this.subject = subject;
        this.subject.registerObserver(this);
        this.subscribers = new ArrayList<>();
    }

    @Override
    public void update(int state) {
        for (Subject subject : subscribers) {
            if (subject.getState() != state) {
                subject.setState(state);
            }
        }
    }

    @Override
    public Subject getSubject() {
        return this.subject;
    }

    @Override
    public void subscribe(Subject subscriber) {
        subscribers.add(subscriber);
    }
}
```

Observer pattern – Diagram of sequence



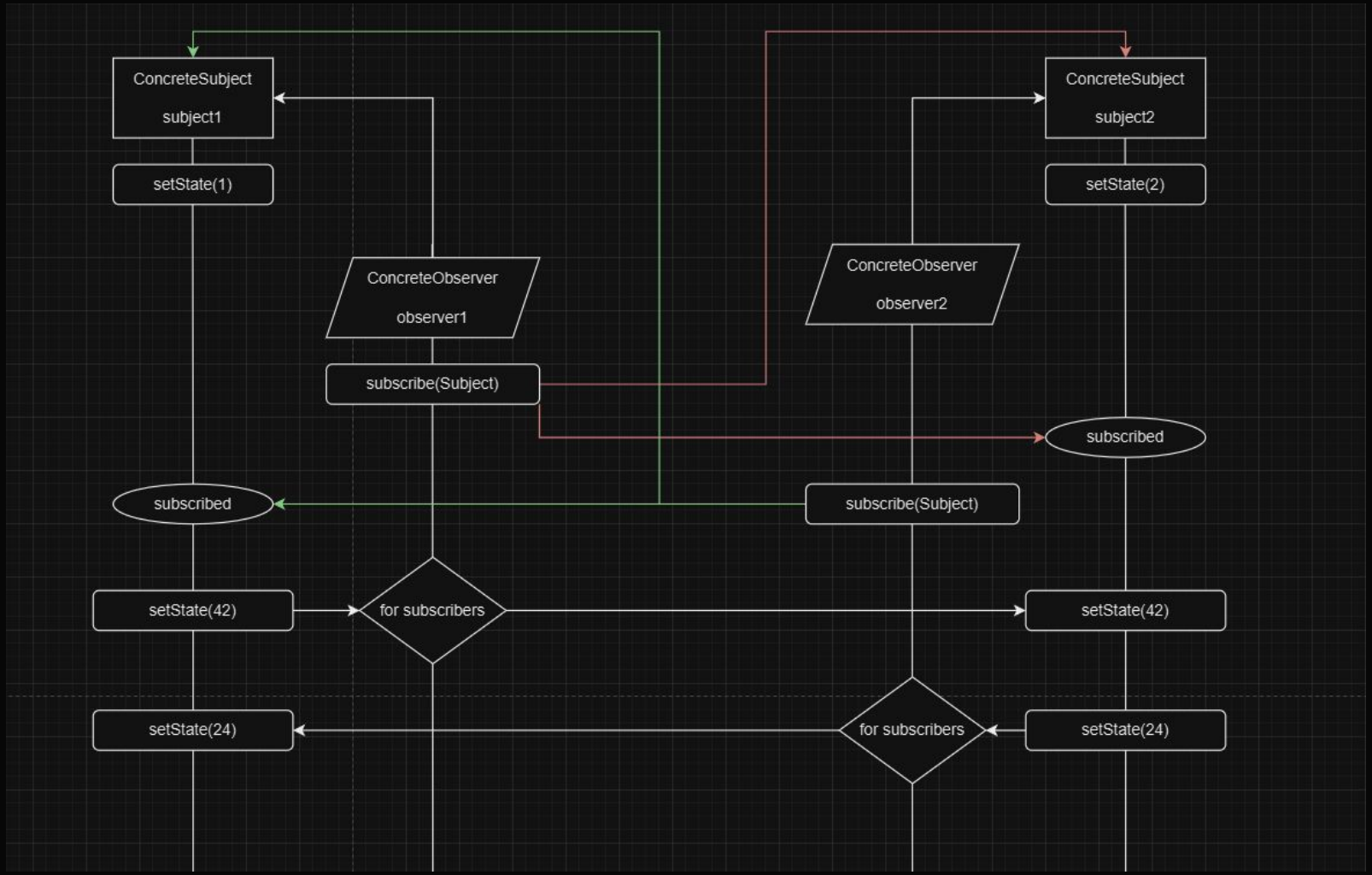
```java
public static void firstExample() {
    ConcreteSubject mainSubject = new ConcreteSubject();

    ConcreteSubject subject1 = new ConcreteSubject();
    ConcreteSubject subject2 = new ConcreteSubject();
    ConcreteSubject subject3 = new ConcreteSubject();

    subject1.setState(state:1);
    subject2.setState(state:2);
    subject3.setState(state:3);

    System.out.println("subject1 state: " + subject1.getState());
    System.out.println("subject2 state: " + subject2.getState());
    System.out.println("subject3 state: " + subject3.getState());

    Observer mainObserver = mainSubject.createObserver();

    mainObserver.subscribe(subject1);
    mainObserver.subscribe(subject2);
    mainObserver.subscribe(subject3);

    mainSubject.setState(state:42);

    System.out.println("subject1 state: " + subject1.getState());
    System.out.println("subject2 state: " + subject2.getState());
    System.out.println("subject3 state: " + subject3.getState());
}
```

```
subject1 state: 1
subject2 state: 2
subject3 state: 3
subject1 state: 42
subject2 state: 42
subject3 state: 42
```


ConcreteSubject
subject1
ConcreteSubject
subject2
ConcreteSubject
subject3
ConcreteSubject
MainSubject
ConcreteObserver
mainObserver
createObserver()
subscribe(this)
subscribe(Subject)
subscribe(this)
subscribe(Subject)
subscribe(this)
subscribe(Subject)
for subscribers
setState(42)
setState(42)
setState(42)
setState(42)

```java
public static void secondExample() {
    ConcreteSubject subject1 = new ConcreteSubject();
    subject1.setState(state:1);
    Observer Observer1 = new ConcreteObserver(subject1);

    ConcreteSubject subject2 = new ConcreteSubject();
    subject2.setState(state:2);
    Observer Observer2 = new ConcreteObserver(subject2);

    Observer1.subscribe(subject2);
    Observer2.subscribe(subject1);

    System.out.println("subject1 state: " + subject1.getState());
    System.out.println("subject2 state: " + subject2.getState());

    subject1.setState(state:42);

    System.out.println("subject1 state: " + subject1.getState());
    System.out.println("subject2 state: " + subject2.getState());

    subject2.setState(state:24);

    System.out.println("subject1 state: " + subject1.getState());
    System.out.println("subject2 state: " + subject2.getState());
}
```

```
subject1 state: 1
subject2 state: 2
subject1 state: 42
subject2 state: 42
subject1 state: 24
subject2 state: 24
```


ConcreteSubject
subject1
ConcreteSubject
subject2
setState(1)
setState(2)
ConcreteObserver
observer1
ConcreteObserver
observer2
subscribe(Subject)
subscribed
subscribed
subscribe(Subject)
setState(42)
for subscribers
setState(42)
setState(24)
for subscribers
setState(24)

Источники

- https://javarush.com/quests/lectures/questcollections.level07.lecture03
- https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/Iterable.html
- https://javarush.com/groups/posts/1884-pattern-iterator