Інженерія програмного забезпечення

Java

Анотації. Рефлексія.
Шаблон Observer

Зміст

7 Анотації. Рефлексія. Шаблон Observer 3
7.1 Теоретичні відомості 3
7.2 Загальне завдання 3
7.3 Запитання для елементарного рівня 3
7.4 Приклад проекту 4
7.4.1 Розробка програми 4
7.4.2 Засоби ООП, що використовувалися 4
7.4.3 Ієрархія і структура класів 5
7.4.4 Опис програми 7
7.5 Текст програми 9
7.5.1 Файл AnnotatedObserver.java 9
7.5.2 Файл Event.java 9
7.5.3 Файл Item.java 9
7.5.4 Файл Items.java 10
7.5.5 Файл ItemsGenerator.java 11
7.5.6 Файл ItemsSorter.java 12
7.5.7 Файл Main.java 12
7.5.8 Файл Observable.java 14
7.5.9 Файл Observer .java 14
7.5.10 Файл MainTest.java 14
7.6 Результати тестування 16

7 АНОТАЦІЇ. РЕФЛЕКСІЯ. ШАБЛОН OBSERVER

Мета:

- придбання навичок використання засобів анотування;
- ознайомлення з механізмом рефлексії;
- реалізація обслуговування колекції об'єктів на основі шаблону проектування Observer;
- використання модульного тестування;
- підготовка документації на основі коментарів інструментом автоматичної генерації javadoc.

7.1 Теоретичні відомості

...

7.2 Загальне завдання

Разработать иерархию классов согласно шаблону Observer и продемонстрировать возможность обслуживания разработанной ранее коллекции (наблюдаемый объект, Observable) различными (не менее двух) наблюдателями (Observers) – отслеживание изменений, упорядочивание, вывод, отображение и т.д. При реализации иерархии классов использовать средства аннотирования (Annotation). Отметить особенности различных политик удержания аннотаций (annotation retention policies). Продемонстрировать поддержку классами концепции рефлексии (Reflection).

Разработать класс для тестирования функциональности приложения.

Использовать комментарии для автоматической генерации документации средствами javadoc.

7.3 Запитання для елементарного рівня

1) ...?

7.4 Приклад проекту

7.4.1 Розробка програми

Реализуем классы, структура которых соответствует схеме п.7.4.3.

Разработаем класс MainTest для проведения теста разработанных классов –ItemsGenerator, ItemsSorter, Items. Реализуем методы:

setUpBeforeClass() – выполняется первым;

testAdd() – тестирует операцию добавления объектов в коллекцию;

testAddDel() – тестирует операции добавления и удаления объектов;

testSort() – тестирует операцию сортировки объектов.

В процессе разработки необходимо обеспечить прохождение всех тестов.

7.4.2 Засоби ООП, що використовувалися

Поведенческий шаблон Observer предоставляет компоненту возможность гибкой рассылки сообщений зарегистрированным получателям.

На некотором объекте сконцентрировано внимание наблюдателей, заинтересованных в получении от него какой-то информации. Потребовав от наблюдающих объектов, чтобы они устанавливали сеансы связи с центральным объектом, можно значительно снизить накладные расходы на коммуникацию, т.к. устанавливать связь будут только объекты, заинтересованные в получении обновленной информации.

Гибкость шаблона позволяет применять его для рассылки информации как отдельным, так и всем компонентам системы.

В соответствии с этим шаблоном, генераторы сообщений (наблюдаемые компоненты) рассылают сообщения, которые представляют события в системе. Эти события обрабатываются одним или несколькими получателями сообщений (компоненты-наблюдатели). Наблюдаемые компоненты отвечают за доставку событий всем заинтересованным наблюдателям (т.е. тем, которые установили сеансы связи). Интерфейс передачи сообщений позволяет наблюдаемым компонентам детализировать события для наблюдателей.

Если наблюдаемый объект многопоточный, он может поддерживать очередь сообщений, приоритеты сообщений, перекрытие сообщений и т.д.

Шаблон можно изменить, чтобы наблюдатели самостоятельно получали сообщения: наблюдаемый объект извещает о том, что событие произошло, а заинтересованные наблюдатели вызывают метод наблюдаемого для получения дополнительной информации о событии.

Аннотации – средство, которое позволяет встроить некоторую информацию (метаданные) в исходные и исполняемые файлы.

Аннотироваться могут классы, методы, поля, параметры, константы enum и сами аннотации.

Политика удержания аннотации определяет, на каком этапе аннотация отбрасывается. Определены три политики:

SOURCE – аннотации удерживаются только в исходном файле,

CLASS – аннотации сохраняются в файле .class во время компиляции, но недоступны JVM во время выполнения,

RUNTIME – аннотации сохраняются в файле .class и доступны во время выполнения.

Механизм рефлексии – позволяет обрабатывать типы, отсутствующие при компиляции, но появившиеся во время выполнения программы.

Рефлексия и наличие логически целостной модели выдачи информации об ошибках дает возможность создавать корректный динамический код.

RTTI позволяет получить информацию о точном типе объекта, когда имеется лишь ссылка базового типа. Использование этой информации подразумевает отказ от всех преимуществ полиморфизма. Рекомендуется использовать именно полиморфные методы, а к RTTI обращаться только в крайнем случае.

Различие между механизмом RTTI и рефлексией состоит в том, что при использовании RTTI файл .class открывается и анализируется компилятором, а при использовании рефлексии файл .class открывается и обрабатывается системой выполнения.

7.4.3 Ієрархія і структура класів

Структура классов и схема их отношений приведена на рис.7.1.

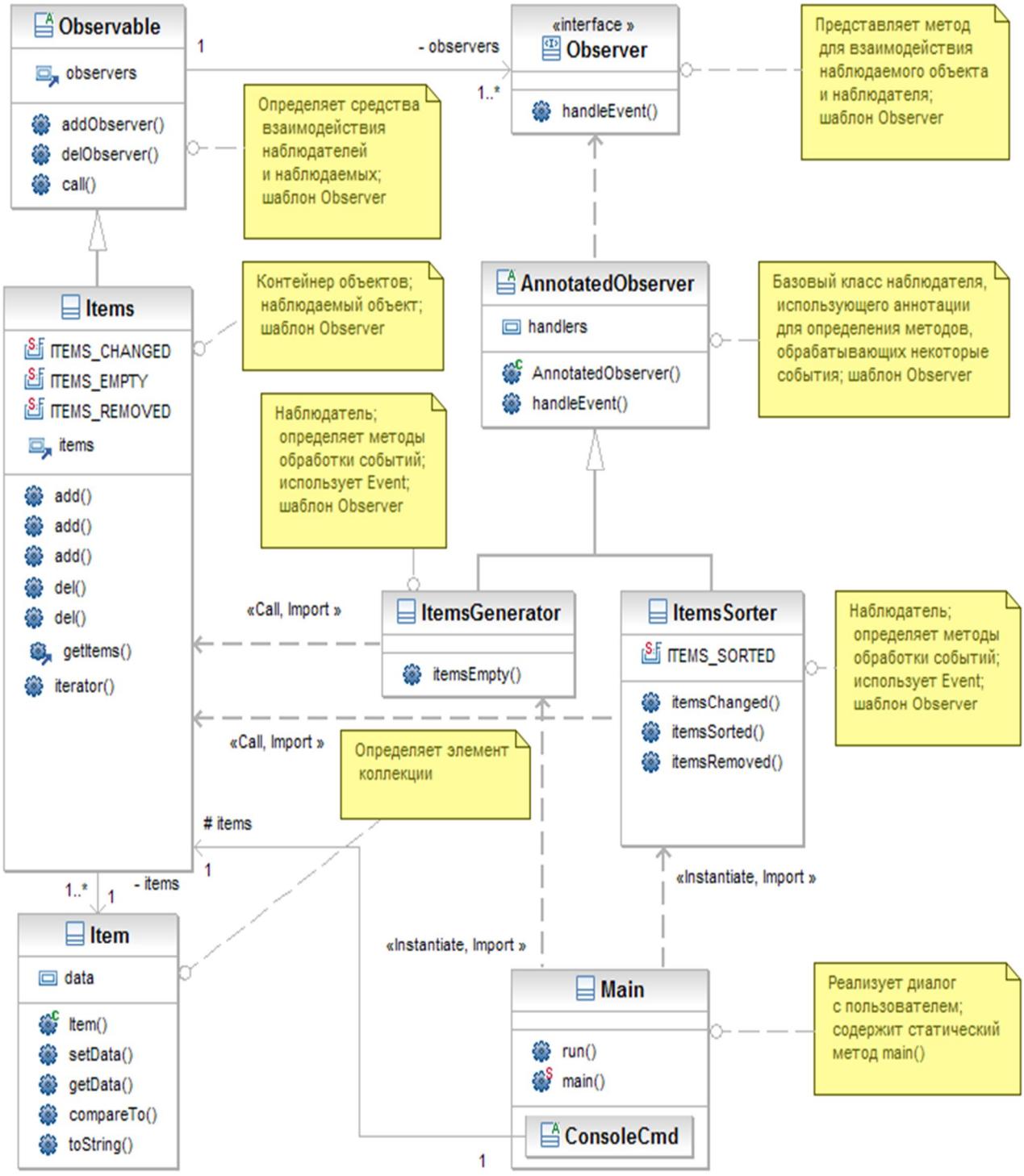


Рисунок 7.1 – Структура класів

7.4.4 Опис програми

Для реализации шаблона Observer использовались следующие классы и интерфейсы:

Observer – интерфейс, представляет метод для взаимодействия наблюдаемого объекта и наблюдателя; шаблон Observer.

Observable – абстрактный класс, определяет средства взаимодействия наблюдателей и наблюдаемых; шаблон Observer.

AnnotatedObserver – абстрактный класс, реализует Observer – базовый класс наблюдателя, использующего аннотации для определения методов, обрабатывающих некоторые события; шаблон Observer.

Event – аннотация времени выполнения для назначения методам наблюдателя конкретных событий.

Item – класс, определяет элемент коллекции.

Items – класс, расширяет Observable, контейнер объектов; наблюдаемый объект; шаблон Observer.

ItemsGenerator – класс, расширяет AnnotatedObserver, наблюдатель; определяет методы обработки событий; использует Event; шаблон Observer.

ItemsSorter – класс, расширяет AnnotatedObserver, наблюдатель; определяет методы обработки событий; использует Event; шаблон Observer.

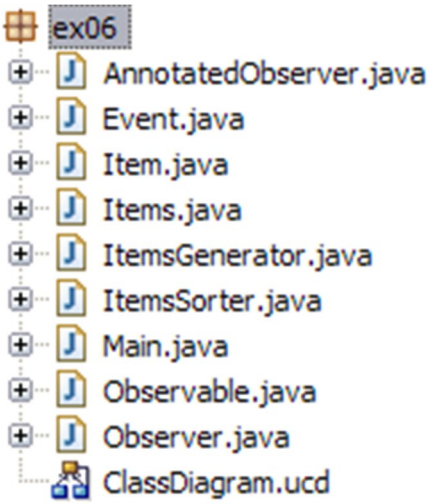
Main – класс, реализует диалог с пользователем; содержит статический метод main().

В данном примере показано, как наблюдаемый объект рассылает всем наблюдателям информацию об обновленном состоянии объекта Items. Наблюдатели при создании заполняют ассоциативный массив парами событие-обработчик. При этом для нахождения обработчиков используется механизм рефлексии – в конструкторе класса AnnotatedObserver просматриваются все методы и, отмеченные аннотацией Event, помещаются в массив. Параметр аннотации определяет идентификатор события. При получении сообщения вызывается обработчик, соответствующий идентификатору.

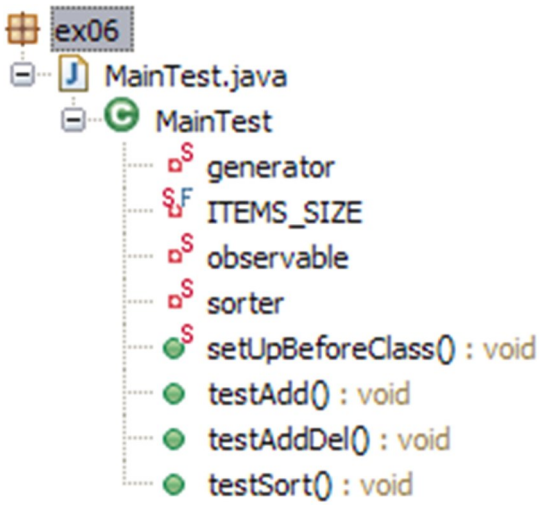
При написании исходного кода используем стиль комментариев документации javadoc.

Структура проекта:

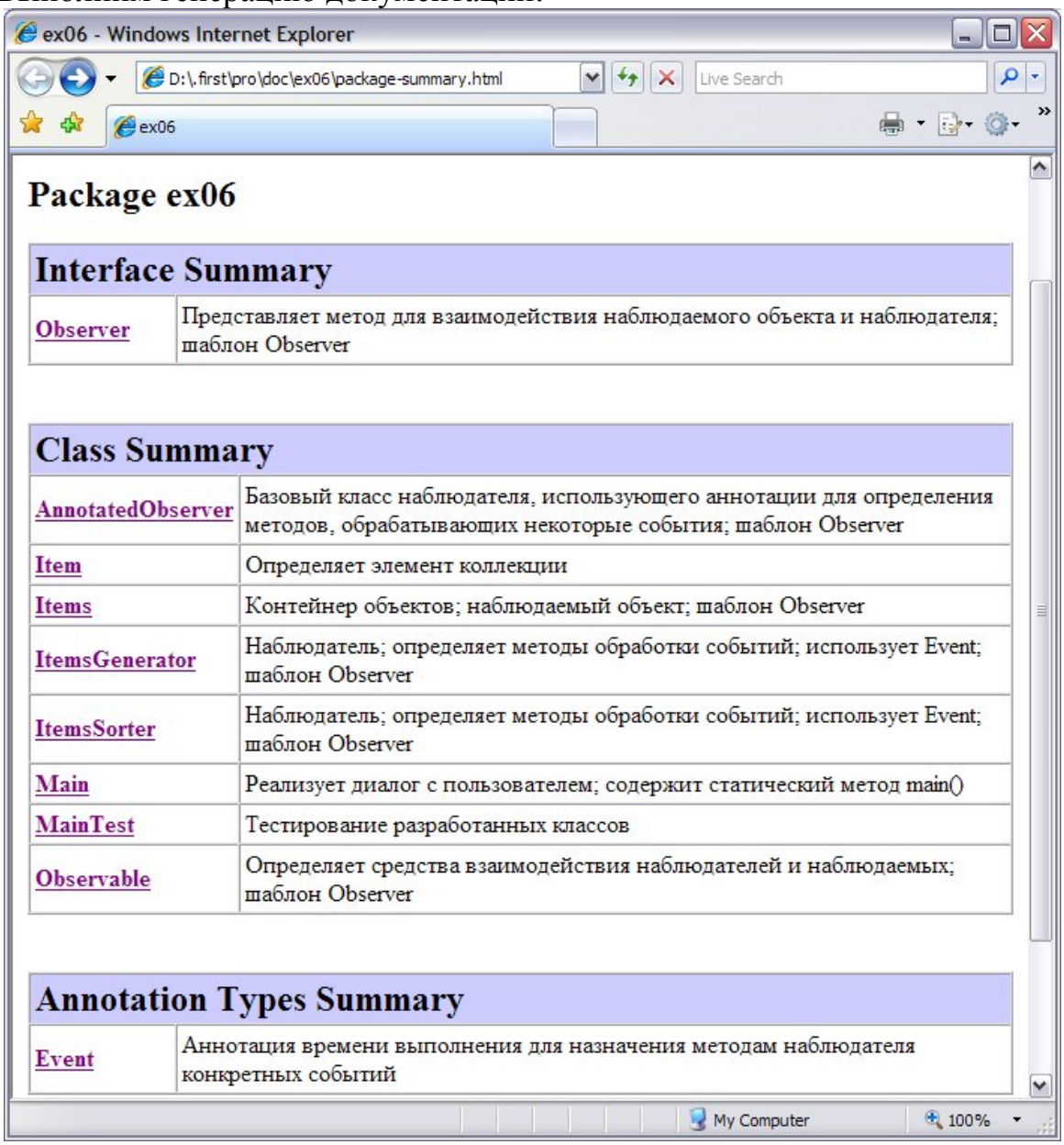
Папка src



Папка test



Выполним генерацию документации:



После проверки работоспособности готовой программы, создадим исполняемый JAR файл ex06.jar

7.5 Текст програми

7.5.1 Файл AnnotatedObserver.java

```
package ex06;

import java.lang.reflect.Method;
import java.util.HashMap;
import java.util.Map;

/** Базовый класс наблюдателя,
 * использующего аннотации
 * для определения методов,
 * обрабатывающих некоторые
 * события; шаблон Observer
 * @author xone
 * @version 1.0
 * @see Observer
 * @see Observable
 */
public abstract class AnnotatedObserver implements Observer {

	/** Ассоциативный массив обработчиков событий; содержит пары событие-обработчик */
	private Map<Object, Method> handlers = new HashMap<Object, Method>();

	/** Заполняет {@linkplain AnnotatedObserver#handlers} ссылками на методы,
	 * отмеченные аннотацией {@linkplain Event}
	 */
	public AnnotatedObserver() {
		for (Method m : this.getClass().getMethods()) {
			if (m.isAnnotationPresent(Event.class)) {
				handlers.put(m.getAnnotation(Event.class).value(), m);
			}
		}
	}

	@Override
	public void handleEvent(Observable observable, Object event) {
		Method m = handlers.get(event);
		try {
			if (m != null) m.invoke(this, observable);
		} catch (Exception e) {
			System.err.println(e);
		}
	}

}
```

7.5.2 Файл Event.java

```
package ex06;

import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;

/** Аннотация времени
 * выполнения для
 * назначения
 * методам наблюдателя
 * конкретных событий
 * @author xone
 * @see AnnotatedObserver
 */
@Retention(RetentionPolicy.RUNTIME)
public @interface Event {
	String value();
}
```

7.5.3 Файл Item.java

```
package ex06;

/** Определяет элемент
 * коллекции
```

```
 * @author xone
 * @see Items
 */
public class Item implements Comparable<Item> {

	/** Информационное поле */
	private String data;

	/** Инициализирует {@linkplain Item#data}
	 * @param data значение для поля {@linkplain Item#data}
	 */
	public Item(String data) {
		this.data = data;
	}

	/** Устанавливает поле {@linkplain Item#data}
	 * @param data значение для поля {@linkplain Item#data}
	 * @return значение поля {@linkplain Item#data}
	 */
	public String setData(String data) {
		return this.data = data;
	}

	/** Возвращает поле {@linkplain Item#data}
	 * @return значение поля {@linkplain Item#data}
	 */
	public String getData() {
		return data;
	}

	@Override
	public int compareTo(Item o) {
		return data.compareTo(o.data);
	}

	@Override
	public String toString() {
		return data;
	}

}
```

7.5.4 Файл Items.java

```
package ex06;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

/** Контейнер объектов;
 * наблюдаемый объект;
 * шаблон Observer
 * @author xone
 * @see Observable
 * @see Observer
 * @see Item
 */
public class Items extends Observable implements Iterable<Item> {

	/** Константа-идентификатор события, обрабатываемого наблюдателями */
	public static final String ITEMS_CHANGED = "ITEMS_CHANGED";
	/** Константа-идентификатор события, обрабатываемого наблюдателями */
	public static final String ITEMS_EMPTY = "ITEMS_EMPTY";
	/** Константа-идентификатор события, обрабатываемого наблюдателями */
	public static final String ITEMS_REMOVED = "ITEMS_REMOVED";

	/** Коллекция объектов класса {@linkplain Item} */
	private List<Item> items = new ArrayList<Item>();

	/** Добавляет объект в коллекцию и извещает наблюдателей
	 * @param item объект класса {@linkplain Item}
	 */
	public void add(Item item) {
		items.add(item);
		if (item.getData().isEmpty()) call(ITEMS_EMPTY);
		else call(ITEMS_CHANGED);
	}

	/** Добавляет объект в коллекцию
```

```
         * @param s передается конструктору {@linkplain Item#Item(String)}
         */
        public void add(String s) {
                add(new Item(s));
        }

        /** Добавляет несколько объектов в коллекцию и извещает наблюдателей
         * @param n количество добавляемых объектов класса {@linkplain Item}
         */
        public void add(int n) {
                if (n > 0) {
                        while (n-- > 0) items.add(new Item(""));
                        call(ITEMS_EMPTY);
                }
        }

        /** Удаляет объект из коллекции и извещает наблюдателей
         * @param item удаляемый объект
         */
        public void del(Item item) {
                if (item != null) {
                        items.remove(item);
                        call(ITEMS_REMOVED);
                }
        }

        /** Удаляет объект из коллекции и извещает наблюдателей
         * @param index индекс удаляемого объекта
         */
        public void del(int index) {
                if ((index >= 0) && (index < items.size())) {
                        items.remove(index);
                        call(ITEMS_REMOVED);
                }
        }

        /** Возвращает ссылку на коллекцию
         * @return ссылка на коллекцию объектов класса {@linkplain Item}
         */
        public List<Item> getItems() {
                return items;
        }

        @Override
        public Iterator<Item> iterator() {
                return items.iterator();
        }

}
```

7.5.5 Файл ItemsGenerator.java

```
package ex06;

/** Наблюдатель;
 * определяет методы
 * обработки событий;
 * использует Event;
 * шаблон Observer
 * @author xone
 * @see AnnotatedObserver
 * @see Event
 */
public class ItemsGenerator extends AnnotatedObserver {

        /** Обработчик события {@linkplain Items#ITEMS_EMPTY};
         * извещает наблюдателей; шаблон Observer
         * @param observable наблюдаемый объект класса {@linkplain Items}
         * @see Observable
         */
        @Event(Items.ITEMS_EMPTY)
        public void itemsEmpty(Items observable) {
                for (Item item : observable) {
                        if (item.getData().isEmpty()) {
                                int len = (int)(Math.random() * 10) + 1;
                                String data = "";
                                for (int n = 1; n <= len; n++) {
                                        data += (char)((int)(Math.random() * 26) + 'A');
                                }
                                item.setData(data);
```

```
                        }
                }
                observable.call(Items.ITEMS_CHANGED);
        }

}
```

7.5.6 Файл ItemsSorter.java

```
package ex06;

import java.util.Collections;

/** Наблюдатель;
 * определяет методы
 * обработки событий;
 * использует Event;
 * шаблон Observer
 * @author xone
 * @see AnnotatedObserver
 * @see Event
 */
public class ItemsSorter extends AnnotatedObserver {

        /** Константа-идентификатор события, обрабатываемого наблюдателями */
        public static final String ITEMS_SORTED = "ITEMS_SORTED";

        /** Обработчик события {@linkplain Items#ITEMS_CHANGED};
         * извещает наблюдателей; шаблон Observer
         * @param observable наблюдаемый объект класса {@linkplain Items}
         * @see Observable
         */
        @Event(Items.ITEMS_CHANGED)
        public void itemsChanged(Items observable) {
                Collections.sort(observable.getItems());
                observable.call(ITEMS_SORTED);
        }

        /** Обработчик события {@linkplain Items#ITEMS_SORTED}; шаблон Observer
         * @param observable наблюдаемый объект класса {@linkplain Items}
         * @see Observable
         */
        @Event(ITEMS_SORTED)
        public void itemsSorted(Items observable) {
                System.out.println(observable.getItems());
        }

        /** Обработчик события {@linkplain Items#ITEMS_REMOVED}; шаблон Observer
         * @param observable наблюдаемый объект класса {@linkplain Items}
         * @see Observable
         */
        @Event(Items.ITEMS_REMOVED)
        public void itemsRemoved(Items observable) {
                System.out.println(observable.getItems());
        }

}
```

7.5.7 Файл Main.java

```
package ex06;

import ex04.ConsoleCommand;
import ex04.Menu;

/** Реализует диалог
 * с пользователем;
 * содержит статический
 * метод main()
 * @author xone
 * @version 6.0
 * @see Main#main
 */
public class Main {

        /** Консольная команда;
         * используется при
         * создании анонимных
```

```
 * экземпляров команд
 * пользовательского
 * интерфейса;
 * шаблон Command
 * @author xone
 * @see ConsoleCommand
 */
abstract class ConsoleCmd implements ConsoleCommand {

	/** Коллекция объектов {@linkplain Items} */
	protected Items items;
	/** Отображаемое название команды */
	private String name;
	/** Символ горячей клавиши команды */
	private char key;

	/** Инициализирует поля консольной команды
	 * @param items {@linkplain ConsoleCmd#items}
	 * @param name {@linkplain ConsoleCmd#name}
	 * @param key {@linkplain ConsoleCmd#key}
	 */
	ConsoleCmd(Items items, String name, char key) {
		this.items = items;
		this.name = name;
		this.key = key;
	}

	@Override
	public char getKey() {
		return key;
	}

	@Override
	public String toString() {
		return name;
	}

}

/** Устанавливает связь наблюдателей с наблюдаемыми объектами;
 * реализует диалог с пользователем
 */
public void run() {
	Items items = new Items();
	ItemsGenerator generator = new ItemsGenerator();
	ItemsSorter sorter = new ItemsSorter();
	items.addObserver(generator);
	items.addObserver(sorter);
	Menu menu = new Menu();
	menu.add(new ConsoleCmd(items, "'v'iew", 'v') {
		@Override
		public void execute() {
			System.out.println(items.getItems());
		}
	});
	menu.add(new ConsoleCmd(items, "'a'dd", 'a') {
		@Override
		public void execute() {
			items.add("");
		}
	});
	menu.add(new ConsoleCmd(items, "'d'el", 'd') {
		@Override
		public void execute() {
			items.del((int)Math.round(Math.random()*(items.getItems().size()-1)));
		}
	});
	menu.execute();
}

/** Выполняется при запуске программы
 * @param args параметры запуска программы
 */
public static void main(String[] args) {
	new Main().run();
}

}
```

7.5.8 Файл Observable.java

```
package ex06;

import java.util.HashSet;
import java.util.Set;

/** Определяет средства
 * взаимодействия
 * наблюдателей
 * и наблюдаемых;
 * шаблон Observer
 * @author xone
 * @version 1.0
 * @see Observer
 */
public abstract class Observable {

	/** Множество наблюдателей; шаблон Observer
	 * @see Observer
	 */
	private Set<Observer> observers = new HashSet<Observer>();

	/** Добавляет наблюдателя; шаблон Observer
	 * @param observer объект-наблюдатель
	 */
	public void addObserver(Observer observer) {
		observers.add(observer);
	}

	/** Удаляет наблюдателя; шаблон Observer
	 * @param observer объект-наблюдатель
	 */
	public void delObserver(Observer observer) {
		observers.remove(observer);
	}

	/** Оповещает наблюдателей о событии; шаблон Observer
	 * @param event информация о событии
	 */
	public void call(Object event) {
		for (Observer observer : observers) {
			observer.handleEvent(this, event);
		}
	}

}
```

7.5.9 Файл Observer .java

```
package ex06;

/** Представляет метод
 * для взаимодействия
 * наблюдаемого объекта
 * и наблюдателя;
 * шаблон Observer
 * @author xone
 * @version 1.0
 * @see Observable
 */
public interface Observer {
	/** Вызывается наблюдаемым объектом для каждого наблюдателя; шаблон Observer
	 * @param observable ссылка на наблюдаемый объект
	 * @param event информация о событии
	 */
	public void handleEvent(Observable observable, Object event);
}
```

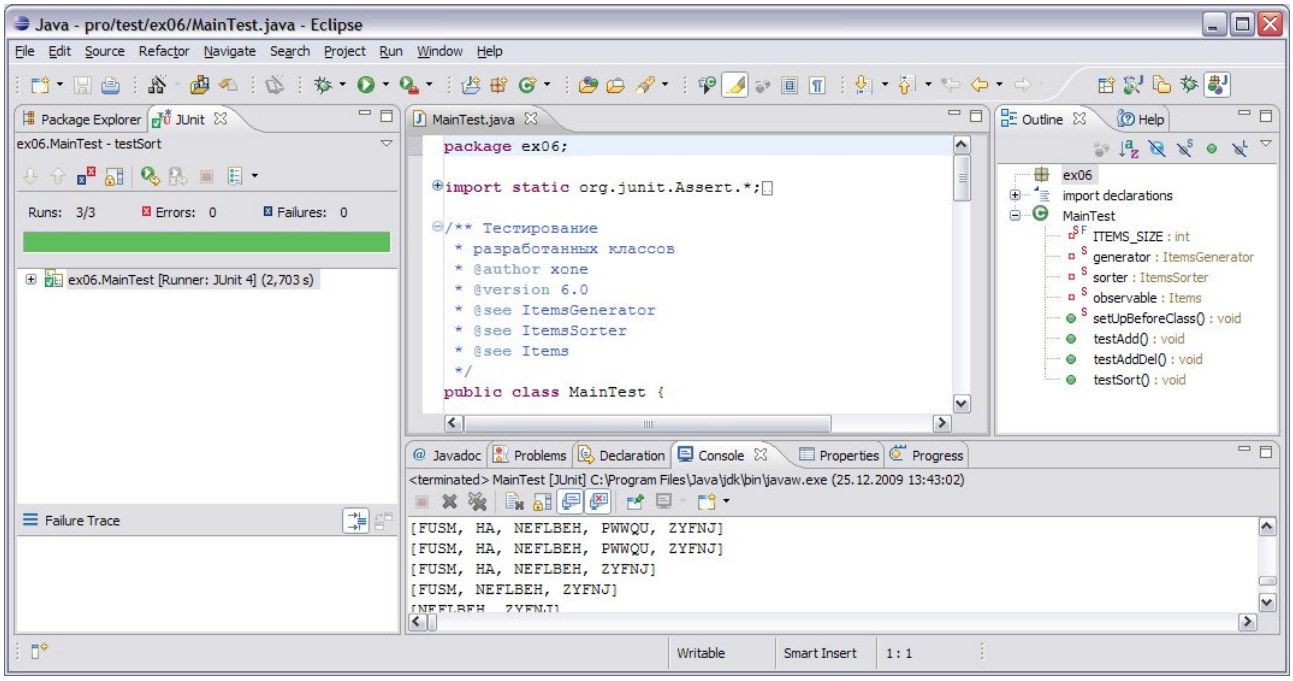
7.5.10 Файл MainTest.java

```
package ex06;

import static org.junit.Assert.*;
import java.util.ArrayList;
import java.util.Collections;
```

```
import java.util.List;
import java.util.Random;
import org.junit.BeforeClass;
import org.junit.Test;

/** Тестирование
 * разработанных классов
 * @author xone
 * @version 6.0
 * @see ItemsGenerator
 * @see ItemsSorter
 * @see Items
 */
public class MainTest {

	/** Количество элементов в коллекции */
	private static final int ITEMS_SIZE = 1000;
	/** Наблюдатель; шаблон Observer */
	private static ItemsGenerator generator = new ItemsGenerator();
	/** Наблюдатель; шаблон Observer */
	private static ItemsSorter sorter = new ItemsSorter();
	/** Наблюдаемый объект; шаблон Observer */
	private static Items observable = new Items();

	/** Выполняется первым */
	@BeforeClass
	public static void setUpBeforeClass() {
		observable.addObserver(generator);
		observable.addObserver(sorter);
	}

	/** Тестирует операцию добавления объектов в коллекцию */
	@Test
	public void	testAdd() {
		observable.getItems().clear();
		observable.add(new Item("AAA"));
		observable.add("AAA");
		observable.add("");
		observable.add(ITEMS_SIZE);
		for (Item item : observable) {
			assertFalse(item.getData().isEmpty());
		}
		assertEquals(ITEMS_SIZE + 3, observable.getItems().size());
	}

	/** Тестирует операции добавления и удаления объектов */
	@Test
	public void	testAddDel() {
		Item tmp;
		observable.getItems().clear();
		observable.add("");
		observable.add(ITEMS_SIZE);
		for (int i = ITEMS_SIZE; i > 0; i--) {
			tmp = observable.getItems().get((new Random()).nextInt(i));
			observable.del(tmp);
		}
		assertEquals(1, observable.getItems().size());
	}

	/** Тестирует операцию сортировки объектов */
	@Test
	public void	testSort() {
		observable.getItems().clear();
		observable.add(ITEMS_SIZE);
		List<Item> items = new ArrayList<Item>(observable.getItems());
		Collections.sort(items);
		assertEquals(items, observable.getItems());
	}

}
```

7.6 Результати тестування

Выполним ex05.MainTest как JUnit Test



Выполним запуск программы из командной строки:

```
java -jar ex06.jar
```

В результате выполнения получим: