компетентностей у студентов технологических колледжей по результатам обучения учебной дисциплине «Информатика и вычислительная техника». Согласно этим критериям предложены следующие уровни сформированности информатических компетентностей у студентов технологических колледжей по результатам обучения учебной дисциплине «Информатика и вычислительная техника»: ознакомительный уровень, репродуктивный уровень, базовый уровень, повышенный уровень, творческий уровень.

Ключевые слова: *информационно-коммуникационные технологии, информатические компетентности, технологические колледжи, компетентностный подход.*

THE LEVEL CHARACTERISTIC OF THE FORMATION OF THE SYSTEM OF INFORMATIONAL COMPETENCIES FOR STUDENTS OF TECHNOLOGICAL COLLEGES DURING THE STUDY OF THE DISCIPLINE «COMPUTER SCIENCE AND COMPUTER TECHNOLOGY»

S.Nakonechna

Resume. *The article is analyzes literature on the introduction of the competence approach in the educational process, the problem of the formation of information competence. The criterion of the formation of the system of information competence in the students of technological colleges during the study of the discipline "Computer science and computer technology" is determined. In accordance with these criteria, the following levels of the formation of information competencein the students of technological colleges during the study of the discipline "Informatics and Computing": the educational level, the reproductive level, the basic level, the elevated level, the creative level are proposed.*

Keywords: *information competence, information and communication technologies, technology colleges, competence approach.*

DOI 10.31392/NPU-nc.series2.2018.20(27).21
УДК: 372.862

Ю.П. Біляй
старший викладач
Національний педагогічний університет імені М.П. Драгоманова

РЕАКТИВНЕ ПРОГРАМУВАННЯ

Анотація. *В статті розглядається процес створення мікросервіса на основі використання новітніх технологій розробки програмного забезпечення для підвищення зацікавленості студентів до вивчення програмування.*

Ключові слова: *реактивне програмування, парадигма програмування, створення мікросервісів, навчання програмування.*

Для зацікавлення програмуванням студентів інформатичних спеціальностей доцільною є демонстрація додатків, що мають очевидне практичне застосування. Проте на створення таких додатків, зазвичай, потрібно багато часу і в таких додатках важко зорієнтуватись студентові. Для покращення розуміння бажано зменшити кількість коду, що потрібно написати, але при цьому потрібно не втратити функціонал програмного продукту. Для цього можна використовувати готові фреймворки, використання яких дозволить приховати реалізацію деяких стандартних методів.

Команда розробників одного з найпопулярніших фреймворків на Java презентувала нову версію: Spring Framework 5. Одне з найбільших нововведень – модель реактивного програмування. Також відбулась презентація нової версії Spring Boot 2 [6], використання якої суттєво спрощує створення мікросервісів на основі реактивного підходу.

Для початку розглянемо поняття реактивності. І тут потрібно зробити відразу чітке розмежування у означеннях.

Реактивна система.

Реактивна система – архітектурний патерн, визначений за деяким набором правил (reactive manifesto). Даний маніфест був розроблений в 2013 році для усунення невизначеності. До 2013 в Європі і США у терміну «reactive» було багато значень. Кожен розумів по-своєму, яку систему можна назвати реактивною. Таким чином створювалась величезна плутанина, і в підсумку був створений маніфест, в якому встановлювались чіткі критерії реактивної системи.

Подивимося на зображення з маніфесту і проаналізуємо більш детально, що означає кожен пункт (див. Рис 1):

Responsive. За даним принципом у системі, що розробляється, опрацювання запитів повинно виконуватись швидко і за певний заздалегідь заданий час. Крім того система повинна бути досить гнучкою для діагностики внутрішніми засобами і налагодження.

Що це означає на практиці? Традиційно в разі запиту деякого сервісу здійснюється його переадресація до бази даних, з якої отримується необхідний обсяг даних, які відображаються на стороні користувача. Використання такого підходу до проектування систем можливе, якщо система досить швидка і база даних не дуже велика. Але якщо час формування запиту-відповіді буде набагато більше очікуваного? Крім того, у користувача могло зникнути з'єднання з мережею на кілька мілісекунд. Тоді всі зусилля щодо вибірки даних і формування відповіді пропадають. Наприклад, використовуючи сервіси «Gmail» або «Facebook», за недостатньо якісного з'єднання з мережею Інтернет, користувач не отримує помилку завантаження, а чекає результат більше звичайного. Крім того, це означає, що відповіді і запити повинні бути впорядковані і послідовні.

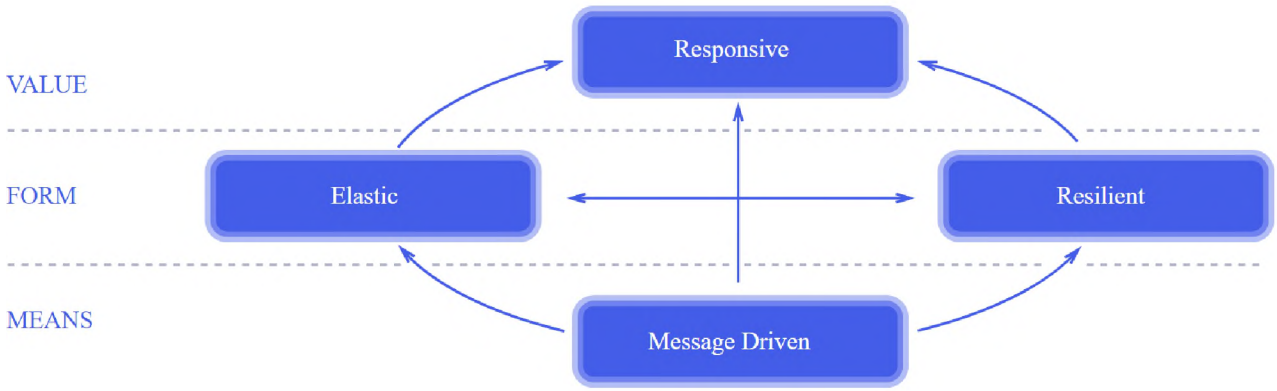

Рис. 1 Критерії реактивної системи

Resilient. Система залишається в робочому стані навіть, якщо один з компонентів перестав функціонувати. Іншими словами, компоненти системи повинні бути достатньо гнучкими і ізольованими один від одного. Досягається це шляхом реплікації. Якщо, наприклад, одна репліка PostgreSQL перестала функціонувати, необхідно зробити так, щоб завжди була доступна інша. Крім того, потрібно забезпечити роботу з великою кількістю екземплярів таких реплік.

Elastic. За даним принципом, розподіл обчислень у системі повинен займати якомога стабільнішу кількість ресурсів в кожен проміжок часу, тобто робота повинна виконуватись без пікових навантажень чи моментів простою. Якщо в деякий момент часу система перебуває під високим навантаженням, то необхідно збільшити кількість екземплярів додатку. У разі малого навантаження вільні ресурси машини повинні бути очищені. Типовий інструмент реалізації даного принципу: «Kubernetes».

Message Driven. Найважливіший пункт для Java-розробника. Взаємозв'язки між сервісами мають здійснюватися через асинхронні повідомлення. Це означає, що в кожному елементі системи генерується запит даних з іншого елемента, але не гарантується отримання результату відразу ж. Замість цього продовжується виконання завдання. Це дозволяє збільшити користь від системних ресурсів і більш гнучко здійснювати управління помилками, що виникають. Зазвичай такий результат досягається з використанням реактивного програмування.

Реактивне програмування – парадигма програмування, використання якої орієнтоване на роботу з потоками даних. Основна концепція реактивного програмування базується на неблокуючому введенні/виведенні. Зазвичай в разі звернення до деякого ресурсу (бази даних, файлу на диску, віддаленого сервера) результат отримується відразу ж (часто в тому самому рядкові). За неблокуючих звернень до ресурсу виконання потоку не зупиняється для отримання даних в разі звернення, а продовжується функціонування. Результат може бути отриманий пізніше і за необхідності. [1]

Практика. Для розробки використаємо фреймворк Spring WebFlux. Це новий фреймворк для реактивного програмування. Для розробки не використовується фреймворк Spring Web MVC тому, що не всі модулі в цьому фреймворку можна використовувати для роботи в реактивному режимі. Проте залишається багато коду із сторонніх бібліотек, наприклад, Tomcat, які засновані на декларативному програмуванні і потоках [4].

В процесі роботи над фреймворком була розроблена невелика специфікація для асинхронної роботи. Така специфікація була додана в Java 9. Але для спрощення у прикладі використаємо Java 8 і Spring Boot 2 [3].

Основні концепції. Використовуючи даний підхід будемо використовувати два основні класи для роботи в реактивному режимі:

- Mono

Клас Mono потрібен для роботи з єдиним об'єктом. Для цього створимо проект і сутність User в ньому:

```
@Data
@NoArgsConstructor
@AllArgsConstructor
public class User {
    private String firstName;
    private String lastName;
}
```

Клас з тестами і підготовленими записами користувачів:

```
public class HabrreactiveApplicationTests {
        private User peter = new User("Peter", "Koval");
        private User lois = new User("Leonid", "Tkach");
        private User brain = new User("Anton", "Dybenko");
}
```

Тест:

```
@Test
public void mono() {
// Створення об'кту
                Mono<User> monoPeter = Mono.just(peter);
// Блокування поточного потоку до того часу поки не буде отримано об'єкт
                User peter2 = monoPeter.block();
// Перевірка відповідності отриманого об'єкта очікуваному
                assertEquals(peter, peter2);
}
```

Крім того, у класі Mono реалізовано багато методів для розв'язування різних типових задач. Наприклад, метод map для перетворення одного типу в інший:

```
@Test
public void blockMono() {
    Mono<User> monoPeter = Mono.just(peter);
// Блокування поточного потоку до тих пір поки не будуть опрацьовані дані
    String name = monoPeter.map(User::getFirstName).block();
                assertEquals(name, "Peter");
}
```

- Flux

Даний клас подібний у використанні до Mono, але використовуючи його розробник може налагодити асинхронну роботу з великою кількістю об'єктів:

```
@Test
public void flux() {
// Створення потоку для відвантаження даних
        Flux<User> fluxUsers = Flux.just(peter, lois, brain);
// Отримання і опрацювання даних
        fluxUsers.subscribe(System.out::println);
}
```

Як і у випадку з Mono у класі Flux реалізовані деякі додаткові корисні методи:

```
@Test
public void fluxFilter() {
    Flux<User> userFlux = Flux.just(peter, leonid, anton);
// Фільтрація даних
    userFlux
    .filter(user -> user.getFirstName().equals("Peter"))
    .subscribe(user -> assertEquals(user, peter));
}
@Test
public void fluxMap() {
    Flux<User> userFlux = Flux.just(peter, leonid, anton);
```

```
// Перетворення типу User в String
    userFlux
        .map(User::getFirstName)
        .subscribe(System.out::println);
}
```

На відміну від стандартних (не віртуальних) потоків в разі завершення операцій основного потоку виконання збирання даних зупиняється, і виконання програми завершується. Наприклад після виконання наступного блоку у консолі дані не будуть відображені:

```
@Test
public void fluxDelayElements() {
    Flux<User> userFlux = Flux.just(peter, leonid, anton);
// Опрацювання події лише після очікування даних протягом однієї
секунди
    userFlux.delayElements(Duration.ofSeconds(1))
        .subscribe(System.out::println);
}
```

Цього можна уникнути за допомогою використання методів з класу `CountDownLatch`:

```
@Test
    public void fluxDelayElementsCountDownLatch()
    throws Exception {
// Створення лічильника з початковим значенням 1
    CountDownLatch countDownLatch = new CountDownLatch(1);
    Flux<User> userFlux = Flux.just(peter, leonid, anton);
// Запуск userFlux з часом перевірки одна секунда
    userFlux
    .delayElements(Duration.ofSeconds(1))
    .doOnComplete(countDownLatch::countDown)
    .subscribe(System.out::println); // виведення кожну секунду
// Очікування значення лічильника
    countDownLatch.await();
}
```

Все це дуже просто і ефективно щодо використання ресурсів. Часткового приросту продуктивності можна досягти з використанням методів що включені у інтерфейс `Stream`.

Розглянемо можливі подальші дії з опрацювання отриманих даних. Наприклад можна побудувати невеликий веб-сервіс, який буде функціонувати в асинхронному режимі.

Для цього можна використати наступні технології:

- База даних MongoDB. Оскільки не у всіх СУБД підтримується асинхронна робота. У MongoDB реалізовано весь потрібний функціонал для підтримки виконання поставлених завдань.
- Драйвер для асинхронної роботи з базою даних. Функціонування стандартного драйвера не передбачено у такому режимі.
- Spring Boot 2. На даний момент Spring Boot знаходиться на стадії розробки, але реліз заплановано на 18 грудня 2017р. Використання його полегшить роботу і зможе заощадити багато часу.
- Gradle. Система для збирання проекту [2].
- Java 8. Не остання на сьогодні версія Java, проте повністю підходить для реалізації поставленого завдання.
- Lombok. Для скорочення коду. Оскільки в Java-коді багато «бойлерплейта»[1]. Типові геттери, сеттери і конструктори, методи ініціалізації, методи toString, hashCode, equals, опрацювання винятків, які ніколи не будуть згенеровані, методи для закриття потоків, блоки синхронізації. Складність в підтримці бойлерплейта в актуальному стані полягає в необхідності внесення змін у коди окремих класів в процесі внесення змін до проекту. Одним із способів уникнути таких проблем є генерація коду, наприклад за допомогою використання методів проекту Lombok [5].

Перш за все потрібно під'єднати всі залежності. У gradle виглядати вони будуть так:

```
    compile('org.springframework.boot:spring-boot-starter-data-
 mongodb-reactive')
```

[1] Код, в якому не описується безпосередньо логіка роботи програми, але який необхідно писати

```
compile('org.springframework.boot:spring-boot-starter-webflux')
compileOnly('org.projectlombok:lombok')
testCompile('org.springframework.boot:spring-boot-starter-test')
testCompile('io.projectreactor:reactor-test')
```

Спочатку створимо клас для користувача програмного засобу:

```
@Data
@AllArgsConstructor
@NoArgsConstructor
@Document
public class User {
    @Id
    private String id;
    private String firstName;
    private String lastName;
}
```

Для успішної компіляції та роботи в IntelliJ Idea необхідно відмітити прапорець у пункті «enable annotation processing» або написати самостійно всі сеттери, гетери і конструктори.

Потім необхідно створити репозиторій з користувачами:

```
import org.faoxis.habrreactivemongo.domain.User;
import org.springframework.data.mongodb.repository
     .ReactiveMongoRepository;
public interface UserRepository extends
ReactiveMongoRepository<User, String> {}
```

Зауважимо, що відбувається успадкування від спеціального інтерфейсу для роботи в реактивному режимі. При детальному перегляді інтерфейсу ReactiveMongoRepository можна побачити, що в результаті викликів методу в систему повертаються об'єкти, декоровані класами Mono і Flux. Це означає, що за будь-якого звернення до бази даних результат не отримується відразу ж. Замість цього отримується потік даних, з якого можна отримати дані за готовністю чи доступністю.

На даний момент багатошарова архітектура є найбільш поширеним способом організації роботи мікросервісної архітектури.

Сервісний шар. Для цього потрібно описати відповідний інтерфейс:

```
public interface UserService {
    Flux<User> get();
    Mono<User> save(User user);
}
```

В силу простоти створюваного сервісу, можна відразу створити і одразу реалізувати кілька корисних методів:

```
@Service
@AllArgsConstructor
public class UserServiceImpl implements UserService {
    private UserRepository userRepository;
    @Override
    public Flux<User> get() {
        return userRepository.findAll();
    }
    @Override
    public Mono<User> save(User user) {
        return userRepository.save(user);
    }
}
```

Зауважимо, що впровадження залежності UserRepository відбувається через конструктор за допомогою анотації AllArgsConstructor. Починаючи з версії Spring 4 можна здійснювати автоматичне впровадження залежностей через конструктор без анотації Autowire.

Шар контролера:

```
import lombok.AllArgsConstructor;
import org.faoxis.habrreactivemongo.domain.User;
import org.faoxis.habrreactivemongo.service.UserService;
import org.springframework.web.bind.annotation.*;
import reactor.core.publisher.Flux;
import reactor.core.publisher.Mono;
@RestController
```

```
@RequestMapping("/users")
@AllArgsConstructor
public class UserController {
    private UserService userService;
    @PostMapping
    public Mono<User> post(@RequestBody User user) {
        return userService.save(user);
    }
    @GetMapping
    public Flux<User> get() {
        return userService.get();
    }
}
```

Після запуску додатка на виконання можна зробити POST запит на localhost:8080/users наступного змісту:

```
{
    "firstName": "Peter",
    "lastName": "Koval"
}
```

У відповідь буде надіслано такий самий об'єкт, але з присвоєним йому id:

```
{
    "id": "5a0bf0fdc48fd53478638c9e",
    "firstName": "Peter",
    "lastName": "Koval"
}
```

Для перевірки функціонування сервісу збережемо ще кілька користувачів і подивимось вміст бази даних. Для цього можна створити запит GET на localhost:8080/users. Результат опрацювання запиту може мати наступний вигляд:

```
[
    {
        "id": "5a0bf0fdc48fd53478638c9e",
        "firstName": "Peter",
        "lastName": "Koval"
    },
    {
        "id": "5a0bf192c48fd53478638c9f",
        "firstName": "Leonid",
        "lastName": "Tkach"
    },
    {
        "id": "5a0bf19ac48fd53478638ca0",
        "firstName": "Anton",
        "lastName": "Dybenko"
    }
]
```

Таким чином створений сервіс функціонує в асинхронному режимі. Щоб продемонструвати цей факт, створимо ще один метод для опрацювання URL методів в контролері:

```
@GetMapping("{lastName}")
    public    Flux<User>    getByLastName(@PathVariable(name    =
"lastName") String lastName) {
        return userService.getByLastName(lastName);
}
```

Таким чином отримується прізвище користувача і виводяться прізвища всіх користувачів з таким прізвищем.

Логіку реалізації можна записати через запити до бази даних. В сервісі таку реалізацію можна запрограмувати за допомогою методу:

```
@Override
    public Flux<User> getByLastName(final String lastName) {
        return userRepository
                .findAll()
                .filter(user ->
    user.getLastName().equals(lastName));
}
```

Оскільки є різниця в роботі з даними і потоками даних, то зазвичай здійснюються певні дії над даними безпосередньо. Але за даного підходу вказується, що слід зробити в потоці даних.

Якщо зробити GET запит за URL localhost:8080/users/, то буде отримано наступний результат:

```
[
    {
        "id": "5a0bf0fdc48fd53478638c9e",
        "firstName": "Peter",
        "lastName": "Koval"
    },
    {
        "id": "5a0bf192c48fd53478638c9f",
        "firstName": "Leonid",
        "lastName": "Tkach"
    },
    {
        "id": "5a0bf19ac48fd53478638ca0",
        "firstName": "Anton",
        "lastName": "Dybenko"
    }
]
```

Таким чином можна побудувати асинхронний сервіс з використанням нового фреймворку SpringBoot 2.0 а також технологій WebFlux і сервером Netty.

Висновки. Через написання додатків з використанням фреймворків на основі реактивного програмування може бути підвищено інтерес студентів до програмування. Розробка програмних засобів з використанням подібних технологій спрощує написання програм, не зменшуючи їх функціонального призначення. Також підвищується зацікавленість студентів до дослідження функціонування та написаннях свої власних фреймоворків.

Список використаних джерел

1. Burchett K. Lowering: A Static Optimization Technique for Transparent Functional Reactivity * [Електронний ресурс] / K. Burchett, G. H. Cooper, S. Krishnamurthi – Режим доступу до ресурсу: https://cs.brown.edu/~sk/Publications/Papers/Published/bck-lowering-opt-trans-frp/paper.pdf.G. H. Cooper and S. Krishnamurthi. Embedding dynamic dataflow in a call-by-value language. In European Symposium on Programming, pages 294–308, 2006.
2. Gradle build tool [Електронний ресурс] – Режим доступу до ресурсу: https://gradle.org/.
3. Java Platform Standard Edition 8 Documentation [Electronic resource] – Mode of access: https://docs.oracle.com/javase/8/docs/.
4. Model-View-Controller and the "Observer" Pattern [Electronic resource] – Mode of access: http://peak.telecommunity.com/DevCenter/Trellis#model-view-controller-and-the-observer-pattern.
5. Project Lombok [Electronic resource] – Mode of access: https://projectlombok.org/.
6. Spring Boot [Electronic resource] – Mode of access: https://projects.spring.io/spring-boot/.

РЕАКТИВНОЕ ПРОГРАММИРОВАНИЕ

Ю.П. Беляй

Аннотация. *В данной статье рассмотрен процесс создания микросервиса на основе современных технологий разработки программного обеспечения для повышения заинтересованности студентов обучению программированию.*

Ключевые слова: *реактивное программирования, парадигма программирования, создание микросервисов, обучение программированию.*

REACTIVE PROGRAMMING

Yu.P. Biliai

Resume. *This article discusses the process of creating a micro service, based on modern software development technologies to increase the interest of students in programming.*

Keywords: *reactive programming, programming paradigm, creating of micro services, programming training.*