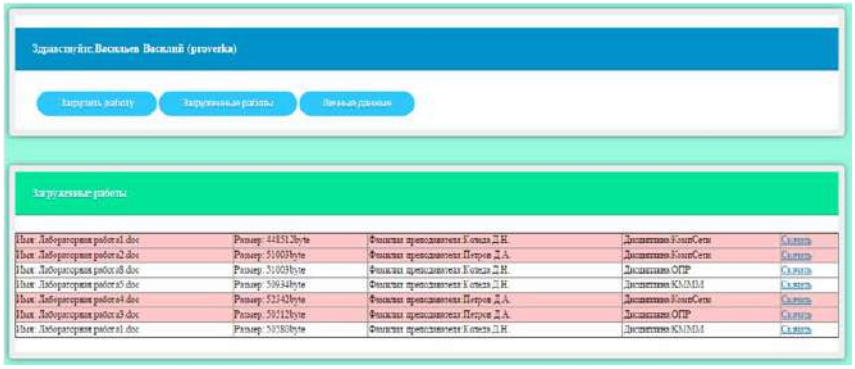

Рисунок 1 — Вид интерфейса электронного ресурса
с информацией загруженных работ учащегося

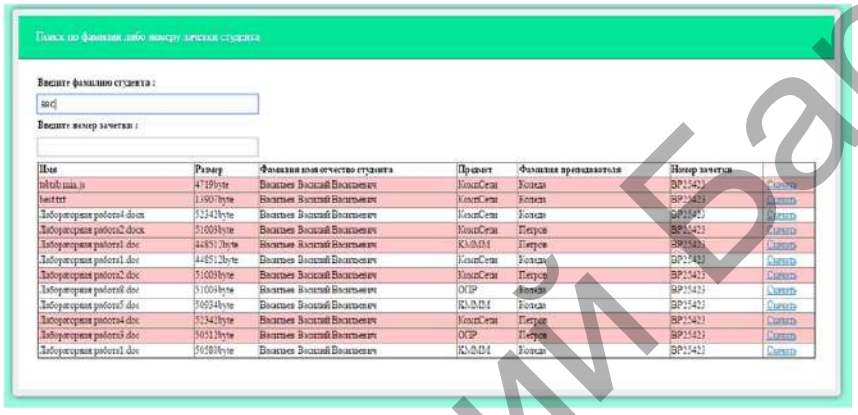

Рисунок 2 — Вид интерфейса электронного
ресурса для поиска работ на проверку

Заключение. Внедрение электронного ресурса дистанционного контроля качества выполненных работ по учебным дисциплинам в систему образования позволяет значительно повысить эффективность сдачи и защиты практических работ. Внедрение элементов дистанционных форм обучения востребовано и актуально в современном мире. Результаты и опыт внедрения электронного ресурса могут быть использованы педагогами в учебном процессе.

Список цитируемых источников

1. *Литвинская, И. Г.* К вопросу о формах организации обучения / И. Г. Литвинская // Коллективный способ обучения : науч.-метод. журн. — 2007. — № 9. — С. 36—47.
2. Из истории развития форм и организации обучения [Электронный ресурс]. — Режим доступа: http://paidagogos.com/?p=87/ . — Дата доступа: 10.09.2013.
3. *Акимова, М. К.* Индивидуальность учащихся и индивидуальный подход / М. К. Акимова, В. П. Козлова. — М., 2002. — 286 с.

УДК 004.422

Ю. В. Корьевиков, К. С. Лукашевич, М. В. Бовкунович
Учреждение образования «Барановичский государственный университет», Барановичи

ИНВЕРСИЯ УПРАВЛЕНИЯ ЗАВИСИМОСТЯМИ

Введение. Инверсия управления (англ. Inversion of Control, IoC) — один из важнейших парадигм (принципов) в современном объектно ориентированном программировании, который используется для уменьшения связности компонентов компьютерных программ, способствуя большей гибкости кода приложений; архитектурное решение внедрения, которое облегчает расширение возможности приложения, при котором контроль

над потоком управления программы остается за каркасом. Инверсия управления решает очень простую, но и очень важную задачу при разработке приложений — уменьшает зависимость между частями системы. Приложение зависит только от некоторой абстракции, сама реализация системы скрыта, в любой момент разработчик может заменить ее на нужную для себя.

Целью данной статьи является рассмотрение основных возможностей использования контроля зависимости между компонентами приложения.

Основная часть. Любое нетривиальное приложение состоит из двух или более классов, которые взаимодействуют друг с другом, реализуя некоторую логику.

Обычно каждый объект ответственен за получение собственных ссылок на объекты, с которыми он взаимодействует (его зависимости). Это может привести к сильной связанности [1].

Основные недостатки сильно связанного кода: проблематично использовать повторно, сложности в понимании, в тестировании, в сопровождении.

С другой стороны, совершенно не связанный код будет делать мало полезного. При выполнении существенных задач классы должны знать друг о друге. Связанность необходима, но должна тщательно контролироваться [1].

Данную проблему позволяет решить один из важных принципов объектно ориентированного программирования — IoC, используемый для уменьшения влияния компонентов в компьютерных программах друг на друга. Также IoC является архитектурным решением интеграции, упрощающим расширение возможностей системы, при котором контроль над ходом управления программы остаётся за интерфейсом. По своему предназначению инверсия управления направлена на то, чтобы предложить простой механизм для предоставления зависимостей компонента (часто называемыми каллабораторами) и управления зависимостями компонентов на протяжении всего их жизненного цикла. Компонент, который требует определенных зависимостей, именуют зависимым объектом, а в случае инверсии управления — целевым объектом.

В настоящее время существуют следующие реализации IoC: шаблон «Фабрика», service locator, внедрение зависимости, контекстный поиск.

Рассмотрим основные формы инверсии управления: внедрение зависимостей и service locator.

Механизм внедрения зависимостей основан на предоставлении объекту его зависимостей извне, а не на их приобретении самим объектом. В полном соответствии с принципом единственной ответственности объект отдаёт заботу о построении требуемых ему зависимостей внешнему, специально предназначенному для этого общему механизму.

Далее представим более детально работу механизма внедрения зависимостей. Приложение имеет объект-ассемблер, который создаёт объект с соответствующей реализацией и внедряет его в поле другого объекта, у которого тип общего интерфейса или конкретной реализации (рисунок 1). Информация о необходимой реализации, например, находится в конфигурационном файле (XML, YML), что позволяет подставлять фиктивные реализации и упрощать Unit-тестирование.

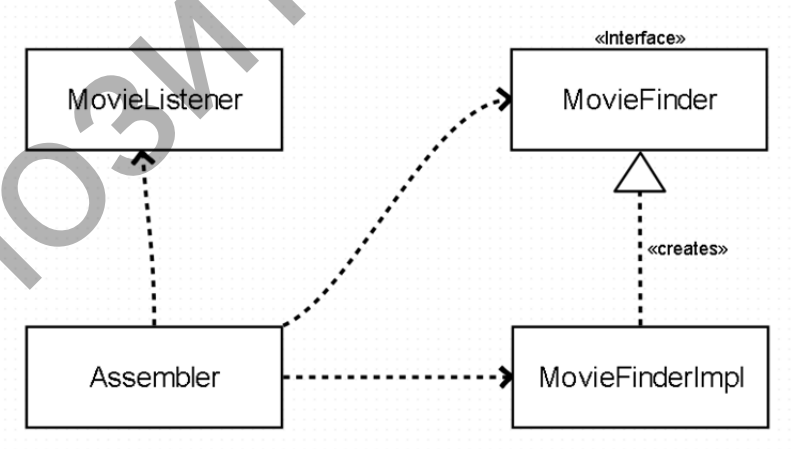

Рисунок 1 — Реализация внедрения зависимости

Внедрение зависимостей (DI, Dependency Injection) — это механизм передачи классу его зависимостей. Существует несколько конкретных видов или паттернов внедрения зависимостей: через конструктор (Constructor Injection), метод класса (Method Injection), интерфейс внедрения (Interface Injection).

Разные виды внедрения зависимостей предназначены для своего спектра задач. Внедрение зависимостей через конструктор — передаются обязательные зависимости класса, без которых работа класса невозможна. Посредством внедрения через метод передаются зависимости, которые используются лишь одним методом, а не целиком всем классом. Внедрение через интерфейс используется в большинстве случаев при разработке приложений.

Service locator — это шаблон проектирования, использующийся для разработки программного обеспечения, которое инкапсулирует процессы, связанные с получением сервиса с сильным слоем абстракции. Основной идеей является то, что приложение должно иметь объект, который знает, как получить все объекты, которые могут понадобиться (рисунок 2). Вместо создания конкретных объектов непосредственно через new, используется «фабричный» объект, который будет распоряжаться и отвечать за создание всех сервисов. Его назначение — разорвать жесткую связь между классом и его зависимостями посредством добавления специального класса-посредника (локатор).

Существует две версии реализации этого паттерна. Локатор сам по себе может быть синглтоном (в классическом виде или в виде класса с набором статических методов), тогда доступ к нему может производиться из любой точки в коде. Или же локатор может передаваться требуемым классам через конструктор или свойство в виде объекта класса или интерфейса.

Оба эти подхода страдают от одних и тех же недостатков, но в первом случае все нижеперечисленные проблемы усиливаются, поскольку в этом случае совершенно любой класс приложения может быть завязан на любой «сервис» [2].

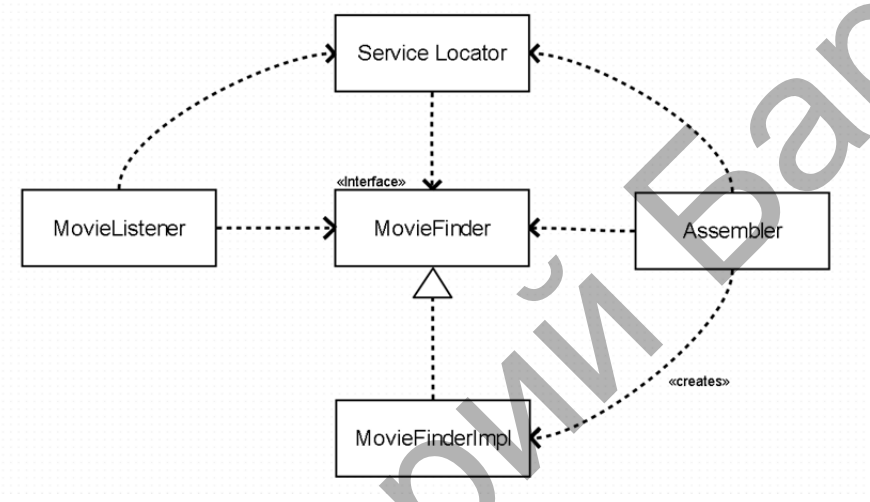

Рисунок 2 — Реализация service locator

Основное различие заключается в том, что с локатором каждый пользователь сервиса имеет зависимость. Локатор может скрыть зависимости от других реализаций, но необходимо видеть локатор. Таким образом, решение между локатором и DI зависит от того, является ли зависимость проблемой.

Заключение. Инверсия управления — один из важнейших принципов в реалиях современной разработки программного обеспечения. Проблема «жесткого» связывания кода одна из краеугольных проблем разработки программного обеспечения, где каждый масштабный продукт состоит из тысяч компонентов. IoC позволяется менять зависимости между компонентами «на лету», что экономит время разработчика. В основу многих технологий положен принцип инверсии контроля, один из примеров — Java-фраемворк Spring, на котором написано большое количество программного обеспечения. Каждый современный разработчик должен иметь четкое представление о работе этого принципа для оптимизации своей деятельности.

Список цитируемых источников

1. *Уоллс, К.* Spring in action / К. Уоллс. — М. : ДМК Пресс, 2013. — 752 с.: ил.
2. DI Паттерны. Service locator [Электронный ресурс]. — Режим доступа: http://sergeyteplyakov.blogspot.com.by/2013/03/di-service-locator.html . — Дата доступа: 01.03.2017.