Изображение выглядит как текст

Автоматически созданное описание\

СОДЕРЖАНИЕ

[Введение 4](#_Toc103800731)

[1 Язык программирования Java 6](#_Toc103800732)

[1.1 Причины выбрать Java для изучения 6](#_Toc103800733)

[1.2 Направление, где используется Java 8](#_Toc103800734)

[2 Темы для видео-уроков 10](#_Toc103800735)

[2.1 Классы 10](#_Toc103800736)

[2.2 Конструкторы 11](#_Toc103800737)

[2.3 Значения в конструкторе 13](#_Toc103800738)

[2.4 Getters и Setters 15](#_Toc103800739)

[2.5 Интерфейсы 18](#_Toc103800740)

[2.6 Collections and Generics 22](#_Toc103800741)

[2.6.1 Collections 22](#_Toc103800742)

[2.6.2 Интерфейс коллекции 24](#_Toc103800743)

[2.6.3 Типы коллекций 26](#_Toc103800744)

[2.6.4 Интерфейс Map 29](#_Toc103800745)

[3 Среда создания видео 33](#_Toc103800746)

[3.1 Создание нового проекта и импорт медиафайлов 33](#_Toc103800747)

[3.2 Добавление обрезанных клипов на временную шкалу 34](#_Toc103800748)

[3.3 Добавление текста и титров 35](#_Toc103800749)

[3.4 Добавление переходов 36](#_Toc103800750)

[3.5 Добавление эффектов 36](#_Toc103800751)

[3.6 Добавление музыки и аудио 37](#_Toc103800752)

[3.7 Коррекция цвета 38](#_Toc103800753)

[3.8 Поделитесь своим фильмом 38](#_Toc103800754)

[Заключение 40](#_Toc103800755)

[Список используемой литературы 42](#_Toc103800756)

[Приложение А 43](#_Toc103800757)

[Приложение Б 44](#_Toc103800758)

[Приложение В 45](#_Toc103800759)

[Приложение Г 46](#_Toc103800760)

# **ВВЕДЕНИЕ**

Данная курсовая работа продолжение работы «Видеоуроки по программированию на языке Java», в ней нет главы «Сравнение между IntelliJ Idea и Eclipse», так как нет подробности писать в этой работе.

В данной работе будут изучаться темы: классы, конструкторы, getters, setters, интерфейсы, коллекции и дженерики.

Данные темы в этой работе дадут вам возможность с уверенностью и долей удачи пойти на стажировку в компании, где вам дадут опыт в программирование и базу знаний для дальнейшего развития себя в профессиональной деятельности.

Основной закон об образовании гласит: "Под дистанционными образовательными технологиями понимаются образовательные технологии, реализуемые преимущественно с использованием информационных и

телекоммуникационных сетей при опосредованном (на расстоянии)

взаимодействии обучающихся и преподавателей".

Дистанционное обучение – это взаимодействие обучающихся и преподавателей на расстоянии, показывающие компоненты, присущие процессу обучения (цели, содержание, методы, алгоритмы, средства обучения) и реализуемое с помощью специальных интернет-технологий или других средств, обеспечивающих интерактивность.

Актуальность дистанционного обучения очевидна: русский педагог А. С. Сеидов объясняет растущую популярность этой формы обучения тем, что в России только 51% студентов университетов моложе 24 лет и лишь четверть в возрасте 18–22 лет. Остальные – взрослые люди и заняты семейными и рабочими делами. Очная форма обучения в университете не всегда удобна для них. Дистанционное образование также является экономически эффективным, если принять во внимание транспортные и организационные расходы очной системы. По прогнозам ученых США ученики средней и старшей школы, студенты университетов будут проводить на очных занятиях лишь 20–30% своего времени, 70% будет приходиться на дистанционное обучение, а остальное – на самообразование. Отсюда понятен растущий интерес к дистанционному обучению – в университете, в школе, в системе дополнительного образования, т. е. на всех уровнях обучения в течение всей жизни.

Можно говорить, что дистанционное обучение открывает новые возможности, значительно расширяя как информационное пространство, так и информационное поле образования.

Средством, которое помогает эффективности дистанционного обучения, могут стать видео-уроки по программированию на языке Java.

1. **Язык программирования Java**

Java – один из самых распространенных и надежных языков для разработки приложений и программного обеспечения. Он имеет широкий спектр различных инструментов, которые могут значительно облегчить этот процесс [1].

Java – это объектно-ориентированный язык программирования. Его можно использовать для написания программного обеспечения на стороне сервера, программ для персональных компьютеров и мобильных приложений для различных целей и задач. Давайте рассмотрим причины, по которым вы должны выбрать Java в качестве языка программирования.

**1.1 Причины выбрать Java для изучения**

Рассмотрим причины выбрать Java для изучения:

1) Сильное Java-сообщество – по данным Java, в мире насчитывается 5 миллионов студентов и 12 миллионов разработчиков языка – от авторов библиотек до целых компаний. Здесь даже новичок может легко найти готовые куски кода для решения своих задач или ответы на любые вопросы.

Сообщество организовано и активно участвует в жизни языка, развивает его, разрабатывает множество библиотек и инструментов, в том числе для разработки серверной части. Благодаря сообществу экосистема Java продолжает развиваться и легко адаптируется к современным требованиям;

2) Разнообразие инструментов – Java имеет большое количество библиотек для решения общих задач, API, сред разработки и фреймворков для создания программ и приложений различной степени сложности. Возможности этого языка постоянно обновляются;

Множество инструментов Java для разработки на стороне сервера означает, что программы могут быть написаны быстрее, с меньшим количеством потенциальных ошибок и проблем, и быстрее выходят на рынок. И эта особенность языка делает его привлекательным для корпоративных приложений.

3) Виртуальная машина Java – это не только язык программирования, но и среда выполнения. Она позволяет писать достаточно высокопроизводительное программное обеспечение. Многие языки программирования используют Java runtime в качестве среды выполнения. Например, Groovy, Clojure, Scala, Kotlin, а также есть версии Python, JRuby, работающие на JVM. Создатели этих языков выбрали виртуальную машину Java из-за ее производительности, кроссплатформенности и возможности использовать множество библиотек и инструментов Java;

4) Кроссплатформенность Java – является кроссплатформенным языком. Недаром один из главных лозунгов этого языка до сих пор – Write once, run anywhere. Он позволяет разработчику создавать переносимый код, то есть написать программу один раз и без изменений выполнять ее в нескольких операционных системах (Windows, Linux и MacOS). JVM позволяет языку достичь такой гибкости;

5) Обратная совместимость – стремление Java к обратной совместимости, с одной стороны, тормозит его рост и развитие, но с другой – является неоспоримым преимуществом, особенно для корпоративных приложений. Эта характеристика дает ощущение стабильности и уверенности в том, что с каждой новой версией языка приложение будет работать;

6) Многочисленные способы изучения Java – сегодня существует множество инструментов для изучения языка. На рынке представлено огромное количество соответствующего материала: книги, статьи, семинары, конференции, курсы;

Но часто бывает так, что специалисту просто не хватает времени или понимания, чтобы сделать первый шаг в той или иной области. Получить базу по определенной теме или вектор движения можно на специализированных курсах и тренингах. Это будет полезно не только для профессионалов, которые хотят изучить возможности новой версии, API или библиотеки. Для начинающих разработчиков, которые только входят в профессию и хотят быть востребованными на рынке и построить успешную карьеру, такой способ получения знаний будет более предпочтительным.

7) Возможность быть востребованным на рынке – роль автоматизации в бизнесе растет. В то же время на рынке труда ощущается нехватка рабочей силы. Согласно отчету CompTIA, в декабре 2020 года наем ИТ-специалистов в США увеличился на 391 000 человек, при этом наибольшая доля новых предложений пришлась на разработчиков программного обеспечения и приложений – 62 900 человек;

По данным главы Российской ассоциации электронных коммуникаций С. А. Плуготаренко, с 24 февраля по 22 марта Россию покинули 50–70 тыс. IT-специалистов. Реагируя на это, правительство РФ приняло ряд мер поддержки для IT-сферы и специалистов, в частности. Не уезжать IT-кадры просил лично премьер-министр М. В. Мишустин [3].

М. В. Мишустин так же подписал несколько приказов о льготах и помощи в поисках работы студентам. Что позволяет молодым IT-специалистам легко найти работу и поддержку от правительства для быстрого продвижения по карьерной лестнице.

Согласно индексу сообщества программистов TIOBE, в апреле 2022 года Java занимает третье место, уступая только языку C и Python. Напомню, что этот индекс широко используется для проверки актуальности ваших навыков программирования или для помощи в принятии стратегического решения.

## **Направление, где используется Java**

Java – это язык, который используется для серверных приложений в крупных корпорациях. Особенно часто Java используется в банках, страховых компаниях, розничных сетях т. д. Например, такие банки, как Deutsche Bank, Citigroup, Barclays, Goldman Sachs и многие-многие другие используют Java для написания бэк-энд и фронт-энд офисных электронных систем и т. д. Еще пример: серверная часть Google+ написана на Java. Примеры типов приложений , написанных на языке Java:

1) вэб-приложения – широкое применение Java нашла в электронной коммерции и в вэб-приложениях. Например, если говорить о рынке Европы, США, многие вэб-приложения госучреждений, страховых, образовательных, оборонительных учреждений написаны на Java;

2) трейдинговые приложения – например, LMAX, Murex;

3) Android приложения – если у Вас телефон на Android, то все приложения в нём написаны на Java с использованием Google и Android API;

4) Десктопные приложения, программные средства и средства разработки – например, Eclipse, Netbeans IDE, jEdit (Programmer`s Text Editor), jDownloader (open-source download management tool), Azureus/Vuse;

5) Встраиваемые системы – Java используется на смарт-картах и сенсорах. Например, в банковских карточках используется Java;

Java все еще остается прогрессивным и востребованным языком программирования. Он универсален, его несложно освоить и, самое главное, для этого созданы все условия [1].

# **2 Темы для видео-уроков**

## 

## **2.1 Классы**

Обычная причина для определения нового класса – это инкапсуляция связанных данных в объект, который может рассматриваться как единое целое. Таким образом, мы можем использовать объекты в качестве параметров и возвращаемых значений, а не передавать и возвращать несколько значений.

Пример, который будет реализован – это класс Time, представляющий время суток. Данные, заключенные в объекте Time, включают час, минуту и число секунд. Поскольку каждый объект Time содержит эти значения, определяем атрибуты для их хранения.

Атрибуты также называются переменными экземпляра, поскольку каждый экземпляр имеет свои собственные переменные.

Первый шаг – решить, какого типа должна быть каждая переменная. Кажется очевидным, что час и минута должны быть целыми числами. Чтобы было интереснее, сделаем секунду типа doble (переменная плавающей точкой).

Переменные экземпляра объявляются в начале определения класса, вне любого метода. Сам по себе этот фрагмент кода является законным определением класса, показанный на рисунке 1:

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 1

Класс Time является публичным, что означает, что его можно использовать в других классах. Но переменные экземпляра являются приватными, что означает, что доступ к ним возможен только из класса Time. Если попытается прочитать или записать их из другого класса, получим ошибку компилятора.

Частные переменные экземпляра помогают изолировать классы друг от друга, так что изменения в одном классе не потребуют изменений в других классах. Это также упрощает то, что нужно знать другим программистам, чтобы использовать ваши классы. Такая изоляция называется сокрытием информации.

## **2.2 Конструкторы**

После объявления переменных экземпляра следующим шагом является определение конструктора, который представляет собой специальный метод, который инициализирует объект. Синтаксис конструкторов похож на синтаксис других методов, за исключением следующего:

– имя конструктора совпадает с именем класса;

– конструкторы не имеют возвращаемого типа (и возвращаемого значения);

– ключевое слово static опускается.

Вот пример конструктора для класса Time, показанный на рисунке 2:

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 2

Этот конструктор не принимает никаких аргументов. Каждая строка инициализирует переменную экземпляра в 0 (это полночь для объекта Time).

Имя this – это ключевое слово, обозначающее созданный объект. Можем использовать его так же, как и имя любого другого объекта. Например, можем читать и записывать переменные экземпляра this, а также передавать его в качестве аргумента в других методах. Но не объявив this, не можем присвоить ему имя.

Распространенной ошибкой при написании конструкторов является размещение в конце оператора return. Как и методы void, конструкторы не возвращают значения.

Чтобы создать объект Time, необходимо использовать оператор new, как показано на рисунке 3:

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3

Когда используем new, Java создает объект и вызывает конструктор для инициализации переменных экземпляра. Когда конструктор завершает работу, new возвращает ссылку на новый объект. В этом примере ссылка присваивается переменной time, которая имеет тип Time. На рисунке 4

показан результат.

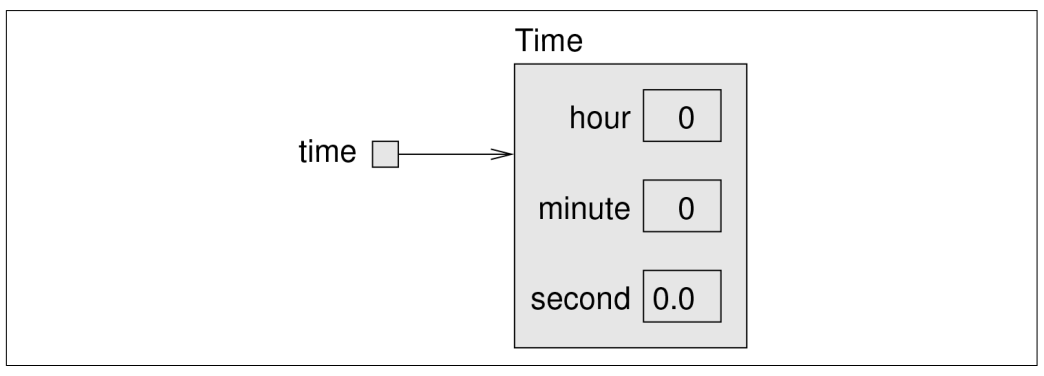


Рисунок 4

Новички иногда допускают ошибку, используя new в конструкторе, пример на рисунке 5:

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 5

Это приводит к бесконечной рекурсии, так как new вызывает тот же конструктор, который снова использует new, который снова вызывает конструктор и так далее.

## **2.3 Значения в конструкторе**

Как и другие методы, конструкторы могут быть перегружены, что означает, что можем предоставить несколько конструкторов с различными параметрами. Java знает, какой конструктор следует вызывать, сопоставляя аргументы, которые предоставляются, с параметрами конструктора.

Обычно предоставляется как конструктор по умолчанию, который не принимает никаких аргументов, как в предыдущем случае, так и конструктор значений, как этот показано на рисунке 6:

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 6

Чтобы вызвать этот конструктор, необходимо предоставить аргументы оператору new. Следующий пример на рисунке 7 создает объект Time, представляющий долю секунды до полудня:



Рисунок 7

Перегрузка конструкторов дает возможность сначала создать объект, а затем заполнить его атрибутами, или собрать всю информацию до создания самого объекта.

Как только освоитесь, писать конструкторы станет скучно. Можем быстро написать их, просто взглянув на список переменных экземпляра. Более того, некоторые IDE могут генерировать их за вас.

Вот полное определение класса на данный момент на рисунке 8:

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 8

Обратите внимание, как второй конструктор объявляет параметры hour, minute и second. Java позволяет объявлять параметры (и локальные переменные) с теми же именами, что и переменные экземпляра. Они не обязаны использовать те же имена, но это обычная практика. практика.

Правая часть this.hour = hour; относится к параметру hour, так как он был объявлен самым последним. Такая ситуация называется теневой, потому что параметр "скрывает" переменную экземпляра с тем же именем.

Java предоставляет ключевое слово this, чтобы могли получить доступ к переменным экземпляра, независимо от затенения. В результате этот конструктор копирует значения из параметров в переменные экземпляра.

## **2.4** **Getters и Setters**

Вспомним, что переменные экземпляра Time являются приватными. Можем получить к ним доступ изнутри класса Time, но, если попытаться прочитать или записать их из другого класса, компилятор сообщит об ошибке.

Класс, который использует объекты, определенные в другом классе, называется клиентом. Например, на рисунке 9, здесь новый класс под названием TimeClient:

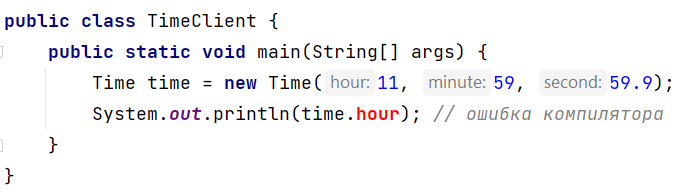


Рисунок 9

Если скомпилировать этот код, то получим сообщение об ошибке типа hour имеет приватный доступ в Time. Есть три способа решить эту проблему:

– сделать переменные экземпляра общедоступными;

– предоставить методы для доступа к переменным экземпляра;

– решить, что это не проблема, и не давать другим классам доступ к переменным экземпляра.

Первый вариант привлекателен тем, что он прост. Но вот в чем проблема: когда класс A получает прямой доступ к переменным экземпляра класса B, A становится зависимым от B. Если что-то в B изменится позже, вероятно, что и A придется изменить.

Но если A использует только методы для взаимодействия с B, A и B менее зависимы, что означает, что можем вносить изменения в B, не затрагивая A (при условии, что не меняем параметры метода). Поэтому обычно избегают делать переменные экземпляра общедоступными.

Второй вариант – предоставить методы, которые обращаются к переменным экземпляра. Например, можем захотеть, чтобы переменные экземпляра были доступны только для чтения; то есть код в других классах должен иметь возможность читать их, но не записывать. Можем сделать это, предоставив один метод для каждой переменной экземпляра, как показано на рисунке 10:

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 10

Методы, подобные этим, формально называются асессорами, но чаще их называют getters. По традиции, метод, который получает переменную с именем something, называется getSomething.

Можем исправить ошибку компилятора в TimeClient, используя getter, на примере рисунка 11:

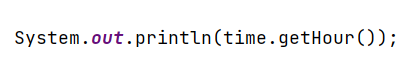


Рисунок 11

Если решим, что TimeClient также должен иметь возможность изменять переменные экземпляра Time, можем предоставить методы и для этого, например, как на рисунке 12:

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 12

Эти методы формально называются мутаторами, но чаще известны как setters.

Соглашение об именовании аналогично; метод, который устанавливает что-то, обычно называется setSomething.

Написание getters и setters может стать скучным занятием, но многие IDE могут генерировать их за вас на основе переменных экземпляра.

## **2.5** **Интерфейсы**

Java расширяет концепцию абстрактных методов с помощью интерфейсов. Часто бывает желательно указать группу абстрактных методов, определяющих определенное поведение объекта, не привязывая его к какой-либо реализации. В Java это называется интерфейсом. Интерфейс определяет набор методов, которые должен реализовать класс. Класс в Java может объявить, что он реализует интерфейс, если он реализует требуемые методы. В отличие от расширения абстрактного класса, класс, реализующий интерфейс, не обязан наследоваться от какой-либо определенной части иерархии наследования или использовать конкретную реализацию.

Интерфейсы подобны значкам отличия бойскаутов или девочек-скаутов. Скаут, научившийся строить скворечник, может носить на рукаве маленькую нашивку с его изображением. Это говорит всему миру: "Я знаю, как построить скворечник". Аналогично, интерфейс interface – это список методов, которые определяют некоторый набор поведения для объекта. Любой класс, который реализует каждый метод, перечисленный в интерфейсе, может объявить во время компиляции, что он реализует интерфейс и носить, как знак отличия, дополнительный тип – тип интерфейса.

Типы интерфейсов действуют подобно типам классов. Можем объявить переменные как типы интерфейса, можем объявить аргументы методов, чтобы они принимали интерфейсные типы, и можем указать, что возвращаемый тип метода является типом интерфейса. В каждом случае имеется в виду, что что любой объект, реализующий интерфейс (т. е. носящий нужный значок), может исполнять эту роль. В этом смысле интерфейсы ортогональны к иерархии классов. Они границы того, каким объектом является элемент, и рассматривают его только с точки зрения того. того, что он может делать. Класс может реализовать столько интерфейсов, сколько пожелает. Таким образом, интерфейсы в Java заменяют большую часть необходимости множественного наследования в других языках.

языках (и все его грязные сложности).

Интерфейс выглядит, по сути, как чисто абстрактный класс (т. е. класс, имеющий только абстрактными методами). Определяете интерфейс с помощью ключевого слова interface, и перечисляете его методы без тел, только прототипы (сигнатуры), пример показан на рисунке 13:

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 13

В предыдущем примере определен интерфейс Driveable с четырьмя методами. Допустимо, но не обязательно, объявлять методы в интерфейсе с модификатором abstract. Более важно то, что методы интерфейса всегда считаются публичными, и можем по желанию объявить их таковыми.

Почему публичными? В противном случае пользователь интерфейса не обязательно сможет их увидеть, а интерфейсы обычно предназначены для описания поведения объекта, а не его реализации.

Интерфейсы определяют возможности, поэтому принято называть интерфейсы по их возможностям. Driveable, Runnable и Updateable – хорошие названия интерфейсов. Любой класс, который реализует все методы, может затем объявить, что он реализует интерфейс, используя специальный пункт implements в определении класса. Например, как на рисунке 14:

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 14

Здесь класс Automobile реализует методы интерфейса Driveable и объявляет себя типом Driveable с помощью ключевого слова implements.

Как показано на рисунке в примере, другой класс, например Lawnmower, также может реализовать интерфейс Driveable. На рисунке 15 показано, что интерфейс Driveable реализован двумя разными классами. Хотя вполне возможно, что и Automobile, и Lawnmower могут произойти от какого-то примитивного вида транспортного средства, в данном сценарии это не обязательно.

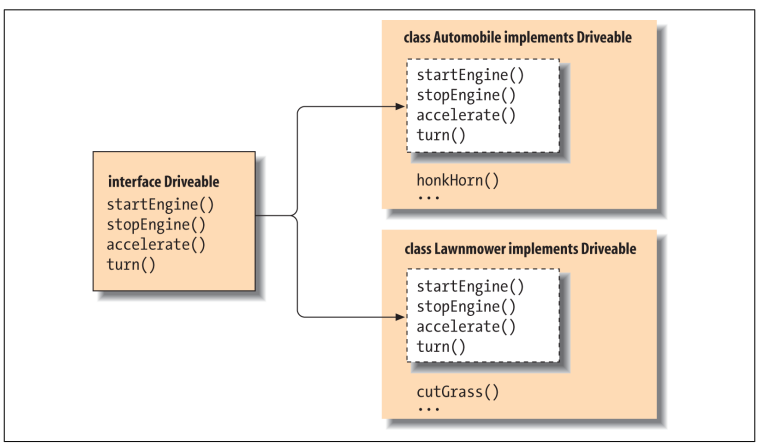


Рисунок 15

После объявления интерфейса появился новый тип Driveable. Можем объявить переменные типа Driveable и присвоить им любой экземпляр объекта Driveable, как на рисунке 16:

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 16

И Automobile, и Lawnmower реализуют Driveable, поэтому их можно считать взаимозаменяемыми объектами этого типа.

## **2.6** **Collections and Generics**

По мере того, как начинаем использовать наши растущие знания об объектах для решения все более интересных задач, возникает один повторяющийся вопрос. Как хранить данные, которыми манипулируем в процессе решения этих задач? Определенно будем использовать переменные всех различных типов, также понадобятся более крупные, более причудливые варианты хранения данных. Массивы, о которых говорили в главе "Массивы", – это начало, но у массивов есть некоторые ограничения. В этой главе рассмотрим, как получить эффективный, гибкий доступ к большим объемам данных. В этом поможет Java Collections API. Также увидим, как работать с различными типами данных, которые можем хранить в этих больших контейнерах, как это реализовать с отдельными значениями в переменных.

### 2.6.1 Collections

Коллекции – это структуры данных, которые являются основополагающими для всех видов программирования. Всякий раз, когда нужно обратиться к группе объектов, имеем какую-либо коллекцию. На уровне основного языка Java поддерживает коллекции в виде массивов. Но массивы статичны, и поскольку они имеют фиксированную длину, они неудобны для групп объектов, которые растут и уменьшаются в течение жизни приложения. Массивы также не очень хорошо представляют абстрактные отношения между объектами. На заре платформы Java для решения этих задач было всего два основных класса: класс java.util.Vector, представляющий динамический список объектов, и класс java.util.Hashtable, хранящий карту пар ключ/значение. Сегодня в Java существует более комплексный подход к коллекциям, называемый Collections Framework. Старые классы все еще существуют, но они были встроены во фреймворк (с некоторыми эксцентричностями) и, как правило, больше не используются.

Несмотря на концептуальную простоту, коллекции являются одной из самых мощных частей любого языка программирования. Коллекции реализуют структуры данных, которые лежат в основе управления сложными проблемами. Большая часть фундаментальной информатики посвящена описанию наиболее эффективных способов реализации определенных типов алгоритмов над коллекциями. Имея в своем распоряжении эти инструменты и понимая, как их использовать, можем сделать свой код намного меньше и быстрее. Это также может избавить от необходимости заново изобретать колесо.

Оригинальный фреймворк Collections Framework имел два основных недостатка. Первый заключался в том, что коллекции по необходимости были не типизированными и работали только с недифференцированными объектами, а не с конкретными типами, такими как даты и строки. Это означало, что программисты должны были выполнять приведение типов каждый раз, когда брали объект из коллекции. Это противоречило безопасности типов, обеспечиваемой Java во время компиляции. Но на практике это было не столько проблемой, сколько просто громоздко и утомительно. Вторая проблема заключалась в том, что по практическим причинам коллекции могли работать только с объектами, а не с примитивными типами. Это означало, что каждый раз, когда хотели поместить число или другой примитивный тип в коллекцию, приходилось сначала хранить его в классе-обертке, а затем распаковывать при извлечении. Сочетание этих факторов делало код, работающий с коллекциями, менее читабельным и более опасным.

Все изменилось с введением общих типов и autoboxing примитивных значений. Во-первых, введение общих типов позволило сделать так, чтобы действительно безопасные с точки зрения типов коллекции находились под контролем программиста. Во-вторых, введение autoboxing и unboxing примитивных типов означает, что в коллекциях можно рассматривать объекты и примитивы как равные. Сочетание этих новых возможностей может значительно сократить объем написанного кода и добавить безопасности. Как увидим, все классы коллекций теперь используют эти возможности.

Структура коллекций основана на нескольких интерфейсах из пакета java.util. Эти интерфейсы разделены на две иерархии. Первая иерархия восходит к интерфейсу Collection. Этот интерфейс (и его потомки) представляет собой контейнер, в котором хранятся другие объекты. Вторая, отдельная иерархия основана на интерфейсе Map, который представляет собой группу пар ключ/значение, где ключ может быть используется для извлечения значения эффективным способом.

### 2.6.2 Интерфейс коллекции

Матерью всех коллекций является интерфейс с соответствующим названием Collection. Он служит в качестве контейнера, в котором хранятся другие объекты, его элементы. Он не определяет, как именно организованы объекты; например, он не говорит, разрешены ли дубликаты объектов или они каким-либо образом упорядочены. Такие детали оставлены для дочерних интерфейсов. Тем не менее, интерфейс Collection определяет некоторые базовые операции, общие для всех коллекций, например, один из них на рисунке 17:

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 17

Добавляет указанный объект в данную коллекцию. Если операция прошла успешно, этот метод возвращает true. Если объект уже существует в данной коллекции, а коллекция не допускает дубликатов, возвращается false. Кроме того, некоторые коллекции доступны только для чтения. Такие коллекции выбрасывают исключение UnsupportedOperationException, если вызывается этот метод, который показан на рисунке 18.



Рисунок 18

Удаляет указанный объект из данной коллекции. Как и метод add(), этот метод возвращает true, если объект удален из коллекции. Если объект не существует в этой коллекции, возвращается false. Коллекции, доступные только для чтения, выбрасывают сообщение UnsupportedOperationException при вызове этого метода, который показан на рисунке 19.



Рисунок 19

Возвращает true, если коллекция содержит указанный объект, пример на рисунке 20.



Рисунок 20

Возвращает количество элементов в данной коллекции, пример на рисунке 21.



Рисунок 21

Возвращает true, если в данной коллекции нет элементов, пример на рисунке 22.



Рисунок 22

Просматривает все элементы в данной коллекции. Этот метод возвращает итератор – объект, который можно использовать для перебора элементов коллекции.

Кроме того, методы addAll(), removeAll() и containsAll() принимают

другую коллекцию и добавляют, удаляют или проверяют все элементы предоставленной коллекции. коллекции.

### 2.6.3 Типы коллекций

Интерфейс Collection имеет три дочерних интерфейса. Set представляет коллекцию в которой не допускаются дубликаты элементов. List – это коллекция, элементы которой имеют определенный порядок. Интерфейс Queue представляет собой буфер для объектов с понятием "головного" элемента, который является следующим в очереди на обработку.

Set не имеет никаких методов, кроме тех, которые он наследует от Collection. Оно просто обеспечивает соблюдение правила отсутствия дубликатов. Если попытается добавить элемент, который уже существует в Set, метод add() просто вернет false. SortedSet сохраняет элементы в установленном порядке; как отсортированный список, который не может содержать дубликатов. Можем получить подмножества (которые также сортируются) с помощью методов subSet(), headSet() и tailSet(). Эти методы принимают один или пару элементов, которые обозначают границы. Методы first(), last() и comparator() предоставляют доступ к первому элементу, последнему элементу и объекту, используемому для сравнения элементов.

Java 7 добавила NavigableSet, который расширяет SortedSet и добавляет методы для поиска ближайшего совпадения, большего или меньшего, чем целевое значение, в пределах порядка сортировки набора.

Этот интерфейс может быть эффективно реализован с помощью таких методов, как пропускные списки, которые позволяют быстро находить упорядоченные элементы.

Следующим дочерним интерфейсом Collection является List. List – это упорядоченная коллекция, похожая на массив, но с методами для манипулирования позицией элементов в списке, пример добавления элемента в список на рисунке 23:



Рисунок 23

Добавляет указанный элемент в конец списка, пример на рисунке 24.



Рисунок 24

Вставляет заданный объект в указанную позицию в списке. Если позиция меньше нуля или больше длины списка, будет выброшено исключение IndexOutOfBoundsException. Элемент, который ранее находился на указанной позиции, и все элементы после него перемещаются на одну индексную позицию вверх, пример на рисунке 25.



Рисунок 25

Удаляет элемент в указанной позиции. Все последующие элементы перемещаются вниз на одну индексную позицию, пример на рисунке 26.



Рисунок 26

Возвращает элемент в заданной позиции, пример на рисунке 27.



Рисунок 27

Изменяет элемент в заданной позиции на указанный объект. По индексу уже должен существовать объект, иначе возникнет исключение IndexOutOfBoundsException выброшено.

Тип E в этих методах относится к параметризованному типу элемента класса List. Collection, Set и List – это интерфейсные типы. Это пример функции Generics.

Queue – это коллекция, которая действует как буфер для элементов. Очередь поддерживает порядок вставки элементов, помещенных в нее, и имеет понятие "головного" элемента. Очереди могут быть по принципу "первый вошел – первый вышел" (FIFO) или "последний вошел – первый вышел" (LIFO), в зависимости от реализации:

Метод offer() пытается поместить элемент в очередь, возвращая true в случае успеха. Различные типы очередей могут иметь различные лимиты или ограничения на типы элементов (включая емкость). Этот метод отличается от метода add(), унаследованного от Collection, тем, что он возвращает булево значение, а не выбрасывает исключение, указывающее на то, что элемент не может быть принят, пример на рисунке 28.

Изображение выглядит как текст, оранжевый

Автоматически созданное описание

Рисунок 28

Метод poll() удаляет элемент, находящийся во главе очереди, и возвращает его. Этот метод отличается от метода Collection remove() тем, что, если очередь пуста, вместо исключения возвращается null, пример на рисунке 29.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 29

Возвращает головной элемент, не удаляя его из очереди. Если очередь

пуста, возвращается null, пример на рисунке 30.



Рисунок 30

### 2.6.4 Интерфейс Map

Каркас коллекций также включает java.util.Map, который представляет собой коллекцию пар ключ/значение. Другие названия для карты – "словарь" или "ассоциативный массив". Карты хранят и извлекают элементы с ключевыми значениями; они очень полезны для таких вещей, как кэш или минималистичные базы данных. Когда храним значение в карте, связываем ключевой объект со значением. Когда нужно найти значение, карта извлекает его, используя ключ.

С помощью generics тип Map параметризуется двумя типами: один для ключей и один для значений. В примере на рисунке 31 фрагменте используется HashMap эффективный, но неупорядоченный тип реализации карты:

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 31

В устаревшем коде карты просто отображают типы Object на типы Object и требуют соответствующего приведения для получения значений.

Основные операции над Map просты. В следующих методах тип K обозначает тип параметра key, а тип V – тип параметра value, пример на рисунке 32:

Добавляет указанную пару ключ/значение в карту. Если карта уже содержит значение для указанного ключа, старое значение заменяется и возвращается в качестве результата, пример на рисунке 32.



Рисунок 32

Извлекает из карты значение, соответствующее ключу, пример на рисунке 33.



Рисунок 33

Удаляет значение, соответствующее ключу, из карты. Удаленное значение возвращается, пример на рисунке 34.



Рисунок 34

Возвращает количество пар ключ/значение в этой карте, пример на рисунке 35:



Рисунок 35

Этот метод возвращает набор, содержащий все ключи в этой карте, пример на рисунке 36.



Рисунок 36

Используйте этот метод для получения всех значений в этой карте. Возвращаемая коллекция может содержать дубликаты элементов, пример на рисунке 37.



Рисунок 37

Этот метод возвращает набор, содержащий все пары ключ/значение (в виде объектов Map.Entry) в этой карте, пример на рисунке 38.



Рисунок 38

Map имеет один дочерний интерфейс, SortedMap. Сортированная карта содержит пары ключ/значение, отсортированные в определенном порядке в соответствии со значениями ключей. Он предоставляет методы subMap(), headMap() и tailMap() для получения отсортированных подмножеств карты. Как и Sorted Set, он также предоставляет метод comparator(), который возвращает объект, определяющий способ сортировки ключей карты. Java 7 добавили NavigableMap с функциональностью, параллельной NavigableSet; а именно, она добавляет методы для поиска в отсортированных элементах элемента, большего или меньшего, чем целевое значение.

Следует уточнить, что хотя Map и является родственником, она не является буквально типом Collection (Map не расширяет интерфейс Collection). Можем задаться вопросом, почему.

Кажется, что все методы интерфейса Collection имеют смысл для Map, за исключением iterator(). Map, опять же, имеет два набора объектов: ключи и значения, и отдельные итераторы для каждого из них. Вот почему Map не реализует Collection. Если хотим получить представление карты в виде коллекции с ключами и значениями, можем использовать метод entrySet().

Еще одно замечание о картах: некоторые реализации карт (включая стандартную Java HashMap) позволяют использовать null в качестве ключа или значения, а другие – нет.

# **3 Среда создания видео**

В течение следующих восьми шагов пройдем весь путь от первого открытия Final Cut Pro до представления готового видеопроекта. К концу этой главы сможем использовать изученные приемы для базового редактирования в Final Cut Pro X в своих собственных проектах.

Программа для редактирования была взята для создания видео по следующим плюсам:

– данная программа лучше сжимает размер готовой продукции, чем ее конкуренты;

– программа дает на выходе «чисты» материал, который не подвергся уменьшению качества картинки;

– имеет инструменты работать с разными форматами видео, ограничений почти не, кроме новых, которые в недалеком будущем будут добавлены разработчиками;

– наличие библиотек, с обширными шаблонами для создания видеоматериала;

– отсутствие цветного искажения при корректировке цветов в постобработке.

Главным преимуществом перед остальными конкурентами, является ее производительность. Программа использует многоядерную обработку, что дает отсутствие сбоев и зависаний при работе или предпросмотре видеоматериала.

## **3.1 Создание нового проекта и импорт медиафайлов**

Final Cut Pro X организует весь свой контент в библиотеки, события и проекты. Библиотека – это самый высокий уровень, и обычно понадобится только одна из них, если только не работаете над множеством несвязанных проектов. Внутри библиотек находятся события, которые могут содержать такие медиафайлы, как видеоклипы, изображения, песни, а также проекты. Проект – это контейнер для вашего видео, в котором хранятся все ваши решения по редактированию в процессе создания видео.

При первом открытии Final Cut Pro библиотека и событие будут созданы автоматически, поэтому нужно будет только создать проект. Подробный пример выполнение пунктов в приложении А видео А.1**.**

1) Открыть Final Cut Pro X. Через несколько секунд должен появиться главный интерфейс.

2) В верхней строке меню выбрать Файл > Новый > Проект (или Cmd+N), чтобы создать новый проект. Введите название проекта и выберите использовать автоматические настройки.

3) Снова в верхней строке меню выбрать Файл > Импорт > Медиа (или Cmd+I). Откроется окно «Импорт медиа», в котором можно выбрать все аудио и видеофайлы, которые можно импортировать в проект.

4) Выбрать все медиафайлы, которые хотите включить в проект, и нажмите кнопку «Импортировать выбранные».

5) Закройте окно «Импорт медиафайлов».

## **3.2 Добавление обрезанных клипов на временную шкалу**

В браузере должны появиться миниатюры всех только что импортированных медиафайлов. Чтобы создать видео, нужно будет расположить и обрезать эти медиафайлы на временной шкале. Временная шкала показывает предварительный просмотр всего отснятого материала и имена файлов. Подробный пример выполнение пунктов в приложении А видео А.2

1) В браузере можно установить точки входа и выхода для каждого видеоклипа, удерживая Option, щелкнув миниатюру и перетащив ее мышью. На клипе появится желтый контур, представляющий диапазон, который будет добавлен к видео на временной шкале.

2) Можно перетащить клипы из браузера вниз на временную шкалу в том порядке, в котором нужно их воспроизводить слева направо. Можно начать с установочного кадра, чтобы создать сцену для зрителя, прежде чем погрузиться в тему вашего видео.

3) Обрежьте каждый клип, подведя курсор мыши к внутреннему краю его миниатюры на временной шкале. На месте курсора появится инструмент "Обрезка". Можем перетащить его вперед или назад, чтобы изменить окончание клипа. Сделать это с обеих сторон каждого добавляемого клипа.

4) Можем изменить порядок клипов в последовательности, можем перетащить любой клип с временной шкалы в другое место. Магнитная временная шкала Final Cut Pro автоматически вставит и расположит клипы. Убедитесь, что при таком расположении ни один клип не окажется сверху другого (по вертикали).

5) Переместите головку воспроизведения в начало временной шкалы и нажмите Play (сочетание клавиш Space). Можем увидеть предварительный просмотр вашего видео в окне просмотра.

## **3.3 Добавление текста и титров**

Final Cut Pro поставляется с десятками титров, расположенных в браузере, куда впервые импортировали медиафайлы в шаге 1. Можем легко добавить их в свой фильм, перетащив их вниз на временную шкалу. Подробный пример выполнение пунктов в приложении Б видео Б.3.

1) Над «Браузером» щелкните значок боковой панели «Титры» и генераторы, чтобы открыть ее (или нажмите Option+Cmd+1).

2) Выберите нужный титр и перетащите его вниз на временную шкалу. На этот раз убедитесь, что титр расположен над другими видеоклипами, потому что сможете увидеть его, только если он будет сверху.

3) Обрежьте начальную и конечную точки заголовка так же, как это было сделано с видеоклипами в шаге 1.

4) Переместите головку воспроизведения, чтобы можно было просмотреть заголовок на экране в средстве просмотра. Дважды щелкните по тексту и введите текст, чтобы заменить его на свой собственный. Вы также можете внести изменения в заголовок, щелкнув значок Инспектора заголовков в правом верхнем углу экрана.

## **3.4 Добавление переходов**

Когда дело доходит до переходов, иногда меньше значит больше, нет ничего плохого в "жестком разрезе" между двумя клипами, поэтому нужно использовать переходы тонко и только там, где они имеют смысл. Два отличных способа использования переходов – показать течение времени или перейти от одной сцены к другой. Подробный пример выполнение пунктов в приложении Б видео Б.4.

1) Откройте браузер переходов, щелкнув по значку (или нажмите Ctrl+Cmd+5).

2) Выберите переход и перетащите его между двумя любыми клипами, чтобы добавить анимированный переход между ними.

3) Можем регулировать длину перехода, перетаскивая его край внутрь или наружу.

4) Если нужно настроить переход, можем открыть «Инспектор» (Command+4) и внести изменения.

## **3.5 Добавление эффектов**

Теперь, когда завершили базовую аранжировку на временной шкале, можем добавить немного производственной ценности с помощью встроенных в Final Cut Pro эффектов. Подробный пример выполнение пунктов в приложении В видео В.5.

1) Щелкните значок браузера эффектов, чтобы открыть его (сочетание клавиш Cmd+5).

2) Найдите эффект, который нужно добавить. Эффекты можно предварительно просмотреть, проведя мышью по миниатюре.

3) Перетащите эффект на клип.

4) Рядом со значком браузера эффектов выберите эффект и перетащите его между двумя любыми клипами, чтобы добавить между ними анимированный эффект.

5) Если нужно настроить эффект, можем открыть инспектор (Cmd+4) и внести изменения.

## **3.6 Добавление музыки и аудио**

Можем добавить свою собственную музыку, импортировав ее так же, как добавляли видео в шаге 1, но в этом уроке будут использовать некоторые встроенные в Final Cut Pro композиции. Все они не имеют авторских прав, а значит, не придется беспокоиться о лицензировании, как в случае со многими песнями. Подробный пример выполнение пунктов в приложении В видео В.6.

1) Щелкните значок боковой панели «Фото» и аудио над «Браузером», чтобы открыть ее (или нажмите Shift+Cmd+1). Выберите «Звуковые эффекты».

2) Введите «Jingles», чтобы увидеть все песни, доступные в Final Cut Pro, для их предварительного просмотра можно нажать маленькую кнопку воспроизведения.

3) Когда найдете понравившуюся песню, перетащите ее вниз на временную шкалу под вашими видеоклипами. Увидите, что волновая форма песни отображается в виде зеленой звуковой дорожки.

4) Укоротите песню так, чтобы ее длина совпадала с длиной видеодорожки на временной шкале.

5) Если нужно внести коррективы в песню, выберите клип и откройте инспектор. Здесь можем отрегулировать громкость и эквалайзер.

6) Вернувшись на «Временную шкалу», сделайте так, чтобы песня затухла, потянув за ручку «Fade» в конце зеленого аудиоклипа.

## **3.7 Коррекция цвета**

Далее предстоит выровнять баланс белого с помощью встроенного в Final Cut Pro инструмента «Balance Color». Это позволит удалить любые цветовые оттенки и быстро и легко подчеркнуть лучшие качества отснятого материала. Подробный пример выполнение пунктов в приложении Г видео Г.7.

1) Выделите все видеоклипы на временной шкале. Вокруг них должен появиться желтый контур.

2) Щелкните значок «Улучшения» под «Средством просмотра», чтобы открыть выпадающее меню.

3) Выберите «Баланс» цвета или Option+Cmd+B.

4) Должны заметить улучшение цветов и контрастности всех видеоклипов.

## **3.8 Поделитесь своим фильмом**

Теперь, когда завершили свой проект, пришло время поделиться им с миром. В Final Cut Pro есть встроенные опции экспорта, позволяющие делиться видео непосредственно на YouTube, Facebook, Vimeo и т. д., но для этой главы создадите высококачественный «Мастер-файл». Подробный пример выполнение пунктов в приложении Г видео Г.8.

1) Щелкните в любом месте временной шкалы.

2) В верхней строке меню выберите Файл > Поделиться > Мастер-файл.

3) Нажмите кнопку Далее и выберите место для сохранения готового видео.

4) Рендеринг конечного видео может занять несколько минут. Можем проверить ход работы, нажав на значок «Фоновые задачи» в верхней части экрана.

После завершения рендеринга нового видео должно появиться в Quicktime. Можем поделиться этим «Мастер-файлом» в любом месте, но он будет довольно большим, поэтому сначала его нужно сжать.

Поздравляем, ваше видео завершено! Теперь, когда научились импортировать клипы, добавлять их на временную шкалу, затем добавлять переходы, титры, эффекты и музыку, фактически изучили основы работы с Final Cut Pro для начинающих. По мере роста навыков и выработки собственного ритма можете менять последовательность шагов для разных проектов.

Конечно, предстоит еще многому научиться, к тому же есть масса замечательных дополнений и плагинов, которые могут сделать Final Cut Pro еще проще и мощнее.

# **ЗАКЛЮЧЕНИЕ**

В заключении курсовой работы можно подвести итоги: почему стоит выбрать Java для работы и развития своих знаний в этой области, что узнали за весь курс изучения Java, что изучать после начального введения в программирования на языке Java, с какими трудностями столкнулись.

1) Java – это многопрофильный язык программирования, поскольку он используется для написания логики на стороне сервера для распределенных систем, а также настольных, мобильных и веб-приложений. Поэтому ориентация на Java дает широкий выбор направления в мире разработки. Легко можно начать заниматься другим видом разработки, не меняя языка программирования [3].

В то же время Java – очень удобная платформа с точки зрения наличия готовых решений и компонентов [3]. Наличие огромного сообщества делает процесс обучения и вхождения в профессию более легким и быстрым [9]. В финансовом плане спрос на Java-разработчиков растет с каждым днем, поэтому Java достаточно стабильная среда, где можно быть уверенным в завтрашнем дне [3].

2) За курс обучения изучены основы языка программирования Java: классы, конструкторы, getters, setters, интерфейсы, коллекции и дженерики.

3) В данный период времени важно стремится попасть в IT, так как порог входа стал ниже и будет легче вступить в должность junior-разработчика.

4) В создании главы 3 по созданию видеоматериала были трудности в поиске информации, как создавать и пользоваться программой. В «русском» интернет очень мало бесплатных и хороших материалов для изучения. Мне пришлось прибегнуть к зарубежным источникам, в которых был нужный и качественный материал, чтобы написать краткий и подробный материал, который был понятен любому пользователю.

По процессу работы мне удалось выяснить, что данный материал можно продавать на цифровом рынке.

Прибыль будет исходить от покупателя, который будет создавать и продвигать свои видеоуроки по моим сценариям для массового потребителя. Что позволит освободить меня от ненужных трудностей в виде создания, продвижения и обслуживания платформы данных видеоуроков.

Если вдруг язык Java не понравился написанием и чтением кода, можно перейти на изучение других языков программирования, так как основные темы, которые изучены в данном курсе, не сильно отличаются, разница в других языках программирования – только синтаксическое написание кода.

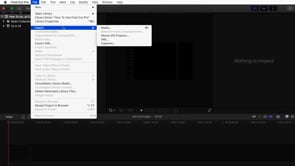
По моему мнению, можно научиться программировать с помощью видеоуроков на языке Java, в удобное время и в удобной форме.

# **СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ**

1. 7 причин изучать Java 23 августа 2021 // – URL: <https://vc.ru/dev/284968-7-prichin-vybrat-java-dlya-izucheniya> (дата обращения 12.04.2022)
2. Язык программирования Java: особенности, популярность, ситуация на рынке труда 15 марта 2021 // – URL: <https://ru.hexlet.io/blog/posts/yazyk-programmirovaniya-java-osobennosti-populyarnost-situatsiya-na-rynke-truda> (дата обращения 12.04.2022)
3. «Ъ»: нехватка IT-кадров в России сменится на их избыток // – URL: <https://habr.com/ru/news/t/662025/> (дата обращения 12.04.2022)

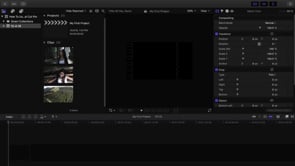
# **ПРИЛОЖЕНИЕ А**

Создание нового проекта и импорт медиафайлов

[](https://player.vimeo.com/video/688632961?h=97a19b5089&app_id=122963)

Видео А.1 Создание нового проекта и импорт медиафайлов

Добавление обрезанных клипов на временную шкалу

[](https://player.vimeo.com/video/688632937?h=0a9d412763&app_id=122963)

Видео А.2 Добавление обрезанных клипов на временную шкалу

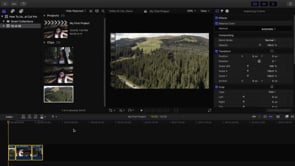
# **ПРИЛОЖЕНИЕ Б**

Добавление текста и титров

[](https://player.vimeo.com/video/688632913?h=e87cce5263&app_id=122963)

Видео Б.3 Добавление текста и титров

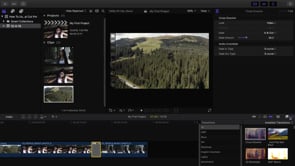
Добавление переходов

[](https://player.vimeo.com/video/688632925?h=5763faf4e2&app_id=122963)

Видео Б.4 Добавление переходов

# **ПРИЛОЖЕНИЕ В**

Добавление эффектов

[](https://player.vimeo.com/video/688632892?h=751f50bbbb&app_id=122963)

Видео В.5 Добавление эффектов

Добавление музыки и аудио

[](https://player.vimeo.com/video/688632905?h=d16dc8a91c&app_id=122963)

Видео В.6 Добавление музыки и аудио

# **ПРИЛОЖЕНИЕ Г**

Коррекция цвета

[](https://player.vimeo.com/video/688632945?h=a1ac8e937c&app_id=122963)

Видео Г.7 Коррекция цвета

Поделитесь своим фильмом

[](https://player.vimeo.com/video/688632973?h=48dc949faf&app_id=122963)

Видео Г.8 Поделитесь своим фильмом