Санкт-Петербургский государственный университет

Технологии программирования

Группа 20.Б07–мм

Вивдич Софья Андреевна

Модификация Java UI для удовлетворения требований проекта OpenJDK CRaC

Отчёт по учебной практике
в форме «Решение»

Научный руководитель:
ассистент кафедры системного программирования СПбГУ
Козлов А.П.

Санкт-Петербург
2022

Оглавление

Введение

Java входит в тройку самых популярных языков программирования, согласно индексу TIOBE [12].
Для Java существует множество реализаций спецификации платформы, которые называются JDK. OpenJDK — эталонная реализация Java.

Java часто используется в таких индустриях, как мобильная разработка, финансовые системы, анализ данных и встроенные системы — там, где от программ требуется производительность. С этим у Java–программ есть проблемы: во–первых, долгий старт, во–вторых, необходимость предварительного разогрева приложения.

Первая проблема связана с тем, что каждая программа состоит из множества классов, которые необходимо сначала загрузить, а затем проинициализировать. Для решения этой проблемы реализована JIT–компиляция (Just–in–Time, динамическая компиляция), которая даёт прирост производительности, однако тоже требует ресурсы на выполнение. Другое решение проблемы — технология Application Class Data Sharing, которая помогает ускорять запуск и экономить память за счет оптимизации загрузки классов. Однако использование этой технологии дает лишь небольшой выигрыш [4].

Вторая проблема частично решается Ahead–of–Time компиляцией (то есть компиляции перед исполнением), которая позволяет предсгенерировать нативный код, который можно использовать вместо траты времени на JIT–компиляцию. Однако этот процесс тоже требует времени. Другое решение — технология GraalVM Native Image, позволяющая получить нативный образ программы при помощи Ahead–of–Time–компиляции из Java в машинный код. Однако у GraalVM Native Image есть ограничения на применимость, например динамическая загрузка классов.

В качестве одного из решений проблем медленного старта приложений на Java на основе OpenJDK был создан проект CRaC — Coordinated Restore at Checkpoint [16], [18].
CRaC предоставляет две операции:

- checkpoint — сохранение состояния процесса в образ и разрыв всех соединений
- restore — восстановление оригинального процесса из сохраненного образа

Использование этих операций на оптимальной производительности даст запуск виртуальной машины уже разогретой, за счет того что операция restore развернет оригинальный процесс из сохраненного при создании образа с уже загруженными и проинициализированными классами и готовыми JIT–компиляциями.
Таким образом, CRaC решает ранее обозначенные проблемы Java–программ.

Если выполнилась операция checkpoint, CRaC обязуется восстановить процесс в первоначальном виде, что говорит о надежности сохраненных образов. Для этого CRaC реализует следующие принципы:

- приложение обязано управлять внутренними ресурсами (иначе — проблемы с внешним и внутренним состоянием);
- CRaC отслеживает опасные состояния и отменяет checkpoint в случаев выявления проблем.

Графическая подсистема — это пример связи с внешней средой, наличие которой не позволяет создать образ.

1. Постановка задачи

Целью данной работы является поддержка графических приложений в проекте OpenJDK CRaC, в частности обзор и модификация графических библиотек Java.

Для достижения цели были поставлены следующие задачи.

1. Сделать обзор библиотек Java для разработки графических приложений.
2. Определить взаимосвязи между библиотеками и построить план поддержки графических приложений.
3. Выбрать и модифицировать графическую библиотеку для поддержки CRaC на ОС Linux.

2. Обзор

В данном разделе представлены обзор самых популярных библиотек Java для разработки графических приложений, комментарии к нему и выявление взаимосвязей между библиотеками.

2.1. Библиотека Abstract Window Toolkit

AWT (Abstract Window Toolkit) — это исходная платформо-независимая оконная библиотека графического интерфейса языка Java. Сейчас AWT является частью Java Foundation Classes (JFC) — стандартного API для реализации графического интерфейса в Java–программе [2]. AWT является частью спецификации Java SE (Java Platform Standard Edition — стандартная версия платформы Java 2), поэтому обязан быть реализован в любой реализации Java, в том числе в OpenJDK [1].

AWT предоставляет пользователю набор собственных компонентов пользовательского интерфейса, инструменты графики, а также надежную модель обработки событий.

AWT является основной библиотекой по созданию GUI (Graphical User Interface). Основной принцип — AWT–приложение выглядит поразному на разных операционных системах, так как сохраняет внешний вид собственной оконной системы. Это возможно благодаря тому, что AWT не самостоятельно рисует графические объекты и обеспечивает их функциональность, а делегирует это собственным компонентам GUI — peers, используя одноименный набор классов. Использование peers устраняет необходимость повторной реализации функциональных возможностей, инкапсулированных в собственных компонентах управления окнами [17]. Таким образом, компоненты AWT являются “тяжеловесными”, так как имеют соответствующую нативную структуру.
Такой подход имеет спорные преимущества. Например, данная реализация AWT обеспечивает пользователям разных платформ привычный каждому внешний вид и функциональность приложения, однако доставляет разработчику проблемы, так как нет гарантии, что графи-

ческий объект, поведение которого устраивает разработчика на одной платформе, будет вести себя корректно на другой.
Еще одна особенность такой реализации AWT — компоненты могут иметь только прямоугольную форму.

Рассмотрим иерархию классов AWT на картинке 1.

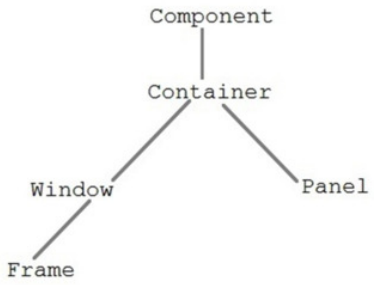

Рис. 1: Иерархия классов AWT

Базовым классом для всех компонентов AWT является абстрактный класс Component. Этот класс описывает основные свойства компонентов: положение, шрифт, цвета, видимость и др. Компоненты группируются в контейнеры (описываются классом Container, наследником Component) при помощи методов add (добавить компонент), remove (удалить компонент) и removeAll (удалить все компоненты). Класс Panel — это контейнер, который используется для содержания компонентов. В панель присоединяются другие объекты приложения. Далее — окно верхнего уровня. Оно создается классом Window, не имеет границ. Frame — окно верхнего уровня, имеющее заголовок, границы и изменяемый размер.
При создании приложения сначала необходимо создать окно верхнего уровня — объект класса Frame.

Для отрисовки компонента используется метод paint, имеющий в качестве аргумента объект абстрактного класса Graphics — базового класса для всех графических контекстов, которые позволяют приложению

использовать компоненты, реализованные на различных устройствах, а также изображения за пределами экрана [14]. Компоненты и их наследники переопределяют метод paint и задают алгоритм отрисовки, используя методы класса Graphics.

Любое графическое приложение подразумевает некое взаимодействие с пользователем. Нажатие на кнопку, ввод текста, выбор элемента списка и пр. — эти действия пользователя, называемые событиями, нуждаются в обработке и управлении. Низлежащая среда выполнения генерирует событие, оно отправляется в программу, и она отвечает на это событие соответствующим образом.

События можно разделить на две группы:

- События переднего плана — события вызваны действиями пользователя над графическими компонентами.
- Фоновые события — события вызваны действиями конечного пользователя. Например:
  - программный сбой;
  - завершение работы.

В модели событий AWT два ключевых участника:

- Источник события — объект, действие над которым приводит к генерации события.
- Обработчик события (или слушатель) — объект, зарегистрированный в текущем источнике события, который получает от него уведомление о событии и генерирует ответ. События могут передаваться нескольким получателям.

Обработка события выглядит следующим образом:

- пользователь совершает действие, генерируется событие;
- создается объект соответствующего класса событий;
- объект события отправляется в метод зарегистрированного слушателя;
- выполнение метода.

Итог: программа отреагировала на событие.
Описанная модель получила название «делегирование».

2.2. Библиотека Swing

Библиотека Swing была разработана компанией Sun Microsystems. Swing, как и AWT, относится к библиотеке классов Java Foundation Classes. Swing является частью спецификации Java SE, поэтому обязан быть реализован в любой реализации Java, в том числе в OpenJDK [9]. Swing полностью написан на Java, использует графическую подсистему по минимуму, а именно использует AWT для создания окна операционной системы, а затем рисует изображения остальных компонентов в это окно с помощью Java2D и отвечает на все действия пользователя, решая, что делать, не делегируя это операционной системе. Все это делает Swing полностью переносимым, приложение выглядит и ведет себя одинаково на разных системах.

Рассмотрим иерархию классов Swing на картинке 2. Для создания графического интерфейса приложения необходимо использовать контейнеры верхнего уровня — окна операционной системы, в которых размещаются компоненты пользовательского интерфейса. К контейнерам верхнего уровня относятся JFrame, JWindow, JDialog, JApplet, которые представляют собой тяжеловесные компоненты (так как связаны с операционной системой), все остальные компоненты являются легковесными.

Любое Swing–приложение имеет хотя бы один контейнер верхнего уровня. Этот контейнер — корень иерархии включения, которая содержит все компоненты Swing внутри контейнера верхнего уровня.

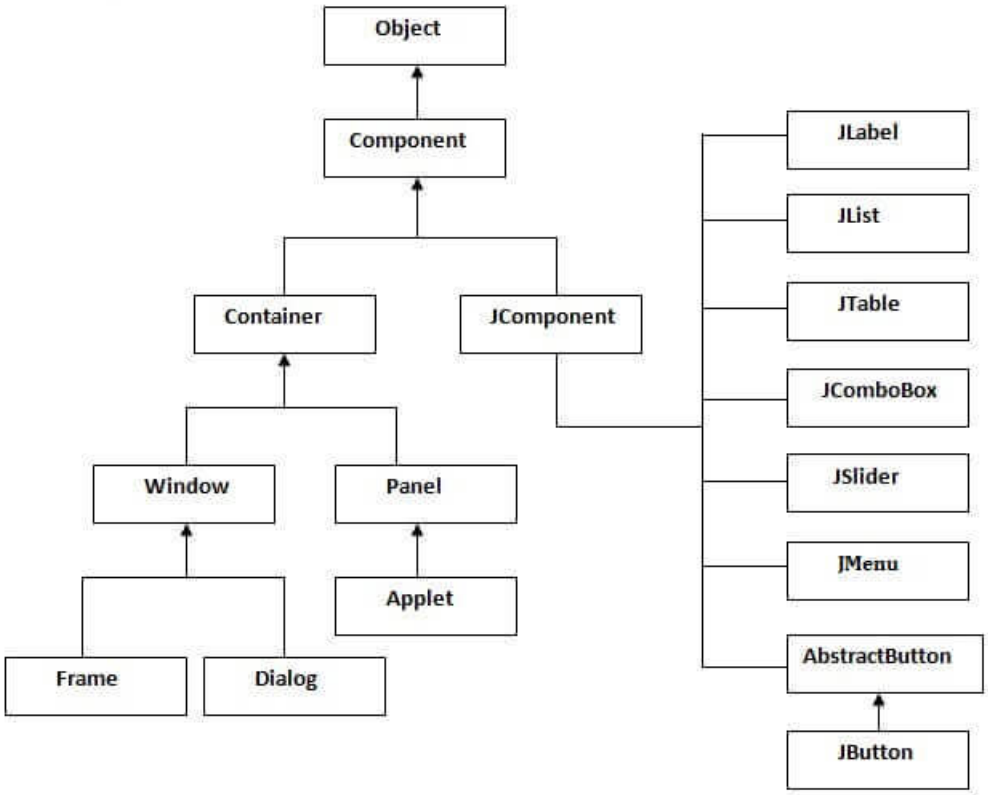

Рис. 2: Иерархия классов Swing

Архитектура Swing API реализует паттерн MVC (Model View Controller):

- Model — данные компонента.
- View — визуальное представление данных компонента.
- Controller — принимает входные данные view и отражает изменения в model.

Swing дает разработчику возможность изменить внешний вид (Look) и поведение (Feel) компонентов — Look and Feel. Благодаря этой технологии программист может сделать так, что бы приложение выглядело и, главное, вело себя одинаково на разных операционных системах (при

таком выборе проще отладка приложения). Второй вариант — приложение может выглядеть родным для каждой операционной системы, т.е. приложение будет похоже на нативное, где б его ни запустили.
При этом JRE (Java Runtime Environment) под разные операционные системы включают в себя разный набор Look and Feel. Существует возможность подключения стороннего Look and Feel. По умолчанию используется Look and Feel Metal, который во всех ОС и оконных менеджерах выглядит более-менее одинаково.
Примеры Look and Feel:

- CrossPlatformLookAndFeel (Metal) — родной Look and Feel для Java–приложений.
- SystemLookAndFeel — родной Look and Feel для системы, где запускается приложение.
- Multiplexing — использование нескольких Look and Feel одновременно.

По умолчанию используется Look and Feel Metal, так как выглядит более–менее одинаково для разных операционных систем.

Обработка событий происходит так же, как и в AWT. Источник и слушатель — два ключевых участника событий.

2.3. Библиотека Standard Widget Toolkit

Standard Widget Toolkit (SWT) — библиотека с открытым исходным кодом для разработки GUI, представляет собой кроссплатформенную оболочку для графических библиотек конкретных платформ. Разработана фондом Eclipse [8].

Особенности и принципы SWT:

- обеспечивает функциональность встроенных виджетов для платформы Eclipse независимо от операционной системы [5];

- компоненты, которые поддерживаются графической подсистемой, работают как в AWT;
- остальная часть компонентов и функций написана на Java;
- использует операционную систему напрямую через C.

Таким образом, SWT объединяет в себе принципы работы AWT и Swing.

Для реализации SWT на разных платформах использованы низкоуровневые методы реализации.

Если рассматривать реализацию какого-либо конкретного виджета SWT на разных платформах, то реализация виджета на конкретной платформе не является переносимой, однако код приложения, использующий этот виджет, переносится. Это достигается следующим образом: SWT предоставляет разные классы каждого конкретного виджета для каждой платформы, однако сигнатура всех общедоступных методов одинакова. Код Java, вызывающий SWT, не знает и не заботится о том, на какой именно класс виджета ссылается во время выполнения [11]. Вызов Java передается прямо в операционную систему, код C гарантированно демонстрирует такое же поведение. Все написано на Java с использованием терминологии и документации операционной системы.

Структура приложения SWT:

- создание экрана (экземпляр класса Display);
- создание главного окна приложения (экземпляр класса Shell);
- создание виджетов главного окна;
- регистрация обработчиков событий виджетов.

Обработка событий происходит аналогично AWT и Swing: необходимо добавить к созданным экземплярам виджетов обработчики событий, а затем выполнить действия, если произошло событие.

2.4. Библиотека JavaFX

Все рассмотренные библиотеки отражают двумерный графический интерфейс с помощью стандартных алгоритмов отрисовки.

JavaFX — это инструментарий GUI для Java, который предоставляет возможности по работе с двумерной и трехмерной графикой.
Начиная с восьмой версии, Java включала в себя библиотеку JavaFX, однако с версии Java 11 JavaFX больше не входит в Java SE и не разрабатывается компанией Oracle [6]. Сейчас JavaFX — отдельный проект [7].

JavaFX не привязана к OpenJDK, обладает кроссплатформенностью, поэтому будет работать на всех платформах, где установлена исполняемая среда Java.

Особенности JavaFX:

- использует аппаратное 3D ускорение и возможности GPU (Graphics Processing Unit);
- использование стилей CSS;
- использование специального формата FXML для создания GUI
- возможно взаимодействие с библиотекой Swing
- следует архитектуре MVC

JavaFX–приложение состоит из иерархии трех основных компонентов. Она представлена на картинке 3.

- Stages — начальное окно, содержит остальные компоненты.
- Scene — содержание stage; несколько сцен реализуются как граф объектов.
- Nodes — элементы управления.

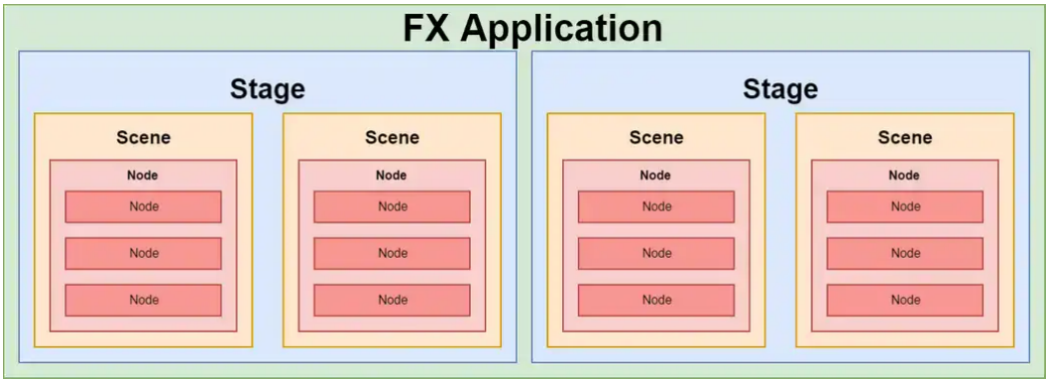

Рис. 3: Иерархия компонентов JavaFX

Разработчики могут получить доступ к существующим функциям библиотеки Java, такой как Swing [10].

2.5. Взаимосвязи между библиотеками

На основе всего вышеизложенного была построена таблица 1 сравнения библиотек. Библиотеки сравниваются по трем характеристикам: двумерная/трехмерная графика, является ли библиотека частью Java SE, на чем основана.

Библиотека	2D | 3D	Java SE	основано на:
AWT	2D	+	нативный образ компонентов
Swing	2D	+	AWT (в минимальной степени)
SWT	2D	—	AWT + Swing
JavaFX	2D и 3D	с java8 по java11	AWT (в минимальной степени)

Таблица 1: Сравнение библиотек

Таким образом, выявлены следующие взаимосвязи:

- Swing, JavaFX основаны на AWT (в минимальной степени)
- SWT основан на Swing + AWT

Можно сделать вывод, что графические подсистемы Java используют функциональность AWT для взаимодействия с графикой операционной системы. Логичный шаг — начать поддержку графических приложений именно с AWT. Этим занимается мой коллега Кузнецов И.А. (третий курс, ПИ). Следующий шаг — поддержка Swing–приложений. Swing–приложение будет запускаться на любой Java–поддерживаемой платформе. Кроме того, уже многие графические приложения написаны с использованием Swing. По этим причинам для модификации выбрана именно эта библиотека.

3. Обзор используемых технологий

3.1. Процессор аннотаций для генерации исходных файлов

У компонентов Java Swing десятки методов. В ходе работы возникла необходимость одинаковым образом переопределить все эти методы, а именно:

1. добавить StackTrace и рассмотреть некоторые его элементы;
2. построить операцию по данному методу;
3. добавить операцию в список при выполнении условия.

В остальном методы не изменяются, поэтому завершаются вызовом метода суперкласса.

Таким образом, необходимо сгенерировать новые классы компонентов, состоящие из одинаковым образом переопределенных методов исходных классов компонентов. Для этих целей используется процессор аннотаций.

Проект делится на два модуля Maven: AnnotationProcessing, в котором обрабатывается аннотация, и AnnotationClient, который содержит аннотированные классы–обертки над исходными классами компонентов. Созданная аннотация @Crac показывает, что необходимо сгенерировать новый класс на основе того класса, к которому она добавлена.

В модуле AnnotationProcessing указано, что обрабатывается аннотация @Crac. Далее, чтобы создать процессор аннотации, требуется расширить класс AbstractProcessor и реализовать метод process. Реализация этого метода и генерация классов описаны в разделе “Ход работы”.

3.2. Java Reflection для исследования программы

Для генерации новых классов необходимо обрабатывать информацию об исходных. Для этих целей использовалась рефлексия — функция языка Java, которая позволяет исполняемой программе исследовать саму себя [13]. Особенно часто использовались следующие возможности рефлексии:

1. Class.forName — для загрузки класса с заданным именем;
2. getMethods — для получения методов класса;
3. getMethod — для получения метода с заданным именем и параметрами;
4. invoke — вызов метода для заданного объекта (иногда со списком значений параметров);
5. newInstance — для создания нового экземпляра объекта.

Использование рефлексии позволило обрабатывать информацию о классах для генерации новых.

4. Ход работы

4.1. Подготовка окружения

Для начала требовалось подготовить окружение, в котором будет происходить реализация решений. В него входит подготовка отладочной сборки OpenJDK 14 с проектом CRaC.

4.2. Выбор библиотеки для модификации

На этом этапе были изучены принципы и особенности работы приложений, написанных с использованием графических библиотек AWT, Swing, SWT, JavaFX. Были выявлены взаимосвязи, построен план модификации библиотек для поддержки графических приложений в проекте OpenJDK CRaC (первая ступень — AWT, вторая — Swing), для данного проекта выбрана библиотека Swing.

4.3. Подготовка тестового приложения

Для модификации библиотеки Swing была выбрана следующая стратегия: предполагаем, что AWT уже модифицирована. Это означает, что, если сделать checkpoint в графическом приложении, состоящем из окна операционной системы, то после restore состояние процесса корректно восстановится.

Было подготовлено тестовое Swing–приложение, для которого идет тренировка имитации CRaC.

Следующая задача: необходимо модифицировать приложение таким образом, чтобы запоминать запросы по созданию графических компонентов Swing, для того чтобы корректно восстанавливать их из образов. После успеха с существующем приложением, в него будут добавлены новые компоненты и цикл будет повторён, пока не будут включены все элементы Swing.

4.4. Начало реализации

Необходимо организовать сохранение действий пользователя, чтобы воссоздать графическое приложение без его вмешательства.

Было рассмотрено небольшое приложение [15]. Действия пользователя: создание компонентов (фрейм, панель, кнопка), добавление кнопки к панели, панели — к фрейму. Были созданы прототипы этих методов в виде классов операций. Например, за добавление кнопки к панели отвечает класс AddButtonToPanel. У классов есть единственный метод — run(), который отражает функционал соответствующего классу метода. При создании компонента пользователем соответствующая операция сохраняется в список. При восстановлении для каждой операции из списка вызывается метод run(). В результате получаем вид исходного приложения.

Описанная идея с некоторыми модификациями была взята за основу работы. Вместо создания класса для каждого метода было решено сгенерировать новые классы компонентов на основе исходных с переопределением методов. Для этого был использован процессор аннотаций.

4.5. Процессор аннотаций

Первым шагом создается пустой класс–обертка над существующим классом Java–компонента. Над этим классом добавляется аннотация @CRaC.

Второй шаг — обработка аннотации. В методе process запускается цикл по всем элементам, над которыми добавлена аннотация. Для текущего элемента сохраняются его полное имя и суперкласс, с методов которого начнется генерация. Далее вызывается метод генерации класса. По имени аннотированного элемента строится имя нового класса и имя пакета. После добавления конструктора класса начинается генерация методов: необходимо переопределить методы всех суперклассов

до java.lang.Object. Переопределение методов начинается с суперкласса аннотированного элемента, после чего рассматривается суперкласс суперкласса, и процесс повторяется.

4.6. Переопределение методов

В теле метода строится соответствующая ему операция. Операция добавляется в список, если этот метод был вызван пользователем, а не был частью вызова другого действия. Для проверки этого условия в каждый метод добавляется StackTrace. Условие выполнено, если имя второго элемента StackTrace (класс, откуда был вызван текущий метод) начинается не с “java.”. В таком случае, построенная операция добавляется в список. Метод завершается вызовом того же метода суперкласса.

4.7. Построение операции по методу

Для построения операции по методу создается экземпляр класса Operation со следующими полями:

1. имя класса, в котором определен соответствующий метод;
2. имя метода;
3. упорядоченная таблица <тип параметра — значение>;
4. объект, на котором будет выполнена операция.

В классе операций определен единственный метод run(), который, используя перечисленные поля, участвует в восстановлении приложения.

4.8. Восстановление графического приложения

У сгенерированных классов есть две особенности. Во–первых, все классы имеют поле java.lang.Object newObject, которое необходимо для

восстановления приложения. Во–вторых, сгенерированные классы реализуют интерфейс UpdateComponent, метод update(Object obj) которого необходим для обновления значения поля newObject.

При создании компонента пользователем в список добавляется операция с особым именем метода — “createClassName”. При воспроизведении операции с таким именем вызывается метод update, при помощи которого ”пересоздается” объект newObj. Все следующие операции в списке, которые ранее выполнялись над соответствующим объектом, берут значение поля newObj этого объекта и выполняют операции над ним. Таким образом, newObj становится компонентом восстановленного приложения.

Для остальных операций списка вызывается метод run(), который устроен следующим образом:

- получение класса по его имени;
- если таблица параметров пуста: поиск метода по имени в классе;
- если таблица параметров не пуста: поиск метода по имени и списку типов параметров в классе;
- вызов полученного метода для объекта.

5. Тестирование

Тестирование проводится на постоянно усложняющемся приложении, которое будет содержать все больше компонентов и вызывать все больше методов. Записывая и затем воспроизводя действия пользователя, ожидается создание абсолютно идентичного приложения. Это обеспечивается корректным заполнением списка операций и правильной обработкой вызовов методов. В настоящий момент удается восстанавливать простые приложения 4.

Рис. 4: Восстановление простого приложения Swing

Заключение

В ходе выполнения данной работы были достигнуты следующие результаты:

- Сделан обзор библиотек Java для разработки графических приложений
- Определены взаимосвязи между библиотеками
- Подготовлено тестовое приложение Swing [19]
- Сделан прототип модификации Swing
  - сгенерирован новый набор классов с помощью аннотаций
  - разработана и реализована модель данных для записи операций над объектами Swing
  - удалось воссоздать простое графическое приложение

Код доступен в GitHub репозитории [3].

Список литературы

[1] AWT Oracle. — Access mode: https://docs.oracle.com/javase/7/docs/api/java/awt/package-summary.html (online; accessed: 2021-11-20).

[2] Abstract Window Toolkit. — Access mode: https://ru.wikipedia.org/wiki/Abstract_Window_Toolkit (online; accessed: 2021-11-20).

[3] AnnotationCrac Repository. — Access mode: https://github.com/SVivdich02/AnnotationCrac (online; accessed: 2022-06-05).

[4] Application Class Data Sharing. — Access mode: https://medium.com/@toparvion/appcds-for-spring-boot-applications-first-contact-6216db6a4194 (online; accessed: 2021-12-05).

[5] Eclipce SWT. — Access mode: https://www.eclipse.org/articles/Article-SWT-Design-1/SWT-Design-1.html (online; accessed: 2021-11-13).

[6] JavaFX WIKI. — Access mode: https://ru.wikipedia.org/wiki/JavaFX (online; accessed: 2021-12-03).

[7] JavaFX как отдельный проект. — Access mode: https://openjfx.io/ (online; accessed: 2021-12-03).

[8] Standard Widget Toolkit. — Access mode: https://ru.wikipedia.org/wiki/Standard_Widget_Toolkit (online; accessed: 2021-11-13).

[9] Swing Oracle. — Access mode: https://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html (online; accessed: 2021-10-18).

[10] Взаимодействие JavaFX и Swing. — Access mode: https://coderlessons.com/tutorials/java-tekhnologii/vyuchi-javafx/javafx-obzor (online; accessed: 2021-12-03).

[11] Виджеты SWT и их реализация. — Access mode: https://www.eclipse.org/articles/Article-SWT-Design-1/SWT-Design-1.html (online; accessed: 2021-11-14).

[12] Индекс TIOBE самых популярных языков программирования. — Access mode: https://www.tiobe.com/tiobe-index/ (online; accessed: 2021-12-05).

[13] Использование Java Reflection. — Access mode: https://www.oracle.com/technical-resources/articles/java/javareflection.html (online; accessed: 2022-05-19).

[14] Отрисовка компонентов AWT. — Access mode: https://docs.oracle.com/javase/7/docs/api/java/awt/Graphics.html (online; accessed: 2021-11-20).

[15] Приложение Swing с восстановлением компонентов. — Access mode: https://github.com/SVivdich02/Swing_CRaC/tree/SimpleTestApp/TestApp (online; accessed: 2022-05-20).

[16] Проект OpenJDK CRaC. — Access mode: https://openjdk.java.net/projects/crac/ (online; accessed: 2021-11-17).

[17] Работа собственных компонентов графической подсистемы. — Access mode: https://proxy.library.spbu.ru:2409/library/view/graphic-java-1-2/0130796662/0130796662_ch01lev1sec3.html (online; accessed: 2021-11-02).

[18] Реализация CRaC. — Access mode: https://github.com/openjdk/crac (online; accessed: 2021-11-17).

[19] Тестовое приложение Swing. — Access mode: https://github.com/SVivdich02/Swing_CRaC (online; accessed: 2022-05-19).