Министерство культуры Российской Федерации

федеральное государственное бюджетное образовательное учреждение

высшего образования

**«ВСЕРОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ ИНСТИТУТ**

**КИНЕМАТОГРАФИИ имени С.А. ГЕРАСИМОВА»**

**ВГИК VGIK**

**«Сергиево-Посадский филиал Всероссийского государственного института кинематографии имени С.А. Герасимова»**

**КУРСОВАЯ РАБОТА**

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Раздел 1 Основы проектирования программного обеспечения

МДК.04.01 Технология разработки проекта

ПМ.04 Обеспечение проектной деятельности

Специальность 230701 Прикладная информатика (по отраслям),

базовая подготовка

Тема курсовой работы: Разработка Java-приложения в системе программирования IntelliJ IDEA «Моделирование работы воздушного шара»

Группа 292

Студент\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (И.О. Фамилия)

Руководитель\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (И.О. Фамилия)

г. Сергиев Посад

2017

Министерство культуры Российской Федерации

федеральное государственное бюджетное образовательное учреждение

высшего образования

**«ВСЕРОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ ИНСТИТУТ**

**КИНЕМАТОГРАФИИ имени С.А.ГЕРАСИМОВА»**

**ВГИК VGIK**

**«Сергиево-Посадский филиал Всероссийского государственного института кинематографии имени С.А. Герасимова»**

**КУРСОВАЯ РАБОТА**

**ЗАДАНИЕ**

|  |
| --- |
| Студент Скворцов Павел Дмитриевич группа 292 |
|  |
| Раздел 1 Основы проектирования программного обеспечения  МДК.04.01 Технология разработки проекта  ПМ.04 Обеспечение проектной деятельности  Специальность 230701 Прикладная информатика (по отраслям), базовая подготовка  Тема курсовой работы: Разработка Java-приложения в системе программирования IntelliJ IDEA «Моделирование работы воздушного шара» |
| Утверждена протоколом № \_\_\_\_ от «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2017 года предметно-цикловой  комиссии вычислительной техники |
|  |
| Исходные данные для курсовой работы: Язык программирования Java, система программирования IntelliJ IDEA, пользовательские данные, набор тестовых данных |
| Введение |
| Содержание основной части курсовой работы |
| Основная часть  1 Изучение предметной области  2 Проектирование  3 Реализация  4 Отладка и тестирование  5 Внедрение и сопровождение  Заключение  Литература  Приложения  Практическая часть курсовой работы - программное приложение, презентация, документация  Задание выдал руководитель «\_\_» «\_\_» 2017 г \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Терешко Л.В.  Задание принял к исполнению «\_» «\_\_» 2017 г \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Скворцов П.Д.  Курсовая работа принята «\_\_\_\_» «\_\_\_\_» 2017 г. с оценкой\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Руководитель курсовой работы Терешко Л.В. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

**СОДЕРЖАНИЕ**

ВВЕДЕНИЕ 5

1. ИЗУЧЕНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ 7
   1. Постановка задачи 7
   2. Описание предметной области 8
   3. Вывод 8
2. ПРОЕКТИРОВАНИЕ 9
   1. Проектирование программного обеспечения при структурном подходе 11
      1. Структурная схема разрабатываемого программного обеспечения 13
      2. Функциональная схема 14
      3. Метод пошаговой детализации при составлении алгоритмов 15
      4. CASE-технологии 16
   2. Проектирование программного обеспечения при объектном подходе 18
   3. Вывод 19
3. РЕАЛИЗАЦИЯ 20
   1. Вывод 22
4. ОТЛАДКА И ТЕСТИРОВАНИЕ 23
   1. Термины и определения 23
   2. Тестирование «белого ящика» и «черного ящика» 24
   3. Отладка программ 25
   4. Основные возможности программы «Моделирование работы самолета» курсовой работы 25
   5. Вывод 27
5. ВНЕДРЕНИЕ И СОПРОВОЖДЕНИЕ 28
   1. Виды программных документов 28
   2. Пояснительная записка 30
   3. Руководство пользователя 31
   4. Руководство системного программиста 31
   5. Вывод 32

ЗАКЛЮЧЕНИЕ 33

ЛИТЕРАТУРА 34

ПРИЛОЖЕНИЕ А Техническое задание 35

ПРИЛОЖЕНИЕ Б Диаграмма вариантов использования 37

ПРИЛОЖЕНИЕ В Диаграмма топологии 38

ПРИЛОЖЕНИЕ Г Диаграмма состояний 39

ПРИЛОЖЕНИЕ Д Диаграмма активности 40

ПРИЛОЖЕНИЕ Е Диаграмма последовательности действий 41

ПРИЛОЖЕНИЕ Ж Диаграмма классов 42

ПРИЛОЖЕНИЕ З Диаграмма кооперации 43

ПРИЛОЖЕНИЕ И Диаграмма компонентов 44

ПРИЛОЖЕНИЕ К Пояснительная записка 45

ПРИЛОЖЕНИЕ Л Описание используемых классов 47

ПРИЛОЖЕНИЕ М Документация, созданная IntelliJ IDEA 48

ПРИЛОЖЕНИЕ Н Текст программы 54

ПРИЛОЖЕНИЕ О Описание использования программы 60

**ВВЕДЕНИЕ**

Данная работа посвящена одному из самых перспективных на сегодняшний день языков программирования – языку Java. Технологию Java можно без преувеличения назвать революционной в индустрии разработки программного

обеспечения.

Сегодня создание программного обеспечения представляет собой чрезвычайно тяжелое занятие. Трудности связаны с разнообразием архитектур машин, операционных систем, графических оболочек и т. д. Стремительный рост

технологий, связанных с сетью Internet, дополнительно усложняет эту задачу. К

сети Internet подключены компьютеры самых разных типов - IBM PC, Macintosh,

рабочие станции Sun и другие. Даже в рамках IBM-совместимых компьютеров,

существует несколько платформ, например, MS Windows 9x/Me/XP/NT/2000, OS/2, Solaris, различные разновидности операционной системы UNIX графической оболочкой X­Windows и т. д. Все эти системы образуют единую сеть, которая должна работать как одно целое, обеспечивая при этом высокий уровень

безопасности информации. Под влиянием указанных факторов резко возрастает

уровень требований, предъявляемый к программному обеспечению.

Современные приложения должны быть безопасны, работать в распределенной среде, быть нейтральны к архитектуре. Платформа Java располагает мощной встроенной системой безопасности, включающая API-интерфейсы и механизмы, которые используются для обеспечения безопасности апплетов и приложений Java. Среди особенностей языка программирования Java, которые обеспечивают надежность и безопасность можно выделить:

* архитектурную независимость;
* интерпретируемость;
* переносимость, отсутствие указателей;
* строгую типизацию;
* эффективный механизм обработки исключений.

Важными неотъемлимыми компонентами системы безопасности Java являются **Java Cryptography Architecture (JCA)** – архитектура шифрования Java, а также **Java Cryptography Extension (JCE)** – криптографическое расширение Java.

Все эти факторы привели к необходимости нового взгляда на сам процесс создания и распределения приложений на множестве машин различной архитектуры. Требования к переносимости заставили отказаться от традиционного способа создания и доставки бинарных файлов, содержащих машинные коды и, следовательно, привязанных к определенной платформе. Созданная компанией Sun Microsystems система разработки Java удовлетворяет всем этим требованиям.

**Цели данной курсовой работы:**

* углубление знаний в области разработки продуктов на языке Java;
* описание и рассмотрение особенностей языка Java;
* анализ примеров программ, которые показывают все вышеперечисленные достоинства этого языка;
* формирование навыков разработки и оформления программной, текстовой и графической документации;
* развитие навыков устных сообщений по содержанию курсовой работы.

**Задачи курсовой работы:**

* изучить предметную область;
* разработать программный продукт «моделирование работы воздушного шара» с помощью среды разработки IntelliJ IDEA;
* разработать программную документацию;
* разработать диаграммы в среде проектирования Rational Rose.

1. **ИЗУЧЕНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ**
   1. **Постановка задачи**

Данный этап является наиболее важным при создании любого программного продукта. На данном этапе необходимо определить весь функционал создаваемого продукта, чтобы снизить количество ошибок при его создании и дальнейшем использовании. Производится выбор архитектуры, среды разработки программного обеспечения, интерфейса пользователя и анализируются входные и выходные данные.

На данном этапе необходимо ознакомиться с терминами и изучить предметную область.

**Задача** — проблемная ситуация с явно заданной целью, которую необходимо достичь.

**Постановка задача** — точная формулировка решения задачи на персональном компьютере с описанием входной и выходной информации.

**Входная информация** — информация, которую получает человек или устройства.

**Выходная информация** — информация, которая получается после обработки человеком или устройством.

На стадии постановки задачи определяют состав и характеристики исходных данных, устанавливается требуемая форма исходных результатов и условия, которым они должны удовлетворять, выбирается общий подход к решению, осуществляется разбиение задачи на подзадачи, определяется последовательность их решения и так далее. Постановка задачи обычно завершается словесным описанием основных характеристик и особенностей решаемой задачи.

Для того, чтобы приступить к разработке программного продукта, необходимо сформировать требования, которым он должен соответствовать.

Перед разработкой программы и созданием документации необходимо ознакомиться с ЕСПД.

**ЕСПД** — комплекс государственных стандартов РФ, устанавливающих взаимосвязанные правила разработки, оформления и обращения программ и программной документации.

В данной курсовой работе постановка задачи имеет следующую формулировку:

Создать приложение в среде программирования IntelliJ IDEA «моделирование работы воздушного шара».

* 1. **Описание предметной области**

Этот этап позволяет проанализировать предметную область. На данном этапе необходимо разобраться с наиболее значимыми определениями предметной области, чтобы облегчить процесс разработки программного обеспечения.

**Java-апплет** — прикладная программа, чаще всего написанная на языка программирования Java в форме байт кодов .

**Модель** — это система, исследование которой служит средством для получения информации о другой системе.

**Воздушный шар** — летательный аппарат в котором для полета используется газ, который легче воздуха. Состоит из заполненной газом оболочки и прикреплённой к ней корзины или прицепной кабины.

* 1. **Вывод**

На этапе изучения предметной области были раскрыты понятия задачи, постановки задачи, входной и выходной информации, ЕСПД, Java-апплет, модель, воздушный шар .

Также на данном этапе разработан программный документ «Техническое задание» (ПРИЛОЖЕНИЕ А).

1. **ПРОЕКТИРОВАНИЕ**

Несмотря на значимость рассмотренных ранее этапов, этап проектирования позволяет определить спецификации создаваемого программного продукта, обеспечить понимание архитектуры, интерфейса, различных характеристик. Именно данный этап позволяет определить жизненный цикл программного продукта и разобраться в его особенностях.

На данном этапе также необходимо рассмотреть некоторые термины.

**Проектирование** — процесс определения архитектуры, компонентов, интерфейсов и других характеристик системы или её части. Результатом проектирования является проект.

**Проект**— целостная совокупность моделей, свойств или характеристик, описанных в форме, пригодной для реализации системы.

**Архитектура**— общее описание структуры и функций ЭВМ на уровне, достаточном для понимания принципов работы и системы команд ЭВМ, не включающее деталей технического и физического устройства компьютера.

**Жизненный цикл программного продукта**— промежуток времени, в который данный программный продукт будет актуален.

**Интерфейс** — совокупность средств, методов и правил взаимодействия (управления, контроля и т.д.) между элементами системы

Существуют два стиля проектирования: эволюционное и предварительное проектирование.

Методология Extreme Programming (XP) бросила вызов многим устоявшимся представлениям о разработке программного обеспечения. Пожалуй, наиболее противоречивой идеей является отказ от предварительного проектирования в пользу более эволюционного подхода. Противники XP считают, что это возврат к разработкам типа «code and fix» («пишем и правим»). Для приверженцев же новой методологии это отказ от техник проектирования (например, UML), их принципов. Незачем беспокоиться о проектировании, считают они. Достаточно внимательно «вслушиваться» в свой код, и проектирование образуется само собой.

В большинстве случаев эволюционное проектирование – это нечто ужасное. В конце концов, все равно вместо дизайна системы вы получаете просто набор из специфических решений, каждое из которых затрудняет дальнейшие изменения в программном коде. Часто это вообще нельзя назвать дизайном. Будет не только сложнее вносить в систему изменения, но и отыскивать и исправлять ошибки, которые начинают множиться с катастрофической быстротой.

Предварительное проектирование – полная противоположность эволюционному. При разработке ПО проектировщики заранее продумывают все основные вопросы. При этом они не пишут программный код, поскольку не создают программный продукт, а только разрабатывают его дизайн. В своей работе они могут использовать такие техники, как UML, что позволяет им абстрагироваться от некоторых подробностей разработок, относящихся непосредственно к программированию. Как только проектный план готов, его можно передавать в другой отдел (или даже в другую компанию), где будут вестись работы по непосредственному созданию системы.

* 1. **Проектирование программного обеспечения при структурном подходе**

При проектировании сложного программного обеспечения прежде всего необходимо определить структурные компоненты и связи между ними. Полученная в результате структура ПО должна быть представлена в виде структурной или функциональной схем и спецификаций ее компонентов.

* + 1. Структурная схема разрабатываемого программного обеспечения. Структурной называют схему, отражающую состав и взаимодействие по управлению частей разрабатываемого программного обеспечения.

Структурная схема определяется архитектурой разрабатываемого ПО.

Разработку структурной схемы программы обычно выполняют методом пошаговой детализации.

Структурные схемы пакетов программ разрабатывают для каждой программы пакета по отдельности, поскольку организация программ в пакеты не предусматривает передачи управления между ними.

Компонентами структурной схемы программной системы или программного комплекса могут служить программы, подсистемы, базы данных, библиотеки ресурсов и т.п.

Пример структурной схемы программного комплекса для решения математических задач приведен на рисунке 1.

Диспетчер

Блок вывода

результатов

Блок решения

Рисунок 1 – Пример структурной схемы программного комплекса

Как правило, для программных систем разрабатывается схемы, которая дает более полное представление о проектируемом программном обеспечении с точки зрения взаимодействия его компонентов между собой и с внешней средой.

* + 1. **Функциональная схема.**

Функциональная схема (ГОСТ 19.701 – 90) – это схема взаимодействия компонентов программного обеспечения с описанием информационных потоков, состава данных в потоках и указанием используемых файлов и устройств. Для изображения функциональных схем используют специальные обозначения, установленные стандартом, которые показаны в таблице 1.

Таблица 1 – Обозначения элементов функциональных схем

|  |  |  |
| --- | --- | --- |
| **Название блока** | **Обозначение** | **Назначение блока** |
| Сохраненные данные |  | Для обозначения таблиц и других структур данных, которые должны быть сохранены без уточнения типа устройства |
| Оперативное запоминающее устройство |  | Для обозначения таблиц и других структур данных, хранящихся в оперативной памяти |
| Запоминающее устройство с прямым доступом |  | Для обозначения таблиц и других структур данных, хранящихся на магнитных дисках |
| Документ |  | Для обозначения таблиц и других структур данных, выводимых на печать |
| Ручной ввод |  | Для обозначения ручного ввода данных с клавиатуры |
| Дисплей |  | Для обозначения данных, выводимых на дисплей компьютера |

Функциональные схемы более информативны, чем структурные.

На рисунке 2 приведена функциональная схема программного комплекса, реализующего различные методы сортировки массивов.

Программы

сортировки

Программы

сортировки

Программы

сортировки

Программы

сортировки

Программы

сортировки

3,98

Результат

Результат

Результат

Результат

Данные

Данные

Данные

Данные

Рисунок 2 – Пример функциональной схемы программного комплекса

* + 1. **Метод пошаговой детализации при составлении алгоритмов.**

Метод пошаговой детализации реализует нисходящий подход к программированию и предполагает пошаговую разработку алгоритма. Можно выделить следующие этапы:

1. Создается описание программы в целом. Определяются основные логические шаги, требуемые для решения задачи, даже пока известно, как их выполнить. Эти логические шаги могут отражать различные физические способы решения или могут быть удобными групповыми именами для тех действий, выполнение которых представляется довольно смутно. Последовательности шагов, требуемых для решения задачи, записываются на обычном языке или на псевдокоде.
2. В общих терминах детализируется описание шагов, введенных на этапе 1. В детализированное описание может входить обозначение циклических структур, в то время как действия внутри циклов могут по-прежнему оставаться неясными. Таким образом, выполняются только общие эскизы сложных действий.
3. На этом и последующих уровнях в виде последовательных итераций производятся те же действия, что описаны на этапе 2. При каждой новой итерации уточняются детали, оставшиеся неясными после предыдущих итерация, и создаются более определенные описания. По мере выполнения итераций неопределенные детали становятся все проще и проще, так что на каком-то этапе могут быть полностью описаны.
4. Разработка завершена: в модульном виде получено описание требуемой программы. Перевод этого описания в программу на конкретном языке программирования должен быть достаточно простой задачей.
   * 1. **CASE-технологии.**

CASE-технологии (Computer-Aided Software/System Engineering – разработка программного обеспечения/систем с использованием компьютерной поддержки) – это реализованные в виде программных продуктов технологические системы, ориентированные на создание сложных программных систем и поддержку их полного жизненного цикла или его основных этапов. В настоящее время CASE-технологии используются не только для производства ПП, но и как мощный инструмент решения исследовательских и проектных задач (структурный анализ предметной области, моделирование деловых предложений с целью решения задач оперативного и стратегического планирования и управления ресурсами.)

CASE-технологии начали развиваться в связи с развитием методологии структурного программирования. Их развитие стало возможным благодаря тому, что формализация в структурном программировании оказалась наиболее приемлемой для автоматизации. Таким образом, CASE-средства являются результатом эволюционного развития отрасли инструментальных (или технологических) средств.

CASE-средства обладают следующими основными достоинствами:

* повышают качество создаваемого ПО с помощью средств автоматического контроля;
* ускоряют процесс проектирования и разработки;
* позволяют за короткое время создавать прототип будущей системы, что позволяет на ранних этапах оценить желаемый результат;
* освобождают разработчика от рутинной работы, частично генерируя коды программ;
* поддерживают технологии повторного использования компонентов ПО;
* поддерживают развитие и сопровождение разработки.

При использовании CASE-технологий изменяются фазы жизненного цикла программного продукта, как показано в таблице 2.

Таблица 2 – Сравнительная характеристика этапов жизненного цикла ПО

|  |  |
| --- | --- |
| **Традиционная технология** | **CASE-технология** |
| Анализ | Прототипирование |
| Проектирование | Проектирование спецификаций |
|  | Контроль проекта |
| Кодирование | Кодогенерация |
| Тестирование | Системное тестирование |
| Сопровождение | Сопровождение |

Наиболее просто автоматизируемыми оказались стадии «контроль проекта» и «кодогенерация», хотя все остальные этапы жизненного цикла ПО также поддерживаются CASE-технологиями. Кроме изменения содержания фаз, существенно изменилось распределение трудозатрат по фазам, как показано в таблице 3.

Таблица 3 – Распределение трудозатрат по фазам жизненного цикла ПО

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Технология** | **Этапы разработки, %** | | | |
| **Анализ** | **Проектирование** | **Кодирование** | **Тестирование** |
| Традиционная | 20 | 15 | 20 | 45 |
| CASE-1 | 30 | 30 | 15 | 25 |
| CASE-11 | 40 | 40 | 5 | 15 |

Таблица 4 содержит сравнительную характеристику целей и содержания этапов жизненного цикла ПО при традиционной разработке и с помощью CASE-технологий.

Таблица 4 – Цели и содержание этапов жизненного цикла ПО

|  |  |  |
| --- | --- | --- |
| **№ п/п** | **Традиционная разработка** | **CASE-технология** |
| 1 | Основные усилия – на кодирование и тестирование | Основные усилия – на анализ и проектирование |
| 2 | «Бумажные» спецификации | Быстрое итеративное прототипирование |
| 3 | Ручное кодирование | Автоматическая кодогенерация |
| 4 | Ручное документирование | Автоматическая генерация документации |
| 5 | Тестирование кодов | Автоматический контроль проекта |
| 6 | Сопровождение кодов | Сопровождение спецификаций проектирования |

CASE-технология базируется на спиральной модели жизненного цикла ПО. На начальных этапах жизненного цикла (анализ требования, проектирование спецификаций, предварительное и детальное проектирование) проверяется и обосновывается реализуемость технических решений путем создания прототипов. Эта работа повторяется на каждом витке спирали, причем каждый следующий виток характеризуется более высокой степенью детализации ПО. Окончанием витка является уточнение целей и характеристик проекта и планирование работ следующего витка спирали. Тем самым реализуется нисходящий принцип проектирования.

Чем же принципиально CASE-технология отличается от традиционной технологии разработки ПО? Девизом разработчиков CASE-технологий является фраза «одна картинка стоит тысячи слов». Поэтому при использовании CASE-средств функционирование объекта (разрабатываемого ПО) отражается в различных схемах, таблицах, диаграммах и т.д.

Большинство CASE-технологий основано на парадигме методология/метод/нотация/средство.

**Методология**на основе некоторого подхода определяет шаги работы, их последовательность, а также правила распределения и назначения методов.

**Метод** определяет способ достижения той или иной цели.

**Нотацией** называют систему обозначений, используемых для описания структуры системы, элементов данных, этапов обработки и других компонентов. Нотации могут быть графические (представление моделей в виде диаграмм, графов и т.п.) и текстовые (описания моделей на формальных и естественных языках).

**Средства** – инструментарий для поддержки методов. Эти инструменты обеспечивают работу пользователей-разработчиков при создании и редактировании проекта в интерактивном режиме, выполняют проверки соответствия компонентов и кодируют на некотором языке программирования модули ПО.

* 1. **Проектирование программного обеспечения при объектном подходе**

Задачи проектирования включают в себя две составляющие: логическое и физическое проектирование программных продуктов. Логическое проектирование заключается в разработке классов для реализации экземпляров – объектов. Для этого требуется подробное описание полей и методов классов, а также связей между ними. Для этого используются статические диаграммы классов и объектов, динамические – последовательностей состояний и кооперации. Физическое проектирование предполагает построение программных компонентов из ранее определенных классов и объектов и размещение их на конкретных вычислительных устройствах. Разрабатываемые на этом этапе диаграммы – компонентов и развертывания [1]

* 1. **Вывод**

На этапе проектирования было рассмотрено понятия проектирование, проект, архитектура, жизненный цикл программного продукта, интерфейс.

Также на данном этапе были созданы диаграммы UML, а именно: диаграмма вариантов использования (ПРИЛОЖЕНИЕ Б), диаграмма топологии (ПРИЛОЖЕНИЕ В), диаграмма состояний (ПРИЛОЖЕНИЕ Г), диаграмма активности (ПРИЛОЖЕНИЕ Д), диаграмма последовательности действий (приложение Е), диаграмма классов (ПРИЛОЖЕНИЕ Ж), диаграмма кооперации (приложение З) и диаграмма компонентов (ПРИЛОЖЕНИЕ И).

Помимо всех диаграмм, была также создана пояснительная записка с алгоритмом решения задачи курсовой работы (ПРИЛОЖЕНИЕ К), описание используемых классов (ПРИЛОЖЕНИЕ Л) и документация, созданная IntelliJ IDEA (ПРИЛОЖЕНИЕ М).

1. **РЕАЛИЗАЦИЯ**

После осуществления предыдущих этапов был начат процесс реализации программного продукта в объектно-ориентированной среде IntelliJ IDEA , основу которой составляет язык Java. Данная среда имеет ряд преимуществ, таких как:

— все программы получают улучшенную отладку;

— простой визуальный интерфейс;

— простые и функциональные способы форматирования баз данных;

— возможность самостоятельного построения элементов приложения.

К минусам данной среды можно отнести не очень хорошую оптимизацию на некоторых версиях операционных систем и ошибки, возникающие при работе с данной средой.

Необходимо также рассмотреть особенности языка Pascal.

Для того, чтобы приступить к процессу создания данного раздела необходимо разобраться в некоторых понятиях.

**Java** — объектно-ориентированный язык, удобный и надёжный в эксплуатации благодаря таким своим достоинствам, как многозадачность, поддержка протоколов Internet и многоплатформенность. Java – это интерпретируемый язык, и каждая Java-программа компилируется для гипотетической машины, называемой Виртуальная Машина Java. Результатом такой компиляции является байт-код Java, который в свою очередь может выполняться на любой операционной системе при условии наличия там системы времени выполнения Java, которая интерпретирует байт-код в реальный машинный код конкретной системы.

Однако, такая универсальность данной технологии рождает недостаток –

требовательность к ресурсам компьютера. Так как Java-программы не содержат

машинного кода и при их запуске включается в работу система времени

выполнения Java, их производительность заметно ниже, чем у обычных программ,

составленных, например, на языке программирования C++. Данный недостаток

становится с течением времени всё менее ощутим, в следствии роста

вычислительной мощности компьютерных систем.

Язык Java является объектно-ориентированным и поставляется с достаточно

объемной библиотекой классов. Библиотеки классов Java значительно упрощают

разработку приложений, предоставляя в распоряжение программиста мощные

средства решения распространенных задач. Поэтому программист может больше

внимания уделить решению прикладных задач, а не таких, как, например,

организация динамических массивов, взаимодействие с операционной системой или реализация элементов пользовательского интерфейса.

**Операционная система** — совокупность программ, управляющих работой всех устройств персонального компьютера и процессом выполнения прикладных программ.

**Программирование** — процесс создания компьютерных программ, путем создания четного алгоритма действий.

**Объектно-ориентированное программирование (ООП)** — методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования.

**Приложение** — программа, предназначенная для выполнения определённых задач и рассчитанная на непосредственное взаимодействие с пользователем.

**Программа**  — комбинация компьютерных инструкций и данных, позволяющая аппаратному обеспечению вычислительной системы выполнять вычисления или функции управления.

**IntelliJ IDEA** — интегрированная среда разработки программного обеспечения на многих языках программирования, в частности Java, JavaScript, Python, разработанная компанией JetBrains.

Данная программа имеет следующий интерфейс, который состоит из:

* главное окно программы. В нем находится основное меню и панели инструментов;
* менеджер проектов. В нем находятся все проекты, с которыми уже велась разработка;
* панель навигатора. Показывает открытый класс;
* кно проектов. Здесь можно создавать и изменять классы, писать программный код;
* меню. Позволяет быстро создать или открыть проект, скомпилировать его.

Описание среды разработки **IntelliJ IDEA**:

**IntelliJ IDEA** — [интегрированная среда разработки программного обеспечения](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D1%80%D0%B5%D0%B4%D0%B0_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8) на многих языках программирования, в частности [Java](https://ru.wikipedia.org/wiki/Java), [JavaScript](https://ru.wikipedia.org/wiki/JavaScript), [Python](https://ru.wikipedia.org/wiki/Python), разработанная компанией [JetBrains](https://ru.wikipedia.org/wiki/JetBrains).

Первая версия появилась в январе [2001 года](https://ru.wikipedia.org/wiki/2001_%D0%B3%D0%BE%D0%B4) и быстро приобрела популярность, как первая среда для Java с широким набором интегрированных инструментов для [рефакторинга](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D1%84%D0%B0%D0%BA%D1%82%D0%BE%D1%80%D0%B8%D0%BD%D0%B3), которые позволяли программистам быстро реорганизовывать исходные тексты программ. Дизайн среды ориентирован на продуктивность работы программистов, позволяя сконцентрироваться на функциональных задачах, в то время как IntelliJ IDEA берёт на себя выполнение рутинных операций.

Начиная с шестой версии продукта IntelliJ IDEA предоставляет интегрированный инструментарий для разработки графического пользовательского интерфейса. Среди прочих возможностей, среда хорошо совместима со многими популярными [свободными](https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D0%BE%D0%B1%D0%BE%D0%B4%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) инструментами разработчиков, такими как [CVS](https://ru.wikipedia.org/wiki/Concurrent_Versions_System), [Subversion](https://ru.wikipedia.org/wiki/Subversion), [Apache Ant](https://ru.wikipedia.org/wiki/Apache_Ant), [Maven](https://ru.wikipedia.org/wiki/Maven) и [JUnit](https://ru.wikipedia.org/wiki/JUnit). В феврале 2007 года разработчики IntelliJ анонсировали раннюю версию плагина для поддержки программирования на языке [Ruby](https://ru.wikipedia.org/wiki/Ruby).

Начиная с версии 9.0, среда доступна в двух редакциях: **Community Edition** и **Ultimate Edition**. Community Edition является полностью [свободной](https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D0%BE%D0%B1%D0%BE%D0%B4%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) версией, доступной под лицензией [Apache 2.0](https://ru.wikipedia.org/wiki/Apache_License), в ней реализована полная поддержка [Java SE](https://ru.wikipedia.org/wiki/Java_SE), [Groovy](https://ru.wikipedia.org/wiki/Groovy), [Scala](https://ru.wikipedia.org/wiki/Scala_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)), а также интеграция с наиболее популярными [системами управления версиями](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F_%D0%B2%D0%B5%D1%80%D1%81%D0%B8%D1%8F%D0%BC%D0%B8). В редакции Ultimate Edition, доступной под коммерческой лицензией, реализована поддержка [Java EE](https://ru.wikipedia.org/wiki/Java_EE), [UML](https://ru.wikipedia.org/wiki/UML)-диаграмм, подсчёт [покрытия кода](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BA%D1%80%D1%8B%D1%82%D0%B8%D0%B5_%D0%BA%D0%BE%D0%B4%D0%B0), а также поддержка других систем управления версиями, языков и [фреймворков](https://ru.wikipedia.org/wiki/%D0%A4%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA).

* 1. **Вывод**

На данном этапе были рассмотрены понятия программирование, операционная система, объектно-ориентированное программирование, приложение, программа, а так же были рассмотрены особенности среды для программирования IntelliJ IDEA. Описана среда программирования IntelliJ IDEA, язык Java, рассмотрены особенности и преимущества.

Был создан программный документ «Текст программы» (ПРИЛОЖЕНИЕ Н).

1. **ОТЛАДКА И ТЕСТИРОВАНИЕ**

Данный этап помогает исключить ошибки, которые могут возникнуть во время использования программы путем последовательного выполнения программного кода.

Данный этап раскрывает следующие термины.

**Тестирование программного обеспечения** – это процесс проверки соответствия заявленных к продукту требований и реально реализованной функциональности, осуществляемый путем наблюдения за его работой в искусственно созданных ситуациях и на ограниченном наборе тестов, выбранных определенным образом.

**Тестирование** — процесс выполнения программ с целью обнаружения факта наличия ошибок.

**Отладка** — этап разработки компьютерной программы, на котором обнаруживают, локализуют и устраняют ошибки.

Есть различные подходы и средства отладки; в качестве основного средства используются встроенные программные отладчики, которые включают в себя пользовательский интерфейс для пошагового выполнения программы: оператор за оператором, функция за функцией, с остановками на некоторых строках исходного кода или при достижении определённого условия.

Отладка программ заключается в проверке правильности работы программы и аппаратуры. Программа, не содержащая синтаксических ошибок, тем не менее, может содержать логические ошибки, не позволяющие программе выполнять заложенные в ней функции. Логические ошибки могут быть связаны с алгоритмом программы или с неправильным пониманием работы аппаратуры, подключённой к портам микроконтроллера.

**Уровни тестирования:**

* **модульное тестирование**. Тестируется минимально возможный для тестирования компонент, например, отдельный класс или функция;
* **интеграционное тестирование**. Проверяется, есть ли какие-либо проблемы в интерфейсах и взаимодействии между интегрируемыми компонентами, например, не передается информация, передается некорректная информация;
* **системное тестирование**. Тестируется интегрированная система на ее соответствие исходным требованиям.
  1. **Термины и определения**

Выполнение программы с целью обнаружения ошибок называется **тестированием**. Виды ошибок и способы их обнаружения приведены в таблице 5.

Таблица 5 – Виды программных ошибок и способы их обнаружения

|  |  |
| --- | --- |
| **Виды программных ошибок** | **Способы их обнаружения** |
| Синтаксические | Статический контроль и диагностика компиляторами и компоновщиками |
| Ошибки выполнения, выявляемые автоматически:   * переполнение, защита памяти; * несоответствие типов; * зацикливание | Динамический контроль:  аппаратурой процессора;  run-time системы программирования;  операционной системой – по превышению лимита времени |
| Программа не соответствует спецификации | Целенаправленное тестирование |
| Спецификация не соответствует требованиям | Испытания, бета – тестирование |

* 1. **Методология Agile**

**Гибкая методология разработки (от англ. - Agile software development)** - манифест, определяющий способ мышления и содержащий основные ценности и принципы, на которых базируется несколько подходов (фреймворков, от англ. **framework**— каркас, структура) к разработке программного обеспечения (хотя в последнее время идет тенденция и попытки применения гибкой методологии разработки к иным направлениям деятельности, не только в части информационных технологий), подразумевающих под собой интерактивную разработку, периодического (динамического) предоставления (обновления) требований от Заказчика и их реализацию посредством самоорганизующихся рабочих групп, сформированных из экспертов различного профиля (разработчики, тестировщики, внедренцы и т.д.). Такой перевод Agile, как "гибкая методология разработки" не совсем корректен т.к. обычно Agile не называют методологией, а вот подходы на основе данного манифеста и есть методологии, но с точки зрения Agile их называют - фреймворки. На данный момент существует множество фреймворков (методологий), подходы которых базируются на гибкой методологии разработки, например такие, как: Scrum, Extreme programming, FDD, DSDM и т.д.

**4.2.1Разновидность методологий гибкой разработки**

На основании ценностей и принципов, определенных в Agile Manifesto были сформированы следующие гибкие методологии разработки:

**Agile Modeling (AM)** — данный подход в основе своей определяет процедуры моделирования (в т.ч. проверка модели кодом) и документирования в рамках разработки программного обеспечения. В меньшей степени описаны процедуры проектирования и построения диаграмм на UML. также не затронуты области разработки, тестирования, управления проектом, развертывания и сопровождения.

**Agile Unified Process (AUP)** - унифицированная версия методологии RUP (IBM Rational Unified Process), которая была сформирована Скоттом Амблером. AUP определяет модель создания программного обеспечения в рамках бизнес-приложений.

**Agile Data Method (ADM)** — набор итеративных методик гибкой разработки программного обеспечения, в рамках которых делается упор на формирование требований и решений посредством сотрудничества различных кросс-функциональных команд.

**Dynamic Systems Development Method (DSDM)** - итеративный и инкрементный подход, базирующийся концепции быстрой разработки приложений - Rapid Application Development (RAD), упор в котором делается на максимальное привлечение конечного пользователя к разработке программного продукта.

**Essential Unified Process (EssUP)** - подход, разработанный Иваром Якобсоном (Ivar Jacobson), содержит в себе методы итеративной разработки программного обеспечения, с упором на архитектуру продукта и наработанные практики команды (по сути заимствованные из RUP, CMMI и Agile Development). Идея заключается в том, что вы используете только те практики и методы, которые применимы в конкретной ситуации. На основе выбранных методов и практик определяется целевой процесс. В отличие от RUP, где все практики и методы взаимосвязаны, в данном случае появляется гибкость и возможность вычленить из всего доступного объема именно необходимые элементы (методы и практики).

**Extreme programming (XP)** - идея экстремального программирования заключается в том, чтобы использовать уже имеющиеся лучшие практики в области разработки программного обеспечения, подняв их на новый (экстремальный) уровень. Например в отличие от обычной практики, когда один программист последовательно проверяет написанный код за своим коллегой, в экстремальном программировании данная проверка осуществляется параллельно, что увеличивает скорость выпуска продукта, но и риски тоже.

**Feature driven development (FDD)** — основное ограничение, которое накладывается в рамках данного подхода, это "каждая функция должна быть реализована не более, чем за две недели". Т.е. если реально разработать функцию за один присест, то это хорошо, в противном случае данная функция должна разбиться на несколько и реализовываться постепенно.

**Getting Real (GR)** — в рамках данного подхода исключены процедуры функциональных спецификаций, использующийся для веб-приложений. Разработка начинается от обратного, изначально разрабатывается интерфейс и дизайн, а потом сама функциональность.

**OpenUP (OUP)** — данный подход определяет итеративно-инкрементальный метод разработки программного обеспечения. Разработан на основе RUP. В рамках данного метода определен жизненный цикл разработки (фаза запуска, фаза уточнения, фаза разработки и передачи заказчику). Благодаря определенной этапности и контрольных точек, повышается эффективность контроля и мониторинга хода реализации проекта, как следствие своевременное принятие решений по проекту.

**Lean software development** - данный подход основан на концепции бережливого управления производственным предприятием (lean production, lean manufacturing).

**Scrum** - один из самых распространенных подходов гибкой разработки программного обеспечения, определяет правила управления процессом разработки с применением существующих практик разработки. Упор осуществляется на вовлеченность Заказчика в процесс (возможность после каждого этапа менять или уточнять требования к создаваемому продукту), что позволяет вовремя определить отклонения и внести необходимые изменения.

* 1. **Основные возможности программы «Моделирование работы воздушного шара» курсовой работы**

Программа «Моделирование работы воздушного шара», выполняемая в рамках данной курсовой работы, имеет вид, изображенный на рисунке 4.

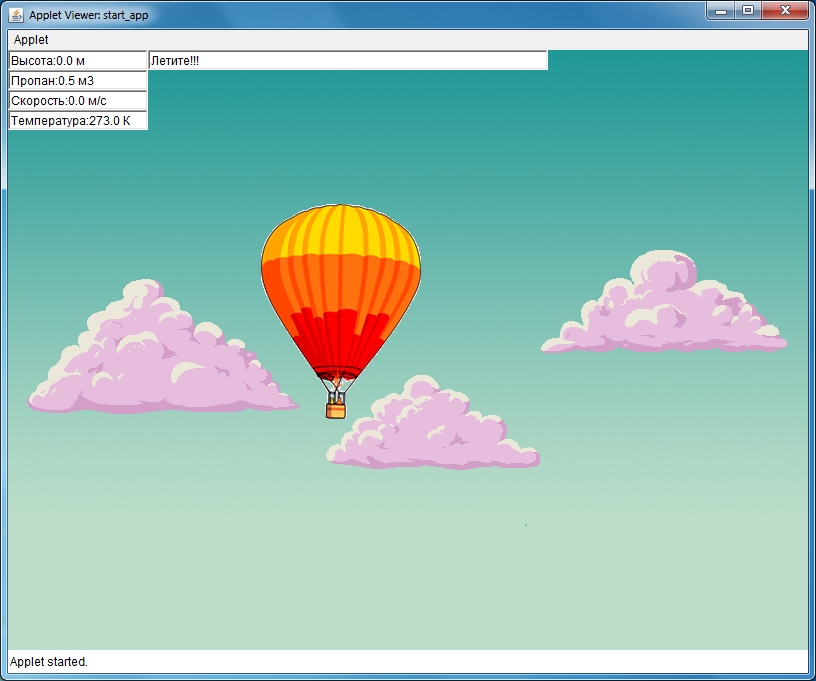


Рисунок 4 – Общий вид программы

Управление воздушным шаром в программе осуществляется клавишами «Space», которая заправляет воздушный шар пропаном(результат заправки пропаном выводится для пользователя). Для того, чтобы набрать скорость и начать полет необходимо нажать клавишу «Вверх» на клавиатуре, скорость и высота начнут увеличиваться. Для прекращения полета необходимо постепенно снизить скорость, для этого необходимо нажать клавишу «Вниз».

Если воздушный шар начнет падение и конется нижнего края карты, произойдет проигрыш. При этом будет выводиться изображение взрыва шара. Результат проигрыша представлен на рисунке 5.

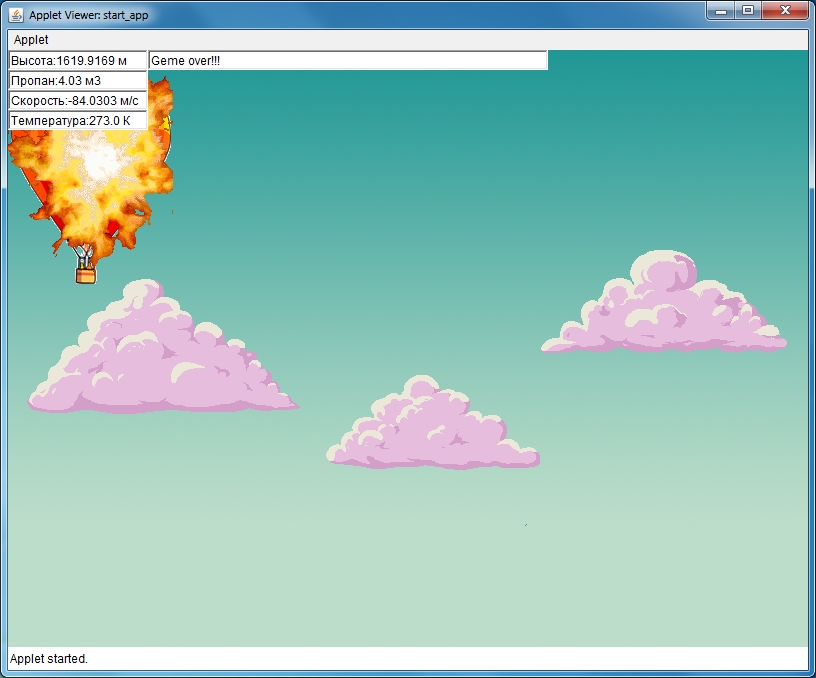


Рисунок 5 – Результат проигрыша

* 1. **Вывод**

На этапе «Тестирование и отладка» были рассмотрены основные термины и определения, раскрыто понятие тестирования программ, описаны основные возможности программы, разрабатываемой в рамках данной курсовой работы.

1. **ВНЕДРЕНИЕ И СОПРОВОЖДЕНИЕ**

Составление программной документации – очень важный процесс. Стандарт, определяющий процессы жизненного цикла программного обеспечения, даже предусматривает специальный процесс, посвященный указанному вопросу. При этом на каждый программный продукт должна разрабатываться документация двух типов: для пользователей различных групп и для разработчиков. Отсутствие документации любого типа для конкретного программного продукта недопустимо.

При подготовке документации не следует забывать, что она разрабатывается для того, чтобы ее можно было использовать, и потому она должна содержать все необходимые сведения.

* 1. **Виды программных документов**

К программным относят документы, содержащие сведения, необходимые для разработки, сопровождения и эксплуатации программного обеспечения. Документирование программного обеспечения осуществляется в соответствии с Единой системой программной документации (ГОСТ 19.XXX). Так, ГОСТ 19.101-77 устанавливает виды программных документов для программного обеспечения различных типов. Ниже перечислены основные программные документы по этому стандарту и указано, какую информацию они должны содержать.

*Спецификация* должна содержать перечень и краткое описание назначения всех файлов программного обеспечения, в том числе и файлов документации на него, и является обязательной для программных систем, а также их компонентов, имеющих самостоятельное применение.

*Ведомость держателей подлинников* (код вида документа – 05) должна содержать список предприятий, на которых хранятся подлинники программных документов. Необходимость этого документа определяется на этапе разработки и утверждения технического задания только для программного обеспечения со сложной архитектурой.

*Текст программы* (код вида документа – 12) должен содержать текст программы с необходимыми комментариями. Необходимость этого документа определяется на этапе разработки и утверждения технического задания.

*Описание программы* (код вида документа – 13) должно содержать сведения о логической структуре и функционировании программы.

*Ведомость эксплуатационных документов* (код вида документа – 20) должна содержать перечень эксплуатационных документов на программу, к которым относятся документы с кодами 30, 31, 32, 33, 34, 35, 46. Необходимость этого документа также определяется на этапе разработки и утверждения технического задания.

*Формуляр* (код вида документа – 30) должен содержать основные характеристики программного обеспечения, комплектность и сведения об эксплуатации программы.

*Описание применения* (код вида документа – 31) должно содержать сведения о назначении программного обеспечения, области применения, применяемых методах, классе решаемых задач, ограничениях для применения, минимальной конфигурации технических средств.

*Руководство системного программиста* (код вида документа – 32) должно содержать сведения для проверки, обеспечения функционирования и настройки программы на условия конкретного применения.

*Руководство программиста* (код документа – 33) должно содержать сведения для эксплуатации программного обеспечения.

*Руководство оператора* (код вида документа – 34) должно содержать сведения для обеспечения процедуры общения оператора с вычислительной системой в процессе выполнения программного обеспечения.

*Описания языка* (код вида документа – 35) должно содержать описания синтаксиса и семантики языка.

*Руководство по техническому обслуживанию* (код вида документа – 46) должно содержать сведения для применения тестовых и диагностических программ при обслуживании технических средств.

*Программа и методика испытаний* (код вида документа – 51) должны содержать требования, подлежащие проверке при испытании программного обеспечения, а также порядок и методы их контроля.

*Пояснительная записка* (код вида документа – 81) должна содержать информацию о структуре и конкретных компонентах программного обеспечения, в том числе схемы алгоритмов, их общее описание, а также обоснование принятых технических и технико-экономических решений.

Составляется на стадии эскизного и технических проектов.

Прочие документы (коды вида документа – 90-99) могут составляться на любых стадиях разработки, т.е. на стадиях эскизного, технического и рабочего проектов.

* 1. **Пояснительная записка**

Пояснительная записка должна содержать всю информацию, необходимую для сопровождения и модификации программного обеспечения: сведения о его структуре и конкретных компонентах, общее описание алгоритмов и их схемы, а также обоснование принятых технических и технико-экономических решений.

Содержание пояснительной записки по стандарту (ГОСТ 19.404–79) должно включать следующие разделы:

* введение;
* назначение и область применения;
* технические характеристики;
* ожидаемые технико-экономические показатели;
* источники, используемые при разработке.

В разделе *Введение* указывают наименование программы и документа, на основании которого ведется разработка.

В разделе *Назначение и область применения* указывают назначение программы и дают краткую характеристику области применения.

Раздел *Технические характеристики* должен содержать следующие подразделы:

* постановку задачи, описание применяемых математических методов и допущений и ограничений, связанных с выбранным математическим аппаратом;
* описание алгоритмов и функционирования программы с обоснованием принятых решений;
* описание и обоснование выбора способа организации входных и выходных данных;
* описание и обоснование выбора состава технических и программных средств на основании проведенных расчетов или анализов.

В разделе *Ожидаемые технико-экономические показатели* указывают технико-экономические показатели, обосновывающие преимущество выбранного варианта технического решения.

В разделе *Источники, использованные при разработке*, указывают перечень научно-технических публикаций, нормативно-технических документов и других научно-технических материалов, на которые есть ссылки в исходном тексте.

Пояснительная записка составляется профессионалами в области разработки программного обеспечения для специалистов того же уровня квалификации. Следовательно, в ней уместно использовать специальную терминологию, ссылаться на специальную литературу и т.п.

* 1. **Руководство пользователя**

В настоящее время часто используют еще один эксплуатационный документ, в который отчасти входит руководство системного программиста, программиста и оператора. Этот документ называют Руководством пользователя. Появление такого документа явилось следствием широкого распространения персональных компьютеров, работая на которых пользователи совмещают в своем лице трех указанных выше специалистов.

Составление документации для пользователей имеет свои особенности, связанные с тем, что пользователь, как правило, не является профессионалом в области разработки программного обеспечения.

Руководство пользователя содержит следующие разделы:

* общие сведения о программном продукте;
* описание установки;
* описание запуска;
* инструкция по работе;
* сообщения пользователю.

Раздел *Общие сведения о программе* обычно содержит наименование программного продукта, краткое описание его функций, реализованных методов и возможных областей применения.

Раздел *Установка* обычно содержит подробное описание действий по установке программного продукта и сообщений, которые при этом могут быть получены.

В разделе *Запуск*, как правило, описаны действия по запуску программного продукта и сообщений, которые при этом могут быть получены.

Раздел *Инструкция по работе* обычно содержит описание режимов работы, форматов ввода-вывода информации и возможных настроек.

Раздел *Сообщения пользователю* должен содержать перечень возможных сообщений, описание их содержания и действий, которые необходимо предпринять по этим сообщениям.

* 1. **Руководство системного программиста**

По ГОСТ 19.503-79 руководство системного программиста должно содержать всю информацию, необходимую для установки программного обеспечения, его настройки и проверки работоспособности. Кроме того, как указывалось выше, в него часто включают и описание необходимого обслуживания, которое раньше приводилось в руководстве оператора (ГОСТ 19.505-79) и/или руководстве по техническому обслуживанию (ГОСТ 19.508-19). В настоящее время данную схему используют для составления руководства системному администратору.

Руководство системного программиста должно содержать следующие разделы:

* общие сведения о программном продукте;
* структура;
* настройка;
* проверка;
* дополнительные возможности;
* сообщения системного программисту.

Раздел *Общие сведения о программе* должен включать описание назначения и функций программы, а также сведения о технических и программных средствах, обеспечивающих выполнение данной программы.

В разделе *Структура программы* должны быть приведены сведения о структуре программы, ее составных частях, о связях между составными частями и о связях с другими программами.

В разделе *Настройка программы* должно быть приведено описание действий по настройке программы на условия практического применения.

В разделе *Проверка программы* должно быть приведено описание способов проверки работоспособности программы, например, контрольные примеры.

В разделе *Дополнительные возможности* должно быть приведено описание дополнительных возможностей программы и способов доступа к ним.

В разделе *Сообщения системного программисту* должны быть указаны тексты сообщений, выдаваемых в ходе выполнения настройки и проверки программы, а также в ходе ее выполнения, описание их содержания и действий, которые необходимо предпринять по этим сообщениям [1].

* 1. **Вывод**

На этапе «Внедрение и сопровождение» были рассмотрены основные виды программных документов в соответствии с ЕСПД и создан программный документ «Описание использования программы» (ПРИЛОЖЕНИЕ)

**ЗАКЛЮЧЕНИЕ**

В результате выполнения курсовой работы были выполнены следующие задачи:

* изучить предметную область;
* разработать программный продукт «моделирование работы воздушного шара» с помощью среды разработки IntelliJ IDEA;
* разработать программную документацию;
* разработать диаграммы в среде проектирования Rational Rose.

Программный продукт «Моделирование работы воздушного шара» был успешно протестирован и приведен в рабочее состояние.

**ЛИТЕРАТУРА**

**Основная**

1. Гагарина Л.Г, Кокорева Е.В, Виснадул Б.Д. Технология разработки программного обеспечения: учебник / под ред. Профессора   
   Л.Г. Гагариной. – М. : ИД «ФОРУМ» – ИНФРА-М, 2008. – 400 с.

**Интернет-ресурсы**

1. Методология Agile [Электронный ресурс]. - URL: http://mahamba.com/ru/gibkaya-metodologiya-razrabotki-agile (дата обращения 10.12.2017)
2. Игра [Электронный ресурс]. – URL: [https://ru.wikipedia.org/wiki/game](https://ru.wikipedia.org/wiki/game%20) (дата обращения 10.12.2017)
3. Постановка задачи [Электронный ресурс]. – URL: <http://dic.academic.ru/dic.nsf/ruwiki/235187> (дата обращения 10.12.2017)
4. Реализация программного продукта [Электронный ресурс]. – URL: <http://studopedia.su/9_21324_realizatsiya-programmnogo-produkta.html> (дата обращения 10.12.2017)
5. Технологии создания компьютерных игр [Электронный ресурс]. – URL: [http://www.bestreferat.ru/referat-183172.html](http://www.bestreferat.ru/referat-183172.html%20) (дата обращения 10.12.2017)
6. Шаблон проектирования [Электронный ресурс]. – URL: [https://ru.wikipedia.org/wiki/Шаблон\_проектирования#Основные](https://ru.wikipedia.org/wiki/Шаблон_проектирования%23Основные%20) (дата обращения 09.12.2017)
7. Технология разработки программного обеспечения [Электронный ресурс]. – URL: http://www.tehprog.ru/index.php\_page=lecture12.html (дата обращения 08.12.2017)
8. IntelliJ IDEA. Java [Электронный ресурс]. – URL:

[https://ru.wikipedia.org/wiki/ IntelliJ IDEA\_and\_Java](https://ru.wikipedia.org/wiki/%20IntelliJ%20IDEA_and_Java%20) (дата обращения 05.12.2017)

**ПРИЛОЖЕНИЕ А**

(информационное)

**Техническое задание**

* 1. **Введение**

Настоящее техническое задание распространяется на разработку программы «Моделирование работы воздушного шара», используемой в развлекательных целях.

* 1. **Основание для разработки**
     1. Программа разрабатывается на основе учебного плана специальности «Прикладная информатика по отраслям».
     2. Наименование работы:

«Моделирование работы воздушного шара».

* + 1. Исполнитель: Скворцов Павел Дмитриевич.
    2. Соисполнители: нет.
  1. **Назначение**

Программа предназначена для использования в развлекательных целях.

* 1. **Требования к программе**
     1. Требования к составу и параметрам технических средств

Система должна работать на IBM – совместимых персональных компьютерах.

Минимальная конфигурация:

* тип процессора Pentium и выше;
* объем оперативного запоминающего

устройства 32 Мб и более;

* объем свободного места на жестком

диске 40 Мб.

Рекомендуемая конфигурация:

* тип процессора Pentium II 400;
* объем оперативного запоминающего

устройства 128 Мб;

* объем свободного места на жестком

диске 60 Мб.

* + 1. Требования к программной совместимости

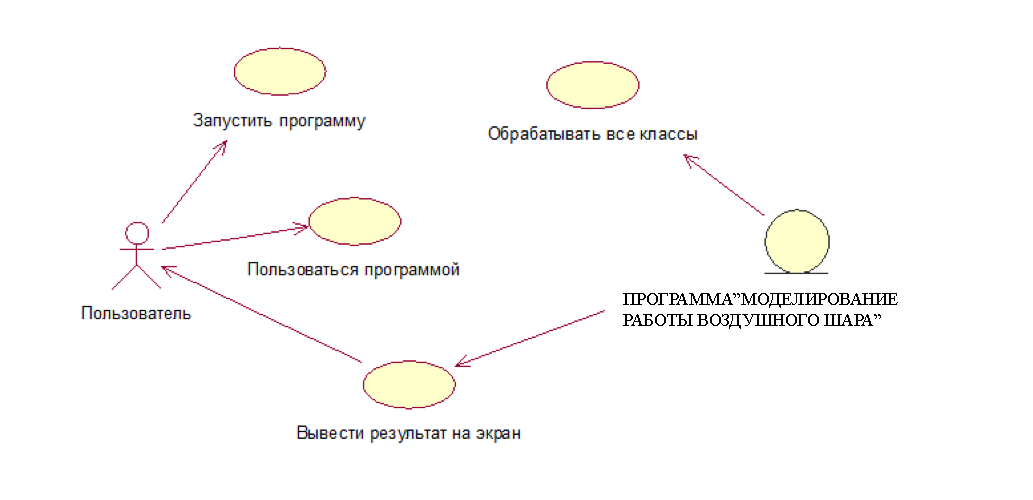
Программа должна работать под управлением семейства операционных систем Win 32 (Windows 95/98/2000ME/XP и т.п.), совместимых с виртуальной машиной Java.

* 1. **Требования к программной документации**
     1. Разрабатываемые программные модули должны быть самодокументированы, т. е. тексты программ должны содержать все необходимые комментарии.
     2. Разрабатываемая программа должна включать справочную информацию о работе программы.
     3. В состав сопровождающей документации должны входить:
        1. Руководство пользователя.

**ПРИЛОЖЕНИЕ Б**

(информационное)

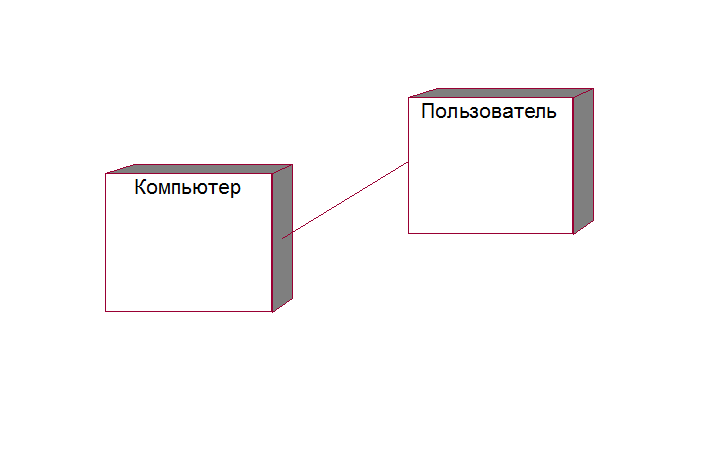
**Диаграмма вариантов использования**



**ПРИЛОЖЕНИЕ В**

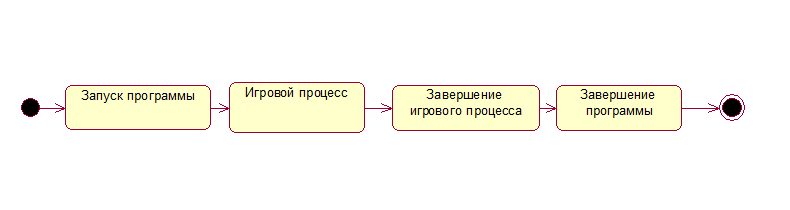
(информационное)

**Диаграмма топологии**

****

**ПРИЛОЖЕНИЕ Г**

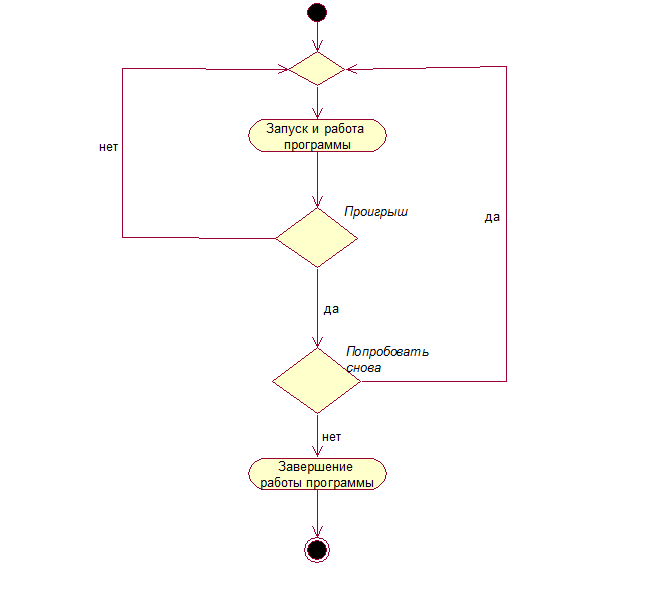
(информационное)

**Диаграмма состояний**

**ПРИЛОЖЕНИЕ Д**

(информационное)

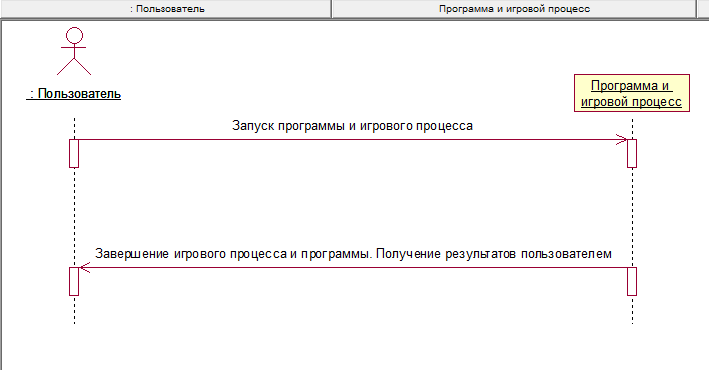
**Диаграмма активности**

****

**ПРИЛОЖЕНИЕ Е**

(информационное)

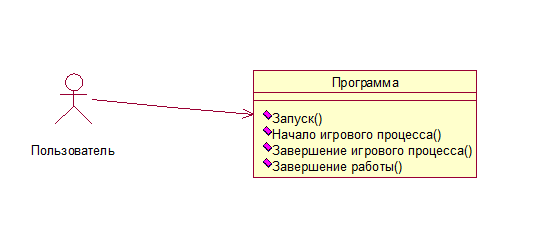
**Диаграмма последовательности действий**

****

**ПРИЛОЖЕНИЕ Ж**

(информационное)

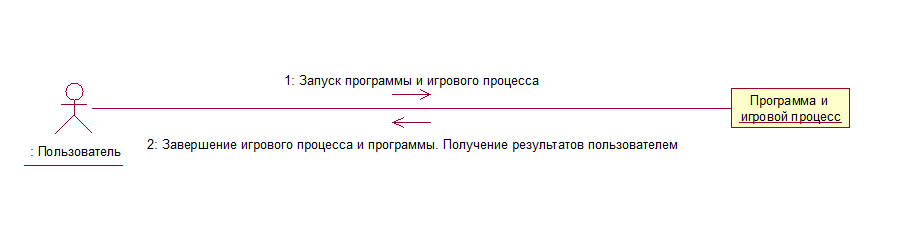
**Диаграмма классов**

****

**ПРИЛОЖЕНИЕ З**

(информационное)

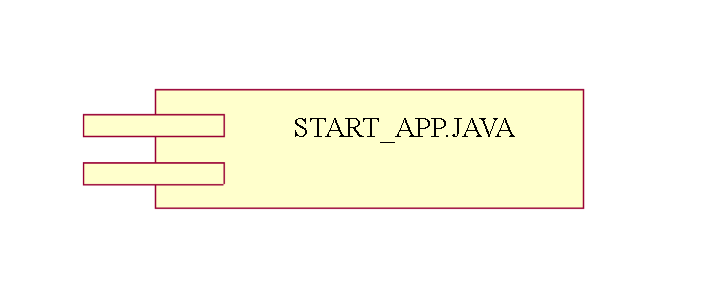
**Диаграмма кооперации**

****

**ПРИЛОЖЕНИЕ И**

(информационное)

**Диаграмма компонентов**

****

**ПРИЛОЖЕНИЕ К**

(информационное)

**Пояснительная записка**

1. **Введение**
   1. **Наименование программы**

Пояснительная записка ГОСТ 19.404-79.

Наименование программы – «Моделирование работы воздушного шара».

Обозначение – СПФ ВГИК. 292.09-01.81.

1. **Назначение и область применения**

Программа используется в развлекательных целях, область применения может быть различна.

**Описание алгоритма работы программы**

Начало

Запуск и работа программы

balloon\_phisical.java

main.java

start\_app.java

DesktopLauncher.java

1

Проигрыш?

нетда

Попробовать снова?

да

нет

Завершение работы программы

Конец

**ПРИЛОЖЕНИЕ Л**

(информационное)

**Описание используемых классов**

**Класс start\_app.java**

Класс предназначен для запуска программы на компьютере. Содержит все необходимые конфигурации для запуска.

**Класс balloon\_phisical.java**

Класс предназначен для описания физических свойств воздушного шара, задания объекту свойств.

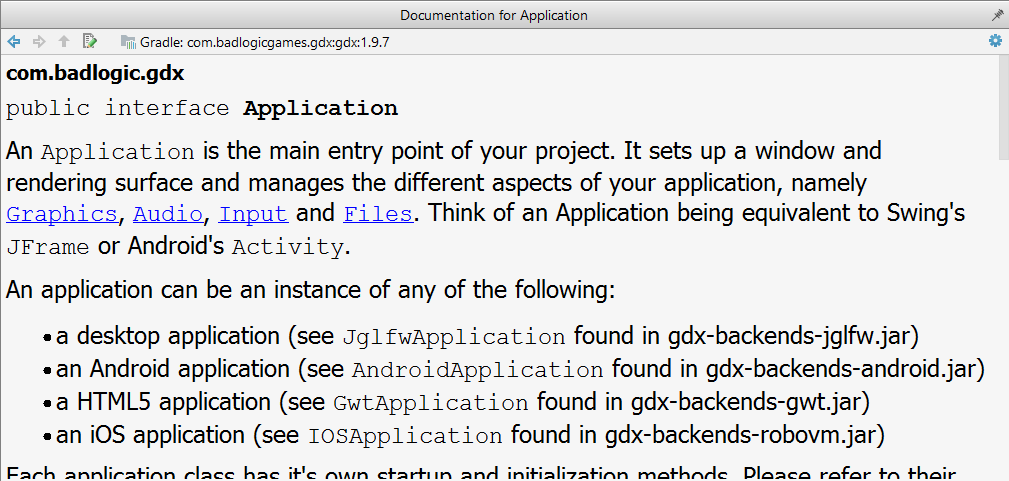
**Класс main.java**

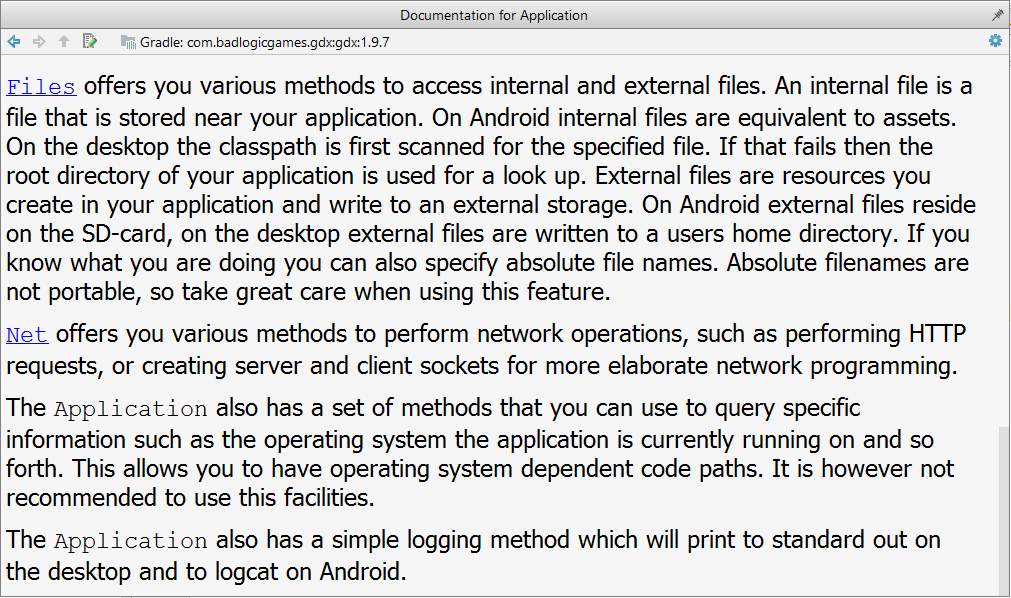
Основной игровой класс, в котором прописаны все основные игровые методы (загрузка в память всех необходимых элементов, отрисовка элементов игрового окна, очистка ресурсов).

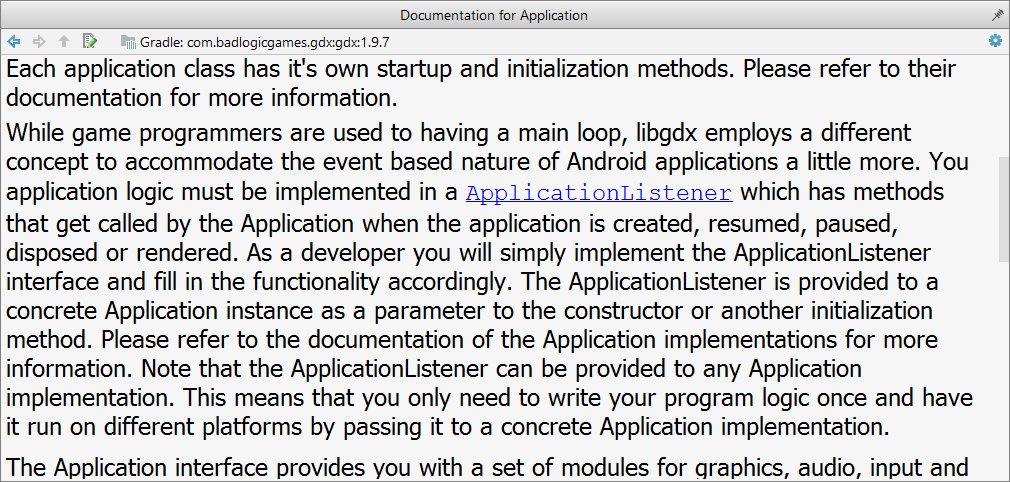
**ПРИЛОЖЕНИЕ М**

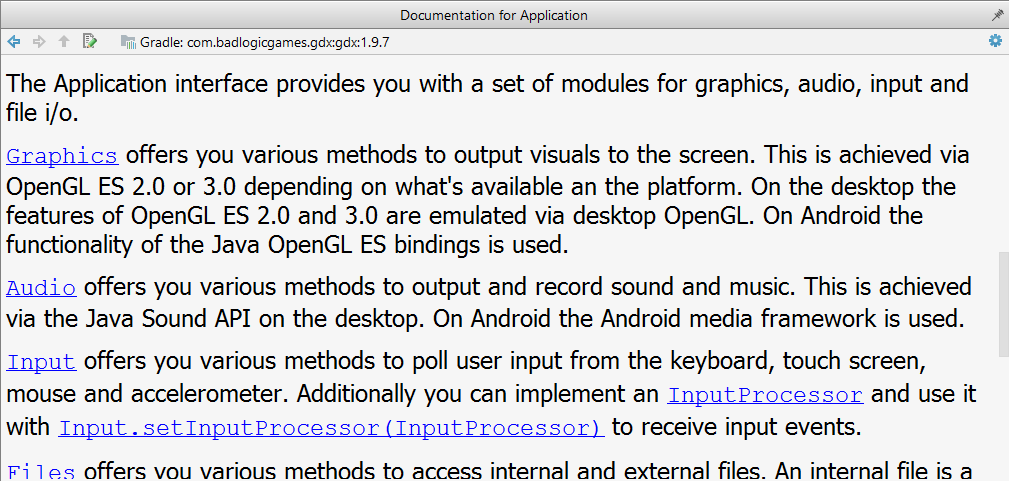
(информационное)

**Документация, созданная IntelliJ IDEA**

****

****

****

****

**ПРИЛОЖЕНИЕ Н**

(информационное)

**Текст программы**

**start\_app.java:**

**import java.applet.\*;**

**import java.awt.\*;**

**import java.awt.event.ActionEvent;**

**import java.awt.event.ActionListener;**

**import java.awt.event.KeyEvent;**

**import java.awt.event.KeyListener;**

**import java.awt.event.MouseAdapter;**

**import java.awt.event.MouseEvent;**

**import java.awt.event.MouseListener;**

**import java.awt.event.MouseMotionAdapter;**

**import javax.swing.JButton;**

**import javax.swing.JFrame;**

**import javax.swing.JPanel;**

**import java.io.ByteArrayOutputStream;**

**import java.io.FileInputStream;**

**import java.io.FileNotFoundException;**

**import java.io.FileOutputStream;**

**import java.io.IOException;**

**import java.io.ObjectInputStream;**

**import java.io.ObjectOutput;**

**import java.io.ObjectOutputStream;**

**import java.math.\*;**

**public class start\_app extends Applet implements KeyListener, Runnable**

**{**

**/\*\***

**\* Автор: Скворцов П.Д.**

**\*/**

**private static final long serialVersionUID = 550417251998005070L;**

**balloon\_phisical ball;**

**Point position;**

**Point position\_d;**

**String nmfile;**

**Image Img;**

**Image Img\_fon;**

**Thread animator\_ball;**

**Thread animator\_fon;**

**volatile boolean pleaseStop;**

**int helth=1;**

**int kl=0;**

**int x = 5, y = 5;**

**int dx = 2, dy = 1;**

**int pos;**

**int derection = 0;**

**int first\_y;**

**int v=0;**

**final TextField text\_1=new TextField("null");**

**final TextField text\_2=new TextField("null");**

**final TextField text\_3=new TextField("null");**

**final TextField text\_4=new TextField("null");**

**final TextField text\_5=new TextField("Начало полета");**

**final Button butt\_1=new Button ("Заправить");**

**final Button butt\_2=new Button ("Отремонтировать");**

**public void init()**

**{**

**setSize(/\*DEFAULT\_WIDTH\*/800,/\*DEFAULT\_HEIGHT\*/ 600);**

**setName("ButtonTest");**

**setLayout(null);**

**first\_y=150;**

**ball= new balloon\_phisical();**

**position=new Point(0,0);**

**position\_d=new Point(0,0);**

**position.x=250;**

**position.y=first\_y-(int)ball.h/40;**

**first\_y=position.y;**

**position\_d.x=(position.x+300)-(int)((position.x+300)/1100)\*1100;**

**position\_d.y=(position.y+373)-(int)((position.y+373)/973)\*973;**

**pos = 10;**

**Img = getImage(getDocumentBase(), "ball.gif");**

**Img\_fon = getImage(getDocumentBase(), "fon\_1.jpg");**

**this.addKeyListener( this);**

**this.setFocusable(true);**

**this.add(text\_1);**

**this.add(text\_2);**

**this.add(text\_3);**

**this.add(text\_4);**

**this.add(text\_5);**

**// this.add(butt\_1);**

**// this.add(butt\_2);**

**text\_1.setBounds(0, 0, 140, 20);**

**text\_2.setBounds(0, 20, 140, 20);**

**text\_3.setBounds(0, 40, 140, 20);**

**text\_4.setBounds(0, 60, 140, 20);**

**text\_5.setBounds(140, 0, 400, 20);**

**butt\_1.setBounds(660, 0, 140, 20);**

**butt\_2.setBounds(660, 20, 140, 20);**

**butt\_1.addActionListener(new ActionListener()**

**{**

**public void actionPerformed(final ActionEvent e)**

**{**

**run();**

**}**

**});**

**butt\_2.addActionListener(new ActionListener()**

**{**

**public void actionPerformed(final ActionEvent e)**

**{**

**ball.event\_without\_fire();**

**}**

**});**

**}**

**public void paint(final Graphics g)**

**{**

**g.drawImage(Img\_fon, 0, 0,this);**

**if (helth==1)**

**g.drawImage(Img,position.x,position.y,this);**

**else**

**{**

**g.drawImage(Img,0,15,this);**

**text\_5.setText("Конец игры");**

**stop();**

**}**

**}**

**public void keyTyped(KeyEvent arg0) {**

**}**

**public void keyPressed(KeyEvent arg0) {**

**Graphics panel=this.getGraphics();**

**int dd=10;**

**int Key = arg0.getKeyCode();**

**switch(Key)**

**{**

**case KeyEvent.VK\_LEFT:**

**if (arg0.isShiftDown())**

**dd=20;**

**if (ball.h>0)**

**position.x=position.x-4;**

**break;**

**case KeyEvent.VK\_RIGHT:**

**if (arg0.isShiftDown())**

**dd=20;**

**if (ball.h>0)**

**position.x=position.x+4;**

**break;**

**case KeyEvent.VK\_UP:**

**ball.event\_fire();**

**break;**

**case KeyEvent.VK\_DOWN:**

**break;**

**case KeyEvent.VK\_SPACE:**

**if (ball.h==0)**

**ball.v\_tank=ball.v\_tank+0.25;**

**break;**

**}**

**}**

**public void keyReleased(KeyEvent arg0)**

**{**

**}**

**public void animate\_ball()**

**{**

**int k;**

**long timeStart;**

**long timeFinish;**

**timeStart=System.currentTimeMillis();**

**while (true)**

**{**

**timeFinish=System.currentTimeMillis();**

**if ((int)(timeFinish-timeStart)>100 )**

**break;**

**}**

**int old\_x=position.x;**

**int old\_y=position.y;**

**if (ball.h>0)**

**ball.event\_without\_fire();**

**position.y=first\_y-(int)ball.h/40;**

**int dd\_x=Math.abs(old\_x-position.x);**

**int dd\_y=Math.abs(old\_y-position.y);**

**nmfile = "ball.gif";**

**Img = getImage(getDocumentBase(), nmfile);**

**int bbb=0;**

**kl=0;**

**Img\_fon = getImage(getDocumentBase(), "fon\_1.jpg");**

**while (position.y<-373)**

**{**

**kl++;**

**position.y=position.y+973;**

**bbb=1;**

**if (kl==6)**

**{**

**helth=0;**

**break;**

**}**

**if (kl==5)**

**{**

**Img\_fon = getImage(getDocumentBase(), "fon\_3.jpg");**

**if (position.y<first\_y)**

**{**

**Img = getImage(getDocumentBase(), "bum.gif");**

**}**

**break;**

**}**

**Img\_fon = getImage(getDocumentBase(), "fon\_2.jpg");**

**}**

**while (position.x<-300)**

**{**

**position.x=position.x+1100;**

**bbb=1;**

**}**

**while (position.x>800)**

**{**

**position.x=position.x-1100;**

**bbb=1;**

**}**

**text\_5.setText("Режим полета");**

**if (Math.abs(ball.s\_0)>40)**

**text\_5.setText("Снизьте скорость");**

**if (ball.v\_tank<0.5)**

**text\_5.setText("Мало пропана");**

**if (Math.abs(ball.s\_0)>40 && ball.v\_tank<0.5)**

**text\_5.setText("Снизьте скорость. Мало пропана");**

**if (ball.h<2000 && Math.abs(ball.s\_0)>40)**

**{**

**Img = getImage(getDocumentBase(), "bum\_atom.gif");**

**helth=0;**

**bbb=1;**

**}**

**if (bbb==1)**

**repaint();**

**else**

**repaint(old\_x,old\_y, 300+dd\_x,373+dd\_y);**

**text\_1.setText("Высота:"+String.valueOf((float)ball.h)+" м");**

**text\_2.setText("Пропан:"+String.valueOf((float)ball.v\_tank)+" м3");**

**text\_3.setText("Скорость:"+String.valueOf((float)ball.s\_0)+" м/с");**

**text\_4.setText("Температура:"+String.valueOf((float)ball.t\_0)+" К");**

**}**

**public void run()**

**{**

**while(!pleaseStop)**

**{**

**animate\_ball();**

**try**

**{**

**Thread.sleep(10);**

**}**

**catch(InterruptedException e) {}**

**}**

**}**

**public void start()**

**{**

**animator\_ball = new Thread(this);**

**pleaseStop = false;**

**animator\_ball.start();**

**}**

**public void stop()**

**{**

**pleaseStop = true;**

**}**

**}**

}  
}

**balloon\_phisical.java:**

**/\*\***

**\* Автор: Скворцов П.Д.**

**\*/**

**public class balloon\_phisical**

**{**

**// константы**

**protected double r\_air\_out; // плотность воздуха снаружи шара**

**protected double r\_gas; // плотность пропана**

**protected double v\_ball;// объем воздушного шара**

**protected double q\_gas;// удельная теплота сгарания пропана**

**protected double c\_air; // удельная теплоемкость воздуха**

**protected double v\_part; // объем пропана вышедшего за одно нажатие**

**protected double g; // 9.8 Н/м^2**

**protected double m\_ball;// масса карзины**

**protected double q\_cool;// величина на которую остывает воздух в шаре за ед.времени**

**// изменяемые величины**

**protected double r\_air\_in; // плотность воздуха внутри шара**

**protected double t\_0;// начальная температура**

**protected double s\_0; // скорость шара начальная**

**protected double v\_tank;// объем пропана в болоне**

**protected double h;// высота шара над землей**

**protected double k\_air;// коэффициент сопротивления воздуха**

**public balloon\_phisical()**

**{**

**// константы**

**r\_air\_out=1.293; // плотность воздуха снаружи шара**

**r\_gas=550; // плотность пропана**

**v\_ball=800;// объем воздушного шара**

**q\_gas=115000000;// удельная теплота сгарания пропана**

**c\_air=1010; // удельная теплоемкость воздуха**

**v\_part=0.005; // объем пропана вышедшего за одно нажатие**

**g=9.8; // 9.8 Н/м^2**

**m\_ball=90;// масса карзины**

**q\_cool=150000;**

**// изменяемые нулевые величины**

**r\_air\_in=1.293; // плотность воздуха внутри шара**

**t\_0=273;// начальная температура**

**s\_0=0; // скорость шара начальная**

**v\_tank=0.5;// объем пропана в болоне**

**h=0;// высота шара над землей**

**k\_air=4;**

**}**

**public double event\_fire()**

**{**

**double d\_t,a,chislit, znamenat;**

**double t\_stage=5;**

**if (v\_tank<=0)**

**{**

**v\_tank=0;**

**event\_without\_fire();**

**}**

**else**

**{**

**chislit=q\_gas\*v\_part-q\_cool;**

**znamenat=r\_air\_in\*c\_air\*v\_ball;**

**d\_t=((q\_gas\*v\_part-q\_cool)/(r\_air\_in\*c\_air\*v\_ball));**

**r\_air\_in=(r\_air\_in\*t\_0/(t\_0+d\_t));**

**a=-g+(r\_air\_out\*g\*v\_ball)/(m\_ball+r\_air\_in\*v\_ball);**

**System.out.println("t\_0="+t\_0+" h="+h);**

**System.out.println("d\_t="+d\_t+" chislit="+chislit+" znamenet="+znamenat+" a="+a);**

**System.out.println("r\_in="+r\_air\_in+" r\_out="+r\_air\_out);**

**a=a/k\_air;**

**if (a>2)**

**a=2;**

**h=h+s\_0\*t\_stage+a\*t\_stage\*t\_stage/2;**

**s\_0=s\_0+a\*t\_stage;**

**if (s\_0>80)**

**s\_0=80;**

**t\_0=t\_0+d\_t;**

**v\_tank=v\_tank-v\_part;**

**if (h<=0)**

**{**

**s\_0=0;**

**h=0;**

**}**

**}**

**return h;**

**}**

**public double event\_without\_fire()**

**{**

**double d\_t,a,chislit, znamenat;**

**double t\_stage=5;**

**chislit=-q\_cool;**

**znamenat=r\_air\_in\*c\_air\*v\_ball;**

**d\_t=((-q\_cool)/(r\_air\_in\*c\_air\*v\_ball));**

**r\_air\_in=(r\_air\_in\*t\_0/(t\_0+d\_t));**

**a=-g+(r\_air\_out\*g\*v\_ball)/(m\_ball+r\_air\_in\*v\_ball);**

**System.out.println("t\_0="+t\_0+" h="+h);**

**System.out.println("d\_t="+d\_t+" chislit="+chislit+" znamenet="+znamenat+" a="+a);**

**a=a/k\_air;**

**h=h+s\_0\*t\_stage+a\*t\_stage\*t\_stage/2;**

**s\_0=s\_0+a\*t\_stage;**

**if (s\_0>80)**

**s\_0=80;**

**t\_0=t\_0+d\_t;**

**if (t\_0<273)**

**t\_0=273;**

**if (h<=0)**

**{**

**s\_0=0;**

**h=0;**

**}**

**return h;**

**}**

**}**

**main.java:**

**package classes;**

**public class main {**

**/\*\***

**\* @param args**

**\*/**

**/\*\***

**\* Автор: Скворцов П.Д.**

**\*/**

**public static void main(String[] args)**

**{**

**balloon\_phisical a=new balloon\_phisical();**

**for (int i=0;i<=10;i++)**

**{**

**System.out.println("t\_0="+a.t\_0+" h="+a.h);**

**System.out.println(a.event\_fire());**

**System.out.println(a.event\_fire());**

**System.out.println(a.event\_fire());**

**System.out.println(a.event\_fire());**

**System.out.println(a.event\_without\_fire());**

**System.out.println(a.event\_without\_fire());**

**System.out.println(a.event\_without\_fire());**

**System.out.println(a.event\_without\_fire());**

**System.out.println(a.event\_without\_fire());**

**}**

**}**

**}**

**ПРИЛОЖЕНИЕ О**

(информационное)

**Описание использования программы**

**Введение:**

Java-апплет «Моделирование работы воздушного шара».

Разработка приложения ведется в соответствии с заданием преподавателя.

**Назначение и область применения:**

Приложение используется в развлекательных целях.

**Постановка задачи на разработку приложения:**

Создать приложение «Моделирование работы воздушного шара» в среде программирования IntelliJ IDEA на языке программирования Java.

**Технические характеристики**

**Описание функционирования приложения:**

Главной задачей разработанного приложения является использование приложения в развлекательных целях.

**Описание метода организации входных данных:**

Для ввода данных предусмотрены клавиши на клавиатуре, отвечающие за управление воздушным шаром.

**Описание метода организации выходных данных:**

Для вывода данных предусмотрены поля, дающие информацию высоте, количестве пропана, скорости, температуре.

**Источники, использованные при разработке:**

Стандарты ЕСПД и конспект лекций по предмету «Основы проектирования программного обеспечения».