ЧОУ ВО «Тольяттинская академия управления»

Кафедра прикладной информатики и высшей математики

Б1.В.ДВ.03.02

ОЦЕНОЧНЫЕ МАТЕРИАЛЫ (СРЕДСТВА)

Учебная дисциплина	*Технология разработки объектно-ориентированных приложений на Java*
По направлению подготовки	*09.03.03*
	«Прикладная информатика»
Профиль (программа бакалавриата)	*«Прикладная информатика в цифровой экономике»*
Форма обучения	*Очная*

Оценочные материалы (средства) дисциплины рассмотрены (актуализированы) и утверждены на заседании кафедры прикладной информатики и высшей математики

Протокол заседания № 10 от «25» мая 2026 г.

Заведующий кафедрой Стрекалова Наталья Борисовна

2023

1. ПЕРЕЧЕНЬ КОМПЕТЕНЦИЙ И ЭТАПЫ ИХ ФОРМИРОВАНИЯ В ПРОЦЕССЕ ОБУЧЕНИЯ ПО ДИСЦИПЛИНЕ

Оценочные материалы (средства) сформированы в соответствии с ФГОС ВО - бакалавриат по направлению подготовки 09.03.03 «Прикладная информатика», утвержденный приказом Министерства образования и науки Российской Федерации от 19.09.2017 № 922 (с изменениями и дополнениями от 26.11.2020, 08.02.2021). В соответствии с матрицей компетенций основной профессиональной образовательной программы «Прикладная информатика в цифровой экономике» (уровень бакалавриата) в процессе обучения по дисциплине «Технология разработки объектно-ориентированных приложений на Java» происходит формирование закрепленных за дисциплиной компетенций обучающихся. Оценка сформированности компетенций на каждом этапе обучения происходит через оценку планируемых результатов обучения по дисциплине (знаний, умений, навыков).

Результаты освоения образовательной программы (компетенции)	Индикаторы достижения компетенций	Планируемые результаты обучения по дисциплине	Этапы формирования компетенции (семестры и темы)	Оценочное средство
ПК-2 Способен проектировать информационные системы	**ПК-2.1**. Моделирует прикладные процессы предметной области с учетом внедрения сквозных цифровых технологий и передовых ИТ-решений прикладной сферы	***знать:*** - особенности разработки объектно-ориентированных программ на языке Java, их возможности и достоинства; - основные приемы объектно-ориентированного решения задач на языке Java; ***уметь:*** - проектировать семейства классов в заданной прикладной области и применять их для решения профессиональных задач на языке Java; ***владеть:*** навыками моделирования и описания моделей семейств классов;	**4 семестр** Тема 1. Синтаксис языка Java, классы в языке Java. Тема 2. Наследование и инкапсуляция в языке Java. Тема 3. Разработка классов в языке Java. Тема 4. Наследование и интерфейсы в языке Java. Тема 5. Обобщённые типы и коллекции значений в языке Java.	**4 семестр** - устный опрос по темам 1, 2 - практические работы по темам 3, 4, 5 - ответ на теоретический вопрос экзамена - решение практической задачи на экзамене

	ПК-2.2. Разрабатывает модель информационной системы, в том числе с опорой на современные облачные решения	**Знать:** - основные приемы создания компонентов информационных систем для обеспечения и анализа прикладных процессов **Уметь:** - проектировать компоненты информационных систем для обеспечения и анализа прикладных процессов на языке Java **Владеть:** - навыками применения стандартных библиотек классов при разработке компонентов информационных систем для обеспечения и анализа прикладных процессов на языке Java.	**4 семестр** Тема 3. Разработка классов в языке Java. Тема 4. Наследование и интерфейсы в языке Java. Тема 5. Обобщённые типы и коллекции значений в языке Java.	**4 семестр** - практические работы по темам 3,4,5 - ответ на теоретический вопрос экзамена - решение практической задачи на экзамене
ПК-3 Способен разрабатывать информационные системы	**ПК-3.1**. Осуществляет кодирование на современных языках программирования, в том числе используя возможности специализированных цифровых платформ для индивидуальной и совместной разработки информационных систем	**Знать:** - знает синтаксис, типы данных и операторы языка Java, правила разработки программ и подпрограмм, описания классов, обращения к его методам и данным; **Уметь:** - разрабатывать объектно-ориентированные программы и программные прототипы на языке Java для решения прикладных задач; **Владеть:** - навыками написания программного кода в объектно-ориентированной технологии;	**4 семестр** Тема 1. Синтаксис языка Java, классы в языке Java. Тема 2. Наследование и инкапсуляция в языке Java. Тема 3. Разработка классов в языке Java. Тема 4. Наследование и интерфейсы в языке Java. Тема 5. Обобщённые типы и коллекции значений в языке Java. Тема 6. Работа со строками в языке	**4 семестр** - устный опрос по темам 1, 2 - практические работы по темам 3, 4, 5, 6 - ответ на теоретический вопрос экзамена - решение практической задачи на экзамене

			Java.	
	ПК-3.2. Проводит тестирование компонентов информационных систем, в том числе используя возможности специализированных цифровых платформ	**Знать:** - основные приемы тестирование компонентов информационных систем на языке Java; **Уметь:** - составлять блоки тестов компонентов информационных систем на языке Java **Владеть:** - навыками применения тестов для информационных систем на языке Java;	**4 семестр** Тема 3. Разработка классов в языке Java. Тема 4. Наследование и интерфейсы в языке Java. Тема 5. Обобщённые типы и коллекции значений в языке Java.	**4 семестр** - практические работы по темам 3,4,5 - ответ на теоретический вопрос экзамена - решение практической задачи на экзамене

2. ОЦЕНОЧНЫЕ МАТЕРИАЛЫ (СРЕДСТВА) ДЛЯ ТЕКУЩЕГО КОНТРОЛЯ И ОЦЕНКИ ЗНАНИЙ, УМЕНИЙ, НАВЫКОВ, ОБЕСПЕЧИВАЮЩИХ ФОРМИРОВАНИЯ КОМПЕТЕНЦИЙ В ПРОЦЕССЕ ОБУЧЕНИЯ ПО ДИСЦИПЛИНЕ

Устные опросы

Вопросы для проведения устного опроса
по теме «Синтаксис языка Java, классы в языке Java.»

1. Особенности языка и платформы Java.
2. Классификация программ по типу исполнения (компилируемые, интерпретируемые, исполняемые на виртуальных машинах). Виртуальная машина Java. JIT-компиляция.
3. Создание простейшей программы на Java, её компиляция в байт-код и запуск.
4. Средства разработки Java-приложений. Интегрированные среды разработки.
5. Встроенные типы данных. Способы задания литералов различных типов.
6. Хранение данных в памяти ЭВМ.
7. Приведение типов (явное и автоматическое). Константы и переменные.
8. Типы Java-приложений, их особенности.
9. Типы данных в языке Java: простые и ссылочные типы.
10. Массивы в Java: массивы простых типов и массивы объектов.
11. Оператор присваивания. Порядок действий (приоритет операторов).
12. Арифметические операторы. Операторы инкремента и декремента.
13. Встроенный класс Math. Псевдослучайные числа.
14. Операторы сравнения и логические операторы.
15. Операторы ветвления. Условный оператор. Минимизация количества проверок.
16. Операторы ветвления. Оператор множественного выбора. Его сравнение с условным оператором.
17. Встроенный класс String. Строковые операции.
18. Стандартные потоки ввода-вывода. Организация ввода и вывода данных. Класс Scanner.

Вопросы для *проведения устного опроса*

по теме «Наследование и инкапсуляция в языке Java.»

1. Классы в языке Java: особенности реализации, определение класса.
2. Классы в языке Java: управление доступом к элементам класса; понятие пакета.
3. Классы в языке Java: поля класса.
4. Классы в языке Java: методы, конструкторы.
5. Интерфейсы в языке Java: определение интерфейса, реализация интерфейса.
6. Понятие Java API. Основные пакеты Java API.
7. События в Java: понятие события; типы событий; иерархия классов событий.
8. События в Java: модель делегирования событий.
9. События в Java: интерфейсы блоков прослушивания событий; способы реализации блока прослушивания.
10. События в Java: использование внутренних классов, классов-адаптеров для упрощения обработки событий.
11. Исключения: понятие исключения; классы исключений; необходимость обработки исключений.
12. Исключения: операторы языка Java, используемые для обработки исключений.
13. Исключения: организация обработки исключений; определение собственных исключений.
14. Потоки вычислений: понятия процесса, потока.
15. Ввод/вывод в Java: основные понятия.
16. Ввод/вывод в Java: основные группы классов и интерфейсов пакета java.io.

Критерии оценивая устного опроса:

- оценка «*отлично*» выставляется студенту, если он ответил развернуто на один из заданных вопросов, активно дополнял ответы других студентов или задавал им дополнительные вопросы;
- оценка «*хорошо*» выставляется студенту, если он ответил развернуто на один из заданных вопросов или ответил кратко на ряд вопросов;
- оценка «*удовлетворительно*» выставляется студенту, если он кратко ответил на один из задаваемых вопросов или просто дополнял ответы других студентов, задавал им дополнительные вопросы;
- оценка «*неудовлетворительно*» выставляется студенту, если он не смог ответить правильно на заданный вопрос, либо не отвечал на вопросы и не участвовал в их обсуждении.

Практические работы

Практическое задание по теме «Разработка классов в языке Java.»

Практическая работа №1

ОБЪЕКТНО-ОРИЕНТИРОВАННАЯ КОНЦЕПЦИЯ. ИГРА ЖИЗНИ

Цель работы: изучить предметную область, в которой предстоит работать всю остальную часть лабораторного практикума. Приобретение навыков моделирования Игры жизни.

Методические указания.

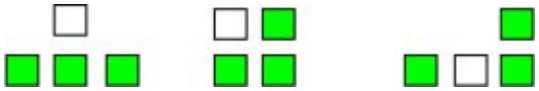
Игра жизни - одна из наиболее известных 2-мерных моделей клеточной структуры. Это не настоящая игра, так как в ней нет игроков, нет победителей и побежденных. Это игра, в которой начальная позиция и правила определяют все, что случится после этого. Больше информации об игре и ее обоснование см. по ссылкам

http://www.math.com/students/wonders/life/life.html и *http://mathworld.wolfram.com/CellularAutomaton.html*.

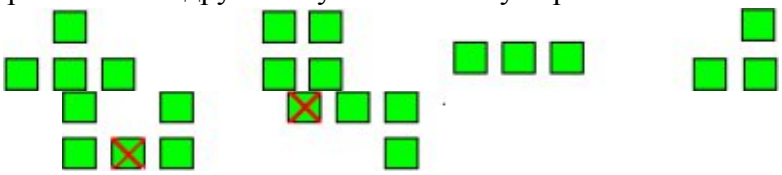
Правила Игры жизни: Игра жизни происходит на сетке из квадратных клеток - как шахматная доска, - расширяясь произвольным образом по всем
направлениям. Клетка может быть живой или мертвой.

Отметка на клетке показывает живую клетку. Пустая клетка является мертвой. Каждая клетка в сетке имеет восемь смежных клеток по всем направлениям, включая диагональные. Чтобы выполнить один шаг игры, подсчитайте число живых соседей для каждой клетки. Что случится дальше -
зависит от этого числа:

1. Правило рождения: Мертвая клетка, имеющая ровно три живых соседа становится живой клеткой



2. Правило выживания: Живая клетка с двумя или тремя живыми соседями остается жить.
3. Правило смерти: Во всех других случаях клетка умирает или остается мертвой.



Число живых соседей всегда вычисляется до применения правила.

Ход работы.

1. ***Определение объектов в игре***

Определим объекты, вовлеченные в игру (объекты обычно определяются поиском в требованиях имен существительных).
Какие объекты вы можете определить в игре?
Вот несколько вопросов, которые могут помочь определить объекты:

- Как представляется сама игра? - Итерационное выполнение процессов
- В чем состоит игра? – Выживание или отброс лишних клеток
- Что влияет на развитие игры во времени? - Выполнение условий выживания или смерти.

Поскольку на вопросы, приведенные выше, может быть несколько подходящих ответов, возможно, правильным будет сказать, что наиболее очевидными объектами являются игра, доска и правило, и эти объекты составляют игру.

2. ***Определение основных классов***

На основании результатов предыдущего шага, какие классы составляют систему игры? Каковы имена классов, их данные и поведение?
Основными классами игры могут быть Game, Board и Rule. Игра имеет доску и ряд правил, а также индекс "поколение" для определения того,
сколько поколений досок отслеживается. Доска имеет клетки, и она инициализируется при создании некоторой конфигурацией живых и мертвых клеток. Правило может иметь имя и тип и вычисляется для каждой клетки на доске. Типом правила может быть: рождение, выживание или смерть. Когда оно вычисляется, проверяется тип правила, и на базе типа выполняются различные вычисления.

Вывод: в этой работе мы проанализировали предметную область игры, которую будем реализовывать в остальных работах.

Практическая работа №2

ПОСТРОЕНИЕ КЛАССОВ JAVA

Цель работы: знакомство с Java Development Tools в Eclipse. приобретение навыков создания классов при помощи Eclipse, а также с использованием инструментов рефакторинга Java в Eclipse.

Ход работы:

1. *Создание проекта*

Создадим новый проект Java с именем LifeGame. Для создания проекта выберем из выпадающего меню: File -> New -> Project -> Java -> Java Project и щелкнем на Next. Задаём имя проекта и щелкаем на Finish.

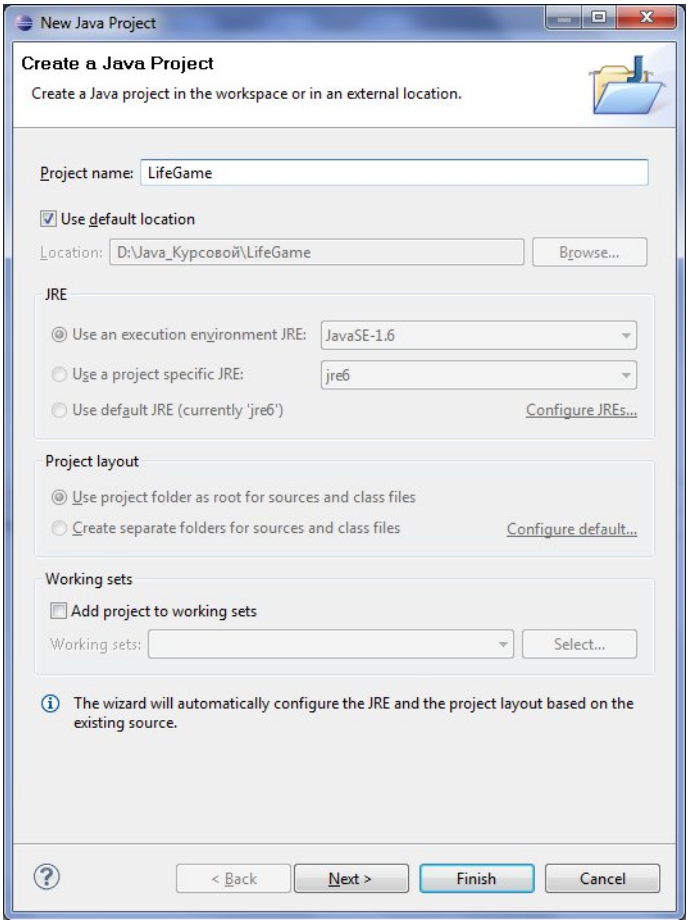


Рис.1 Создание нового проекта

2. Создание пакета

Создадим пакет, который будет содержать прикладные классы для игры. Пакеты группируют классы по функциональности и по их пространствам имен. Обычно классы, которые представляют модель, находятся в одном пакете, классы, которые используются для тестирования, находятся в другом пакете и т.д. Для создания нового пакета выберем проект и выберем New -> Package из контекстного меню. В качестве имени пакета - org.eclipse.lifegame.domain и щелкаем на Finish.

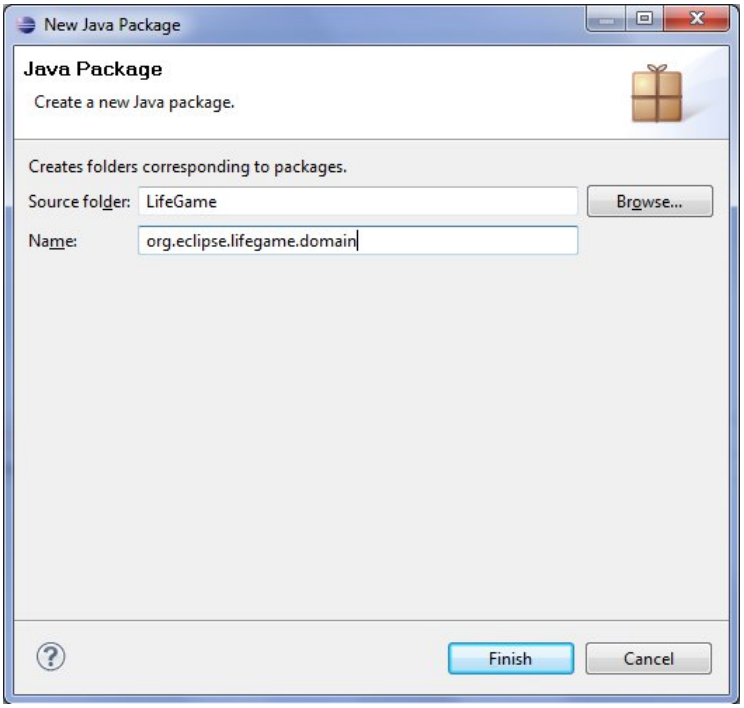


Рис. 2. Создание нового пакета.

*3.*Создание классов Game и Board

Создадим класс Board, который будет иметь закрытое поле cells, являющееся 2-мерным массивом целых чисел. Поле cells должно быть 10x10 и содержать после запуска игры нули или единицы.

Сгенерим методы-акцессоры для поля. Добавим конструктор класса Board, который будет принимать в качестве параметра 2-мерный массив целых чисел. Он будет представлять шаблон (начальное состояние клеток) для игры, когда она запустится. Также заместим конструктор по умолчанию, чтобы просто инициализировать клетки в 2-мерном (10x10) массиве при создании.

Определим в классе два метода разворачивания: evolve(Game) и evolve(Game,int). Эти методы должны иметь возвращаемый тип void и должны проходить через клетки и на основании правил развивать следующую стадию. Метод развития, который принимает целый параметр, развивает доску через много стадий (в зависимости от параметра). Оставим эти методы пока пустыми.

Заместим метод toString() класса Board, чтобы он просто возвращал строку "Board for game of life" . В следующих работах вы добавите больше поведения в этот метод.

Создадим класс Game, который имеет поле board типа Board и поле generation типа int (показывающее, сколько поколений доска развивалась).

Сгенерим методы-акцессоры для полей. Добавим конструктор класса Game, который будет принимать в качестве параметра доску и инициализировать поле board значением параметра, а поколение - числом 1.

*4.*Создание экземпляров Game и Board

Используем Scrapbook из предыдущей работы для создания экземпляра класса Board. Классы не будут видны в Scrapbook. Сначала добавим проект LifeGame, как зависимость, к проекту, содержащему Scrapbook.

Выберем проект ExperimentLab и Properties из контекстного меню. Щелкнем на Java Build Path и выберем закладку Projects. Выберем проект LifeGame и нажмем на OK.

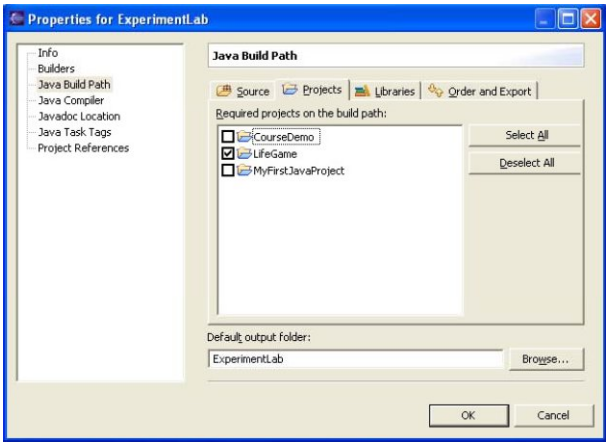


Рис. 3. Добавление проекта в Java Build Path.

В Scrapbook выберем Set Imports из контекстного меню и выберем кнопку Add Packages. Выберем пакет, который содержит классы, и щелкнем на OK дважды.

Рис. 5. Импорт пакета.

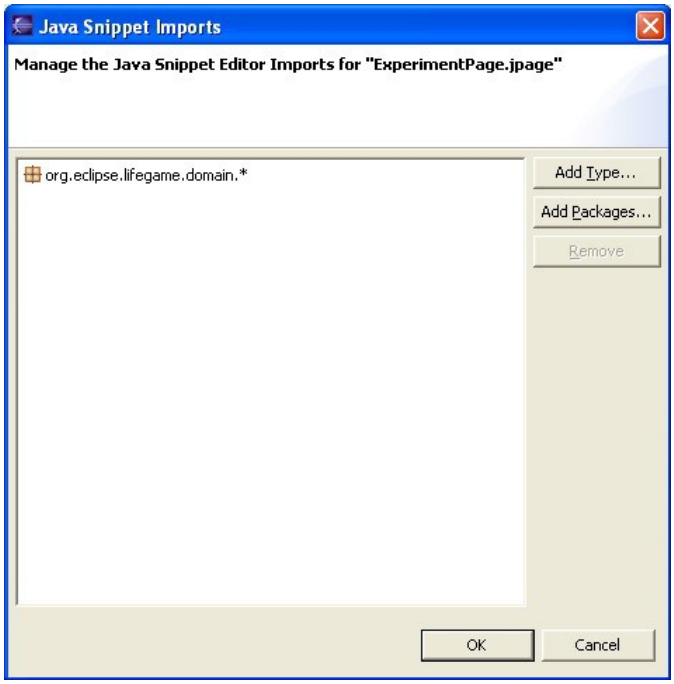


Рис. 6. Результат импорта пакета в Scrapbook.

Создадим экземпляр класса Game, передавая ему созданный объект доски.
Проинспектируем объект игры.
В Scrapbook получим код:

```
int [][] boardCells = {
{0, 0, 0, 0, 0, 0, 0 , 0, 0, 1},
{0, 0, 0, 0, 0, 0, 0 , 0, 0, 1},
{0, 0, 0, 0, 0, 0, 0 , 0, 0, 1},
{0, 0, 0, 0, 0, 0, 0 , 0, 0, 1},
{0, 0, 0, 0, 0, 0, 0 , 0, 0, 1},
{0, 0, 0, 0, 0, 0, 0 , 0, 0, 1},
{0, 0, 0, 0, 0, 0, 0 , 0, 0, 1},
{0, 0, 0, 0, 0, 0, 0 , 0, 0, 1},
{0, 0, 0, 0, 0, 0, 0 , 0, 0, 1},
{0, 0, 0, 0, 0, 0, 0 , 0, 0, 1}};
Board board = new Board(boardCells); Game game = new Game(board); game
```

Выводы: в этой работе создали базовые классы для Игры жизни, и познакомились с написанием конструкторов, методов и определением полей.

Критерии оценивая:

- оценка «*отлично*» выставляется студенту, если задание выполнено в полном объеме, студент хорошо ориентируется в технологии разработки функций пользователя, программа логично и корректно демонстрирует работу всех функций;
- оценка «*хорошо*» выставляется студенту, если он выполнил задание почти в полном объеме (может отсутствовать пункт), допускает небольшие «проектно-технологические» ошибки в разработке функций;
- оценка «*удовлетворительно*» выставляется студенту, если он выполнил первые 3 пунктов задания;
- оценка «*неудовлетворительно*» выставляется студенту, если он выполнил менее 3 пунктов задания.

Практическое задание по теме «Обобщённые типы и коллекции значений в языке Java.»

Практическая работа №3

ОПЕРАТОРЫ УПРАВЛЕНИЯ

Цель работы: научиться использовать операторы управления для реализации некоторого поведения классов игры. В частности, реализовывать шаблон Singleton для класса Game и метод toString() класса Board.

Выполнение работы

Шаг 1: Реализация шаблона Singleton для класса Game

Необходим только один экземпляр класса Game в одном приложении. Этот экземпляр будет представлять саму игру. Если в одной JVM может быть создано не более одного экземпляра класса, это называется шаблоном Singleton (одиночка). На этом шаге мы реализуем этот шаблон для класса Game.

Определим закрытое статическое поле с именем theInstance в классе Game. Это поле будет хранить экземпляр класса, так что тип поля должен быть Game. Добавим закрытый статический метод getInstance() в класс Game, который будет возвращать экземпляр класса, сохраненный в поле theInstance. В методе проверим, прежде всего, не имеет ли поле значение null, если да, то создадим новый экземпляр класса Game и присвоим его этому полю. Такая реализация называется ленивой инициализацией.

Изменим конструктор класса Game на закрытый, чтобы только сам класс мог его использовать. Это не даст возможности кому-либо еще создавать экземпляры класса.

Используем созданный сейчас шаблон в Scrapbook для инспектирования кода. Проинспектируем выражение Game.getInstance();

Шаг 2: метод toString() класса Board

В предыдущих работах мы определили метод toString() класса Board, как возвращающий простую String. Метод toString() является текстовым представлением объекта. Когда объект печатается на стандартную консоль Java, используется этот метод. Доска содержит клетки, которые являются нулями или единицами. Когда доска печатается на консоли, для клеток, содержащих нули, должны печататься пробелы, а для клеток, содержащих единицы должны печататься символы X.

Реализуем метод toString(), чтобы он возвращал строку на основании этой спецификации. Это позволит нам увидеть на Console, как выглядит наша доска. Чтобы реализовать правильное поведение, нам понадобится перебрать массив массивов и проверить каждую ячейку. Используем класс StringBuffer для добавления символов в конец строки.

Используем Scrapbook, чтобы проверить метод toString() путем создания и распечатки экземпляра класса на Console. Код Scrapbook будет выглядеть следующим образом:

```
int [][] boardCells = {
{0, 0, 0, 0, 0, 0, 0 , 0, 0, 1},
{0, 0, 0, 0, 0, 0, 0 , 0, 0, 1},
{0, 0, 0, 0, 0, 0, 0 , 0, 0, 1},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 1},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 1},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 1},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 1},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 1},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 1},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 1}};
```

Board board = new Board(boardCells); System.out.println(board); Этот код будет выведен на Console так:

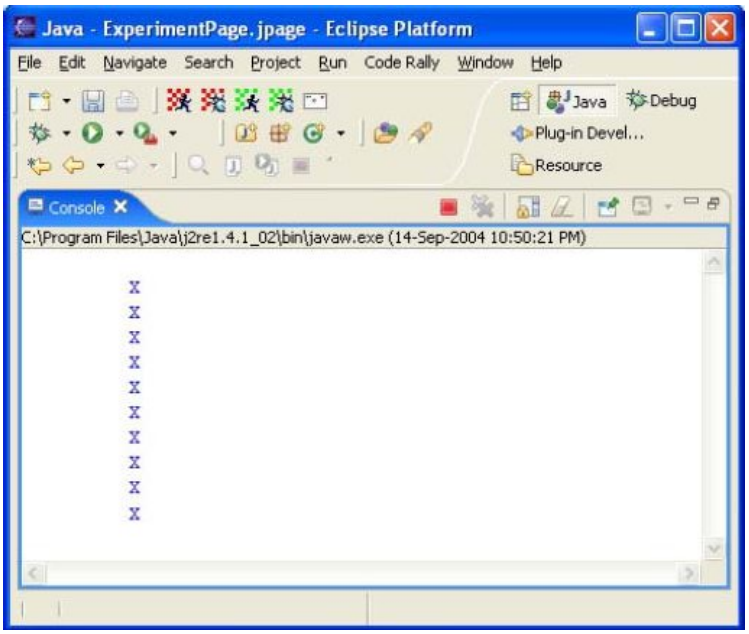


Рис. 1. Вывод в консоль метода toString() класса Board.

Выводы: В этой работе мы научились использовать операторы управления для реализации шаблона Singleton для класса Game и метода toString() класса Board.

Критерии оценивая:

- оценка «*отлично*» выставляется студенту, если задание выполнены в полном объеме, программа работают без ошибок, студент хорошо ориентируется в технологии разработки структуры и работы с файлами;
- оценка «*хорошо*» выставляется студенту, если он выполнил задание почти в полном объеме (может отсутствовать пункты), программы работают без ошибок;
- оценка «*удовлетворительно*» выставляется студенту, если он выполнил первые 3 заданий, соблюдая технологию работы;
- оценка «*неудовлетворительно*» выставляется студенту, если он выполнил менее 3 заданий.

Практическое задание по теме «Наследование и интерфейсы в языке Java.»

Практическая работа №4

НАСЛЕДОВАНИЕ

Цель работы: научиться использовать наследование для улучшения проекта, повышения гибкости и возможностей сопровождения системы.

Выполнение работы:

Наследование помогает при рефакторинге кода, а также при правильном представлении объектов (т.е. оно делает объект более близким к реальной

жизни). Довольно часто некоторые общие свойства похожих классов абстрагируются в абстрактном суперклассе и повторно используются в подклассах. Абстрактный класс - это класс, который не может иметь экземпляров. Интерфейсы используются в языке Java для большего абстрагирования, поскольку они определяют только объектный протокол.

Интерфейсы также используются для кросс-иерархического полиморфизма (перекрестного наследования). В этой работе мы будем использовать и абстрактные классы, и интерфейсы. Мы сосредоточимся на 2- мерном моделировании клеточной структуры в Игре жизни. Конкретно в этой работе мы будем делать систему гибкой для обработки различных версий игр моделирования клеток.

Шаг 1: Перепроектирование класса Board

Класс Board, который мы определили в предыдущих работах, содержит 2-мерную клеточную структуру, это означает, что он в реальности является 2-мерной доской. Всякий раз, когда мы захотите ввести новую игру (задачу), нам понадобиться менять класс. Различие между досками состоит в их клетках, поскольку 1-мерная доска является 1-мерным массивом, 2-мерная доска является 2-мерным массивом и т.д. Также различается реализация протокола evolve(), поскольку на разных досках игра развивается по-разному. Изменим имя класс Board на TwoDimensionalBoard, поскольку это то,

что на самом деле реализуется в классе Board. Чтобы изменить имя класса, выберем класс и выберем опцию Refactor ->Rename из контекстного меню. Eclipse откроет диалог для изменения имени.

Создадим абстрактный класс Board с абстрактным протоколом развития

- методами evolve(Game, int) и evolve(Game). Это заставит все подклассы класса Board реализовать у себя эти методы. Изменим суперкласс класса TwoDimensionalBoard на Board (:extends Board:).

Когда мы изменяем имя класса, изменяются все ссылки, это означает, что поле board в классе Game теперь будет типа TwoDimensionalBoard. Чтобы сохранить это поле родовым, изменим его тип на Board.

Изменим его методы-акцессоры, чтобы они возвращали и принимали также тип Board.

Шаг 2: Перепроектирование класса Rule - шаблона Strategy

Класс Rule, который мы определили до этого, имеет поле name. Оно может быть индикатором типа правила. Правило должно иметь метод doesApply(), который возвращает значение boolean, зависящее от того, применимо ли правило для текущей клетки или нет. Протокол должен сначала проверять имя правила, а потом на его основании производить различные вычисления. При таком подходе возможны некоторые потенциальные проблемы в сопровождении при введении нового правила, поскольку метод должен будет измениться. Лучшим подходом является наличие специфического класса Rule для каждого правила в игре. Каждый класс должен будет реализовывать собственный протокол doesApply(). Если мы сделаем так, то при введении нового правила в игру, будет создаваться новый специфический класс со своей реализацией метода doesApply(). Мы можем абстрагировать поведение правил в абстрактном классе Rule.

Изменим определение класса Rule на abstract. Создадим абстрактные классы DeathRule, SurvivalRule и BirthRule как подклассы класса Rule, создадим TwoDimensionalDeathRule, TwoDimensionalSurvivalRule и TwoDimensionalBirthRule - подклассы только что созданных абстрактных классов.

Метод doesApply() весьма специфичен для разных размерностей игры. Другими словами, метод 2-мерного правила принимает в качестве параметра 2-мерный массив клеток. Как мы можем заставить все 2-мерные правила иметь одинаковый протокол и в то же время наследовать разным суперклассам? Ответом является перекрестное наследование и интерфейсы.

Создадим новый интерфейс TwoDimensionalRule и определим методы:

public int getCellValue();
public boolean doesApply(**int**[][] cells, **int** i, **int** j);

Реализуем метод getCellValue() во всех 2-мерных классах. Метод должен просто возвращать значение, которое должно быть 1 для правил рождения и выживания и 0 для правила смерти. Изменим все классы TwoDimensionalXYZRule, чтобы реализовать созданный интерфейс. Добавим поле rules в класс Game. Поле будет массивом правил, и в нем должно быть точно 3 правила, записанные в массив. Шаблон, который мы сейчас реализовали и который показан на картинке выше, известен как шаблон Strategy.

Шаг 3: Импорт классов Rule

Поскольку в классах правил много кодирования, просто импортируем все классы правил (3. Эти классы реализуют методы на основе простых правил, заданных в требованиях для этой игры.

Шаг 4: Перепроектирование класса Game

Сделаем ранее определенный класс Game абстрактным. Eclipse выдает ошибку, поскольку не могут создаваться экземпляры этого класса, а мы создали экземпляр в реализации "одиночки". Создадим класс TwoDimensionalGame, наследующий классу Game. Конструктор по умолчанию в классе Game инициализирует правила, и он выглядит следующим образом:

```
public Game(){
this.rules = new Rule[3];
}
```

Инициализируем массив rules в конструкторе класса Game, чтобы создать экземпляры соответствующихклассов 2-мерных правил, определенных на предыдущем шаге.
Конструктор выглядит следующим образом:

```
private TwoDimensionalGame(){ setBoard(new TwoDimensionalBoard());
getRules()[0] = new TwoDimensionalBirthRule(); getRules()[1] = new
TwoDimensionalSurvivalRule(); getRules()[2] = new TwoDimensionalDeathRule();
}
```

Шаг 5: Реализация протокола Board

Реализуем методы развития в классе TwoDimensionalBoard. Метод evolve(Game, int) просто выполняет цикл до переданного ему индекса (int), в каждой итерации вызывает метод evolve(Game) и печатает себя на стандартной консоли. Код выглядит таким образом:

```
public void evolve(Game aGame, int index){ if (index == 0) return;
for (int i = 1; i < = index; i++){ evolve(aGame); System.out.println(this);
}
}
```

Метод evolve(Game) перебирает клетки доски и для каждой клетки проверяет, применимо ли к ней какое-либо правило игры. Если правило применяется, то текущее значение клетки устанавливается в значение правила. Если никакое правило к клетке не применимо, то просто выводится на консоль сообщение о том, что к текущей клетке не применимо никакое правило к клетке Код выглядит таким образом:

```
public void evolve(Game aGame){ boolean doesAnyApply = false; TwoDimensionalRule rule =
null; for (int i = 0; i < cells.length; i++){
for (int j = 0; j < cells[0].length; j++){ doesAnyApply = false;
for (int k = 0; k < aGame.getRules().length; k++){ rule =
(TwoDimensionalRule)aGame.getRules()[k]; if (rule.doesApply(getCells(), i, j)) {
getCells()[i][j] = rule.getCellValue(); doesAnyApply = true;
break;
}
}
if (!(doesAnyApply))
System.out.println("No rules apply for cell[" + i + "][" + j + "]");
}
}
}
```

Шаг 6: Реализация метода run в классе Game

Добавим метод run() с возвращаемым значением типа void в класс Game. Метод просто вызывает метод развития игровой доски и передает ему поколение игры и саму игру в качестве параметров.

Метод выглядит так:

```
public void run(){ getBoard().evolve(this, getGeneration());
}
```

Шаг 7: Тестирование игры

Изменим класс LifeGameTester, чтобы создавать новый экземпляр TwoDimensionalBoard, присваивать некоторые инициализированные boardCells доске, а затем получать экземпляр TwoDimensionalGame, назначать ему ранее созданный board, назначать экземпляру 10 как generation и, наконец, выполнять run для экземпляра.

Метод main выглядит так:

```
public static void main(String[] args) {
int[][] boardCells = {
{0, 0, 0, 1, 0, 0, 0 , 0, 0, 0},
{0, 0, 1, 0, 1, 0, 0 , 0, 0, 0},
{0, 0, 0, 1, 0, 0, 0 , 0, 0, 0},
{0, 0, 0, 1, 0, 0, 0 , 0, 0, 0},
{0, 0, 1, 0, 1, 0, 0 , 0, 0, 0},
{0, 0, 0, 1, 0, 0, 0 , 0, 0, 0},
{0, 0, 0, 1, 0, 0, 0 , 0, 0, 0},
{0, 0, 1, 0, 1, 0, 0 , 0, 0, 0},
{0, 0, 1, 0, 1, 0, 0 , 0, 0, 0},
{0, 0, 0, 1, 0, 0, 0 , 0, 0, 0}};
```

```
TwoDimensionalBoard board = new
TwoDimensionalBoard(boardCells);
TwoDimensionalGame game = TwoDimensionalGame.getInstance(); game.setBoard(board);
game.setGeneration(10); game.run();
}
```

Запустим класс тестера. Пронаблюдаем поведение при многократном запуске с разными значениями клеток. После рефакторинга и перепроектирования классов мы можем иметь диаграмму классов, как на картинке ниже (Рис. 1).

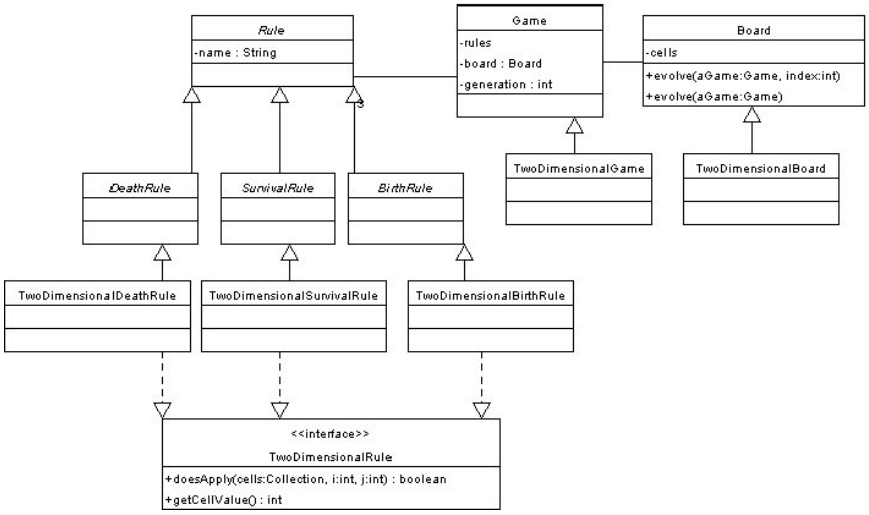


Рис. 1 Диаграмма классов

Вывод: мы научились использовать наследование, абстрактные классы, интерфейсы и инструменты рефакторинга Eclipse.

Критерии оценивая:

- оценка «*отлично*» выставляется студенту, если задание выполнено в полном объеме, студент хорошо ориентируется в технологии разработки, программа логично и корректно демонстрирует работу всех функций;
- оценка «*хорошо*» выставляется студенту, если он выполнил задание почти в полном объеме (может отсутствовать пункт), допускает небольшие «проектно-технологические» ошибки в разработке функций;
- оценка «*удовлетворительно*» выставляется студенту, если он выполнил первые 3 пунктов задания;
- оценка «*неудовлетворительно*» выставляется студенту, если он выполнил менее 3пунктов задания.

Практическое задание по теме «Работа со строками в языке Java.»

Практическая работа №5

КОЛЛЕКЦИИ

Цель работы: научиться использовать коллекции для реализации истории развития доски в ходе игры.

Выполнение работы:

Когда доска развивается, ячейки меняют свои значения, и доска переходит в совсем другое состояние. В конце игры доска имеет состояние нового поколения, а все предыдущие состояния потеряны. Мы реализуем историю развития доски в класса Game.

Шаг 1: Добавление поля boardHistory в класс Game Определите новое поле boardHistory типа ArrayList в классе Game. Сгенерируйте для поля методы-акцессоры. В конструторе Game инициализируйте поле новым экземпляром ArrayList.

Шаг 2: Добавление протокола манипулирования историей

Определим новый метод addToHistory(Board) в классе Game. Метод должен принимать в качестве параметра экземпляр класса Board и просто добавлять его к коллекции boardHistory игры. Определим другой метод в классе Game, clearHistory(), который будет удалять все элементы из коллекции boardHistory.

Шаг 3: Использование протокола манипулирования историей

Изменим метод run() в классе Game, чтобы он очищал историю доски перед запросом к доске на развитие. После того, как доска развилась в свое новое состояние, она добавляется к истории доски прежде, чем она разовьется опять.

Модифицируем метод evolve(Game, int) класса Board, чтобы добавлять доску к истории игры после каждого развития. Мы можем использовать выражение aGame.addToHistory(this); в цикле for.

Шаг 4: Реализация распечатки истории

В предыдущей работе мы добавили распечатку доски на стандартной консоли всякий раз, когда доска развивается, в метод evolve(Game,int) . Это было сделано в целях тестирования, чтобы было видно, как клетки доски изменяют значения. В действительности ответственность за распечатку доски будет у несколько иного объекта. Чтобы поддержать это, мы должны удалить оператор печати из метода evolve(Game,int) . После этого добавим в класс Game метод printHistory(), который будет перебирать коллекцию истории и в каждой итерации печатать доску на консоли. Каждый объект в коллекции приведем к его типу. BoardHistory будет содержать объекты типа Board, так что в каждой итерации по истории нам понадобится приводить объекты к этому типу.

Метод выглядит так:

```
public void printHistory(){
Iterator iterator = getBoardHistory().iterator(); Board board = null;
while (iterator.hasNext()){ board = (Board)iterator.next(); System.out.println(board);
}
}
```

Шаг 5: Добавления к "одиночке"

Когда мы используем шаблон "одиночки", мы используем единственный экземпляр. В динамической среде разработки, когда мы регулярно изменяем состояние и поведение объекта, это может привести к нежелательному поведению. Нам нужно очищать "одиночку" всякий раз, когда мы делаем изменения, чтобы быть уверенными, что он веден себя правильно.

Добавим открытый статический метод в класс TwoDimensionalGame,

который будет просто сбрасывать экземпляр в null:

```
public static void clearInstance(){ theInstance = null;
}
```

Шаг 6: Тестирование кода

Изменим класс тестера для использования реализованного протокола печати истории после выполнения игры. Запустим тестер снова.

Вывод: мы научились использовать коллекции Java для отслеживания и сохранения истории поведения доски в игре.

Практическая работа №6

КОЛЛЕКЦИИ

Цель работы: научиться использовать коллекции для реализации истории развития доски в ходе игры.

Выполнение работы:

Когда доска развивается, ячейки меняют свои значения, и доска переходит в совсем другое состояние. В конце игры доска имеет состояние нового поколения, а все предыдущие состояния потеряны. Мы реализуем историю развития доски в класса Game.

Шаг 1: Добавление поля boardHistory в класс Game Определите новое поле boardHistory типа ArrayList в классе Game. Сгенерируйте для поля методы-акцессоры. В конструторе Game инициализируйте поле новым экземпляром ArrayList.

Шаг 2: Добавление протокола манипулирования историей

Определим новый метод addToHistory(Board) в классе Game. Метод должен принимать в качестве параметра экземпляр класса Board и просто добавлять его к коллекции boardHistory игры. Определим другой метод в классе Game, clearHistory(), который будет удалять все элементы из коллекции boardHistory.

Шаг 3: Использование протокола манипулирования историей

Изменим метод run() в классе Game, чтобы он очищал историю доски перед запросом к доске на развитие. После того, как доска развилась в свое новое состояние, она добавляется к истории доски прежде, чем она разовьется опять.

Модифицируем метод evolve(Game, int) класса Board, чтобы добавлять доску к истории игры после каждого развития. Мы можем использовать выражение aGame.addToHistory(this); в цикле for.

Шаг 4: Реализация распечатки истории

В предыдущей работе мы добавили распечатку доски на стандартной консоли всякий раз, когда доска развивается, в метод evolve(Game,int) . Это было сделано в целях тестирования, чтобы было видно, как клетки доски изменяют значения. В действительности ответственность за распечатку доски будет у несколько иного объекта. Чтобы поддержать это, мы должны удалить оператор печати из метода evolve(Game,int) . После этого добавим в класс Game метод printHistory(), который будет перебирать коллекцию истории и в каждой итерации печатать доску на консоли. Каждый объект в коллекции приведем к его типу. BoardHistory будет содержать объекты типа Board, так что в каждой итерации по истории нам понадобится приводить объекты к этому типу.

Метод выглядит так:

```
public void printHistory(){
Iterator iterator = getBoardHistory().iterator(); Board board = null;
while (iterator.hasNext()){ board = (Board)iterator.next(); System.out.println(board);
}
}
```

Шаг 5: Добавления к "одиночке"

Когда мы используем шаблон "одиночки", мы используем единственный экземпляр. В динамической среде разработки, когда мы регулярно изменяем состояние и поведение объекта, это может привести к нежелательному поведению. Нам нужно очищать "одиночку" всякий раз, когда мы делаем изменения, чтобы быть уверенными, что он веден себя правильно.

Добавим открытый статический метод в класс TwoDimensionalGame,

который будет просто сбрасывать экземпляр в null:

```
public static void clearInstance(){ theInstance = null;
}
```

Шаг 6: Тестирование кода

Изменим класс тестера для использования реализованного протокола печати истории после выполнения игры. Запустим тестер снова.

Вывод: мы научились использовать коллекции Java для отслеживания и сохранения истории поведения доски в игре.

Практическая работа №7

ПОТОКИ

Цель работы: работа с потоками, приобретение навыков в написании данных в файл и считывании из файла, используя потоки.

Выполнение работы:

Шаг 1: Создание методов Game

Добавим два новых метода в класс Game с именами: *public void saveCurrentBoardToFile(String filename); public void loadBoardFromFile(String filename);*

Эти методы должны записывать игровую доску в заданный файл, а также читать доску из заданного файла. Другими словами, метод записи должен сериализовать игровую доску в файл, а метод чтения десериализовать объект доски из файла и устанавливать в него игровую доску.

Шаг 2: Создание абстрактных методов Board

Добавим два абстрактных метода в класс Board с именами: *public abstract void saveCurrentBoardToFile(String filename); public abstract void loadBoardFromFile(String filename);* ***Шаг 3: Проверка ваших методов***

Проверим реализацию путем вычисления следующего кода в Scrapbook: int[][] boardCells = {

{0, 0, 0, 1, 0, 0, 1 , 0, 1, 0},

{0, 0, 1, 0, 1, 0, 1 , 0, 0, 0},

{1, 0, 0, 1, 0, 0, 1 , 0, 0, 0},

{0, 0, 0, 1, 0, 0, 1 , 0, 0, 0},

{0, 0, 1, 0, 1, 0, 1 , 1 ,0, 0},

{0, 0, 0, 1, 0, 0, 1 , 1, 1, 0},

{0, 0, 0, 1, 0, 0, 1 , 1, 0, 1},

{1, 0, 1, 0, 1, 0, 1 , 0, 1, 0},

{0, 0, 1, 0, 1, 0, 1 , 0, 1, 1},

{0, 0, 0, 1, 0, 0, 1 , 0, 0, 1}};

TwoDimensionalBoard board = new TwoDimensionalBoard(boardCells);
TwoDimensionalGame.clearInstance();
TwoDimensionalGame game = TwoDimensionalGame.getInstance(); game.setBoard(board);
game.saveCurrentBoardToFile("c:/game.txt");
int[][] newBoardCells = {
{1, 1},
{1, 1}};
board = new TwoDimensionalBoard(newBoardCells); game.setBoard(board);
System.out.println(game.getBoard()); game.loadBoardFromFile("c:/game.txt");
System.out.println(game.getBoard());

В этом коде мы сначала создаем доску, а затем сохраняем ее в файле. Затем мы создаем новую, меньшую доску, присваиваем ее игре, чем, следовательно, удаляем старую игровую доску. После этого мы распечатываем новую доску, загружаем старую доску, присваиваем ее игре и распечатываете ее, так что вы визуально проверите, работают ли загрузка и запись. Всегда можем проверить, что было записано в файл, просмотрев его "вручную" при помощи текстового процессора.

При сериализации объекта в файл записываются двоичные данные, но если мы записываем другие данные, они должны быть читабельными.

Выводы: в данной работе мы научились использовать потоки Java для сохранения игровой доски в файле и загрузки ее из файла.

Практическая работа №8

ПОСТРОЕНИЕ ГРАФИЧЕСКОГО ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ ПРИ ПОМОЩИ SWT

Цель работы: знакомство с Standard Widget Toolket (SWT) в Eclipse; получение навыков в использовании одного из наиболее популярных менеджеров раскладки: FormLayout, многих элементов (widgets) SWT и построении графических пользовательских интерфейсы, используя элементы SWT.

Выполнение работы:

Шаг 1: Запуск приложения SWT

Для запуска независимого приложения SWT установим аргументы VM в закладке Run Wizzard/argument в следующие (обратим внимание на прямые слеши - /):

-Djava.library.path="C:/Program Files/eclipse/plugins/org.eclipse.swt.win32_3.0.0/os/win32/x86" где C:/Program Files/eclipse заменяется вашим каталогом инсталляции Eclipse.

Шаг 2: Разработка списка To Do

Разработаем представление с именем SecondToDoList. Добавим внешний файл swt.jar в маршрутвашего проекта. Этот файл находится в каталоге подключения SWT каталога инсталляции Eclipse.

Будем использовать ToDoListView - находится в Lab13, - который содержит код из слайда, как стартовую точку для нового представления.

Шаг 3: Модель светофора

Разработаем модель светофора, которая является визуальным представлением светофора. Модель светофора обеспечивается, как класс

TraficLight- находится вLab 13:

```
public class TrafficLight {
int currentState;
// Конструктор, который создает красный светофор
public TrafficLight() { currentState = 1;
}
// Перевод светофора в следующее состояние
public int advance() { setState(getState() % 3 + 1); return currentState;
}
// Возврат состояния светофора (как числа от 1 до 3)
public int getState() {
return currentState;
}
// Установка состояния светофора (как числа от 1 до 3)
// Если число выходит за пределы, не делается ничего
public void setState(int newState) { if ((newState > 0) && (newState <4)) currentState =
newState;
}
// Возврат строкового представления светофора
public String toString() {
switch (getState()) {
case 1: return "Red Traffic Light"; case 2: return "Yellow Traffic Light"; case 3: return "Green
Traffic Light";
};
return "Broken Traffic Light";
}
public boolean isRed() {
return getState() == 1;
```

```
}
public boolean isYellow() {
return getState() == 2;
}
public boolean isGreen() {
return getState() == 3;
}
}
```

Представление будет содержать следующие элементы:

1. Текст (Text)
2. Флажок (CheckButton)
3. Выпадающий список (Combo)
4. Три радиокнопки (Radio Button)
5. Кнопку (Push Button) Advance
6. Список (List)

Можно разместить элементы любым образом, только чтобы они не перекрывали друг друга. Общее размещение будет выглядеть так, как показано на картинке ниже (Рис.1).

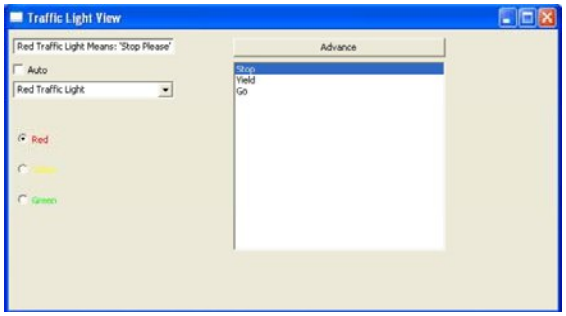


Рис.1 Общее размещение

Каждый элемент может быть использован для установки модели, за исключением элемента Text (показан в верхнем левом углу). Например, нажатие на кнопку Advance изменяет состояние семафора. Мы можем выбрать цвет в List или использовать Radio Buttons, или использовать Combo. Установка флажка Auto заставляет цвет циклически меняться каждую секунду. Важной частью решения является то, что независимо от того, какой элемент устанавливает состояние семафора, все другие элементы должны быть изменены для отражения нового состояния.

Когда мы создаем разные типы кнопок, мы задаем разные параметры для конструктора кнопки:

pushButton = new Button(shell, SWT.PUSH); //создает обычную кнопку radioButton = new Button(shell, SWT.RADIO); //создает радиокнопку checkBox = new Button(shell, SWT.CHECK); //создает флажок

Для создания Timer, который посылает сообщение каждые n миллисекунд, мы используем следующее: new javax.swing.Timer (n, new TimerEventHandler()); Сообщение actionPerformed посылается в TimerEventHandler. Это класс, который мы создаем и используем для выполнения действий через определенные периоды:

```
private class TimerEventHandler implements ActionListener { public void
actionPerformed(ActionEvent e) {
// код обработчика
}
}
```

Большинство элементов имеет свойства цвета. Можно посмотреть их в конкретных классах, но обычно делают следующее:

aButton.setForeground (new Color(display, 255,0,0));

Вывод: получили навыки в использовании одного из наиболее популярных менеджеров раскладки: FormLayout, многих элементов (widgets) SWT и построении графических пользовательских интерфейсы, используя элементы SWT.

Критерии оценивая:

- оценка «*отлично*» выставляется студенту, если задание выполнено в полном объеме, программа работают без ошибок, студент хорошо ориентируется в технологии построения визуального интерфейса, а разработанные формы отвечают современным требованиям эргономики;
- оценка «*хорошо*» выставляется студенту, если он выполнил задание в полном объеме, компоненты не достаточно полно настроены; созданный интерфейс не в полной мере отвечает современным требованиям эргономики;
- оценка «*удовлетворительно*» выставляется студенту, если он выполнил работу не в полном объеме, но не менее 60%; компоненты настроены частично; интерфейс не в полной мере отвечает современным требованиям эргономики;
- оценка «*неудовлетворительно*» выставляется студенту, если он выполнил менее 60% заданий.

3. ОЦЕНОЧНЫЕ МАТЕРИАЛЫ (СРЕДСТВА) ДЛЯ ПРОВЕДЕНИЯ ПРОМЕЖУТОЧНОЙ АТТЕСТАЦИИ ПО ДИСЦИПЛИНЕ

ВОПРОСЫ ДЛЯ ПОДГОТОВКИ К ЭКЗАМЕНУ

ПК-2 Способен проектировать информационные системы
ПК-2.1. Моделирует прикладные процессы предметной области с учетом внедрения сквозных цифровых технологий и передовых ИТ-решений прикладной сферы
Обучающийся знает: основные приемы объектно-ориентированного решения задач на языке Java; особенности разработки объектно-ориентированных программ на языке Java, их возможности и достоинства.
Оценка достижения обучающимся запланированного результата обучения осуществляется по результатам его участия в устных опросах и ответа на экзамене.

1. Классы в языке Java: особенности реализации, определение класса.
2. Классы в языке Java: управление доступом к элементам класса; понятие пакета.
3. Классы в языке Java: поля класса.
4. Классы в языке Java: методы, конструкторы.
5. Интерфейсы в языке Java: определение интерфейса, реализация интерфейса.
6. Оператор присваивания. Порядок действий (приоритет операторов).
7. Арифметические операторы. Операторы инкремента и декремента.
8. Встроенный класс Math. Псевдослучайные числа.
9. Операторы сравнения и логические операторы.
10. Операторы ветвления. Условный оператор. Минимизация количества проверок.
11. Операторы ветвления. Оператор множественного выбора. Его сравнение с условным оператором.
12. Встроенный класс String. Строковые операции.
13. Стандартные потоки ввода-вывода. Организация ввода и вывода данных. Класс Scanner

ПК-2 Способен проектировать информационные системы
ПК-2.2. Разрабатывает модель информационной системы, в том числе с опорой на современные облачные решения

Обучающийся знает: основные приемы создания компонентов информационных систем для обеспечения и анализа прикладных процессов.
Оценка достижения обучающимся запланированного результата обучения осуществляется по результатам его участия в устных опросах и ответа на экзамене.

1. Понятие Java API. Основные пакеты Java API.
2. События в Java: понятие события; типы событий; иерархия классов событий.
3. События в Java: модель делегирования событий.
4. События в Java: интерфейсы блоков прослушивания событий; способы реализации блока прослушивания.
5. События в Java: использование внутренних классов, классов-адаптеров для упрощения обработки событий.

ПК-3 Способен разрабатывать информационные системы
ПК-3.1. Осуществляет кодирование на современных языках программирования, в том числе используя возможности специализированных цифровых платформ для индивидуальной и совместной разработки информационных систем
Обучающийся знает: знает синтаксис, типы данных и операторы языка Java, правила разработки программ и подпрограмм, описания классов, обращения к его методам и данным.
Оценка достижения обучающимся запланированного результата обучения осуществляется по результатам его участия в устных опросах и ответа на экзамене.

1. Особенности языка и платформы Java.
2. Классификация программ по типу исполнения (компилируемые, интерпретируемые, исполняемые на виртуальных машинах). Виртуальная машина Java. JIT-компиляция.
3. Создание простейшей программы на Java, её компиляция в байт-код и запуск.
4. Средства разработки Java-приложений. Интегрированные среды разработки.
5. Встроенные типы данных. Способы задания литералов различных типов.
6. Хранение данных в памяти ЭВМ.
7. Приведение типов (явное и автоматическое). Константы и переменные.
8. Типы Java-приложений, их особенности.
9. Типы данных в языке Java: простые и ссылочные типы.
10. Массивы в Java: массивы простых типов и массивы объектов.

ПК-3 Способен разрабатывать информационные системы
ПК-3.2. Проводит тестирование компонентов информационных систем, в том числе используя возможности специализированных цифровых платформ
Обучающийся знает: знает основные приемы тестирование компонентов информационных систем на языке Java.
Оценка достижения обучающимся запланированного результата обучения осуществляется по результатам его участия в устных опросах и ответа на экзамене.

1. Исключения: понятие исключения; классы исключений; необходимость обработки исключений.
2. Исключения: операторы языка Java, используемые для обработки исключений.
3. Исключения: организация обработки исключений; определение собственных исключений.
4. Потоки вычислений: понятия процесса, потока.
5. Ввод/вывод в Java: основные понятия.
6. Ввод/вывод в Java: основные группы классов и интерфейсов пакета java.io.

ТИПОВЫЕ ЗАДАНИЯ И ЗАДАЧИ ДЛЯ ПОДГОТОВКИ К ЭКЗАМЕНУ

ПК-2 Способен проектировать информационные системы
ПК-2.1. Моделирует прикладные процессы предметной области с учетом внедрения сквозных цифровых технологий и передовых ИТ-решений прикладной сферы
Обучающийся умеет: проектировать семейства классов в заданной прикладной области и применять их для решения профессиональных задач на языке Java..

Обучающийся владеет: - навыками применения стандартных библиотек классов для разработки информационных систем на языке Java.
Задание 1. Реализовать класс для работы с комплексными числами: сложение, умножение, перевод из одной формы в другую, взятие модуля и аргумента
Задание 2. Реализовать класс для работы с матрицами: сложение, умножение (если возможно), транспонирование, вычисление определителя (для матриц 2-го и 3- го порядка)

ПК-2 Способен проектировать информационные системы
ПК-2.2. Разрабатывает модель информационной системы, в том числе с опорой на современные облачные решения

Обучающийся умеет: разработать объектно-ориентированные программы и программные прототипы на языке Java для решения прикладных задач.
Обучающийся владеет: навыками написания программного кода в объектно-ориентированной технологии.
Задание 1. Считать данные из заданного пользователем файла и подсчитать частотность употребления в этом файле всех символов английского алфавита. Результаты записать в файл с заданным именем.
Задание 2. Модифицировать задания 1 и 2 (Из компетенции ПК-2.1) так чтобы можно было создавать матрицы из комплексных чисел и выполнять операции над ними.

ПК-3 Способен разрабатывать информационные системы
ПК-3.1. Осуществляет кодирование на современных языках программирования, в том числе используя возможности специализированных цифровых платформ для индивидуальной и совместной разработки информационных систем
Обучающийся умеет: разрабатывать объектно-ориентированные программы и программные прототипы на языке Java для решения прикладных задач.
Обучающийся владеет: навыками написания программного кода в объектно-ориентированной технологии.
Задание 1. Фруктовая лавка. Создать абстрактный класс Фрукт и классы Яблоко, Груша, Абрикос расширяющие его. Класс Фрукт содержит:
а) переменную вес,
б) завершенный метод printManufacturerInfo(){System.out.print("Made in Ukraine");}
в) абстрактный метод, возвращающий стоимость фрукта, который должен быть переопределен в каждом классе наследнике. Метод должен учитывать вес фрукта.

ПК-3 Способен разрабатывать информационные системы
ПК-3.2. Проводит тестирование компонентов информационных систем, в том числе используя возможности специализированных цифровых платформ
Обучающийся умеет: составлять блоки тестов компонентов информационных систем на языке Java.
Обучающийся владеет: навыками применения тестов для информационных систем на языке Java.

Задание 1. Создать несколько объектов разных классов. Подсчитать общую стоимость проданных фруктов. А также общую стоимость отдельно проданных яблок, груш и абрикос.

ОБРАЗЕЦ ЭКЗАМЕНАЦИОННОГО БИЛЕТА

Экзаменационный билет состоит из одного теоретического вопроса и двух практических задания: первое на проектирование приложения (программы), второе – на программирование спроектированного приложения.

Тольяттинская Академия управления	Дисциплина Технология разработки объектно-ориентированных приложений на Java

Вариант 1

1. События в Java: использование внутренних классов, классов-адаптеров для упрощения обработки событий.
2. Считать данные из заданного пользователем файла и подсчитать частотность употребления в этом файле всех символов английского алфавита. Результаты записать в файл с заданным именем.
3. Реализовать класс для работы с комплексными числами: сложение, умножение, перевод из одной формы в другую, взятие модуля и аргумента.

4. МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ, ОПРЕДЕЛЯЮЩИЕ ПРОЦЕДУРЫ ОЦЕНИВАНИЯ ЗНАНИЙ, УМЕНИЙ, НАВЫКОВ, ХАРАКТЕРИЗУЮЩИХ ЭТАПЫ ФОРМИРОВАНИЯ КОМПЕТЕНЦИЙ

КРИТЕРИИ ОЦЕНИВАНИЯ СФОРМИРОВАННОСТИ КОМПЕТЕНЦИЙ

Планируемые результаты обучения (показатели достижения заданного уровня освоения компетенций)	Критерии оценивания результатов обучения				
	1	2	3	4	5
ПК-2 Способен проектировать информационные системы **ПК-2.1**. Моделирует прикладные процессы предметной области с учетом внедрения сквозных цифровых технологий и передовых ИТ-решений прикладной сферы					
знать: - особенности разработки объектно-ориентированных программ на языке Java, их возможности и достоинства; - основные приемы объектно-ориентированного решения задач на языке Java	Не знает	Фрагментарные знания основных приемов объектно-ориентированного решения задач на языке Java, особенностей разработки объектно-ориентированных программ языке Java, их возможностей и достоинств	Общие, но не структурированные знания основных приемов объектно-ориентированного решения задач на языке Java, особенностей разработки объектно-ориентированных программ языке Java, их возможностей и достоинств	Сформированные, но содержащие отдельные пробелы знания основных приемов объектно-ориентированного решения задач на языке Java, особенностей разработки объектно-ориентированных программ языке Java, их возможностей и достоинств	Сформированные систематизированные прочные знания основных приемов объектно-ориентированного решения задач на языке Java, особенностей разработки объектно-ориентированных программ языке Java, их возможностей и достоинств
уметь: проектировать семейства классов в заданной прикладной области и применять их для решения профессиональных задач на языке Java	Не умеет	Частично освоенные умения проектировать семейства классов в заданной прикладной области и применять их для решения профессиональных задач на языке Java	В целом освоенные, но не подкрепляемые знаниями и самостоятельностью выполнения умения проектировать семейства классов в заданной прикладной области и применять их для решения профессиональных задач на языке Java	В целом сформированные и самостоятельно выполняемые, но не всегда интеллектуально обоснованные умения проектировать семейства классов в заданной прикладной области и применять их для решения профессиональных задач на языке Java	Успешно сформированные, самостоятельно и осознанно выполняемые умения проектировать семейства классов в заданной прикладной области и применять их для решения профессиональных задач на языке Java
владеть: навыками моделирования и описания моделей	Не владеет	Фрагментарно сформированные навыки применения	В целом сформированные, но выполняемые на	Сформированные, но не устойчивые в сложных ситуациях	Успешно сформированные, устойчивые, технологическ

семейств классов		стандартных библиотек классов для разработки информационных систем на языке Java	элементарном уровне навыки применения стандартных библиотек классов для разработки информационных систем на языке Java	навыки применения стандартных библиотек классов для разработки информационных систем на языке Java	и грамотно выполняемые навыки применения стандартных библиотек классов для разработки информационных систем на языке Java
ПК-2 Способен проектировать информационные системы **ПК-2.2.** Разрабатывает модель информационной системы, в том числе с опорой на современные облачные решения					
Знать: основные приемы создания компонентов информационных систем для обеспечения и анализа прикладных процессов	Не знает	Фрагментарные знания основных приемов создания компонентов информационных систем для обеспечения и анализа прикладных процессов	Общие, но не структурированные знания основных приемов создания компонентов информационных систем для обеспечения и анализа прикладных процессов	Сформированные, но содержащие отдельные пробелы знания основных приемов создания компонентов информационных систем для обеспечения и анализа прикладных процессов	Сформированные систематизированные прочные знания основных приемов создания компонентов информационных систем для обеспечения и анализа прикладных процессов
Уметь: - проектировать компоненты информационных систем для обеспечения и анализа прикладных процессов на языке Java	Не умеет	Частично освоенные умения проектировать компоненты информационных систем для обеспечения и анализа прикладных процессов на языке Java.	В целом освоенные, но не подкрепляемые знаниями и самостоятельностью выполнения умения проектировать компоненты информационных систем для обеспечения и анализа прикладных процессов на языке Java.	В целом сформированные и самостоятельно выполняемые, но не всегда интеллектуально обоснованные умения проектировать компоненты информационных систем для обеспечения и анализа прикладных процессов на языке Java.	Успешно сформированные, самостоятельно и осознанно выполняемые умения проектировать компоненты информационных систем для обеспечения и анализа прикладных процессов на языке Java.
Владеть: навыками применения стандартных библиотек классов при разработке компонентов информационн	Не владеет	Фрагментарно сформированные навыки применения стандартных библиотек классов при разработке	В целом сформированные, но выполняемые на элементарном уровне навыки применения стандартных	Сформированные, но не устойчивые в сложных ситуациях навыки применения стандартных библиотек	Успешно сформированные, устойчивые, технологически и грамотно выполняемые навыки применения

ых систем для обеспечения и анализа прикладных процессов на языке Java		компонентов информационных систем для обеспечения и анализа прикладных процессов на языке Java	библиотек классов при разработке компонентов информационных систем для обеспечения и анализа прикладных процессов на языке Java	классов при разработке компонентов информационных систем для обеспечения и анализа прикладных процессов на языке Java	стандартных библиотек классов при разработке компонентов информационных систем для обеспечения и анализа прикладных процессов на языке Java
ПК-3 Способен разрабатывать информационные системы **ПК-3.1**. Осуществляет кодирование на современных языках программирования, в том числе используя возможности специализированных цифровых платформ для индивидуальной и совместной разработки информационных систем					
Знать: - знает синтаксис, типы данных и операторы языка Java, правила разработки программ и подпрограмм, описания классов, обращения к его методам и данным	Не знает	Фрагментарные знания синтаксиса, типов данных и операторов языка Java, правил разработки программ и подпрограмм, описания классов, обращения к его методам и данным	Общие, но не структурированные знания синтаксиса, типов данных и операторов языка Java, правил разработки программ и подпрограмм, описания классов, обращения к его методам и данным	Сформированные, но содержащие отдельные пробелы знания синтаксиса, типов данных и операторов языка Java, правил разработки программ и подпрограмм, описания классов, обращения к его методам и данным	Сформированные систематизированные прочные знания синтаксиса, типов данных и операторов языка Java, правил разработки программ и подпрограмм, описания классов, обращения к его методам и данным
Уметь: - разрабатывать объектно-ориентированные программы и программные прототипы на языке Java для решения прикладных задач	Не умеет	Частично освоенные умения разработать объектно-ориентированные программы и программные прототипы на языке Java для решения прикладных задач	В целом освоенные, но не подкрепляемые знаниями и самостоятельностью выполнения умения разработать объектно-ориентированные программы и программные прототипы на языке Java для решения прикладных задач	В целом сформированные и самостоятельно выполняемые, но не всегда интеллектуально обоснованные умения разработать объектно-ориентированные программы и программные прототипы на языке Java для решения прикладных задач	Успешно сформированные, самостоятельно и осознанно выполняемые умения разработать объектно-ориентированные программы и программные прототипы на языке Java для решения прикладных задач
Владеть: - навыками написания программного кода в	Не владеет	Фрагментарно сформированные навыки написания	В целом сформированные, но выполняемые на	Сформированные, но не устойчивые в сложных ситуациях	Успешно сформированные, устойчивые, технологическ

объектно-ориентированной технологии		программного кода в объектно-ориентированной технологии	элементарном уровне навыки написания программного кода в объектно-ориентированной технологии	навыки написания программного кода в объектно-ориентированной технологии	и грамотно выполняемые навыки написания программного кода в объектно-ориентированной технологии
ПК-3 Способен разрабатывать информационные системы **ПК-3.2**. Проводит тестирование компонентов информационных систем, в том числе используя возможности специализированных цифровых платформ					
Знать: основные приемы тестирование компонентов информационных систем на языке Java	Не знает	Фрагментарные знания основных приемов тестирования компонентов информационных систем на языке Java	Общие, но не структурированные знания основных приемов тестирования компонентов информационных систем на языке Java	Сформированные, но содержащие отдельные пробелы знания основных приемов тестирования компонентов информационных систем на языке Java	Сформированные систематизированные прочные знания основных приемов тестирования компонентов информационных систем на языке Java
Уметь: составлять блоки тестов компонентов информационных систем на языке Java	Не умеет	Частично освоенные умения составлять блоки тестов компонентов информационных систем на языке Java	В целом освоенные, но не подкрепляемые знаниями и самостоятельностью выполнения умения составлять блоки тестов компонентов информационных систем на языке Java	В целом сформированные и самостоятельно выполняемые, но не всегда интеллектуально обоснованные умения составлять блоки тестов компонентов информационных систем на языке Java	Успешно сформированные, самостоятельно и осознанно выполняемые умения составлять блоки тестов компонентов информационных систем на языке Java
Владеть: навыками применения тестов для информационных систем на языке Java	Не владеет	Фрагментарно сформированные навыки применения тестов для информационных систем на языке Java	В целом сформированные, но выполняемые на элементарном уровне навыки применения тестов для информационных систем на языке Java	Сформированные, но не устойчивые в сложных ситуациях навыки применения тестов для информационных систем на языке Java	Успешно сформированные, устойчивые, технологически грамотно выполняемые навыки применения тестов для информационных систем на языке Java

ПРОЦЕДУРА ПРОВЕДЕНИЯ ПРОМЕЖУТОЧНОЙ АТТЕСТАЦИИ И КРИТЕРИИ ОЦЕНКИ РЕЗУЛЬТАТОВ ОБУЧЕНИЯ ПО ДИСЦИПЛИНЕ

Для оценки результатов обучения по дисциплине учебным планом предусмотрен экзамен, проводимый в письменной или устной формах. При устном ответе обязательным является подготовка студентом простого или развернутого ответа. При выставлении итоговой оценки за дисциплину необходимо учитывать результаты текущего контроля (или: результаты текущего контроля являются допуском к экзамену) и критерии оценивания сформированности компетенций:

- оценка «отлично» выставляется студенту за такие знания, когда обучающийся показывает всестороннее, систематическое и глубокое усвоение всего объема учебно-программного материала, умеет свободно выполнять задания, предусмотренные программой курса, осмысленно применяет полученные знания на практике, усвоивший основную и знакомый с дополнительной литературой, рекомендованной программой курса, не допускает ошибок при воспроизведении знаний в объеме пройденного программного материала, грамотное и логически стройное изложение материала при ответе, а также в письменных работах и выполняет последние уверенно и аккуратно, легко отвечает на видоизмененные вопросы, на которых нет прямых ответов в учебной литературе; практическое задание (заданий) выполнено безошибочно и в полном соответствии с требованиями, обучающийся демонстрирует устойчивые навыки работы и осознанно исполняемые действия для разработки программных продуктов, способность вносить коррективы и обосновывать свои действия;
- оценка «хорошо» выставляется студенту тогда, когда обучающийся выявляет полные знание учебно-программного материала, отвечает без особых затруднений на вопросы преподавателя, успешно выполняет предусмотренные в программе курса задания, знакомый с основной литературой, рекомендованной программой курса, умеет применять полученные знания на практике, в устных ответах не допускает серьезных ошибок и легко устраняет отдельные неточности с помощью дополнительных вопросов преподавателя, в письменных работах делает незначительные ошибки; практическое задание (задания) выполнено с незначительными замечаниями и ошибками, в соответствии с требованиями, обучающийся демонстрирует навыки работы и осознанно исполняемые действия для разработки программных продуктов, но затрудняется внести коррективы;
- оценка «удовлетворительно» обучающийся обнаруживает усвоение основного учебно-программного материала в объеме, необходимом для дальнейшей учебы и дальнейшей профессиональной деятельности, справляющийся с выполнением предусмотренных в программе курса заданий, но испытывает затруднение при его самостоятельном воспроизведении и требует дополнительных и уточняющих вопросов преподавателя, предпочитает отвечать на вопросы воспроизводящего характера и путается при ответах на видоизмененные вопросы, умеет применять полученные знания на практике, знакомый с основной литературой, рекомендованной программой курса, допускает ошибки в письменных работах; практическое задание (задания) выполнено с грубыми ошибками и множеством замечаний, с отклонениями от требований, обучающийся демонстрирует непрочные навыки работы, затрудняется в объяснении применяемых средств и конструктивов для разработки программных продуктов, но способен внести отдельные коррективы в работу;
- оценка «неудовлетворительно» выставляется студенту, обнаружившему в знаниях учебно-программного материала, принципиальные ошибки в выполнении предусмотренных программой курса заданий, в письменных работах допускаются грубые ошибки; практическое задание (задания) не выполнено (программа не работает) или выполнено со значительными отклонениями от требований, с грубыми ошибками и множеством замечаний, обучающийся демонстрирует частично сформированные навыки работы, затрудняется в объяснении применяемых средств и конструктивов для разработки программных продуктов, не способен внести коррективы в работу.

5. ЛИСТ СОГЛАСОВАНИЯ

Составил:

Н.Б. Стрекалова, д.п.н., доцент

(подпись)

Заведующий кафедрой

Н.Б. Стрекалова, д.п.н., доцент

(подпись)

Заведующий выпускающей кафедрой

Н.Б. Стрекалова, д.п.н., доцент

(подпись)