# Методические материалы по дисциплинам

## Инжиниринг ПО. Часть 1. Паттерны программирования и проектирования ПО

Дисциплина ориентирована на дальнейшее развитие навыков программирования и проектирования ПО с использованием шаблонов проектирования и стандартных технологических средств языка программирования и среды разработки, получение навыков написания качественного управляемого кода в проектах небольшого и среднего размера.

**Баллы рейтинга**

|  |  |
| --- | --- |
| № л.р. | Балл |
| 1 | 10 |
| 2 | 10 |
| 3 | 7 |
| 4 | 13 |
| РГР | 40 |
| Зачет | 20 |

### Лабораторный практикум

**Лабораторная работа 1. Графический интерфейс с динамической разметкой.** Разработать графический интерфейс с программируемой разметкой: переменное количество элементов, их расположение и перемещение, регулярная обработка событий.

1. Игра «Пятнашки» с произвольной размерностью. Поле создается на основе двумерного массива TextField-ов.
2. Игра «Судоку» с произвольной размерностью. Поле создается на основе двумерного массива TextField-ов.
3. Электронная таблица типа Excel с формулами – арифметические действия со значениями ячеек и константами.Таблица создается на основе двумерного массива TextField-ов. Контекстное меню по правой кнопке.
4. Турнирная таблица - симметричное заполнение, определение суммы очков и места. Контекстное меню по правой кнопке. Поле создается на основе двумерного массива TextField-ов.
5. Игровой автомат с прокруткой картинок. Анимация прокрутки с плавным перемещением картинки в панели.
6. Редактор разметки для кнопок: создание, установка начального текста и иконки, перетаскивание по форме, изменение размеров. Вариант: по правой кнопке поднимается форма с основными параметрами элемента: размеры, положение, цвет фона и текста, размеры шрифта заполняются текущими значениями параметров. При закрытии – применяются к элементам. Вариант: перетаскивание элемента drag-and-drop.
7. Игра «Шашки». Расстановка шашек, выполнение ходов по правилам: простой ход, взятие, без выхода в «дамки» (клик по начальной/конечной позиции)
8. Игра «Домино» - выставление костей в цепочку, кость – пара кнопок с иконками или изображение.
9. «Рулетка» - вращающееся колесо числами или иконками. При остановке выводится выпадающее число или иконка. При отсутствии средств поворота изображения перемещать горизонтально расположенное текстовое поле с числом или иконку по дуге окружности. Реалистичная анимация вращения с постепенной остановкой.
10. Визуализатор графа. Вершины – текстовые поля, ребра - прорисовываемые линии. Добавление/удаление вершин и дуг. При перемещении вершин все связанные дуги «тянутся»

Аниматор сортировки: элементы сортировки – текстовые поля с числами. Графически отображается процесс перестановки элементов в массиве или перемещения из массива в массив. При выборе минимального элемента подсвечиваются текущий сравниваемый и минимальный. Перенос и обмен элементов производится в виде «вверх-влево(вправо)-вниз». Варианты сортировки:

1. Однократное слияние. Обменная сортировка частей (минимальная анимация), слияние – реалистичная анимация. (cprog 4.6).
2. Сортировка Шелла с переменным шагом. Подсветка текущей сортируемой группы (cprog 4.6).
3. Циклическое слияние, группы подсвечивать различнымы цветами, реалистичное перемещение из входных массивов в выходной (cprog 4.6).
4. Разделение – быстрая сортировка, подсветка текущей сортируемой части (cprog 7.2).
5. Разделение: подсвечивается медиана, последовательность раскидывается в два динамических массива, по окончании они сбрасываются в исходный, анимация – элементы разбрасывается налево-направо, затем части возвращаются на исходный уровень. Подсветка текущей сортируемой части.
6. Слияние (cprog 7.2).
7. Лексикографическая c разбрасыванием исходного массива на 10 групп по значению очередной цифры и последующим объединением (cprog 7.7)

**Лабораторная работа 2. Рефлексия и аннотации.** С использование средств рефлексии разработать универсальный сервис, применимый для объектов произвольного класса. Передается имя класса или его описатель класса Class. Разработать универсальный сервис, использующий аннотированные элементы класса.

1. Класс, выполняющий клонирование произвольного объекта, передаваемого по ссылке. Создает экземпляр объекта, просматривает список полей, копирует примитивные поля, для ссылочных типов – рекурсивно вызывает клонирование.
2. Редактор содержимого произвольного объекта, передаваемого по ссылке. Оконный класс со списком полей данных объекта - примитивных типов.
3. Класс - упорядоченный вектор по заданному по имени произвольному элементу данных класса - примитивному типу или строке. Класс содержит вектор, ссылку на класс хранимых данных и имя поля, по которому производится сортировка. Разработать метод сортировки, алгоритмы вставки с сохранением порядка и двоичного поиска.
4. Собственный JSON-сериализатор/десериализатор с сериализацией ссылок и массивов ссылок.
5. Собственный XML-сериализатор с сериализацией ссылок и массивов ссылок.
6. Сериализатор/десериализатор с собственным текстовым форматом сериализации ссылок и массивов ссылок.
7. Класс – документатор. При вводе имени класса создает документ в формате pdf или html c описанием структуры объекта, а также тех классов, на которые в объекте имеются ссылки или массивы ссылок. Обработка производится рекурсивно, с исключением повторной обработки уже просмотренных классов.
8. Класс – документатор, аналогичный вар. 6, но для содержимого объекта. Вывод содержимого объекта и объектов, доступных по ссылкам и через массивы ссылок. Создает документ в формате pdf или html.
9. Класс – конечный автомат с вызовом методов, *прикрепляемых* к переходам и состояниям по имени. В описании автомата перечисляется список дуг в виде *входное состояние – входной символ – состояние перехода – имя метода, вызываемое при переходе.* Сами методы реализуются в производном классе для класса-автомата.
10. Собственный оконный класс с вызовом методов обработки событий по имени. Оконный класс, производный от разработанного оконного класса, использует аннотации для ссылок на кнопки вида

@Button (x="100" y="200" w="50" h="20" click="proc1")

private JButton bb;

В классе должен быть определен метод обработки события

private void proc1(){…..}

Конструктор класса просматривает аннотированные поля производного класса, создает кнопки с заданными параметрами, размещает в окне, создает обработчик события на собственный код, записывает ссылку в аннотированное свойство (bb). При выполнении клика по кнопке обработчик находит ассоциированное имя метода, находит Method по этому имени и вызывает его.

*Замечание:* В имени bb содержится ссылка на созданный Button. Протестировать в производном классе

1. «Дизассемблер Java». Сервис получает объект произвольного класса и конвертирует его заголовок в тест на языке Java.
2. Конвертер произвольного класса в Kotlin. Сервис получает объект произвольного класса и конвертирует его заголовок в тест на языке Kotlin (Scala) или ООП-язык с другой нотацией.
3. Фабрика на основе аннотированных констант. Класс содержит аннотированные константы вида

**public final static int *ModBusPriorityLevels***=4;  
**@CONST**(group = "Priority", title = "Interrupt", clazz = A.class)  
**public final static int *MBPInterrupt*** = 0;  
**@CONST**(group = "Priority", title = "High", clazz = B.class)  
**public final static int *MBPHigh*** = 1;

Каждая константа имеет свое имя, название (title), группу, значение и присоединенный класс. Получив класс с аннотированными константами, компонента вернуть ArrayList описателей констант (имя, название (title), значение и присоединенный класс). Тест получает от класса ArrayList описателей, создает в GUI выпадающий список названий и при выборе элемента создает соответствующий объект класса.

1. Аннотированный метод с асинхронным вызовом аннотацию вида

@RxMethod(info=**"output = sum of 3 inputs"**, name = **"sum"**)  
**public void** sum(String signature,

@RxParam(name=**"inp1"**) RxIntegerValue a,

@RxParam(name=**"inp2"**) RxIntegerValue b,

@RxParam(name=**"inp3"**) RxIntegerValue c)

**throws** RxException {...}

Сервис получает объект произвольного класса и создает ArrayList(HashMap) аннотированных методов. При передаче сервису запроса вида sum:inp1=15,inp2=16, inp3=4 (в любом формате) ищет в классе аннотированный метод и асинхронно вызывает его в объекте.

1. Аннотированное свойство класса имеет вид

@RxField(info="",name = "**Inp1**")

**private** int vv=0;

Сервис получает объект произвольного класса и создает ArrayList(HashMap) аннотированных свойств. При передаче сервису запроса вида inp1=15 (в любом формате) ищет в классе аннотированное свойство и асинхронно записывает или читает его значение в объекте.

1. Двоичный сериализатор аннотированных свойств вида

@RxField(info="",name = "**Inp1**")

**private** int vv=0;

Сервис получает объект произвольного класса и создает ArrayList(HashMap) аннотированных свойств. При получении объекта сериализует его **в двоичный файл** в формате: имя класса (readUTF/writeUTF), количество полей, последовательность: имя поля (readUTF/writeUTF)-значение в двоичном формате (int,short,double,String). Десериализатор открывает файл и десериализует из него объект.

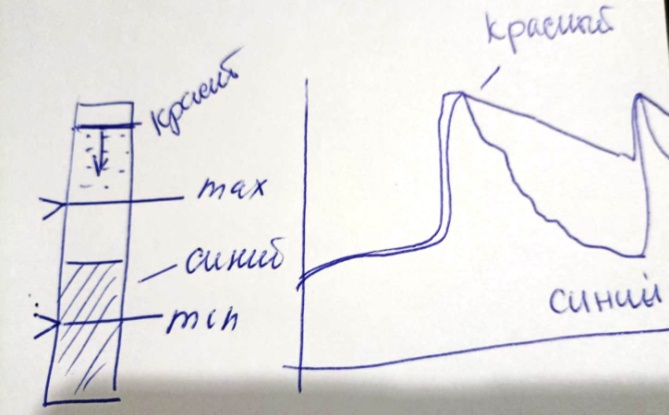
1. Сериализатор аннотированных свойств вида в текстовом формате

@RxField(info="",name = "**Inp1**")

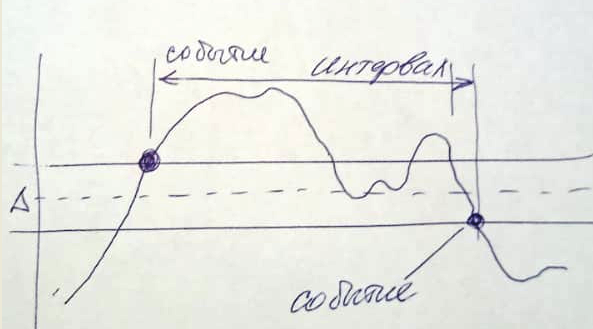
**private** int vv=0;

Сервис получает объект произвольного класса и создает ArrayList(HashMap) аннотированных свойств. При получении объекта сериализует его **в текстовый файл** в формате: имя класса, количество полей, последовательность: имя поля - значение в тестовом формате. Десериализатор открывает файл и десериализует из него объект. **Лабораторная работа 3. Разработка элемента GUI с динамическим поведением.** Производный класс на основе базового элемента графического интерфейса (JPanel) с динамическим поведением. Класс способен накапливать, отображать и возвращать накопленные данные. При необходимости периодической передачи данных с определенной частотой получает при конструировании интерфейс слушателя события с методом *очередное значение данных*.

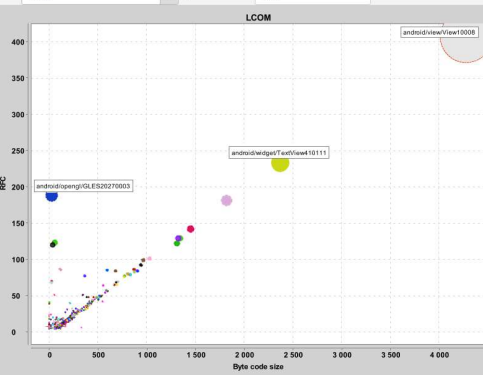
1. *Пиковый индикатор*. Вертикальный полосковый индикатор уровня с установкой минимального и максимального порога и *инертным индикатором максимума*: текущий уровень – синяя полоска, максимальный – красная. При появлении пика он устанавливает уровень красной полоски, которая при затем линейно снижается со времени. В момент превышения максимального порога генерирует событие.

**

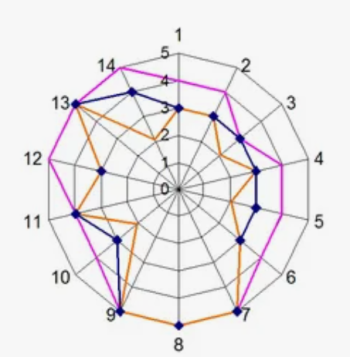
1. Стрелочный инерционный прибор. При изменении значения имитирует и отображает затухающие колебания вокруг нового значения. При входе и выходе из красной зоны, обозначенной на индикаторе, генерирует событие.
2. График записываемых значений с отображением N последних записанных точек и *прокруткой* графика при выводе. Можно запросить время записи и значение N последних записанных точек.
3. Индикатор уровня с зонами нормального уровня (зеленый), предупреждения (желтый) и аварии (красный). Изображает вертикальную полоску высотой, пропорциональной значению и цветом, зависящим от диапазона. Параметры настройки: минимальное и максимальное значение (диапазон), нижняя и верхняя границы аварии, нижняя и верхняя границы предупреждения.
4. Панель, отображающая координаты точки на плоскости с отображением N последних записанных точек. При конструировании задается размер панели, диапазон значений по координатам, количество точек N.
5. Пиковый индикатор. При плавном изменении значений входного потока данных (разность значение по модулю меньше D0) зеленый цвет индикатора, при разности значений от D0 до D1 – желтый цвет индикатора, больше D1 – красный. При изменении цвета зеленый-желтый, желтый-красный – задерживает цвет на заданный интервал. При изменении цвета – CallBack c параметрами: время, значение, разность, тип события. Параметры конструктора: D0,D1, задержка цвета.
6. Светофор. Отображает светофор с состояниями *красный – красный/желтый желтый – зеленый – зеленый мигающий – желтый.* Программируется продолжительность фаз. При смене состояния генерирует событие. Цифровая индикация оставшегося времени для текущей фазы. Включение и выключение светофора. В выключенном состоянии – *желтый мигающий.*
7. Гистограмма, запоминающая количество отсчетов, их сумму, сумму квадратов и количество попаданий в карманы. Отображение текущего вида гистограммы в нормированном виде (максимальное значение по высоте панели). Можно запросить количество отсчетов в каждом кармане, среднее и дисперсию множества записанных значений. Задается диапазон и количество карманов.
8. Индикатор уровня с CallBack по переходу через границу (компаратор). Изображает вертикальную полоску высотой, пропорциональной значению. При создании передается CallBack для событий пересечения границы: время, направление пересечения. Параметры настройки: минимальное и максимальное значение (диапазон отображения), граница и ширина границы (дельта гистерезиса для исключения «дребезга»).



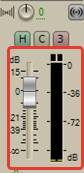
1. Стрелочно-цифровой секундомер или часы с прогресс-индикатором отметки событий на циферблате. Событие содержит тестовое сообщение. Можно запросить любое из N последних событий.
2. Диаграмма: отображение точек с 4 параметрами double x,y,z,w на панели: x,y преобразуются в координаты по осям, z – в цвет (функция преобразования значения), v в диаметр окружности. Повторное масштабирование при добавлении точки (при необходимости).



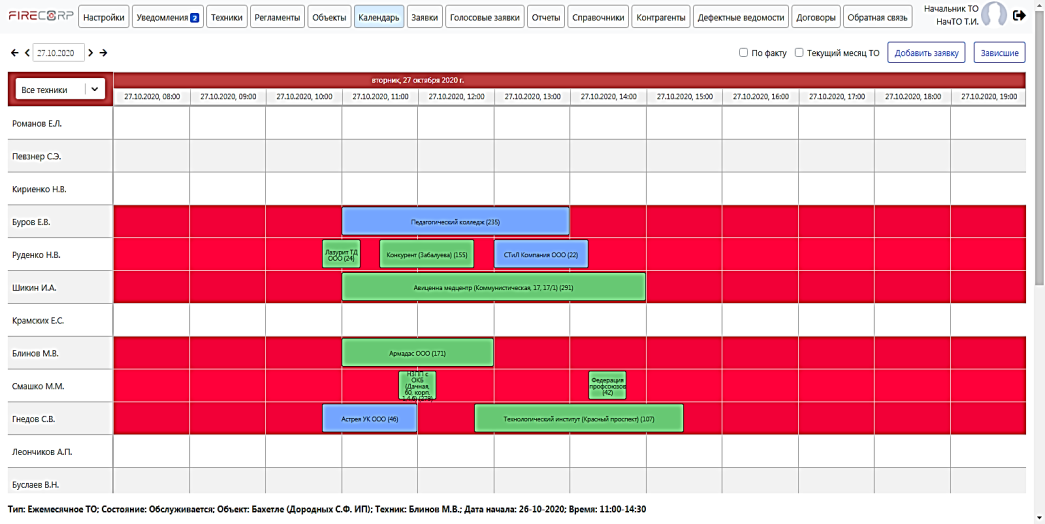
1. Индикатор состояний с набором изображений, соответствующих состояниям. В конструкторе задаются имена файлов или идентификаторы ресурсов и соответствие их состояниям, минимальная частота смены. При передаче состояния отображает соответствующую картинку, при быстрой смене состояний поддерживает очередь, чтобы изображения не мелькали.
2. Радиальная диаграмма. Задается количество отображаемых параметров в группе. Количество групп может быть разным (задается), каждой группе соответствует свой цвет. Для каждого параметра – значение, соответствующее норме. На диаграмме обозначения уровень нормы и удвоенной нормы. При изменении любого значения в любой группе – перерисовка диаграммы, для значения, превышающего удвоенную норму, специальный значок (само значение отображается на уровне удвоенной нормы).



1. Движок уровня и индикатор значения в логарифмической шкале (децибел, dB). Задается целое значение V0, которое является нулевым уровнем. Индикатор, при получении значения вычисляет его уровень в dB – **20\*log10(V/V0)**. Диапазон шкалы задается (например, +/-100dB). При перемещении движка через CallBack возвращается значение позиции в исходном целом виде (преобразование рассчитать).



1. План-календарь задач (работ). Имеется список исполнителей и заданный период времени. Исполнителю добавляется работа и интервал начала/завершения. CallBack по клике по индикатору работы с ее параметрами (исполнитель, дата/время начала/окончания)



1. Прогресс-индикатор исполнения работы. Задается интервал времени исполнения работы, добавляется значение <дата/время – процент выполнения>. Процент выполнения визуализируется цветом. Достижение 100% или завершение срока с незавершенной работой отмечается на диаграмме и передается событием в CallBack.



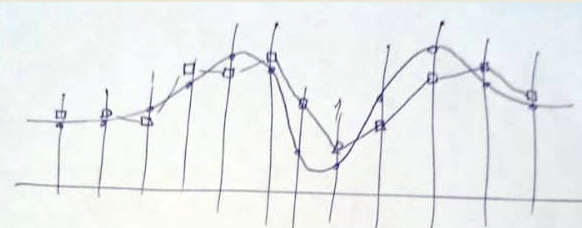
1. Двухуровневая палитра выбора цвета. Задается палитра в виде групп цветности. При клике по элементу открывается (или проецируется в соседнюю панель) для более точного выбора цвета. При клике по точке заполняются поля компонент цветности RGB. При вводе значений компонент – обратное позиционирование к палитре (Можно предложить собственный вариант выденения цветов).



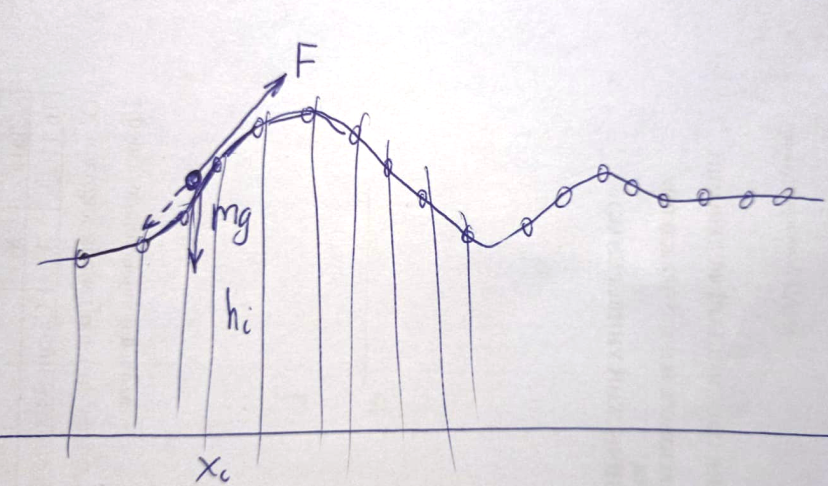
**Лабораторная работа 4. Модель с подпиской на события.** Класс-контроллер реализует в отдельном потоке в реальном времени *модель поведения физического объекта или системы.* Классы GUI (формы) получают ссылку на объект-контроллер с *интерфейсом команд* и *подписываются* на получение от него периодических данных и *событий* через соответствующий интерфейс (шаблон *наблюдатель*). При появлении в контроллере события последний вызывает соответствующий метод во всех подписанных объектах. При закрытии объекта класса GUI он *отписывается* от контроллера. События и команды в интерфейсах должны обеспечивать объектам внешнего представления полный набор возможностей управления и отображения состояния физического объекта. При тестировании в качестве подписчика на события можно использовать:

* графический элемент GUI, разработанный в л.р.3;
* подходящие элемент GUI других бригад;
* библиотеки для построения графиков;
* экспорт данных в Excel и построение на их основе элементов визуализации.

1. Физический объект – кастрюля на плите: текущее положение конфорки, температура и объем содержимого, состояние нагревания и кипения, теплоотдача при открытой и закрытой крышке.
2. Физический объект – звуковая волна в формате wave-файла. Ручное проматывание и автоматическое проигрывание. Объекты – адаптеры для изменения амплитуды и задержки сигнала, микширования – суммирования сигналов от разных источников (паттерн **прокси)**.
3. Физический объект – точка, движущаяся в гравитационном поле других стационарных точек в двумерной системе координат. Управление – ускорение и торможение по вектору скорости, или по заданному углу к вектору скорости.
4. Модель системы потребителей электроэнергии. Элемент – чайник с автоматическим выключением при закипании, лампочка, компьютер с источником бесперебойного питания. Потребители имеют устанавливаемую мощность и состояние включено/выключено, а также модель поведения в зависимости от наличия и отсутствия напряжение (нагрев и остывание чайника, работа бесперебойника при отключенном питании). Модели поведения потребителей также моделируются соответствующими контроллерами с внутренними потоками. Возможно подключение и отключение произвольного количества потребителей разных типов. Фиксируется общее потребление и моделируется отключение сети при перегрузке.
5. Физическая модель. «Грузик» раскачивается и колеблется на пружине. Имеется начальное отклонение грузика от R-R0 – от исходного размера пружины и угол поворота fi. На грузик действуют сила сжатия/растяжения пружины и сила тяжести (проекция m\*g\*sin(fi) вызывает угловое ускорение m\*g\*cos(fi) – складывается с силой сжатия/растяжения.
6. Физическая модель. «Грузик» раскачивается и колеблется на растяжимой нити (резинке) . Имеется начальное отклонение грузика от R-R0 – от исходного размера и угол поворота fi. На грузик действуют сила растяжения нити (если длина больше R0) и сила тяжести (проекция m\*g\*sin(fi), всегда)
7. Физическая модель. Автомобили, движущиеся от светофора. Имеется стандартная скорость движения автомобиля по трассе, ограничения ускорения (разгон, торможение). Автомобили въезжают на трассу с рандомным интервалом. Смоделировать стратегию разгона: ускорение (разгон) автомобиля зависит от дистанции с предыдущим, собственной скорости и скорости предыдущего. Торможение у светофора моделируется путем мгновенной остановки с заданной дистанцией. Определить максимальную интенсивность входного потока при заданных параметрах разгона. Сравнить со стратегией одновременного и одинакового разгона для всех (рассчитать).
8. Физическая модель. Следящая систем (система автоматического регулирования). Имеется объект управления с заданной массой (инертность движения) и внешняя траектория движения (по одной из осей). Управление заключается в прикладывании силы, пропорциональной (коэффициент) разности между текущим положением точки и траектории (величина силы ограничена). Возможно предложить свой алгоритм управления, основанный на измерении разницы координат, скоростей их изменения, предыдущего тренда и пр.. Для более точного моделирования возможна литейная интерполяция промежуточных точек траектории.



1. Моделирование взлета 3-ступечатой ракеты-носителя. Задана масса выводимого аппарата, каждая ступень имеет собственный вес, вес топлива, силу тяги. Задан показатель – сила тяги на 1 кг веса топлива. Сила тяги ступени постоянна и превышает вес оставшихся ступеней на момент начала работы. Отделение ступени на момент исчерпания топлива. CallBack по событиям отделения ступени, периодическое событие – текущее состояние: количество ступеней, масса, скорость, высота от точки старта.
2. Физическая модель управления движением транспортного средства с заданной массой (автомобиль, ж/д состав) по рельефу. Рельеф задан множеством точек – xi,hi Между ними значения генерируются при помощи линейной интерполяции. Можно задавать ограниченную текущую силу (ускорение) разгона/торможения. В противовес (или по совпадающему направлению) действует проекция силы тяжести. CallBack по событиям – скатывание назад, периодическое событие с текущим состоянием: координата, скорость.



1. Физический объект – автомобиль, движущийся по прямой трассе со светофорами. Модели движения точечной массы с заданным положительным или отрицательным ускорением (педали *газа и тормоза*). Модель получает линейное расположение светофоров, возможно добавление светофора на указанном расстояния от последнего. Событие – *проезд на красный*, данные модели – текущая координата, скорость, расстояние до ближайшего светофора.
2. Движение по трассе c поворотами. Трасса определяется последовательностью точек с координатами x,y на плоскости. Промежуточные точки определяются при помощи линейной интерполяции. Автомобиль имеет массу, допустимый диапазон ускорения разгона и торможения, допустимый угол поворота руля – направление ускорения по отношению к вектору скорости. Трасса имеет заданную ширину. CallBack по событиям– выезд за пределы трассы, периодическое событие с текущим состоянием: координата, скорость.

### Расчетно-графическая работа

РГР выполняется в виде разработки одного из шаблонов проектирования. Пояснительная записка должна содержать:

* учебно-методический материал по шаблону - учебники, учебные пособия, лекционный материал, тематические сайты;
* материал по практике применения шаблона - форумы, тематические сайты;
* описание разработки;
* результаты тестирования.

Все сторонние материалы должны быть снабжены корректно оформленными библиографическими ссылками.

Варианты заданий на основе л.р. 4

**Разработка многооконного интерфейса (20 баллов).** Разработать многооконный GUI, в котором в каждом окне имеется оригинальный набор датчиков и элементов управления. Окна открываются из основного окна. Количество экземпляров окон не ограничено. Главное окно создает объект-контроллер и передает создаваемым окнам ссылку на него.

**Дополнительные опции. Java Native Interface (JNI) (10 баллов).** Реализовать часть наиболее трудоемких методов в виде native-функций на Си++, сравнить производительность.

**Дополнительные опции (10 баллов). Потоковое программирование в приложении.** Реализовать времяёмкие операции в виде отдельных потоков, параллельных потоку GUI, предусмотреть синхронизацию окончания операции, аварийное завершение, возможность периодического вывода параметров фонового потока в индикатор прогресса.

**Дополнительные опции (10 баллов). Тестовые, двоичные и сериализуемые потоки данных.** Реализовать средства сохранения и загрузки полного состояния всех объектов программы в тестовый, двоичный и сериализуемый потоки.

**Дополнительные опции (10 баллов). Использование JDBC-библиотек для работы с базами данных.** Реализовать средства сохранения и загрузки параметров настройки, собираемой статистики и прочих данных в БД SQLite.

Варианты заданий. Паттерны программирования (проектирования)

1. *Шаблон MVC*. **Сетевая (локальная) игра типа «Морской бой», «домино».** Модель хранит структуру данных игры - расстановку кораблей, и над ней выполняются методы по управлению игрой. Контроллер определяет порядок ходов, проверяет возможность выполнения хода. Внешнее представление (view) связано с контроллером двунаправленным интерфейсом: контроллер управляет отображением элементов игрового поля, выводит текстовые сообщения, получает от представления события – клик по элементу поля, начало новой игры, завершение и т.п.. Варианты игры:
   * Локальная: одна модель, один контроллер, два представления для игроков
   * Две модели, два контроллера, два представления. Контроллеры поддерживают соединение и передают команды: сделан ход, синхронизация моделей, сброс и начальная установка игры
2. *Шаблон MVC (классический)*. Представление имеет ряд текстовых полей, отображающих параметры модели физического объекта. На каждый параметр представление отдельно подписывается к модели на событие, связанное с его изменением. Интерфейс подписки идентичен для всех параметров. Для каждого параметра создается оригинальный контроллер, обрабатывающий команду изменения значения параметра со стороны представления. Модель описывается набором зависимостей, параметры которой могут быть *входными, выходными (результатами) и выходными с возможностью задания начальных значений (инициализацией).*
3. *Шаблон – прототип.* **Класс таблицы с произвольной структурой столбцов**. Хранимые данные разных типов - целые, вещественные, дата, время, GPS-координаты создаются на основе абстракции данных с функционалом: имя класса, парсинг из сроки и вывод в строку, клонирование, сравнение и сложение с объектом того же типа. Строка таблицы определяется набором объектов-прототипов для столбцов. Сама строка также клонируется. Таблица состоит из вектора строк-имен, строки-прототипа и строк самой таблицы. Функционал: создание таблицы, добавление столбцов, добавление строк, сортировка по столбцу с заданным номером, сложение строк, сохранение и загрузка из файла. Программный и интерфейс и оконное приложение. Тестирование на больших данных - импорт из Excel, генерация тестовых таблиц.
4. *Шаблон – приспособленец*. **Работа с деревом версий текстового файла**. Файл редактируется по словам. Класс словаря содержит хэш-таблицу адаптеров со ссылками на оригинальные слова в виде *ключ – само слово*. Адаптер содержит количество ссылок на слово из всех версий текста. Каждая версия текста – вектор ссылок на адаптеры. При добавлении или вставке слова в версию текста оно ищется в фабрике, при нахождении счетчик в адаптере увеличивается на 1. Иначе добавляется в словарь с новым адаптером. При удалении счетчик ссылок уменьшается. Изменение слова рассматривается как последовательность операций удаления и вставки. Вся структура данных сериализуется в файл. Протестировать шаблон на сказке *Репка*.
5. *Шаблон – композиция для древовидной системы.* **Графический редактор** с группировкой/разгруппировкой элементов, изменением размеров и порядка отображения - перенос на передний/задний план, перетаскиванием объектов, сохранением картинки в файл. Абстрактный класс элемента, группы элементов и конкретных графических объектов - окружность, полигон, строка текста. Ограничивающий прямоугольник, селекция выбора объектов по точке и прямоугольнику.
6. *Шаблон Command.* Группа команд обработки объекта с общим интерфейсом *Do/ReDo/UnDo*. Производный класс запоминает параметры, необходимые для выполнения прямой и обратной команды. Класс – менеджер команд поддерживает очередь команд ограниченной длины, текущий обрабатываемый объект, методы *Do/ReDo/UnDo*, выбирая их из очереди и вызывая соответствующие методы в объектах-командах. **Графический редактор** набором команд редактирования графических объектов - создать объект, переместить, изменить размер, удалить, переместить на передний и задний план.
7. *Шаблон – пул потоков*. Класс потока представляет собой поток, запускаемый при создании объекта. Содержит ссылку на исполняемый код через *Runnable*, код завершения и ссылку на менеджер пула. Код потока содержит цикл, в котором засыпает, после пробуждения выполняет исполняемый код и уведомляет менеджер о своем завершении и засыпает. Менеджер потоков содержит вектор таких потоков. При обращении к менеджеру с запросом, содержащим код исполнения и код завершения, выбирается поток из пула, заполняется данными и пробуждается. Если свободного потока нет, то запрос ставится в очередь. Провести сравнительное тестирование для потока запросов с пулом и при запуске потоков обычным образом.
8. *Шаблон – прокси (фильтр).* Класс с интерфейсом текстового потока при конструировании делегируется к однотипному объекту источнику и **отфильтровывает набор слов,** передаваемый при конструировании, используется внутренняя очередь символов для отложенного распознавания. Протестировать на цепочке фильтров для разных наборов слов.
9. *Шаблон – сессия* для соединения на сокетах в виде библиотеки для клиента/сервера. Клиент и сервер используют синхронный обмен *запрос-ответ*. При первоначальном установлении соединения клиент получает уникальный идентификатор, сервер создает дескриптор соединения. Клиент нумерует передаваемые сообщения, сервер сохраняет номер и ответ на последнее переданное сообщение. Сервер периодически или после каждого изменения сохраняет дескрипторы в файл. Клиент также запоминает в файле идентификатор сессии, номер и само последнее переданное сообщение. Обеспечить сохранность последовательности (отсутствие пропадания и дублирования) сообщений при перезагрузке клиента и сервера. Реализовать модель **банкомата и платежной системы.**
10. *Модель параллельных запросов к серверу от группы потоков.* Множество потоков может посылать независимые запросы к серверу через единственное соединение. Запрос ставится в очередь, нумеруется, клиентский процесс засыпает. Поток передачи от клиента выбирает сообщения из очереди и передает в соединение. Поток приема на сервере, приняв очередное сообщение, запускает поток исполнения запроса, по завершении которого в очередь ответов ставится ответ, в котором сохраняется порядковый номер запроса. Поток передачи ответов на сервере выбирает сообщения из очереди и передает в соединение. Поток приема ответов определяет по номеру в сообщении поток, передавший запрос и пробуждает его. Ответное сообщение находится в запросе, переданном потоком, и выводится им. Промоделировать **группу потоков, посылающих случайные слова, которые сервер переворачивает со случайной задержкой**.
11. *Шаблон – кэш объектов*. Разработать класс - кэш объектов с возможностью изменения размера кэша, сбора статистики и применения различных стратегий вытеснения (FIFO, LRU, RAND). Использовать для кэширования слов текстового файла при последовательном чтении. Кэш – хеш-таблица с ключом-словом и необходимыми параметрами для моделирования, например, номер последнего обращения для LRU. При чтении очередного слова проверяется его наличие в кэше. При отсутствии производится замещение. Для выбранного файла строится зависимость доли попадания в кэш в зависимости от размера кэша и способа вытеснения. Сравнить результаты для файлов с разным содержимым - художественное произведение, технический текст.
12. *Шаблон – итератор.* Операции над итератором: установка на первый, последний, следующий, предыдущий, по логическому номеру, извлечение через итератор, вставка на позицию итератора, замещение, удаление. Структура данных: двухуровневый массив ссылок (двумерный массив), двоичное дерево, дерево с данными в конечных вершинах, список с массивом ссылок [3-2]. Использование нескольких итераторов. ***Замечание по теме****:* для корректной реализации операции удаления в основном классе использовать вектор созданных итераторов. При удалении одним из них элемента остальные, которые на него ссылаются, генерируют исключение при очередном обращении. Рассмотреть другие варианты корректного разделения.

Разработка классов, использующих внутренние потоки для промежуточной буферизации данных. Для моделирования прикладных процессов использовать потоки, которые засыпают на случайный момент времени, после чего читают/записывают очередную порцию данных постоянного или случайного размера. Предусмотреть сбор статистики в классах буферизации – средний объем данных в буфере.

1. Класс буферизованного ввода в реальном времени. При конструировании получает параметр – физический поток данных с интерфейсом *InputStream*. Использует внутренний циклический буфер или односвязный список блоков, содержащих массив байтов фиксированной размерности (буферный пул). Создает поток, который читает байты из входного потока и записывает в циклический буфер. При заполнении циклического буфера засыпает. Метод чтения извлекает из циклического буфера очередной байт, возвращает -1 при окончании данных в потоке-источнике, блокируется при отсутствии данных в циклическом буфере.
2. Класс – *PipedStream* с циклическим буфером данных. Класс *PipedOutputStream* имеет циклический буфер, в который пишет поток байтов, блокируя текущий поток при заполнении буфера. Класс *PipeInputStream* получает ссылку на *PipedOutputStream* и при чтении данных либо блокируется при их отсутствии, либо извлекает данные, деблокируя поток записи.
3. Класс отложенной записи. При открытии файла классом с присоединенным интерфейсом *OutputStream* создается *ByteOutputStream*, в который пишутся данные потока. При закрытии объект, содержащий имя файл и байтный массив ставится в очередь, из которой фоновый поток их извлекает и пишет файлы. Сравнить среднее время записи в обычный и *отложенный файл.*