Само задание

Описание задания

Разработать программу на языке JAVA, использующую графический пользовательский интерфейс для управления моделированием динамического процесса.

Параметры моделирования должны задаваться до включения моделирования процесса. После включения элементы интерфейса, отвечающие за параметры моделирования, необходимо заблокировать (при окончании снова разблокировать).

Предусмотреть возможность ускорения и замедления времени моделирования для изучения медленно- или быстротекущих процессов.

Примечание

При разработке моделей разрешается использовать эвристические формулы. Использование точных физических формул не требуется.

Вариант задания

Движения автомобиля вверх по наклонной плоскости

Автомобиль движется вперед по наклонной плоскости с ускорением. Нажимая педаль газа автомобиль ускоряется. Отжимая педаль газа происходит ускоренное движение назад (появляется отрицательное ускорение, направленное в обратную сторону движения). При достижении конца маршрута автомобиль останавливается и моделирование завершается.

Задаваемые параметры: длина маршрута, наклон плоскости, мощность двигателя, коэффициент трения скольжения.

Элементы управления моделью: педаль газа.

Визуализируемые значения: скорость движения, пройденный маршрут, время с момента начала движения.

Пример выполнения задания

Разработать модель чайника, отражающую процесс нагревания и остывания воды в реальном времени.

Основные исходные данные

Чайник задан в виде цилиндрического сосуда (даны его радиус r и высота h).

Комнатная температура T_0 считается равной 20° С.

Даны мощность электрического нагревателя P и его КПД η.

Потерями тепла при нагревании воды в чайнике можно пренебречь.

При достижении водой температуры 100° С чайник автоматически отключается.

Пользователь с помощью графического интерфейса может регулировать: радиус r и высоту h цилиндра, мощность P и КПД η электрического нагревателя, объем воды $V_в$ в чайнике. Коэффициент теплоотдачи α принять равным 200 Вт/(м^2·К).

Требования к программе

Параметры моделирования должны задаваться до включения моделирования процесса. После включения элементы интерфейса, отвечающие за параметры моделирования, необходимо заблокировать (при окончании снова разблокировать).

Предусмотреть возможность ускорения и замедления времени моделирования для изучения медленно- или быстротекущих процессов.

Разработка класса физической модели чайника

Для разработки модели чайника в JAVA как нельзя лучше подходит объектноориентированный подход, позволяющий инкапсулировать весь процесс управления объектом (чайником) внутри класса.

На первом шаге проектирования класса определяют свойства класса и методы управления свойствами. В языке JAVA предусмотрено три модификатора доступа к свойствам и методам: public (свойства и методы доступны извне класса, т.е. их можно изменять или вызывать и любой части программы, где объявлен экземпляр класса), private (свойства и методы доступны только внутри класса) и protected (свойства и методы доступны внутри класса и его потомков). По умолчанию (если модификатор не указан) всем членам класса задается свойство public. Кроме того, в языке JAVA с помощью ключевого слова final можно задавать свойства-константы, значения которым присваиваются только один раз и в дальнейшем их нельзя изменять.

После анализа исходных данных в задании можно сформировать список свойств, необходимых для создания модели чайника (таблица 1).

Таблица 1. Свойства класса модели чайника

Название переменной	Тип	Модификатор доступа	Описание
Свойства физической модели чайника			
height	double	private	Высота цилиндрического сосуда

radius	double	private	Радиус цилиндрического сосуда
power	double	private	Мощность электрического нагревателя
efficiency	double	private	КПД электрического нагревателя
fillingPercentage	double	private	Процент заполнения чайника водой
Свойства для организации процесса моделирования			
modelingOn	boolean	private	Флаг режима моделирования
teapotOn	boolean	private	Флаг включения/выключения электрического нагревателя
timerValue	double	private	Время с момента переключения режима чайника
lastTemperature	double	private	Температура воды в момент переключения режима чайника
Свойства, содержащие физические константы			
THERMAL_ CAPACITANCE_ OF_WATER	double	final private	Удельная теплоемкость воды
ROOM_ TEMPERATURE	double	final private	Комнатная температура

Как видно из таблицы, свойства разбиты на три группы. Первая группа "свойства чайника" отражает основные параметры физической модели. Эти свойства можно сделать открытыми (доступными извне). В этом случае перед процессом моделирования нужно будет явно задавать их значения в программе. Однако в таком случае ничто не помешает изменить их в процессе моделирования нагрева воды и это может привести к непредсказуемым последствиям. Поэтому следует сделать их частными (private) и разработать методы, ограничивающие к ним доступ.

Вторая группа свойств предназначена для организации процесса моделирования. Переменная modelingOn содержит флаг, обозначающий запущен ли процесс моделирования (если он запущен, то свойства физической модели изменять нельзя - это основное назначение флага). Переменная teapotOn содержит флаг состояния работы электрического нагревателя. Если он включен, то нагрев будет происходить по одному закону, если выключен, то по другому. Переменная timerValue предназначена для хранения времени, прошедшего с момента переключения режима электронагревателя (отсчет начинается с 0). Переменная lastTemperature содержит опорную температуру воды, которая была при переключении режима электронагревателя.

В последней группе свойств содержатся физические константы, использующиеся в расчетных формулах.

Описание свойств класса модели чайника

```
public class
TeapotModel {


  // СВОЙСТВА ЧАЙНИКА
  // свойства физической модели   private double height;            //
высота цилиндрического сосуда, м   private double radius;           //
```

```
радиус цилиндрического сосуда, м   private double power;              //
мощность эл. нагревателя, Вт   private double efficiency;        // КПД
эл. нагревателя
  private double fillingPercentage; // процент заполнения чайника водой
  // свойства для организации процесса моделирования   private boolean modelingOn;       //
флаг режима моделирования   private boolean teapotOn;          // флаг включения/выключения
чайника   private double timerValue;        // время с момента переключения режима работы
чайника, с   private double lastTemperature;   // опорная температура воды, град. Цельсия
  // физические константы
  final private double THERMAL_CAPACITANCE_OF_WATER = 4.2e3; // удел. теплоемкость воды
final private double ROOM_TEMPERATURE = 20;                // комнатная температура
}
```

Следующий шаг работы над классом - создание конструктора, в котором будут инициализироваться описанные свойства значениями по умолчанию.

```
TeapotModel() {
    // заполнение свойств значениями по умолчанию     this.height
= 0.16;     this.radius            = 0.05;     this.power
= 2000;     this.efficiency        = 0.9;
this.fillingPercentage = 80;     this.modelingOn
= false;     this.teapotOn          = false;
this.timerValue        = 0;     this.lastTemperature   =
this.ROOM_TEMPERATURE; }
```

Третий шаг - организация доступа к свойствам класса. Доступ к физическим свойствам модели удобно осуществлять с помощью методов установки (set) и возвращения (get) значений. Процедура возвращения значений просто возвращает значение свойства. Процедура установки перед инициализацией свойства проверяет, не включен ли режим моделирования и не выходит ли задаваемое значение за допустимый диапазон. Таким образом, осуществляется контроль целостности данных и снижается риск неадекватного поведения модели (ошибочной работы класса). Пример методов доступа к свойству КПД электронагревателя

```
public double getEfficiency() {
return this.efficiency;
}


  public void setEfficiency(double e)
{
if (!this.modelingOn) {       if      (e
<= 0.1) e = 0.1;   else if (e >=
1.0) e = 1.0;         this.efficiency
= e;
  }
}
```

Четвертый шаг - организация моделирования, т.е. разработка метода, рассчитывающего текущее значение температуры воды в чайнике. Моделирование включает в себя два процесса: нагревание и остывание воды. Для обоих процессов нужно вывести соответствующие физические формулы. Выбор той или иной формулы зависит от состояния электронагревателя (флага teapotOn). Другими словами, цель данного этапа - вывод зависимостей температуры воды $T_в$ от начального значения температуры T_0 (опорного значения температуры lastTemperature) и времени t, прошедшего с переключения режима электронагревателя (timerValue)

$T_в = f\,t\,T(\ ,_0)$. Для расчета формулы нагревания воды

необходимо воспользоваться соображениями из термодинамики. Количество теплоты Q, необходимое для нагревания воды в сосуде, определяется из уравнения

$Q = \Delta cm\,T$, (1) где c - удельная теплоемкость воды, Дж/(кг·К); m - масса воды, кг; $\Delta T = T_в - T_0$ - разница между начальной и конечной температурой. Количество теплоты, выделяемое электронагревателем

$Q = \eta Pt$, (2) где η - КПД нагревателя; P - мощность, Вт; t - время работы, с. Сопоставляя формулы (1) и (2) легко получить искомое уравнение

$$T\,t\,T_в(\,,_0) = \frac{\eta\ P}{}\ t + T_0\,.\,cm$$

Считая, что масса 1 кг воды равна 1 литру, массу воды m можно найти, определив объем цилиндра

$$m = V \cdot 1000 = \pi r\,h^2 \cdot 1000.$$

Формулу остывания воды можно вывести из Закона охлаждения Ньютона dQ

$$\frac{}{} = \alpha A\,T(\,_s - T)\,, \qquad dt$$

где α - коэффициент теплоотдачи, Вт/(м^2·К); A - площадь поверхности тела, через которую передается тепло; T - температура тела; T_s - температура окружающей среды. Поскольку $Q = CT$, где C - полная теплоемкость тела, то уравнение можно записать как $dT\ \alpha A$

$$\frac{}{} \quad \frac{}{} = \frac{}{C}(T_s - T) = k\,T(\,_s - T)\,.\,dt$$

Решение этого дифференциального уравнения

$$T\,t\,T_в(\,,_0) = + - T_s \qquad (T_0\,T\,e_s)\ _{-kt},$$

где T_0 - значение температуры воды в момент начала остывания.

Определить значение коэффициента k можно вычислив значения полной теплоемкости воды

$$C = cm$$

и площади поверхности цилиндра

$$A = 2\pi\ \pi rh + 2\ r^2.$$

Учитывая, что по условию α= 200 Вт/(м^2·К), конечная формула примет вид

$$k = 200 \cdot \frac{2\pi\,\pi r h + 2\,r^2}{cm}.$$

Используя обведенные формулы нетрудно составить алгоритм, рассчитывающий температуру воды

```
// МЕТОД, ВЫЧИСЛЯЮЩИЙ ТЕМПЕРАТУРУ ВОДЫ  public
double getWaterTemperature() {
  // если моделирование не включено - возвращем значение последней температуры
if (!this.modelingOn) return this.lastTemperature;
  // определение массы воды   double waterMass = Math.PI * this.radius *
this.radius * this.height
                * this.fillingPercentage/100 * 1000;
  // определение температуры воды при нагреве   if
(this.teapotOn) {
          return this.efficiency * this.power * this.timerValue
                        / (this.THERMAL_CAPACITANCE_OF_WATER * waterMass)
                        + this.lastTemperature;
  // определение температуры воды при остывании    }
else {     double k = (2 * Math.PI * this.radius *
this.height
                        + 2 * Math.PI * this.radius * this.radius)
                        / (this.THERMAL_CAPACITANCE_OF_WATER * waterMass) * 200;
return (this.lastTemperature - this.ROOM_TEMPERATURE)
                        * Math.exp(-k * this.timerValue)
                        + this.ROOM_TEMPERATURE;
  }
}
```

Последний шаг - организация управления процессом моделирования. В нее входит создание методов запуска/останова процесса моделирования, управления процессом моделирования (включение/выключение электронагревателя) и обновление внутреннего таймера.

Метод запуска процесса моделирования включает флаг modelingOn, сбрасывает таймер, опорную температуру воды и флаг состояния электронагревателя. Метод останова моделирования просто сбрасывает флаг modelingOn. Методы включения и выключения электронагревателя сначала сохраняют опорную температуру, затем устанавливают флаг teapotOn в соответствующее положение и сбрасывают внутренний таймер.

```
// МЕТОДЫ УПРАВЛЕНИЯ ПРОЦЕССОМ МОДЕЛИРОВАНИЯ
// запуск моделирования
public void start() {   this.modelingOn      =
true;   this.teapotOn        = false;
this.timerValue      = 0;   this.lastTemperature
= this.ROOM_TEMPERATURE;  }


// останов моделирования  public
void stop() {
this.modelingOn = false;
}
// включение электронагревателя public void turnOn()
{   this.lastTemperature =
this.getWaterTemperature();   this.teapotOn        = true;
this.timerValue      = 0;
}



// выключение электронагревателя public void
turnOff() {
```

```
  this.lastTemperature = this.getWaterTemperature();
this.teapotOn        = false;   this.timerValue
= 0;  }
```

Метод обновления внутреннего таймера должен добавлять количество прошедших секунд в переменную timerValue. Кроме того, с целью реализации автоматического отключения чайника при достижении температуры кипения воды, в этом методе необходимо предусмотреть процедуру обнуления таймера, сохранения опорного значения температуры и отключения электронагревателя.

```
// обновление значения внутреннего таймера  public
void refreshTimer(double elapsedTime) {
  // обновление значения таймера
this.timerValue += elapsedTime;
  // автоматическое отключение чайника при достижении температуры 100 град
if (this.getWaterTemperature() >= 100) {
this.teapotOn        = false;     this.timerValue
= 0;     this.lastTemperature = 100;
  }
}
```

Внешней программе, использующей класс модели чайника, необходимо контролировать момент автоматического отключения чайника. Для этого она должна получать доступ к значению флага работы электронагревателя teapotOn.

```
// возвращение статуса чайника (вкл или выкл?) public
boolean getTeapotStatus() {   return this.teapotOn; }
```

Проектирование графического интерфейса

После разработки модели необходимо реализовать управление ее параметрами и процессом моделирования с помощью графического интерфейса. Графический интерфейс разрабатывается в среде NetBeans с использованием кроссплатформенной библиотеки Swing. Элементы управления можно разделить на четыре группы: элементы управления моделированием (запуск, останов, ускорение времени, значение таймера времени), элементы настройки параметров модели (установка значений высоты и радиуса цилиндра, мощности и КПД электронагревателя, процента заполнения чайника водой), элементы управления чайником (кнопка вкл/выкл электронагревателя) и элементы визуализации процесса нагревания (шкала температуры воды, значение температуры). Используя графические элементы JPanel, JLabel, JButton, JSpinner можно составить интерфейс, представленный на рис. 1. Названия и описания некоторых элементов представлены в табл. 2.

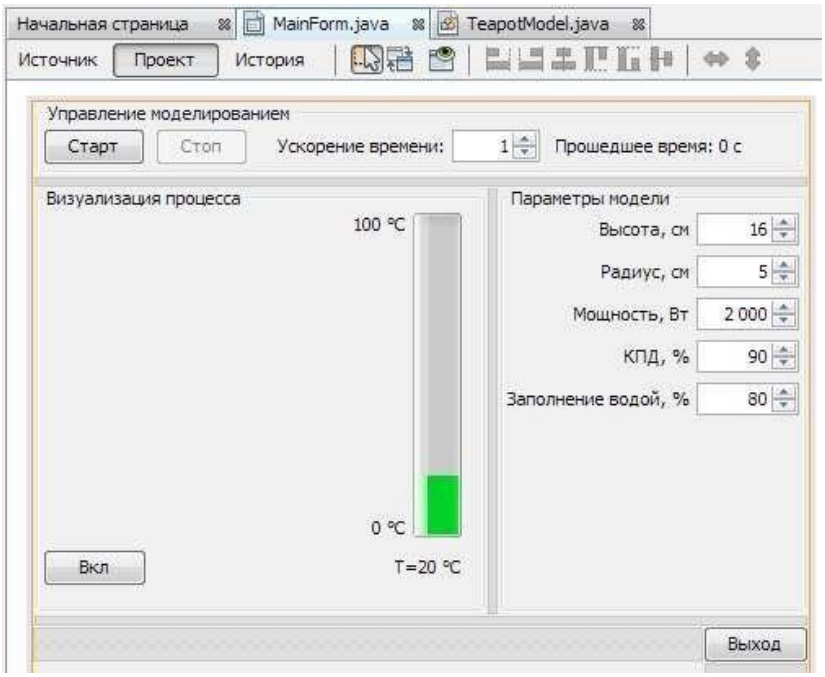

Рис. 1. Разработка интерфейса управления моделью чайника

Таблица №2. Названия и описание некоторых графических элементов

Название	**Тип**	**Описание**
butStart	JButton	Запуск процесса моделирования. При нажатии кнопки блокируются кнопки управления параметрами модели (height, radius, power, efficiency, fillingPercentage) и активируется процесс моделирования.
		Также блокируется сама эта кнопка и активируется кнопка останова stop.
butStop	JButton	Кнопка останова моделирования. Останавливает процесс моделирования, блокирует саму себя и активирует кнопку запуска start.
spnTimeMult	JSpinner	Меняет шаг моделирования в секундах.
lblElapsedTime	JLabel	Таймер, показывающий количество прошедших секунд с начала моделирования.
spnHeight	JSpinner	Устанавливает высоту цилиндрического сосуда чайника
spnRadius	JSpinner	Устанавливает радиус цилиндрического сосуда чайника
spnPower	JSpinner	Устанавливает мощность электронагревателя
spnEfficiency	JSpinner	Устанавливает КПД электронагревателя
spnFillingPercentage	JSpinner	Устанавливает процент заполнения чайника водой
tglTeapotOnOff	JToggleButton	Включает и выключает электронагреватель в чайнике
pbarTemperature	JProgressBar	Показывает текущую температуру воды в чайнике
butExit	JButton	Выход из программы

После размещения всех графических элементов начинается программирование функций. В первую очередь, в класс формы необходимо добавить переменные (т.е. создать свойства класса формы): экземпляр разработанного класса TeapotModel и счетчик времени timeValue (не путать со счетчиком внутри класса чайника – timerValue!).

```
// Создание экземпляра класса "Модель чайника"
private TeapotModel myTeapot = new TeapotModel();
// Накопитель времени моделирования private double
timeValue = 0;
```

Кроме элементов интерфейса, таких как кнопки, ползунки, элементы ввода, в событийноуправляемой модели программирования важное место занимает таймер. Таймер позволяет запланировать выполнение какой-либо важной задачи в будущем или выполнять ее с периодичностью через определенные промежутки времени. Для управления динамическим процессом в реальном времени необходимо, чтобы программа с помощью таймера через определенные промежутки времени вычисляла текущее значение температуры воды в чайнике и показывала его пользователю.

В графическом пакете Swing таймер реализован в специальном классе Timer. При вызове конструктора класса ему необходимо передать интервал времени в миллисекундах и ссылку на объект ActionListener, который отвечает за обработку события (вызывает метод actionPerformed()). Запуск и останов таймера осуществляется с помощью вызова методов класса start() и stop(). Пример создания экземпляра класса Timer

```
// Создание и настройка таймера javax.swing.Timer timer = new
javax.swing.Timer(1000, new ActionListener() {   public void
actionPerformed(ActionEvent e) {     // обработка события
}
});
```

Обработчик события таймера увеличивает значение программного счетчика времени timeValue и значение счетчика времени timerValue внутри класса модели чайника TeapotModel (вызывать метод refreshTimer()). Значение, на которое нужно увеличить счетчик времени берется из элемента spnTimeMult (типа JSpinner). Далее вычисляется текущая температура воды в чайнике (вызывается соответствующий метод класса TeapotModel) и визуализируется с помощью элементов pbarTemperature и lblTemperature. Последнее действие в обработчике – проверка автоматического отключения чайника. Если отключение произошло, то нужно изменить состояние кнопки tglTeapotOnOff (включено или выключено).

```
// Создание и настройка таймера javax.swing.Timer timer = new
javax.swing.Timer(1000, new ActionListener() {   public void
actionPerformed(ActionEvent e) {      // вычисление времени
шага
    double time = Double.parseDouble(spnTimeMult.getValue().toString());
    // инкремент времени моделирования
timeValue += time;
    // обновление таймера в модели чайника
myTeapot.refreshTimer(time);
    // вычисление текущей температуры воды
    int tmp = (int) myTeapot.getWaterTemperature();
// установка значения ползунка и подписи
pbarTemperature.setValue(tmp);
lblTemperature.setText("T=" + tmp + " °C");
// индикация прошедшего времени
    lblElapsedTime.setText("Прошедшее время: " + timeValue + " c");
    // проверка на автоматическое выключение чайника     if
(!myTeapot.getTeapotStatus()) {
tglTeapotOnOff.setSelected(false);
```

```
    } else {       tglTeapotOnOff.setSelected(true);
    }
  }
});
```

Последний шаг – добавление функциональности к кнопкам. Необходимо запрограммировать кнопку старта и останова моделирования (butStart, butStop). При нажатии кнопки старта butStart должна заблокироваться сама кнопка, разблокироваться кнопка останова butStop, заблокироваться элементы управления параметрами модели. Затем необходимо установить параметры модели чайника (свойства height, radius, power, efficiency, fillingPercentage), запустить моделирование (вызвать метод start() экземпляра класса myTeapot()) и системный таймер (вызвать метод stop() объекта timer). Нажатие кнопки butStop приводит к обратным действиям. Нажатие кнопки tglTeapotOnOff просто переключает режим работы электронагревателя в чайнике.

```
        private void
butStartActionPerformed(java.awt.event.ActionEvent evt) {
  // смена блокировки элементов управления
butStart.setEnabled(false);   butStop.setEnabled(true);
spnHeight.setEnabled(false);
spnRadius.setEnabled(false);
spnPower.setEnabled(false);
spnEfficiency.setEnabled(false);
spnFillingPercentage.setEnabled(false);   // установка свойств
модели чайника
myTeapot.setHeight(Double.parseDouble(spnHeight.getValue
().toString()) / 100);
myTeapot.setRadius(Double.parseDouble(spnRadius.getValue
().toString()) / 100);
myTeapot.setPower(Double.parseDouble(spnPower.getValue()
.toString()));   myTeapot.setEfficiency(Double.parseDouble(spnEfficiency.
getValue().toString()) / 100);
myTeapot.setFillingPercentage(Double.parseDouble(spnFill
ingPercentage.getValue().toString()));
  // запуск моделирования
myTeapot.start();   //
запуск таймера
timer.start();
  // обнуление счетчика времени моделирования   timeValue
= 0;
}
 private void butStopActionPerformed(java.awt.event.ActionEvent evt) {
  // смена блокировки кнопок   butStart.setEnabled(true);
butStop.setEnabled(false);   spnHeight.setEnabled(true);
spnRadius.setEnabled(true);   spnPower.setEnabled(true);
spnEfficiency.setEnabled(true);   spnFillingPercentage.setEnabled(true);
  // останов моделирования
myTeapot.stop();   //
останов таймера
timer.stop();
}  private void tglTeapotOnOffActionPerformed(java.awt.event.ActionEvent
evt) {                                        // влючение или выключение
чайника   if (!myTeapot.getTeapotStatus()) {     myTeapot.turnOn();    } else
{     myTeapot.turnOff();
  } }
```

Визуализация динамического процесса с помощью пользовательской графики

В среде JAVA предусмотрена возможность отрисовки произвольной графики с помощью класса Graphics. В этом классе реализованы методы компьютерной графики для отрисовки примитивов: линий, прямоугольников, овалов, надписей и т.п. [4] (табл. 3).

Таблица 3. Описание некоторых методов класса Graphics

Метод	Описание
`void clearRect(int x, int y, int width, int height)`	Очищает указанный прямоугольник, заполняя его цветом фона текущей поверхности рисунка.
`void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)`	Чертит рамку круга или эллиптической дуги, вписанной в указанную прямоугольную область.
`void drawLine(int x1, int y1, int x2, int y2)`	Чертит линию, используя текущий цвет, между точками (x1, y1) и (x2, y2).
`void drawOval(int x, int y, int width, int height)`	Чертит рамку овала, вписанного в указанную прямоугольную область.
`void drawPolygon(int[] xPoints, int[] yPoints, int nPoints)`	Чертит рамку многоугольника, определенного массивами координат xPoints и yPoints.
`void drawRect(int x, int y, int width, int height)`	Чертит рамку прямоугольника.
`void drawRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)`	Чертит рамку прямоугольника со скругленными углами.
`void drawString(String str, int x, int y)`	Чертит текст, указанный в строке str, используя текущий шрифт графического контекста и цвет.
`void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle)`	Заполняет круг или эллиптическую дугу, вписанную в указанную прямоугольную область.
`void fillOval(int x, int y, int width, int height)`	Заполняет овал, вписанный в указанную прямоугольную область.
`void fillPolygon(int[] xPoints, int[] yPoints, int nPoints)`	Заполняет замкнутый многоугольник, определенный массивами координат xPoints и yPoints.
`void fillRect(int x, int y, int width, int height)`	Заполняет указанную прямоугольную область.
`void fillRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)`	Заполняет указанную прямоугольную область со скругленными углами.
`Color getColor()`	Возвращает текущий цвет графического контекста.
`void setColor(Color c)`	Задает текущий цвет графического контекста.

Удобной особенностью графического интерфейса Swing является то, что любой графический элемент содержит контекст рисования (класс Graphics). Поэтому для отрисовки произвольных фигур достаточно всего лишь получить ссылку на контекст рисования с помощью метода getGraphics() и использовать его методы.

Пример реализации простейших графических операций показан на рис. 2. В новом проекте добавлены 2 кнопки JButton и панель JPanel, на которой и происходит рисование.

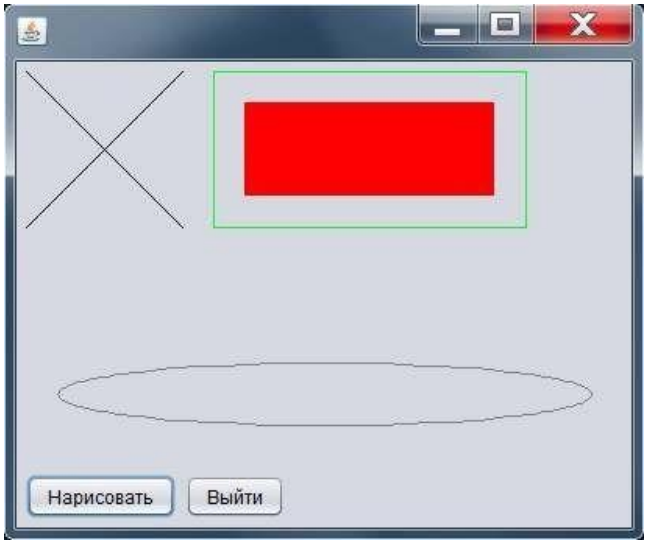

Рис. 2. Пример использования класса Graphics

Исходный код программы (обработка события нажатия кнопки "Нарисовать")

```
private void butDrawActionPerformed(java.awt.event.ActionEvent evt) {
  // вызов ссылки на объект класса Graphics из панели JPanel
java.awt.Graphics g = pnlPanel.getGraphics();
  // вычисление ширины и высоты панели (контекста рисования)   int
w = pnlPanel.getWidth();   int h = pnlPanel.getHeight();
  // прорисовка двух линий
  g.drawLine(0, 0, 100, 100);
  g.drawLine(0, 100, 100, 0);
  // проприсовка каркаса прямоугольника зеленого цвета
g.setColor(new java.awt.Color(0, 255, 0));
  g.drawRect(120, 0, 200, 100);
  // прорисовка заполненного прямоугольника красного цвета
g.setColor(new java.awt.Color(255, 0, 0));
  g.fillRect(140, 20, 160, 60);
  // прорисовка серого овала внизу панели, растянуго вдоль всей ее ширины
g.setColor(Color.gray);
  g.drawOval(20, h - 60, w - 40, 40); }
```

Координаты графического контекста отсчитываются от левого верхнего угла графического элемента (в данном случае панели JPanel). Первая инструкция рисования - прорисовка нисходящей линии с координатами (x: 0; y: 0) - (x: 100; y: 100). Вторая инструкция - прорисовка перпендикулярной линии перпендикулярно предыдущей. Обе линии нарисованы черным цветом (цвет по умолчанию). Чтобы установить цвет рисования необходимо вызвать метод setColor() и передать ему экземпляр класса Color. Цвет можно задавать в системе RGB (при этом необходимо вызвать конструктор класса со значениями трех каналов - reg, green и blue) или использовать заранее предустановленный (указывается статическое свойство с названием цвета). После линий рисуется зеленая рамка прямоугольника, прямоугольник, залитый красным цветом и овал серого цвета. Причем, благодаря использованию относительных координат (ширины и высоты панели), овал всегда расположен в нижней части панели и растягивается на полную ширину при масштабировании окна.

Для большей наглядности протекания динамического процесса его лучше визуализировать. На примере чайника это можно сделать как показано на рис. 3. В данном примере стенки чайника показаны в виде каркаса прямоугольника, внутри которого прямоугольник, заполненный цветом, обозначающим температуру воды (синий - комнатная температура, оранжевый - средняя, красный

- температура кипения). Соотношение сторон зависит от соотношения высоты и радиуса цилиндрической емкости.

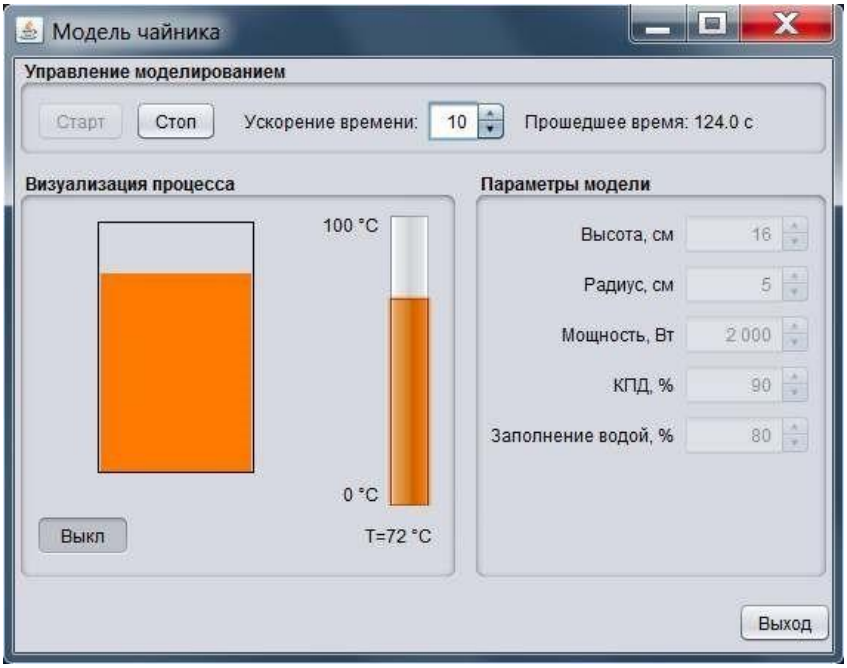

Рис. 3. Визуализация нагрева воды в чайнике

Следуя принципу инкапсуляции все процедуры прорисовки нужно перенести в отдельный метод класса модели чайника TeapotModel. При вызове метода необходимо передавать ссылку на класс графического контекста Graphics и ширину и высоту области рисования.

```
// прорисовка чайника в контексте рисования  public void
drawTeapot(java.awt.Graphics g, int w, int h) {
  // очистка контекста рисования
g.clearRect(0, 0, w, h);
  // вычисление соотношения сторон (прямоугольника, обозначающего чайник)
double aspect = this.radius * 2 / this.height;   //
вычисление относительной высоты и ширины чайника
  int rectHeight = h - 1;   int rectWidth  = (int)
(rectHeight * aspect);
  // вычисление координаты x, начиная с которой будет рисоваться
чайник    int left = w/2 - rectWidth/2;   // прорисовка контура чайника
g.setColor(Color.black);
  g.drawRect(left, 0, rectWidth, rectHeight);
  // вычисление высоты и верхней точки прямоугольника, обозначающего воду
int waterHeight = (int) (rectHeight * this.fillingPercentage / 100) - 1;
int waterTop    = rectHeight - waterHeight;
  // нормализованный показатель температуры (число от 0.0 до 1.0)   double
tmp = (this.getWaterTemperature() - this.ROOM_TEMPERATURE)
             / (100 - this.ROOM_TEMPERATURE);   // вычисление
цвета, обозначающего температуру воды
  int colorRed, colorGreen, colorBlue;
  // от 20 до 60 град цвет воды от синего до оранжевого   if
(tmp <= 0.5) {     colorRed   = (int) (255 * (tmp *
2));     colorGreen = (int) (200 * (tmp * 2));
  colorBlue  = (int) (255 * (1 - tmp * 2));
  // от 60 до 100 град цвет воды от оранжевого до красного
  } else {  colorRed   = 255;  colorGreen =
   (int) (200 * (2 - 2 * tmp));  colorBlue  =
   0;
  }
  // создание экземпляра класса цвета
```

```
 Color waterColor = new Color(colorRed, colorGreen, colorBlue);  //
прорисовка прямоугольника, обозначающего воду
 g.setColor(waterColor);
 g.fillRect(left + 2, waterTop, rectWidth - 3, waterHeight); }
```

Разработанный метод может вызываться, например, из обработчика события таймера. Для этого нужно добавить графический элемент (например, панель JPanel с именем pnlTeapot) и вставить следующие строки кода в процедуру actionPerformed().

```
// вызов ссылки на объект Graphics
java.awt.Graphics g = pnlTeapot.getGraphics();
// вычисление ширины и высоты панели
int h = pnlTeapot.getHeight(); int w =
pnlTeapot.getWidth(); // прорисовка
чайника
myTeapot.drawTeapot(g, w, h);
```

При каждом обновлении состояния модели чайника будет прорисовываться соответствующая картинка.

Список рекомендуемой литературы

1. Учебная карта по общим сведениям о разработке на Java https://netbeans.apache.org/kb/docs/java-se_ru.html
2. Шилдт, Г. Java. Полное руководство, 10-е изл.: Пер. с англ. - СПб.: ООО "Альфа книга", 2018. - 1488 с.
3. Блинов, И. Н., Романчик, В. С. Java from EPAM : учеб.-метод. пособие / И. Н. Блинов, В. С. Романчик. — Минск : Четыре четверти, 2020. — 560 с.
4. Class Graphics2D https://docs.oracle.com/en/java/javase/17/docs/api/java.desktop/java/awt/Graphics2D.html