МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ  
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Кафедра вычислительной техники

Отчет по лабораторной работе №4

по дисциплине «Технология программирования»

Тема: «Основы программирования на Java.

Обработка событий. Механизм делегирования событий. Разработка графического интерфейса приложения.»

Вариант №7

Выполнил:

студент группы АВТ-716

Рягузов Андрей

Проверил:

Михайленко Дмитрий Анатольевич

Новосибирск, 2020 г.

#### Практические задания

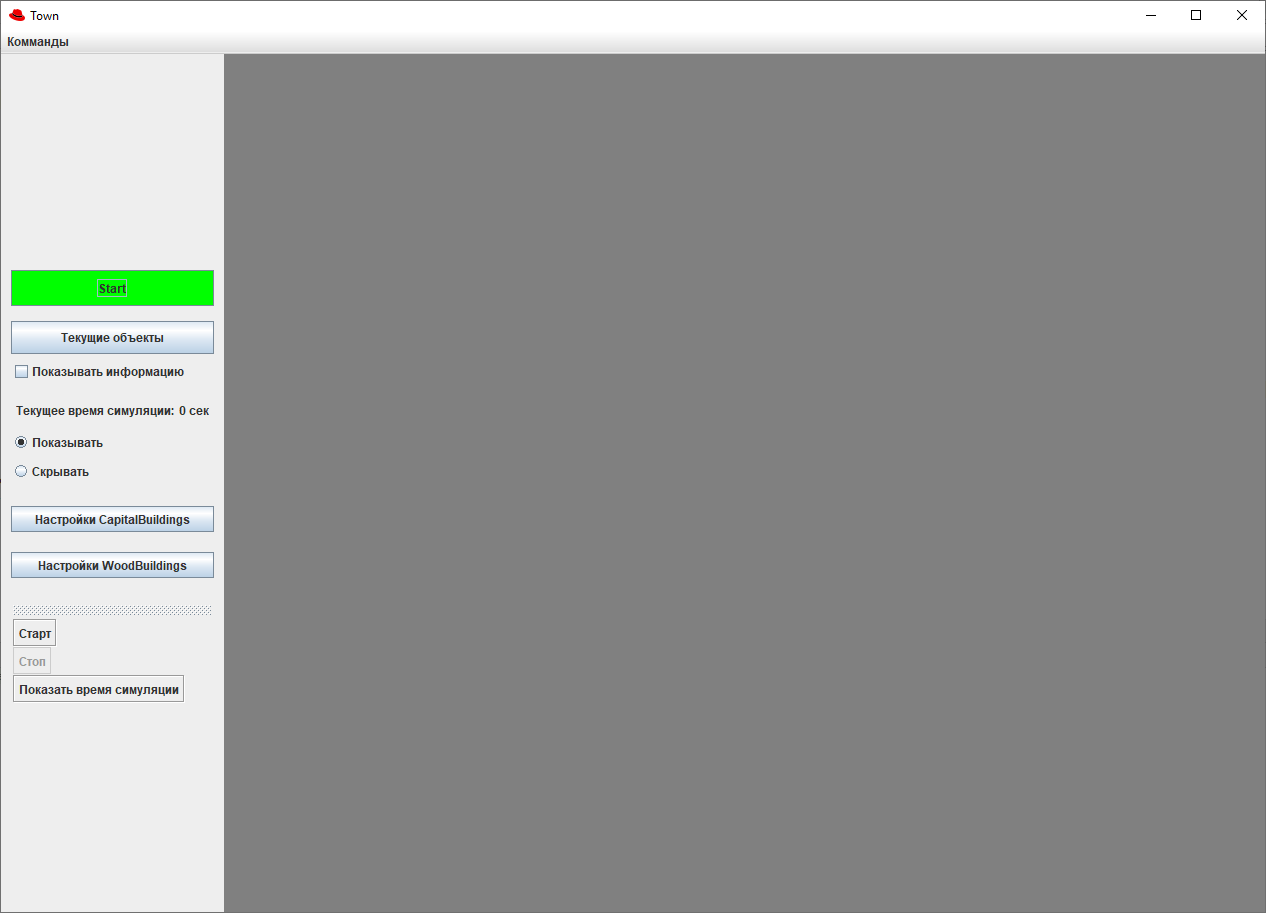
1. Изучить особенности реализации и работы потоков в Java, управлением приоритетами потоков и синхронизацией потоков.
2. Доработать программу, созданную в лабораторной работе № 3:
3. создать абстрактный класс BaseAI, описывающий «интеллектуальное поведение» объектов. Класс должен создавать поток, обеспечивающий движения объектов коллекции;
4. реализовать класс BaseAI для каждого из видов объекта, включив в него поведение, описанное в индивидуальном задании по варианту;
5. синхронизовать работу потоков расчета интеллекта объектов, их рисования и генерации новых объектов. Рисование должно остаться в основном потоке;
6. добавить в панель управления кнопки для остановки и возобновления работы интеллекта каждого вида объектов. Реализовать через управление монитором (методы wait() и notify());
7. добавить в панель управления выпадающие списки для выставления приоритетов каждого из потоков.

#### Задание по варианту

1. Капитальные дома двигаются (в городах будущего и не такое возможно) в левую верхнюю четверть области симуляции (т.е. прямоугольник с верхним-левым углом в точке 0;0, шириной/длиной = (w/2;h/2), где w и h – ширина и длина области симуляции) со скоростью V по прямой. Конечная точка движения – случайная точка в пределах этой области. Если дом сгенерировался сразу в этой области, то он никуда не движется. По прибытии в конечную точку дом больше не движется.
2. Деревянные дома после генерации начинают двигаться в нижнюю правую четверть области симуляции (т.е. прямоугольник с верхним-левым углом в точке w/2;h/2, шириной/длиной = (w/2;h/2), где w и h – ширина и длина области симуляции) со скоростью V по прямой. Конечная точка движения – случайная точка в пределах этой области. Если дом сгенерировался сразу в этой области, то он никуда не движется. По прибытии в конечную точку дом больше не движется.

**Результаты работы программы**

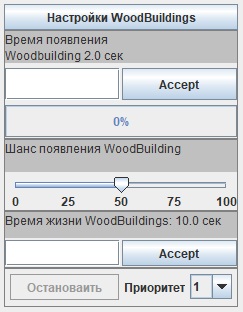
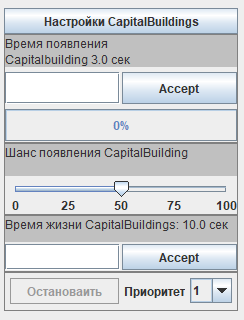
При запуске программы пользователь видит следующее окно:



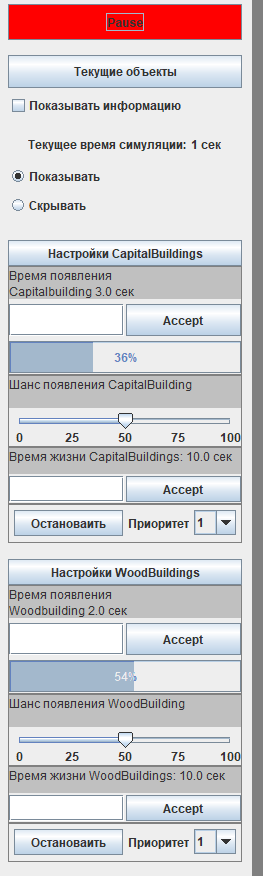
Управление рождением и движением объектов в заданные углы происходит в отдельных потоках. В интерфейс было вынесено управление этими потоками



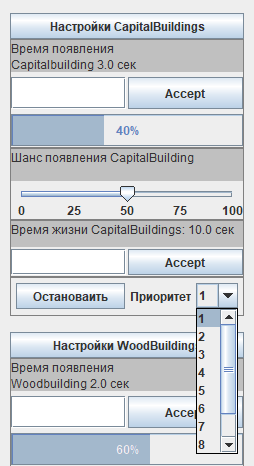
Рождение и движение домов управляется из настроек



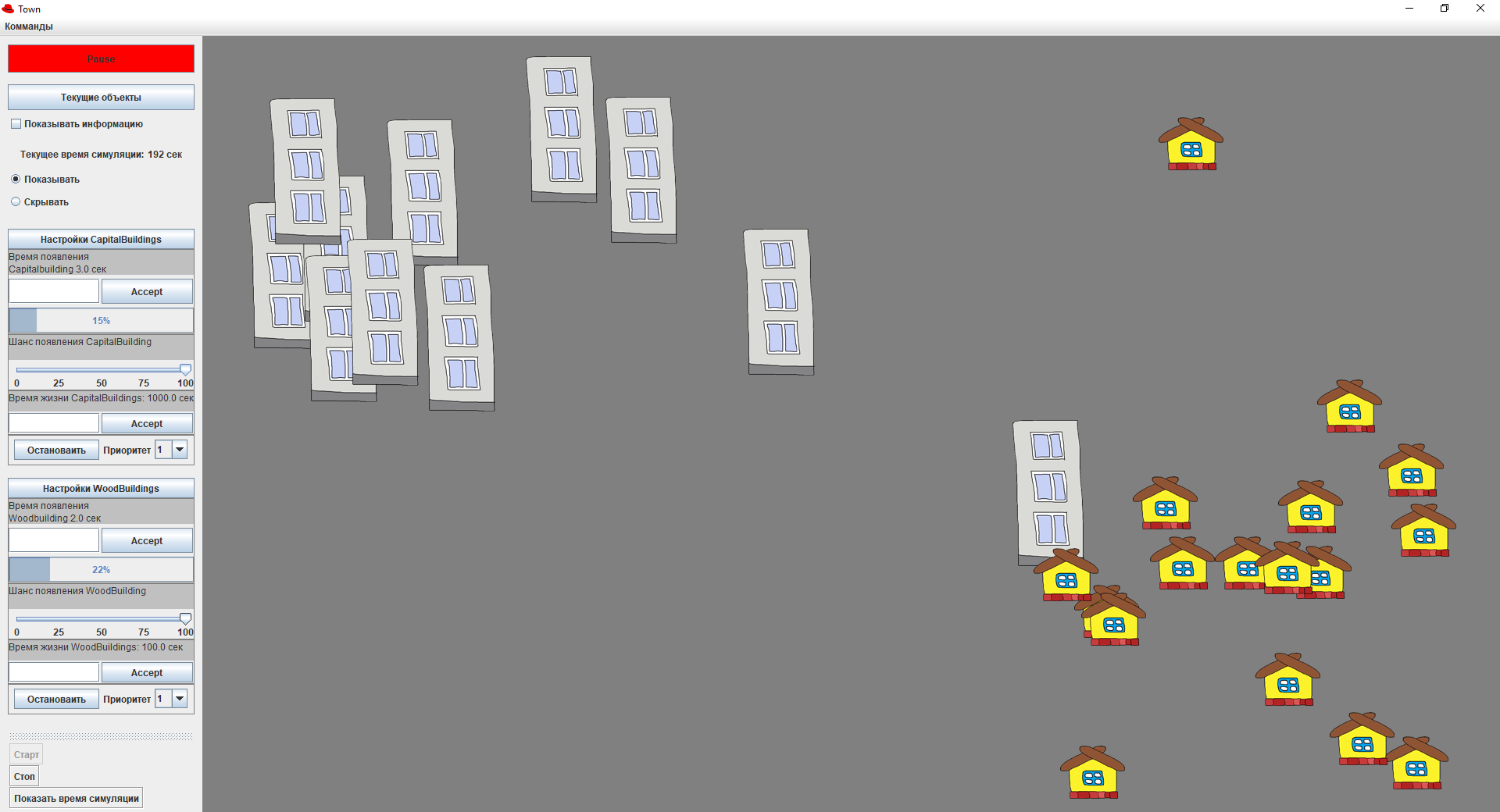
Остановка и возобновление доступно только когда программа запущена



Потокам можно выставить приоритет через комбобоксы от 1 до 10



Дома осуществляют движение в заданные точки в углах экрана, если дом сгенерировался сразу в этой области, то он никуда не движется



**Листинг программы**

***Main.java***

public class Main{

public static void main(String[] args){

App app = new App();

app.setVisible(true);

app.run();

}

}

***App.java***

**import** Buildings.BuildingFactory;  
**import** Buildings.CapitalAI;  
**import** Buildings.WoodAI;  
**import** SubClasses.Timer;  
  
**import** javax.swing.\*;  
**import** java.awt.\*;  
**import** java.awt.event.ActionEvent;  
**import** java.awt.event.ActionListener;  
**import** java.awt.event.ItemEvent;  
**import** java.awt.event.ItemListener;  
**import** java.awt.event.KeyEvent;  
  
**public class** App **extends** JFrame {  
 **private** Timer **\_simulationTimer**;  
 **private** LogDialog **logDialog**;  
 **private** GUI **gui**;  
 **private** Habitat **habitat**;  
  
 **private** JMenuBar **jMenuBar** = **new** JMenuBar();  
 **private** JMenu **jmCommands**;  
 **private** JMenuItem **jmiStart**;  
 **private** JMenuItem **jmiStop**;  
 **private** JMenuItem **jmiTime**;  
  
 **private** BuildingFactory **\_buildingFactory**;  
  
 **private** WoodAI **\_woodAi**;  
 **private** CapitalAI **\_capitalAi**;  
 **private** JLayeredPane **\_habbitViewLayeredPane** = **new** JLayeredPane();  
  
 **public** App() {  
 **super**(**"Town"**);  
 **\_simulationTimer** = **new** Timer(System.*currentTimeMillis*());  
 **\_buildingFactory** = **new** BuildingFactory(**\_simulationTimer**, **\_habbitViewLayeredPane**);  
 **\_woodAi** = **new** WoodAI(**\_simulationTimer**, **\_buildingFactory**);  
 **\_capitalAi** = **new** CapitalAI(**\_simulationTimer**, **\_buildingFactory**);  
 InitGui();  
 }  
  
 **private void** InitGui()  
 {  
 **this**.setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);  
 **this**.setResizable(**true**);  
  
 **logDialog** = **new** LogDialog(**this**);  
 **habitat** = **new** Habitat(**\_buildingFactory**, **\_habbitViewLayeredPane**);  
 **gui** = **new** GUI(**\_buildingFactory**, **\_woodAi**, **\_capitalAi**);  
  
 **jmCommands** = **new** JMenu(**"Комманды"**);  
 **jmiStart** = **new** JMenuItem(**"Старт"**);  
 **jmiStop** = **new** JMenuItem(**"Стоп"**);  
 **jmiTime** = **new** JMenuItem(**"Cкрыть время симуляции"**);  
  
 setLayout(**new** BorderLayout());  
 add(**gui**, BorderLayout.***WEST***);  
 add(**habitat**, BorderLayout.***CENTER***);  
 **this**.pack();  
  
 **this**.setBounds(270, 50, 1280, 920);  
  
 KeyboardFocusManager.*getCurrentKeyboardFocusManager*().addKeyEventDispatcher(**new** MyDispatcher());  
 **gui**.**buttonStart**.addActionListener(**new** Butlist());  
 **gui**.**tbStart**.addActionListener(**new** Butlist());  
 **gui**.**tbStop**.addActionListener(**new** Butlist());  
  
 **gui**.**showTimeButton**.addItemListener(**new** TimerVisibleStatelist());  
 **gui**.**hideTimeButton**.addItemListener(**new** TimerVisibleStatelist());  
  
 **gui**.**tbTime**.addActionListener(**new** ActionListener() {  
 **public void** actionPerformed(ActionEvent e) {  
 **if**(**gui**.**showTimeButton**.isSelected())  
 {  
 **gui**.**hideTimeButton**.setSelected(**true**);  
 **gui**.**showTimeButton**.setSelected(**false**);  
 }**else** {  
  
 **gui**.**hideTimeButton**.setSelected(**false**);  
 **gui**.**showTimeButton**.setSelected(**true**);  
  
 }  
 }  
 });  
 **jmiStart**.addActionListener(**new** Butlist());  
 **jmiStop**.addActionListener(**new** Butlist());  
 **jmiStop**.setEnabled(**false**);  
  
 **jmiTime**.addActionListener(**new** ActionListener() {  
 **public void** actionPerformed(ActionEvent e) {  
 **if**(**gui**.**showTimeButton**.isSelected())  
 {  
 **gui**.**hideTimeButton**.setSelected(**true**);  
 **gui**.**showTimeButton**.setSelected(**false**);  
 }**else** {  
 **gui**.**hideTimeButton**.setSelected(**false**);  
 **gui**.**showTimeButton**.setSelected(**true**);  
 }  
 }  
 });  
  
 **jmCommands**.add(**jmiStart**);  
 **jmCommands**.add(**jmiStop**);  
 **jmCommands**.add(**jmiTime**);  
 **jMenuBar**.add(**jmCommands**);  
 setJMenuBar(**jMenuBar**);  
  
 **logDialog**.setVisible(**false**);  
 }  
  
 **public void** Run() {  
 **while** (JFrame.*getFrames*() != **null**) {  
 **\_simulationTimer**.update(System.*currentTimeMillis*());  
 **habitat**.Update(**\_simulationTimer**.**workTime**);  
  
 **gui**.**workTime**.SetTime(**\_simulationTimer**.**workTime**);  
  
 **gui**.changeProgressBars(**\_woodAi**.GetProgress(), **\_capitalAi**.GetProgress());  
 }  
 }  
  
 **private class** Butlist **implements** ActionListener {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 **if** (**habitat**.IsPaused()) {  
 Start();  
 } **else** {  
 Pause();  
 }  
 **habitat**.repaint();  
 }  
 }  
  
 **private class** TimerVisibleStatelist **implements** ItemListener{  
 **public void** itemStateChanged(ItemEvent e){  
   
 **if**(**gui**.**showTimeButton**.isSelected())  
 {  
 **gui**.**workTime**.setVisible(**true**);  
   
 **jmiTime**.setText(**"Скрыть время симуляции"**);  
 **gui**.**tbTime**.setText(**"Скрыть время симуляции"**);  
 }**else** {  
 **gui**.**workTime**.setVisible(**false**);  
   
 **jmiTime**.setText(**"Показать время симуляции"**);  
 **gui**.**tbTime**.setText(**"Показать время симуляции"**);  
 }  
 }  
 }  
  
 **private class** MyDispatcher **implements** KeyEventDispatcher {  
 @Override  
 **public boolean** dispatchKeyEvent(KeyEvent e) {  
 **if** (e.getID() == KeyEvent.***KEY\_RELEASED***) {  
 **if**(e.getKeyCode() == KeyEvent.***VK\_B*** && **habitat**.IsPaused()){  
 Start();  
 **habitat**.RepaintLayered();  
 }  
 **if** (e.getKeyCode() == KeyEvent.***VK\_E*** && !**habitat**.IsPaused()) {  
 Pause();  
 **habitat**.RepaintLayered();  
 }  
   
 **if** (e.getKeyCode() == KeyEvent.***VK\_T***){  
 **if**(!**logDialog**.isActive()){  
 **boolean** isVisible = **gui**.**workTime**.isVisible();  
 **if** (!isVisible)  
 {  
 **gui**.**showTimeButton**.setSelected(**true**);  
 **gui**.**hideTimeButton**.setSelected(**false**);  
 }  
 **else**{  
 **gui**.**hideTimeButton**.setSelected(**true**);  
 **gui**.**showTimeButton**.setSelected(**false**);  
 }  
 }  
 }  
   
 }  
 **return false**;  
 }  
 }  
  
 **private void** Start(){  
   
 **gui**.**buttonStart**.setText(**"Pause"**);  
 **gui**.**buttonStart**.setBackground(Color.***RED***);  
  
 **gui**.**tbStart**.setEnabled(**false**);  
 **gui**.**tbStop**.setEnabled(**true**);  
  
 **jmiStart**.setEnabled(**false**);  
 **jmiStop**.setEnabled(**true**);  
  
 **habitat**.StopHandler(**\_simulationTimer**.**workTime**);  
 **\_simulationTimer**.unpause(System.*currentTimeMillis*());  
 **\_woodAi**.Start();  
 **\_capitalAi**.Start();  
 **gui**.SetThreadButtonEnable(**true**);  
 }  
  
 **private void** Pause() {  
 **gui**.SetThreadButtonEnable(**false**);  
 **if** (**gui**.**buttonStart**.isEnabled()) {  
 **\_simulationTimer**.pause(System.*currentTimeMillis*());  
  
 **if** (**gui**.**checkBox\_showInfo**.isSelected()) {  
 **\_woodAi**.Pause();  
 **\_capitalAi**.Pause();  
 **long** woodBuildCount = **\_buildingFactory**.GetAliveWoodBuildingsCount();  
 **long** capitalBuildCount = **\_buildingFactory**.GetAliveCapitalBuildingsCount();  
  
 **gui**.**buttonStart**.setEnabled(**false**);  
 **gui**.**tbStart**.setEnabled(**false**);  
 **gui**.**tbStop**.setEnabled(**false**);  
 **jmiStart**.setEnabled(**false**);  
 **jmiStop**.setEnabled(**false**);  
  
 **logDialog**.Update(**\_buildingFactory**.GetAliveBuildings().size(), woodBuildCount, capitalBuildCount,  
 **\_simulationTimer**.**workTime**);  
 **logDialog**.setVisible(**true**);  
 } **else** {  
 **\_simulationTimer**.stop(System.*currentTimeMillis*());  
 **\_woodAi**.Stop();  
 **\_capitalAi**.Stop();  
 **gui**.**buttonStart**.setText(**"Start"**);  
 **gui**.**buttonStart**.setBackground(Color.***GREEN***);  
  
 **gui**.**tbStart**.setEnabled(**true**);  
 **gui**.**tbStop**.setEnabled(**false**);  
 **jmiStart**.setEnabled(**true**);  
 **jmiStop**.setEnabled(**false**);  
  
 **habitat**.StartHandler(**\_simulationTimer**.**workTime**);  
 }  
 }  
 }  
  
 **public void** DialogResult(**int** res){  
   
 **gui**.**buttonStart**.setEnabled(**true**);  
  
 **if** ( res == 1 ){   
 **gui**.**buttonStart**.setText(**"Start"**);  
 **gui**.**buttonStart**.setBackground(Color.***GREEN***);  
   
 **habitat**.StartHandler( **\_simulationTimer**.**workTime** );  
 **\_woodAi**.Stop();  
 **\_capitalAi**.Stop();  
 **gui**.SetThreadButtonEnable(**false**);  
  
 **\_simulationTimer**.stop(System.*currentTimeMillis*());  
  
 **jmiStart**.setEnabled(**true**);  
 **jmiStop**.setEnabled(**false**);  
 **gui**.**tbStart**.setEnabled(**true**);

***Habitat.java***

**import** javax.swing.\*;  
  
**import** Buildings.BaseBuild;  
**import** Buildings.BuildingFactory;  
**import** Buildings.CapitalBuild;  
**import** Buildings.WoodBuild;  
**import** SubClasses.Timer;  
  
**import** java.awt.Color;  
**import** java.awt.GridLayout;  
**import** java.util.ArrayList;  
**import** java.util.Random;  
  
**public class** Habitat **extends** JPanel{  
 **private static final long *\_serialVersionUID*** = 1L;  
 **private boolean \_isPaused** = **true**;  
  
 **private** JLayeredPane **\_habbitViewLayeredPane**;  
 **private** BuildingFactory **\_buildingFactory**;  
 **private** Timer **\_simulationTimer**;  
  
 **public void** StartHandler(**double** currentTime ) {  
 **\_isPaused** = **true**;  
 **\_habbitViewLayeredPane**.removeAll();  
 }  
  
 **public void** StopHandler(**double** currentTime ) {  
 **\_isPaused** = **false**;  
 }  
  
 **public void** PauseHandler(){  
 **\_isPaused** = **true**;  
 }  
  
 **public void** ContinueHandler(){  
 **\_isPaused** = **false**;  
 }  
  
 **public void** RepaintLayered() {  
 **\_habbitViewLayeredPane**.repaint();  
 }  
  
 **public boolean** IsPaused(){  
 **return \_isPaused**;  
 }  
  
 **public** Habitat(BuildingFactory buildingFactory, JLayeredPane habbitViewLayeredPane) {  
 **\_buildingFactory** = buildingFactory;  
 **\_habbitViewLayeredPane** = habbitViewLayeredPane;  
 setLayout(**new** GridLayout(1, 1));  
 setBounds(0, 0, 250, 250);  
 setBackground(Color.***gray***);  
 add(**\_habbitViewLayeredPane**);  
 }  
  
  
 **public void** Update(**double** currentTime) {  
 **if** (!**\_isPaused**) {  
 **\_buildingFactory**.RemoveOld();  
  
 *//\_buidingFactory.MoveAll();* }  
   
 **\_habbitViewLayeredPane**.repaint();  
 }  
}

***BaseBuild.java***

**package** Buildings;  
**import** SubClasses.Area;  
  
**import** javax.swing.\*;  
**import** java.util.Random;  
**import** java.util.UUID;  
  
  
**public abstract class** BaseBuild **implements** IBehavior {  
 **private int \_speed**;  
 **private int \_x**, **\_y**;  
 **private** JLayeredPane **\_habbitViewLayeredPane**;  
  
 **protected** UUID **\_id**;  
 **protected double \_bornTime**;  
 **public double** GetBornTime() {**return \_bornTime**;}  
  
 **protected double \_lifeTime**;  
 **public double** GetLifeTime(){**return \_lifeTime**;}  
 **public void** SetLifeTime(**double** value){ **\_lifeTime** = value;}  
  
 **protected** Area **\_finishArea**;  
  
 **private boolean \_isOnPlace**;  
  
 **private** ImageIcon **\_image**;  
 **private int Width**;  
 **private int Height**;  
 **public** JLabel **label** = **new** JLabel();  
  
  
 **public void** setX(**int** x)  
 {  
 **\_x** = x;  
 **label**.setBounds(**\_x**, **\_y**, **Width**, **Height**);  
 }  
  
 **public void** setY(**int** y)  
 {  
 **\_y** = y;  
 **label**.setBounds(**\_x**, **\_y**, **Width**, **Height**);  
 }  
  
 **public int** getX()  
 {  
 **return \_x**;  
 }  
  
 **public int** getY()  
 {  
 **return \_y**;  
 }  
  
 **public int** getWidth(){  
 **return Width**;  
 }  
  
 **public int** getHeight(){  
 **return Height**;  
 }  
  
 **public** BaseBuild(String path, **double** bornTime, **double** lifeTime, JLayeredPane habbitViewLayeredPane, **int** speed){  
 **\_id** = UUID.*randomUUID*();  
 **\_bornTime** = bornTime;  
 **\_lifeTime** = lifeTime;  
 **\_habbitViewLayeredPane** = habbitViewLayeredPane;  
 **\_speed** = speed;  
  
 loadImage(path);  
 CalculateCreationPosition();  
 **label**.setBounds(**\_x**, **\_y**, **Width**, **Height**);  
 **label**.setIcon(**\_image**);  
  
 **int** intIndex = **\_y** + **Height**;  
 Integer index = Integer.*valueOf*(intIndex);  
 **\_habbitViewLayeredPane**.add( **label**, index, -1);  
 }  
  
 **private void** loadImage(String path)  
 {  
 **try** {  
 **\_image** = **new** ImageIcon(path);  
 } **catch** (Exception e) {  
 *//****TODO: handle exception*** }  
 **Width** = **\_image**.getIconWidth();  
 **Height** = **\_image**.getIconHeight();  
 }  
  
 **private void** CalculateCreationPosition()  
 {  
 Random myRand = **new** Random();  
 **\_x** = myRand.nextInt((**int**) **\_habbitViewLayeredPane**.getSize().getWidth() - **Width**);  
 **\_y** = myRand.nextInt((**int**) **\_habbitViewLayeredPane**.getSize().getHeight() - **Height**);  
 }  
  
 **protected void** CalculateFinishPosition()  
 {  
 **if**(**\_finishArea**.IsIn(**\_x**, **\_y**))  
 {  
 **\_finishArea**.SetEndPoint(**\_x**, **\_y**);  
 **return**;  
 }  
  
 **\_finishArea**.CalculateEndPoint(**Width**, **Height**);  
 }  
  
  
 **public void** move(**int** dx, **int** dy) {  
 **if**(**\_isOnPlace** = **true**) **return**;  
  
 **if**(**\_x** != **\_finishArea**.GetEndX())  
 {  
 **if**(**\_x** > **\_finishArea**.GetEndX())  
 **\_x**--;  
 **else  
 \_x**++;  
 move();  
 **return**;  
 }  
  
 **if**(**\_y** != **\_finishArea**.GetEndY()) {  
 **if**(**\_y** > **\_finishArea**.GetEndY())  
 **\_y**--;  
 **else  
 \_y**++;  
 move();  
 **return**;  
 }  
  
 **\_isOnPlace** = **true**;  
 }  
  
 **private void** move(){  
 **label**.setBounds(**\_x**, **\_y**, **Width**, **Height**);  
 **try** {  
 Thread.*sleep*(1);  
 } **catch** (InterruptedException e) {  
 System.***out***.println(**"Thread has been interrupted"**);  
 }  
 }  
  
 **public void** Move()  
 {  
 move(0, 0);  
 }  
  
 @Override  
 **public** String toString()  
 {  
 **return this**.getClass() + **" Время рождения: "** + **\_bornTime**/1000 + **" cек"**;  
 }  
}

***BasAI.java***

**package** Buildings;  
  
**import** SubClasses.Timer;  
**import** java.util.Random;  
  
**public abstract class** BaseAI **extends** Thread  
{  
 **private double \_currentTime** = 0;  
 **private volatile** Timer **\_simulationTimer**;  
 **private** Timer **\_buildTimer** = **new** Timer(**\_currentTime**);  
 **private int \_timerProgressValue**;  
  
 **protected** BuildingFactory **\_buildingFactory**;  
 **private boolean \_isWorking**;  
  
 **public** BaseAI(Timer simulationTimer, BuildingFactory buildingFactory)  
 {  
 **super**();  
 **\_simulationTimer** = simulationTimer;  
 **\_buildingFactory** = buildingFactory;  
  
 **\_buildTimer**.unpause(0);  
 setPriority(Thread.***MAX\_PRIORITY***);  
 start();  
 }  
  
 **public synchronized void** run()  
 {  
 **while** (**true**)  
 {  
 **if**(!**\_isWorking**) {  
 **try** {  
 wait();  
 } **catch** (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
  
 **for** (BaseBuild build : GetBuildings())  
 {  
 build.Move();  
 }  
  
 **\_currentTime** = **\_simulationTimer**.GetCurrentTime();  
 **\_buildTimer**.update(**\_currentTime**);  
 **\_timerProgressValue** = (**int**) (**\_buildTimer**.**workTime** / 10 / GetN());  
 Random myRand = **new** Random();  
  
 **if** (**\_timerProgressValue** >= 100) {  
 **if** (myRand.nextDouble() < GetP()) {  
 Create();  
 }  
 **\_buildTimer**.restart(**\_currentTime**);  
 }  
 }  
 }  
 **public int** GetProgress() {  
 **return \_timerProgressValue**;  
 }  
  
 **public synchronized void** Start()  
 {  
 **\_timerProgressValue** = 0;  
 **\_isWorking** = **true**;  
 notify();  
 }  
  
 **public synchronized void** UnPause()  
 {  
 **\_isWorking** = **true**;  
 **\_buildTimer**.unpause(**\_currentTime**);  
 notify();  
 }  
  
 **public boolean** IsWorking(){**return \_isWorking**;}  
  
 **public void** Pause()  
 {  
 **\_isWorking** = **false**;  
 **\_buildTimer**.pause(**\_currentTime**);  
 }  
  
 **public void** Stop()  
 {  
 **\_timerProgressValue** = 0;  
 **\_isWorking** = **false**;  
 **\_buildTimer**.restart(**\_currentTime** );  
 }  
  
  
 **protected abstract double** GetN();  
 **protected abstract double** GetP();  
 **protected abstract void** Create();  
 **protected abstract** BaseBuild[] GetBuildings();  
}

***BuildingCollection.java***

**package** Buildings;  
  
**import** java.util.\*;  
**import** java.util.stream.Collectors;  
  
**public class** BuildingCollection {  
 **private** HashSet<UUID> **\_ids** = **new** HashSet<UUID>();  
 **private** Vector<BaseBuild> **\_buildings** = **new** Vector<BaseBuild>();  
 **private** TreeMap<UUID, Double> **\_bornTime** = **new** TreeMap<UUID, Double>();  
  
 **private** ArrayList<WoodBuild> **\_woodBuildings** = **new** ArrayList<>();  
 **public** ArrayList<WoodBuild> WoodBuildings() {**return \_woodBuildings**;}  
  
 **private** ArrayList<CapitalBuild> **\_capitalBuildings** = **new** ArrayList<>();  
 **public** ArrayList<CapitalBuild> CapitalBuildings(){**return \_capitalBuildings**;}  
  
 **public synchronized void** AddCapitalBuilding(CapitalBuild build)  
 {  
 **\_capitalBuildings**.add(build);  
 Add(build);  
 }  
  
 **public synchronized void** AddWoodBuilding(WoodBuild build)  
 {  
 **\_woodBuildings**.add(build);  
 Add(build);  
 }  
  
 **private synchronized void** Add(BaseBuild build)  
 {  
 **\_buildings**.add(build);  
 **\_ids**.add(build.**\_id**);  
 **\_bornTime**.put(build.**\_id**, build.**\_bornTime**);  
 }  
   
 **public synchronized void** RemoveAll(List<BaseBuild> builds)  
 {  
 **\_buildings**.removeAll(builds);  
 **\_ids**.removeAll(builds.stream().map(x -> x.**\_id**).collect(Collectors.*toList*()));  
 **for** (UUID id :builds.stream().map(x -> x.**\_id**).collect(Collectors.*toList*()))  
 {  
 **\_bornTime**.remove(id);  
 }  
 }  
  
 **public synchronized** Vector<BaseBuild> GetAliveBuildings()  
 {  
 **return \_buildings**;  
 }  
  
 **public synchronized** List<BaseBuild> GetOldRemoved(**double** time)  
 {  
 *//ConcurrentModificationException 10/10  
 //LINQ в джаве вроде как есть, когда-нибудь разберусь и с ним* List<BaseBuild> buildsForRemoves = **\_buildings**.stream().filter(build -> time - build.**\_bornTime** >= build.**\_lifeTime**).collect(Collectors.*toList*());  
 **if**(!buildsForRemoves.isEmpty())  
 RemoveAll(buildsForRemoves);  
 **return** buildsForRemoves;  
 }  
  
 **public void** MoveAll()  
 {  
 **for** (BaseBuild build :**\_buildings**)  
 {  
 build.Move();  
 }  
 }  
}

***BuildingFactory.java***

**package** Buildings;  
  
**import** SubClasses.Timer;  
  
**import** javax.swing.\*;  
**import** java.util.\*;  
  
**public class** BuildingFactory {  
 **private** Timer **\_simulationTimer**;  
 **private** JLayeredPane **\_habbitViewLayeredPane**;  
 **private** BuildingCollection **\_buildings** = **new** BuildingCollection();  
  
 **private double \_woodBuildingLifeTime** = 10000;  
 **public double** GetWoodBuildingLifeTime(){**return \_woodBuildingLifeTime** / 1000;}  
 **public void** SetWoodBuildingLifeTime(**double** value){ **\_woodBuildingLifeTime** = value \* 1000;}  
  
 **private double \_capitalBuildingLifeTime** = 10000;  
 **public double** GetCapitalBuildingLifeTime(){**return \_capitalBuildingLifeTime** / 1000;}  
 **public void** SetCapitalBuildingLifeTime(**double** value){ **\_capitalBuildingLifeTime** = value \* 1000;}  
  
 **public** BuildingFactory(Timer simulationTimer, JLayeredPane habbitViewLayeredPane)  
 {  
 **\_simulationTimer** = simulationTimer;  
 **\_habbitViewLayeredPane** = habbitViewLayeredPane;  
 }  
  
 **public** WoodBuild CreateWoodBuilding()  
 {  
 WoodBuild temp = **new** WoodBuild(**\_simulationTimer**.**workTime**, **\_woodBuildingLifeTime**, **\_habbitViewLayeredPane**, 20);  
 **\_buildings**.AddWoodBuilding(temp);  
 **return** temp;  
 }  
  
 **public** CapitalBuild CreateCapitalBuilding()  
 {  
 CapitalBuild temp = **new** CapitalBuild(**\_simulationTimer**.**workTime**, **\_capitalBuildingLifeTime**, **\_habbitViewLayeredPane**, 1);  
 **\_buildings**.AddCapitalBuilding(temp);  
 **return** temp;  
 }  
  
 **public void** RemoveOld()  
 {  
 List<BaseBuild> removed = **\_buildings**.GetOldRemoved(**\_simulationTimer**.**workTime**);  
 **for** (BaseBuild build : removed)  
 {  
 **\_habbitViewLayeredPane**.remove(build.**label**);  
 }  
 }  
  
 **public** Vector<BaseBuild> GetAliveBuildings()  
 {  
 **return \_buildings**.GetAliveBuildings();  
 }  
  
 **public long** GetAliveWoodBuildingsCount()  
 {  
 **return \_buildings**.GetAliveBuildings().stream().filter(x -> x **instanceof** WoodBuild).count();  
 }  
  
 **public long** GetAliveCapitalBuildingsCount()  
 {  
 **return \_buildings**.GetAliveBuildings().stream().filter(x -> x **instanceof** CapitalBuild).count();  
 }  
  
 **public** ArrayList<WoodBuild> WoodBuildings(){**return \_buildings**.WoodBuildings();}  
 **public** ArrayList<CapitalBuild> CapitalBuildings(){**return \_buildings**.CapitalBuildings();}  
}

***CapitalBuild.java***

**package** Buildings;  
  
**import** SubClasses.Area;  
  
**import** javax.swing.\*;  
  
**public class** CapitalBuild **extends** BaseBuild {  
 **private static double** *DefaultP* = 0.5;  
 **private static double** *DefaultN* = 3;  
  
 **private static double** *\_P* = 0.5;  
 **private static double** *\_N* = 3;  
  
 **public** CapitalBuild(**double** bornTime, **double** lifeTime, JLayeredPane habbitViewLayeredPane, **int** speed)  
 {  
 **super**(**"src/img/capitalHouse.png"**, bornTime, lifeTime, habbitViewLayeredPane, speed);  
  
 **int** width = (**int**) habbitViewLayeredPane.getSize().getWidth();  
 **int** height = (**int**) habbitViewLayeredPane.getSize().getHeight();  
  
 **\_finishArea** = **new** Area(0, 0, width/2, height/2);  
 CalculateFinishPosition();  
 }  
  
 **public static double** GetDefaultN(){  
 **return** *DefaultN*;  
 }  
  
 **public static double** GetN(){  
 **return** *\_N*;  
 }  
 **public static void** SetN(**double** N){  
 *\_N* = N;  
 }  
  
 **public static double** GetP(){  
 **return** *\_P*;  
 }  
 **public static void** SetP(**double** P){  
 *\_P* = P;  
 }  
}

***WoodBuild.java***

**package** Buildings;  
  
**import** SubClasses.Area;  
  
**import** javax.swing.\*;  
  
**public class** WoodBuild **extends** BaseBuild {  
 **private static double** *DefaultP* = 0.5;  
 **private static double** *DefaultN* = 2;  
  
 **private static double** *\_P* = 0.5;  
 **private static double** *\_N* = 2;  
  
 **public** WoodBuild(**double** bornTime, **double** lifeTime, JLayeredPane habbitViewLayeredPane, **int** speed){  
 *//this(500, 0);* **super**(**"src/img/woodHouse.png"**, bornTime, lifeTime, habbitViewLayeredPane, speed);  
  
 **int** width = (**int**) habbitViewLayeredPane.getSize().getWidth();  
 **int** height = (**int**) habbitViewLayeredPane.getSize().getHeight();  
  
 **\_finishArea** = **new** Area(width/2, height/2, width, height);  
 CalculateFinishPosition();  
 }  
  
 **public static double** GetDefaultN(){  
 **return** *DefaultN*;  
 }  
  
 **public static double** GetN(){**return** *\_N*;}  
 **public static void** SetN(**double** N){  
 *\_N* = N;  
 }  
  
 **public static double** GetP(){  
 **return** *\_P*;  
 }  
 **public static void** SetP(**double** P){  
 *\_P* = P;  
 }  
}

***NumberInput.java***

**package** SubClasses;  
  
**import** Buildings.CapitalBuild;  
**import** Buildings.WoodBuild;  
**import** javafx.beans.property.SimpleDoubleProperty;  
  
**import** java.awt.\*;  
**import** java.awt.event.ActionEvent;  
**import** java.awt.event.ActionListener;  
  
**import** javax.swing.\*;  
**import** javax.swing.border.BevelBorder;  
  
**public class** NumberInput **extends** JPanel {  
  
 **private static final long *serialVersionUID*** = 1L;  
 **private** String **\_type**;  
  
 **private** JProgressBar **\_progressBar** = **new** JProgressBar();  
 **private** JTextArea **\_texttimeToCreate** = **new** JTextArea();  
 **private** JTextField **\_inputText** = **new** JTextField();  
 **private** JButton **\_acceptButton** = **new** JButton(**"Accept"**);  
  
 **public** NumberInput(String type) {  
 setLayout(**new** GridLayout(3, 1, 5, 5));  
 setBorder(BorderFactory.*createLineBorder*(Color.***GRAY***));  
  
 **\_type** = type;  
 **\_acceptButton**.addActionListener(**new** Buttlist());  
  
 **\_texttimeToCreate**.setText(**"Время появления \n"** + **\_type** + **" build"**);  
 **\_texttimeToCreate**.setFocusable(**false**);  
 **\_texttimeToCreate**.setBackground(Color.***LIGHT\_GRAY***);  
 add(**\_texttimeToCreate**);  
  
 **if** (**\_type** == **"Wood"**) { SetTimeText(WoodBuild.*GetN*()); }  
 **if** (**\_type** == **"Capital"**) { SetTimeText(CapitalBuild.*GetN*()); }  
  
 **\_progressBar**.setStringPainted(**true**);  
  
 JPanel supportPanel = **new** JPanel();  
 supportPanel.setLayout(**new** GridLayout(1, 2, 2, 2));  
 supportPanel.add(**\_inputText**);  
 supportPanel.add(**\_acceptButton**);  
 add(supportPanel);  
 add(**\_progressBar**);  
 }  
  
 **public void** SetSliderProgress(**int** value)  
 {  
 **\_progressBar**.setValue(value);  
 }  
  
 **private void** SetTimeText(**double** value)  
 {  
 **\_texttimeToCreate**.setText(**"Время появления \n"** + **\_type** + **"building "** + value + **" сек"**);  
 }  
  
 **private class** Buttlist **implements** ActionListener{  
 **public void** actionPerformed(ActionEvent e) {  
 String regex = **"\\d{1,10}\\.\\d{1,6}|\\d{1,10}"**;  
 **if**(**\_inputText**.getText().matches(regex)){  
 **double** value = Double.*parseDouble*(**\_inputText**.getText());  
 SetValue(value);  
 }  
 **else**{  
 **double** value = **\_type** == **"Wood"** ? WoodBuild.*GetDefaultN*() : CapitalBuild.*GetDefaultN*();  
 SetValue(value);  
 *//****TODO: Dialog here*** }  
 **\_inputText**.setText(**""**);  
 }  
  
 **private void** SetValue(**double** value)  
 {  
 SetTimeText(value);  
 **if** (**\_type** == **"Wood"**) { WoodBuild.*SetN*(value); }  
 **if** (**\_type** == **"Capital"**) { CapitalBuild.*SetN*(value); }  
 }  
 }  
  
}

***LifeTimeManager.java***

**import** Buildings.BaseBuild;  
**import** Buildings.BuildingFactory;  
**import** sun.awt.image.BufferedImageDevice;  
  
**import** javax.swing.\*;  
**import** java.awt.\*;  
**import** java.awt.event.ActionEvent;  
**import** java.awt.event.ActionListener;  
  
**public abstract class** LifeTimeManager<T **extends** BaseBuild> **extends** JPanel  
{  
 **protected** BuildingFactory **\_buildingFactory**;  
 **protected** String **\_text**;  
  
 **private static final long *serialVersionUID*** = 1L;  
 **private** JTextArea **\_textLifeTime** = **new** JTextArea();  
 **private** JTextField **\_inputLifeTime** = **new** JTextField();  
 **private** JButton **\_acceptButton** = **new** JButton(**"Accept"**);  
 **private** JPanel **\_inputArea** = **new** JPanel();  
  
 **public** LifeTimeManager(BuildingFactory buildingFactory)  
 {  
 **\_buildingFactory** = buildingFactory;  
  
 InitGui();  
 ConfigureButton();  
 }  
  
 **public abstract void** SetLifeTime(**double** value);  
  
 **protected void** SetText(String text)  
 {  
 **\_textLifeTime**.setText(text);  
 }  
 **private void** InitGui()  
 {  
 setLayout(**new** GridLayout(2, 1, 2, 2));  
 setBorder(BorderFactory.*createLineBorder*(Color.***GRAY***));  
  
 **\_textLifeTime**.setAlignmentY(0.5f);  
 **\_textLifeTime**.setFocusable(**false**);  
 **\_textLifeTime**.setBackground(Color.***LIGHT\_GRAY***);  
 add(**\_textLifeTime**);  
  
 **\_inputArea**.setLayout(**new** GridLayout(1, 2, 2, 2));  
 **\_inputArea**.add(**\_inputLifeTime**);  
 **\_inputArea**.add(**\_acceptButton**);  
 add(**\_inputArea**);  
 }  
 **private void** ConfigureButton()  
 {  
 **\_acceptButton**.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e)  
 {  
 **double** temp = Double.*parseDouble*(**\_inputLifeTime**.getText());  
 **if** (temp > 0)  
 SetLifeTime(temp);  
 **else** JOptionPane.*showMessageDialog*(**null**,  
 **"Дома не Бенджамин Баттон"**,  
 **"LifeTime Error"**,  
 JOptionPane.***ERROR\_MESSAGE***);  
 **\_inputLifeTime**.setText(**""**);  
 }  
 });  
 }  
}

***GUI.java***

**import** java.awt.\*;  
**import** java.awt.event.ActionEvent;  
**import** java.awt.event.ActionListener;  
**import** javax.swing.\*;  
  
**import** Buildings.BuildingFactory;  
**import** Buildings.CapitalAI;  
**import** Buildings.WoodAI;  
**import** SubClasses.TimePanel;  
  
**public class** GUI **extends** JPanel {  
 **public** JButton **buttonStart** = **new** JButton(**"Start"**);  
 **public** JButton **\_showAliveBuildings** = **new** JButton(**"Текущие объекты"**);  
  
 **private** ButtonGroup **showTimeGroup**;  
 **public** JRadioButton **showTimeButton**;  
 **public** JRadioButton **hideTimeButton**;  
  
 **public** JCheckBox **checkBox\_showInfo**;  
  
 **private** SettingsManager **\_settingsManagerWoodBuild**;  
 **private** SettingsManager **\_settingsManagerCapitalBuild**;  
 **private** BuildingFactory **\_buildingFactory**;  
 **private** WoodBuildingThreadManager **\_woodBuildingThreadManager**;  
  
  
 **public** TimePanel **workTime**;  
   
 **private** JToolBar **toolBar**;  
 **public** JButton **tbStart**,  
 **tbStop**,  
 **tbTime**;  
  
 GUI(BuildingFactory buildingFactory, WoodAI woodAi, CapitalAI capitalAi)  
 {  
 **\_buildingFactory** = buildingFactory;  
 **\_settingsManagerWoodBuild** = **new** WoodBuildingsSettingsManager(**\_buildingFactory**, woodAi);  
 **\_settingsManagerCapitalBuild** = **new** CapitalBuildingsSettingsManager(**\_buildingFactory**, capitalAi);  
  
  
 **buttonStart**.setBackground(Color.***GREEN***);  
   
 **checkBox\_showInfo** = **new** JCheckBox(**"Показывать информацию"**);  
   
 **workTime** = **new** TimePanel();  
 **showTimeGroup** = **new** ButtonGroup();  
 **showTimeButton** = **new** JRadioButton(**"Показывать"**, **true**);  
 **hideTimeButton** = **new** JRadioButton(**"Скрывать"**, **false**);  
   
 **toolBar** = **new** JToolBar(**"Toolbar"**, JToolBar.***VERTICAL***);  
 **tbStart** = **new** JButton(**"Старт"**);  
 **tbStop** = **new** JButton(**"Стоп"**);  
 **tbStop**.setEnabled(**false**);  
 **tbTime** = **new** JButton(**"Показать время симуляции"**);  
  
 **showTimeGroup**.add(**showTimeButton**);  
 **showTimeGroup**.add(**hideTimeButton**);  
   
 setComponentOrientation(ComponentOrientation.***LEFT\_TO\_RIGHT***);   
   
 setLayout(**new** GridBagLayout());   
 GridBagConstraints constraints = **new** GridBagConstraints();   
  
 **int** gridy = 0;  
 constraints.**anchor** = GridBagConstraints.***WEST***;  
 constraints.**fill** = GridBagConstraints.***HORIZONTAL***;  
 constraints.**insets** = **new** Insets(15, 10, 0, 10);  
 constraints.**ipady** = 10;  
 constraints.**gridy** = gridy++;  
 add(**buttonStart**, constraints);  
  
 constraints.**insets** = **new** Insets(15, 10, 0, 10);  
 constraints.**gridy** = gridy++;  
 constraints.**ipady** = 7;  
 add(**\_showAliveBuildings**, constraints);  
  
 constraints.**insets** = **new** Insets(2, 10, 0, 10);  
 constraints.**ipady** = 7;  
 constraints.**gridy** = gridy++;  
 add(**checkBox\_showInfo**, constraints);  
  
 constraints.**insets** = **new** Insets(10, 10, 0, 10);  
 constraints.**ipady** = 5;  
 constraints.**gridy** = gridy++;  
 add(**workTime**, constraints);  
  
 constraints.**insets** = **new** Insets(0, 10, 0, 10);  
 constraints.**gridy** = gridy++;  
 add(**showTimeButton**, constraints);  
  
 constraints.**insets** = **new** Insets(0, 10, 0, 10);  
 constraints.**gridy** = gridy++;  
 add(**hideTimeButton**, constraints);  
  
 constraints.**insets** = **new** Insets(20, 10, 0, 10);  
 constraints.**gridy** = gridy++;  
 add(**\_settingsManagerCapitalBuild**, constraints);  
  
 constraints.**insets** = **new** Insets(15, 10, 0, 10);  
 constraints.**gridy** = gridy++;  
 add(**\_settingsManagerWoodBuild**, constraints);  
  
 constraints.**insets** = **new** Insets(20, 10, 0, 10);  
 constraints.**gridy** = gridy++;  
  
  
  
 **toolBar**.add(**tbStart**);  
 **toolBar**.add(**tbStop**);  
 **toolBar**.add(**tbTime**);  
 add(**toolBar**, constraints);  
  
 ConfigureButtons();  
 }  
  
 **private void** ConfigureButtons()  
 {  
 **\_showAliveBuildings**.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 JOptionPane.*showMessageDialog*(**null**,  
 *//Момент с возвращаемым типом коллекции не особо ясен, верну как это было бы логичным* **new** JList(**\_buildingFactory**.GetAliveBuildings()),  
 **"Текущие здания"**,  
 JOptionPane.***PLAIN\_MESSAGE***);  
 }  
 });  
 }  
  
 **public void** SetThreadButtonEnable(**boolean** value)  
 {  
 **\_settingsManagerWoodBuild**.SetThreadButtonEnable(value);  
 **\_settingsManagerCapitalBuild**.SetThreadButtonEnable(value);  
 }  
  
 **public void** changeProgressBars(**int** woodBuildProgress, **int** capitalBuildProgress)  
 {  
 **\_settingsManagerWoodBuild**.changeProgressBar(woodBuildProgress);  
 **\_settingsManagerCapitalBuild**.changeProgressBar(capitalBuildProgress);  
 }  
}

***SettingsManager.java***

**import** javax.swing.\*;  
  
**import** Buildings.BaseAI;  
**import** Buildings.BuildingFactory;  
**import** SubClasses.NumberInput;  
  
**import** java.awt.\*;  
**import** java.awt.event.ActionEvent;  
**import** java.awt.event.ActionListener;  
  
**public abstract class** SettingsManager **extends** JPanel {  
 **private static final long *serialVersionUID*** = 1L;  
  
 **protected** CreationFrequencyManager **\_creationFrequencyManager**;  
 **protected** LifeTimeManager **\_lifeTimeManager**;  
 **protected** NumberInput **\_numberInput**;  
 **protected** BuildingThreadManager **\_buildingThreadManager**;  
 **protected** BaseAI **\_ai**;  
  
 **protected** BuildingFactory **\_buildingFactory**;  
 **protected** String **\_type**;  
  
 **private** JButton **\_showHideButton**;  
 **private boolean \_isVisible** = **false**;  
 **private** JPanel **\_components**;  
  
 **public** SettingsManager(String type, BuildingFactory buildingFactory, BaseAI ai) {  
 **\_buildingFactory** = buildingFactory;  
 **\_type** = type;  
 **\_ai** = ai;  
 InitGui();  
 }  
  
 **protected abstract void** InitGuiComponent();  
  
 **private void** InitGui()  
 {  
 setLayout(**new** BorderLayout());  
 **\_numberInput** = **new** NumberInput(**\_type**);  
 **\_showHideButton** = **new** JButton();  
  
 InitGuiComponent();  
 **\_components** = **new** JPanel();  
 **\_components**.setLayout(**new** BoxLayout(**\_components**, BoxLayout.***Y\_AXIS***));  
 **\_components**.add(**\_numberInput**);  
 **\_components**.add(**\_creationFrequencyManager**);  
 **\_components**.add(**\_lifeTimeManager**);  
 **\_components**.add(**\_buildingThreadManager**);  
 add(**\_showHideButton**, BorderLayout.***PAGE\_START***);  
 add(**\_components**);  
  
 ConfigureButton();  
 **\_components**.setVisible(**\_isVisible**);  
 }  
  
 **protected void** SetButtonText(String value)  
 {  
 **\_showHideButton**.setText(value);  
 }  
  
 **private void** ConfigureButton()  
 {  
 **\_showHideButton**.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 **\_isVisible** = !**\_isVisible**;  
 **\_components**.setVisible(**\_isVisible**);  
 }  
 });  
 }  
  
 **public void** changeProgressBar(**int** progress) {  
 **\_numberInput**.SetSliderProgress(progress);  
 }  
 **public void** SetThreadButtonEnable(**boolean** value)  
 {  
 **\_buildingThreadManager**.ChangeState(value);  
 }  
}

***LogDialog.java***

import SubClasses.Log;

import javax.swing.JDialog;

import java.awt.FlowLayout;

import java.awt.Container;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import javax.swing.\*;

// 1 - была нажата "ОК"

// 2 - была нажата "Отмена"

public class LogDialog extends JDialog

{

private Log log;

private JButton ok;

private JButton cansel;

private Container container;

public LogDialog(App owner)

    {

        super(owner, "Log", true);

setDefaultCloseOperation(JDialog.DO\_NOTHING\_ON\_CLOSE);

setLayout(new FlowLayout());

log = new Log();

ok = new JButton("OK");

cansel = new JButton("Отмена");

ok.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent event) {

setVisible(false);

log.clear();

owner.DialogResult(1);

}

});

cansel.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent event) {

setVisible(false);

log.clear();

owner.DialogResult(0);

}

});

container = getContentPane();

container.add(log );

container.add(ok);

container.add(cansel);

this.pack();

setBounds(600, 300, 316, 195);

}

public void Update(int buildCount, int woodCount, int capitalCount, double simulationTime)

{

log.setVisible(true);

log.appendIntAtrib( "Buildings count", buildCount);

log.appendIntAtrib( "Wood count", woodCount);

log.appendIntAtrib( "Capital count", capitalCount);

log.appendIntAtrib( "Total simulation time (sec.)", (int)(simulationTime / 1000) );

}

}

***Log.java***

package SubClasses;

import java.awt.Color;

import java.awt.Font;

import javax.swing.JPanel;

import javax.swing.JTextArea;

public class Log extends JPanel{

private static final long serialVersionUID = 1L;

private final String head = " LOG:\n";

private JTextArea textArea = new JTextArea(head, 5, 25);

private Font font = new Font("Verdana", Font.PLAIN, 12);

public Log(){

textArea.setBackground(Color.darkGray);

textArea.setEditable(false);

textArea.setFont(font);

textArea.setForeground(Color.LIGHT\_GRAY);

textArea.setTabSize(4);

add(textArea);

}

public void appendIntAtrib(String str, int X){

textArea.append(" ");

textArea.append(str);

textArea.append(" ---> ");

textArea.append(String.valueOf(X));

textArea.append("\n");

}

public void clear(){

textArea.setText(head);

}

}

***TimePanel.java***

package SubClasses;

import javax.swing.\*;

public class TimePanel extends JPanel{

private static final long serialVersionUID = 1L;

private JLabel textWorkTime = new JLabel("Текущее время симуляции:");

private JLabel valueWorkTime = new JLabel();

public TimePanel(){

textWorkTime.setBounds(0, 0, 200, 40);

add(textWorkTime);

valueWorkTime.setBounds(0, 0, 150, 40);

add(valueWorkTime);

}

public void SetTime(double currentTime){

valueWorkTime.setText(String.valueOf(((int)currentTime)/1000) + " сек");

}

public boolean ChangeVisibleState(){

setVisible(!isVisible());

return isVisible();

}

}

***Timer.java***

package SubClasses;

public class Timer {

public double workTime;

public double startTime;

private double diff;

private boolean isPaused;

public Timer(double currentTimeMillis) {

restart(currentTimeMillis);

pause(currentTimeMillis);

}

public void update(double currentTimeMillis) {

if(!isPaused)

workTime = currentTimeMillis - startTime - diff;

else

diff = currentTimeMillis - startTime - workTime;

}

public void restart(double currentTimeMillis) {

unpause(currentTimeMillis);

diff = 0;

workTime = 0;

startTime = currentTimeMillis;

}

public void unpause(double currentTimeMillis){

isPaused = false;

}

public void pause(double currentTimeMillis){

isPaused = true;

}

public boolean isPaused(double currentTimeMillis){

return isPaused;

}

public void stop(double currentTimeMillis){

restart(currentTimeMillis);

pause(currentTimeMillis);

}

}

***CreationFrequencyManager.java***

**import** javax.swing.\*;  
**import** java.awt.\*;  
**import** javax.swing.event.ChangeEvent;  
**import** javax.swing.event.ChangeListener;  
  
**public abstract class** CreationFrequencyManager **extends** JPanel {  
 **private static final long *serialVersionUID*** = 1L;  
  
 **protected** JSlider **\_slider** = **new** JSlider();  
 **private** JTextArea **\_textChanceToCreate** = **new** JTextArea();  
  
 **public** CreationFrequencyManager()  
 {  
 InitGui();  
 }  
  
 **protected void** SetText(String value)  
 {  
 **\_textChanceToCreate**.setText(**"Шанс появления "** + value);  
 }  
  
 **private void** InitGui()  
 {  
 setLayout(**new** GridLayout(2, 1, 5, 5));  
 setBorder(BorderFactory.*createLineBorder*(Color.***GRAY***));  
  
 **\_textChanceToCreate**.setAlignmentY(0.5f);  
 **\_textChanceToCreate**.setFocusable(**false**);  
 **\_textChanceToCreate**.setBackground(Color.***LIGHT\_GRAY***);  
  
 **\_slider**.setPaintLabels(**true**);  
 **\_slider**.setMajorTickSpacing(25);  
  
 add(**\_textChanceToCreate**);  
 add(**\_slider**);  
  
 ConfigureButton();  
 }  
  
 **public abstract void** SetFrequency(**int** value);  
  
 **private void** ConfigureButton()  
 {  
 **\_slider**.addChangeListener(**new** ChangeListener() {  
 @Override  
 **public void** stateChanged(ChangeEvent e) {  
 SetFrequency(((JSlider) e.getSource()).getValue() / 100);  
 }  
 });  
 }  
}

***BuildingThreadManager.java***

**import** Buildings.BaseAI;  
  
**import** javax.swing.\*;  
**import** java.awt.\*;  
**import** java.awt.event.ActionEvent;  
**import** java.awt.event.ActionListener;  
  
**public abstract class** BuildingThreadManager **extends** JPanel {  
 **private** String[] **\_threadPriority** = **new** String[]{**"1"**,**"2"**,**"3"**,**"4"**,**"5"**,**"6"**,**"7"**,**"8"**,**"9"**, **"10"**};  
  
 **protected** BaseAI **\_ai**;  
 **private** JButton **\_stopBuildings** = **new** JButton();  
 **private** JComboBox **\_combobox** = **new** JComboBox(**\_threadPriority**);  
 **private** JLabel **\_label** = **new** JLabel(**"Приоритет"**);  
 **private boolean \_isEnable**;  
  
 **public** BuildingThreadManager(BaseAI ai)  
 {  
 **\_ai** = ai;  
  
 InitGui();  
 }  
  
 **private void** InitGui() {  
 setBorder(BorderFactory.*createLineBorder*(Color.***GRAY***));  
  
 **\_stopBuildings**.setText(**"Остановаить"**);  
 **\_stopBuildings**.setEnabled(**false**);  
  
 add(**\_stopBuildings**);  
 add(**\_label**);  
 add(**\_combobox**);  
  
 ConfigureButton();  
 }  
  
 **private void** ConfigureButton()  
 {  
 **\_stopBuildings**.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 **if**(**\_ai**.IsWorking())  
 {  
 **\_ai**.Pause();  
 **\_stopBuildings**.setText(**"Возобновить"**);  
 }  
 **else** {  
 **\_ai**.UnPause();  
 **\_stopBuildings**.setText(**"Остановаить"**);  
 }  
 }  
 });  
  
 **\_combobox**.addActionListener(**new** ActionListener() {  
 @Override  
 **public void** actionPerformed(ActionEvent e) {  
 **\_ai**.setPriority(Integer.*parseInt*((String)((JComboBox)e.getSource()).getSelectedItem()));  
 }  
 });  
 }  
  
 **public void** ChangeState(**boolean** isEnable)  
 {  
 **\_isEnable** = isEnable;  
 **\_stopBuildings**.setEnabled(**\_isEnable**);  
 }  
}