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 №3

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.

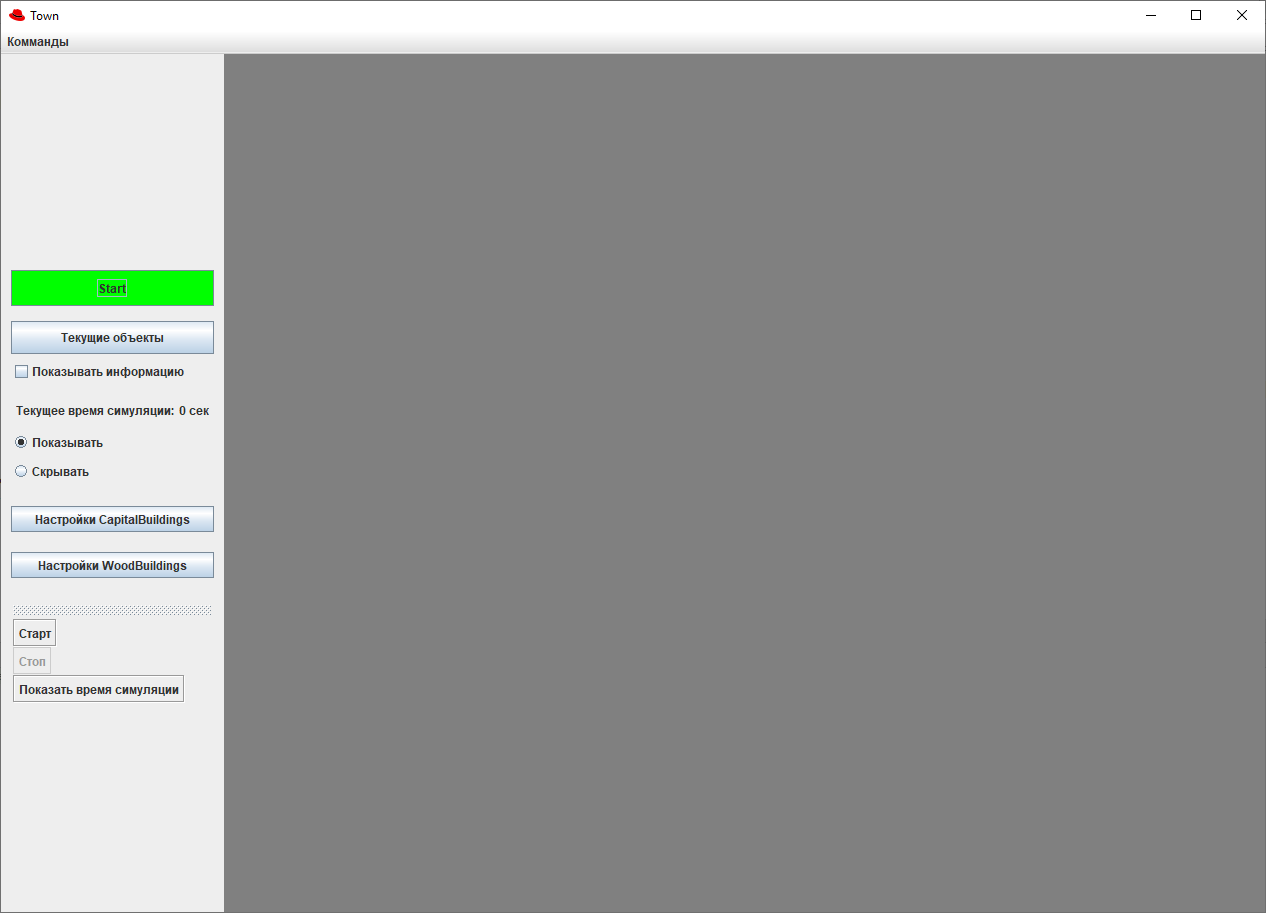
#### Практические задания

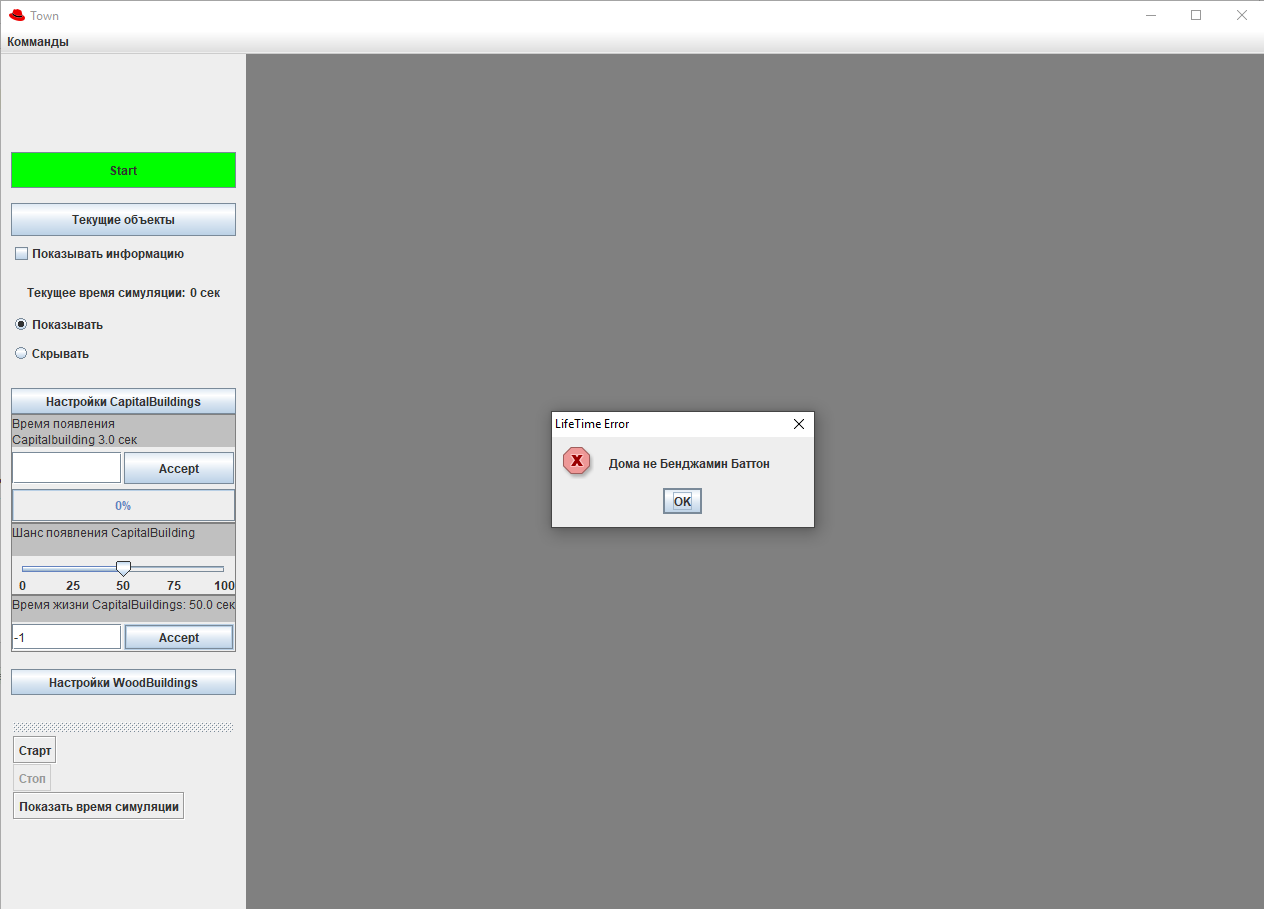
1. Изучить особенности реализации классов-коллекций в Java.
2. Доработать программу, созданную в лабораторной работе № 2:
3. добавить генерируемым объектам понятия «время рождения» и «время жизни». Время рождения устанавливается в момент генерации объекта и по значению соответствует времени, прошедшему от начала симуляции. Время жизни – время, через которое объект должен исчезнуть, считая от времени рождения;
4. вынести установку параметров времени жизни объектов в пользовательский интерфейс. Для каждого типа объекта должно задаваться собственное время. Рекомендуется использовать текстовые поля, но следуют помнить о проверке на ввод некорректных данных;
5. добавить генерируемым объектам уникальные целочисленные идентификаторы (случайные числа), которые назначаются при генерации объекта. Для хранения сгенерированных идентификаторов используйте коллекцию удобную для поиска по варианту;
6. использовать коллекции по варианту. При генерации объекта происходит добавление его в коллекцию (в класс добавить поле идентификатора), а во вторую коллекцию: идентификаторы существующих объектов, в третью идентификатор + время рождения. При возникновении события по таймеру обойдите коллекцию и удалите все объекты, время жизни которых истекло, а также все данные во вспомогательных коллекциях;

добавьте в панель управления кпопку «Текущие объекты». По нажатию на эту кнопку появляется модальное диалоговое окно, содержащее список всех «живых» объектов на момент нажатия со временем их рождения (тип объекта, время рождения, идентификатор). В класс диалогового окна должна передаваться коллекция с хранением объектов по времени рождения. Типы коллекций задаются вариантом

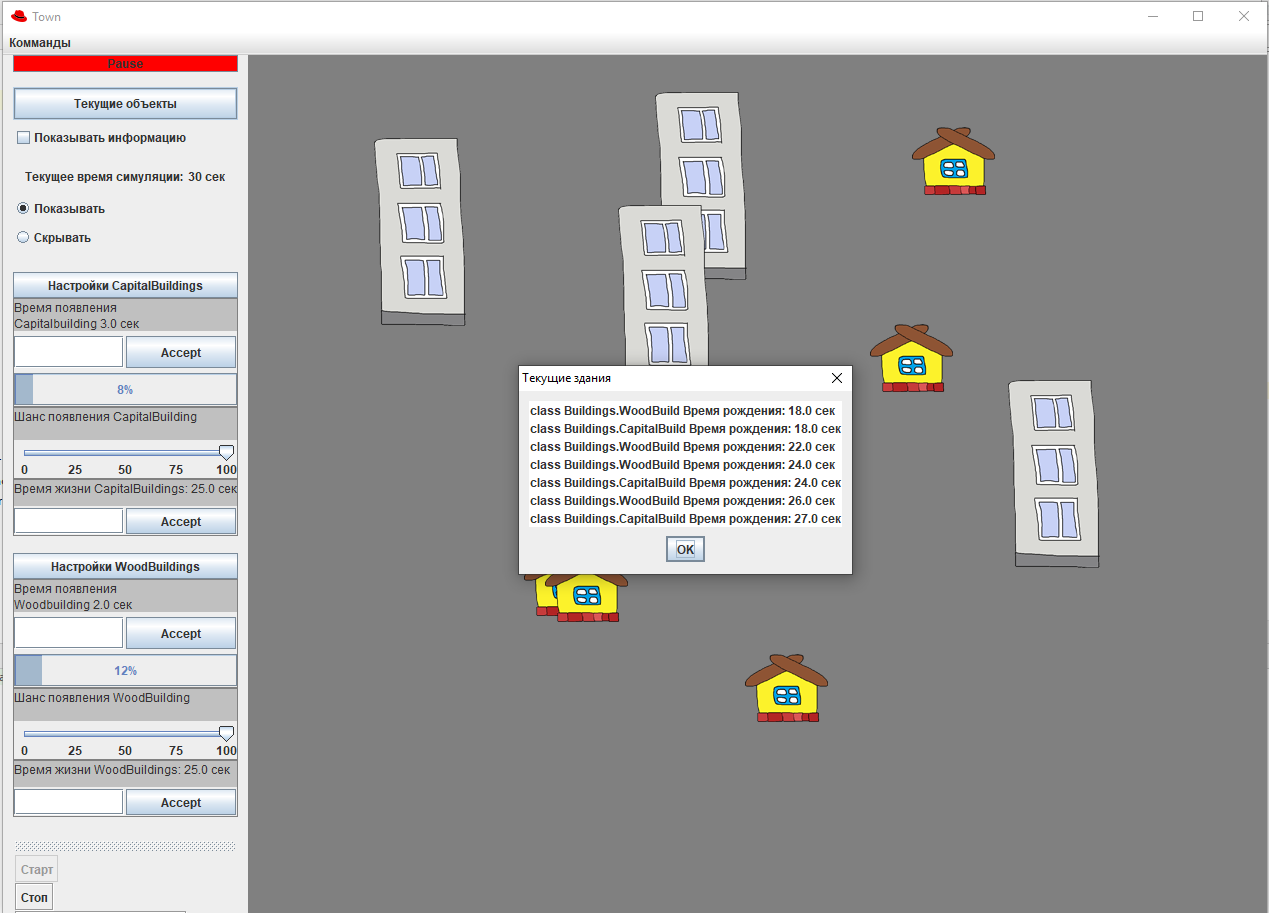
**Результаты работы программы**

При запуске программы пользователь видит следующее окно:



Появилась возможность устанавливать время жизни объекта. При недопустимом введённом значении время жизни не меняется, а пользователь получает уведомление 

При нажатии на кнопку “Текущие объекты” показываются список все сгенерированных объектов (класс и время рождения).

****

**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** 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** JLayeredPane **\_habbitViewLayeredPane** = **new** JLayeredPane();  
  
 **public** App() {  
 **super**(**"Town"**);  
 **\_simulationTimer** = **new** Timer(System.*currentTimeMillis*());  
 **\_buildingFactory** = **new** BuildingFactory(**\_simulationTimer**, **\_habbitViewLayeredPane**);  
 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**);  
  
 **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(**habitat**.GetWoodProgress(), **habitat**.GetCapitalProgress());  
 }  
 }  
  
 **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());  
 }  
  
 **private void** Pause(){  
 **if**(gui.buttonStart.isEnabled()){  
 \_simulationTimer.pause( System.currentTimeMillis() );  
   
 **if**(gui.checkBox\_showInfo.isSelected()){  
  
 **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**{  
 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 );  
   
 \_simulationTimer.stop(System.currentTimeMillis());  
 }  
 }   
 }  
  
 **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 );  
   
 \_simulationTimer.stop(System.currentTimeMillis());  
  
 jmiStart.setEnabled(**true**);  
 jmiStop.setEnabled(**false**);  
 gui.tbStart.setEnabled(**true**);  
 gui.tbStop.setEnabled(**false**);  
 }  
 **if** ( res == 0 ){  
 \_simulationTimer.unpause( System.currentTimeMillis() );  
 habitat.ContinueHandler();  
  
 jmiStart.setEnabled(**false**);  
 jmiStop.setEnabled(**true**);  
 gui.tbStart.setEnabled(**false**);  
 gui.tbStop.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** ArrayList<BaseBuild> **\_buildList** = **new** ArrayList<BaseBuild>();  
 **private boolean \_isPaused** = **true**;  
  
 **private int woodTimerProgressValue**;  
 **private int capitalTimerProgressValue**;  
  
  
 **private double currentTime** = 0;  
 **private** Timer **woodTimer** = **new** Timer( **currentTime** );  
 **private** Timer **capitalTimer** = **new** Timer( **currentTime** );  
 **private** JLayeredPane **\_habbitViewLayeredPane**;  
  
 **private** BuildingFactory **\_buidingFactory**;  
  
 **private** Timer **\_simulationTimer**;  
  
 **public void** StartHandler(**double** currentTime ) {  
 **this**.**currentTime** = currentTime;   
 **\_isPaused** = **true**;  
 **woodTimer**.pause( currentTime );  
 **capitalTimer**.pause( currentTime );  
 **\_habbitViewLayeredPane**.removeAll();  
 **woodTimerProgressValue** = 0;  
 **capitalTimerProgressValue** = 0;  
 }  
  
 **public void** StopHandler(**double** currentTime ) {  
 **this**.**currentTime** = currentTime;   
 **\_buildList**.clear();  
 **woodTimer**.restart( currentTime );  
 **capitalTimer**.restart( currentTime );  
 **\_isPaused** = **false**;  
 }  
  
 **public void** PauseHandler(){  
 **\_isPaused** = **true**;  
 }  
  
 **public void** ContinueHandler(){  
 **\_isPaused** = **false**;  
 }  
  
 **public void** RepaintLayered() {  
 **\_habbitViewLayeredPane**.repaint();  
 }  
  
 **public int** GetWoodProgress() {  
 **return woodTimerProgressValue**;  
 }  
  
 **public int** GetCapitalProgress() {  
 **return capitalTimerProgressValue**;  
 }  
  
 **public boolean** IsPaused(){  
 **return \_isPaused**;  
 }  
  
 **public** ArrayList<BaseBuild> GetBuildList(){  
 **return \_buildList**;  
 }  
  
 **public** Habitat(BuildingFactory buidingFactory, JLayeredPane habbitViewLayeredPane) {  
 **\_buidingFactory** = buidingFactory;  
 **\_habbitViewLayeredPane** = habbitViewLayeredPane;  
 setLayout(**new** GridLayout(1, 1));  
 setBounds(0, 0, 250, 250);  
 setBackground(Color.***gray***);  
 add(**\_habbitViewLayeredPane**);  
 }  
  
  
 **public void** Update(**double** currentTime) {  
 **this**.**currentTime** = currentTime;   
 **if** (!**\_isPaused**) {  
 **\_buidingFactory**.RemoveOld();  
  
 Random myRand = **new** Random();  
  
 **woodTimer**.update( currentTime );  
 **capitalTimer**.update( currentTime );  
  
 **woodTimerProgressValue** = (**int**) (**woodTimer**.**workTime** / 10 / WoodBuild.*GetN*());  
 **capitalTimerProgressValue** = (**int**) (**capitalTimer**.**workTime** / 10 / CapitalBuild.*GetN*());  
  
 **if** (**woodTimerProgressValue** >= 100) {  
 **if** (myRand.nextDouble() < WoodBuild.*GetP*()) {  
 AddObj(**\_buidingFactory**.GetWoodBuilding());  
 }  
 **woodTimer**.restart(currentTime);  
 }  
  
 **if** (**capitalTimerProgressValue** >= 100) {  
 **if** (myRand.nextDouble() < CapitalBuild.*GetP*()) {  
 AddObj(**\_buidingFactory**.GetCapitalBuilding());  
 }  
 **capitalTimer**.restart( currentTime );  
 }  
  
 }  
   
 **\_habbitViewLayeredPane**.repaint();  
 }  
  
 **private void** AddObj(BaseBuild obj) {  
 Random myRand = **new** Random();  
 obj.setX(myRand.nextInt((**int**) **\_habbitViewLayeredPane**.getSize().getWidth() - obj.getWidth()));  
 obj.setY(myRand.nextInt((**int**) **\_habbitViewLayeredPane**.getSize().getHeight() - obj.getHeight()));  
 **int** intIndex = obj.getY() + obj.getHeight();  
 Integer index = Integer.*valueOf*(intIndex);  
 **\_habbitViewLayeredPane**.add( obj.**label**, index, -1);  
 }  
}

***BaseBuild.java***

**package** Buildings;  
**import** javax.swing.\*;  
**import** java.util.UUID;  
  
  
**public abstract class** BaseBuild **implements** IBehavior {  
 **private int** \_x, \_y;  
 **protected** UUID \_id;  
 **protected double** \_bornTime;  
 **public double** GetBornTime() {**return** \_bornTime;}  
  
 **protected double** \_lifeTime = 5;  
 **public double** GetLifeTime(){**return** \_lifeTime;}  
 **public void** SetLifeTime(**double** value){ \_lifeTime = value;}  
  
 **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 void** move(**int** dx, **int** dy) {  
 **\_x** += dx;  
 **\_y** += dy;  
 **label**.setBounds(**\_x**, **\_y**, **Width**, **Height**);  
 }  
  
 **public** BaseBuild(String path, **double** bornTime, **double** lifeTime){  
 **\_id** = UUID.*randomUUID*();  
 **\_bornTime** = bornTime;  
 **\_lifeTime** = lifeTime;  
 **\_x** = 0;  
 **\_y** = 0;  
 loadImage(path);  
 **label**.setBounds(**\_x**, **\_y**, **Width**, **Height**);  
 **label**.setIcon(**\_image**);  
 }  
 **public** BaseBuild(String path, **int** x, **int** y){  
 **\_id** = UUID.*randomUUID*();  
 **\_x** = x;  
 **\_y** = y;  
 loadImage(path);  
 **label**.setBounds(**\_x**, **\_y**, **Width**, **Height**);  
 **label**.setIcon(**\_image**);  
 }  
  
 **private void** loadImage(String path)  
 {  
 **try** {  
 **\_image** = **new** ImageIcon(path);  
 } **catch** (Exception e) {  
 *//TODO: handle exception* }  
 Width = \_image.getIconWidth();  
 Height = \_image.getIconHeight();  
 }  
  
 @Override  
 **public** String toString()  
 {  
 **return this**.getClass() + **" Время рождения: "** + \_bornTime/1000 + **" cек"**;  
 }  
}

***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>();  
   
 **public void** Add(BaseBuild build)  
 {  
 **\_buildings**.add(build);  
 **\_ids**.add(build.**\_id**);  
 **\_bornTime**.put(build.**\_id**, build.**\_bornTime**);  
 }  
   
 **public 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** Vector<BaseBuild> GetAliveBuildings()  
 {  
 **return \_buildings**;  
 }  
  
 **public** 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;  
 }  
}

***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 GetWoodBuilding()  
 {  
 WoodBuild temp = **new** WoodBuild(**\_simulationTimer**.**workTime**, **\_woodBuildingLifeTime**);  
 **\_buildings**.Add(temp);  
 **return** temp;  
 }  
  
 **public** CapitalBuild GetCapitalBuilding()  
 {  
 CapitalBuild temp = **new** CapitalBuild(**\_simulationTimer**.**workTime**, **\_capitalBuildingLifeTime**);  
 **\_buildings**.Add(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();  
 }  
}

***CapitalBuild.java***

**package** Buildings;  
  
**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(){  
 **super**(**"src/img/capitalHouse.png"**, 0, 10);  
 }  
  
 **public** CapitalBuild(**double** bornTime, **double** lifeTime)  
 {  
 **super**(**"src/img/capitalHouse.png"**, bornTime, lifeTime);  
 }  
  
 **public** CapitalBuild(**int** x, **int** y){  
 **super**(**"src/img/CapitalHouse.png"**, x, y);  
 }  
  
 **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;  
  
**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(){  
 **super**(**"src/img/woodHouse.png"**, 0, 10);  
 }  
  
 **public** WoodBuild(**double** bornTime, **double** lifeTime){  
 *//this(500, 0);* **super**(**"src/img/woodHouse.png"**, bornTime, lifeTime);  
  
 }  
  
 **public** WoodBuild(**int** x, **int** y){  
 **super**(**"src/img/woodHouse.png"**, x, y);  
 }  
  
 **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** 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**;  
  
 **public** TimePanel **workTime**;  
   
 **private** JToolBar **toolBar**;  
 **public** JButton **tbStart**,  
 **tbStop**,  
 **tbTime**;  
  
 GUI(BuildingFactory buildingFactory)  
 {  
 **\_buildingFactory** = buildingFactory;  
 **\_settingsManagerWoodBuild** = **new** WoodBuildingsSettingsManager(**\_buildingFactory**);  
 **\_settingsManagerCapitalBuild** = **new** CapitalBuildingsSettingsManager(**\_buildingFactory**);  
  
 **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** changeProgressBars(**int** woodBuildProgress, **int** capitalBuildProgress)  
 {  
 **\_settingsManagerWoodBuild**.changeProgressBar(woodBuildProgress);  
 **\_settingsManagerCapitalBuild**.changeProgressBar(capitalBuildProgress);  
 }  
}

***SettingsManager.java***

**import** javax.swing.\*;  
  
**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** BuildingFactory **\_buildingFactory**;  
 **protected** String **\_type**;  
  
 **private** JButton **\_showHideButton**;  
 **private boolean \_isVisible** = **false**;  
 **private** JPanel **\_components**;  
  
 **public** SettingsManager(String type, BuildingFactory buildingFactory) {  
 **\_buildingFactory** = buildingFactory;  
 **\_type** = type;  
  
 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**);  
 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);  
 }  
}

***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);

}

}