

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ
(МИНОБРНАУКИ)

ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО
ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ «САНКТ-ПЕТЕРБУРГСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ
И ОПТИКИ»
(СПбГУ ИТМО)

УТВЕРЖДАЮ
Ректор СПбГУ ИТМО,
докт. техн. наук, профессор
В. Н. Васильев

_____ 2008 г.

ПРОГРАММНОЕ СРЕДСТВО 3GENETIC

ПРОГРАММНЫЙ МОДУЛЬ CORE
ТЕКСТ ПРОГРАММЫ

ЛИСТ УТВЕРЖДЕНИЯ

7.190.00001-01 12 02-ЛУ

Декан факультета «Информационные
технологии и программирование»
докт. техн. наук, профессор
_____ В. Г. Парфенов

Руководитель темы
заведующий кафедрой «Технологии программирования»,
докт. техн. наук, профессор
_____ А. А. Шалыто

| | |
|----------------------|--------------|
| Имя, И. подл. | Подп. и дата |
| Имя, И. дубл. | |
| Имя, И. зам. Имя, И. | |
| Подп. и дата | |

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ
(МИНОБРНАУКИ)

ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО
ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ «САНКТ-ПЕТЕРБУРГСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ
И ОПТИКИ»
(СПбГУ ИТМО)

УТВЕРЖДЕНО
7.190.00001-01 12 02-ЛУ

ПРОГРАММНОЕ СРЕДСТВО 3GENETIC

ПРОГРАММНЫЙ МОДУЛЬ CORE
ТЕКСТ ПРОГРАММЫ

7.190.00001-01 12 02

Листов 37

| | | | | |
|--------------|--------------|--------------|--------------|--------------|
| Имя. N подл. | Подп. и дата | Взам. имя. N | Имя. N дубл. | Подп. и дата |
| | | | | |

АННОТАЦИЯ

В данном документе приводится текст модуля core программного средства 3GENETIC, представляющего собой графическую оболочку генетического алгоритма, а также реализующего все базовую функциональность ядра.

СОДЕРЖАНИЕ

| | |
|--|--|
| Аннотация..... | 2 |
| Содержание..... | 3 |
| 3. Модуль core | 4 |
| 3.1. Пакет CORE.GUI..... | 5 |
| 3.1.1. laboratory/core/gui/chooser/ChooseDialog.java..... | 5 |
| 3.1.2. laboratory/core/gui/chooser/ConfigDialog.java..... | 8 |
| 3.1.3. laboratory/core/gui/main/actions/PauseAction.java | 10 |
| 3.1.4. laboratory/core/gui/main/actions/ResumeAction.java | 11 |
| 3.1.5. laboratory/core/gui/main/actions/TimerListener.java..... | 12 |
| 3.1.6. laboratory/core/gui/main/actions/ChooseEmulatorAction.java | 12 |
| 3.1.7. laboratory/core/gui/main/actions/ChooseGAAction.java | 13 |
| 3.1.8. laboratory/core/gui/main/actions/ChooseIndividualAction.java | 14 |
| 3.1.9. laboratory/core/gui/main/menu/actions/ExitAction.java | 15 |
| 3.1.10. laboratory/core/gui/main/menu/actions/SaveGenerationAction.java | 16 |
| 3.1.11. laboratory/core/gui/main/menu/actions/SavePictureGenerationAction.java | 17 |
| 3.1.12. laboratory/core/gui/main/menu/actions/ShowBestAction.java..... | 18 |
| 3.1.13. laboratory/core/gui/main/menu/actions/ShowGenerationAction.java | 18 |
| 3.1.14. laboratory/core/gui/main/menu/MainMenu.java | 19 |
| 3.1.15. laboratory/core/gui/main/MainFrame.java | 20 |
| 3.1.16. laboratory/core/gui/shower/utils/ShowTableModel.java | 22 |
| 3.1.17. laboratory/core/gui/shower/IndividualDialog.java | 24 |
| 3.1.18. laboratory/core/gui/FrameCreator.java | 26 |
| 3.2. Пакет CORE..... | 27 |
| 3.2.1. laboratory/core/loader/PluginLoader.java..... | 27 |
| 3.2.2. laboratory/core/runner/GARunner.java | 31 |
| 3.2.3. laboratory/core/util/RunnableLock.java | 33 |
| 3.2.4. laboratory/core/InterfaceConfig.java | Ошибка! Закладка не определена. |
| 3.2.5. laboratory/core/Main.java | 33 |
| 3.2.6. laboratory/core/SelectedPlugin.java | 34 |

1. МОДУЛЬ CORE

Модуль core является составной частью программного средства 3GENETIC и реализует базовую функциональность ядра. Исходный текст модуля хранится в 23-х файлах:

1. laboratory/core/gui/chooser/ChooseDialog.java – реализация диалога выбора модуля;
2. laboratory/core/gui/chooser/ConfigDialog.java – реализация диалога настройки модуля;
3. laboratory/core/gui/main/actions/PauseAction.java – реализация действия остановки алгоритма;
4. laboratory/core/gui/main/actions/ResumeAction.java – реализация действия запуска алгоритма после остановки;
5. laboratory/core/gui/main/actions/TimerListener.java – реализация таймера для обновления информации на экране;
6. laboratory/core/gui/main/actions/ChooseEmulatorAction – реализация действия выбора модуля визуализации;
7. laboratory/core/gui/main/actions/ChooseGAAction – реализация действия выбора модуля генетического алгоритма;
8. laboratory/core/gui/main/actions/ChooseIndividualAction – реализация действия выбора модуля особи;
9. laboratory/core/gui/main/menu/actions/ExitAction.java – реализация действия окончания работы программы;
10. laboratory/core/gui/main/menu/actions/SaveGenerationAction.java – реализация действия сохранения текущего поколения особей на жесткий диск;
11. laboratory/core/gui/main/menu/actions/SavePictureAction.java – реализация действия сохранения текущего графика на жесткий диск;
12. laboratory/core/gui/main/menu/actions/ShowBestAction.java – реализация действия по открыванию диалога выбора просмотра лучших особей;
13. laboratory/core/gui/main/menu/actions/ShowGenerationAction.java – реализация действия по открыванию диалога выбора просмотра особей текущего поколения;
14. laboratory/core/gui/main/menu/MainMenu.java – создание меню основного окна программного средства;
15. laboratory/core/gui/main/MainFrame.java – основное окно программного средства;
16. laboratory/core/gui/shower/utills/ShowTableModel.java – реализация внешнего вида таблицы в диалоге выбора особи для просмотра;
17. laboratory/core/gui/shower/IndividualDialog.java – реализация диалога выбора особи для просмотра;
18. laboratory/core/gui/FrameCreator.java – реализация загрузчика графической оболочки;
19. laboratory/core/loader/PluginLoader.java – класс-контейнер загруженных модулей;
20. laboratory/core/runner/GARunner.java – класс управляющий работой генетического алгоритма;
21. laboratory/core/util/RunnableLock.java – реализация многопоточной блокировки потока исполнения;
22. laboratory/core/InterfaceConfig.java – класс-контейнер данных из конфигурационных файлов;
23. laboratory/core/Main.java – входная точка;
24. laboratory/core/SelectedPlugin.java – класс-контейнер содержащий индексы выбранных подключаемых модулей.

1.1. Пакет CORE.GUI

1.1.1. laboratory/core/gui/chooser/ChooseDialog.java

```
package laboratory.core.gui.chooser;

import laboratory.core.InterfaceConfig;
import laboratory.core.SelectedPlugin;
import laboratory.core.loader.PluginLoader;
import laboratory.util.Parser;

import javax.swing.*;
import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;
import javax.swing.table.AbstractTableModel;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

/**
 * @author Dmitry Sokolov
 */

public class ChooseDialog extends JDialog implements ListSelectionListener, ActionListener {

    private static final String[] COLNAME = new String[2];

    private PluginLoader pluginLoader;

    private Label comment;

    private JTable tb;

    private JButton ok;
    private JButton config;

    private int objectIndex;

    public ChooseDialog(Frame owner, PluginLoader pluginLoader, int objectIndex) {
        super(owner, true);

        this.pluginLoader = pluginLoader;
        this.objectIndex = objectIndex;
        Parser properties = InterfaceConfig.getChooseDialogProperties();

        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
        setLocation((screenSize.width - properties.getInt("width")) / 2, (screenSize.height -
        properties.getInt("height")) / 2 - 30);

        ok = new JButton(properties.getString("button.ok"));
        ok.addActionListener(this);

        config = new JButton(properties.getString("button.config"));
```

```

config.addActionListener(this);

COLNAME[0] = properties.getString("column.1");
COLNAME[1] = properties.getString("column.2");

PluginTableModel tm = new PluginTableModel();
tb = new JTable(tm);
tb.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
tb.getColumnModel().getColumn(0).setMinWidth(properties.getInt("count.column.min"));
tb.getColumnModel().getColumn(0).setMaxWidth(properties.getInt("count.column.max"));
tb.setPreferredSize(new Dimension(properties.getInt("width"),
Math.min(pluginLoader.getCount(objectIndex) * tb.getRowHeight(),
properties.getInt("table.height"))));
ListSelectionModel sM = tb.getSelectionModel();
sM.addListSelectionListener(this);
JScrollPane scr = new JScrollPane(tb);

comment = new Label();
setComment(tm.selected);

getContentPane().setLayout(new BorderLayout());
getContentPane().add(scr, BorderLayout.NORTH);
getContentPane().add(comment, BorderLayout.CENTER);
JPanel buttons = new JPanel();
buttons.add(ok);
buttons.add(config);
getContentPane().add(buttons, BorderLayout.SOUTH);

setSize(properties.getInt("width"), properties.getInt("height"));
setVisible(true);
}

private class PluginTableModel extends AbstractTableModel {

int selected;

public PluginTableModel() {
selected = SelectedPlugin.getIndex(objectIndex);
if (objectIndex == PluginLoader.INDEX_EMULATOR) {

config.setEnabled(pluginLoader.getEmulatorProperties(SelectedPlugin.getIndexIndividual(),
selected) != null);
} else {
config.setEnabled(pluginLoader.getProperties(objectIndex, selected) != null);
}
}

public int getColumnCount() {
return 2;
}
}

```

```
public boolean isCellEditable(int rowIndex, int columnIndex) {
    return columnIndex == 0;
}

public Class<?> getColumnClass(int columnIndex) {
    if (columnIndex == 0) return Boolean.class;
    return String.class;
}

public void setValueAt(Object value, int rowIndex, int columnIndex) {
    if (columnIndex == 0) {
        selected = rowIndex;
        if (objectIndex == PluginLoader.INDEX_EMULATOR) {

config.setEnabled(pluginLoader.getEmulatorProperties(SelectedPlugin.getIndexIndividual(),
selected) != null);
        } else {
            config.setEnabled(pluginLoader.getProperties(objectIndex, selected) != null);
        }
    }
    fireTableDataChanged();
}

protected boolean getSelected(int i){
    return selected == i;
}

public int getRowCount() {
    if (objectIndex == PluginLoader.INDEX_EMULATOR) {
        return pluginLoader.getCountEmulator(SelectedPlugin.getIndexIndividual());
    } else {
        return pluginLoader.getCount(objectIndex);
    }
}

public String getColumnName(int columnIndex) {
    return COLNAME[columnIndex];
}

public Object getValueAt(int rowIndex, int columnIndex) {
    if (columnIndex == 1) {
        if (objectIndex == PluginLoader.INDEX_EMULATOR) {
            return pluginLoader.getEmulatorName(rowIndex,
SelectedPlugin.getIndexIndividual());
        } else {
            return pluginLoader.getName(rowIndex, objectIndex);
        }
    }
    return getSelected(rowIndex);
}
}
```



```
private void setPlugin(ListSelectionModel sM) {
    int i = sM.getMinSelectionIndex();
    if ((i < 0) || (i >= tb.getModel().getRowCount())) return;
    setComment(i);
}

private void setComment(int i) {
    if (objectIndex == PluginLoader.INDEX_EMULATOR) {
        comment.setText(pluginLoader.getEmulatorComment(i,
            SelectedPlugin.getIndexIndividual()));
    } else {
        comment.setText(pluginLoader.getComment(i, objectIndex));
    }
}

public void valueChanged(ListSelectionEvent e) {
    ListSelectionModel selectionModel = (ListSelectionModel) e.getSource();
    setPlugin(selectionModel);
}

public void actionPerformed(ActionEvent e) {
    if (e.getSource() == ok) {
        PluginTableModel tm = (PluginTableModel) tb.getModel();
        SelectedPlugin.setIndex(tm.selected, objectIndex);
        dispose();
    } else if (e.getSource() == config) {
        PluginTableModel tm = (PluginTableModel) tb.getModel();
        if (objectIndex == PluginLoader.INDEX_EMULATOR) {
            new ConfigDialog(this,
                pluginLoader.getEmulatorProperties(SelectedPlugin.getIndexIndividual(), tm.selected));
        } else {
            new ConfigDialog(this, pluginLoader.getProperties(objectIndex, tm.selected));
        }
    }
}
}
```

1.1.2. laboratory/core/gui/chooser/ConfigDialog.java

```
package laboratory.core.gui.chooser;

import laboratory.core.InterfaceConfig;
import laboratory.util.Parser;

import javax.swing.*;
import javax.swing.table.AbstractTableModel;
import java.util.Properties;
import java.awt.*;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

/**
```

```
* @author Andrey Davydov  
*/
```

```
public class ConfigDialog extends JDialog implements ActionListener {  
  
    private static final String[] COLNAME = new String[2];  
  
    public ConfigDialog(JDialog owner, Properties properties) {  
        super(owner, true);  
  
        getContentPane().setLayout(new BorderLayout());  
        Parser p = InterfaceConfig.getConfigDialogProperties();  
  
        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();  
        setLocation((screenSize.width - p.getInt("width")) / 2, (screenSize.height - p.getInt("height")) /  
        2 - 30);  
  
        COLNAME[0] = p.getString("column.1");  
        COLNAME[1] = p.getString("column.2");  
  
        ConfigTableModel tm = new ConfigTableModel(properties);  
        JTable tb = new JTable(tm);  
        tb.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);  
        tb.setPreferredScrollableViewportSize(new Dimension(p.getInt("width"),  
        Math.min(properties.size() * tb.getRowHeight(),  
        p.getInt("table.height"))));  
        JScrollPane scr = new JScrollPane(tb);  
        getContentPane().add(scr, BorderLayout.NORTH);  
  
        JButton b = new JButton(p.getString("button.ok"));  
        b.addActionListener(this);  
        getContentPane().add(b, BorderLayout.SOUTH);  
  
        setSize(p.getInt("width"), p.getInt("height"));  
        setVisible(true);  
    }  
  
    private static class ConfigTableModel extends AbstractTableModel {  
  
        private Properties properties;  
        private String[] names;  
  
        public ConfigTableModel(Properties properties) {  
            this.properties = properties;  
            Object[] keys = properties.keySet().toArray();  
            names = new String[keys.length];  
            for(int i = 0; i < keys.length; i++) {  
                names[i] = keys[i].toString();  
            }  
        }  
  
        public int getRowCount() {  
            return properties.size();  
        }  
    }  
}
```

```
}

public int getColumnCount() {
    return 2;
}

public String getColumnName(int columnIndex) {
    return COLNAME[columnIndex];
}

public Object getValueAt(int rowIndex, int columnIndex) {
    if (columnIndex == 0) {
        return names[rowIndex];
    } else {
        return properties.getProperty(names[rowIndex]);
    }
}

public boolean isCellEditable(int rowIndex, int columnIndex) {
    return columnIndex == 1;
}

public Class<?> getColumnClass(int columnIndex) {
    return String.class;
}

public void setValueAt(Object value, int rowIndex, int columnIndex) {
    if (columnIndex == 1) {
        properties.setProperty(names[rowIndex], value.toString());
    }
    fireTableDataChanged();
}

public void actionPerformed(ActionEvent e) {
    dispose();
}
}
```

1.1.3. laboratory/core/gui/main/actions/PauseAction.java

```
package laboratory.core.gui.main.actions;

import laboratory.core.runner.GARunner;

import javax.swing.*;
import java.awt.event.ActionEvent;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

/**
 * @author Dmitry Sokolov
```

```
*/
public class PauseAction extends AbstractAction {

    private GARunner runner;

    private Action newAction;

    private Lock lock;

    public PauseAction(GARunner runner, String title, String descr){
        this.runner = runner;
        newAction = null;
        lock = new ReentrantLock();
        lock.lock();
        putValue(NAME, title);
        putValue(MNEMONIC_KEY, new Integer('p'));
        putValue(SHORT_DESCRIPTION, descr);
    }

    public void setNewAction(Action action) {
        newAction = action;
    }

    public Lock getLock() {
        return lock;
    }

    public void actionPerformed(ActionEvent e) {
        if (e.getSource() instanceof AbstractButton) {
            AbstractButton temp = (AbstractButton) e.getSource();
            lock.lock();
            runner.pause();
            if (newAction != null) temp.setAction(newAction);
        }
    }
}
}
```

1.1.4. laboratory/core/gui/main/actions/ResumeAction.java

```
package laboratory.core.gui.main.actions;

import laboratory.core.runner.GARunner;

import javax.swing.*;
import java.awt.event.ActionEvent;
import java.util.concurrent.locks.Lock;

/**
 * @author Dmitry Sokolov
 */
public class ResumeAction extends AbstractAction{

    private GARunner runner;
```

```
private Action newAction;

private Lock lock;

public ResumeAction(GARunner runner, Lock lock, String title, String description){
    this.runner = runner;
    newAction = null;
    this.lock = lock;
    putValue(NAME, title);
    putValue(MNEMONIC_KEY, new Integer('r'));
    putValue(SHORT_DESCRIPTION, description);
}

public void setNewAction(Action action) {
    newAction = action;
}

public void actionPerformed(ActionEvent e) {
    if (e.getSource() instanceof AbstractButton) {
        AbstractButton temp = (AbstractButton) e.getSource();
        runner.resume();
        lock.unlock();
        if(newAction != null)temp.setAction(newAction);
    }
}
}
```

1.1.5. laboratory/core/gui/main/actions/TimerListener.java

```
package laboratory.core.gui.main.actions;

import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class TimerListener implements ActionListener {

    private JFrame frame;

    public TimerListener(JFrame frame) {
        this.frame = frame;
    }

    public void actionPerformed(ActionEvent e) {
        frame.repaint();
    }
}
```

1.1.6. laboratory/core/gui/main/actions/ChooseEmulatorAction.java

```
package laboratory.core.gui.main.menu.actions;
```

```

import laboratory.core.gui.chooser.ChooseDialog;
import laboratory.core.loader.PluginLoader;
import laboratory.core.SelectedPlugin;
import laboratory.core.InterfaceConfig;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;

/**
 * @author Dmitry Sokolov
 */
public class ChooseEmulatorAction extends AbstractAction {

    private PluginLoader pluginLoader;
    private Frame f;

    public ChooseEmulatorAction(Frame f, PluginLoader pluginLoader) {
        this.pluginLoader = pluginLoader;

        this.f = f;
        putValue(NAME, InterfaceConfig.getMenuProperties().getString("settings.emulator"));
    }

    public void actionPerformed(ActionEvent e) {
        if (pluginLoader.getCountEmulator(SelectedPlugin.getIndexIndividual()) == 0) {
            JOptionPane.showMessageDialog(null,
                InterfaceConfig.getWarnings().getString("emulator") + " " +
                pluginLoader.getName(SelectedPlugin.getIndexIndividual()),
                PluginLoader.INDEX_INDIVIDUAL);
        } else {
            new ChooseDialog(f, pluginLoader, PluginLoader.INDEX_EMULATOR);
        }
    }
}

```

1.1.7. laboratory/core/gui/main/actions/ChooseGAAction.java

```

package laboratory.core.gui.main.menu.actions;

import laboratory.common.ga.GA;
import laboratory.core.SelectedPlugin;
import laboratory.core.InterfaceConfig;
import laboratory.core.gui.chooser.ChooseDialog;
import laboratory.core.loader.PluginLoader;
import laboratory.core.runner.GARunner;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;

/**

```

```
* @author Dmitry Sokolov
*/
public class ChooseGAAction extends AbstractAction {

    private PluginLoader pluginLoader;
    private GARunner runner;
    private Frame f;

    public ChooseGAAction(Frame f, GARunner runner, PluginLoader pluginLoader) {
        this.pluginLoader = pluginLoader;

        this.runner = runner;
        this.f = f;
        putValue(NAME, InterfaceConfig.getMenuProperties().getString("settings.ga"));
    }

    public void actionPerformed(ActionEvent e) {
        new ChooseDialog(f, pluginLoader, PluginLoader.INDEX_GA);
        try {
            GA alg = pluginLoader.getGA(SelectedPlugin.getIndexGA(),
                pluginLoader.getIndividualFactory(SelectedPlugin.getIndexIndividual()));
            if (runner.setGA(alg) == -1) {
                SelectedPlugin.reDoGA();
                JOptionPane.showMessageDialog(null, InterfaceConfig.getWarnings().getString("ga"));
            }
        } catch (Exception t) {
            t.printStackTrace();
            throw new RuntimeException();
        }
    }
}
```

1.1.8. laboratory/core/gui/main/actions/ChooseIndividualAction.java

```
package laboratory.core.gui.main.menu.actions;

import laboratory.common.ga.GA;
import laboratory.core.SelectedPlugin;
import laboratory.core.InterfaceConfig;
import laboratory.core.gui.chooser.ChooseDialog;
import laboratory.core.gui.main.MainFrame;
import laboratory.core.loader.PluginLoader;
import laboratory.core.runner.GARunner;

import javax.swing.*;
import java.awt.event.ActionEvent;

/**
 * @author Dmitry Sokolov
 */
public class ChooseIndividualAction extends AbstractAction {

    private PluginLoader pluginLoader;
```

```

private GARunner runner;
private MainFrame f;

public ChooseIndividualAction(MainFrame f, GARunner runner, PluginLoader pluginLoader) {
    this.pluginLoader = pluginLoader;

    this.runner = runner;
    this.f = f;
    putValue(NAME, InterfaceConfig.getMenuProperties().getString("settings.individual"));
}

public void actionPerformed(ActionEvent e) {
    new ChooseDialog(f, pluginLoader, PluginLoader.INDEX_INDIVIDUAL);
    try {
        GA alg = pluginLoader.getGA(SelectedPlugin.getIndexGA(),
            pluginLoader.getIndividualFactory(SelectedPlugin.getIndexIndividual()));
        if (runner.setGA(alg) == -1) {
            SelectedPlugin.reDoIndividual();
            JOptionPane.showMessageDialog(null, InterfaceConfig.getWarnings().getString("ga"));
        } else {
            f.changeGraphicPanels();
        }
    } catch (Exception t) {
        t.printStackTrace();
        throw new RuntimeException();
    }
}
}
}

```

1.1.9. laboratory/core/gui/main/menu/actions/ExitAction.java

```

package laboratory.core.gui.main.menu.actions;

import laboratory.core.InterfaceConfig;

import javax.swing.*;
import java.awt.event.ActionEvent;

/**
 * @author Dmitry Sokolov
 */
public class ExitAction extends AbstractAction {

    public ExitAction() {
        super(InterfaceConfig.getMenuProperties().getString("file.exit"));
        putValue(SHORT_DESCRIPTION,
            InterfaceConfig.getMenuProperties().getString("file.exit.description"));
    }

    public void actionPerformed(ActionEvent e) {
        int res = JOptionPane.showConfirmDialog(null,
            InterfaceConfig.getWarnings().getString("close"));
    }
}

```



```
        if (res == 0) System.exit(0);
    }
}
```

1.1.10. laboratory/core/gui/main/menu/actions/SaveGenerationAction.java

```
package laboratory.core.gui.main.menu.actions;
```

```
import laboratory.common.ga.Individual;
import laboratory.core.runner.GARunner;
import laboratory.core.InterfaceConfig;
import laboratory.util.Parser;
```

```
import javax.swing.*;
import java.awt.event.ActionEvent;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.List;
```

```
/**
```

```
 * @author Dmitry Sokolov
 */
```

```
public class SaveGenerationAction extends AbstractAction {
```

```
    private GARunner runner;
```

```
    public SaveGenerationAction(GARunner runner) {
        Parser properties = InterfaceConfig.getMenuProperties();
        putValue(NAME, properties.getString("individual.save"));
        putValue(SHORT_DESCRIPTION, properties.getString("individual.save.description"));
        this.runner = runner;
    }
```

```
    public void actionPerformed(ActionEvent e) {
        List<Individual> generation = runner.getGeneration();
        if (generation == null) {
            JOptionPane.showMessageDialog(null, InterfaceConfig.getWarnings().getString("save"));
            return;
        }
        int top = 1;
        try {
            File dir = new File("results");
            dir.mkdir();
            for (File f : dir.listFiles()) {
                f.delete();
            }
            for (Individual i : generation) {
                PrintWriter out = new PrintWriter(new FileWriter(new File(dir,
                    (int) i.fitness() + "_" + top + ".ind")));
                out.print(i);
            }
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
}
```

```

        out.close();
        if (top > 50) break;
        top++;
    }
} catch (IOException t) {
    t.printStackTrace();
    throw new RuntimeException();
}
}
}
}

```

1.1.11. laboratory/core/gui/main/menu/actions/SavePictureGenerationAction.java

```

import laboratory.core.gui.main.MainFrame;
import laboratory.core.InterfaceConfig;
import laboratory.util.Parser;

import javax.imageio.ImageIO;
import javax.swing.*;
import javax.swing.filechooser.FileNameExtensionFilter;
import java.awt.event.ActionEvent;
import java.io.File;

public class SavePictureAction extends AbstractAction {

    private JFileChooser f;

    private MainFrame mainComponent;

    public SavePictureAction(MainFrame mainComponent) {
        Parser properties = InterfaceConfig.getMenuProperties();
        putValue(NAME, properties.getString("file.save"));
        putValue(SHORT_DESCRIPTION, properties.getString("file.save.description"));
        f = new JFileChooser();
        FileNameExtensionFilter filter = new FileNameExtensionFilter(
            "PNG Images", "png");

        f.setFileFilter(filter);
        this.mainComponent = mainComponent;
    }

    public void actionPerformed(ActionEvent e) {
        int r = f.showSaveDialog(mainComponent);
        if (r == JFileChooser.APPROVE_OPTION) {
            File file = f.getSelectedFile();
            File trash = new File(file.getAbsolutePath() + ".png");
            try {
                ImageIO.write(mainComponent.imagePanel(), "png", trash);
            } catch (Exception z) {
                z.printStackTrace();
                throw new RuntimeException();
            }
        }
    }
}

```

```
}  
}  
}
```

1.1.12. laboratory/core/gui/main/menu/actions/ShowBestAction.java

```
package laboratory.core.gui.main.menu.actions;
```

```
import laboratory.common.ga.Individual;  
import laboratory.core.gui.shower.IndividualDialog;  
import laboratory.core.loader.PluginLoader;  
import laboratory.core.runner.GARunner;  
import laboratory.core.InterfaceConfig;  
import laboratory.util.Parser;
```

```
import javax.swing.*;  
import java.awt.event.ActionEvent;  
import java.util.List;
```

```
/**
```

```
 * @author Dmitry Sokolov
```

```
 */
```

```
public class ShowBestAction extends AbstractAction {
```

```
    private GARunner runner;  
    private PluginLoader load;
```

```
    public ShowBestAction(GARunner runner, PluginLoader load){  
        Parser properties = InterfaceConfig.getMenuProperties();  
        putValue(MNEMONIC_KEY, new Integer('b'));  
        putValue(NAME, properties.getString("individual.best"));  
        putValue(SHORT_DESCRIPTION, properties.getString("individual.best.description"));  
        this.runner = runner;  
        this.load = load;  
    }
```

```
    public void actionPerformed(ActionEvent e) {  
        List<Individual> gen = runner.getBest();  
        List<Integer> a = runner.getNumberGen();  
        new IndividualDialog(gen, a, load);  
    }  
}
```

1.1.13. laboratory/core/gui/main/menu/actions/ShowGenerationAction.java

```
package laboratory.core.gui.main.menu.actions;
```

```
import laboratory.common.ga.Individual;  
import laboratory.core.gui.shower.IndividualDialog;  
import laboratory.core.loader.PluginLoader;  
import laboratory.core.runner.GARunner;  
import laboratory.core.InterfaceConfig;  
import laboratory.util.Parser;
```

```
import javax.swing.*;
import java.awt.event.ActionEvent;
import java.util.LinkedList;
import java.util.List;

public class ShowGenerationAction extends AbstractAction {

    private GARunner runner;
    private PluginLoader load;

    public ShowGenerationAction(GARunner runner, PluginLoader load) {
        Parser properties = InterfaceConfig.getMenuProperties();
        putValue(NAME, properties.getString("individual.current"));
        putValue(SHORT_DESCRIPTION, properties.getString("individual.current.description"));
        this.runner = runner;
        this.load = load;
    }

    public void actionPerformed(ActionEvent e) {
        List<Individual> gen = runner.getGeneration();
        if (gen == null) {
            JOptionPane.showMessageDialog(null, "This genetic hadn't been run.");
            return;
        }
        int curGen = runner.getCurrentGeneration();
        LinkedList<Integer> a = new LinkedList<Integer>();
        for (int i = 0; i < gen.size(); i++) {
            a.addLast(curGen);
        }
        new IndividualDialog(gen, a, load);
    }
}
```

1.1.14. laboratory/core/gui/main/menu/MainMenu.java

```
package laboratory.core.gui.main.menu;

import laboratory.core.gui.main.MainFrame;
import laboratory.core.gui.main.menu.actions.*;
import laboratory.core.loader.PluginLoader;
import laboratory.core.runner.GARunner;
import laboratory.core.InterfaceConfig;
import laboratory.util.Parser;

import javax.swing.*;

/**
 * @author Dmitry Sokolov
 */
public class MainMenu extends JMenuBar {

    public MainMenu(GARunner runner, PluginLoader load, MainFrame frame) {
```

```
Parser properties = InterfaceConfig.getMenuProperties();
JMenu file = new JMenu(properties.getString("file"));

JMenuItem close = new JMenuItem();
close.setAction(new ExitAction());

JMenuItem save = new JMenuItem();
save.setAction(new SavePictureAction(frame));

file.add(save);
file.add(close);

JMenu watch = new JMenu(properties.getString("generation"));

JMenuItem showGeneration = new JMenuItem();
showGeneration.setAction(new ShowGenerationAction(runner, load));
JMenuItem showBest = new JMenuItem();
showBest.setAction(new ShowBestAction(runner, load));
JMenuItem saveGeneration = new JMenuItem();
saveGeneration.setAction(new SaveGenerationAction(runner));
watch.add(saveGeneration);
watch.add(showGeneration);
watch.add(showBest);

JMenu settings = new JMenu(properties.getString("settings"));

JMenuItem ga = new JMenuItem();
ga.setAction(new ChooseGAAction(frame, runner, load));
JMenuItem ind = new JMenuItem();
ind.setAction(new ChooseIndividualAction(frame, runner, load));
JMenuItem em = new JMenuItem();
em.setAction(new ChooseEmulatorAction(frame, load));
settings.add(ga);
settings.add(ind);
settings.add(em);

add(file);
add(watch);
add(settings);
}
}
```

1.1.15. laboratory/core/gui/main/MainFrame.java

```
package laboratory.core.gui.main;

import laboratory.common.functor.Functor;
import laboratory.core.InterfaceConfig;
import laboratory.core.SelectedPlugin;
import laboratory.core.gui.main.actions.PauseAction;
import laboratory.core.gui.main.actions.ResumeAction;
import laboratory.core.gui.main.actions.TimerListener;
import laboratory.core.gui.main.menu.MainMenu;
```

```
import laboratory.core.loader.PluginLoader;
import laboratory.core.runner.GARunner;
import laboratory.util.gui.GraphicPanel;
import laboratory.util.Parser;

import javax.swing.*;
import java.awt.*;
import java.awt.image.BufferedImage;

/**
 * @author Dmitry Sokolov
 * <p/>
 * Main frame of application.
 */
public class MainFrame extends JFrame{

    private JTabbedPane tp;

    private GARunner runner;
    private PluginLoader pluginLoader;

    public MainFrame(String title, GARunner runner, PluginLoader pluginLoader){
        super(title);

        this.runner = runner;
        this.pluginLoader = pluginLoader;
        Parser mf = InterfaceConfig.getMainFrameProperties();

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setMinimumSize(new Dimension(mf.getInt("minimum.width"),
mf.getInt("minimum.height")));

        tp = new JTabbedPane();
        changeGraphicPanels();

        add(tp, BorderLayout.CENTER);

        JPanel p = new JPanel(new BorderLayout());

        PauseAction pause = new PauseAction(runner, mf.getString("button.pause"),
mf.getString("description.pause"));
        ResumeAction resume = new ResumeAction(runner, pause.getLock(),
mf.getString("button.resume"), mf.getString("description.resume"));
        runner.setLock(pause.getLock());
        pause.setNewAction(resume);
        resume.setNewAction(pause);
        JButton pr = new JButton();
        pr.setAction(resume);
        p.add(pr);

        add(p, BorderLayout.SOUTH);
```

```

setJMenuBar(new MainMenu(runner, pluginLoader, this));

Timer timer = new Timer(mf.getInt("timer"), new TimerListener(this));
timer.start();
setSize(new Dimension(mf.getInt("width"), mf.getInt("height")));
setVisible(true);
}

public BufferedImage imagePanel() {
    Component temp = tp.getSelectedComponent();
    BufferedImage res = new BufferedImage(temp.getWidth(), temp.getHeight(),
    BufferedImage.TYPE_INT_RGB);
    Graphics2D g = res.createGraphics();
    temp.paint(g);
    g.dispose();
    return res;
}

public void changeGraphicPanels() {
    Functor[] functors = runner.getFunctors();
    tp.removeAll();
    int j = 0;
    for (Functor functor : functors) {
        int i = SelectedPlugin.getIndexIndividual();
        tp.addTab(pluginLoader.getFunctorTitle(j), new GraphicPanel(functor,
        InterfaceConfig.getGraphicPanelProperties(),
        pluginLoader.getIndividualMax(i), pluginLoader.getIndividualCountSigns(i)));
        j++;
    }
}
}
}

```

1.1.16. laboratory/core/gui/shower/utills/ShowTableModel.java

```

package laboratory.core.gui.shower.utills;

import laboratory.common.ga.Individual;
import laboratory.util.Parser;
import laboratory.core.InterfaceConfig;

import javax.swing.table.AbstractTableModel;
import java.util.List;

/**
 * @author Dmitry Sokolov
 */
public class ShowTableModel extends AbstractTableModel {

    private static final String[] COLNAME = new String[4];

    private boolean[] selected;

```

```
private Individual[] generation;
private int[] numberGen;

public ShowTableModel(List<Individual> generation, List<Integer> numberGen) {
    selected = new boolean[generation.size()];
    this.generation = new Individual[generation.size()];
    this.numberGen = new int[generation.size()];
    int top = 0;
    Parser p = InterfaceConfig.getGenerationFrameProperties();
    for(int i = 0; i < 4; i ++){
        COLNAME[i] = p.getString("colname." + (i + 1));
    }
    for (Individual ind : generation) {
        this.generation[top] = ind;
        top++;
    }
    top = 0;
    for (int i : numberGen) {
        this.numberGen[top] = i;
        top++;
    }
}

public int getColumnCount() {
    return 4;
}

public String getColumnName(int columnIndex) {
    return COLNAME[columnIndex];
}

public int getRowCount() {
    return selected.length;
}

public boolean isCellEditable(int rowIndex, int columnIndex) {
    return columnIndex == 0;
}

public Class<?> getColumnClass(int columnIndex) {
    if (columnIndex == 0) return Boolean.class;
    if (columnIndex == 3) return Integer.class;
    return Double.class;
}

public void setValueAt(Object value, int rowIndex, int columnIndex) {
    if (columnIndex == 0) {
        for (int i = 0; i < selected.length; i++) {
            selected[i] = false;
        }
        selected[rowIndex] = true;
    }
}
```



```

    }
    fireTableDataChanged();
}

public Object getValueAt(int rowIndex, int columnIndex) {
    if (columnIndex == 1) {
        return rowIndex + 1;
    }
    if (columnIndex == 2) {
        return generation[rowIndex].fitness();
    }
    if (columnIndex == 3) {
        return numberGen[rowIndex];
    }
    return selected[rowIndex];
}
}

```

1.1.17. laboratory/core/gui/shower/IndividualDialog.java

```

package laboratory.core.gui.shower;

import laboratory.common.ga.Individual;
import laboratory.common.Visualizable;
import laboratory.core.SelectedPlugin;
import laboratory.core.InterfaceConfig;
import laboratory.core.gui.shower.utils.ShowTableModel;
import laboratory.core.loader.PluginLoader;
import laboratory.util.Parser;

import javax.swing.*;
import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.List;

/**
 * @author Dmitry Sokolov
 * <p/>
 * Dialog for preview individuals.
 */
public class IndividualDialog extends JFrame implements ListSelectionListener, ActionListener {

    private List<Individual> generation;

    private int indexSelected;

    private PluginLoader pluginLoader;

    public IndividualDialog(List<Individual> gen, List<Integer> numberGen, PluginLoader
        pluginLoader) {

```

```
generation = gen;
indexSelected = -1;
this.pluginLoader = pluginLoader;
```

```
Parser p = InterfaceConfig.getGenerationFrameProperties();
Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
setLocation((screenSize.width - p.getInt("width")) / 2, (screenSize.height - p.getInt("height")) / 2);
```

```
ShowTableModel tm = new ShowTableModel(gen, numberGen);
JTable tb = new JTable(tm);
tb.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
tb.getColumnModel().getColumn(0).setMinWidth(p.getInt("count.column.min"));
tb.getColumnModel().getColumn(0).setMaxWidth(p.getInt("count.column.max"));
tb.setPreferredScrollableViewportSize(new Dimension(p.getInt("width"),
Math.min(generation.size() * tb.getRowHeight(), p.getInt("table.height"))));
ListSelectionModel sM = tb.getSelectionModel();
sM.addListSelectionListener(this);
JScrollPane scr = new JScrollPane(tb);
```

```
JButton ok = new JButton("OK");
ok.addActionListener(this);
```

```
getContentPane().setLayout(new BorderLayout());
getContentPane().add(scr, BorderLayout.NORTH);
getContentPane().add(ok, BorderLayout.SOUTH);
```

```
setSize(p.getInt("width"), p.getInt("height"));
setVisible(true);
```

```
}
```

```
public void valueChanged(ListSelectionEvent e) {
    ListSelectionModel selectionModel = (ListSelectionModel) e.getSource();
    int i = selectionModel.getMinSelectionIndex();
    if ((i < 0) || (i >= generation.size())) return;
    indexSelected = i;
}
```

```
public void actionPerformed(ActionEvent e) {
    if (indexSelected != -1) {
        try {
            int ii = SelectedPlugin.getIndexIndividual();
            if (pluginLoader.getCountEmulator(ii) > SelectedPlugin.getIndexEmulator()) {
                pluginLoader.getEmulator(ii, SelectedPlugin.getIndexEmulator(), (Visualizable)
generation.get(indexSelected));
            } else {
                JOptionPane.showMessageDialog(null,
InterfaceConfig.getWarnings().getString("emulator") + pluginLoader.getName(ii,
PluginLoader.INDEX_INDIVIDUAL));
            }
        }
    }
}
```

```
        } catch (Exception z) {
            JOptionPane.showConfirmDialog(null,
                InterfaceConfig.getWarnings().getString("open"));
            z.printStackTrace();
        }
    }
    dispose();
}

public int getSelectedIndex() {
    return indexSelected;
}

}
```

1.1.18. laboratory/core/gui/FrameCreator.java

```
package laboratory.core.gui;

import laboratory.core.gui.main.MainFrame;
import laboratory.core.loader.PluginLoader;
import laboratory.core.runner.GARunner;
import laboratory.core.InterfaceConfig;

import java.util.concurrent.locks.Lock;

/**
 * @author Dmitry Sokolov
 * <p/>
 * This class creates main frame of application.
 */
public class FrameCreator implements Runnable {

    private GARunner runner;
    private Lock lock;
    private PluginLoader pluginLoader;

    public FrameCreator(GARunner runner, PluginLoader pluginLoader, Lock lock) {
        this.runner = runner;
        this.lock = lock;
        this.pluginLoader = pluginLoader;
    }

    public void run() {
        new MainFrame(InterfaceConfig.getMainFrameProperties().getString("title"), runner,
            pluginLoader);
        lock.unlock();
    }
}
```

1.2. Пакет CORE

1.2.1. laboratory/core/loader/PluginLoader.java

```
package laboratory.core.loader;

import laboratory.common.Loader;
import laboratory.common.Visualizable;
import laboratory.common.functor.Functor;
import laboratory.common.ga.GA;
import laboratory.common.ga.Individual;
import laboratory.common.ga.IndividualFactory;
import laboratory.core.InterfaceConfig;

import java.awt.*;
import java.io.File;
import java.io.FilenameFilter;
import java.io.IOException;
import java.lang.reflect.InvocationTargetException;
import java.net.URL;
import java.net.URLClassLoader;
import java.util.ArrayList;
import java.util.List;
import java.util.Properties;
import java.util.StringTokenizer;
import java.util.jar.Attributes;
import java.util.jar.JarFile;

/**
 * @author Dmitry Sokolov
 * <p/>
 * This class gets genetic algorithm from specified directory and gives them information about
 * them.
 */
public class PluginLoader {

    private class Plugin {

        public final Loader<?> loader;
        public final String name;
        public String comment;
        public final Properties properties;

        Plugin(String name, String comment, Loader<?> loader, Properties properties) {
            this.loader = loader;
            this.name = name;
            this.comment = comment;
            this.properties = properties;
        }
    }

    private Plugin[][] plugin;
```

```

private Functor[] functors;
private String[] functorTitle;

private List<List<Integer>> emulator;
private double[] individual;
private int[] cs;

public static final int INDEX_GA = 0;
public static final int INDEX_INDIVIDUAL = 1;
public static final int INDEX_EMULATOR = 2;

private Plugin[] load(File dir) throws IOException, NoSuchMethodException,
    ClassNotFoundException, IllegalAccessException, InvocationTargetException,
    InstantiationException {
    FilenameFilter filter = new FilenameFilter() {
        public boolean accept(File dir, String name) {
            return name.endsWith(".jar");
        }
    };
    File[] list = dir.listFiles(filter);
    Plugin[] res = new Plugin[list.length];
    for (int i = 0; i < res.length; i++) {
        JarFile jr = new JarFile(list[i]);
        Attributes attributes = jr.getManifest().getMainAttributes();
        String temp = attributes.getValue(Attributes.Name.EXTENSION_NAME);
        String tmp = attributes.getValue(new Attributes.Name("Comment"));
        if (temp == null) temp = InterfaceConfig.getDefaultName();
        if (tmp == null) tmp = InterfaceConfig.getDefaultComment();
        String mainClass = attributes.getValue(Attributes.Name.MAIN_CLASS);
        ClassLoader cl = new URLClassLoader(new URL[]{list[i].toURI().toURL()});
        Class<Loader<?>> c = (Class<Loader<?>>) cl.loadClass(mainClass);
        Loader<?> loader = c.getConstructor(JarFile.class).newInstance(jr);
        res[i] = new Plugin(temp, tmp, loader, loader.getProperties());
    }
    return res;
}

private Functor[] loadFuncutors(File dir) throws Exception {
    FilenameFilter filter = new FilenameFilter() {
        public boolean accept(File dir, String name) {
            return name.endsWith(".jar");
        }
    };
    File[] list = dir.listFiles(filter);
    Functor[] res = new Functor[list.length];
    functorTitle = new String[list.length];
    for (int i = 0; i < res.length; i++) {
        JarFile jr = new JarFile(list[i]);
        Attributes attributes = jr.getManifest().getMainAttributes();
        String mainClass = attributes.getValue(Attributes.Name.MAIN_CLASS);
        ClassLoader cl = new URLClassLoader(new URL[]{list[i].toURI().toURL()});
        Class<Functor> c = (Class<Functor>) (cl.loadClass(mainClass));
    }
}

```

```

        res[i] = c.getConstructor().newInstance();
        functorTitle[i] = attributes.getValue(new Attributes.Name("Comment"));
    }
    return res;
}

public PluginLoader(File dirGA, File dirIndividual, File dirEmulator, File dirFunctor) throws
Exception {
    plugin = new Plugin[3][];
    plugin[INDEX_GA] = load(dirGA);

    plugin[INDEX_INDIVIDUAL] = load(dirIndividual);
    individual = new double[plugin[INDEX_INDIVIDUAL].length];
    cs = new int[individual.length];
    for (int k = 0; k < individual.length; k++) {
        Plugin ind = plugin[INDEX_INDIVIDUAL][k];
        String s = ind.comment;
        int i = s.indexOf(" ||| ");
        ind.comment = s.substring(i + 5);
        StringTokenizer st = new StringTokenizer(s.substring(0, i));
        individual[k] = Double.parseDouble(st.nextToken());
        cs[k] = Integer.parseInt(st.nextToken());
    }

    plugin[INDEX_EMULATOR] = load(dirEmulator);
    emulator = new ArrayList<List<Integer>>(plugin[INDEX_INDIVIDUAL].length);
    for (int i = 0; i < plugin[INDEX_INDIVIDUAL].length; i++) {
        emulator.add(new ArrayList<Integer>());
    }
    for (int k = 0; k < plugin[INDEX_EMULATOR].length; k++) {
        Plugin e = plugin[INDEX_EMULATOR][k];
        String s = e.comment;
        int i = s.indexOf(" ||| ");
        e.comment = s.substring(i + 5);
        StringTokenizer st = new StringTokenizer(s.substring(0, i));
        while (st.hasMoreTokens()) {
            String name = st.nextToken();
            for (int j = 0; j < plugin[INDEX_INDIVIDUAL].length; j++) {
                if (plugin[INDEX_INDIVIDUAL][j].name.equals(name)) {
                    emulator.get(j).add(k);
                }
            }
        }
    }
    functors = loadFunctors(dirFunctor);
}

private Object getC(int objectIndex, int i, Object... args) throws IOException {
    return plugin[objectIndex][i].loader.load(args);
}

public GA getGA(int i, IndividualFactory fact) throws Exception {

```

```
    return (GA) getC(INDEX_GA, i, fact);
}

public IndividualFactory getIndividualFactory(int i) throws Exception {
    return (IndividualFactory) getC(INDEX_INDIVIDUAL, i);
}

public Frame getEmulator(int individual, int i, Visualizable ind) throws Exception {
    return (Frame) getC(INDEX_EMULATOR, emulator.get(individual).get(i),
        ind.getAttributes());
}

public String getName(int i, int objectIndex) {
    return plugin[objectIndex][i].name;
}

public String getComment(int i, int objectIndex) {
    return plugin[objectIndex][i].comment;
}

public int getCount(int objectIndex) {
    return plugin[objectIndex].length;
}

public int getCountEmulator(int individual) {
    return emulator.get(individual).size();
}

public String getEmulatorName(int i, int individual) {
    return plugin[INDEX_EMULATOR][emulator.get(individual).get(i)].name;
}

public String getEmulatorComment(int i, int individual) {
    return plugin[INDEX_EMULATOR][emulator.get(individual).get(i)].comment;
}

public double getIndividualMax(int i) {
    return individual[i];
}

public int getIndividualCountSigns(int i) {
    return cs[i];
}

public Properties getProperties(int objectIndex, int i){
    return plugin[objectIndex][i].properties;
}

public Properties getEmulatorProperties(int individual, int i) {
    return plugin[INDEX_EMULATOR][emulator.get(individual).get(i)].properties;
}
```

```
public Functor[] getFunctors() {  
    return functors;  
}  
  
public String getFunctorTitle(int i) {  
    return functorTitle[i];  
}  
}
```

1.2.2. laboratory/core/runner/GARunner.java

```
package laboratory.core.runner;  
  
import laboratory.common.functor.Functor;  
import laboratory.common.ga.GA;  
import laboratory.common.ga.Individual;  
  
import java.util.LinkedList;  
import java.util.List;  
import java.util.concurrent.locks.Lock;  
  
/**  
 * @author Dmitry Sokolov  
 */  
public class GARunner implements Runnable {  
  
    private GA alg;  
  
    private boolean stop;  
  
    private boolean pause;  
  
    private Functor[] functors;  
  
    private Lock lock;  
  
    private List<Individual> generation;  
  
    private LinkedList<Individual> best;  
  
    private LinkedList<Integer> numberGen;  
  
    private int generationNumber;  
  
    /**  
     * @param alg genetic algorithm which this runner works with  
     * @param functors functors which will be applied to generation  
     */  
    public GARunner(GA alg, Functor[] functors) {  
        this.alg = alg;  
        this.functors = functors;  
        best = new LinkedList<Individual>();  
        numberGen = new LinkedList<Integer>();  
    }  
}
```



```
    stop = false;
    pause = true;
    generationNumber = 0;
}

/**
 * This method executes algorithm and gives generations to functors.
 */
public void run() {
    while (!stop) {
        if (pause) {
            lock.lock();
            lock.unlock();
        }
        generationNumber++;
        alg.nextGeneration();
        generation = alg.getGeneration();
        if ((best.size() == 0) || (generation.get(0).fitness() > best.getLast().fitness())) {
            best.addLast(generation.get(0));
            numberGen.addLast(generationNumber);
        }
        for (Functor f : functors) {
            f.update(generation);
        }
    }
}

public void tryToInterrupt() {
    stop = true;
}

public void pause() {
    pause = true;
}

public void resume() {
    pause = false;
}

public void setLock(Lock lock) {
    this.lock = lock;
}

public Functor[] getFunctors() {
    return functors;
}

public List<Individual> getGeneration() {
    return generation;
}

public List<Individual> getBest() {
```

```
        return best;
    }

    public List<Integer> getNumberGen() {
        return numberGen;
    }

    public int getCurrentGeneration() {
        return generationNumber;
    }

    public int setGA(GA alg) {
        if (!pause) return -1;
        this.alg = alg;
        for (Functor functor : functors) {
            functor.clear();
        }
        best.clear();
        numberGen.clear();
        generation = null;
        generationNumber = 0;
        return 0;
    }
}
```

1.2.3. laboratory/core/util/RunnableLock.java

```
package laboratory.core.util;

import java.util.concurrent.locks.ReentrantLock;

/**
 * @author Dmitry Sokolov
 */
public class RunnableLock extends ReentrantLock implements Runnable {

    public void run() {
        lock();
    }

}
```

1.2.4. laboratory/core/Main.java

```
package laboratory.core;

import laboratory.core.gui.FrameCreator;
import laboratory.core.loader.PluginLoader;
import laboratory.core.runner.GARunner;
import laboratory.core.util.RunnableLock;

import javax.swing.*;
import java.io.File;
```

```
import java.io.FileReader;
import java.util.Properties;

/**
 * @author Dmitry Sokolov
 */
public class Main {

    private static Properties paths = new Properties();

    public static File getPath(String path) {
        return new File(paths.getProperty(path));
    }

    public static void main(String[] args) throws Exception {
        paths.load(new FileReader("build.properties"));
        PluginLoader pluginLoader = new PluginLoader(getPath("ga.dir"), getPath("individual.dir"),
            getPath("emulator.dir"),
            getPath("functor.dir"));
        GARunner runner = new GARunner(pluginLoader.getGA(SelectedPlugin.getIndexGA()),
            pluginLoader.getIndividualFactory(SelectedPlugin.getIndexIndividual()),
            pluginLoader.getFunctors());
        RunnableLock lock = new RunnableLock();
        try {
            SwingUtilities.invokeAndWait(lock);
        } catch (Exception e) {
            e.printStackTrace();
        }
        SwingUtilities.invokeLater(new FrameCreator(runner, pluginLoader, lock));
        lock.lock();
        lock.unlock();
        runner.run();
    }
}
```

1.2.5. laboratory/core/SelectedPlugin.java

```
package laboratory.core;

import laboratory.core.loader.PluginLoader;

/**
 * @author Dmitry Sokolov
 */
public class SelectedPlugin {

    private static int indexGA = 0;
    private static int previousIndexGA = 0;
    private static int indexEmulator = 0;
    private static int previousIndexEmulator = 0;
    private static int individual = 0;
    private static int previousIndividual = 0;
```

```
public static void setIndexGA(int i) {
    previousIndexGA = indexGA;
    indexGA = i;
    setIndividual(0);
}

public static int getIndexGA() {
    return indexGA;
}

public static void reDoGA() {
    indexGA = previousIndexGA;
    reDoIndividual();
}

public static void setIndividual(int i) {
    previousIndividual = individual;
    individual = i;
    setIndexEmulator(0);
}

public static int getIndexIndividual() {
    return individual;
}

public static void reDoIndividual() {
    individual = previousIndividual;
}

public static void setIndexEmulator(int i) {
    previousIndexEmulator = indexEmulator;
    indexEmulator = i;
}

public static int getIndexEmulator() {
    return indexEmulator;
}

public static void reDoEmulator() {
    indexEmulator = previousIndexEmulator;
}

public static int getIndex(int objectIndex) {
    switch (objectIndex) {
        case PluginLoader.INDEX_EMULATOR:
            return getIndexEmulator();
        case PluginLoader.INDEX_GA:
            return getIndexGA();
        case PluginLoader.INDEX_INDIVIDUAL:
```

```
        return getIndexIndividual();
    default:
        return 111;
    }
}

public static void setIndex(int i, int objectIndex) {
    switch (objectIndex) {
        case PluginLoader.INDEX_EMULATOR:
            setIndexEmulator(i);
            break;
        case PluginLoader.INDEX_GA:
            setIndexGA(i);
            break;
        case PluginLoader.INDEX_INDIVIDUAL:
            setIndividual(i);
            break;
    }
}
}
```

