Инструкция по созданию plugin'ов к виртуальной лаборатории

Давыдов А., Соколов Д., Царев Ф.

13 октября 2010 г.

1 Введение

Все упоминаемые ниже интерфейсы содержатся в файле common.jar, некоторые необязательные, но, возможно, полезные, при написании собственного модуля классы, содержатся в файле util.jar.

2 Общие правила

Создание плагина заключается в реализации соответствующего (*основного*) интерфейса и сборке jar-архива. Для подключения плагина манифест полученного архива должен содержать атрибут Main-Class, который в свою очередь должен ссылаться на класс реализующий один из интерфейсов, наследников PluginLoader.

```
public interface PluginLoader extends Nameable, Descriptable,
        Configurable {
    public Writer getConfigWriter(File file)
            throws FileNotFoundException;
}
public interface Nameable {
    public String getName();
}
public interface Configurable {
    public JDialog getConfigDialog(JFrame owner);
}
public interface Descriptable {
    public String getDescription();
}
```

- getConfigWriter — метод должен сохранять конфигурацию плагина в файл, возвращает открытый поток вывода в записываемый файл;
- getName — метод должен возвращать название плагина;
- getConfigDialog — метод должен возвращать диаловое окно с конфигурационными параметрами плагина.
- getDescription — метод должен возвращать текстовое описание плагина.

Класс, реализующий этот наследника данного интерфейс, должен иметь конструктор, от объекта класса (JarFile) (исключение плагин особи). То есть от того архива, где хранится сам плагин.

3 Плагин, содержащий описание задачи

Основной интерфейс плагина — TaskLoader.

```
public interface TaskLoader extends PluginLoader {
}
```

Данный интерфейс не содержит дополнительных методов.

4 Плагин, содержащий представление особи

Основной интерфейс плагина — IndividualLoader.

```
public interface IndividualLoader<I extends Individual>
        extends PluginLoader {
    public List<IndividualFactory<I>> loadFactories();
    public List<Crossover<I>> loadCrossovers();
    public List<Mutation<I>> loadMutations();
    public List<Fitness<I>> loadFunctions();
    public String getTaskName();

}
```

Данный интерфейс содержит следующие методы:

- loadFactories — метод должен возвращать *фабрику* особей, а именно объект класса, реализующего интерфейс IndividualFactory<I>.

  ```
  public interface IndividualFactory<I extends Individual> {
      public I getIndividual();
  }
  ```

  Единственный метод этого интерфейса должен возвращать случайно сгенерированную особь (интерфейс Individual).

  ```
  public interface Individual {
      public double standardFitness();
  }
  ```

  Метод standardFitness должен возвращать фитнесс-функцию для задачи, котую решает данная особь.

- loadCrossovers — метод должен возвращать список доступных операторов скрещивания.

```
public interface Crossover<I extends Individual> {
    public List<I> apply(List<I> parents);
}
```

Метод apply по списку родительких особей должен возвражать список потомков, отдельно отметим, что особи потомков должны быть **новыми** объектами, это требуется для корректной работы алгоритма.

- loadMutations. loadFunctions — методы аналогичны методу loadCrossovers, должны возвращать список доступных операторов мутации и подсчета фитнесс-функции.

- getTaskName — метод должен возвращать название задачи, которую решает данная особь.

5 Плагин, содержащий схему генетического алгоритма

Основной интерфейс плагина — AlgorithmLoader.

```
public interface AlgorithmLoader<I extends Individual> extends PluginL
    public Algorithm<I> loadAlgorithm(
        List<IndividualFactory<I>> factories, List<Crossover<I>> crosso
        List<Mutation<I>> mutations, List<Selection<I>> sel, List<Fitn
    public String getMessage();
}
```

- loadFactories — метод должен возвращать *схему генетического алгоритма*, а именно объект класса, реализующего интерфейс Algorithm.

```
public interface Algorithm<I extends Individual> {
    public void nextGeneration();
    public List<I> getGeneration();
    public void stop();
}
```

Данный интерфейс содержит следующие методы:

  - nextGeneration — переход к следующему поколению особей.

  - getGeneration — метод должен возвращать текущее поколение особей.
  - stop — в текущей версии не используется.

- getTaskName — метод должен возвращать название задачи, которую решает данная особь.

- getMessage — метод должен возвращать сообщать сообщение о наличии и правильности загрузки опрераторов отбора (Selection).

```
public interface Selection<I extends Individual> {
    public List<FitIndividual<I>> apply(
        List<FitIndividual<I>> population, int n);
}
```

  Единственный метод данного интерфейса должен возвращать список особей, напрямую переходящих в следующее поколение.