**1 ЛАБОРАТОРНАЯ РАБОТА №1**

**ИЗУЧЕНИЕ ДИНАМИЧЕСКОГО ОПРЕДЕЛЕНИЯ ТИПОВ (RTTI)**

**1.1** **Цель работы**

Изучить механизм динамического определения типов, и механизма отражения в Java. Научиться применять возможность динамической загрузки произвольного класса по заданному имени.

**1.2 Краткие теоретические сведения**

В Java вся информация о классе хранится в специальном объекте типа Class. Для каждого класса в программе существует свой объект Class, который создается при загрузке класса, созданного на этапе компиляции и сохраненного на диске в файле с расширением class.

Как получить экземпляр Class, соответствующий данному классу (или

интерфейсу)?

1. Просто добавляем к имени класса суффикс .class, например: Class clazz = String.class; Т.е. используем литерал объекта Сlass.

2. Если мы располагаем экземпляром некоторого класса, может быть даже неизвестного в данной точке программы, можно вызвать метод getClass(), присутствующий в каждом Java-объекте (унаследованный от класса Object).

3. С. Вызываем статический метод класса Class.forName(), который

возвращает экземпляр Class:

Class clazz = Class.forName("Полное\_имя\_класса");

Литералы объектов Class работают с регулярными классами, а также с интерфейсами, массивами и примитивными типами.

Class clazz = byte[].class;

В дополнение, существует стандартное поле называемое TYPE, которое существует для каждого примитивного класса-оболочки. Поле TYPE создает ссылку на объект Class для соответствующего примитивного класса.

В Java встроена возможность динамической загрузки произвольного

класса по заданному имени, и реализуется классом Class.

public static Class forName(String className) throws

ClassNotFoundException;

Метод отыскивает в системе (в пути поиска CLASSPATH) класс с заданным именем className и возвращает соответствующий экземпляр класса Class. Имя класса должно быть полным, т.е. включать имя пакета. Если такой класс отсутствует, возбуждается исключение ClassNotFoundException. После того как получен объект типа Class можно создать экземпляр этого класса.

Class clazz = Class.forName(String className);

Object object = clazz.newInstance();

Для успешного выполнения этого метода у класса должен быть описан конструктор без параметров (конструктор по умолчанию), иначе будет возбуждено исключение InstantiationException.Чтобы работать с таким объектом необходимо знать методы класса, их параметры и возвращаемые значения. Для указания контракта класса используют интерфейсы.

**1.3 Выполнение работы**

**package** laba1;

**import** java.awt.EventQueue;

**import** java.io.File;

**import** java.io.IOException;

**import** java.lang.reflect.Constructor;

**import** java.lang.reflect.Field;

**import** java.lang.reflect.Method;

**import** java.util.Enumeration;

**import** java.util.jar.JarEntry;

**import** java.util.jar.JarFile;

**import** javax.swing.JFrame;

**import** javax.swing.JPanel;

**import** java.awt.BorderLayout;

**import** javax.swing.JTree;

**import** javax.swing.tree.DefaultMutableTreeNode;

**import** javax.swing.tree.DefaultTreeModel;

**import** javax.swing.JTextArea;

**import** javax.swing.JFileChooser;

**import** java.awt.event.ActionListener;

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

**import** javax.swing.JScrollPane;

**import** javax.swing.JSplitPane;

**import** javax.swing.JMenuBar;

**import** javax.swing.JMenuItem;

**import** javax.swing.JMenu;

**import** java.beans.PropertyChangeListener;

**import** java.beans.PropertyChangeEvent;

**public** **class** MainWindow {

**private** JFrame frame;

**private** JarClassLoader loader;

**private** File file;

**private** JarFile jarFile;

**private** JPanel contentPane;

**private** JTree tree;

**private** JTextArea textArea;

/\*\*

\* Launch the application.

\*/

**public** **static** **void** main(String[] args) {

EventQueue.*invokeLater*(**new** Runnable() {

**public** **void** run() {

**try** {

MainWindow window = **new** MainWindow();

window.frame.setVisible(**true**);

} **catch** (Exception e) {

e.printStackTrace();

}

}

});

}

/\*\*

\* Create the application.

\*/

**public** MainWindow() {

initialize();

}

/\*\*

\* Initialize the contents of the frame.

\*/

**private** **void** initialize() {

frame = **new** JFrame();

frame.setBounds(100, 100, 450, 300);

frame.setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);

JSplitPane splitPane = **new** JSplitPane();

frame.getContentPane().add(splitPane, BorderLayout.***CENTER***);

JScrollPane scrollPane = **new** JScrollPane();

splitPane.setRightComponent(scrollPane);

textArea = **new** JTextArea();

scrollPane.setViewportView(textArea);

JScrollPane scrollPane\_1 = **new** JScrollPane();

splitPane.setLeftComponent(scrollPane\_1);

tree = **new** JTree();

tree.addPropertyChangeListener(**new** PropertyChangeListener() {

**public** **void** propertyChange(PropertyChangeEvent arg0) {

onTreeChange();

}

});

scrollPane\_1.setViewportView(tree);

JMenuBar menuBar = **new** JMenuBar();

frame.setJMenuBar(menuBar);

JMenu mnOpen = **new** JMenu("Open");

menuBar.add(mnOpen);

JMenuItem mntmOpenJarfile = **new** JMenuItem("Open jar-file");

mntmOpenJarfile.addActionListener(**new** ActionListener() {

**public** **void** actionPerformed(ActionEvent arg0) {

openJar();

}

});

mnOpen.add(mntmOpenJarfile);

}

**private** String parseToArray(Object[] array, String str) {

StringBuilder builder = **new** StringBuilder();

**for** (**int** i = 0; i < array.length; i++) {

**if** (i == 0)

builder.append(str + "\n");

builder.append(array[i].toString() + "\n");

}

**return** builder.toString();

}

**private** String getClassInfo(String classPath) {

Class clazz;

**try** {

clazz = loader.findClass(classPath);

} **catch** (ClassNotFoundException e) {

e.printStackTrace();

**return** "CRITICAL ERROR";

}

StringBuilder sb = **new** StringBuilder();

sb.append("Class: " + clazz.toGenericString() + "\n");

Class[] interfaces = clazz.getInterfaces();

Field[] fields = clazz.getFields();

Method[] methods = clazz.getMethods();

Constructor[] constructors = clazz.getConstructors();

sb.append(parseToArray(interfaces, "Interfaces: "));

sb.append(parseToArray(fields, "Fields: "));

sb.append(parseToArray(methods, "Methods: "));

sb.append(parseToArray(constructors, "Constructors: "));

**return** sb.toString();

}

**private** **boolean** openJar() {

JFileChooser fileChose = **new** JFileChooser();

fileChose.showOpenDialog(frame);

file = fileChose.getSelectedFile();

**if** (file == **null**)

**return** **false**;

**if** (!createJar())

**return** **false**;

initJar();

createTree();

**return** **true**;

}

**private** **void** openOtherFile(String strName) {

JarEntry entry = jarFile.getJarEntry(strName);

**byte** [] b = **new** **byte**[(**int**) entry.getSize()];

**try** {

jarFile.getInputStream(entry).read(b);

} **catch** (IOException e) {

e.printStackTrace();

}

textArea.setText(**new** String(b));

}

**private** **boolean** createJar() {

**try** {

jarFile = **new** JarFile(file);

} **catch** (IOException e) {

e.printStackTrace();

**return** **false**;

}

**return** **true**;

}

**private** **void** initJar() {

loader = **new** JarClassLoader(jarFile.getName());

Enumeration<JarEntry> itr = jarFile.entries();

**while** (itr.hasMoreElements()) {

textArea.setText(textArea.getText() + itr.nextElement().toString() + "\n");

}

}

**private** **void** createTree() {

DefaultMutableTreeNode root = **new** DefaultMutableTreeNode(jarFile.getName());

DefaultMutableTreeNode start = root;

DefaultMutableTreeNode find;

Enumeration<JarEntry> itr = jarFile.entries();

**while** (itr.hasMoreElements()) {

String[] arrStr = itr.nextElement().toString().split("/");

start = root;

**for** (**int** i = 0; i < arrStr.length; i++) {

find = findNode(start, arrStr[i]);

**if** (find == **null**) {

**for** (**int** j = i; j < arrStr.length; j++) {

DefaultMutableTreeNode node = **new** DefaultMutableTreeNode(arrStr[j]);

start.add(node);

start = node;

}

} **else**

start = find;

}

}

tree.setModel(**new** DefaultTreeModel(root));

}

**private** DefaultMutableTreeNode findNode(DefaultMutableTreeNode start, String s) {

Enumeration itr = start.preorderEnumeration();

**while** (itr.hasMoreElements()) {

DefaultMutableTreeNode node = (DefaultMutableTreeNode) itr.nextElement();

**if** (node.toString().equals(s))

**return** node;

}

**return** **null**;

}

**private** **void** onTreeChange() {

DefaultMutableTreeNode selected = (DefaultMutableTreeNode) tree.getLastSelectedPathComponent();

**if** (selected == **null**) **return**;

String strEnum = createEntryName(selected);

String strClassPath = "";

**if** (isClass(strEnum)){

strClassPath = createNameByEntry(strEnum);

textArea.setText(getClassInfo(strClassPath));

} **else** **if** (isSubClass(strEnum)) {

textArea.setText("sub class: " + strEnum);

} **else** **if** (isOther(strEnum)) {

openOtherFile(strEnum);

} **else** **if** (isDir(strEnum)) {

textArea.setText("Directory: " + strEnum.substring(0, strEnum.length() - 1));

}

**else** {

textArea.setText("File: " + strEnum.substring(0, strEnum.length() - 1));

}

}

**private** String createEntryName (DefaultMutableTreeNode node) {

Enumeration itr = node.pathFromAncestorEnumeration((DefaultMutableTreeNode) tree.getModel().getRoot());

StringBuilder strPath = **new** StringBuilder();

itr.nextElement();

**while** (itr.hasMoreElements()){

DefaultMutableTreeNode dmtNode = (DefaultMutableTreeNode) itr.nextElement();

**if** (dmtNode.isLeaf()){

strPath.append(dmtNode.toString());

} **else**{

strPath.append(dmtNode.toString() + "/");

}

}

**return** strPath.toString();

}

**private** String createNameByEntry(String strEntryName) {

strEntryName = strEntryName.replaceAll("/", ".");

**return** strEntryName.substring(0, strEntryName.length() - 6);

}

**private** **boolean** isDir(String strEnum){

**if** (jarFile.getJarEntry(strEnum) == **null**) **return** **false**;

**return** jarFile.getJarEntry(strEnum).isDirectory();

}

**private** **boolean** isClass(String test){

**return** test.endsWith("class");

}

**private** **boolean** isSubClass(String test){

**return** test.endsWith("class") && test.indexOf("$") >= 0;

}

**private** **boolean** isOther(String test){

**return** test.endsWith(".java") || test.endsWith(".inf") || test.endsWith(".project") ||

test.endsWith(".jpage") || test.endsWith(".txt") || test.endsWith(".MF");

}

}

package laba1;

import java.io.IOException;

import java.io.InputStream;

import java.util.HashMap;

import java.util.Map;

import java.util.jar.JarEntry;

import java.util.jar.JarFile;

/\*\*

\* Клас використовується для створення об'єктів,

\* що дозволяють завантажувати класи із jar-файлу,

\* ім'я якого передаємо через конструктор.

\* Для завантаження класу викликаємо метод findClass(String),

\* до якого передаємо назву класу, як параметр.

\* Роздільниками у назві класу мають бути крапки.

\*/

public class JarClassLoader extends ClassLoader {

private JarFile jar;

private Map<String, Class<?>>loaded = new HashMap<>();

// До конструктору передаємо назву jar-файлу

public JarClassLoader(String jarFileName) {

super(JarClassLoader.class.getClassLoader());

// JarFile objct creation

try {

jar = new JarFile(jarFileName);

} catch (Exception e) {

System.err.println("Open jar file error");

return;

}

}

@Override

public Class<?> findClass(String name) throws ClassNotFoundException {

// name - це повна назва класу у пакеті,

//наприклад "pack1.pack2.MyClass"

// Спроба взяти клас з карти(можливо вже завантажувався)

Class<?>c = loaded.get(name);

if (c != null)

return c;

// Спроба завантажити клас через SystemClassLoader

try {

return findSystemClass(name);

} catch (Exception e) {

}

// Якщо не вийшло, отримуємо байт-код класу із файлу "name"

//і завантажуємо його

byte[] b; //Посилання на масив для байтів класу

// Отримуємо байт код класу,

// текст приватного методу loadClassData наведено далі

b = loadClassData(name);

// Завантажуємо клас

try {

c = defineClass(name, b, 0, b.length);

// Зберігаємо клас у карті

loaded.put(name, c);

} catch (Throwable e) {

throw new ClassNotFoundException(e.getMessage());

}

return c;

}

/\*\*

\* /\*Метод повертає байт код класу name із jar-файлу jar

\*/

private byte[] loadClassData(String name) throws

ClassNotFoundException {

// Формування ім'я entry з імені класу

String entryName = name.replace('.', '/') + ".class";

// Створення об'єкту entry

JarEntry entry = jar.getJarEntry(entryName);

if (entry == null)

throw new ClassNotFoundException(name);

// Створння масиву для байт коду

int size = new Long(entry.getSize()).intValue();

byte[] buf = new byte[size];

// Спроба введення даних

try {

//Метод getInputStream розпаковує файл на який посилається entry

InputStream input = jar.getInputStream(entry);

int count = input.read(buf);

// Контроль кількості введених байтів

if (count<size)

throw new ClassNotFoundException("Error reading class '" + name

+ "' from jar: " + jar.getName());

// Close the input stream

input.close();

} catch (IOException e1) {

throw new ClassNotFoundException(e1.getMessage());

}

// return bytes

return buf;

}

}

**1.4 Выводы**

В ходе лабораторной работы был изучен механизм динамического определения типов и загрузка классов в платформе Java. Рефлексия в Java позволяет получить объект класса Class, с помощью которого можно получить информацию о внутренней структуре класса, также данный механизм позволяет вызывать приватные методы и работать с приватными полями, а также создавать экземпляры класса, что позволяет динамически загружать классы используя имя файла.