Факультет комп’ютерних наук

Залікова робота

«*Крос-платформне програмування*»

**Варіант №1**

П.І.Б. Дібцева Анна Миколаївна

Група КС-21

1. Структура XML файлу. Чим well-formed XML відрізняється від valid? Наведіть приклади well-formed XML та valid XML. (*10 балів*)

XML – это текстовый формат, предназначенный для хранения структурированных данных, для обмена информацией между программами, а также для создания на его основе более специализированных языков разметки.

XML документ должен содержать корневой элемент. Этот элемент является «родительским» для всех других элементов. Все элементы в XML документе формируют иерархическое дерево. Это дерево начинается с корневого элемента и разветвляется на более низкие уровни элементов. И все элементы могут иметь дочерние элементы.

Well-formed XML отличается от valid тем, что он пропускается парсером, а valid XML пропускается и парсером и валидатором.

1. Рефлексія: робота з методами. Динамічні проксі. Створення класу динамічного проксі, створення об’єкту динамічного проксі. (*10 балів*)

Рефлексия – это возможность кода изучать и изменять самого себя.

Для работы с методами существуют такие методы:

* getMethods() – метод, который вернет все методы, включая пронаследованные.
* getDeclaredMethods() – метод, который вернет все методы, принадлежащие анализируемому классу, но без пронаследованных.
* getMethod(String name, Class<?>…params) возвращает объект типа Method из списка методов исследуемого объекта, включая пронаследованные. Параметр name это строка, определяющая имя нужного метода. Далее в параметры передается перечисление типов параметров нужного(искомого) метода.
* gettDeclaredMethod(String name, Class<?>…params) работает по принципу getMethod(String name, Class<?>…params), только возвращает объект типа Method из списка методов исследуемого объекта не учитывая пронаследованые методы.
* invoke(Object obj, Object…args) throws… - метод, вызывающий метод на объекте, который передается в параметр obj, также в параметры передается перечисление значений(параметров) необходимых, для вызова данного метода. Для вызова метода без параметров в параметры метода invoke(obj) передаётся только объект.

Пример работы с методами:

**public class Client{**

**//Methods**

**public String getSurname();**

**...**

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

**class<Client> cls = ClassforName(“Client”);**

**Client obj = cls.newInstance();**

**Method method = cls.getMethod(“getSurname”);**

**String name = m.invoke(obj);**

**}**

**}**

Прокси – это оболочки, которые передают вызов функции, добавляя свою(прописанную) функциональность. Динамический прокси даёт возможность одному классу с одним методом обслуживать несколько вызовов методов для разных классов с разным количеством методов. Он используется, когда реализации классов и/или методов не известны.

Далее приведен код примера:

В Main вводится параметр, создается объект класса EvaluateCube и прокси на этот объект. Далее происходит вызов функции через прокси.

**public class Main {**

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

**Scanner scan = new Scanner(System.*in*);**

**System.*out*.print("х: ");**

**double x = scan.nextDouble();**

**Evaluate f = new EvaluateCube();**

**Evaluate proxy = (Evaluate) Proxy.*newProxyInstance*(f.getClass().getClassLoader(), f.getClass().getInterfaces(), new InvokeHandler(f));**

**proxy.evaluate(x);**

**scan.close();**

**}**

**}**

Класс EvaluateCube в методе evaluate(double x) вычисляет куб значения, переданного в параметры имплементирует интерфейс Evaluate:

**public interface Evaluate {**

**double evaluate(double x);**

**}**

**public class EvaluateCube implements Evaluate {**

**@Override**

**public double evaluate(double x) {**

**return x\*x\*x;**

**}**

**}**

Класс InvokeHandler обрабатывает вызов метода, позволяя добавить дополнительные действия совместно с вызовом самого метода. В данном случае прописан вывод в консоль названия метода, его параметров и их типов, передаваемого значения и результата вызова метода.

**public class InvokeHandler implements InvocationHandler {**

**private Object obj = null;**

**public InvokeHandler(Object obj) {**

**super();**

**this.obj = obj;**

**}**

**@Override**

**public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {**

**Object result = method.invoke(obj, args);**

**StringBuilder str = new StringBuilder();**

**str.append(String.*format*("Метод: %s(", method.getName()));**

**int i = 0;**

**for (Parameter param : method.getParameters()) {**

**str.append(String.*format*("%s %s = %.1f, ", param.getType().getSimpleName(), param.getName(), args[i]));**

**i++;**

**}**

**str.setCharAt(str.length()-2, ')');**

**str.append(String.*format*("=> %.15f%n", result));**

**System.*out*.println(str.toString());**

**return result;**

**}**

**}**

1. Основи роботи з UDP сокетами. Основні етапи створення мережевого додатку за допомогою UDP сокетів. Приклад простого UDPсервера та клієнта: клієнт відправляє рядок на сервер, сервер повертає рядок великими літерами. (*10 балів*)

Протокол UDP не требует установки постоянного подключения. Если сокет должен получать сообщения, то надо привязать его к локальному адресу и одному из портов с помощью метода Bind(). После этого можно отправлять и получать сообщения. Для получения сообщений используется метод ReceiveFrom(). В качестве параметра в метод передается массив байтов, в который надо считать данные, и удаленная точка, с которой приходят эти данные. Метод возвращает количество считанных байтов. Для отправки данных используется метод SendTo(). В метод передается массив отправляемых данных, а также адрес, по которому эти данные надо отправить.

1. Основи *RMI*. Наведіть приклад простого розподіленого додатку, виконаного за технологією *RMI*, який би розв’язував задачу питання 3: клієнт надає серверу рядок, а сервер повертає клієнту рядок, записаний великими літерами. (*10 балів*)

Java RMI - это объектно-ориентированная версия удаленного вызова процедур (RPC). С помощью такого подхода можно без явного использования сокетов вызвать методы для объекта, который существует в другом адресном пространстве или на том же компьютере, или на другом компьютере, подключенном к исходному через сеть. Т.е. с помощью этого подхода, можно организовать взаимодействие объектов, расположенных на разных компьютерах. Эта технология используется для вызова удаленных методов в случаях, когда на всех концах взаимодействия установлена Java.

В структуре RMI приложения существует три стороны: серверная, клиентская часть и реестр объектов.

Задача серверная части программы состоит в создании удаленного объекта (remote object), который нужен для вызова метода. Этот объект обычный объект, за исключением того, что его класс реализует специальный интерфейс Java RMI. После того, как удаленный объект создан, он экспортируется и регистрируется в отдельном приложении, называемом реестром объектов.

Клиентская часть приложения сначала связывается с реестром объектов, для того, чтобы получить ссылку на удаленный объект с заданным именем. После этого, используя полученную ссылку, клиентская часть может вызывать методы на удаленном объекте так, как будто объект хранится в собственном адресном пространстве клиента. Технология RMI обрабатывает детали связи между клиентом и сервером (внутренне используя сокеты) и передает информацию в нужном направлении. Процедура связи полностью скрыта для клиентских и серверных приложений.

Реестр объектов похож на таблицу. Каждая запись таблицы отображает имя объекта на его прокси, который называется заглушка. Серверная часть приложения регистрирует заглушку с указанием имени в реестре объектов. После успешной регистрации заглушки в реестре объектов, объект становится доступным для использования другими объектами. Сразу после этого клиенты могут получить ссылку на удаленный объект из этого реестра (дескриптор объекта, фактически заглушку) и могут вызывать методы на этом объекте.

Работающая программа прикреплена в архиве Task\_4.