*/\*\*This class provides method to build Map from comap XML file.*

*\**

*\* @author Passichenko Victor*

*\* @author Yuri Zemlyanskiy*

*\**

*\*/*

package com.comapping.android.model;

import java.text.ParseException;

import org.w3c.dom.DOMException;

import org.w3c.dom.Document;

import org.w3c.dom.NamedNodeMap;

import org.w3c.dom.Node;

import org.w3c.dom.NodeList;

import com.comapping.android.Log;

public class MapBuilder {

private static final String METADATA\_TAG = "metadata";

private static final String MAP\_ID\_TAG = "id";

private static final String MAP\_NAME\_TAG = "name";

private static final String MAP\_OWNER\_TAG = "owner";

private static final String OWNER\_ID\_TAG = "id";

private static final String OWNER\_NAME\_TAG = "name";

private static final String OWNER\_EMAIL\_TAG = "email";

private static final String TOPIC\_TAG = "node";

private static final String TOPIC\_ID\_TAG = "id";

private static final String TOPIC\_LAST\_MODIFICATION\_DATE\_TAG = "LastModificationData";

private static final String TOPIC\_BGCOLOR\_TAG = "bgColor";

private static final String TOPIC\_FLAG\_TAG = "flag";

private static final String TOPIC\_PRIORITY\_TAG = "priority";

private static final String TOPIC\_SMILEY\_TAG = "smiley";

private static final String TOPIC\_TASK\_COMPLETION\_TAG = "taskCompletion";

private static final String TOPIC\_TEXT\_TAG = "text";

private static final String TOPIC\_ICON\_TAG = "icon";

private static final String ICON\_NAME\_TAG = "name";

private static final String TOPIC\_NOTE\_TAG = "note";

private static final String NOTE\_TEXT\_TAG = "text";

private static final String TOPIC\_TASK\_TAG = "task";

private static final String TASK\_DEADLINE\_TAG = "deadline";

private static final String TASK\_RESPONSIBLE\_TAG = "responsible";

*/\*\**

*\* Method that builds Map from comap XML file.*

*\**

*\* @param xmlDocument*

*\* text from comap XML file*

*\* @return built map*

*\* @throws StringToXMLConvertionException*

*\* when cannot convert given string to XML*

*\* @throws MapParsingException*

*\* when given XML document has wrong format*

*\*/*

public static Map buildMap(String xmlDocument) throws StringToXMLConvertionException, MapParsingException {

Log.i(Log.modelTag, "parsing xml document: \n" + xmlDocument);

long startTime = System.currentTimeMillis();

Document document = DocumentBuilder.buildDocument(xmlDocument);

Map map;

try {

*// parsing metadata*

Node metadata = document.getElementsByTagName(METADATA\_TAG).item(0);

int id = getIntValue(findChildNodeByName(metadata, MAP\_ID\_TAG));

map = new Map(id);

String name = getStringValue(findChildNodeByName(metadata, MAP\_NAME\_TAG));

map.setName(name);

Node ownerNode = findChildNodeByName(metadata, MAP\_OWNER\_TAG);

int ownerId = getIntValue(findChildNodeByName(ownerNode, OWNER\_ID\_TAG));

String ownerName = getStringValue(findChildNodeByName(ownerNode, OWNER\_NAME\_TAG));

String ownerEmail = getStringValue(findChildNodeByName(ownerNode, OWNER\_EMAIL\_TAG));

User owner = new User(ownerId, ownerName, ownerEmail);

map.setOwner(owner);

*// parsing topics*

NodeList nodes = document.getElementsByTagName(TOPIC\_TAG);

if (nodes.getLength() > 0) {

*// first node with TOPIC\_TAG must be root topic*

map.setRoot(buildTopic(nodes.item(0)));

} else {

*// there is no topic in this map*

}

} catch (NullPointerException e) {

e.printStackTrace();

Log.e(Log.modelTag, e.toString());

throw new MapParsingException();

} catch (NumberFormatException e) {

e.printStackTrace();

Log.e(Log.modelTag, e.toString());

throw new MapParsingException();

} catch (DOMException e) {

e.printStackTrace();

Log.e(Log.modelTag, e.toString());

throw new MapParsingException();

} catch (ParseException e) {

e.printStackTrace();

Log.e(Log.modelTag, e.toString());

throw new MapParsingException();

} catch (EnumParsingException e) {

e.printStackTrace();

Log.e(Log.modelTag, e.toString());

throw new MapParsingException();

} catch (StringToXMLConvertionException e) {

e.printStackTrace();

Log.e(Log.modelTag, e.toString());

throw new MapParsingException();

}

long parsingTime = System.currentTimeMillis() - startTime;

Log.w(Log.modelTag, "map was built successfully, parsing time: " + parsingTime);

return map;

}

*/\*\**

*\* Method that builds Topic and it's children from document's node*

*\**

*\* @param node*

*\* @return built topic*

*\* @throws MapParsingException*

*\* @throws ParseException*

*\* @throws EnumParsingException*

*\* @throws StringToXMLConvertionException*

*\*/*

private static Topic buildTopic(Node node) throws MapParsingException, ParseException, EnumParsingException,

StringToXMLConvertionException {

NamedNodeMap attributes = node.getAttributes();

*// parsing attributes*

Topic topic = new Topic();

boolean hasId = false;

for (int i = 0; i < attributes.getLength(); i++) {

Node curAttr = attributes.item(i);

if (curAttr.getNodeName().equals(TOPIC\_ID\_TAG)) {

topic.setId(Integer.parseInt(curAttr.getNodeValue()));

hasId = true;

} else if (curAttr.getNodeName().equals(TOPIC\_LAST\_MODIFICATION\_DATE\_TAG)) {

String strDate = curAttr.getNodeValue();

topic.setLastModificationDate(strDate);

} else if (curAttr.getNodeName().equals(TOPIC\_BGCOLOR\_TAG)) {

topic.setBgColor(Integer.parseInt(curAttr.getNodeValue()));

} else if (curAttr.getNodeName().equals(TOPIC\_FLAG\_TAG)) {

String strFlag = curAttr.getNodeValue();

topic.setFlag(Flag.parse(strFlag));

} else if (curAttr.getNodeName().equals(TOPIC\_PRIORITY\_TAG)) {

topic.setPriority(Integer.parseInt(curAttr.getNodeValue()));

} else if (curAttr.getNodeName().equals(TOPIC\_SMILEY\_TAG)) {

String strSmiley = curAttr.getNodeValue();

topic.setSmiley(Smiley.parse(strSmiley));

} else if (curAttr.getNodeName().equals(TOPIC\_TASK\_COMPLETION\_TAG)) {

String strTaskCompletion = curAttr.getNodeValue();

topic.setTaskCompletion(TaskCompletion.parse(strTaskCompletion));

}

}

if (!hasId) {

throw new MapParsingException();

}

NodeList childNodes = node.getChildNodes();

for (int i = 0; i < childNodes.getLength(); i++) {

Node childNode = childNodes.item(i);

if (childNode.getNodeName().equals(TOPIC\_TEXT\_TAG)) {

topic.setText(getStringValue(childNode));

} else if (childNode.getNodeName().equals(TOPIC\_TAG)) {

topic.addChild(buildTopic(childNode));

} else if (childNode.getNodeName().equals(TOPIC\_ICON\_TAG)) {

String iconName = childNode.getAttributes().getNamedItem(ICON\_NAME\_TAG).getNodeValue();

topic.addIcon(Icon.parse(iconName));

} else if (childNode.getNodeName().equals(TOPIC\_NOTE\_TAG)) {

String note = childNode.getAttributes().getNamedItem(NOTE\_TEXT\_TAG).getNodeValue();

topic.setNote(note);

} else if (childNode.getNodeName().equals(TOPIC\_TASK\_TAG)) {

String deadline = childNode.getAttributes().getNamedItem(TASK\_DEADLINE\_TAG).getNodeValue();

String responsible = childNode.getAttributes().getNamedItem(TASK\_RESPONSIBLE\_TAG).getNodeValue();

Task task = new Task(deadline, responsible);

topic.setTask(task);

}

}

return topic;

}

*/\*\**

*\* Method that returns first child of given node with specified name or null*

*\* if there is no such child*

*\**

*\* @param node*

*\* @param name*

*\* @return found child with specified name or null if there is no such child*

*\*/*

private static Node findChildNodeByName(Node node, String name) {

NodeList childNodes = node.getChildNodes();

for (int i = 0; i < childNodes.getLength(); i++) {

Node childNode = childNodes.item(i);

if (childNode.getNodeName().equals(name)) {

return childNode;

}

}

*//nothing found*

return null;

}

private static String getStringValue(Node node) {

return node.getFirstChild().getNodeValue();

}

private static int getIntValue(Node node) {

return Integer.parseInt(getStringValue(node));

}

}