package xmlcreate;

import java.io.\*;

import java.io.File;

import org.w3c.dom.Document;

import org.w3c.dom.\*;

import javax.xml.parsers.DocumentBuilderFactory;

import javax.xml.parsers.DocumentBuilder;

import org.xml.sax.SAXException;

import org.xml.sax.SAXParseException;

public class XMLCreate{

public static void main (String argv []){

try {

DocumentBuilderFactory docBuilderFactory = DocumentBuilderFactory.newInstance();

DocumentBuilder docBuilder = docBuilderFactory.newDocumentBuilder();

Document doc = docBuilder.parse (new File("Courses.xml"));

// normalize text representation

doc.getDocumentElement ().normalize ();

NodeList courses = doc.getElementsByTagName("course");

int totalcourses = courses.getLength();

System.out.println("Total no of courses : " + totalcourses);

for(int s=0; s<courses.getLength() ; s++){

Node firstcourse = courses.item(s);

if(firstcourse.getNodeType() == Node.ELEMENT\_NODE){

Element firstcourseelement = (Element)firstcourse;

//-------

NodeList firstcourselist = firstcourseelement.getElementsByTagName("code");

Element codeElement = (Element)firstcourselist.item(0);

NodeList textcodeList = codeElement.getChildNodes();

System.out.println("Code : " +

((Node)textcodeList.item(0)).getNodeValue().trim());

//-------

NodeList descriptionList = firstcourseelement.getElementsByTagName("description");

Element descriptionElement = (Element)descriptionList.item(0);

NodeList textdescriptionList = descriptionElement.getChildNodes();

System.out.println("Description : " +

((Node)textdescriptionList.item(0)).getNodeValue().trim());

//----

NodeList teacherList = firstcourseelement.getElementsByTagName("teacher");

Element teacherElement = (Element)teacherList.item(0);

NodeList textteacherList = teacherElement.getChildNodes();

System.out.println("Teacher : " +

((Node)textteacherList.item(0)).getNodeValue().trim());

NodeList sbList = firstcourseelement.getElementsByTagName("schoolBoard");

Element sbElement = (Element)sbList.item(0);

NodeList textsbList = sbElement.getChildNodes();

System.out.println("School Board : " +

((Node)textsbList.item(0)).getNodeValue().trim());

//------

}

}

}catch (SAXParseException err) {

System.out.println ("\*\* Parsing error" + ", line "

+ err.getLineNumber () + ", uri " + err.getSystemId ());

System.out.println(" " + err.getMessage ());

}catch (SAXException e) {

Exception x = e.getException ();

((x == null) ? e : x).printStackTrace ();

}catch (Throwable t) {

t.printStackTrace ();

}

//System.exit (0);

}//end of main

}

RECURSION

public int fibonacci (int f1, int f2, int sum, int counter, int input)

{

if (counter == input)

{

return sum;

}

else {

sum = f1 + f2;

f1 = f2;

f2 = sum;

counter++;

return fibonacci(f1,f2,sum,counter,input);

}

// debugging

// System.out.println (sum);

}

CLASSES

import java.util.ArrayList;

import java.text.DecimalFormat;

public class Player {

private String name;

private ArrayList<House> ownedProperty = new ArrayList<>();

private int balance;

DecimalFormat x = new DecimalFormat ("##,###,###.##");

public Player (String name, ArrayList<House> ownedProperty, int balance) {

this.name = name;

this.ownedProperty = ownedProperty;

this.balance = balance;

}

public String getName () {

return this.name;

}

public String getPropertyListString () {

String output = "";

String location = "";

char l1;

char l2;

for (int i = 0; i < ownedProperty.size(); i++) {

location = String.valueOf(ownedProperty.get (i).getCoordinates());

l1 = location.charAt (0);

l2 = location.charAt (1);

output += l1+ "," + l2 + ":\t$" + x.format(ownedProperty.get(i).getPrice() )+ "\n";

}

if (output.equals (""))

output = "NONE";

return output;

}

public ArrayList<House> getPropertyListArray () {

return ownedProperty;

}

public double getBalance () {

return this.balance;

}

public void setBalance (int num) {

this.balance = num;

}

public void addProperty (House h) {

this.ownedProperty.add(h);

}

}

**METHOD OVERLOADING**

**Argument lists could differ in –**  
1. Number of parameters.  
2. Data type of parameters.  
3. Sequence of Data type of parameters.

**Method overloading** is also known as **Static Polymorphism**.

**Points to Note:**  
1. [**Static Polymorphism**](http://beginnersbook.com/2013/04/runtime-compile-time-polymorphism/) is also known as compile time binding or early binding.  
2. [**Static binding**](http://beginnersbook.com/2013/04/java-static-dynamic-binding/) happens at compile time. Method overloading is an example of static binding where binding of method call to its definition happens at Compile time.

### Example 1: Overloading – Different Number of parameters in argument list

When methods name are same but number of arguments are different.

class DisplayOverloading

{

public void disp(char c)

{

System.out.println(c);

}

public void disp(char c, int num)

{

System.out.println(c + " "+num);

}

}

class Sample

{

public static void main(String args[])

{

DisplayOverloading obj = new DisplayOverloading();

obj.disp('a');

obj.disp('a',10);

}

}

**Output:**

a

a 10

In the above example – method disp() has been overloaded based on the number of arguments – We have two definition of method disp(), one with one argument and another with two arguments.

### Example 2: Overloading – Difference in data type of arguments

In this example, method disp() is overloaded based on the data type of arguments – Like example 1 here also, we have two definition of method disp(), one with char argument and another with int argument.

class DisplayOverloading2

{

public void disp(char c)

{

System.out.println(c);

}

public void disp(int c)

{

System.out.println(c );

}

}

class Sample2

{

public static void main(String args[])

{

DisplayOverloading2 obj = new DisplayOverloading2();

obj.disp('a');

obj.disp(5);

}

}

Output:

a

5

### Example3: Overloading – Sequence of data type of arguments

Here method disp() is overloaded based on sequence of data type of arguments – Both the methods have different sequence of data type in argument list. First method is having argument list as (char, int) and second is having (int, char). Since the sequence is different, the method can be overloaded without any issues.

class DisplayOverloading3

{

public void disp(char c, int num)

{

System.out.println("I’m the first definition of method disp");

}

public void disp(int num, char c)

{

System.out.println("I’m the second definition of method disp" );

}

}

class Sample3

{

public static void main(String args[])

{

DisplayOverloading3 obj = new DisplayOverloading3();

obj.disp('x', 51 );

obj.disp(52, 'y');

}

}

**Output:**

I’m the first definition of method disp

I’m the second definition of method disp

## Lets see few Valid/invalid cases of method overloading

## Case 1:

int mymethod(int a, int b, float c)

int mymethod(int var1, int var2, float var3)

Result: Compile time error. Argument lists are exactly same. Both methods are having same number, data types and same sequence of data types in arguments.

Case 2:

int mymethod(int a, int b)

int mymethod(float var1, float var2)

Result: Perfectly fine. Valid case for overloading. Here data types of arguments are different.

Case 3:

int mymethod(int a, int b)

int mymethod(int num)

Result: Perfectly fine. Valid case for overloading. Here number of arguments are different.

Case 4:

float mymethod(int a, float b)

float mymethod(float var1, int var2)

Result: Perfectly fine. Valid case for overloading. Sequence of the data types are different, first method is having (int, float) and second is having (float, int).

Case 5:

int mymethod(int a, int b)

float mymethod(int var1, int var2)

Result: Compile time error. Argument lists are exactly same. Even though return type of methods are different, it is not a valid case. Since return type of method doesn’t matter while overloading a method.