**Notes: First you prepare how to write a code for syntax and write a code**

**Java: (day one)**

Steps to install java

Steps to install eclipse

Steps to create workspace

Steps to create project

File -> Project

We call Project is program

How to create .java file/class

Project -> new class and give extension .java

Class Employee {

}

how to create packages and what is best way to give name

From solution explorer, select project, right click and select package

Ex: companyname.projectname.foldername (this is common naming standard)

what is main method will do?

Main method is starting point of program

What is variable?

It will store the value in memory

To create variable we specify

Variablename datatype;

what is data type and different data types`

It will represents what type of data

Int

Double

Float

creating property/data members : we create properties at class level

int salary

String firstname

creating method with void : we write methods in

void

**creating method with void and parameter**

public static void swapFunction(int a, int b) {

System.out.println("Before swapping(Inside), a = " + a + " b = " + b);

// Swap n1 with n2

int c = a;

a = b;

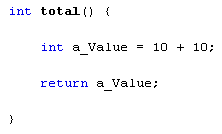
b = c;

System.out.println("After swapping(Inside), a = " + a + " b = " + b);

}

}

creating method with return data type



creating method with return data type and parameter

public static int minFunction(int n1, int n2) {

int min;

if (n1 > n2)

min = n2;

else

min = n1;

return min;

}

creating variable

creating static property All instances shared the value http://crunchify.com/java-static-methods-variables-static-block-and-class-with-example/

creating static method

http://crunchify.com/java-static-methods-variables-static-block-and-class-with-example/

creating object

calling method with no return

public class ExampleVoid {

public static void main(String[] args) {

methodRankPoints(255.7);

}

public static void methodRankPoints(double points) {

if (points >= 202.5) {

System.out.println("Rank:A1");

}else if (points >= 122.4) {

System.out.println("Rank:A2");

}else {

System.out.println("Rank:A3");

}

}

}

calling method with no return and parameter

calling method with return and no parameter

calling method with return and parameter

**public** **static** **int** Add(**int** a, **int** b)

{

**return**(a+b);

}

calling method with return and storing the return data

**package** methods\_example;

**public** **class** Passing\_Returnig {

/\*\*

\* **@param** args

\*/

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

*sayHello*("vijay");

*sayHello*("raj");

**int** sum = *Add*(50,54);

System.*out*.println(sum);

}

//just passing the arguments

**public** **static** **void** sayHello(String name)

{

System.*out*.println("Hello " +name);

}

//returning the arguments

**public** **static** **int** Add(**int** a, **int** b)

{

**return**(a+b);

}

**int** i= *Add*(1,2);

{

System.*out*.println(i);

}}

calling static method

using static property

**Java: Day2**

create classes under multiple packages

calling classes under different packages

write code to handle exceptions with try/catch/finally

**what is final keyword**

A final class cannot br subclass

A final method can't be overridden by subclasses

A final variable can only be initialised once

write code for interface and create class to implement that interface

**write code for creating abstract class**

abstract class Bank{

abstract int getRateOfInterest();

}

class SBI extends Bank{

int getRateOfInterest(){return 7;}

}

class PNB extends Bank{

int getRateOfInterest(){return 8;}

}

class TestBank{

public static void main(String args[]){

Bank b;

b=new SBI();

System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %");

b=new PNB();

System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %");

}}

**implement method overloading**

**public** **class** MethoOverLoadExample {

/\*\*

\* **@param** args

\*/

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

//method overloading is giving the same class name for diff para

System.*out*.println(*ADD*(1,34));

System.*out*.println(*ADD*(34.53,54.65));

System.*out*.println(*ADD*("Vijay","RAJ"));

}

**public** **static** **int** ADD(**int** a, **int** b)

{

**return**(a+b);

}

**public** **static** **double** ADD(**double** a, **double** b)

{

**return**(a+b);

}

**public** **static** String ADD(String a, String b)

{

**return**(a+b);

}

}

**implement method overriding**

implementing polymorphism

**implementing interface**

write a code to save data into excel file and read from excel file (POI and jexcel API)

how to update the data into XML file and read data from XML file

package com.journaldev.xml;

import java.io.File;

import java.io.IOException;

import javax.xml.parsers.DocumentBuilder;

import javax.xml.parsers.DocumentBuilderFactory;

import javax.xml.parsers.ParserConfigurationException;

import javax.xml.transform.OutputKeys;

import javax.xml.transform.Transformer;

import javax.xml.transform.TransformerException;

import javax.xml.transform.TransformerFactory;

import javax.xml.transform.dom.DOMSource;

import javax.xml.transform.stream.StreamResult;

import org.w3c.dom.Document;

import org.w3c.dom.Element;

import org.w3c.dom.Node;

import org.w3c.dom.NodeList;

import org.xml.sax.SAXException;

public class ModifyXMLDOM {

public static void main(String[] args) {

String filePath = "employee.xml";

File xmlFile = new File(filePath);

DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();

DocumentBuilder dBuilder;

try {

dBuilder = dbFactory.newDocumentBuilder();

Document doc = dBuilder.parse(xmlFile);

doc.getDocumentElement().normalize();

//update attribute value

updateAttributeValue(doc);

//update Element value

updateElementValue(doc);

//delete element

deleteElement(doc);

//add new element

addElement(doc);

//write the updated document to file or console

doc.getDocumentElement().normalize();

TransformerFactory transformerFactory = TransformerFactory.newInstance();

Transformer transformer = transformerFactory.newTransformer();

DOMSource source = new DOMSource(doc);

StreamResult result = new StreamResult(new File("employee\_updated.xml"));

transformer.setOutputProperty(OutputKeys.INDENT, "yes");

transformer.transform(source, result);

System.out.println("XML file updated successfully");

} catch (SAXException | ParserConfigurationException | IOException | TransformerException e1) {

e1.printStackTrace();

}

}

private static void addElement(Document doc) {

NodeList employees = doc.getElementsByTagName("Employee");

Element emp = null;

//loop for each employee

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

emp = (Element) employees.item(i);

Element salaryElement = doc.createElement("salary");

salaryElement.appendChild(doc.createTextNode("10000"));

emp.appendChild(salaryElement);

}

}

private static void deleteElement(Document doc) {

NodeList employees = doc.getElementsByTagName("Employee");

Element emp = null;

//loop for each employee

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

emp = (Element) employees.item(i);

Node genderNode = emp.getElementsByTagName("gender").item(0);

emp.removeChild(genderNode);

}

}

private static void updateElementValue(Document doc) {

NodeList employees = doc.getElementsByTagName("Employee");

Element emp = null;

//loop for each employee

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

emp = (Element) employees.item(i);

Node name = emp.getElementsByTagName("name").item(0).getFirstChild();

name.setNodeValue(name.getNodeValue().toUpperCase());

}

}

private static void updateAttributeValue(Document doc) {

NodeList employees = doc.getElementsByTagName("Employee");

Element emp = null;

//loop for each employee

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

emp = (Element) employees.item(i);

String gender = emp.getElementsByTagName("gender").item(0).getFirstChild().getNodeValue();

if(gender.equalsIgnoreCase("male")){

//prefix id attribute with M

emp.setAttribute("id", "M"+emp.getAttribute("id"));

}else{

//prefix id attribute with F

emp.setAttribute("id", "F"+emp.getAttribute("id"));

}

}

}

}

write code to add items to integer, string **array**

write code to retrieve items from integer, string **array**

write code to add items to ArrayList collection

**package** array;

**import** java.util.ArrayList;

**public** **class** AddingArrylist {

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

ArrayList<Integer> myList = **new** ArrayList<Integer>(4);

myList.add(1);

myList.add(2);

myList.add(3);

myList.add(4);

**System.*out*.println(myList);**

}

}

write code to retrieve items from arraylist (using for each loop\_

**package** array;

**import** java.util.ArrayList;

**public** **class** AddingArrylist {

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

ArrayList<Integer> myList = **new** ArrayList<Integer>(4);

myList.add(1);

myList.add(2);

myList.add(3);

myList.add(4);

**for**(Integer x: myList)

{

System.*out*.println(x);

}

}

}

write code to add items HashMap retrieve items HashMap

import java.util.\*;

public class HashMapDemo {

public static void main(String args[]) {

// Create a hash map

HashMap hm = new HashMap();

// Put elements to the map

hm.put("Zara", new Double(3434.34));

hm.put("Mahnaz", new Double(123.22));

hm.put("Ayan", new Double(1378.00));

hm.put("Daisy", new Double(99.22));

hm.put("Qadir", new Double(-19.08));

// Get a set of the entries

Set set = hm.entrySet();

// Get an iterator

Iterator i = set.iterator();

// Display elements

while(i.hasNext()) {

Map.Entry me = (Map.Entry)i.next();

System.out.print(me.getKey() + ": ");

System.out.println(me.getValue());

}

System.out.println();

// Deposit 1000 into Zara's account

double balance = ((Double)hm.get("Zara")).doubleValue();

hm.put("Zara", new Double(balance + 1000));

System.out.println("Zara's new balance: " + hm.get("Zara"));

}

}

Write code to add & retrieve items to hashset

import java.util.\*;

public class HashSetDemo {

public static void main(String args[]) {

// create a hash set

HashSet hs = new HashSet();

// add elements to the hash set

hs.add("B");

hs.add("A");

hs.add("D");

hs.add("E");

hs.add("C");

hs.add("F");

System.out.println(hs);

}

}

write code to connect to JDBC to get rows from employee table

**package** test;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.ResultSet;

**import** java.sql.SQLException;

**import** java.sql.Statement;

//import com.mysql.jdbc.Driver;

**public** **class** MySqlJdbcTest {

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

Connection conn = null;

Statement stmt = null;

ResultSet rs = null;

**try** {

// new com.mysql.jdbc.Driver();

Class.forName(**"com.mysql.jdbc.Driver"**).newInstance();

// conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/testdatabase?user=testuser&password=testpassword");

String connectionUrl = **"jdbc:mysql://localhost:3306/testdatabase"**;

String connectionUser = **"testuser"**;

String connectionPassword = **"testpassword"**;

conn = DriverManager.getConnection(connectionUrl, connectionUser, connectionPassword);

stmt = conn.createStatement();

rs = stmt.executeQuery(**"SELECT \* FROM employees"**);

**while** (rs.next()) {

String id = rs.getString(**"id"**);

String firstName = rs.getString(**"first\_name"**);

String lastName = rs.getString(**"last\_name"**);

System.out.println(**"ID: "** + id + **", First Name: "** + firstName

+ **", Last Name: "** + lastName);

}

} **catch** (Exception e) {

e.printStackTrace();

} **finally** {

**try** { **if** (rs != null) rs.close(); } **catch** (SQLException e) { e.printStackTrace(); }

**try** { **if** (stmt != null) stmt.close(); } **catch** (SQLException e) { e.printStackTrace(); }

**try** { **if** (conn != null) conn.close(); } **catch** (SQLException e) { e.printStackTrace(); }

}

}

}

Difference between string, string buffer, string builder with example

Mutability **Difference**: **String** is immutable, if you try to alter their values, another object gets created, whereas **StringBuffer** and **StringBuilder** are mutable so they can change their values. Thread-Safety **Difference**: The **difference between StringBuffer** and **StringBuilder** is that **StringBuffer** is thread-safe.