**\*JAVA CONCEPTS NOTES\***

**----------------------------------------------------------------------------------------------------Sid's**

**Java Day 1**

**What is a Class ?**

In the real world, you'll often find many individual objects all of the same kind. There may be thousands of other bicycles in existence, all of the same make and model. Each bicycle was built from the same set of blueprints and therefore contains the same components. In object-oriented terms, we say that your bicycle is an *instance* of the *class of objects* known as bicycles. A *class* is the blueprint from which individual objects are created.

**What is an Object ?**

An object is a software bundle of related state and behavior. Software objects are often used to model the real-world objects that you find in everyday life.

**What Is a Package?**

A package is a namespace that organizes a set of related classes and interfaces. Conceptually you can think of packages as being similar to different folders on your computer. You might keep HTML pages in one folder, images in another, and scripts or applications in yet another. Because software written in the Java programming language can be composed of hundreds or thousands of individual classes, it makes sense to keep things organized by placing related classes and interfaces into packages.

The Java platform provides an enormous class library (a set of packages) suitable for use in your own applications. This library is known as the "Application Programming Interface", or "API" for short. Its packages represent the tasks most commonly associated with general-purpose programming.

**Definition:** A *package* is a grouping of related types providing access protection and name space management.

**what is a method ?**

A method is a set of code which is referred to by name and can be called (invoked) at any point in a program simply by utilizing the method's name.  Think of a method as a subprogram that acts on data and often returns a value.

Each method has its own name.  When that name is encountered in a program, the execution of the program branches to the body of that method.  When the method is finished, execution returns to the area of the program code from which it was called, and the program continues on to the next line of code.

**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)

Naming Conventions

Package names are written in all lower case to avoid conflict with the names of classes or interfaces.

Companies use their reversed Internet domain name to begin their package names—for example, com.example.mypackage for a package named mypackage created by a programmer at example.com.

Name collisions that occur within a single company need to be handled by convention within that company, perhaps by including the region or the project name after the company name (for example, com.example.region.mypackage).

Packages in the Java language itself begin with java. or javax.

In some cases, the internet domain name may not be a valid package name. This can occur if the domain name contains a hyphen or other special character, if the package name begins with a digit or other character that is illegal to use as the beginning of a Java name, or if the package name contains a reserved Java keyword, such as "int". In this event, the suggested convention is to add an underscore. For example:

|  |  |
| --- | --- |
| **Legalizing Package Names** | |
| **Domain Name** | **Package Name Prefix** |
| hyphenated-name.example.org | org.example.hyphenated\_name |
| example.int | int\_.example |
| 123name.example.com | com.example.\_123name |

**What a main method will do?**

Main method is starting point of program

A Java application must contain a main() method whose signature looks like this

public static void main(String args[])

The method signature for the main() method contains three modifiers:

* public indicates that the main() method can be called by any object. public, private, protected, and the implicit, friendly.
* static indicates that the main() method is a class method.  void indicates that the main() method has no return value.

The main() method in the Java language is similar to the main() function in C and C++. When you execute a C or C++ program, the runtime system starts your program by calling its main()function first. The main() function then calls all the other functions required to run your program. Similarly, in the Java language, when you execute a class with the Java interpreter, the runtime system starts by calling the class's main() method. The main() method then calls all the other methods required to run your application.

**What is data type and different data types?**

It will represents what type of

Byte

Short

Int

Double

Float

Char

Boolean

* **creating property/data members**

int salary

* **creating method with void**

public static void main(String[] args)

* **public** **class** RankMethodology
* { **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");

**creating method with void and parameter**

public class Fish {

public void simpleMessage(String name) {

System.out.println(“ Hi “ + name);

}

* creating method with return data type

int total ( int aNumber) (

int a\_Value = aNumber + 10;

return a\_Value;

)

}

**creating method with return data type and parameter**

public class Animal {

public void addFriend(String name, Animal animal){

friends.put(name,animal);

}

public Animal callFriend(String name){

return friends.get(name);

}

}

**creating variable**

Local Variable;

Instance Variables;

Class/Static Variables;

**creating static property, creating static method**

import java.io.\*;

public class Employee

{

// salary variable is a private static variable

private static double salary;

// DEPARTMENT is a constant

public static final String DEPARTMENT = "Development ";

public static void main(String args[]) {

salary = 1000;

System.out.println(DEPARTMENT + "average salary:" + salary);

}

}

**creating object**

Point originOne = new Point(23, 94);

Rectangle rectOne = new Rectangle(originOne, 100, 200);

Rectangle rectTwo = new Rectangle(50, 100);

The first line creates an object of the Point class, and the second and third lines each create an object of the Rectangle class**.**

**calling static method**

public class Test4{

public static void method(){

System.out.println("Called");

}

public static void main(String[] args){

Test4 t4 = null;

t4.method();

}

}

**Java: Day2**

* **write code to handle exceptions with try/catch/finally**

public static void main(String args[])

{

try

{

int a=10;

int count=args.length;

args[42]="20";

// System.out.println("Number of values given:"+count);

// if(count==0)

// throw new ArithmeticException("please check denominator");

int result=a/count;

System.out.println("Result :"+result);

}

catch(ArithmeticException ar)

{

//System.out.println(ar.getMessage());

System.out.println("Denominator is zero");

* **what is final keyword**

Ans: In **Java** we use **final keyword** to variables to make its values not to be changed. But I see that you can change the value in the constructor / methods of the class. Again, if the variable is static then it is a compilation error. Here is the code: import **java**. util.

* **write code for interface and create class to implement that interface**

Interface looks like class but it is not a class. An interface can have methods and variables just like the class but the methods declared in interface are by default abstract (only method signature, no body). Also, the variables declared in an interface are public, static & final by default. We will discuss these points in detail, later in this post.

interface MyInterface

{

public void method1();

public void method2();

}

class XYZ implements MyInterface

{

public void method1()

{

System.out.println("implementation of method1");

}

public void method2()

{

System.out.println("implementation of method2");

}

public static void main(String arg[])

{

MyInterface obj = new XYZ();

obj. method1();

}

}

* **write code for creating abstract class**

A class that is declared with abstract keyword, is known as abstract class in java. It can have abstract and non-abstract methods (method with body).

**abstract** **class** Bike{

**abstract** **void** run();

}

**class** Honda4 **extends** Bike{

**void** run(){System.out.println("running safely..");}

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

 Bike obj = **new** Honda4();

 obj.run();

}

}

* **implement method overloading**

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);

}

}

* **implement method overriding**

Declaring a method in **subclass** which is already present in **parent class** is known as method overriding.

class Human{

public void eat()

{

System.out.println("Human is eating");

}

}

class Boy extends Human{

public void eat(){

System.out.println("Boy is eating");

}

public static void main( String args[]) {

Boy obj = new Boy();

obj.eat();

}

}

* **implementing polymorphism**
* class Overload
* {
* void demo (int a)
* {
* System.out.println ("a: " + a);
* }
* void demo (int a, int b)
* {
* System.out.println ("a and b: " + a + "," + b);
* }
* double demo(double a) {
* System.out.println("double a: " + a);
* return a\*a;
* }
* }
* class MethodOverloading
* {
* public static void main (String args [])
* {
* Overload Obj = new Overload();
* double result;
* Obj .demo(10);
* Obj .demo(10, 20);
* result = Obj .demo(5.5);
* System.out.println("O/P : " + result);
* }
* }

public class BaseClass

{

public void methodToOverride() //Base class method

{

System.out.println ("I'm the method of BaseClass");

}

}

public class DerivedClass extends BaseClass

{

public void methodToOverride() //Derived Class method

{

System.out.println ("I'm the method of DerivedClass");

}

}

public class TestMethod

{

public static void main (String args []) {

// BaseClass reference and object

BaseClass obj1 = new BaseClass();

// BaseClass reference but DerivedClass object

BaseClass obj2 = new DerivedClass();

// Calls the method from BaseClass class

obj1.methodToOverride();

//Calls the method from DerivedClass class

obj2.methodToOverride();

}

}

* **implementing interface**

interface MyInterface

{

public void method1();

public void method2();

class XYZ implements MyInterface

{

public void method1()

{

System.out.println("implementation of method1");

}

public void method2()

{

System.out.println("implementation of method2");

}

public static void main(String arg[])

{

MyInterface obj = new XYZ();

obj. method1();

}

}

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

package excelExportAndFileIO;

import java.io.File;

import java.io.FileInputStream;

import java.io.IOException;

import org.apache.poi.hssf.usermodel.HSSFWorkbook;

import org.apache.poi.ss.usermodel.Row;

import org.apache.poi.ss.usermodel.Sheet;

import org.apache.poi.ss.usermodel.Workbook;

import org.apache.poi.xssf.usermodel.XSSFWorkbook;

public class ReadGuru99ExcelFile {

public void readExcel(String filePath,String fileName,String sheetName) throws IOException{

//Create a object of File class to open xlsx file

File file = new File(filePath+"\\"+fileName);

//Create an object of FileInputStream class to read excel file

FileInputStream inputStream = new FileInputStream(file);

Workbook guru99Workbook = null;

//Find the file extension by spliting file name in substring and getting only extension name

String fileExtensionName = fileName.substring(fileName.indexOf("."));

//Check condition if the file is xlsx file

if(fileExtensionName.equals(".xlsx")){

//If it is xlsx file then create object of XSSFWorkbook class

guru99Workbook = new XSSFWorkbook(inputStream);

}

//Check condition if the file is xls file

else if(fileExtensionName.equals(".xls")){

//If it is xls file then create object of XSSFWorkbook class

guru99Workbook = new HSSFWorkbook(inputStream);

}

//Read sheet inside the workbook by its name

Sheet guru99Sheet = guru99Workbook.getSheet(sheetName);

//Find number of rows in excel file

int rowCount = guru99Sheet.getLastRowNum()-guru99Sheet.getFirstRowNum();

//Create a loop over all the rows of excel file to read it

for (int i = 0; i < rowCount+1; i++) {

Row row = guru99Sheet.getRow(i);

//Create a loop to print cell values in a row

for (int j = 0; j < row.getLastCellNum(); j++) {

//Print excel data in console

System.out.print(row.getCell(j).getStringCellValue()+"|| ");

}

System.out.println();

}

}

//Main function is calling readExcel function to read data from excel file

public static void main(String...strings) throws IOException{

//Create a object of ReadGuru99ExcelFile class

ReadGuru99ExcelFile objExcelFile = new ReadGuru99ExcelFile();

//Prepare the path of excel file

String filePath = System.getProperty("user.dir")+"\\src\\excelExportAndFileIO";

//Call read file method of the class to read data

objExcelFile.readExcel(filePath,"ExportExcel.xlsx","ExcelGuru99Demo");

}

}

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

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.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.NamedNodeMap;

import org.w3c.dom.Node;

import org.w3c.dom.NodeList;

import org.xml.sax.SAXException;

public class ModifyXMLFile {

public static void main(String argv[]) {

try {

String filepath = "c:\\file.xml";

DocumentBuilderFactory docFactory =DocumentBuilderFactory.newInstance();

DocumentBuilder docBuilder = docFactory.newDocumentBuilder();

Document doc = docBuilder.parse(filepath);

// Get the root element

Node company = doc.getFirstChild();

// Get the staff element , it may not working if tag has spaces, or

// whatever weird characters in front...it's better to use

// getElementsByTagName() to get it directly.

// Node staff = company.getFirstChild();

// Get the staff element by tag name directly

Node staff = doc.getElementsByTagName("staff").item(0);

// update staff attribute

NamedNodeMap attr = staff.getAttributes();

Node nodeAttr = attr.getNamedItem("id");

nodeAttr.setTextContent("2");

// append a new node to staff

Element age = doc.createElement("age");

age.appendChild(doc.createTextNode("28"));

staff.appendChild(age);

// loop the staff child node

NodeList list = staff.getChildNodes();

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

Node node = list.item(i);

// get the salary element, and update the value

if ("salary".equals(node.getNodeName())) {

node.setTextContent("2000000");

}

//remove firstname

if ("firstname".equals(node.getNodeName())) {

staff.removeChild(node);

}

}

// write the content into xml file

TransformerFactory transformerFactory = TransformerFactory.newInstance();

Transformer transformer = transformerFactory.newTransformer();

DOMSource source = new DOMSource(doc);

StreamResult result = new StreamResult(new File(filepath));

transformer.transform(source, result);

System.out.println("Done");

} catch (ParserConfigurationException pce) {

pce.printStackTrace();

} catch (TransformerException tfe) {

tfe.printStackTrace();

} catch (IOException ioe) {

ioe.printStackTrace();

} catch (SAXException sae) {

sae.printStackTrace();

}

}

}

**Java: Day3**

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

|  |
| --- |
| * static public void addStringElement(String[] arr, int pos, String name){ |

|  |  |
| --- | --- |
| 02 | String[] copy = new String[arr.length+1]; |

|  |  |
| --- | --- |
| 03 | for (int index = 0; index < arr.length+1; index++){ |

|  |  |
| --- | --- |
| 04 | if (index == pos){ |

|  |  |
| --- | --- |
| 05 | copy[index] = name; |

|  |  |
| --- | --- |
| 06 | }else{ |

|  |  |
| --- | --- |
| 07 | copy[index] = arr[index-1]; |

|  |  |
| --- | --- |
| 08 | } |

|  |  |
| --- | --- |
| 09 | } |

|  |  |
| --- | --- |
| 10 | arr = copy; |

|  |  |
| --- | --- |
| 11 | } |

* **write code to retrieve items from integer, string array**
* **write code to add items to ArrayList collection**

**public** **class** TestArray {

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

**int** a[]=**new** **int**[5];

a[0]=10;

a[1]=20;

a[2]=70;

a[3]=40;

a[4]=50;

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

System.***out***.println(a[i]);

}}

* **write code to retrieve items from arraylist**
* **write code to add items HashMap**

public class Program {

public static void main(String[] args) {

// Create new HashMap.

// ... Uses diamond inference on right side.

**HashMap**<String, Integer> hash = new HashMap<>();

// Put three keys with values.

hash.put("dog", 1);

hash.put("cat", 2);

hash.put("rabbit", 3);

// Look up some known values.

int a = hash.get("dog");

int b = hash.get("cat");

// Display results.

System.out.println(a);

System.out.println(b);

}

}

* **write code to retrieve items HashMap**
* This Java program does the followings
* 1. Inserts values into a HashMap
* 2. Retrieve the values of HashMap
* Java Program

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41 | import java.util.HashMap;  import java.util.Iterator;  import java.util.Map;  import java.util.Set;    public class MapDemo  {        public static void main(String args[])      {          Map map = new HashMap();            //Adding values to the HashMap          map.put("test key 1", "test value 1");          map.put("test key 2", "test value 2");          map.put("test key 3", "test value 3");            System.out.println("Retrieving values from HashMap");          retrieveValuesFromListMethod(map);          System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n");          }        /\*This method retrieves values from Map       \*/      public static void retrieveValuesFromListMethod(Map map)      {          Set keys = map.keySet();          Iterator itr = keys.iterator();            String key;          String value;          while(itr.hasNext())          {              key = (String)itr.next();              value = (String)map.get(key); |

* **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(); }

}

}

}

* **create Employee class**

public class Employee

{

    // create data fields

    private String firstName;

    private String lastName;

    private String phoneNumber;

    private String address;

    private int id;

    private String title;

    private double Salary;

    // Construct a default Employee object

    public Employee()

    {

    }

    // Construct a second constructor

    public Employee (String newFirstName, String newLastName)

    {

        firstName = newFirstName;

        lastName = newLastName;

    }

    //This method returns a String with the contents of

    //the variable firstName to whatever object calls it

    public String getFirstName()

    {

        return firstName;

    }

    //This method allows the contents of firstName

    //to be changed to store a different String value, should that be required

    public void setFirstName (String newFirstName)

    {

        firstName = newFirstName;

    }

}

* **Add employee class to list collection**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

class Employee

{

public string employeeID;

public string employeeLastName;

public string employeeFirstName;

public double employeePayrate;

public int employeeHours;

public bool employeeFT;

public Employee(string eID, string eLastName, string eFirstName, stringePhone, double ePayrate, int eHours, bool eFT)

{

this.employeeID = eID;

this.employeeLastName = eLastName;

this.employeeFirstName = eFirstName;

this.employeeID = ePhone;

this.employeePayrate = ePayrate;

this.employeeHours = eHours;

this.employeeFT = eFT;

}

static void Main()

{

List<Employee> employees = new List<Employee>();

employees.Add(new Employee("mtsubas\_e1", "Murtaza", "Mirza", "615-8982323", 10, 90, true));

employees.Add(new Employee("mtsubas\_e2", "Ray", "Angie", "615-898-223", 11, 80, true));

employees.Add(new Employee("mtsubas\_e3", "Burton", "James", "615-898-2323", 12, 70, false));

employees.Add(new Employee("mtsubas\_e4", "Bullard", "Dwight", "615-898-2323", 13, 60, true));

employees.Add(new Employee("mtsubas\_e5", "Apigian", "Charlie", "615-898-2323", 14, 50, true));

employees.Add(new Employee("mtsubas\_e6", "Lee", "Jung-Song", "615-898-2323", 15, 90, true));

employees.Add(new Employee("mtsubas\_e7", "Maier", "Lee", "615-898-2323", 16, 70, false));

employees.Add(new Employee("mtsubas\_e8", "Bamahnziari", "Tammy", "615-898-2323", 17, 60, true));

employees.Add(new Employee("mtsubas\_e9", "Greer", "Matt", "615-898-2323", 18, 50, true));

employees.Add(new Employee("mtsubas\_e10", "Coronnel", "Carlos", "615-898-2323", 19, 80, false));

}

}

* **create method that return list of employee collection**

public class Employee {

int empid;

String name;

int age;

public Employee(int empid,String name,int age)

{

this.empid=empid;

this.name=name;

this.age=age;

}

public int getEmpid() {

return empid;

}

public void setEmpid(int empid) {

this.empid = empid;

}

public String getname() {

return name;

}

public void setname(String name) {

this.name = name;

}

public int getAge() {

return age;

}

public void setAge(int age) {

this.age = age;

}

}

comparator class:

public class Employee\_comparator implements Comparator<Employee> {

@Override

public int compare(Employee object1, Employee object2) {

return object1.getname().compareTo(object2.getname());

}

}