**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

Ans: https://www.tutorialspoint.com/eclipse/eclipse\_workspaces.htm

Steps to create project

File -> Project

We call Project is program

Ans: <https://www.tutorialspoint.com/eclipse/eclipse_create_java_project.htm>

create .java file/class

Project -> new class and give extension .java

When you create a class it will create class name with keyword class and name of class

Class will contain method and properties

Class Employee

{ //begin of class

} //end of class

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

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

<https://www.tutorialspoint.com/eclipse/eclipse_create_java_package.htm>

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

ex:chase.gems.DB

what is main method will do?

Main method is starting point of program

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

Properties will store the value

To create variable we specify

Datatype Variablename;

int salary

String firstname

what is data type and different data types`

It will represents what type of data we are storing

Int

Double

Float

String

bool

What is variable?

It will store the value in memory

To create variable we specify

Datatype Variablename;

creating method with void : we write methods in class

Inside method We write code/steps for manual test case steps

Method syntax :

**Returntype methodname parenthesis**

Ex: Void MethodName()

{

}

Method contains two parts:

Ex: Void MethodName(int param1) //method declaration/signature

{ //method body

}

//creating method with void

Ex: Void MethodName2() //method declaration/signature

{ //method body

}

If we put “void” method don’t return anything from method

creating variable, we can create variables inside method

Variable scope is within method only

Ex: int MethodName()

{

Int var1;

}

creating method with return data type, we can return **int/string/double/float/date** etc

When we are returning from method we have to use **return** keyword inside method

After return key you can return hard coded value:

Return 2

Or

return propertyname

Or

return variablename

**Note:** value that we specify after **return** keyword should be of data type that is specified in method signature

//method that will return hard coded value

Ex: int MethodName()

{

Return 2 //hardcoded value

}

//method that will return property value

Ex: int MethodName()

{

Return empId //property value

}

//method that will return property value

Ex: int MethodName()

{

Int var1=2;

Return var1; // return variable

}

//creating method with return data type and parameter

int MethodName(int id) {

Int id;

return id;

}

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

Static int empid;

creating static method

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

//static method

Static void GetEmpid()

{

}

creating object

calling method with void

calling method with no return and parameter

calling method with return and no parameter

calling method with return and parameter

calling method with return and storing the return data

**package** returnt;

**import** java.lang.invoke.MethodType;

**import** java.lang.reflect.Method;

**public** **class** methodtypes

{

/\* method with void return type and no parameters \*/

**public** **void** printName()

{

System.***out***.println("My name Yochitha");

}

/\*method with void return type and parameters\*/

**public** **void** sum(**int** a, **int** b)

{

System.***out***.println(a+b);

}

/\*method with return type and parameters\*/

**public** **int** sumAll(**int** x, **int** y, **int** z)

{

**int** total= x+y+z;

**return** total;

}

**public** String printGender()

{

String G= "Female";

**return** G;

}

**final** **static** **int** ***x*** = 10;

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

{

methodtypes types = **new** methodtypes();

types.printName();

types.sum(10, 20);

**int** a=types.sumAll(1, 2, 3);

System.***out***.println(" Output for sumAll Method "+a);

String s=types.printGender();

System.***out***.println(" Output for printGender Method "+s);

}

}

calling static method

using static property: it will maintain

**Ans:** public static int empid;

create classes under multiple packages

calling classes under different packages

**package** simple;

**public** **class** Access

{

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

{

AccessModifiers s=**new** AccessModifiers();

s.set();

}

}

**package** simple;

**public** **class** AccessModifiers

{

**protected** **void** set()

{

System.***out***.println(" yochitha ");

}

}

**package** simple;

**public** **class** Modifiers

{

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

{

AccessModifiers d=**new** AccessModifiers();

d.set();

}

}

**package** simple1;

**import** simple.AccessModifiers;

**public** **class** Data

{

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

{

AccessModifiers a=**new** AccessModifiers();

a.set();

}

}

write code to handle exceptions with try/catch/finally

package excephand;

public class First

{

public static void main(String[] args)

{

int a=10;

int b=20;

int c;

System.out.println(" The value of a "+a);

System.out.println(" The value of b "+b);

try

{

c=a/0;

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

}

catch(ArithmeticException e) // catch(Exception (Type) e(object))" General Syntax"

{

System.out.println(e);

System.out.println(" I am going out of Catch block and going to handle the other block of code ");

}

int result;

result=a\*b;

System.out.println(" Result will be "+result);

}

what is final keyword, create final class, final method, final property

write code for interface and create class to implement that interface

write code for creating abstract class

implement method overloading

**package** javaconcepts1;

**public** **class** MethodOverLoading

{

**int** z;

**void** add(**int** a,**int** b)

{

z=a+b;

System.***out***.println(" Result "+z);

}

**void** add(**int** a,**int** b,**int** c)

{

z=a+b+c;

System.***out***.println(" Result "+z);

}

}

**package** javaconcepts1;

**public** **class** Test

{

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

{

MethodOverLoading ml=**new** MethodOverLoading();

ml.add(10, 20);

ml.add(10, 20, 30);

ml.add(20, 20);

ml.add(10, 10, 20);

}

}

implement method overriding

**package** javaconcepts1;

**public** **class** MethodOverRiding

{

**void** display()

{

System.***out***.println(" Yochitha ");

}

}

**package** javaconcepts1;

**public** **class** Simple

{

**void** display()

{

System.***out***.println(" Kolipaka ");

}

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

{

Simple s=**new** Simple();

s.display();

MethodOverRiding ms=**new** MethodOverRiding();

ms.display();

}

}

implementing polymorphism

implementing interface

How to do inheritance in java (using extend keyword)

**package** javaconcepts1;

**public** **class** Base

{

**public** **void** Display()

{

System.***out***.println(" Yochitha ");

}

}

**package** javaconcepts1;

**public** **class** Child **extends** Base

{

**public** **void** Print()

{

System.***out***.println(" Kolipaka ");

}

}

**package** javaconcepts1;

**public** **class** GrandChild **extends** Child

{

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

{

GrandChild gc=**new** GrandChild();

gc.Display();

gc.Print();

}

}

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

**package** simple;

**public** **class** SingleArray

{

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

{

**int**[] a={1,2,3,4};

String[] s={"yochitha", "soumya"};

System.***out***.println(" First Element "+a[0]);

System.***out***.println(" First Element "+a[1]);

System.***out***.println(" First Element "+a[2]);

System.***out***.println(" First Element "+a[3]);

System.***out***.println(" First Element "+s[0]);

System.***out***.println(" First Element "+s[1]);

}

}

**package** simple;

**public** **class** DoubleArray

{

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

{

String[][] a=**new** String[2][2];

a[0][0]=" yo";

a[0][1]=" yochi";

a[1][0]=" yochit";

a[1][1]=" yochitha";

System.***out***.println(" Data "+a[0][0]);

System.***out***.println(" Data "+a[0][1]);

System.***out***.println(" Data "+a[1][0]);

System.***out***.println(" Data "+a[1][1]);

}

}

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

write code to add items to ArrayList collection

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

**package** javaconcepts1;

**import** java.util.ArrayList;

**public** **class** ArrayListExample

{

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

{

ArrayList<Integer> al=**new** ArrayList<>();

al.add(10);

al.add(20);

al.add(30);

al.add(40);

System.***out***.println(" Elements till now "+al);

al.add(1, 99);

al.add(3, 33);

System.***out***.println(" Elements till now "+al);

al.remove(2);

System.***out***.println(" Elements till now "+al);

}

}

**package** javaconcepts1;

**import** java.util.LinkedList;

**public** **class** LinkedListExample

{

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

{

LinkedList<String> ll=**new** LinkedList<>();

ll.add(" Yochitha ");

ll.add(" kolipaka ");

ll.add(" soumya ");

System.***out***.println(" Data in the LinkedList are "+ll);

ll.addFirst(" saibersys ");

ll.addLast(" Selenium ");

System.***out***.println(" Data in the LinkedList are "+ll);

ll.removeLast();

System.***out***.println(" Data in the LinkedList are "+ll);

}

}

**package** javaconcepts1;

**import** java.util.Vector;

**public** **class** VectorExample

{

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

{

Vector<Integer> v=**new** Vector<>();

v.add(1);

v.add(2);

v.add(3);

v.add(3);

v.add(4);

System.***out***.println(" Elements ..."+v);

v.add(0, 0);

System.***out***.println(" Elements ..."+v);

v.remove(2);

System.***out***.println(" Elements ..."+v);

}

}

write code to add items HashMap

write code to retrieve items HashMap

**package** javaconcepts1;

**import** java.util.HashMap;

**public** **class** HashMapExample

{

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

{

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

hm.put(1, " Yochitha ");

hm.put(2, " Ellie ");

hm.put(3, " Rihanna ");

hm.put(4, " Selena ");

hm.put(5, " Yochitha ");

hm.get(5);

System.***out***.println(" Data "+hm);

hm.replace(3, "Ashok");

System.***out***.println(" Data "+hm);

}

}

Write code to add items to hashset

Write code to retrieve items to hasset

**package** javaconcepts1;

**import** java.util.HashSet;

**public** **class** HashSetExample

{

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

{

HashSet<String> hs=**new** HashSet<>();

hs.add(" Sai ");

hs.add(" hai ");

hs.add(" jai ");

hs.add(" bye ");

System.***out***.println(" Data "+hs);

hs.add(" yo ");

System.***out***.println(" Data "+hs);

hs.add(" yo ");

System.***out***.println(" Data "+hs);

hs.remove(" jai ");

System.***out***.println(" Data "+hs);

}

}

**package** javaconcepts1;

**import** java.util.LinkedHashSet;

**public** **class** LinkedHashSetExample

{

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

{

LinkedHashSet<String> lhs=**new** LinkedHashSet<>();

lhs.add(" yo ");

lhs.add(" so ");

lhs.add(" as ");

lhs.add(" sri ");

lhs.add(" ash ");

lhs.add(" as ");

System.***out***.println(" Data "+lhs);

lhs.remove(" yo ");

System.***out***.println(" Data "+lhs);

}

}

**package** javaconcepts1;

**import** java.util.TreeSet;

**public** **class** TreeSetExample

{

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

{

TreeSet<Integer> ts=**new** TreeSet<>();

TreeSet<String> ts1=**new** TreeSet<>();

ts1.add(" Yochitha ");

System.***out***.println(" Elements "+ts1);

ts.add(1111);

ts.add(0);

ts.add(2222);

ts.add(3333);

System.***out***.println(" Data "+ts);

ts.remove(2222);

System.***out***.println(" Data "+ts);

}

}

6

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

package com.theopentutorials.jdbc.to;

import java.util.Date;

public class Employee {

private int empId;

private String empName;

private Date dob;

private double salary;

private int deptId;

public int getEmpId() {

return empId;

}

public void setEmpId(int empId) {

this.empId = empId;

}

public String getEmpName() {

return empName;

}

public void setEmpName(String empName) {

this.empName = empName;

}

public Date getDob() {

return dob;

}

public void setDob(Date dob) {

this.dob = dob;

}

public double getSalary() {

return salary;

}

public void setSalary(double salary) {

this.salary = salary;

}

public void setDeptId(int deptId) {

this.deptId = deptId;

}

public int getDeptId() {

return deptId;

}

//toString()

}

create Employee class

Add employee class to list collection

create method that return list of employee collection

Difference between string, string buffer, string builder with example

**String**  
  
Stringis *immutable*  ( once created can not be changed )object.

String can not be used by two threads simultaneously.  
String  once assigned can not be changed.  
  
String  demo = " hello " ;  
// The above object is stored in constant string pool and its value can not be modified.  
  
  
demo="Bye" ;     //new "Bye" string is created in constant pool and referenced by the demo variable              
 // "hello" string still exists in string constant pool and its value is not overrided but we lost reference to the  "hello"string    
  
**StringBuffer**  
  
StringBufferis mutable means one can change the value of the object . The object created through StringBuffer is stored in the heap . StringBuffer  has the same methods as the StringBuilder , but **each method in StringBuffer is synchronized**that is **StringBuffer is thread safe** .   
  
Due to this it does not allow  two threads to simultaneously access the same method . Each method can be accessed by one thread at a time .  
  
But being thread safe has disadvantages too as the performance of the StringBuffer hits due to thread safe property .

Thus  StringBuilder is faster than the StringBuffer when calling the same methods of each class.  
  
StringBuffer value can be changed , it means it can be assigned to the new value . Nowadays its a most common interview question ,the differences between the above classes .  
String Buffer can be converted to the string by using   
toString() method.  
  
StringBuffer demo1 = new StringBuffer("Hello") ;  
// The above object stored in heap and its value can be changed .

demo1=new StringBuffer("Bye");  
// Above statement is right as it modifies the value which is allowed in the StringBuffer  
  
**StringBuilder**  
  
StringBuilder  is same as the StringBuffer , that is it stores the object in heap and it can also be modified . The main difference between the StringBuffer and StringBuilder is that**StringBuilder is also not thread safe.**  
StringBuilder is fast as it is not thread safe .    
  
  
StringBuilder demo2= new StringBuilder("Hello");  
// The above object too is stored in the heap and its value can be modified  
demo2=new StringBuilder("Bye");   
// Above statement is right as it modifies the value which is allowed in the StringBuilder

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

**public** String[][] FacebookValues() **throws** BiffException, IOException

{

File f=**new** File("D:\\facebook.xls");

Workbook wb=Workbook.*getWorkbook*(f);

Sheet s=wb.getSheet(1);

**int** row=s.getRows();

**int** col=s.getColumns();

String data[][]=**new** String[row][col];

**for**(**int** i=1;i<row;i++)

{

**for**(**int** j=0;j<col;j++)

{

Cell values=s.getCell(j,i);

data[i][j]=values.getContents();

}

}

**return** data;

}

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

protected void addNode(String tagName, String value, Node parent)

{

    Document dom = parent.getOwnerDocument();

**// Create a new Node with the given tag name**

    Node node = dom.createElement(tagName);

**// Add the node value as a child text node**

    Text nodeVal = dom.createTextNode(value);

    Node c = node.appendChild(nodeVal);

**// Add the new node structure to the parent node**

    parent.appendChild(node);

}