**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

**import** Practice.Method;

**public class** MethodTypes {

/\*

\* method with void return type and no parameters

\*/

**Public void** printName(){

System.***out***.println("My name is kavya");

}

/\*method with void return type and parameters

\*

\*/

**publicvoid** 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[]){

Method Typestypes = **new**MethodTypes();

Method method = **new**Method();

types.printName();

types.sum(6, 5);

System.***out***.println(types.sumAll(2, 3, 4));

System.***out***.println(types.printGender());

String result = types.printGender();

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

System.***out***.println(method.validateHeader());

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

}

}

creating variable

**public class** Student {

**public** String lastName = "";

**public** String firstName = "";

**public** String DOB = "";

**public** String stuStat(Students) {

**return**"Last Name: " + s.lastName + " First Name: " + s.firstName + " DOB:" + s.DOB;

}

/\* to call a method using an object

\* student s1 = new student();

\*s1.stuStat(student1); /\*

\*/

**public** Student(String lastName, String firstName, String DOB) {

**this**.lastName = lastName;

**this**.firstName = firstName;

**this**.DOB = DOB;

}

**publicstaticvoid** main(String args[]) {

Studentstudent1 = **new**Student("kav", "Guduru", "May20");

System.***out***.println(student1.stuStat(student1));

Studentstudent2 = **new**Student("chandra", "guduru","Aug24");

System.***out***.println(student3.stuStat(student3));

Studentstudent3 = **new**Student("srinu", "Guduru", "February2");

System.***out***.println(student4.stuStat(student4));

}

}

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/

**public class** StaticDemo {

**static** {

System.***out***.println("My instructor is kavya");

*setTestString*("This is static block's string");

*setTestValue*(2);

}

**static**{

System.***out***.println("This is static block 2 ");

}

**Private static int Testvalue**;

**Public int** getTestValue(){

**Return** *TestValue*;

}

**Public static void** set TestValue(**int** testValue){

**if**(testValue>0)

StaticDemo.TestValue = testValue;

System.***out***.println("setTestValue method:" + testValue);

}

**publicstatic** String *testString*;

/\*\*

\* **@return** the testString

\*/

**publicstatic** String getTestString() {

**return***testString*;

}

/\*\*

\* **@param**testString the testString to set

\*/

**Public static void** setTestString(String testString) {

StaticDemo.*testString* = testString;

System.***out***.println("setTestString method: " + testString);

}

// static util method

**Public static int** subValue(**int**i, **int**j) {

**int** sum = i+j;

**Return** sum;

}

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

**Int** res = StaticDemo.*subValue*(6, 7);

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

}

}

using static property

**Java: Day2**

create classes under multiple packages

calling classes under different packages

**import** Practice.Method;

**public class** Parent {

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

// **TODO** Auto-generated method stub

System.***out***.println("I started learning Java");

//Add 2 numbers

**Int** a=3;

**Int** b=4;

**Int** sum=a+b;

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

System.***out***.println("sum is " + sum);

System.***out***.println("Navigating to home page");

Method m= **new** Method();

System.***out***.println(m.validateHeader());

//MethodObject.methodName

//ClassObject.methodname

}

}

**package** Practice;

**public class** Method {

**public** String validateHeader()

{

System.***out***.println("Header liks validated");

**return**"Pass";

}

}

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

**public class** ExceptionExample {

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

// **TODO** Auto-generated method stub

**int**a,b;

// it contains the code which has the possibility of error

**try**{

System.***out***.println("This is try block");

a=100;

b=0;

**int**result=a/b;

}

//compile(checked) and runtime(unchecked) errors

**catch**(ArithmeticException e){

System.***out***.println("This is catch block");

e.printStackTrace();

}

//it enables to close the input streams

**finally**{

System.***out***.println("This is a final block");

}

}

}

what is final keyword

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

write code for interface and create class to implement that interface

**package** Practice;

**public class** DrawingDemo {

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

// **TODO** Auto-generated method stub

Circle circle = **new**Circle();

circle.draw();

Square square= **new** Square();

square.draw();

}

}

**package** Practice;

//google

**Public interface** Shape {

**Public void** draw();

}

**package** Practice;

**public class** Circle **implements** Shape {

@Override

**Public void** draw() {

System.***out***.println("Drawing circle");

}

}

**package** Practice;

**public class** Square **implements** Shape {

@Override

**publicvoid** draw() {

// **TODO** Auto-generated method stub

System.***out***.println("Drawing square");

}

}

write code for creating abstract class

**package** Practice;

//method with bodies - concrete methods, and methodprototyes- without body

**Public abstract class** Bike {

**abstract void** run();

**void** description(){

System.***out***.println("I am a two wheeler");

}

}

**package** Practice;

**public class** Honda **extends** Bike {

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

// **TODO** Auto-generated method stub

Honda honda= **new**Honda();

honda.run();

honda.description();

}

@Override

**void** run() {

// **TODO** Auto-generated method stub

System.***out***.println("I have a Honda bike");

}

}

**package** Practice;

**public class** Scooty **extends** Bike {

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

// **TODO** Auto-generated method stub

Scooty Scooty= **new** Scooty();

Scooty.run();

Scooty.description();

}

@Override

**void** run() {

// **TODO** Auto-generated method stub

System.***out***.println("I have a Scooty");

}

}

implement method overloading

**public class** MethodOverloading {

**public int** multiply(**int**a, **int**b){

**return**a\*b;

}

**Public long** multiply(**long**a, **long**b){

**return**a\*b;

}

**Public void** multiply(**int**a ,String b){

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

}

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

// **TODO** Auto-generated method stub

MethodOverloading methodOverloading = **new** MethodOverloading();

System.***out***.println(methodOverloading.multiply(5, 6));

methodOverloading.multiply(4,"prasu");

System.***out***.println(methodOverloading.multiply(1L, 2L));

}

}

implement method overriding

**publicclass**MethodOverriding {

**publicstaticvoid** main(String[] args) {

// **TODO** Auto-generated method stub

// similar to abstraction

}

}

**import**java.util.Scanner;

**publicclass**ArrayDemo {

**publicstaticvoid** main(String[] args) {

// **TODO** Auto-generated method stub

**int**[] array = **newint**[5];

Scanner scanner = **new**Scanner(System.***in***);

System.***out***.println("Enter the values of the array");

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

array[i] = scanner.nextInt();

}

System.***out***.println("Array elements are:");

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

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

}

}

}

write code to add items to ArrayList collection

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

importjava.util.ArrayList;

importjava.util.List;

public class ArrayListDemo {

//3 imp collection frameworks: list, set, map

//List:ArrayList,LinkedList,vector

//Set:HashSet,LinkedHashSet,TreeSet

//Map:HashTable,HashMap,LinkedHashMap,TreeMap

public static void main(String args[]){

List<Integer> list = new ArrayList<Integer>();

list.add(1);

list.add(1, 3);

list.add(5);

list.add(5);

list.add(7);

System.out.println("List Size:" +list.size());

for(Integer x:list){

System.out.println(x);

}

//list can contain duplicate elements

list.remove(0);

System.out.println("List Size:" +list.size());

for(Integer x:list){

System.out.println(x);

}

}

}

write code to add items HashMap

write code to retrieve items HashMap

importjava.util.HashMap;

importjava.util.Map;

public class HashMapDemo {

//Map consist of key value pairs

public static void main(String[] args) {

// TODO Auto-generated method stub

//It can only take single duplicate key and ,multiple duplicate values

Map<Integer,String> names = new HashMap<>();

names.put(1, "sowmya");

names.put(2, "sravani");

names.put(3, "venkat");

names.put(4, "rakesh");

System.out.println(names.keySet());

System.out.println(names.values());

names.remove(4);

System.out.println(names.keySet());

System.out.println(names.values());

}

}

Write code to add items to hashset

Write code to retrieve items to hasset

importjava.util.HashSet;

importjava.util.Set;

public class HashSetDemo {

public static void main(String[] args) {

// TODO Auto-generated method stub

//Set cannot take duplicate values

Set<String> names = new HashSet<>();

names.add("sowmya");

names.add("Rakesh");

names.add("sravani");

names.add("venkat");

names.add("sindhu");

names.add("chandu");

System.out.println(names.size());

for(String s :names){

System.out.println(s);

}

names.remove("pari");

System.out.println(names.size());

}

}

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

create Employee class

Add employee class to list collection

create method that return list of employee collection

importjava.util.ArrayList;

importjava.util.List;

public class Employee {

String fname;

String lname;

Employee(String fname,Stringlname){

this.fname = fname;

this.lname = lname;

}

static List<Employee>emplist = new ArrayList<>();

public List<Employee>getEmpList(){

returnemplist;

}

public static void main(String[] args) {

// TODO Auto-generated method stub

Employee e1 = new Employee("sowmya","koneru");

Employee e2 = new Employee("sravani","Donavalli");

emplist.add(e1);

emplist.add(e2);

List<Employee> list = e1.getEmpList();

for(Employee e :list){

System.out.println(e.fname+" "+e.lname);

}

}

}