Java Introduction

What is Java?

Java is a popular programming language, created in 1995.

It is owned by Oracle(Initially, Sun Micro System), and more than 3 billion devices run Java.

It is used for:

- Mobile applications (specially Android apps)

- Desktop applications

- Web applications

- Web servers and application servers

- Games

- Database connection

- And much, much more!

Why Use Java?

**-** Java works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc.)

- It is one of the most popular programming language in the world

-It is easy to learn and simple to use

- It is open-source and free

- It is secure, fast and powerful

- It has a huge community support (tens of millions of developers)

- Java is an object oriented language which gives a clear structure to programs and allows code to be reused, lowering development costs

- As Java is close to C++ and C#, it makes it easy for programmers to switch to Java or vice versa

What is a Class ?

- A Java class can be defined as a template or blueprint which describes state/behavior of it’s object.

- In Other Word a class is used to create Objects.

So in Java, every Project begins with a class name, and that class must match the filename.

Let's create our first Java file, called MyClass.java, which can be done in any text editor

The file should contain a "Hello World" message, which is written with the following code:

**File Name MyClass.java**

Program:

public class MyClass {

public static void main(String[] args) {

System.out.println("Hello World");

}

}

**Explanation:**

Every line of code that runs in Java must be inside a class. In our example, we named the class MyClass. A class should always start with an uppercase first letter.

Note: Java is case-sensitive: "MyClass" and "myclass" has different meaning.

The name of the java file must match the class name. When saving the file, save it using the class name and add ".java" to the end of the filename. To run the example above on your computer, make sure that Java is properly installed: Go to the Get Started Chapter for how to install Java. The output should be:

## The main Method

## The main() method is required and you will see it in every Java program:

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

## Any code inside the main() method will be executed. You don't have to understand the keywords before and after main. You will get to know them bit by bit while reading this tutorial.

## For now, just remember that every Java program has a class name which must match the filename, and that every program must contain the main() method.

## System.out.println()

## Inside the main() method, we can use the println() method to print a line of text to the screen:

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

## System.out.println("Hello World");

## }

What are Objects ?

## Objects are nothing but the instance of the class.

## - A single class can create any number of unique objects.

Creating objects

- In java an object is created when someone says “new”.

- At each “new” , a new object of a class is created. E.g. new MyClass

- Objects lives in Java heap.

What is java heap?

Java heap is nothing but the memory space taken by JVM from the OS.

All objects are created in this heap(space). Whenever JVM encounters “new” keyword, it creates an object in heap.

• When there is no space in heap to create objects, then JVM throws “Out of Memory” error.

Creating First Object

• public class Student

{

public static void main(String[] args)

{

new Student();

}

}

Garbage Collection

Garbage Collection is the mechanism provided by JVM, to clean out the Heap, so that new

objects can be created.

• It destroys the objects which are “not in use” or eligible for “garbage collection”.

• Any object is said to be eligible for garbage collection (GC) if there is no “Reference

Variable” attached to it.

• So what is a “Reference Variable”??????

Java Variables and its type

• Variables are named space of memory which stores the data.

• There are two types of variables :

* + Primitive variables
  + Reference variables

• Reference variables are those variables which stores only address of an “object.

Assigning Objects to Reference Variable

• Before creating a reference variable we have to specify “object of which class” it is going to refer.

• In other word we need to specify the “class type” of the reference variable. E.g.

public class Test

{

public static void main(String[] args)

{

Test t = new Test();

//where t is the reference variable which stores the

Address of Test object in heap

}

}

Primitive and Non primitive data types

|  |  |
| --- | --- |
| **Primitive Data type** | **Non-Primitive Data type** |
| Primitive data types are defined by the programming language. | Non-Primitive (or Reference) data type are defined by programmer. |
| These are - Integer type  - Floating type  - Character   - Boolean | In this the data type of the variable is the Class whose object it is going to refer. Test t = new Test(); //where Test is the data type of |

Declaration and initialization of primitive variables

**Integer**

-int i = 234242425;

- long l = 284798247287427427428947l; // //suffix ‘l’ is must otherwise compiler will treat it as integer.

**Float**

**-** float = 2342.34f; //suffix ‘f’ is must otherwise compiler will treat it as a double

**double** = 298472847242478927.2942949274;

**Character**

Char c = ‘j’; // only single character is allowed;

**Boolean**

Boolean b = true // only true or false is allowed

Java String and String Concatenation

- In java “String” is a class and not a data type and it can be instantiated like other classes

String s = new String();

- String Concatenation is basically a way to combine two or more strings into a single string. This is done by using ‘+’ operator.

String s = “We” + “are” + “learning” + “java” + “.”

- String values can be concatenate with any other datatype. boolean b = true;String s = “this is” + “ “ + b

|  |  |
| --- | --- |
| **Arithmetic operators** | **Unary operators** |
| • ‘+’ additive operators/string concatenate • ‘-’ subtraction operator • ‘\*’ multiplication operator • ‘/’ division operator • ‘%’ remainder operator | • ‘++’ increment operator • ‘--‘ decrement operator • ‘!’ logical compliment operator |

**Java Operator Continues**

|  |  |
| --- | --- |
| **Equality and Relational** | **Conditional Operators** |
| • ‘==‘ Equal to • ‘!=‘ Not equal to • ‘>’ Greater than • ‘<‘ Less than • ‘>=‘ Greater than or equal to • ‘<=‘ Lesser than or equal to | • ‘&&’ Conditional – And • ‘||’ Conditional – Or |

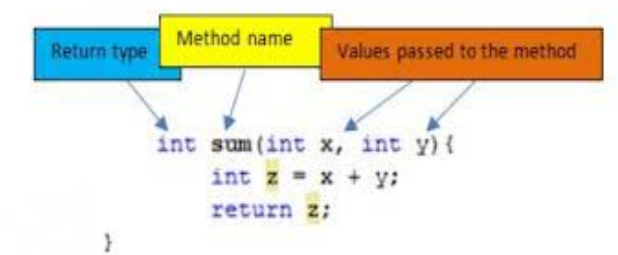
**Day 2**

**Git**

**What are Methods ?**

- Methods are block of statements which are used to do a specific task.

- Methods are generally used to divide a large code into manageable chunks of codes.



public int sum (int x, int y){

int z = x+y;

return z;

}

1. Access/NonAccess Modifier

2. returnType

3. MethodName ( Parameter List)

4. {

5. //method body or statement block

6. }

**Method Return Type**

• It May be possible that a method returns a value or doesn’t returns any value.

Return type of a method is nothing but the data type of the value returned by the method.

e.g.

- If method returns an integer value then it’s return type will be ‘int’, for float ‘float’, for character ‘char’, for boolean ‘boolean’ and so on.

- If a method doesn’t returns any value then it’s return type will be “void”.

• If a method returns a value then the last statement should be a “return statement”

• Any code after return statement is not reachable

**Parameter List**

• Method Parameters are the variables which are declared in the declaration of method.

e.g

**public int go(int x) {}**

• The method parameters can be nothing, one or more than one. For multiple parameters they are separated by comma.

e.g.

Public int go (int x, float y, String s, char c){ }

**Method Body**

• Body of a method starts from opening curly braces ‘{‘ and ends at closing curly braces ‘}’.

• All statements of methods goes within the pair of curlybraces.

• Execution of method stops on either ‘return’ statement or closing curly braces ‘}’.

e.g.

public int go(int x, int y)

{

Int sum = x + y;

Return sum;

}

But how to call a method???????

We see a example in the program.

**Instance variable Vs Local variable**

|  |  |
| --- | --- |
| **Instance Variables** | **Local Variable** |
| 1. Instance variables are those variables which are declared within a class. | 1. Local variable are those variables which are declared within a method. |
| 2. These are known as ‘instance variables’ because each instance of the class (object) have it’s own copy of instance variable. | 2. These are known as ‘Local variable’ because they formed within method and dies at the end of the method |

**Accessing Object’s methods and instance variables**

- The methods and instance variables of any object can be accessed by the help of dot operator (.) on reference variable

- referenceVariable.instanceVariable/method

e.g.

For class Animal having eat() method :

Animal a = new Animal();

a.eat();

**Method Arguments and method calling**

• Arguments are the values which are passed during calling of a method. These arguments are stored in the parameters of the method.

• Number of arguments must be equal to the number of parameters

• Calling a void method: a.eat();

• Calling a single argument method: a.eat(45);

• Calling a multiple argument method: a.eat(343, 43, ‘c’, “wsfsf”)

**Predefined Methods**

Predefined methods are those methods which are already defined in java and ready to use.

• We can call these methods and can use in our code directly. e.g. Math.random();

• Math.random() is a method defined in Math class which is used to generate a random number between o.o to .9 in double type

• Calling Math.random() : Int x = (int) (Math.random() \* 4);

//It will generate the number from 0 to 3

**Method Calling Methods**

- The method can call other methods in chain-like pattern.

e.g. Public **class** Test

{

**public** **void** firstMethod()

{

Test2 t = **new** Test2();

t.secondMethod(); // calling other method

}//end of method

}//end of class

**IF, IF-Else, IF-Else-IF statements**

IF statement:

This is a decision making statement, which will execute it’s codes **if** and only **if** the condition is **true**.

If(condition)

statement; // codes

• If-**else** statement:

In **this** the ‘**else**’ codes will be execute **if** and only **if** the condition is **false**. If(condition)

statement;//codes

**else**

statement; // codes

• If-**else**-**if** :

This is combination of **if**-**else** statements.

If(condition)

Statements;

Else **if** (condition)

Statements

Else **if** (condition)

Statements;

Else Statements;

**import** java.util.\*;

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

{

B t = **new** B();

Scanner scan = **new** Scanner(System.***in***); //ignore

System.***out***.println("enter a value");

**int** input = scan.nextInt(); //ignore

scan.close(); //ignore

**int** randomNum = (**int**)(Math.*random*()\* 10);

**if**(input == randomNum)

System.***out***.println("well done smarty");

**else**

t.number(randomNum);

System.***out***.println("randomNum is ----> "+randomNum);

}

**public** **void** number(**int** randomNum)

{

**if** (randomNum == 0)

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

**else** **if** (randomNum == 1)

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

**else** **if** (randomNum == 2)

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

**else** **if** (randomNum == 3)

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

**else** **if** (randomNum == 4)

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

**else**

System.***out***.println("Above 4");

}

}

**Day 3**

Loops

Loops are the way of repeating lines of codes until loop condition is met.!!!!

Loops in java :

¬ While loop

¬ Do-while loop

¬ For loop

¬ Enhanced for loop

**While loop**

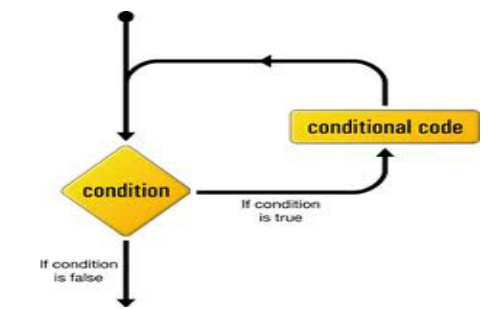
• While loop repeats a block of code until the condition is true

While (condition) {

// block of codes

}

//where condition is nothing but the Boolean expression

****

**Do-while loop**

Do-while loop is similar to the “while loop” but the only difference is that, in this the loop block is guaranteed to run at least one time!!!!

Syntax:

do

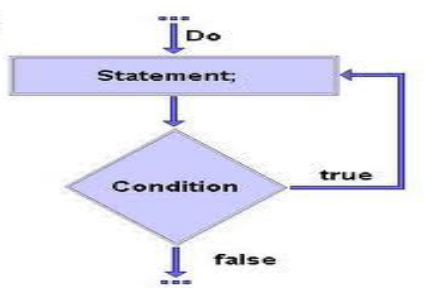
{

//all codes here

}

While (condition)

//since condition appears at the end, therefore the code block executes at least one



**For loop**

In for loop the initialization, condition checking and updating of loop element is done in a single line.

For(initialization; condition; update)

{

// codes

}

• The initialization step is executed first, and only once. This step allows you to declare and initialize any loop control variables.

• Next, the Boolean expression is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute.

• After the body of the for loop executes, the flow of control jumps back up to the update statement. This statement allows you to update any loop control variables.

**Nesting of loops**

• The placing of one loop inside the body of another loop is known as Nesting of loops.

• While working with nesting loops the outer loop will change only when inner loop is completely finished.

**for** (**int** outer =0; outer<5 ; outer++)

{

**for** (**int** inner= 0; inner <3; inner++)

{

System.***out***.println("outer is " + outer + "inner is" + inner);

}//inner loop ends

}//outer loop ends

}

**Break and Continue statements**

• Break and Continue statements are used to change the normal flow of compound statement.

• The break statement immediately jumps to the end of the compound statement.

• The continue statement immediately jumps to the next iteration of the compound statement.

**for** (**int** outer=0; outer< 12; outer++) {

**if**(outer ==3)

**continue**;

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

**if**(outer ==7)

**break**;

}

for (int i = 0; i < 10; i++) {

if (i == 4) {

break;

}

System.out.println(i);

}

for (int i = 0; i < 10; i++) {

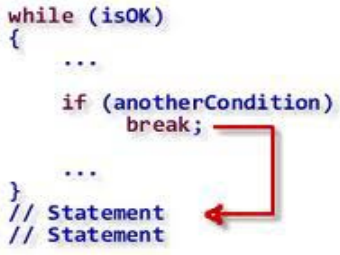
if (i == 4) {

continue;

}

System.out.println(i);

}



**Switch statement**

• Switch statement is the shorthand for multiple ‘if-else’ statement,

which allow us to choose a single path from a number of execution

path.

• Switch statement works with char, short, byte, int and String.

**switch**(x)

{

**case** 1:

System.out.println("case1");

**break**;

**case** 2:

System.out.println("case2");

**break**;

**case** 3:

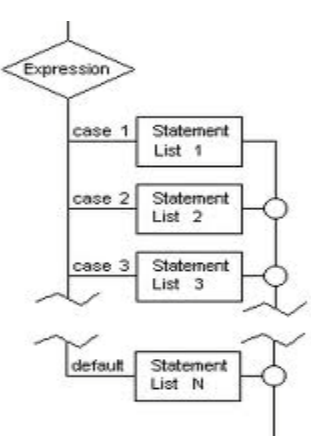
System.out.println("case 3");

**break**;

**default**:

System.out.println("default case");

}



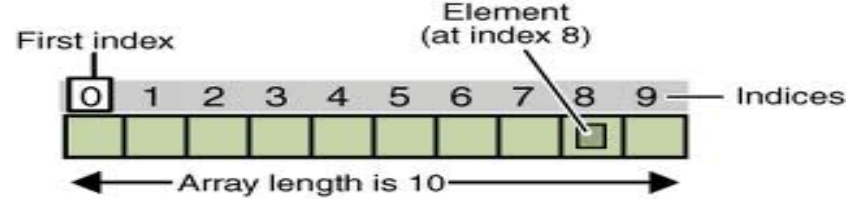
**Arrays**

Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.

• Each variable in an array is known as ‘array element.

• Each variable of array is referenced by a particular integer number which is known as ‘array index’.

• The total number variables in array decide the length of the array.



**Declaration and initialization of array**

• In java array is an object, therefore it is declared and initializes like an object.

• Declaration of array variable:

int[] array;

• Constructing the array:

new int[(length of the array)];

• Assigning array to array variable:

array = new int[(length of the array)];

• Initialization of array:

array[0] = 34;

• Declaration and initialization in single line:

Int[] array = { 34, 56, 7, 23, 34,};

**Initialization of array using loop**

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

**int** [] array = **new** **int**[34];

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

{

array[i]=i;

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

}

}

**Multidimensional arrays**

• A multidimensional array is an array containing one or more arrays.

To create a two-dimensional array, add each array within its own set of curly braces:

int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };

myNumbers is now an array with two arrays as its elements.

To access the elements of the myNumbers array, specify two indexes: one for the array, and one for the element inside that array. This example accesses the third element (2) in the second array (1) of myNumbers:

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

**int**[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };

**int** x = myNumbers[1][2];

System.***out***.println(x); // Outputs 7

}

We can also use a for loop inside another for loop to get the elements of a two-dimensional array (we still have to point to the two indexes):

public class MyClass {

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

**int**[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };

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

**for**(**int** j = 0; j < myNumbers[i].length; ++j) {

System.***out***.println(myNumbers[i][j]);

}

}

}

**Program #1 : pyramid of Stars**

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

{

**for** (**int** outer =1; outer<=5; outer++ )

{

**for** (**int** inner = 0; inner<outer; inner++)

{

System.***out***.print("\*");

}

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

}

}

**Enhanced for loop**

• “Enhanced for loop” is introduced in java 5, in order to simply the way to iterate a collection or

array.

• In this the loop continues till the last element of the collection or array.

**Syntax**

for (type variable : arrayname) {

...

}

The following example outputs all elements in the cars array, using a "for-each" loop:

Ex:

String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

for (String i : cars) {

System.out.println(i);

}

The example above can be read like this: for each String element (called i - as in index) in cars, print out the value of i.

If you compare the for loop and for-each loop, you will see that the for-each method is easier to write, it does not require a counter (using the length property), and it is more readable.

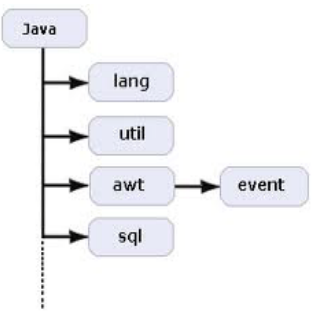
**Day 4**

**Packages in java**

• A package is the grouping of related types providing access protection and named space management.

• Packages are created by using the keyword “package” and it should be first line of the source file.

• In order to use classes of other packages we have to use “import” statements.



Using import to access packages

**package package1;**

**• public class demoTest {**

**• public void go()**

**• {**

**• System.out.println("in different package");**

**• }**

import package1.\*;

• public class Test {

• int x = 6;

• public static void main(String[] args)

• {

demoTest t = new demoTest();

• t.go();

• }

• }

package first;

public class ClassInFirst {

public void go()

{

System.out.println("in first package");

}

}

package third;

import first.second.\*;

import first.\*;

public class ClassInThird {

public static void main(String[] args)

{

ClassInFirst c = new ClassInFirst();

ClassInSecond d = new ClassInSecond();

c.go();

d.go();

}

}

package first.second;

public class ClassInSecond {

public void go()

{

System.out.println("in second package");;

}

}

**Access Modifiers in java**

Access modifiers specifies access level of a java component.

• Access modifiers can be divided into two categories:

1) Class level

2) Member level

**Class level access modifiers**• Public :

If a class is marked as public then it is accessible anywhere in java world.

Public class demo {}

• Default:

If a class have no modifier, then it will be marked as ‘default’ implicitly, then it is accessible in it’s package only

Class demo()

**Class Member level access modifier**• Public:

If a member is marked as public then it is accessible in whole java world.

• Default:

If a member have no modifier, then it will be marked as ‘default’ implicitly, and accessible in it’s package only.

• Protected:

If a member is marked as protected then it is accessible in it’s package. It is also accessible outside the package but through “inheritance” only.(????)

• Private:

If a member is marked as private then it is accessible in it’s class only

**Static**

A static method means that it can be accessed without creating an object of the class, unlike public:

**Object Orientation Programming - (OOPS)**

Any language is said to be object oriented if it supports following object properties:

1. Encapsulation

2. Inheritance

3. Polymorphism

4. Data abstraction (or interfaces)

• Encapsulation can be described as the mechanism in which we “encapsulate” our code in such a way that it can not be randomly accessed by other code outside the class.

The meaning of Encapsulation, is to make sure that "sensitive" data is hidden from users. To achieve this, you must:

• Declare class variables/attributes as private

• provide public get and set methods to access and update the value of a private variable

• If we want to include encapsulation in our code then we have to do following things:

1. Always make instance variable private.

2. Always make public accessor methods and force calling code to use these methods instead of directly calling the instance variables.

3. Use naming convention set() and get() for these

methods.

Problem scenario without encapsulation

• public class Duck {

• private int size;

• public void display()

• {

• if (size<0)

• System.out.println("incorrect size");

• else if(size>10)

• System.out.println("bigger duck!!!");

• else if (size < 10)

• System.out.println("smaller duck!!");

}

}

• public class Test {

• public static void main(String[] args)

• {

• Duck d = new Duck();

• d.size = 45;

• d.display();

• }

• }

**Getters and Setters**

Getters and Setters are nothing but the methods which are used to “set” and “get” the value of instance variables.

In previous chapter, private variables can only be accessed within the same class (an outside class has no access to it). However, it is possible to access them if we provide public get and set methods.

• Setters : Setters catch the “value” of instance variable in it’s parameter and “set” or assign this value to the instance variable. Setters have always parameters and no return.

• Getters: Getters returns the value of a instance variable to it’s “caller”. It only returns the already set value of a instance variable. Getters don’t have parameters and always return something

public class Duck {

private int size = 12;

public void setSize(int x)

{

if (x<=0);

else if(x>=25);

else

size = x;

}

public int getSize()

{

return size;

}

public void display()

{

if(size>10)

System.out.println("bigger duck!!!");

else if (size < 10)

System.out.println("smaller duck!!");

}

}

public class Test {

public static void main(String[] args)

{

Duck d = new Duck();

d.setSize(45);

System.out.println("the encapsulated size" +

d.getSize());

d.display();

}

}

**Benefits of Encapsulation** • Code becomes more maintainable and flexible.

• In future we can change our code, without breaking some other code, which depends on our code.

• The class have total control over what is going to be stored over it’s fields.

• The user of the class don’t know how class stores the data.

**Inheritance**

- Inheritance can be defined as the process in which one object acquires the properties of others

- By using inheritance the information becomes more manageable and in a hierarchical order.

- In other word we can say the Inheritance is the relationship between super class and subclass.

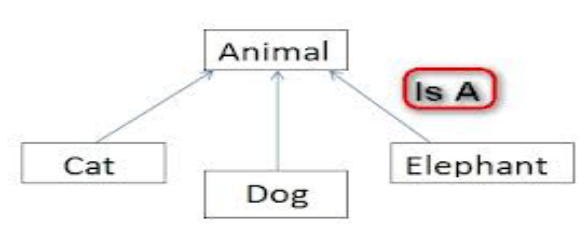
**Super class vs. Sub class**

• Super class:

**-** This is the class which contains common features of all subclasses

• Sub class:

- This is the class which inherits all the features of super class.



**Using Inheritance**

Inheritance relationship is created by using the keyword “extends”.

public class Animal {} // as superclass

• public class Dog extends Animal {} // Dog is subclass of Animal

• public class Cat extends Dog {} // Cat is subclass of Animal

public class Animal {

public void eat()

{

System.out.println("eating habit");

}

public void roam()

{

System.out.println("raoming habit");

}

}

public class Cat extends Animal{

public void sound()

{

System.out.println("meow meow!!!");

}

}

public class Dog extends Animal{

public void sound()

{

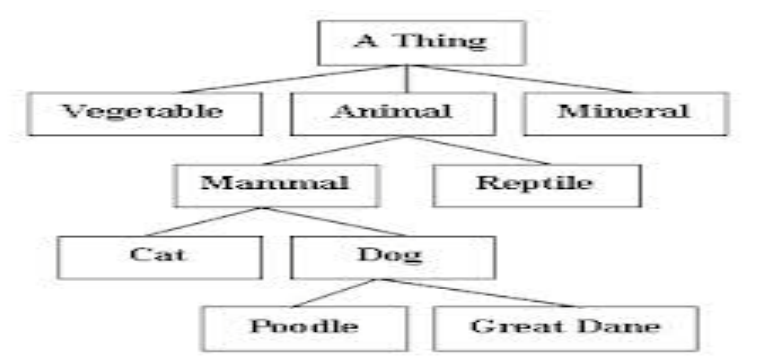
System.out.println("woof woof!!!");

}

}

**Inheritance Tree**

Inheritance Tree is basically a tree structure (upside down) that maps inheritance hierarchies of the classes.



**Method calling in Inheritance tree**

In inheritance tree the method calling is started from the lowest class in the tree.

e.g. if we call eat() method of “poddle” class then it starts searching from “Poddle” to “A Thing”.

**Method Overriding**

- Method Overriding is basically a ability to define a behavior which is specific to that sub class.

- Method Overriding RULES:

1. The arguments and the return type must be same as of superclass method.

2. The access level can’t be decreased.

Ex:

**The final Keyword**

If you don't want other classes to inherit from a class, use the final keyword:

If you try to access a final class, Java will generate an error:

final class Vehicle {

...

}

class Car extends Vehicle {

...

}