**Class definition:**

A class is a template, or plan, or blueprint from which we can create objects. A class describes properties(variables) and behavior(methods) of objects of its type.

Object: an object is the instance of a class or an object is an example of a class or an object is the reality of a class.

An object contains state (values of variables).

Method: a block of code which is used to write some functionality.

What is a java program?

A java program is a combination of java classes.

There can be any no. of java classes but utmost(max) one class can be public.

When there is a public class then the name of the program and the name of the public class should match otherwise **CE**.

d

A.java or B.java or C.java or RaviKanth.java. B.java

class A

{

}

public class B

{

}

class C

{

}

class A

{

}

class B

{

}

class C

{

}

Modifiers are of two types:

1. **Access modifiers**:- (public, protected, <default>, private)
2. **Non-Access modifiers** :- (final, static, abstract, strictfp, native, synchronized, transient and volatile)

Access limits: - whether a class can be accessed from everywhere (every package) or not.

Non-access modifiers specify the behavioral limits of a class, a method, a variable, a block.

The access modifiers specify the access limits of a class, a method, a variable and a constructor.

**Types of Classes:**

2-Types of classes

1. Outerclass
2. Inner class

The allowed access modifiers applicable for an outer class are public and default, and the non-access modifiers applicable for an outer class are final,abstract, and strictfp.

The allowed Access modifiers applicable for inner class are public, protected, <default>, private and the non-access modifiers applicable for an inner class are static,final, abstract, and strictfp.

**Public class:** it can be accessed from every package.

**Default class:** it can be accessed from only within the package.

**Final class:** final class cannot be inherited.

**Abstract class:** cannot be instantiate (means can’t create objects)

**Strictfp class:** if a class is declared as strictfp then all the floating-point calculations in the methods of that class have to follow IEEE754 standards so that we can get platform independent results.

In general, a class can contain variables and methods (instance block, static block, constructors).

**Variable Types:**

There are basically 2types of variables, primitive variables and reference variables.

These 2types are further divided into 3types based on the position of declaration and behavior: instance variables, static variables and local variables.

**Data types:**

In java, every variable should be declared with some data type before it is used, otherwise its CE.

The data type specifies the amount of memory that should be reserved for that variable to hold some value.

Ex:

int a = 10;

there are primarily 8-data types are there in java, they are called as primitive data types.

Primitive data types (8)

Numeric data types (6) non-numeric data types (2)

Integer data types (4) floating point data types (2)

Integer data types:

byte, short, int, long

floating point data types:

float, double

non-numeric data types:

Boolean, char

numeric data types are also called as signed data types because we can represent them both in +ve values and in -ve values.

**byte:**

size: 1 byte(8bits)

out of 8 bits, 1 bit is reserved for sign bit, remaining 7 bits represent actual value.

* **- - - - - - -**

**Msb(most significant bit)**

If msb if 1 then it’s a -ve number, if msb is 0 then it’s a +ve number.

range: -27  to 27 – 1 (-128 to 127)

0 1 1 1 1 1 1 1 -🡪 max number

1+2+4+8+16+32+64=127

1 0 0 0 0 0 0 0 🡪 min number -128

For -ve numbers, we take two’s complement form.

1st step: 1 1 1 1 1 1 1

2nd step : add 1

-----------------------------------------------------

1+ 1 =0 result , 1 carry

= 1 1 1 1 1 1 1 0 = -128

Ex:

byte b1=10;//valid

byte b2=127;//valid

btye b3=128;//invalid,CE

byte b4=-128;//valid

**short:**

size:2byte(16bits)

range: -32768 to 32767

**int:**

size: 4bytes

range: -2147483648 to 2147483647

**long:**

size: 8bytes

range: -263 to 263-1

the above 4 data types represent numbers without decimal points. If we want to represent numbers with decimal point then we have to use floating point data types.

**Float:**

Size: 4 bytes

Range: -3.4\*e38 to 3.4\*e38

**Double:**

Size: 8bytes

Range: -1.7\*e308 to 1.7\*e308

* By default, every integer number is of int data type.
* By default, every floating-point number is of double type.

byte b1=130;//CE

byte b2=60;//valid

short s=100;//valid

long l1=100;//valid

long l2=100L;//valid

float f1= 123.456;//invalid, CE

float f2=123.456f;//valid

double d1=123.456;//valid

double d2=123.456d;//valid

double d3=123.456f;//valid

**Boolean:**

size and range are not applicable, the allowed values are true, false (must be in lower case).

boolean b1=1;//CE

boolean b2=true;

**char:**

size:2bytes

range: 0 to 65535

char ch1=’a’;//valid

char ch2=97;

char ch3=’\n’;//valid

String s=”Hyderabad”;

**Types of variables:**

Basically there are 2types of variables.

Primitive variables and reference variables.

Primitive variables represent primitive values.

Ex:

Int a = 10;

a is primitive variable.

Reference variable.

Reference variables represent objects.

Ex:

String s=new String();

s is a reference variable.

Above 2types of variables are further divided into 3types:

Instance variables, static variables and local variables.

**Instance Variable**

**Definition:** if the value of a variable is varied from object to object then we call those variables as instance variables, for every object a separate copy of instance variables is created.

**Declaration:** instance variables are declared inside a class but outside a method, block, constructor.

**Static variables:**

**Definition**: if the value of a variable is not varied from object to object then we call those variables as static variables, for every object a single copy is created.

**Declaration:** static variable is declared inside a class, outside a method, block, constructor but by using static keyword.

**Local variables:**

**Definition:**The variables that are used for temporary requirement are called as local variables.

**Declaration:**Local variables are declared inside a method or block or constructor.

**Instance method:**

If we are using instance variables directly inside a method (it means we are talking about an object) then declare that method as instance method.

**Static method:**

If we are using instance variables indirectly (whether we use static variables or not) then declare that method as static method.

Access:

Inside an instance method we can access both instance variables/methods directly and static variables/methods directly.

Inside a static method, we can access static variables/methods directly but instance variables/methods indirectly.

Ex:

**package** pack1;

**publicclass** Test {

**int**a=10;

**staticint***b*=20;

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

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

System.***out***.println(*b*);//20

Test t=**new** Test();

System.***out***.println(t.a);//10

t.m1();

*m2*();

}

**void** m1()

{

System.***out***.println(a);//10

System.***out***.println(*b*);//20

}

**staticvoid** m2()

{

System.***out***.println("static method");

}

}

Abstract method:

A method that does not contain body is called as abstract method.

Abstract method ends with semicolon.

Ex:

Void m2(); //abstract method

Void m1()

{

//method body

//concrete method

}

If a class contains at least one abstract method then that class should be declared as abstract class.

We cannot create object for abstract class.

If a class contains all concrete methods still that class can be declared as abstract class, because nobody should be allowed to create an object for that class. (we cannot create objects for abstract class.)

**IS-A relationship:**

Also, called as inheritance.

We use extends keyword to implement inheritance.

The advantage is that we get code reusability.

Inheritance is the process where one class acquires the properties(variables) and behavior(methods) of other class.

The class which inherits the variables/methods of other class is called as sub-class or child class or derived class.

The class whose variables/methods are inherited is called as super class or parent class or base class.

**Properties of inheritance:**

1. All the variables/methods of parent class are by default available to child class.

Hence on child class reference variables we can call both parent class variables/methods and child class variables/methods.

1. None of the variables/methods of child class are available to the parent class hence on parent class reference variable we can call only parent class variables/methods but we cannot call child class variables/methods.
2. We can use parent class reference variable to refer/hold child class object.
3. We cannot use child class reference variable to hold parent class object.

**Ex:**

**package** pack1;

**class** Parent{

**int**a=10;

**void** m1()

{

System.***out***.println("parent method,not overridden");

}

**void** m11()

{

System.***out***.println("m11 overridden-parent");

}

}

**publicclass** Child **extends** Parent {

**int**b=20;

**void** m2()

{

System.***out***.println("child class specific method");

}

**void** m11()//overriding method

{

System.***out***.println("m11 overriding method-child");

}

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

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

Parent p1=**new** Parent();

//parent reference variable referring parent class object

p1.m1();//parent method,not overridden

System.***out***.println(p1.a);//10

//p1.m2(); CE

//System.out.println(p1.b); CE

Child c1=**new** Child();

c1.m1();//valid - parent method,not overridden

c1.m2();//child class specific method

System.***out***.println(c1.a);//valid- 10

System.***out***.println(c1.b);//20

Parent p2=**new** Child();//now overriding comes into picture.

//parent class reference variable(p2) can be used

//to refer child class object

p2.m1();//parent method,not overridden

p2.m11();//m11 overriding method-child

//p2.m2(); CE because m2() is not available in the parent

//Child c2=new Parent(); CE

//child class reference variable(c2) cannot be used

//to refer parent class object

}

}

Ex2:

Class A

{

}

Class B extends A

{

}

For class B, direct parent is A and Object class is indirect parent.

Ex3:

Class A

{

}

If our class does not extend any other class then our class becomes child class of Object class.

Ex4: multiple inheritance is not allowed in java.

Class A

A

B

C

Multilevel inheritance allowed in java

{

}

Class B extends A

{

}

Class C extends A, B //CE

{

}

Class C extends B

{

A a=new B();//valid

A a1=new C();//valid

A B

C

Multiple inheritance is not allowed in java.

}

HAS-A relationship

Also called as composition or aggregation.

No specific keyword to implement HAS-A relationship but in general we use new keyword.

class Car

{

Engine e=new Engine();

e.accelarate();

}

class Bus

Public class Engine//cannot be abstract

{

int accelerate()

{

}

}

{

Engine e=new Engine();

e.accelarate();

}

class Lorry

{

Engine e=new Engine();

e.accelarate();

}

Car HAS-A Engine, Bus Has-A Engine…..etc

**Modifiers:**

There are 2types of modifiers

1.access modifiers(AMs)

AMs specify access limits of a class, or method, or variable or constructor.

public, protected, <default>, private

2.non-access modifiers(NAMs)

NAMs specify behavioral limits of a class, or a method, or a variable or block.

final, static, abstract, strictfp, native, synchronized, transient, volatile.

**AMs chart:**

Inner Class

Method

Variable

Inner constructor

Public

Protected

<default>

Private

Note:the above class is an inner class.

For an outer class, the applicable AMs are public and <default>.

Ex1:outer class or normal class

Class A

{

}

Ex2:Inner class

Class A//outer class

{

Class B//inner class

{

}

}

**NAMs chart**:

method

Class(Outer, Inner)

variable

Package statement(folder structure):

Its an encapsulation mechanism to group related classes and interfaces into a single module.

Ex1:

The classes and interfaces that are used to interact with DB are grouped into a separate package called java.sql package.

Ex2:

The classes and interfaces that are used to perform IO operations are grouped into a separate package called java.io package.

When we write our own package statements, we have to use the domain name of the company in reverse.

Ex:

com.icicibank.loan.carloan.Account;

**package** com.icici.loan.carloan;

**publicclass** Account {

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

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

}

}

**Import statement:**

Import statement is used to import classes or interfaces of other packages in the current package.

**Class level modifiers:**

the AMs applicable for an outer class are public and default.

The NAMs applicable for an outer class are final, abstract, strictfp.

**1.public class**

Public classes can be accessed from everywhere(from any package).

Ex1:save the below class as A.java

package pack1;

public class A{

public void m1()

{

System.out.println(“m1 method from pack1 package”);  
}

}

Now lets try to access class A which is in pack1 package from class B which is in pack2 package.

Save the below class as B.java

package pack2;

import pack1.A;

class B {

Public static void main(String[] args) {

A a=new A();

a.m1();//valid

}

}

>javac A.java

>javac B.java

>java B (we have to execute/run class B , because main method is in class B).

Default class:

We can access default class only from within the package, we cannot access a default class from outside the package.

Ex2:save the below program as C.java

Below are the classes A and C present in pack1 package.

package pack1;

class A{

public void m1()

{

System.out.println(“m1 method from pack1 package”);  
}

}

class C {

Public static void main(String[] args) {

A a=new A();

a.m1();//valid

}

}

final class:

a final class cannot be extended i.e., we cannot create child class for a final class.

Ex:

final class A

{

}

class B extends A // CE,we cannot subclass a final class.

{

}

Note: every method of a final class is final bydefault.

Abstract class:

An abstract class cannot be instantiated, i.e., we cannot create an object for an abstract class.

Ex:

abstract class A

{

}

class B extends A

{

Public static void main(String[] args)

{

A a=new A();// CE

}

}

Note:

1. if a class contains at least one abstract method then the class should be declared as abstract class.
2. If a class contains no abstract methods still we can declare that class as abstract class because we should not be allowed to create an object for that class.

strictfp class :

if a class is declared as strictfp then all the floating point operations of every method will have to follow IEEE 754 standards, so that we can get platform independent result.

Method level modifiers:

1. Access modifiers:

Public method:

Public methods can be accessed from everywhere(from every package) provided the class should also be public.

Ex:

**package** pack2;

**import** pack1.A;

**publicclass** B {

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

A a=**new** A();//valid

a.m1();//CE, because m1() is not public.

}

}

**package** pack1;

**public class** A {

**void** m1()

{

sopln("m1 method from class A");

}

}

Protected method:

Protected method can be accessed from

i>same package every class(ex1)

ii>from other packages, only in child classes and using child class reference only(ex2).

Ex1:save the below program as B.java(both class A and class B are in the same package)

**package** pack1;

**class** A {

**protectedvoid** m1()

{

System.***out***.println("m1 method from class A");

}

}

**publicclass** B

{

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

{

A a=**new** A();

a.m1();

}

}

Ex2:

**Save this program as B.java**

**package** pack2;

**import** pack1.A;

**publicclass** B **extends** A {

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

B b=**new** B();

b.m1();//valid

A a=new A();//valid

//a.m1();//CE

}

}

**Save this program as A.java**

**package** pack1;

**publicclass** A {

**protectedvoid** m1()

{

sopln("m1 method from class A");

}

}

Note: if class B is not child class of A then we cannot access protected method of class A.

Default method:

if a method is declared as default then it can be accessed from any class within the same package.

Ex1:

Save the below program as B.java

**package** pack1;

**class** A {

**void** m1() {

System.***out***.println("m1 method from class A");

}

}

**class** B {

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

A a=**new** A();

a.m1();

}

}

Ex2:

Save the below program as A.java

**package** pack1;

**publicclass** A {

**void** m1()

{

System.***out***.println("m1 method from class A");

}

}

Save the below program as B.java

**package** pack2;

**import** pack1.A;

**publicclass** B {

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

A a=**new** A();//valid

a.m1();//CE

}

}

Private method:

Private methods can be accessed only within the class, we cannot access private methods outside the class.

Ex1:

**package** pack1;

**publicclass** A {

**privatevoid** m1()

{

System.***out***.println("m1 method from class A");

}

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

{

A a=**new** A();

a.m1();//valid

}

}

Ex2:save the below program as B.java

**package** pack1;

**class** A {

**privatevoid** m1() {

System.***out***.println("m1 method from class A");

}

}

class B {

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

A a=**new** A();//valid

a.m1();//CE

}

}

**Access modifiers for variables:**

About instance variables and static variables:

Same above rules are applicable for variables also.

B) Non-Access modifiers for methods:

Except transient and volatile remaining 6 NAMs are applicable for a method.

**Final method:**

A final method cannot be overridden.

If a method in the parent class is declared as final then we cannot override that method in the child class.

Ex:

**package** pack1;

**publicclass** A {

**final void** m1()

{

System.***out***.println("m1 method from class A");

}

}

**class** B **extends** A

{

**Void** m1()//CE

{

System.***out***.println("overriding m1 method");

}

}

}

**Static method:**

if a method uses instance variables directly it means it is talking about an object, then declare that method as instance method.

Directly means without using object/reference variable.

If a method uses instance variables indirectly(using a reference variable) then declare that method as static method.

Ex:

**package** pack1;

**publicclass** A {

**int**x=10;//instance variable

**void** m1() //instance method

{

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

}

**staticvoid** m2() //static method

{

A a=**new** A();//create an object

System.***out***.println(a.x);//indirectly

}

**staticvoid** m3() //static method

{

*m2*();//directly

}

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

{

A a=**new** A();

a.m1();

*m2*();//static method can be accessed directly from static area

//static are means: static method, static block.

*m3*();

}

}

**Static variables can be accessed**

i>directly from static area (static method or static block) and from instance area (instance method, instance block, constructor).

Directly means without creating any object.

ii>indirectly from static area and from instance area.

Indirectly means using a reference variable.

We will get a reference variable after creating an object.

**Abstract method:**

a method that has no method body is called as abstract method.

Abstract method is terminated by a semicolon.

If a class contains at least one abstract method then it should be declared as abstract class.

Ex:

abstract void m1();

an abstract method of a class should be given implementation by its child class.

Ex2:

class A //CE

{

abstract void m1();

}

Ex3:

abstract class A //valid

{

abstract void m1();

}

Ex4:

abstract class A

{

abstract void m1();

}

class B extends A

{

public void m1()

{

System.out.println(“hi”);

}

public static void main(String[] args)

{

A a=new A();//CE

B b=new B();//valid

b.m1();//valid

}

}

Ex5:

abstract class A

{

void m1()

{

System.out.println(“hi”);

}

}

Above code is valid, even though no method is abstract.

**strictfp method:**

if a method is declared as strictfp then all the floating-point calculations in that method will have to follow IEEE 754 standards so that we can get platform independent results.

**native method:**

the code for the native methods is not in java.

Native methods take the help of underlying OS code.

The purpose of native methods is to make use of already existing OS code.

Ex:

native void sleep(int millisecs); <- prototype

….

Thread.sleep(1000);//usage

**synchronized methods:**

if multiple threads try to operate on a single java object then there is a chance of data inconsistency, we can solve this problem by declaring the method as synchronized.

When a method is synchronized then at a time only one thread is allowed to operate on that method, if any other thread comes then it has to wait.

Ex: joint account

NAMs applicable for variables:

final, static, transient, volatile.

**static variable:**

if the value of a variable is not varied from object to object then we call it as static variable.

static variable are declared inside a class but outside a block, method, constructor.

We use static keyword to declare static variables.

**transient variable:**

while saving objects to a file, if we don’t want to save particular variables’ values then we have to declare those variables as transient.

If a variable is declared as transient then while saving that variable to a file, its original value will not be save, only default value will be saved.

We use transient keyword for sensitive data.

transient int password=976543;//0

transient String password=”hyd123”;//null

**default values:**

for variables of type byte, short, int, long the default value is 0.

For variables of type float, double the default is 0.0.

For Boolean type of variables the default value is false.

For char type of variables the default value is space.

For any object (object reference) the default value is null.

volatile variable: not required to learn, not important.

**final variable:**

types of variables:

depending upon the value represented by a variable, all the variables are of 2types.

1.primitive variables

2.reference variables

Depending upon the position of declaration and behavior, the above 2types of variables are further divided into 3types.

1.instance variables

2.static variables

3.local variables.

Instance variables:

If the value of a variable is varied from object to object then we call those variables as instance variables.

Instance variables are declared inside a class but outside block, constructor or method.

Ex: int rollno;

String sname;

int m1,m2,m3;

2.static variables

If the value of a variable is not varied from object to object then we call those variables as static variables.

Static variables are declared inside a class, outside a block, constructor or method but by using static keyword.

Ex:

static String colname;

3.local variables

The variables which are used for temporary requirement are called as local variables.

Local variables are declared inside a block, constructor or method.

final variable.

All the above 3 types of variables can be declared as final variable.

For a final variable, we cannot change the value.

Ex:

**package** pack1;

**class** A

{

**Int** a1=10;//instance variable

**Final int** a2=11;//final instance variable

**Static int** *b1*=20;//static variable

**Final static int *b2***=22;//final static variable

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

{

**Int** c=30;//local variable

System.***out***.println(c);//30

**Finalint** d=40;//final local variable

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

c=50;//valid

System.***out***.println(c);//50

//d=60;//CE

System.***out***.println(*b1*);//20

*b1*=200;//valid

System.***out***.println(*b1*);//200

//b2=222;//CE

A a=**new** A ();

System.***out***.println(a.a1);//10

a.a1=100;

System.***out***.println(a.a1);//100

System.***out***.println(a.a2);//11

// a.a2=110;//CE

}

}

Note:

1. At a time, we can use only one AM
2. At a time, we can use more than one NAM
3. At a time, we can use one AM and more than one NAM
4. The order of AMs/NAMs doesn’t matter.
5. An abstract class cannot contain final methods but a final class cannot contain abstract methods.
6. We can declare a class with abstract and strictfp at a time.
7. We cannot declare a class with both final and abstract.

For methods

1. A method can take 6NAM, out of 6 abstract is one, we cannot combine abstract modifier with remaining 5 NAMs.