# Java source file structure

A.java

public class B{

}

public class A{

}

File name should be public class name

Compile Time error:

class B is public, should be declared in a file named B.java

A.java There will be 4 classes after

public class A{ compile javac A.java

    public static void main(String[] args) { A.calss B.class C.class D.class

        System.out.println("a");

    } when executing A.class

} java A

class B{ the corresponding class’s

    public static void main(String[] args) { main method will be execute

        System.out.println("a"); “a”

    }

} when run class D

class C{ java D

    public static void main(String[] args) { Error:

        System.out.println("a"); Main method not found in class D

    }

} when run class H

class D{ java H

Error:

} ClassNotFoundException

But : Recommended

Only one class per file , with same as class name

Adv:

Readability and maintain ability

## Compiler is innocent

public class A{

    public static void main(String[] args) {

        ArrayList a = new ArrayList();

    }

}

Compile Time error:

cannot find symbol

ArrayList a = new ArrayList(); //based on structure compiler



symbol: class ArrayList guesses it is a class



location: class A

public class A{

    public static void main(String[] args) {

        System.out.println(ArrayList);

    }

}

Compile Time error:

cannot find symbol

ArrayList a = new ArrayList(); //based on structure compiler



symbol: variable ArrayList guesses it is a variable



location: class A

### Fully qualified name



public static void main(String[] args) {

        java.util.ArrayList a = new java.util.ArrayList(); }

So every time we need to tell fully qualified name to compiler

Its increase the lines, reduces readability.

### Import

import java.util.ArrayList;

public class A{

    public static void main(String[] args) {

        ArrayList a = new ArrayList();

    }

So we need to use import statement

It’s just typing short cut

Now ArrayList === java.util.ArrayList

So Import is only compile time improve compile time

But run time is same

#### Import category

Import

Explicit Implicit

(clearly and in detail)

Import java.util.ArrayList import java.util.\*

Recommended Explicit because improves readably

import java.util.ArrayList;



import java.util;



import java.util.ArrayList.\*;



import java.util.\*;



import java.util.\*;

import java.sql.\*;

public class A{

    public static void main(String[] args) {

        Date d = new Date();

Compile Time error:

both class java.sql.Date in java.sql

and class java.util.Date in java.util match

note : List in java.util.List and java.aws.List

import java.util.Date;

import java.sql.\*;

public class A{

    public static void main(String[] args) {

        Date d = new Date();

System.out.println(d.getClass().getName());

    }

Now explicit import will be considered

While resolving order of giving priority

1, explicit class import

2, classes in current working directory (default package)

3, implicit class import

I need to use Patterns class

java java.util.\*



|- util java.\*



|- regex java.util.regex.\*



|- Patterns no need to import



When importing all class/ interfaces in that package will be available but not sub package’s classes/interfaces

This 2 package classes not required to import

java.lang.\*

current working directory (default package)

#### Dynamic include/ load on fly/ load on demand

imprt java.lang.\*;

not .class file will be loaded at the beginning

when we use a particular class then only that class will be loaded

### Static import

Java 1.5 introduces

For-each

Var-arg method

AutoBoxing-AutoUnBoxing

Generic

Co-variant return types

Queue

Annotation

Enum

Static import (flop..)

If not specific requirement it’s not recommended to use

public class A{

    public static void main(String[] args) {

        System.out.println(Math.sqrt(10));

        System.out.println(Math.sqrt(43));

        System.out.println(Math.random());

Rather than use Math.sqrt() Every time,

We can use static import and sqrt()

//import static java.lang.Math.sqrt;

import static java.lang.Math.\*;

public class A{

    public static void main(String[] args) {

        System.out.println(sqrt(10));

        System.out.println(sqrt(23));

        System.out.println(random());

class Test{

static String s = “hi”;

}

Test.s.length()

is String type, so we can happily call String class’s method

s is static variable we can access it with class name

class System{

static PrintStream out;

}

System.out.println(“ji”)

System is a out is a static println() is

Class present in variable present in method in

Java.lang pack system class of the PrintStream

type PrintStream class

//import static java.lang.Math.sqrt;

import static java.lang.Math.\*;

import static java.lang.System.out;

public class A{

    public static void main(String[] args) {

        out.println(sqrt(10));

out is static variable present in System class, so we can access it by using System

but we can also use static import

while resolving static members compiler always consider this order

current class static members

explicit static import

implicit static import

//import static java.lang.Integer.MAX\_VALUE; priority 2

import static java.lang.Byte.\*; priority 3

public class A{

    //static int MAX\_VALUE =12; priority 1

    public static void main(String[] args) {

        System.out.println(MAX\_VALUE);

    }

}

explicit import

import packageName.className

explicit static import

import static packageName.className.staticMember

(Member == method/variable)

Ex:

Import static java.lang.Math.sqrt; //squrt() static method

Import static java.lang.Integer.MAX\_VALUE;

implicit import

import packageName.\*

implicit static import

import static packageName.className.\*

import java.lang.Math.\*; within class import what?



import static java.lang.Math.\*; all static member within Math



import java.lang.Math.sqrt; no static keyword for static mem



import static java.lang.Math.sqrt(); for static mem no need ()



import java.lang.Math.sqrt.\*; import what??



import static java.lang.Math.sqrt; explicit import sqrt()



import java.lang;



import static java.lang;



import java.lang.\*; implicit import all classes with that pack



import static java.lang.\*; there can’t be vari/method



#### Ambiguity in Import

Pack2

Pack1

Rare (Date, List) Common

Class/interface with same name

method/variable with same name

* 2 packages contain classes with same name is very rare

Hence ambiguity problem is very rare

* 2 classes contain variables or methods with same name is

Very common Hence ambiguity problem is very common in static import

Note:

Usage of static import reduces readability make confusions

“”

“” 10000 line code

“” now I don’t know this sqrt() method from

Sop(sqrt(4)); which class

“” so use Math.sqrt(4) will be easy

Import

To import classes/interfaces of a particular package

So no need fully qualified name for every time

Static import

To import static members of a particular class/Interface

So no need Class.staticMem (class name. )

## Package

Group of related thinks

Encapsulation (grouping)

It’s an encapsulation mechanism to group related interfaces and classes into a single unit

Ex:

All classes and interfaces which are related for database operations are grouped into a single package which is java.sql package

### Advantage

1. To resolve naming conflicts

That is unique identification of our component

1. It improves modularity of application

Order module

Order delivery module

Order processing module

com.xyz.order.process

com.xyz.order

1. So Maintainability will improve
2. Security

Package1



### Naming convention for packages

Using internet domain name in reverse

Com.icicbank.lorn.housing.Account

Client internet domain name

reverse

Class name

Sub

Module name

Module name

package com.corejava.pack.inro;

public class A{

    public static void main(String[] args) {

        System.out.println(Integer.MAX\_VALUE);

    }

}

Now this will compile fine. ( javac A.java )

But .class file is created in current working directory (cwd)

We need to create .class file in

cwd

javac –d . A.java

|- com

|- corejava

|- pack

current working

directory (cwd)

Destination to place .class file

Is |- intro

|\*A.class

.class file will be placed in corresponding package structure

If following structure is not created then the folders will be automatically created

package com.corejava.pack.inro;

Javac –d F: A.java

Can be any valid directory name (ex D:\Work\Study)

F

|- com

|- corejava

|- pack

|- intro

|\*A.class

javac -d D:\work\example A.java

this will create /work/example folders at D:

after com/corjava/pack/itro then place A.class there

javac -d Z:\work\example A.java

Z directory is not found compile time error

#### Execution .class

Current working directory

|- com

|- corejava

|- pack

|- intro

|\*A.class

java com.corejava.pack.intro.A

#### nots

package com.corejava.pack.inroduction1;

package com.corejava.pack.inroduction;

public class A{

At most only one package statement can be in .java file

Otherwise

ComileTimeError : class, interface, or enum expected

import java.util.Date;

package com.corejava.pack.inroduction;

public class A{

First statement should be package

Otherwise

ComileTimeError : class, interface, or enum expected

Because after import compiler expect class, interface, or enum

At most one (0/1) order is

Package statement;

Import statement;

Class/ interface/ enum declaration

Any numbers (0=<) important

Any numbers (0=<)

An empty source file is valid. hence these are valid

Import statement;

Class A{

}

Package statement;

Import statement;

Package statement;

# Class level modifiers

When we are writing our own classes we have to provide some information about our class to the JVM like..

Whether this class can be accessible from anywhere or not

Child class creation possible or not

Object creation is possible or not

….

We can specify this information by using appropriate modifiers

public final native

private abstract strictfp

<default> static transient

protected synchronized volatile

Applicable modifiers for **Top level classes**

Applicable modifiers for **inner classes**

## Access Specifiers Vs. Access Modifiers

In old languages like C++ public, private, default, protected is considered as Access specifiers. But in java there is no terminology as Access specifies. All are Access modifiers

## public

If a class as public then we can access that class from anywhere

package com.modifiers.\_public.b;

//import com.modifiers.\_public.a.\*;

class B{

p s void main(String[] args) {

A a = new A();

a.m1();

package com.modifiers.\_public.a;

public class A{

public void m1() {

System.out.println("hello");

}

}

javac –d . A.java

javac –d . B.java

CompileTimeError: cannot find symbol A

1, After giving fullyQualifiedName for A (import A)

CompileTimeError: com.modifiers.\_public.a.A is not public, cannot be access from outside package

2, After change class A as public

**Cwd**

package com.modifiers.\_public.b;

import com.modifiers.\_public.a.\*;

class B{

|\*A.java

|\*B.java

|- com

|- modifiers used it with create B.class here

|- \_public fullyqualifiedname

|- a |- b

|\*A.class |\*B.class

## <default>

If a class declared as Default then we can access it only with in that package

Default assess known as package level access

In order to access that default A we need to create B.class with in that same package

**Cwd**

package com.modifiers.\_default.a;

//import com.modifiers.\_default.a.\*;

class B{

|\*A.java

|\*B.java

|- com

|- modifiers no need to create B.class here

|- \_defalult fullyqualifiedname

|- a

|\*A.class

|\*B.class

## final

Can apply for class, method, and variable

### final method

#### override not allowed

class Parent{

public void property() {

S o p("money gold house");

}

public final void meriage() {

S o p("subulaxmi");

}

}

class Child extends Parent{

public void meriage() {

S o p("nivetha");

}

}

Whatever methods parent has is by default available to the child. If the child not satisfied with parent method implementation then child is allowed to redefine that method based on its requirement. (override)

If parent class method is declared as final, then child class cannot override that method. Because it’s implementation is final.

### Final class

#### Inheritance not allowed

class Child extends Parent{

+ 5 methods(){ }

}

final class Parent{

10 methods(){ }

}

compileTimeError: cannot inherit from final Parent

* Every method present in side final class is final. Because we can’t inherit from final class so can’t override.
* Every variable present in side final class need not to be final. Because we can change it’s value.

class A{

static final x = 10;

p s v main(String arg[]) {

x = 12;

}

### Advantage Disadvantage

Security Oops concepts missing

Unique implementation Inheritance

Polymorphism (override, overload)

If there is no specific requirement not recommended using.

## abstract

Applicable for class and method, but not for variable

### Abstract method

abstract class Vehicle{

abstract public int getNoOfWheels();

public final int getNoOfSheets() {

return 44;

}

}

class Bus extends Vehicle{

public int getNoOfWheels(){

return 7;

}

}

We can provide guide lines to the child class. Like which methods compulsory child has to implement.

Abstract method should ends with ;

Abstract method never talks about implementation if any modifier talks about implementation

abstract public class A{

    abstract public final void m1();

compleTimeError:

illegal combination of modifiers: abstract and final

### Abstract Class

For any java class if we are not allowed to create object (because of partial implementation) such class we need to declared with abstract modifier.

Instantiation (creating instance) is not possible

abstract class Vehicle{

abstract public int getNoOfWheels();

public final int getNoOfSheets() {

return 44;

}

}

Vehicle v = new Vehicle();

compleTimeError:

Vehicle is abstract;

cannotbe instatiated

#### Abstract class Vs. Abstract Method

If a class has at least one abstract method then that class should define as abstract.

Reason:

If class Vehicle is not abstract I can create objects

vehicle.getNoOfWheels();

There is no answer

So we can’t allow to object instantiation

Even though class doesn’t contains any abstract method. Still we can declare class abstract if we don’t want instantiation.

So abstract class can have 0 number of abstract methods also.

Ex:

* HttpServer is not containing any abstract method but it’s a abstract class
* Every **adapter** class is recommended to declare as abstract but it doesn’t contain any abstract method

    public void m1();

compileTimeError: missing method body, or declare abstract

public class A{

    abstract public void m1();

compileTimeError:

A is not abstract and does not override abstract method m1() in A

abstract class A{

    abstract public void m1();

}

class B extends A{ }

compileTimeError:

B is not abstract and does not override abstract method m1() in A

### Final vs abstract

1. Abstract method compulsory should override to provide implementation. Whereas we can’t override final methods hence final abreact combination is illegal for methods.
2. For final classes we can’t create child classes whereas for abstract classes we should create child class for provide implementation. Hence final abstract combination is illegal for classes.

It is highly recommended to use abstract key word. Provide several oop features like polymorphism, inheritance

## strictfp

Strictfp applicable for class and method but not for variable

Usually the result of floating point mathematic is varid from platform to platform. If we want platform independent result then we should go for strictfp method.

### Strictfp methods

class B {

strictfp public double m1(){

return 23.0/7;

}

floating point operations should

follw IEEE 754 standard

strictfp methods always talks about implementation but

abstract method never talks about implementation. Hence

abstract stricfp combination is illegal for methods

class B {

strictfp abstract double m1();

### Strictfp classes

If a class declared as stricfp then every floating point operation should follow IEEE standards so we can get platform independent result.

abstract strictfp class B {

public double m1(){

return 23.0/7;

}

m2(){}, m3(){}, .. m10(){}

abstract int m11();

abstract int m12();

All concrete methods m1..m10

Has to follow

IEEE 754 standards

Class level there is no conflict between abstract, strictfp combination

# Member level modifiers

(Method or variable level modifiers)

## Public members

If a member declares as public we can access that members from anywhere

package pack1;

class A {

public double m1(){

return 23.0/7;

}

}

package pack2;

import pack1.A;

class B {

p s v main(String arg[]){

A a = new A();

Sop(a.m1());

CompileTimeError: pack1.A is not public, cannot be access from outside package

But corresponding class should be visible. That is before checking member visibility we have to check class visibility.

So if class and methods are public then only we can access it from outside package

## Default members

If member declared as default then we can access that member only within that package.

That is from outside package we can’t access.

package pack2;

import pack1.A;

class B {

p s v main(String arg[]){

A a = new A();

Sop(a.m1());

package pack1;

public class A {

double m1(){

return 23.0/7;

}

}

CompileTimeError: m1() is not public in A; cannot be accessed from outside package

## Private members

If member is private that class should be accessible within that class

From outside of the class we can’t access

### Abstract and private

class Child extends Parent {

private abstract m1(){

return 23.0/7;

}

public abstract class Parent {

private abstract m1();

}

Abstract methods need to be available to child classes to provide implementation, but private methods are not available to child classes. Hence private abstract combination is illegal for methods.

## Protected members

If a member declared as protected then we can access that member anywhere within the current package. But only in child class outside package.

Package hello;

Protected = default + child

package com.modifiers.\_public.a;

public class A{

protected void m1(){

sop("miss understod");

};

}

class C extends A{

p s v main(String[] args) {

A a = new A();

a.m1();

C c = new C();

c.m1();

A ac = new C();

ac.m1();

}

package com.modifiers.\_public.b;

import com.modifiers.\_public.a.A;

class B extends A{

p s void main(String[] args) {

A a = new A();

//a.m1(); ---|1|

B b = new B();

b.m1();

A ab = new B();

//ab.m1(); ---|2|

}

Line 1, 2 is wrong. Remaining code is perfectly valid.

compileTimeError:

m1() has protected access in A

We can access protected member within the current package anywhere either by using parent reference or by using child reference.

But we can access protected member in outside package only in child classes, and we should use that child reference only.

package com.modifiers.\_public.d;

import com.modifiers.\_public.b.B;

class D extends B{

p s void main(String[] args) {

A a = new A();

a.m1();

B b = new B();

b.m1();

D d = new D();

D.m1();

A ad = new D();

ad.m1();

A ab = new B();

ab.m1();

B bd = new D();

bd.m1();

}

A is grand parent

protected m1(){

B is parent

D is child

D d = new D()

d only can access m1().

A a = new A(), B(), D()

a can’t access m1().

B b = new B(), D()

b can’t access m1().

## Summary of visibility

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **visibility** |  | **private** | **default** | **protected** | **public** |
| Within same class |  |  |  |  |
| From child class of same package |  |  |  |  |
| From non-child class of same package |  |  |  |  |
| From **child class** of outside package |  |  | should use child reference only |  |
| From non-child class of outside package |  |  |  |  |

The most restricted access-modifier is private; most restricted access-modifier is public.

private< default< protected< public

Recommended modifiers for data members (Variables)

Private

Recommended modifiers for methods

Public

## Final

### instance variable (obj level variable)

If the value of a variable is varied from obj to obj is called instance variable.

Class Student{

Int id;

String name;

}

For every instance obj a separate copy of instance variable will be created

Class Test{

Int x;

P s v main(String arg[]){

Test t = new Test();

Sop(t.x); // 0

For instance variable we don’t need to perform initialization, jvm will provides default values.

### Final instance variable

class Test{

final int x;

P s v main(String arg[]){

Test t = new Test();

Sop(t.x); // 0

compileTimeError:

variable x not initialized in the default constructor

If the instance variable declared as final then compulsory we have to perform initialization explicitly. Whether we are using or not

, and jvm won’t provide default value

For final instance variable we should perform initialization before constructor completion

That is the following are various places for initialization

#### At the time of declaration

class Test{ class Test{

final int x = 0; final int x;

} x = 2;

compileTimeError: <identifier> expected

#### Inside instance block

(Instance block will be executed before constructor)

class Test{

final int x;

{

x = 0;

}

}

#### Inside constructor

class Test{

final int x;

Test(){

x = 0;

}

}

When assign value after constructor

class Test{

final int x;

void m1(){

x = 0;

}

}

compileTimeError:

cannot assign a value to final variable x

### Static variable (class level variable)

If the value of variable is not varied from obj to obj then we have to declared that variable at class level by using static modifier

In the case of instance variable for every obj separate copy will be created

In the case of static variable a single copy will be created at class level and sheared to every obj of that class

class Student{

int id;

string name;

static string school;

// default value “”

For static variable we don’t need to perform initialization, jvm will provides default values.

### Final static variable

class Test{

static final int x;

compileTimeError:

variable x not initialized in the default constructor

before class loading completion compulsory we should perform initialization

That is the following are various places for initialization

# Interfaces