**JAVA Source file structure:**

* A java program can have multiple classes and the name can be anything but for the main class, the .class file will be created and we can run only one main class at a time.
* If we use multiple main classes in one program, we get compile time error
* In a java program, more than one public program is not allowed
* If there is a public class, the name of the program and the public class should be same.
* If there is no public class, name of the program can be same.
* We cannot have multiple public classes.

Implicit Import: we can give the import statement implicitly as in java.util.\* i.e., we can use any class from the package java.util

Explicit Import: we can give the import statement explicitly as in java.util.ArrayList i.e., we can use only Arrayist class from java.util

For readability purpose, we have to use Explicit Import.

We donot need to import two packages in a program. The import is by default implemented in the program:

* Java.lang.\*
* Default package which is the current working directory package

**Class Modifiers:**

Public—Can be used in other packages as well by importing the package

<default>--cannot be used in other package. We can use the default class only in the current package. If we use this class in other packages, then system throws compile time error.

Abstract—

* Implementation is not proper/partial implementation and cannot be instantiated. If there is atleast one abstract method, then the class should be abstract class.
* Abstract method will have only declarations but not implementation. The implementations/definition will be defined in the child classes and child classes should contain the implementation of all abstract methods of abstract parent class

Example of abstract method: public abstract void getwheels();

If the abstract method ends with “;” then it is the declaration and called abstract class.

If a method has a {} then we can say it has implementation and not abstract class

* Abstract class may not have abstract method. Abstract class can have zero abstract methods as well.

Final—We cannot create a child class for the final classes and final variables cannot be used for other references.

Strictfp-Strict FloatingPoint

If there is an inner class. i.e., inside a class, if we have another class, then the other modifiers for inner classes are:

Protected: Can be used in the same package and if it is outside the package, we can use only in the child classes with child references only.

EExamples of protected method

**Example1**: Pckg pack1

Public void class A{

Protected void m1(){

Sysout(“this is protected method 1”)

}

}

Public class B extends A{

Psvm(String[] args){

A a=new A();

a.m1();

B b=new B();

b.m1();

A a1=new B();

a1.m1();

}

o/p: this is protected method 1

this is protected method 1

this is protected method 1

**Example 2:**

Pckg pack1

Public void class A{

Protected void m1(){

Sysout(“this is protected method 1”)

}

}

Packg pack2

Import pack1.A

Public class C extends A{

Psvm(String[] args){

A a=new A();//--compile time error because, protected method cannot be used on parent reference

a.m1();

C c=new C();

c.m1();//--No error as it is the child reference.

A a1=new C();

a1.m1();//--compile time error because, protected method cannot be used on parent reference

}

}

o/p: this is protected method 1

Private: can be used within the class

Static

Private<default<protected<public

**OOPS Concepts:**

Interface: This has just declaration of the class but not implementation. The implementations will be there in the child classes.

Interface A{

Public void m1();

Public void m2();

}

Class B implements A{

Public void m1(){

}

Public void m2(){

}

}

If there is no implementation for at least one method, system throws compile time error.

We can either give the implementation or we can derive the class as abstract class as there is partial implementation.

If all methods are not implemented then we have to create a subclass of the child

Class SubB extends B{

Public void m2(){

}

}

Data Hiding:

We have to use private variables and give the validation in the public to return the private variable.

For example: To open a mail, we have ti give userid and password.so user id and password will be public and depending on the validation, the private will be called.

Class A{

Private double interest;

…

Public void getinterest(){

Validations

Condition and based on that return interest;

}}

Abstarction:

Hiding the implementation and showing the declarations/services that we offer.

Eg: Bank ATM will have withdraw, statement etc. We can just see these services but we cannot see the implementation of these. This is abstraction.

Encapsulation:

grouping of all member variables and its implementation is encapsulation. Java class is the example of encapsulation.

Encapsulation=DataHiding+Abstraction

Tightly Encapsuled: If the parent class contains private variable, that particular class is tightly encapsuled:

Class A{

Private int x=1;

}

Class B extends A{

Int y=7;

}

Class C extends A(

Private int z=9;

)

Here ClassA is tightly encapsuled.

Class B is not tightly encapsuled because,it does not contain private variable.

Class C has private variable and its parent also has provate. Hece C is tightly encapsuled.

Class A{

int x=1;

}

Class B extends A{

Private Int y=7;

}

Class C extends B(

int z=9;

)

Class A is not tightly encapsuled because, there is no private variable.

Class B is not tightly encapsuled because, its parent is not .

Class c is not tightly encapsuled because its parent is not.

Inheritance:

* Also known as Is a relationship
* Code Reusability
* Can implement using extends keyword

Example:

Class P{

Public void m1{  
sysout(“Parent class”)

}}

Class C extends P{

Public void m2{  
sysout(“Child class”)

}}

Class Test{

Psvm(String[] args){

P p=new P();

p.m1();

p.m2();--Compile time error

C c=new C();

c.m1();

c.m2();

P p=new C();--parent class reference can be used to create child class object but cannot call child class methods

p.m1();

p.m2()—compile time error as parent cannot call child class methods

C c=new P();==compile time error as child reference cannot be used to create parent class object

}}

* The super class for java is Object. The methods in this class are used in all the classes
* The super class for the exception and errors is Throwable

Types of Inheritance:

1. Single Inheritance: If single child class extends single parent class, then its called single inheritance
2. Multiple Inheritance: The process of extending multiple parent classes at a time is multiple inheritance. Not supported in java

If there are same methods in two parent classes, then child class will face ambiguity problem to which method needs to be used in child class as the names are same.

For Interfaces this is possible because, Inherence just have declarations and definition can be given in implementation class and hence no ambiguity.

Interface A{

M1();}

Interface B{

M1()}

Interface C extends A,B{

Definition for m1(){

}}

1. Multilevel Inheritance: class A🡨Class B**🡨**Class C
2. Hierarchical Inheritance: Reverse of multiple inheritance.

Classs A{}

Class B extends A{}

Class C extends A{}

Group of single inheritances are hierarchical inheritance

1. Hybrid Inheritance: If group of inheritances are used at a time. Not supported in java

**Method Signature:**

Class A{

Public void m1(int i){}

Public int m2(int i){}

}

* Methodsignature is m1(int i) and m2(int i)
* While calling the method in another class with A class reference, compiler will check the method table with the proper method signature to resolve the method call.

Class B{

Public static void main(String[] args){

A a=new A();

a.m1(20);--signature is m1(int i)

a.m2(10)—signature is m1(int i)

a.m3(“Divya”);--signature is m3(String a)—there is no method signature in class A to resolve the method call. Hence we get compile time error}}

* Two methods with same name and signature should not be present though the return type is different.

Example:

Class A{

Public void m1(int i){}

Public int m1(int j)—compile time error because while calling the method, compiler will be confused to take the valid method with same signature{

Return 10;}}

**Method Overloading:**

A method with different args and same name is called method Overloading

Class test{

Public void m1(int i){

Sysout(“int arg”)--}

Public void m1(String s){

Sysout(“string args”)}

Public void m1(Float f){

Sysout(“float args”)}

Public void m1(double d){

Sysout(“double args”)}

Public void m1(String Buffer s){

Sysout(“String buffer args”)}

Public void m1(Object o){

Sysout(“object”)}

Public void m1(int… i){

Sysout{“int var args”}}

Class A{

Psvm(string[] args){

Test t=new test();

t.m1(10)/\*—As there are two methods ie., m1(int i ) and ,1(int… i)—compiler would first give preferemce to m1(Int I) as this is the old datatype and varargs are introduced later. The out put is int args\*/

t1.m1(10.5)/\*This will point to m1(foat f) method and if float is nnot present, compiler will search for the next level data type ie., long. The series is int🡪float🡪long🡪double

so if float is not available, compiler will search for long value method and output is printed accordingly otherwise, system throws compile ti,e error\*/

t.m1(“”divvya”);//string args. As object is the parent class this method call satisfies both object and string args method, in this case the preference goes to child

t.m1(new StringBuffer())”;//--String bffer args

}

}

All above are method overloading

Public void m1(int I,floaf f){

Sysout(“int float”)}

Public void m1(float f, int i){

Sysout(“float int”)}}

t.m1(10,10.5f)—int float

t.m1(10.5,10)—float int

t.m1(10,10)—as both satisfy this, we get Compile time error

t.m1(10.5,10.5)—compile time error

class Animal{

}

Class lion extends Animal{

Class test

{

Public void m1(Animal a){

Sysout(“animal”)}

Public void m1(lion l){

Sysout(“lion”)}

Psvm(string[] args){

Test t-new test();

Animal a-=new Animal();

t.m1(a());--animal

lion l-=new lion()  
;

t.m1(l));--lion;

Animal a1=new lion();

t.m1(a1)—animal because compiler uses to resolve method resolution is based on reference type but not runtime object}

**Overriding**:

Class p{

Public void m1(){

Sysout(“parent method”)}}

Class child extends p{

Public void m1(){

Sysout(“child one”)}}

Class test{

Psvn(string[] args){

P p=new P();

p.m1()—parent method

child c=new chicd();

c.m1()—child one

P p1=new child();

P1.m1()—child one: here based on runtime object, method resolution will be done by jvm}}

If the implementation in the parent class needs to be changed according to child class, we can give the same method name but with different implementation.

Overriding is also known as Dynamic Polymorphism, Runtime polymorphism, late Binding.

* In overriding, method signature must be same.
* From 1.5version, covariant return types are allowed

Class p{

Public object m1(){}

Return null;}

Class c extends p{

Public String m1(){}

Return null;}

* If we use the child of object like string as return type, then this is called co-varient
* Wrapper classes of Number🡪byte, short, integer, long, float, double
* Covariant return types are not applicable for primitives but applicable only for object types
* Private and final methods cannot be overridden.
* While overriding, there is no restriction for unchecked exceptions but when the child class throws checked exception, the parent class should throw child checked exception or its parent

Throwable:

Exception:

RunTime Exception:

Null Pointer exception

Arithmetic Exception

Class Cast Exception etc

IO Exception:

FNFE

EOF Exception

IE:

SE etc

Error:

VM exception

Exception Runtime and Error exceptions, all other exceptions are checked exceptions

* The modifier in child class in overriding should not be below the modifier of parent:

Public>protected>default>private

Class p: public

Class c: protected

O/p: compiler time error

Class p: default

Class c: protected

o/p: allowed

Method Hiding:

If child and parent has static methods in overriding, then this is called static method.

The method resolution in method hiding will be taken care by compiler based on reference type.

Non static to static or static to non static/instance methods overriding is not possible

Overriding with varargs:

If the child class method overrides parent class method using var args, then it is method overloading as we are changing the data type.

Class P{

public void m1(int… i)

}

Class c extends p{

Public void m1(int 1){}

}

Or vise versa is overloading. We cannot override var args method

Overriding with variables:

Class P{

String s=”divya”;

}

Class c extends p{

String s=”Advika”;}

Class test{

P p=new p();

Sysout(“p.m1”)--divya

C c=new c();

Sysout(“c.m1”)--Advika

P p1 =new c();

Sysout(“p1.m1”)--divya

}

The above example is called shadowing. In shadowing, the method resolution is done by compiler based on the reference type.

Three pillars of OOPs:

Encapsulation: built for security

Inheritance: built for reusability

Polymorphism: Has many forms and for flexibility

Object TypeCasting:

Three rules:

Object o=new String(“divya”);

String s=(String)o;--no error

StringBuffer sb=( StringBuffer)o;--rerror because, there is no relation between String and stringbuffer, we get runtime error.

A b=(c)d;

Rule 1: c and d has to be related like child to parent or parent to child. Otherwise we will get compile time error

Rule 2: (c)d is now c type and hence c should be either same as A or should be child of A. Otherwie we will get compile time error

Rule 3: c should be same as d or d should be the derived type of c. Otherwise we get runtime error.

* While typecasting, a new ref variable will be assigned to the same object.