Resource: <https://www.udemy.com/join/login-popup/?next=/object-oriented-programming-oops-for-java-certification/learn/v4/overview>

**Lec 1:**

* A java program can contain any number of classes.
* But can only contain zero or one public class
* If there is no public class, you can name java program anything
* If there is a public class, then the name of the program must match the name of public class
* When java program compiles it creates .class files for each class defined in that program
* Each .class file can be executed separately and will run main method defined in it if any
* If no main method defined, face an error!
* java <classname> (cmd line command to run java program)
* javac <javafileName> (cmd line command to compile java program)

**Lec 2:** **(import)**

* Import is used to make the pre-written java classes available to the program
* Explicit import. E.g., import java.util.AL
* Implicit import (make all the classes present inside this pkg available) e.g., import java.util.\*

**Lec 3: (import)**

* No need to write import statements for these packages
* Java.lang and default pkg (current working directory)
* because classes from these pkgs are commonly required by every program
* when importing a pkg, all classes and interfaces in that pkg are by default available but not the classes and interfaces in sub pkgs
* if you need any class from a sub pkg u need to write the import statement until that sub pkg level.
* For example if we want to use pattern class which is available in java.util.regex.pattern then:
  + Import java.util.regex.pattern.\* (valid)
  + Import java.\* (invalid)
  + Import java.util.\* (invalid)
  + Import java.util.regex.\* (invalid)

**Lec 4: (package)**

* Encapsulation mechanism to group related classes and interfaces into a single unit
* Every class should be part of some pkg
* Packages:
  + avoid naming conflicts (Date class from util pkg and sql pkg can be differentiated using FQPN)
  + improve modularity (by packging all the related classes/interfaces together)
  + improve maintainability
  + provide security (e.g.: a class that is not public cannot be accessed outside the pkg)
* Pkg name can be declared using package keyword
* Packages should be named by using internet domain name in reverse (e.g.: com.ferguson.utils)
* Javac -d . test.java (will compile program and place the .class file inside the pkg)
* Use FQN to run java class inside the pkg (e.g.: java com.ferguson.utils.test)

**Lec 5: (package)**

* In any java source file (means java prgrm) atmost 1 pkg name is allowed
* In any java program if pkg name needs to be declared, first non-comment statement should be the pkg name
* Following order is important:
  + Pkg statement
  + Import statement
  + Class/interface/enum decleration

**Lec 8,14, 15 & 16: (class level modifiers)**

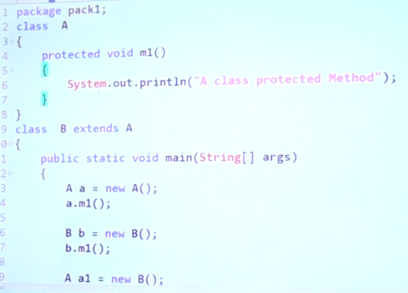
* Whenever writing a class we have to specify some info about that class to the JVM
* Modifiers describe the behavior of the class.
* Some of the modifiers for top level classes:
  + Public: access class from anywhere (global level)
  + <default>: accessible only within the pkg (pkg level)
  + Abstract: Incomplete implementation, object creation is not possible
  + Final: child class creation is not allowed
* For inner classes all of the above plus the following:
  + Private: accessible only within the same class (class level)
  + Protected: accessible within the same pkg anywhere. From outside pkg accessible only in child classes and only using child reference (Protected = <default> + kids)
  + Static
* Recommended modifier for variable is Private (bcz data hiding is essential in OOP)
* Recommended modifier for method is Public (bcz methods are kind od service and should be available to maximum users)
* Wrt scope:

private<default<protected<public

* All of the modifiers described above are only applicable at class level not method level except Final.
* Example of protected methods:

1. Within same package

m1() can be called successfully by following ways: (with both parent reference and child reference)





1. If we consider the same example but move the child class to another pkg then:

m1() can only be called by using child class reference (only ‘B b = new B()’ will work). Because in outside pkgs protected members can only be accessed in child classes and only using child references.

**Lec 9: (abstract method)**

* Abstract methods have only declaration but not implementation. E.g.:
  + For a class vehicle, if we want to tell the num of wheels it has, we can’t tell until we know which type of vehicle it is so we can declare an abstract method for that as follows.
  + public abstract int getNumOfWheels();
  + (notice statement ends with ; instead of {})
* Child classes are responsible to provide implementation of these abstract mehods

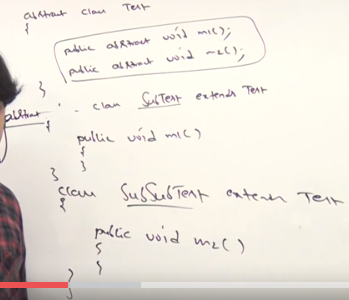
**Lec 10: (abstract class)**

* Partially implemented class = abstract class
* It may contain abstract methods or incomplete implementation of methods
* If the implementation of the class is not complete, we must declare them as abstract
* If a class is abstract no one is allowed create objects or call its methods directly
* If a class contains at least one abstract method, class should be declared as abstract because class is not complete hence object should not be created for that class
* If you feel implementation of class is not complete feel free to declare as abstract even if it doesn’t have any abstract method

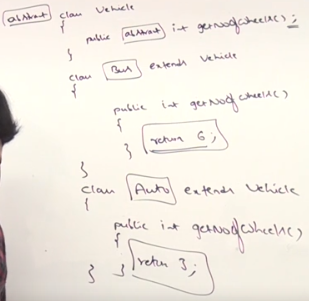
**Lec 11: (abstract class vs method)**

* Whenever you extend an abstract class (means create a child class) you must provide implementation for each abstract method
* If you don’t want to provide implementation for all the abstract methods of the parent class, you can declare child class as abstract. In this case next level child class is responsible for providing the implementation of the remaining abstract methods.

Ex:



* An example using abstraction:



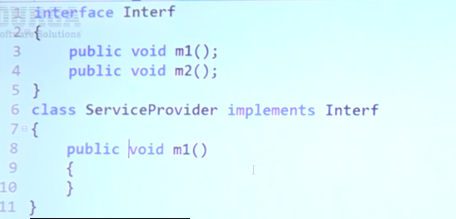
An object can be created for class Bus and Auto but not for class Vehicle.

* **Why do we need abstraction?**

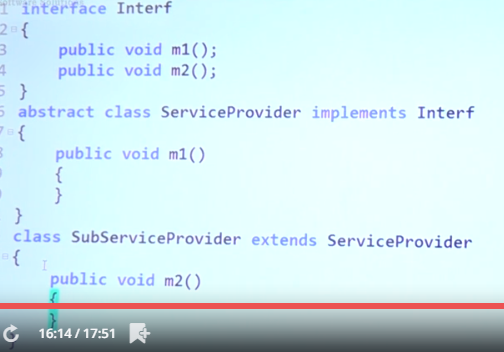
If we have certain methods that are mandatory to implement, we can declare them as abstract methods in the class. In this way any child class extending our class will be forced to provide implementation for these methods. Just as in the example of Vehicle class above

**Lec 19, 20: (interface)**

* Any service requirement specification or any contract b/w client and service provider is considered as interface. It does not talk about implementation
* Whenever implementing any interface method, compulsory declare it as public. Because methods in an interface are public by default and while implementing an interface you cannot reduce the scope while overriding the interface methods.



* Whenever implementing interface we should be providing implementation for each and every abstract method in that interface. If can’t provide implementation for all methods declare class as abstract.
* But then child class of this class has to provide implementation for this method



**Misc:**

* Parent child class concept

Consider the following classes:

(parent class A)

Class A

{

Public void m1()

{

}

}

(child class B)

Class B extends A

{

}

Now we can create object and call m1() in following ways:

A a = new A();

a.m1();

B b = new B();

b.m1();

A a = new B(); (parent reference can be used to hold child objects)

a.m1();