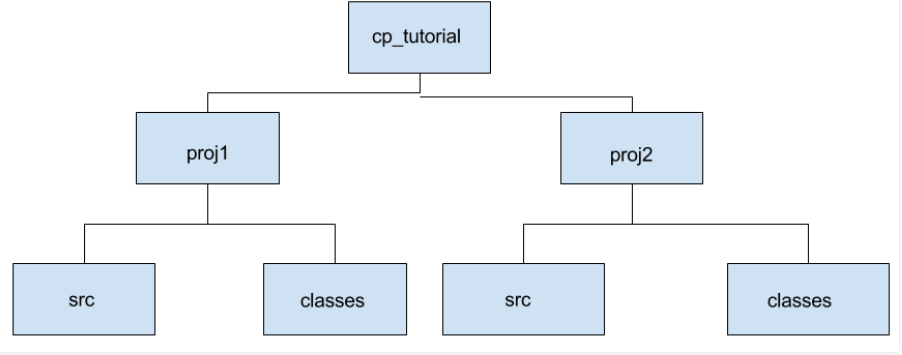
1. **Process oriented programming:** It defines a series of steps to manipulate that manipulate data.
2. **Object oriented programming:** It organizes a program around data i.e., objects and set of well- defined interfaces to that data.
   1. The programming challenge was how to write the logic, not how to define the data. Object-oriented programming takes the view that what we really care about are the objects we want to manipulate rather than the logic required to manipulate them.
3. **OOP Principles**: OOP has four cornerstones-
   1. **Abstraction**: Abstraction/Hierarchical abstraction is a process through which a complex system is broken down to its component objects with each object defining their unique set of behavior.
      1. Abstraction lets you focus on what the object does instead of how it does.
   2. **Encapsulation**: Encapsulation is the mechanism that binds together data and code that manipulates it and protect it through well-defined interface.
      1. Abstraction hides details at the design level, while Encapsulation hides details at the implementation level.
      2. The basis of encapsulation in java is a Class.
   3. **Inheritance**: It’s the process by which one object acquires properties of another object.
      1. Inheritance defines a IS-A relationship.
      2. Through inheritance a child class need to define attributes specific to them and inherit the general behavior from parent class.
      3. Inheritance promotes code reuse.
   4. **Polymorphism**: **Polymorphism is declaring a uniform interface that isn't type aware, leaving implementation details to concrete types that implement the interface.**

**polymorphism** means that objects that belong to the same branch of a hierarchy, when sent the same message (that is, when told to do the same thing), can manifest that behavior differently.

1. Two qualities differentiate the Java language from purely object-oriented languages such as Smalltalk. First, the Java language is a mixture of objects and primitive types.
2. **Classpath**: Classpath is the location from where jvm starts execution of a program.
   1. import keyword is used in Java to import classes from current project’s classpath. **You can import classes from different packages but from same classpath. It is to be remembered that packaging of a class starts from classpath.**
   2. If you want to use a class from proj1 in proj2 then we need to include the classpath of proj1 in the command while compiling ad running program.
      * 1. cp\_tutorial/proj2/src>javac -d ../classes -cp ../../proj1/classes MainClass.java
        2. **-d option:** It is used to store the output to different directory. If we don’t use this option then class file will be created in src directory.
        3. java -cp ../../proj1/classes; MainClass
   3. If we use jar files then replace classes folder with jar folder name.



1. Main method in Java is entry point for any core Java program.
   1. public static void main(String args[])/ public static void main(String … args)
   2. Since the main method is static Java virtual Machine can call it without creating any instance of a class which contains the main method.

If main method were not declared static than JVM must create instance of main Class and since constructor can be overloaded and can have arguments there would not be any certain and consistent way for JVM to find main method in Java.

* 1. Since the main method is public in Java, JVM can easily access and execute it.
  2. you can use a final, synchronized and strictfp modifier in the signature of the main method in Java.
  3. java 1.7 if there isn't any main method that would be a runtime error even if there are static blocks present in the class.

1. **Association**: If two objects are related to each other they it’s called association.

Association A---->B

* 1. **Composition**: If A and B two classes are related to each other such that, B ceased to exist, when A is destroyed, then the association between two objects is known as Composition.
     1. **Ex**: An example is Car and Engine.
     2. Composition A-----<filled>B
     3. you can use [final keyword](http://javarevisited.blogspot.sg/2011/12/final-variable-method-class-java.html) to represent Composition. Since in Composition, Owner object expects a part object to be available and functions, by making it final, you provide guarantee that, when Owner will be created, this part object will exist.
     4. If one object is-part-of another object e.g. Engine is part of Car, then association or relationship between them is Composition.

**public** **class** **Car** {

//final will make sure engine is initialized

**private** **final** Engine engine;

**public** **Car**(){

engine = **new** Engine();

}

}

**class** **Engine** {

**private** String type;

}

* 1. **Aggregation**: While if A and B are associated with each other, such that B can exist without being associated with A, then this association in known as Aggregation.
     1. **Ex**: An example is Company and Employee.
     2. Aggregation A-----<>B
     3. Aggregation is a lighter form of Composition, where a sub-part object can meaningfully exist without main objects.
     4. if one object just has another object e.g. Car “has” the driver then it's Aggregation.

**public** **class** **Organization** {

**private** List employees;

}

**public** **class** **Person** {

**private** String name;

}

1. **Cohesion**: Cohesion indicate the degree to which a class has a single, well-focused purpose.
   1. Benefit of high cohesion is that classes with a well-focused purpose tend to be more reusable than other classes.
   2. high cohesive classes are typically much easier to maintain.
   3. **Ex**: Model class performing DB operations.
2. **Coupling**: Coupling is the degree to which one class knows about another class.
   1. If the only knowledge that class A has about class B, is what class B has exposed through its interface, then class A and class B are said to be loosely coupled that is good thing.
   2. **Ex**: A class using instance variable of another class.