**OOPS**

**Interface or Abstract Class**

**What is the difference between an Abstract Class and an Interface?**  
Well, isn't it an interesting question or for few, a foolish question?  
  
For a number of Java newbies, the only difference between two is, "You can't have method body in Interface.". Well, I would say that it is technically (Implementation vise) correct, but doesn't reflect the concept.  
  
**Abstract Class**  
An abstract class, is the most generalized form of an Entity hierarchy (In Java, a Class is an entity), such that it doesn't represent a real entity on its own, but provides a basis on which real entities can be build within that hierarchy. An abstract class provides a default behavior and also provide abstract behavior (Operations that must be override by the real classes).  
  
**Interface**  
An interface, on the other hand is a contract that states that you have to agree to provide following operations if you agree to abide by me, i.e. an entity that agrees to abide by the contract of interface, will have to provide the behavior (operations) declared in interface.  
  
**Interface or Abstract Class**  
Next question is when to use abstract class and when use an Interface? Though it many a depends on requirements as well, but I follow a thumb rule,

* If I have to build a class hierarchy, I should consider generalization. Many a time I will reach to most crude form of generalization, i.e. at an abstract level. I would declare abstract methods and will also define common behavior for few of the methods. This would be my abstract class.
* On the other hand, if I want to expose a particular behavior to the client, then I would define an interface.

Further, I would suggest that one should go through different Java APIs, and spend some time understanding when and why they used interface or abstract class. Java APIs are excellent source for learning on building design blocks for any OOPs language.

**Association, Aggregation, Composition, Abstraction, Generalization, Realization, Dependency**

These terms signify the relationships between classes. These are the building blocks of object oriented programming and very basic stuff. But still for some, these terms look like Latin and Greek. Just wanted to refresh these terms and explain in simpler terms.

**Association**

Association is a relationship between two objects. In other words, association defines the multiplicity between objects. You may be aware of one-to-one, one-to-many, many-to-one, many-to-many all these words define an association between objects. Aggregation is a special form of association. Composition is a special form of aggregation.

http://javapapers.com/wp-content/uploads/2010/06/association.jpg

***Example:*** A Student and a Faculty are having an association.

**Aggregation**

Aggregation is a special case of association. A directional association between objects. When an object ‘has-a’ another object, then you have got an aggregation between them. Direction between them specified which object contains the other object. Aggregation is also called a “Has-a” relationship.

http://javapapers.com/wp-content/uploads/2010/06/aggregation.jpg

***Example:*** A car has a stereo system. A stereo system can exist without a car. There exists aggregation between car and stereo.

**Composition**

Composition is a special case of aggregation. In a more specific manner, a restricted aggregation is called composition. When an object contains the other object, if the contained object cannot exist without the existence of container object, then it is called composition.

http://javapapers.com/wp-content/uploads/2010/06/composition.jpg

***Example:*** A class contains students. A student cannot exist without a class. There exists composition between class and students.

**Difference between aggregation and composition**

Composition is more restrictive. When there is a composition between two objects, the composed object cannot exist without the other object. This restriction is not there in aggregation. Though one object can contain the other object, there is no condition that the composed object must exist. The existence of the composed object is entirely optional. In both aggregation and composition, direction is must. The direction specifies, which object contains the other object.

***Example:***A Library contains students and books. Relationship between library and student is aggregation. Relationship between library and book is composition. A student can exist without a library and therefore it is aggregation. A book cannot exist without a library and therefore its a composition. For easy understanding I am picking this example. Don’t go deeper into example and justify relationships!

**Abstraction**

Abstraction is specifying the framework and hiding the implementation level information. Concreteness will be built on top of the abstraction. It gives you a blueprint to follow to while implementing the details. Abstraction reduces the complexity by hiding low level details.

***Example:*** A wire frame model of a car.

**Generalization**

Generalization uses a “is-a” relationship from a specialization to the generalization class. Common structure and behaviour are used from the specializtion to the generalized class. At a very broader level you can understand this as inheritance. Why I take the term inheritance is, you can relate this term very well. Generalization is also called a “Is-a” relationship.

http://javapapers.com/wp-content/uploads/2010/06/generalization.jpg

***Example:*** Consider there exists a class named Person. A student is a person. A faculty is a person. Therefore here the relationship between student and person, similarly faculty and person is generalization.

**Realization**

Realization is a relationship between the blueprint class and the object containing its respective implementation level details. This object is said to realize the blueprint class. In other words, you can understand this as the relationship between the interface and the implementing class.

http://javapapers.com/wp-content/uploads/2010/06/realization.jpg

***Example:*** A particular model of a car ‘GTB Fiorano’ that implements the blueprint of a car realizes the abstraction.

**Dependency**

Change in structure or behaviour of a class affects the other related class, then there is a dependency between those two classes. It need not be the same vice-versa. When one class contains the other class it this happens.

http://javapapers.com/wp-content/uploads/2010/06/dependency.jpg

***Example:*** Relationship between shape and circle is dependency.