Android Code Test

Assumptions:

1. Comment and Annotation class have their own implementation and do not share is-a relationship between them.
2. Different remote stores (e.g Parse, REST, etc) have their own approach of storing classes. i.e Parse Remote store saves comment and annotation in different way. Likewise way of storing Comment and Annotation is different in Parse Remote store than REST Remote Store.
3. Focus here is on design pattern to define structural relationship between classes rather than looking for low level details like networking code. Since it does not involve any Android specific components, I did the code in Netbeans using Java.

Design Pattern Used: Dependency Injection Pattern
Based on the requirement there needs to be decoupling between the store-able content(i.e Comment, Annotation) and methods of storing them (i.e REST, Parse). I created RemoteStore interface which should be implemented by remote storage classes e.g CommentRestRemoteStore and CommentParseRemoteStore. Classes implementing RemoteStoreInjector interface are used to inject the concrete implementation classes. Use of enum distinguishes remote store types and a new type can be added any time. I used constructor based dependency injection because Application class would not exist without any remote store. If it requires that we need to update remote store instance after it's creation then we can also use setter based injection.

Why this pattern?
The main benefit of DI pattern is writing modular codes which can be tested easily. For this case we can just provide a mock implementation of remote store and inject it with new enum store type called MOCK(say). Also I did a simple JunitTest for Application class.

Why used enum?
I used enum because it will make code more readable. Assigning a static final int for different remote store types could bring issue in using Switch case because one can pass any integer value as parameter.
So enum also forces to pass a valid argument.

How is it open for extension?
If we have to store a new class (say Annotation), then it should implement Storable interface. Storable is a marker interface which just tells that the class can be saved in remote store. If Annotation class requires saving via REST remote store then a new class AnnotationRestRemoteStore should be created which implements RemoteStore and provides concrete implementation of storing Annotation class. If later Annotation class is required to save through Parse remote store then a new class AnnotationParseRemoteStore is created that implements RemoteStore and has its own concrete implementation. If there comes another remote store then we can specify a new enum type for it and then just created another implementation of remote store. All of this remote store classes can be injected by AnnotationRemoteStoreInjector based on enum RemoteStoreType. This does not need modification to the existing code. It provides an easy way to test code with mock implementation.

Making it easier
To handle the injection part, we can use Dagger(now Dagger2) . Dagger makes it easy to inject classes based on annotation and still maintain a testable code.