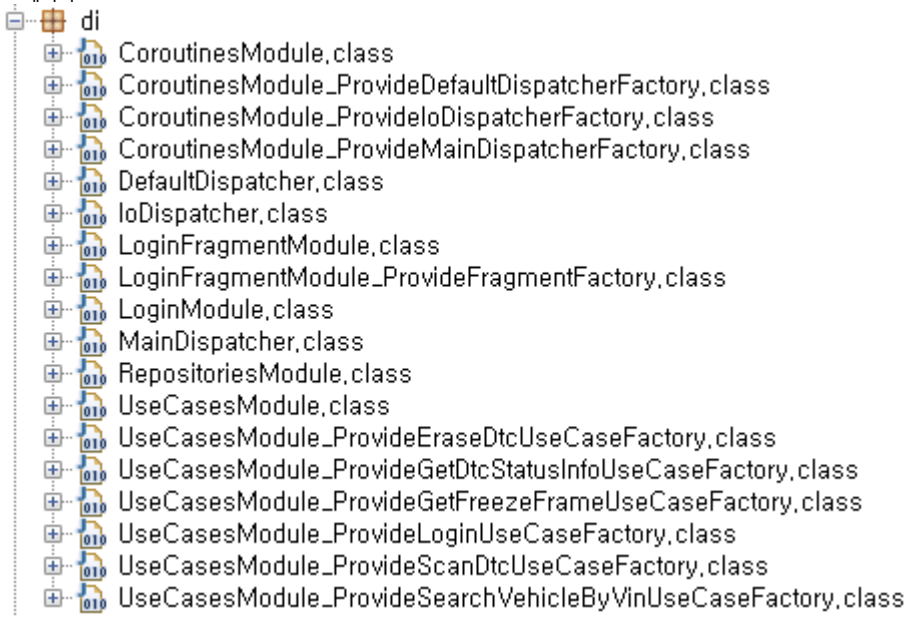
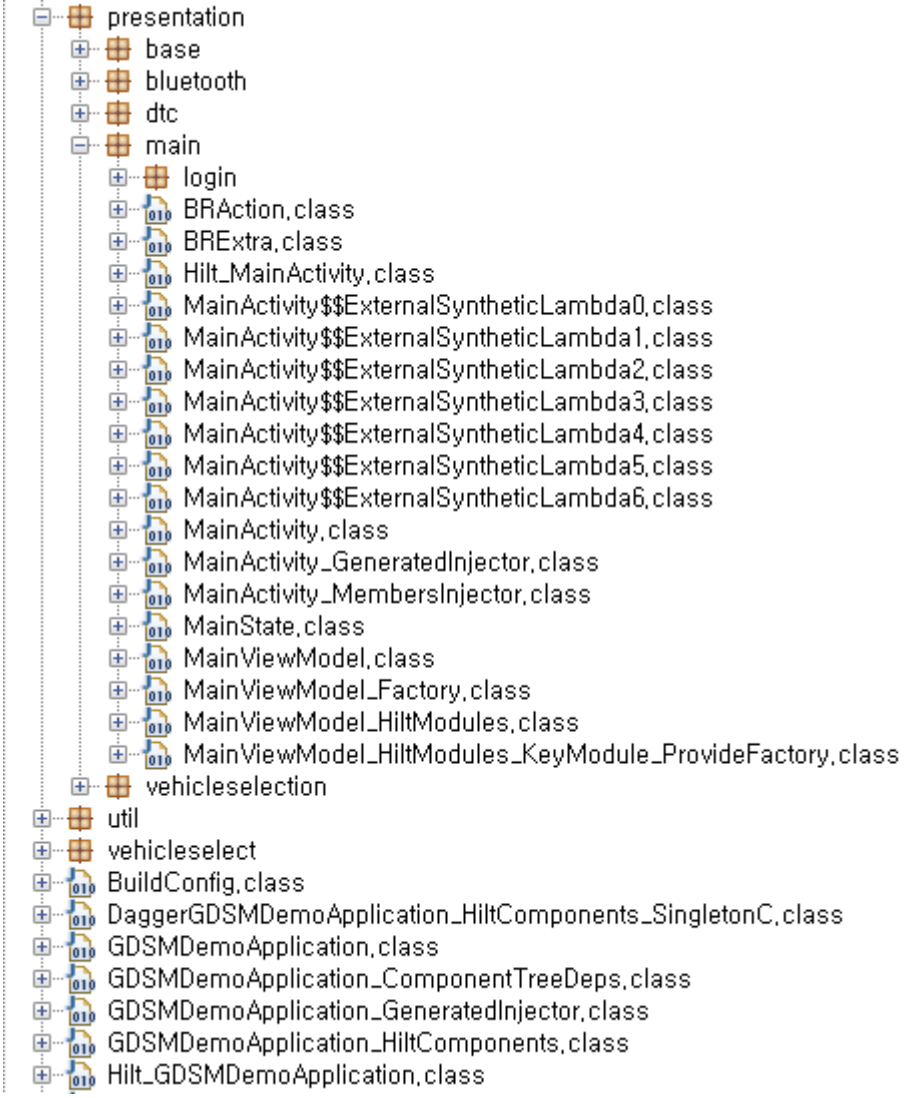
Injection 과정

[Dex Analysis]

- di 패키지



- Application 및 MainActivity 패키지



- Hilt_{앱명} 파일에는 @HiltAndroidApp 클래스가 생성된다.
  : 컴포넌트 생성 명령 포함

: applicationContext 를 Hilt Module 에 넣어주는 코드가 추가
: 이러한 과정이 있어서@ApplicationContext qualifier 로 context 를 주입받을 수 있게 해준다.

- {모듈명}_Provide{provide 타입}Factory 파일을 보면, 우리가주입해줄 값들의 인스턴스화가 진행된다.

```
public final class LoginFragmentModule_ProvideFragmentFactory implements Factory<LoginFragment> {
  private final Provider<Fragment> fragmentProvider;

  private final LoginFragmentModule module;

  public LoginFragmentModule_ProvideFragmentFactory(LoginFragmentModule paramLoginFragmentModule, Provider<Fragment> paramProvider) {
    this.module = paramLoginFragmentModule;
    this.fragmentProvider = paramProvider;
  }

  public static LoginFragmentModule_ProvideFragmentFactory create(LoginFragmentModule paramLoginFragmentModule, Provider<Fragment> paramProvider) {
    return new LoginFragmentModule_ProvideFragmentFactory(paramLoginFragmentModule, paramProvider);
  }

  public static LoginFragment provideFragment(LoginFragmentModule paramLoginFragmentModule, Fragment paramFragment) {
    return (LoginFragment)Preconditions.checkNotNullFromProvides(paramLoginFragmentModule.provideFragment(paramFragment));
  }

  public LoginFragment get() {
    return provideFragment(this.module, (Fragment)this.fragmentProvider.get());
  }
}
```

- Dagger{앱명}_HiltComponents_SingletonC 파일 :Hilt의 주입 기능들이 이루어 진다.

  : 먼저 각각 Provider 를 생성해 준다.

```
private void initialize(ApplicationContextModule param1ApplicationContextModule) {
  this.mainEventBusProvider = DoubleCheck.provider(new SwitchingProvider(this.singletonCImpl, 0));
  this.dtcManagerProvider = DoubleCheck.provider(new SwitchingProvider(this.singletonCImpl, 1));
  this.dBManagerProvider = DoubleCheck.provider(new SwitchingProvider(this.singletonCImpl, 2));
  this.xmlHelperProvider = DoubleCheck.provider(new SwitchingProvider(this.singletonCImpl, 4));
  this.vCICommProvider = DoubleCheck.provider(new SwitchingProvider(this.singletonCImpl, 5));
  this.sQLiteHelperProvider = DoubleCheck.provider(new SwitchingProvider(this.singletonCImpl, 6));
  this.dtcRepositoryImplProvider = DoubleCheck.provider(new SwitchingProvider(this.singletonCImpl, 3));
  this.appPreferenceManagerProvider = DoubleCheck.provider(new SwitchingProvider(this.singletonCImpl, 7));
  this.webDownloaderProvider = DoubleCheck.provider(new SwitchingProvider(this.singletonCImpl, 9));
  this.loginRepositoryImplProvider = DoubleCheck.provider(new SwitchingProvider(this.singletonCImpl, 8));
  this.vinSearchManagerProvider = DoubleCheck.provider(new SwitchingProvider(this.singletonCImpl, 11));
  this.vehicleSelectionRepositoryImplProvider = DoubleCheck.provider(new SwitchingProvider(this.singletonCImpl, 10));
}
```

- SwitchingProvider의 인자로 provider의 스코프와 id가 들어간다.

  이 provider의 id는

```
public T get() {
  switch (this.id) {
    default:
      throw new AssertionError(this.id);
    case 11:
      return (T)new VinSearchManager((SQLiteHelper)this.singletonCImpl.sQLiteHelperProvider.get());
    case 10:
      return (T)new VehicleSelectionRepositoryImpl((VinSearchManager)this.singletonCImpl.vinSearchManagerProvider.get(), CoroutinesModule_
    case 9:
      return (T)new WebDownloader();
    case 8:
      return (T)new LoginRepositoryImpl((WebDownloader)this.singletonCImpl.webDownloaderProvider.get(), CoroutinesModule_ProvideIoDispatch
    case 7:
      return (T)new AppPreferenceManager(ApplicationContextModule_ProvideContextFactory.provideContext(this.singletonCImpl.applicationCont
    case 6:
      return (T)new SQLiteHelper();
    case 5:
      return (T)new VCIComm(ApplicationContextModule_ProvideContextFactory.provideContext(this.singletonCImpl.applicationContextModule));
    case 4:
      return (T)new XmlHelper();
    case 3:
      return (T)new DtcRepositoryImpl((XmlHelper)this.singletonCImpl.xmlHelperProvider.get(), (VCIComm)this.singletonCImpl.vCICommProvider
    case 2:
      return (T)new DBManager();
    case 1:
      return (T)new DtcManager();
    case 0:
      break;
  }
  return (T)new MainEventBus();
}
```

  provider에서 특정 값을 가져오는데 사용이 된다.

  get()은 어디서 사용이 될까?

  동일 파일에서 inject{주입될 파일명}2라는 함수에서 사용된다.

```
public void injectMainActivity(MainActivity param1MainActivity) {
  injectMainActivity2(param1MainActivity);
}

private MainActivity injectMainActivity2(MainActivity param1MainActivity) {
  MainActivity_MembersInjector.injectMainEventBus(param1MainActivity, (MainEventBus)this.singletonCImpl.mainEventBusProvider.get());
  return param1MainActivity;
}
```

  이 함수를 보면 주입될 액티비티를 인자로 받고 있고, 이 액티비티를inject{주입받은 변수명} 메서드를 통해 사용되고 있다.

  이 메서드는 {주입될 액티비티명}_MemberInjector라는 파일에서 정의가 되어있다.

  이로써 inject{주입될 액티비티명}2 함수로 의존성 주입이 이루어지는 것을 확인할 수 있다.

이 함수는 언제 호출이 될까?

```
public abstract class Hilt_MainActivity<VM extends BaseViewModel, VB extends ViewBinding> extends BaseActivity<VM, VB> {
  private boolean injected = false;

  Hilt_MainActivity() {
    _initHiltInternal();
  }

  private void _initHiltInternal() {
    addOnContextAvailableListener(new OnContextAvailableListener() {
          public void onContextAvailable(Context param1Context) {
            Hilt_MainActivity.this.inject();
          }
        });
  }

  protected void inject() {
    if (!this.injected) {
      this.injected = true;
      ((MainActivity_GeneratedInjector)((GeneratedComponentManagerHolder)UnsafeCasts.unsafeCast(this)).generatedComponent()).injectMainActivity((MainActivity)UnsafeCasts.unsafeCast(this));
    }
  }
}
```

Hilt_{주입받을 액티비티명} 파일이 컴파일 되면서 @AndroidEntryPoint 가 붙은 클래스의 부모클래스가 이 클래스로 바이트코드에서 변환이 이루어짐.

이 파일을 보면클래스가 초기화 되면서 위에서 봤던 inject{주입될 액티비티명}를 호출하는 inject()함수를 호출하게 된다.

이로써 super.onCreate에서 주입이 어떻게 가능한지 알 수 있다.

[Injection Sequence Diagram]