Overview

- Our dagger graph could use a little optimazation.
- Everything is a @Singleton!
- Which means everything is in memory!

Binds vs Provides

- Binds does not create an extra factory classes
- Because Binds methods are just a method declaration, they are expressed as abstract methods — no implementation is ever created and nothing is ever invoked.
- Provides creates a Factory class for the provided object

Binds vs Provides

```
@Provides
public TimeoutHandler provideTimeoutHandler(PaakTimeoutHandler paakTimeoutHandler) {
    return paakTimeoutHandler;
}
```

```
@Binds
abstract TimeoutHandler provideTimeoutHandler(PaakTimeoutHandler paakTimeoutHandler);
```

```
@Provides
public PayloadProvider providePayloadProvider(ConsumerAccessKeyDataDao cakDataDao, PreferenceManager prefe
    return new PaakPayloadProvider(cakDataDao, preferenceManager);
}
```

```
@Binds
abstract PayloadProvider providePayloadProvider(PaakPayloadProvider paakPayloadProvider);
```

Static

- Static method calls are faster, particularly in Android, because they avoid a virtual method table lookup.
- If you don't need to access an object's fields, make your method static. Invocations will be about 15%-20% faster.

Static

```
@Provides
public Base64 provideBase64() {
    return new Base64Handler();
}
```

```
@Provides
static public Base64 provideBase64() { return new Base64Handler(); }
```

```
@Provides
public Sha256 provideSha256() { return new Sha256Handler(); }
```

```
@Provides
static public Sha256 provideSha256() { return new Sha256Handler(); }
```

Constructor Injection

- If object construction does not require special constructs to create.
- Removes unnecessary modules if all provided objects in the module can be instantiated via constructor injection.
- Straight forward.

Constructor Injection

```
@Provides
@Singleton
NgsdnUserSQLiteHelper provideNgsdnUserSQLiteHelper(@ForApplication Context context, @InMemoryStorage boolean useInMemoryStorage, DatabaseEncryp
    return new NgsdnUserSQLiteHelper(context, useInMemoryStorage, databaseEncryptionUtil);
}
```

```
@Inject
public NgsdnUserSQLiteHelper(@ForApplication Context context, @InMemoryStorage boolean useInMemoryStorage, DatabaseEncryptionUtil databaseEncryp
    super(context, useInMemoryStorage ? null : DATABASE_NAME, factory: null, DATABASE_VERSION);
    this.context = context;
    this.databaseEncryptionUtil = databaseEncryptionUtil;
    SQLiteDatabase.loadLibs(context);
}
```

Scopes

- We need to be careful of scopes
- We need to be mindful of holding every object into memory
- If an object is not needed anymore at a point in time it shouldn’t need to persist throughout the lifetime of the application.