## 4.3. Aggregating `@Configuration` classes with `@Import`

Thus far, we've seen how to break up bean definitions into multiple `@Configuration` classes and how to reference those beans across `@Configuration` boundaries. These scenarios have required providing all `@Configuration` classes to the constructor of a `JavaConfigApplicationContext`, and this is not always ideal. Often it is preferable to use an *aggregation* approach, where one `@Configuration` class logically *imports* the bean definitions defined by another.

The `@Import` annotation provides just this kind of support, and it is the direct equivalent of the `<import/>` element found in Spring beans XML files.

```
@Configuration
public class DataSourceConfig {
    @Bean
    public DataSource dataSource() {
        return new DriverManagerDataSource(...);
    }
}

@Configuration
@AnnotationDrivenConfig
@Import(DataSourceConfig.class) // <-- AppConfig imports DataSourceConfig
public class AppConfig extends ConfigurationSupport {
    @Autowired DataSourceConfig dataSourceConfig;

    @Bean
    public void TransferService transferService() {
        return new TransferServiceImpl(dataSourceConfig.dataSource());
    }
}
```

The bootstrapping of this application is simplified, as it only needs to supply `AppConfig` when instantiating a `JavaConfigApplicationContext`.

```
public class Main {
    public static void main(String[] args) {
        JavaConfigApplicationContext ctx =
            new JavaConfigApplicationContext(AppConfig.class); // specifies single class
        // ...
    }
}
```

Multiple configurations may be imported by supplying an array of classes to the `@Import` annotation

```
@Configuration
@Import({ DataSourceConfig.class, TransactionConfig.class })
public class AppConfig extends ConfigurationSupport {
    // @Bean methods here can reference @Bean methods in DataSourceConfig or TransactionConfi
}
```