```
package com.bjdbc.dao;
@Repository
class AccountDao {
   private final String SQL_GET_ALL_ACCOUNTS =
"select account_no, account_holder_nm, account_type, ifsc_code,
balance from account";

   @Autowire
   private JdbcTemplate jdbcTemplate;

   public List<AccountBo> findAll() {
      return jdbcTemplate.query(SQL_GET_ALL_ACCOUNTS, (rs,
rowNum)-> {
         AccountBo bo = new AccountBo();
         bo.setAccountNo(rs.getInt(1));
         bo.setAccountHolderName(rs.getString(2));
         // map all the columns into attributes
         return bo;
      });
   }
}
```

```
package com.bjdbc.service;
@Service
class AccountService {
   @Autowire
   private AccountDao accountDao;

   @Transactional(readOnly = true)
   public List<AccountDto> getAllAccounts() {
      return accountDao.findAll().map(bo ->
{}).collect(Collectors.toList());
   }
}
```

DriverManagerDataSourceAutoConfiguration

spring-boot-starter-jdbc
|-jdbc
|-other module dependencies
|-third-party libraries
|-auto-configurations

opinionated view: h2 database dependency is found under classpath of the application if available: then configures DriverManagerDataSource as bean definition with default values

mysql database driver: classpath

application.properties|yml
---
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/hibdb
spring.datasource.username=root
spring.datasource.password=welcome1

[FRAMEWORK CONFIGURATION]

```
db.properties
-------------
db.driverClassName=com.mysql.cj.jdbc.Driver
db.url=jdbc:mysql://localhost:3306/hibdb
db.username=root
db.password=welcome1
```

```
@Configuration
@ComponentScan(basePackages = "com.bjdbc.dao")
@PropertySource("classpath:db.properties")

class PersistenceConfig {

   @Bean
   public DataSource dataSource(@Value("${db.driverClassName}") String driverClassName,
                                @Value("${db.url}") String url,
                                @Value("${db.username}") String username,
                                @Value("${db.password}") String password) {
      DriverManagerDataSource dataSource = new DriverManagerDataSource(url, username, password);
      dataSource.setDriverClassName(driverClassName);
      return dataSource;
   }

   @Bean
   public JdbcTemplate jdbcTemplate(DataSource dataSource) {
      return new JdbcTemplate(dataSource);
   }

   @Bean
   public PlatformTransactionManager transactionManager(DataSource dataSource) {
      return new DataSourceTransactionManager(dataSource);
   }
}
```

AutoConfiguration components takes care of configuring the framework components as bean definitions
AutoConfigurations are nothing but Java Configuration classes, that are pre-written with logic for configuring framework classes as beans
By default AutoConfiguration classes configures Framework components with default values. Incase if we want to customize them, we need to supply our own values asking them to configure

Supply our own values as input to the auto-configuration classes by placing them into the env object of the ioc container.
@PropertySource = Instead SpringApplication class takes care of loading the application.properties|yml in to the env object

JdbcTemplateAutoConfiguration
opinionated view:
1. jdbc module is in classpath
2. DataSource bean definition in ioc container
then configure JdbcTemplate as bean definition

PlatformTransactionManagerAutoConfiguration
|-tx module (classpath)
|-dataSource
DataSourceTransactionManager should be as bean definition