```
                         Date : 25/01/2021
                         Spring Boot 9AM
                           Mr. RAGHU
        ---------------------------------------------
           Spring Boot : Data JPA (JpaRepository)

=> JpaRepository internally extends PaginingAndSortingRpository.
 ie call Operations in Crud and PageAndSort also available in
 JpaRepository.

=> Jpa concepts: Date and Time, Lobs(BLOB, CLOB), Collection Mapping,
                   ASSOCIATION MAPPING, JOINS, COMPONENT MAPPING, .

=> JpaRepository we can define custom queries
  a) findBy
  b) @Query

=> JpaBased operations flow: findAll(), it will not return Iterable.
   returns List<T>.

=> Association Mapping (1...1/1...*/*...1/*...*) with
   Joins for data fetch.

=> Standard Projections for data fetch using custom queries.
=> Procedure calls using Data JPA.
------------------------------------------------------------------
                         Working with Date and Time

=> Annotation : @Temporal needs input from one Enum  : TemporalType
  which has 3 possible values.
   DATE, TIME, TIMESTAMP.

=> Use variable java.util.Date(C) that internally maps with
   different types based on enum selection.

        DATE  -- ex: 10/01/2021
        TIME  -- 9:00:00
        TIMESTAMP -- ex: 10/01/2021 9:00:00


=> ** If we did not specify any Format in code, default storage is
     TIMESTAMP.

--Example--
Name: SpringBoot2DataJpaRepoJpaEx
Dep : Data Jpa, lombok, MySQL

1. Model
package in.nareshit.raghu.model;

import java.util.Date;

import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;
```

```java
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
@AllArgsConstructor
@Entity
public class Product {
        @Id
        private Integer pid;
        private String pcode;
        private Double pcost;

        @Temporal(TemporalType.DATE)
        private Date dteA;
        @Temporal(TemporalType.TIME)
        private Date dteB;
        @Temporal(TemporalType.TIMESTAMP)
        private Date dteC;
}
```

2. Repo

```java
package in.nareshit.raghu.repo;

import org.springframework.data.jpa.repository.JpaRepository;

import in.nareshit.raghu.model.Product;

public interface ProductRepository
        extends JpaRepository<Product, Integer> {

}
```

3. Runner class

```java
package in.nareshit.raghu.runner;

import java.util.Date;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;

import in.nareshit.raghu.model.Product;
import in.nareshit.raghu.repo.ProductRepository;

@Component
public class TestJpaRepoRunner implements CommandLineRunner {

        @Autowired
        private ProductRepository repo;

        @Override
        public void run(String... args) throws Exception {
                repo.save(new Product(
                                100, "ABC", 500.0,
                                new Date(), new Date(), new Date()
                                )
                                );
        }

}
--Formatting java.util.Date ---
package in.nareshit.raghu;

import java.text.SimpleDateFormat;
import java.util.Date;

public class Test {

        //https://docs.oracle.com/javase/7/docs/api/java/text/SimpleDa
teFormat.html
        public static void main(String[] args) {
                Date dte = new Date();
                SimpleDateFormat sdf= new SimpleDateFormat("MMM
dd,YYYY hh:mm:ss SSS");
                String pattern = sdf.format(dte);
                System.out.println(pattern);
        }
}
----------------------------------------------------------
*) In JDBC API java.sql (Date, Time, Timestamp).

   ----------------------------------------------------------
                LOB (Large OBjects) - @Lob

=> BLOB - byte[] + @Lob
   [Images, Audio, Videos, ..etc]
=> CLOB - char[] + @Lob (JPA Annotation)
   [Large Text Data...]


--Model class--
package in.nareshit.raghu.model;

import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Lob;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
@AllArgsConstructor
@Entity
public class Product {
        @Id
        private Integer pid;
        private String pcode;
        private Double pcost;
        @Lob
        private byte[] img;
        @Lob
        private char[] data;


}
```

2. Repo

```
package in.nareshit.raghu.repo;

import org.springframework.data.jpa.repository.JpaRepository;

import in.nareshit.raghu.model.Product;

public interface ProductRepository
        extends JpaRepository<Product, Integer> {

}
```

3. Runner class

```
package in.nareshit.raghu.runner;

import java.io.FileInputStream;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;

import in.nareshit.raghu.model.Product;
import in.nareshit.raghu.repo.ProductRepository;

@Component
public class TestJpaRepoRunner implements CommandLineRunner {

        @Autowired
        private ProductRepository repo;

        @Override
        public void run(String... args) throws Exception {
                FileInputStream fis = new
FileInputStream("F:/Images/SpringBoot6PM_03112020.png");
                byte[] img = new byte[fis.available()];
                fis.read(img);

                String dataStr = "HEllo abcdefgh!HEllo abcdefgh!HEllo
abcdefgh!HEllo abcdefgh!HEllo abcdefgh!HEllo abcdefgh!HEllo
abcdefgh!HEllo abcdefgh!HEllo abcdefgh!HEllo abcdefgh!HEllo
abcdefgh!HEllo abcdefgh!";
                char[] data=dataStr.toCharArray();

                repo.save(new Product(
                                100, "ABC", 500.0,
                                //new Date(), new Date(), new Date()
                                img,data
                                )
                                );
                fis.close();
        }

}
```

Collections Mapping

Data Jpa Supports Storing Collections data as DB tables.
In Model class, all primitives are stored in one table (Parent Table).
For Every Collection variable one child table is created.

Suported Types : List, Set and Map.