JPA and Hibernate in 10 Steps with Spring Boot and H2

In this section, we learn the basics of JPA and understand how it compares with the earlier attempts of persistence frameworks used to store data to database.

- How does it compare to JDBC?
- How does it compare to Spring JDBC?

We will understand the basic concepts of JPA

- Entities
- Mappings
- Relationships

We will create a Spring Boot project with Spring Initializr and create an Entity and Repository to get a big picture of JPA, Spring Data and Spring Data JPA.

JPA and Hibernate in 10 Steps with H2

Steps

- Step 1 : Object Relational Impedence Mismatch - Understanding the problem that JPA solves
- Step 2 : World before JPA - JDBC, Spring JDBC and myBatis
- Step 3 : Introduction to JPA
- Step 4 : Creating a JPA Project using Spring Initializr
- Step 5 : Defining a JPA Entity - User
- Step 6 : Defining a Service to manage the Entity - UserService and EntityManager
- Step 7 : Using a Command Line Runner to save the User to Database
- Step 8 : Magic of Spring Boot and In Memory Database H2
- Step 9 : Introduction to Spring Data JPA
- Step 10 : More JPA Repository : findById and findAll

Step 1 : Object Relational Impedence Mismatch - Understanding the problem that JPA solves

> *What problem does JPA solve?*

Java is an object oriented programming language & Relational databases are used to store data.The way we design objects is different from the way the relational databases are designed.

- Mismatch in Design - Object Oriented vs Normal Forms
- Mismatches in naming - Java Fields vs Table Columns
- Relationships between objects are expressed in a different way compared with relationship between tables.

> *Must Read - http://www.springboottutorial.com/introduction-to-jpa-with-spring-boot-data-jpa*

Step 2 : World before JPA - JDBC, Spring JDBC and myBatis

Before the emergence of JPA and Hibernate, we relied on JDBC, Spring JDBC and myBatis to interact with Relational Databases.

- JDBC, Spring JDBC and myBatis involve writing queries.
- In big application, queries can become complex. Especially when we retrieve data from multiple tables.
- This creates a problem whenever there are changes in the structure of the database.

JDBC

Lot of code

```
Connection connection = datasource.getConnection();

PreparedStatement st = connection.prepareStatement(
                "Update todo set user=?, desc=?, target_date=?, is_done=? where id=?");

st.setString(1, todo.getUser());
st.setString(2, todo.getDesc());
st.setTimestamp(3, new Timestamp(
                todo.getTargetDate().getTime()));
st.setBoolean(4, todo.isDone());
st.setInt(5, todo.getId());
```

```
st.execute();

st.close();

connection.close();
```

Spring JDBC

Simpler than JDBC but still queries

```
jdbcTemplate
.update("Update todo set user=?, desc=?, target_date=?, is_done=? where id=?",
        todo.getUser(),
        todo.getDesc(),
        new Timestamp(todo.getTargetDate().getTime()),
        todo.isDone(),
        todo.getId());
```

myBatis

MyBatis removes the need for manually writing code to set parameters and retrieve results. It provides simple XML or Annotation based configuration to map Java POJOs to database.

```
@Override
@Update("Update todo set user=#{user}, desc=#{desc}, target_date=#{targetDate}, is_done=#{isDone} where id=#{ic
public void updateTodo(Todo todo) throws SQLException;
```

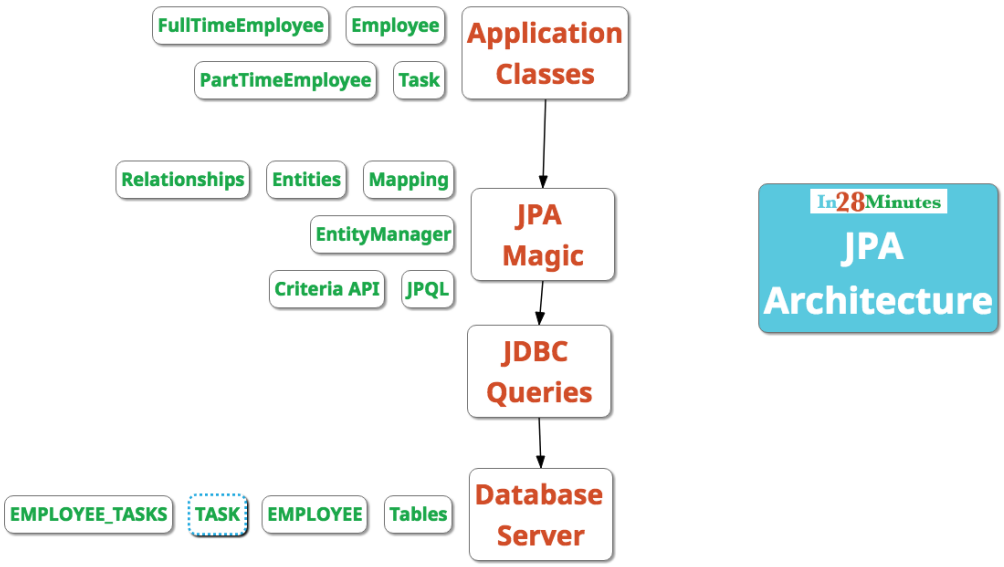
Step 03 : Introduction to JPA

Hibernate (and ORM - Object Relational Mapping) emerged as a result of this big idea:

> *Instead of writing queries by hand, what if we can map tables(relations) to java objects and generate queries*

JPA came in later when Java EE attempted to standize ORM frameworks.

Hibernate went on to become the preferred implementation for JPA (in addition to offering excellent additional features)

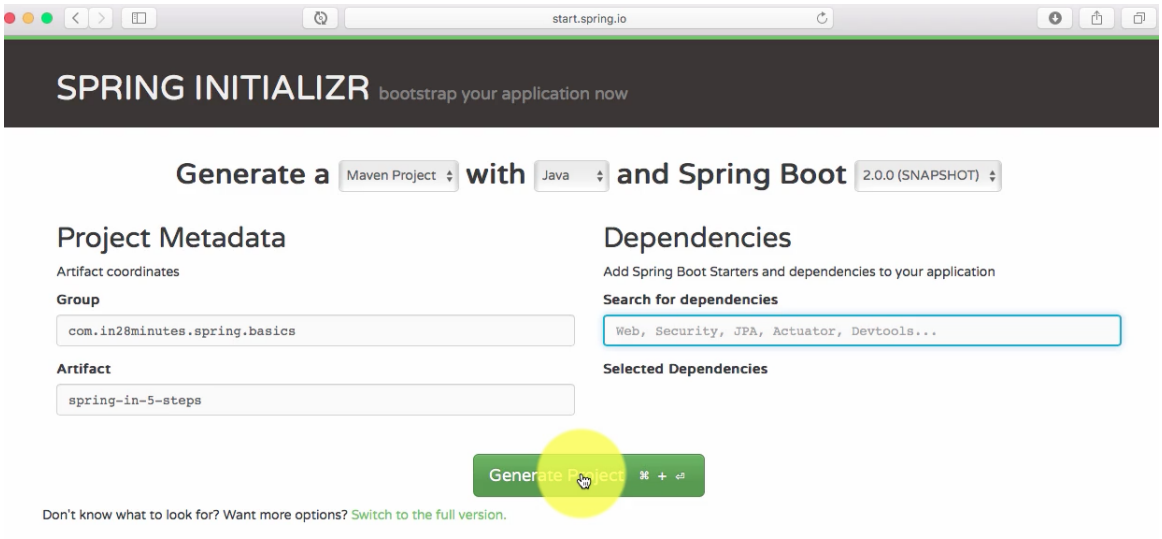


Step 4 : Creating a JPA Project using Spring Initializr

Creating a Spring Project with Spring Initializr is a cake walk.

> *Spring Initializr http://start.spring.io/ is great tool to bootstrap your Spring Boot projects.*

TODO - Insert image



As shown in the image above, following steps have to be done

- Launch Spring Initializr and choose the following
  - Choose `com.in28minutes.learning.jpa` as Group
  - Choose `jpa-in-10-steps` as Artifact
  - Choose Following Dependencies
    - Web
    - JPA
    - H2
- Click Generate Project.
- Import the project into Eclipse.
- If you want to understand all the files that are part of this project, you can go here.

JPA vs Hibernate

Hibernate is one of the most popular ORM frameworks.

JPA defines the specification. It is an API.

- How do you define entities?
- How do you map attributes?
- How do you map relationships between entities?
- Who manages the entities?
- Hibernate is one of the popular implementations of JPA.

Hibernate understands the mappings that we add between objects and tables. It ensures that data is stored/retrieved from the database based on the mappings.

Hibernate also provides additional features on top of JPA. But depending on them would mean a lock in to Hibernate. You cannot move to other JPA implementations like Toplink.

Step 5 : Defining a JPA Entity - User

Let's define a User Entity with a primary key.

```
package com.in28minutes.learning.jpa.jpain10steps.entity;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;

//Table - User
@Entity
public class User {

	@Id
	@GeneratedValue
	private long id;

	private String name;

	private String role;

	protected User() {

	}

	public User(String name, String role) {
		super();
		this.name = name;
		this.role = role;
	}

	public long getId() {
```

```
		return id;
	}

	public String getName() {
		return name;
	}

	public String getRole() {
		return role;
	}

	@Override
	public String toString() {
		return String.format("User [id=%s, name=%s, role=%s]", id, name, role);
	}
}
```

Step 6 : Defining a Service to manage the Entity - UserService and EntityManager

Lets define a repository to manage the User entity.

```
package com.in28minutes.learning.jpa.jpain10steps.service;

import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.transaction.Transactional;

import org.springframework.stereotype.Repository;

import com.in28minutes.learning.jpa.jpain10steps.entity.User;

@Repository
@Transactional
public class UserDAOService {

	@PersistenceContext
	private EntityManager entityManager;

	public long insert(User user){
		entityManager.persist(user);
		return user.getId();
	}
}
```

Important things to note

- @Repository: Spring Annotation to indicate that this component handles storing data to a data store.
- @Transactional: Spring annotation used to simplify transaction management
- @PersistenceContext: A persistence context handles a set of entities which hold data to be persisted in some persistence store (e.g. a database). In particular, the context is aware of the different states an entity can have (e.g. managed, detached) in relation to both the context and the underlying persistence store.
- EntityManager : Interface used to interact with the persistence context.
- entityManager.persist(user): Make user entity instance managed and persistent i.e. saved to database.

Notes from http://docs.oracle.com/javaee/6/api/javax/persistence/EntityManager.html#createNamedQuery(java.lang.String)

> *An EntityManager instance is associated with a persistence context. A persistence context is a set of entity instances in which for any persistent entity identity there is a unique entity instance. Within the persistence context, the entity instances and their lifecycle are managed. The EntityManager API is used to create and remove persistent entity instances, to find entities by their primary key, and to query over entities.*

> *The set of entities that can be managed by a given EntityManager instance is defined by a persistence unit. A persistence unit defines the set of all classes that are related or grouped by the application, and which must be colocated in their mapping to a single database.*

Step 7 : Using a Command Line Runner to save the User to Database

CommandLineRunner interface is used to indicate that this bean has to be run as soon as the Spring application context is initialized.

```
package com.in28minutes.learning.jpa.jpain10steps;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;

import com.in28minutes.learning.jpa.jpain10steps.entity.User;
import com.in28minutes.learning.jpa.jpain10steps.service.UserDAOService;

@Component
public class UserDaoServiceCommandLineRunner implements CommandLineRunner{

	private static final Logger log =
			LoggerFactory.getLogger(UserDaoServiceCommandLineRunner.class);

	@Autowired
	private UserDAOService userDaoService;

	@Override
	public void run(String... arg0) throws Exception {
```

```
                User user = new User("Jack", "Admin");
                //New User is created : User [id=1, name=Jack, role=Admin]
                long insert = userDaoService.insert(user);
                log.info("New User is created : " + user);
        }
}
```

Step 8 : Magic of Spring Boot and In Memory Database H2

The simple answer is all the magic is happening because of Spring Boot and Starter Projects.

Recommended Reading

- Spring Boot Auto Configuration - http://www.springboottutorial.com/spring-boot-auto-configuration
- Spring Boot Starter Project - http://www.springboottutorial.com/spring-boot-starter-projects
- Spring Boot Starter Parent - http://www.springboottutorial.com/spring-boot-starter-parent

H2 Console - We will enable h2 console in /src/main/resources/application.properties

```
spring.h2.console.enabled=true
```

FAQ

- Where is the database created?
    - In Memory - Using H2
- What schema is used to create the tables?
    - Created based on the entities defined
- Where are the tables created?
    - Created based on the entities defined
    - In Memory - Using H2
- Can I see the data in the database?
    - http://localhost:8080/h2-console
    - Use db url jdbc:h2:mem:testdb
- Where is Hibernate coming in from?
    - Through Spring Data JPA Starter
- How is a datasource created?
    - Through Spring Boot Auto Configuration

Magic of Spring Boot and in Memory Database

- Zero project setup or infrastructure
- Zero Configuration
- Zero Maintainance
- Easy to use for Learning and Unit Tests
- Simple Configuration to switch to a real database

Step 9 : Introduction to Spring Data JPA

Spring Data aims to provide a consistent model for accessing data from different kinds of data stores.

UserService (which we created earlier) contains a lot of redundant code which can be easily generalized. Spring Data aims to simplify the code below.

```
package com.in28minutes.learning.jpa.jpain10steps.service;

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

import com.in28minutes.learning.jpa.jpain10steps.entity.User;

public interface UserRepository extends JpaRepository<User, Long>{

}
```

```
package com.in28minutes.learning.jpa.jpain10steps;

import java.util.List;
import java.util.Optional;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;

import com.in28minutes.learning.jpa.jpain10steps.entity.User;
import com.in28minutes.learning.jpa.jpain10steps.service.UserRepository;

@Component
public class UserRepositoryCommandLineRunner implements CommandLineRunner{

        private static final Logger log =
                        LoggerFactory.getLogger(UserRepositoryCommandLineRunner.class);
```

```
	@Autowired
	private UserRepository userRepository;

	@Override
	public void run(String... arg0) throws Exception {
		User user = new User("Jill", "Admin");
		userRepository.save(user);
		log.info("New User is created : " + user);

	}

}
```

Step 10 : More JPA Repository : findById and findAll

```
package com.in28minutes.learning.jpa.jpain10steps;

import java.util.List;
import java.util.Optional;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;

import com.in28minutes.learning.jpa.jpain10steps.entity.User;
import com.in28minutes.learning.jpa.jpain10steps.service.UserRepository;

@Component
public class UserRepositoryCommandLineRunner implements CommandLineRunner{

	private static final Logger log =
			LoggerFactory.getLogger(UserRepositoryCommandLineRunner.class);

	@Autowired
	private UserRepository userRepository;

	@Override
	public void run(String... arg0) throws Exception {
		User user = new User("Jill", "Admin");
		userRepository.save(user);
		log.info("New User is created : " + user);

		Optional<User> userWithIdOne = userRepository.findById(1L);
		log.info("User is retrived : " + userWithIdOne);

		List<User> users = userRepository.findAll();
		log.info("All Users : " + users);
	}

}
```

log

```
HibernateJpaAutoConfiguration matched:
 - @ConditionalOnClass found required classes 'org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBe
 - HibernateEntityManager found class 'org.hibernate.ejb.HibernateEntityManager' (HibernateJpaAutoConfiguratior

DataSourceAutoConfiguration matched:
 - @ConditionalOnClass found required classes 'javax.sql.DataSource', 'org.springframework.jdbc.datasource.embe

JpaBaseConfiguration#entityManagerFactory matched:
 - @ConditionalOnMissingBean (types: org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean,javax.p

JpaBaseConfiguration#transactionManager matched:
 - @ConditionalOnMissingBean (types: org.springframework.transaction.PlatformTransactionManager; SearchStrategy
```

Complete Code

/pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
	<modelVersion>4.0.0</modelVersion>

	<groupId>com.in28minutes.learning.jpa</groupId>
	<artifactId>jpa-in-10-steps</artifactId>
	<version>0.0.1-SNAPSHOT</version>
	<packaging>jar</packaging>

	<name>jpa-in-10-steps</name>
	<description>Demo project for Spring Boot</description>

	<parent>
```

```
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-parent</artifactId>
		<version>2.0.0.RELEASE</version>
		<relativePath /> <!-- lookup parent from repository -->
	</parent>

	<properties>
		<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
		<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
		<java.version>1.8</java.version>
	</properties>

	<dependencies>

		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-data-jpa</artifactId>
		</dependency>

		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-web</artifactId>
		</dependency>

		<dependency>
			<groupId>com.h2database</groupId>
			<artifactId>h2</artifactId>
			<scope>runtime</scope>
		</dependency>

		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-test</artifactId>
			<scope>test</scope>
		</dependency>
	</dependencies>

	<build>
		<plugins>
			<plugin>
				<groupId>org.springframework.boot</groupId>
				<artifactId>spring-boot-maven-plugin</artifactId>
			</plugin>
		</plugins>
	</build>

	<repositories>
		<repository>
			<id>spring-snapshots</id>
			<name>Spring Snapshots</name>
			<url>https://repo.spring.io/snapshot</url>
			<snapshots>
				<enabled>true</enabled>
			</snapshots>
		</repository>
		<repository>
			<id>spring-milestones</id>
			<name>Spring Milestones</name>
			<url>https://repo.spring.io/milestone</url>
			<snapshots>
				<enabled>false</enabled>
			</snapshots>
		</repository>
	</repositories>

	<pluginRepositories>
		<pluginRepository>
			<id>spring-snapshots</id>
			<name>Spring Snapshots</name>
			<url>https://repo.spring.io/snapshot</url>
			<snapshots>
				<enabled>true</enabled>
			</snapshots>
		</pluginRepository>
		<pluginRepository>
			<id>spring-milestones</id>
			<name>Spring Milestones</name>
			<url>https://repo.spring.io/milestone</url>
			<snapshots>
				<enabled>false</enabled>
			</snapshots>
		</pluginRepository>
	</pluginRepositories>

</project>
```

/src/main/java/com/in28minutes/learning/jpa/jpain10steps/entity/User.java

```
package com.in28minutes.learning.jpa.jpain10steps.entity;

import javax.persistence.Entity;
```

```
import javax.persistence.GeneratedValue;
import javax.persistence.Id;

//Table - User
@Entity
public class User {

	@Id
	@GeneratedValue
	private long id;

	private String name;

	private String role;

	protected User() {

	}

	public User(String name, String role) {
		super();
		this.name = name;
		this.role = role;
	}

	public long getId() {
		return id;
	}

	public String getName() {
		return name;
	}

	public String getRole() {
		return role;
	}

	@Override
	public String toString() {
		return String.format("User [id=%s, name=%s, role=%s]", id, name, role);
	}
}
```

/src/main/java/com/in28minutes/learning/jpa/jpain10steps/JpaIn10StepsApplication.java

```
package com.in28minutes.learning.jpa.jpain10steps;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class JpaIn10StepsApplication {

	public static void main(String[] args) {
		SpringApplication.run(JpaIn10StepsApplication.class, args);
	}
}
```

/src/main/java/com/in28minutes/learning/jpa/jpain10steps/service/UserDAOService.java

```
package com.in28minutes.learning.jpa.jpain10steps.service;

import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.transaction.Transactional;

import org.springframework.stereotype.Repository;

import com.in28minutes.learning.jpa.jpain10steps.entity.User;

@Repository
@Transactional
public class UserDAOService {

	@PersistenceContext
	private EntityManager entityManager;

	public long insert(User user){
		entityManager.persist(user);
		return user.getId();
	}
}

/*
 * Spring Data JPA
 *
 *
```

```
*
*/
```

/src/main/java/com/in28minutes/learning/jpa/jpain10steps/service/UserRepository.java

```
package com.in28minutes.learning.jpa.jpain10steps.service;

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

import com.in28minutes.learning.jpa.jpain10steps.entity.User;

public interface UserRepository extends JpaRepository<User, Long>{

}
```

/src/main/java/com/in28minutes/learning/jpa/jpain10steps/UserDaoServiceCommandLine

```
package com.in28minutes.learning.jpa.jpain10steps;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;

import com.in28minutes.learning.jpa.jpain10steps.entity.User;
import com.in28minutes.learning.jpa.jpain10steps.service.UserDAOService;

@Component
public class UserDaoServiceCommandLineRunner implements CommandLineRunner{

	private static final Logger log =
			LoggerFactory.getLogger(UserDaoServiceCommandLineRunner.class);

	@Autowired
	private UserDAOService userDaoService;

	@Override
	public void run(String... arg0) throws Exception {
		User user = new User("Jack", "Admin");
		//New User is created : User [id=1, name=Jack, role=Admin]
		long insert = userDaoService.insert(user);
		log.info("New User is created : " + user);
	}
}
```

/src/main/java/com/in28minutes/learning/jpa/jpain10steps/UserRepositoryCommandLine

```
package com.in28minutes.learning.jpa.jpain10steps;

import java.util.List;
import java.util.Optional;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;

import com.in28minutes.learning.jpa.jpain10steps.entity.User;
import com.in28minutes.learning.jpa.jpain10steps.service.UserRepository;

@Component
public class UserRepositoryCommandLineRunner implements CommandLineRunner{

	private static final Logger log =
			LoggerFactory.getLogger(UserRepositoryCommandLineRunner.class);

	@Autowired
	private UserRepository userRepository;

	@Override
	public void run(String... arg0) throws Exception {
		User user = new User("Jill", "Admin");
		userRepository.save(user);
		log.info("New User is created : " + user);

		Optional<User> userWithIdOne = userRepository.findById(1L);
		log.info("User is retrived : " + userWithIdOne);

		List<User> users = userRepository.findAll();
		log.info("All Users : " + users);
```

```
        }
}
```

/src/main/resources/application.properties

```
spring.jpa.show-sql=true
spring.h2.console.enabled=true
logging.level.org.springframework=debug
```

/src/test/java/com/in28minutes/learning/jpa/jpain10steps/JpaIn10StepsApplicationTests.j

```
package com.in28minutes.learning.jpa.jpain10steps;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit4.SpringRunner;

@RunWith(SpringRunner.class)
@SpringBootTest
public class JpaIn10StepsApplicationTests {

        @Test
        public void contextLoads() {
        }

}
```