**1)What is stream api in java ?**

A Stream in Java can be defined as a sequence of elements from a source. The source of elements here refers to a Collections or Arrays that provides data to the Stream.

• Java streams are designed in such a way that most of the stream operations (called intermediate operations) return a Stream. This helps to create a chain of stream operations. This is called stream pipe-lining.

• Java streams also support the aggregate or terminal operations on the elements. The aggregate operations are operations that allow us to express common manipulations on stream elements quickly and clearly, for example, finding the max or min element, finding the first element matching giving criteria, and so on.

• Not that a stream maintains the same ordering of the elements as the ordering in the stream source.

Example:

Stream<Integer> stream = Stream.of(1,2,3,4,5,6,7,8,9);

stream.forEach(p -> System.out.println(p));

**2)How do you create a stream ?**

Creating Streams

The given below ways are the most popular different ways to build streams from collections.

2.1. Stream.of()

In the given example, we are creating a stream of a fixed number of integers.

Stream<Integer> stream = Stream.of(1,2,3,4,5,6,7,8,9);

stream.forEach(p -> System.out.println(p));

2.2. Stream.of(array)

In the given example, we are creating a stream from the array. The elements in the stream are taken from the array.

Stream<Integer> stream = Stream.of( new Integer[]{1,2,3,4,5,6,7,8,9} );

stream.forEach(p -> System.out.println(p));

2.3. List.stream()

In the given example, we are creating a stream from the List. The elements in the stream are taken from the List.

List<Integer> list = new ArrayList<Integer>();

for(int i = 1; i< 10; i++){

list.add(i);

}

Stream<Integer> stream = list.stream();

stream.forEach(p -> System.out.println(p));

**3)Explain about Intermediate and Terminal operations in java ?**

Stream provides various operations that can be chained together to produce results. Stream operations can be classified into two types.

* Intermediate Operations
* Terminal Operations

1. Intermediate Operations

Intermediate operations return a stream as the output, and intermediate operations are not executed until a terminal operation is invoked on the stream. This is called lazy evaluation, and it is discussed in detail in the later section (Lazy Evaluation).

**filter()**

The filter() method returns a stream with the stream's elements that match the given predicate. Predicate is a functional interface in Java that accepts a single input and can return a boolean value.

**Example**

public class Main {

public static void main(String[] args) {

final List<Integer> list = new ArrayList<>(Arrays.asList(1, 2, 3, 4, 5));

final List<Integer> ans = list.stream()

.filter(value -> value % 2 == 0)

.collect(Collectors.toList());

System.out.println(Arrays.toString(ans.toArray()));

}

}

**Output:**

[2, 4]

**Explanation** This example filters the even values based on the predicate (value -> value % 2 == 0) passed to it.

**map()**

The map() method returns a stream with the resultant elements after applying the given function on the stream elements.

**Example**

public class Main {

public static void main(String[] args) {

final List<Integer> list = new ArrayList<>(Arrays.asList(1, 2, 3, 4, 5));

final List<Integer> ans = list.stream()

.map(value -> value \* 10)

.collect(Collectors.toList());

System.out.println(Arrays.toString(ans.toArray()));

}

}

**Output**

[10, 20, 30, 40, 50]

**Explanation**

In the example the map() method is called with the function value -> value \* 10 on the stream. The function is called for all values of the stream, and hence the result contains all stream values multiplied by 10.

2. Terminal Operations

Terminal operations produce the results of the stream after all the intermediate operations are applied, and we can no longer use the stream once the terminal operation is performed. **forEach()**

The forEach() method iterates and performs the specified action for each stream element. **For parallel stream, it doesn't guarantee to maintain the order of the stream**.

**Example**

public class Main {

public static void main(String[] args) {

final List<Integer> list = new ArrayList<>(Arrays.asList(1, 2, 3, 4, 5));

list.stream().forEach(System.out::println);

}

}

**Output**

1

2

3

4

5

**Explanation**

The forEach() method iterates and prints all the stream values.

**4)what is functional interface ?**

A functional interface is an interface that contains only a single abstract method (a method that doesn’t have a body). The main reason why we need functional interfaces is that we can use them in lambda expressions and method references. This way, we reduce boilerplate code.

You don’t have to include the abstract keyword because, by default, a method declared inside a functional interface is abstract. Only one abstract method is allowed, but it accepts multiple default and static methods.

Example:

@FunctionalInterface

interface FunctionalInterface{

public int multiplyNumber(int x, int y);

}

public class FunctionalInterfaceDemo {

public static void main(String[] args) {

FunctionalInterface = (x, y) -> x \* y;

System.out.println("The result is: " functionalInterface.multiplyNumber(2, 3));

}

}

Output:

The result is: 6

**5)How do you declare and implement a functional interface ?**

functional interface contains only one abstract method.it can have any number or static methods along with object class method.Java provides predefined functional interfaces .For examples:Runnable,ActionListenener,Comparable etc.

@FunctionalInterface

interface i1{

void show(String s);

}

public class demo implements i1 {

public void show(s){

System.out.println(s);

}

public static void main(String[]args)

{

demo d=new demo();

d.show("lambda expresssion");

}

}

Here is only one abstract method in interface i1 so this interface is known as functional interface and by default it is a public abstract method in your interface.

**6)Explain about predefined Functional Interface ?**

Java 8 has provided some Predefined (Built-in) Functional Interfaces to make our programming easier. Moreover, Predefined Functional Interfaces include most commonly used methods which are available to a programmer by default. In our day to day programming many times we come across re-occurring functionalities to be developed. In that case, we can utilize the predefined functional interfaces instead of creating our own every time. They will obviously save our development time and minimize chances of mistakes.

Functional interfaces defined in java.util.function package can be categorized into five types-

1. Consumer- Consumes the passed argument and no value is returned.

2. Supplier- Takes no argument and supplies a result.

3. Function- Takes argument and returns a result.

4. Predicate- Takes argument and returns a boolean result (true or false).

5. Operators- Functional interfaces categorized under Operator are specialized Function where the passed argument and result are of the same type.

**7)How is Stream Api related to Collections ?**

Java Collections framework is used for storing and manipulating group of data. It is an in-memory data structure and every element in the collection should be computed before it can be added in the collections.

Stream API is only used for processing group of data. It does not modify the actual collection, they only provide the result as per the pipelined methods.

Example of Collections

public class CollectiosExample {

public static void main(String[] args) {

List<String> laptopList = new ArrayList<>();

laptopList.add("HCL");

laptopList.add("Apple");

laptopList.add("Dell");

Comparator<String> com = (String o1, String o2)->o1.compareTo(o2);

Collections.sort(laptopList,com);

for (String name : laptopList) {

System.out.println(name);

}

}

}

Example of Streams

public class StreamsExample {

public static void main(String[] args) {

List<String> laptopList = new ArrayList<>();

laptopList.add("HCL");

laptopList.add("Apple");

laptopList.add("Dell");

laptopList.stream().sorted().forEach(System.out::println);

}

}

**8)What is Lambda Expressions in java ?**

Here is how we can define lambda expression in Java.

(parameter list) -> lambda body

The new operator (->) used is known as an arrow operator or a lambda operator. The syntax might not be clear at the moment. Let's explore some examples,

Suppose, we have a method like this:

double getPiValue() {

return 3.1415;

}

We can write this method using lambda expression as:

() -> 3.1415

Here, the method does not have any parameters. Hence, the left side of the operator includes an empty parameter. The right side is the lambda body that specifies the action of the lambda expression. In this case, it returns the value 3.1415.

**9)Explain about date and time api in java ?**

Java Dates

Java does not have a built-in Date class, but we can import the java.time package to work with the date and time API. The package includes many date and time classes. For example:

Class Description

LocalDate Represents a date (year, month, day (yyyy-MM-dd))

LocalTime Represents a time (hour, minute, second and nanoseconds (HH-mm-ss-ns))

LocalDateTime Represents both a date and a time (yyyy-MM-dd-HH-mm-ss-ns)

DateTimeFormatter Formatter for displaying and parsing date-time objects

If you don't know what a package is, read our Java Packages Tutorial.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Display Current Date

To display the current date, import the java.time.LocalDate class, and use its now() method:

import java.time.LocalDate; // import the LocalDate class

public class Main {

public static void main(String[] args) {

LocalDate myObj = LocalDate.now(); // Create a date object

System.out.println(myObj); // Display the current date

}

}

The output will be:

2023-03-28

Display Current Time

To display the current time (hour, minute, second, and nanoseconds), import the java.time.LocalTime class, and use its now() method:

Example

import java.time.LocalTime; // import the LocalTime class

public class Main {

public static void main(String[] args) {

LocalTime myObj = LocalTime.now();

System.out.println(myObj);

}

}

The output will be:

13:45:15.646507

**10)Explain about filter and map Operations ?**

**filter()**

The filter() method returns a stream with the stream's elements that match the given predicate. Predicate is a functional interface in Java that accepts a single input and can return a boolean value.

**Example**

public class Main {

public static void main(String[] args) {

final List<Integer> list = new ArrayList<>(Arrays.asList(1, 2, 3, 4, 5));

final List<Integer> ans = list.stream()

.filter(value -> value % 2 == 0)

.collect(Collectors.toList());

System.out.println(Arrays.toString(ans.toArray()));

}

}

**Output:**

[2, 4]

**Explanation** This example filters the even values based on the predicate (value -> value % 2 == 0) passed to it.

**map()**

The map() method returns a stream with the resultant elements after applying the given function on the stream elements.

**Example**

public class Main {

public static void main(String[] args) {

final List<Integer> list = new ArrayList<>(Arrays.asList(1, 2, 3, 4, 5));

final List<Integer> ans = list.stream()

.map(value -> value \* 10)

.collect(Collectors.toList());

System.out.println(Arrays.toString(ans.toArray()));

}

}

**Output**

[10, 20, 30, 40, 50]

**Explanation**

In the example the map() method is called with the function value -> value \* 10 on the stream. The function is called for all values of the stream, and hence the result contains all stream values multiplied by 10.

**1. What is Spring Boot?**

Spring Boot is called a microservice framework that is built on top of the spring framework. This can help developers to focus more on convention rather than configuration.

The main aim of Spring boot is to give you a production-ready application. So, the moment you create a spring-boot project, it is runnable and can be executed/deployed on the server.

It comes with features like autoconfiguration, auto dependency resolution, embedded servers, security, health checks which enhances the productivity of a developer.

**2. What is the difference between Spring and Spring Boot?**

Difference between Spring and Spring boot are as follows:

Spring –

Is a dependency injection framework.

It is basically used to manage the life cycle of java classes (beans). It consists of a lot of boilerplate configuration.

Uses XML based configuration.

It takes time to have a spring application up and running and it’s mainly because of boilerplate code.

Spring boot-

It is a suite of pre- configured frameworks and technologies which helps to remove boilerplate configuration.

Uses annotations.

It is used to create a production-ready code.

**3. What is actuator in spring boot?**

An actuator is one of the best parts of spring boot which consists of production-ready features to help you monitor and manage your application.

With the help of an actuator, you can monitor what is happening inside the running application.

Actuator dependency figures out the metrics and makes them available as a new endpoint in your application and retrieves all required information from the web. You can identify beans, the health status of your application, CPU usage, and many more with the actuator. By using @Endpoint annotation, you can create a custom endpoint.

**4. How to change port in spring boot?**

The default port number to start your SpringBoot application is 8080.

Just to change the port number, you need to add server.port=8084c(your port number) property in your application.properties file and start your application.

**5. What is JPA in spring boot?**

JPA is basically Java Persistence API. It’s a specification that lets you do ORM when you are connecting to a relational database which is Object-Relational Mapping.

So, when you need to connect from your java application to relational database, you need to be able to use something like JDBC and run SQL queries and then you get the results and convert them into Object instances.

ORM lets you map your entity classes in your SQL tables so that when you connect to the database , you don’t need to do query yourself, it’s the framework that handles it for you.

And JPA is a way to use ORM, it’s an API which lets you configure your entity classes and give it to a framework so that the framework does the rest.

**6. How to save image in database using spring boot?**

First configure mysql in your spring boot application.

Then you can map your entities with your db tables using JPA.

And with the help of save() method in JPA you can directly insert your data into your database

@RestController

@RequestMapping("/greatleasrning")

public class Controller {

@Autowired

private final GreatLearningRepository greatLearningRepository;

public Controller(GreatLearningRepository greatLearningRepository) {

}

In above case, your data which may be in JSON format can be inserted successfully into database.

@RequestMapping(method = RequestMethod.POST)

ResponseEntity<?> insert(@RequestBody Course course) {

greatLearningRepository.save(course);

return ResponseEntity.accepted().build();

}

}

**7. What is auto configuration in spring boot?**

AutoConfiguration is a process by which Spring Boot automatically configures all the infrastructural beans. It declares the built-in beans/objects of the spring specific module such as JPA, spring security and so on based on the dependencies present in your applications class path.

For example: If we make use of Spring JDBC, the spring boot autoconfiguration feature automatically registers the DataSource and JDBCTemplete bean.

This entire process of automatically declaring the framework specific bean without the need of writing the xml code or java config code explicity is called Autoconfiguration which is done by springBoot with the help of an annotation called @EnableAutoconfiguration alternatively @SpringBootApplication.

**8. How to fetch data from database in spring boot?**

You can use the following steps to connect your application with MySQL database.

1. First create a database in MySQL with create DATABASE student;

2. Now, create a table inside this DB:

CREATE TABLE student(studentid INT PRIMARY KEY NOT NULL AUTO\_INCREMENT, studentname VARCHAR(255));

3. Create a SpringBoot application and add JDBC, MySQL and web dependencies.

4. In application.properties, you need to configure the database.

spring.datasource.url=jdbc:mysql://localhost:3306/studentDetails

spring.datasource.username=system123

spring.datasource.password=system123

spring.jpa.hibernate.ddl-auto=create-drop

5. In your controller class, you need to handle the requests.

package com.student;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.jdbc.core.JdbcTemplate;

import org.springframework.web.bind.annotation.RestController;

@RestController

public class JdbcController {

@Autowired

JdbcTemplate jdbc;

@RequestMapping("/save")

public String index(){

jdbc.execute("insert into student (name)values(GreatLearnings)");

return "Data Entry Successful";

}

}

6. Run the application and check the entry in your Database.

**9. What is bootstrapping in spring boot?**

One of the way to bootstrap your spring boot application is using Spring Initializer.

you can go to the official website of spring and select your version, and add you groupID, artifactId and all the required dependencies.

And then you can create your restEndpoints and build and run your project.

There you go, you have bootstrapped your spring boot application.

**10. What is dependency injection in spring boot?**

Dependency injection is a way through which the Spring container injects one object into another. This helps for loose coupling of components.

For example: if class student uses functionality of department class, then we say student class has dependency of Department class. Now we need to create object of class Department in your student class so that it can directly use functionalities of department class is called dependency injection.

**11. How to configure hibernate in spring boot?**

The important and required dependency to configure hibernate is:

spring-boot-starter-data-jpa

h2 (you can also use any other database)

Now, provide all the database connection properties in application.properties file of your application in order to connect your JPA code with the database.

Here we will configure H2 database in application.properties file:

spring.datasource.url=jdbc:h2:file:~/test

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=test

spring.datasource.password=test

spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

spring.h2.console.enabled=true

spring.h2.console.path=/h2-console

Adding the above properties in your application.properties file will help you to interact with your database using JPA repository interface.

**12. Mention the advantages of Spring Boot.**

Advantages of Spring Boot –

It allows convention over configuration hence you can fully avoid XML configuration.

SpringBoot reduces lots of development time and helps to increase productivity.

Helps to reduce a lot of boilerplate code in your application.

It comes with embedded HTTP servers like tomcat, Jetty, etc to develop and test your applications.

It also provides CLI (Command Line Interface) tool which helps you to develop and test your application from CMD.

**13. Explain what is thyme leaf and how to use thymeleaf?**

Thymeleaf is a server-side java template engine which helps processing and creating HTML, XML, JavaScript , CSS, and text. Whenever the dependency in pom.xml (in case of maven project) is find, springboot automatically configures Thymeleaf to serve dynamic web content.

Dependency: spring-boot-starter-thymeleaf

We can place the thyme leaf templates which are just the HTML files in src/main/resources/templates/ folder so that spring boot can pick those files and renders whenever required.

Thymeleaf will parse the index.html and will replace the dynamic values with its actual value that is been passed from the controller class.

That’s it, once you run your Spring Boot application and your message will be rendered in web browsers.

**14. Can we change the port of the embedded Tomcat server in Spring boot?**

Yes, you can change the port of embedded Tomcat server in Spring boot by adding the following property in your application.properties file.

server.port=8084

The default port number of the tomcat server to run the spring boot application is 8080, which is further possible to change it.

So we can change the port of tomcat following ways given below:-

Using application.properties

Using application.yml

Using EmbeddedServletContainerCustomizer interface.

Using WebServerFactoryCustomizer interface.

Using Command-Line Parameter.

**15. Mention the steps to connect Spring Boot application to a database using JDBC?**

Below are the steps to connect your Spring Boot application to a database using JDBC:

Before that, you need to add required dependencies that are provided by spring-boot to connect your application with JDBC.

Step 1: First create a database in MySQL with create DATABASE student;

Step 2: Now, create a table inside this DB:

CREATE TABLE student(studentid INT PRIMARY KEY NOT NULL AUTO\_INCREMENT,

studentname VARCHAR(255));

Step 3: Create a springBoot and add JDBC,mysql and web dependencies.

Step 4: In application.properties, you need to configure the database.

spring.datasource.url=jdbc:mysql://localhost:3306/studentDetails

spring.datasource.username=system123

spring.datasource.password=system123

spring.jpa.hibernate.ddl-auto=create-drop

Step 5: In your controller class, you need to handle the requests.

package com.student;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.jdbc.core.JdbcTemplate;

import org.springframework.web.bind.annotation.RestController;

@RestController

public class JdbcController {

@Autowired

JdbcTemplate jdbc;

@RequestMapping("/save")

public String index(){

jdbc.execute("insert into student

(name)values(GreatLearnings)");

return "Data Entry Successful";

}

}

Step 6: Run the application and check the entry in your Database.

Step 7: You can also go ahead and open the URL and you will see “Data Entry Successful” as your output.

**16. What are the @RequestMapping and @RestController annotation in Spring Boot used for?**

The @RequestMapping annotation can be used at class-level or method level in your controller class.

The global request path that needs to be mapped on a controller class can be done by using @RequestMapping at class-level. If you need to map a particular request specifically to some method level.

Below is a simple example to refer to:

@RestController

@RequestMapping("/greatLearning")

public class GreatLearningController {

@RequestMapping("/")

String greatLearning(){

return "Hello from greatLearning ";

}

@RequestMapping("/welcome")

String welcome(){

return "Welcome from GreatLearning";

}

}

The @RestController annotation is used at the class level.

You can use @RestController when you need to use that class as a request handler class.All the requests can be mapped and handled in this class.

@RestController itself consists @Controller and @ResponseBody which helps us to remove the need of annotating every method with @ResponseBody annotation.

Below is a simple example to refer to for use of @RestController annotation:

@RestController

@RequestMapping(“bank-details”)

public class DemoRestController{

@GetMapping(“/{id}”,produces =”application/json”)

public Bank getBankDetails(@PathVariable int id){

return findBankDetailsById();

}

}

Here, @ResponseBody is not required as the class is annotated with @RestController.

**17. What do you understand by auto-configuration in Spring Boot and how to disable the auto-configuration?**

AutoConfiguration is a process by which Spring Boot automatically configures all the infrastructural beans. It declares the built-in beans/objects of the spring-specific module such as JPA, spring-security, and so on based on the dependencies present in your application’s classpath.

For example: If we make use of Spring JDBC, the spring boot autoconfiguration feature automatically registers the DataSource and JDBCTemplete bean.

This entire process of automatically declaring the framework-specific bean without the need of writing the XML code or java-config code explicitly is called Autoconfiguration which is done by spring-boot with the help of an annotation called @EnableAutoconfiguration alternatively @SpringBootApplication.

1. You can exclude the attribute of @EnableAutoConfiguration where you don’t want it to be configured implicity in order to disable the spring boot’s auto-configuration feature.

2. Another way of disabling auto-configuration is by using the property file:

For example:

spring.autoconfigure.exclude=

org.springframework.boot.autoconfigure.mongo.MongoAutoConfiguration,

org.springframework.boot.autoconfigure.data.MongoDataConfiguration,

In the above example, we have disabled the autoconfiguration of MongoDB.

**18. What do you understand by Spring Data REST?**

By using Spring Data Rest, you have access to all the RESTful resources that revolves around Spring Data repositories.

Refer the below example:

@RepositoryRestResource(collectionResourceRel = "greatlearning", path = "sample")

public interface GreatLearningRepo extends CustomerRepository< greatlearning, Long> {

}

Now you can use the POST method in the below manner:

{

“Name”:”GreatLearning”

}

And you will get response as follow:

{

“Name”:”GreatLearning”

}

\_\_\_\_\_\_\_\_\_\_

{

"name": "Hello greatlearning "

"\_links": {

"self": {

"href": "<a href="http://localhost:8080/sample/1">http://localhost:8080/ greatlearning /1</a>"

},

" greatlearning ": {

“href": "<a href="http://localhost:8080/sample/1">http://localhost:8080/ greatlearning /1</a>"

}

}

In the above, you can see the response of the newly created resource.

**19. What do you think is the need for Profiles?**

The application has different stages-such as the development stage, testing stage, production stage and may have different configurations based on the environments.

With the help of spring boot, you can place profile-specific properties in different files such as

application-{profile}.properties

In the above, you can replace the profile with whatever environment you need, for example, if it is a development profile, then application-development.properties file will have development specific configurations in it.

So, in order to have profile-specific configurations/properties, you need to specify an active profile.

**20. How to insert data in mysql using spring boot?**

First configure mysql in your spring boot application.

Then you can map your entities with your db tables using JPA.

And with the help of save() method in JPA, you can directly insert your data into your database.

@RestController

@RequestMapping("/greatleasrning")

public class Controller {

@Autowired

private final GreatLearningRepository greatLearningRepository;

public Controller(GreatLearningRepository greatLearningRepository) {

this. greatLearningRepository = greatLearningRepository;

}

In the above case, your data which may be in JSON format can be inserted successfully into the database.

@RequestMapping(method = RequestMethod.POST)

ResponseEntity<?> insert(@RequestBody Course course) {

greatLearningRepository.save(course);

return ResponseEntity.accepted().build();

}

}

**21. How to use crud repository in spring boot?**

In order to use crud repository in spring boot, all you have to do is extend the crud repository which in turn extends the Repository interface as a result you will not need to implement your own methods.

Create a simple spring boot application which includes below dependency:

spring-boot-starter-data-jpa, spring-boot-starter-data-rest

And extend your repository interface as shown below:

package com.greatlearning;

import java.util.List;

import org.springframework.data.repository.CrudRepository;

import org.springframework.data.rest.core.annotation.RepositoryRestResource;

@RepositoryRestResource

public interface GreatLearning extends CrudRepository<Candidate, Long>

{

public List<Candidate> findById(long id);

//@Query("select s from Candidate s where s.age <= ?")

public List<Candidate> findByAgeLessThanEqual (long age);

}

**22. What are the basic Annotations that spring boot offers?**

First of all, we have to know about the annotations. Annotations are used to instruct the intention of the programmers.

As the name suggests, spring boot annotations is a form of Metadata that provides the whole data about the program. In other ways, we can define it as annotations are used to provide supplemental information about the program. It is not part of the program.

It does not change the programs which are already compiled.

Core Spring Framework Annotation:-

@Required:-

@Required applies to the bean setter method.

This indicates that the annotated bean must be populated at the configuration time with the required property; if the following case is not satisfied, it throws an exception BeanInitializationException.

@Autowired:-

In the spring framework, spring provides annotation-based auto–wiring by providing @Autowired annotation.

It is used to auto-wire spring bean on setter methods, instance variables and constructors., When we use the annotation @Autowired, the spring container auto-wires the bean factory by matching the data type.

Other Annotations which are provided by Spring Boot, Spring Framework, and In Spring MVC are:-

@configuartion.

@Componentscan

@Bean

@component.

@Controller.

@service.

@Repository

@EnableAutoConfiguaration

@SpringBootApplication.

@RequestMapping

@GetMapping

@PostMapping.