C:\WORK\java8.txt - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

imp.txt | git.txt | CI.txt | reactjs.txt | java8.txt | db.txt | db-segmentation.txt | ng6.txt | node.txt | work.txt | VisaFeatureSelect.txt | temp.txt | New Text Document.txt

```
Java Version 8 Feature:
> support for functional programming
> Nashorn new javascript engine
> Calling JavaScript from Java
> new api for date time manipulation
> new streaming api
-------------------------------
jdk
> java development kit.
> compile document & package java program.
> Along with jre it includes interprater/loader, compiler(javac), archiver(jar), document generator (javadoc) and others
jre
> java runtime environment in which java bycode can be executed executed.
> It implements the JVM (Java Virtual Machine) and provides all the class libraries and other support files that
JVM uses at runtime.
So JRE is a software package that contains what is required to run a Java program.
Basically, it's an implementation of the JVM which physically exists.
jvm
>java virtual machine.
>It is an abstract machine. It is a specification that provides run-time environment in which java bytecode can be executed.
>JVM follows three notations: Specification(document that describes the implementation of the Java virtual machine), Implementation
(program that meets the requirements of JVM specification) and Runtime Instance (instance of JVM is created whenever you write a
java command on the command prompt and run class).
-------------------------------
which compiler java use
JIT(just in time) first time compiles the whole program, next time compilation process it just compiles whichever the code got modified.
So it wont compile whole program all the time. This is one reason Java performance is high

Can we use static public void main() ?
Yes
-------------------------------
Data type
primitive data type - byte, short, int, long, float, double, boolean, char ( string in java is not prmitive, its object type)
reference data type - object and other array
>> java is not 100% object oriented bcoz primitive types are not objects.
-------------------------------
Methods in Object class
equals(), hashcode(), wait(), getClass(), toString(), notify(), notifyAll()
-------------------------------
Package: A java package is a group of similar types of classes, interfaces and sub-packages.
Advantage of Java Package
1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.
2) Java package provides access protection.
3) Java package removes naming collision.javac -d Destination_folder file_name.java
compile java package:
javac -d directory javafilename
javac -d . Animal.java    (same dir)
```

length : 81,194 lines : 1,981 Ln : 130 Col : 1 Sel : 0 | 0 Windows (CR LF) UTF-8 INS

TeX file

6:28 PM
10/15/2019

C:\WORK\java8.txt - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

```
javac -d . Animal.java    (same dir)
javac -d animals Animal.java   (animals dir)
-----------------------------
LIST VS SET - List can contain duplicate elements whereas Set contains unique elements only.

Java HashSet Example
import java.util.*;
class TestCollection9{
 public static void main(String args[]){
  //Creating HashSet and adding elements
  HashSet<String> set=new HashSet<String>();
   ///List<String> al= new ArrayList<String>(); // for list
  set.add("Ravi");
  set.add("Vijay");
  set.add("Ravi");
  set.add("Ajay");
  //Traversing elements
  Iterator<String> itr=set.iterator();
  while(itr.hasNext()){
   System.out.println(itr.next());
  }
 }
}

HashMap - unlike arrayList it store items in "key/value" pairs

this -> is a keyword in Java which is used as a reference to the object of the current class


HASHMAP VS HASHLIST
HashSet is a set, e.g. {1,2,3,4,5}
HashMap is a key -> value (key to value) map, e.g. {a -> 1, b -> 2, c -> 2, d -> 1}
-------------------------------------
Optional:: is a container object used to contain not-null objects. Optional object is used to represent null with absent value.

ex and use case:
 public class Java8Tester {

   public static void main(String args[]) {
      Java8Tester java8Tester = new Java8Tester();
      Integer value1 = null;
      Integer value2 = new Integer(10);

      //Optional.ofNullable - allows passed parameter to be null.
      Optional<Integer> a = Optional.ofNullable(value1);

      //Optional.of - throws NullPointerException if passed parameter is null
```

length : 81,194 lines : 1,981 Ln : 130 Col : 1 Sel : 0 | 0 Windows (CR LF) UTF-8

TeX file

```
        Optional<Integer> b = Optional.of(value2);
        System.out.println(java8Tester.sum(a,b));
    }

    public Integer sum(Optional<Integer> a, Optional<Integer> b) {
        //Optional.isPresent - checks the value is present or not

        System.out.println("First parameter is present: " + a.isPresent());
        System.out.println("Second parameter is present: " + b.isPresent());

        //Optional.orElse - returns the value if present otherwise returns
        //the default value passed.
        Integer value1 = a.orElse(new Integer(0));

        //Optional.get - gets the value, value should be present
        Integer value2 = b.get();
        return value1 + value2;
    }
}

---------------------------------------
what is java bean:
JavaBeans are Java classes which adhere to an extremely simple coding convention.
All you have to do is to1. All properties private (use getters/setters)
1. implement java.io.Serializable interface - to save the state of an object
2. use a public empty argument constructor - to instantiate the object
3. provide public getter/setter methods - to get and set the values of private variables (properties ).

@Beans - backbone of spring application, managed by spring IOC container (It uses Dependency Injection (DI) to manage
components and these objects are called Spring Beans.)
@Bean
Spring @Bean Annotation is applied on a method to specify that it returns a bean to be managed by Spring context.
Spring Bean annotation is usually declared in Configuration classes methods.
---------------------------------------
By default, the Java super class java.lang.Object provides two important methods for
comparing objects: equals() and hashcode()

equals(Object obj): a method provided by java.lang.Object that indicates whether some other object passed as
an argument is "equal to" the current instance. The default implementation provided by the JDK is based on
memory location — two objects are equal if and only if they are stored in the same memory address.

hashcode(): a method provided by java.lang.Object that returns an integer representation of the object memory address.
By default, this method returns a random integer that is unique for each instance. This integer might change between
several executions of the application and won't stay the same.
-----------------------------
Serialization is a mechanism of converting the state of an object into a byte stream.
Deserialization is the reverse process where the byte stream is used to recreate the actual
Java object in memory. This mechanism is used to persist the object.
-----------------------------
scanner class - popular way to read input from stdin is by using the Scanner class and specifying the Input Stream as System.in.
Alternatively, you can use the BufferedReader class.
```

```
Scanner scanner = new Scanner(System.in);
String myString = scanner.next();
int myInt = scanner.nextInt();
scanner.close();

System.out.println("myString is: " + myString);
System.out.println("myInt is: " + myInt);
------------------------------
Access Control Modifiers::
Default - Visible to the package. No modifiers are needed.
Private - Visible to the class only.
Public - Visible to the world.
Protected - Visible to the package and all subclasses.

Non-Access Control Modifiers::
transient - non access modifier.  Attributes and methods are skipped when serializing the object containing them
synchronized - non access modifier.    Methods can only be accessed by one thread at a time
volatile- non access modifier.  The value of an attribute is not cached thread-locally, and is always read from the "main memory"

static, final, abstract
------------------------------
process vs thread:
   process is a running instance of application over the OS ( can be multiprocessing). each process have separate memory space.
   thread are within the process, multiple thread can be within the process

wrapper class:
   way where can convert java primitive type to reference type.
   Boxing & Unboxing :
   int i =10;
   Integer bx = new Integer(i);   //boxing.  bx is object now
   int unbx = bx.getValue();    // unboxing
   Integer abx = i;  // autoboxing  similar Integer abx = new Integer(i);
   int l = abx;   //auto unboxing
------------------------------
final, finally, finalize
final - apply on method, class & variable. class can not be inherited, method cannot overrise, variable cannot changed.
finally - used with execption, run whether exception are not.
finalize - called before garbage collector.

System.gc() -> called garbage collector.
------------------------------
String vs StringBuffer vs StringBuilder

String - Immutable, reverse does not exist
StringBuffer - Mutable, Thread safe means synchronized, reverse exist
StringBuilder - Mutable, Not thread safe, performance bcoz no synchronization overhead

String str = new String("hi");
StringBuffer buffer = new StringBuffer("hi");
StringBuilder builder = new StringBuilder("hi");
```

```
System.out.println(str + " hello");     //hi hello
System.out.println(buffer + " hello");  //hi hello
System.out.println(builder + " hello"); // hi hello
-------------------------------
Heap & Stack memory

Heap - anything which is constructed dynamically store in heap
   Point xx = new Point(1,2);  // xx(in stack) is a variable have reference of object(in heap)

Usage-
Stack: store only local primitive variables and reference variable to Object in Heap space
Heap: any object created stored in Heap

Memory -
Stack: - used by one thread of execution,
Heap:can be used by all part of application

Access:
Stack:object stored globally accsible,
Heap: can not access by other threads

Memort Mgmt-
Stack- LIFO manner to free memory.
Heap - based on generation associated to each object

Lifetime-
Stach - exist till executon of thread
Heap - exist till end of application execution
---------------------------------
ArrayList vs Vector::

ArrayList is not-synchronized, Vector is Synchronized (that why slow)
ArrayList does not define the array size, vector does.
In am element inserted in ArrayList, its increased its arry size by 50%, while vector defaults to doubling its size.
ArrayList use only iterator for traversing while Vector (Except HashTable) use both Enumeration and iterator.
-----------------------------------
HashMap vs HashTable::

Hashmap is non-synchronized(that why fast) and not thread safe, while hashtable is synchronized(slow) and therad safe means shared with many thread.
Hashmap can be synchronized calling the Collections.synchronizedmap(hashmap). HashTable can not be unsynchrnized.
HashMap allow one null key and multiple null values, Hashtable does not allow any null keys or values
Hashmap travesed by iterator. Hashtable with both Enumeration and iterator.
Hashmap inherit abstractMap Class, while HashTable inherit DictionaryClass
-----------------------------------
Three different synchronized Map implementations in the Java:
1 - Hashtable - old ( extends obsolete Dictionary)
2 - Collections.synchronizedMap(Map)-> creates a blocking Map which will degrade performance. Use it if you need to ensure data consistency, and each thread needs to have an up-to-date view of the map
3 - ConcurrentHashMap - > It allows concurrent modification of the Map from several threads without the need to block them, Use if performance is critical, and each thread only inserts data to the map, with reads happening
less frequently.
-----------------------------------
```

```
---------------------------------
String comparision::
equals() vs == operator

equals - compare value in the object written in the heap. == compare only the reference

String foo = new String("abc");
String bar = new String("abc");

if(foo==bar)  // false

if(foo.equals(bar)) //true
// s1.equalsIgnoreCase(s3)  ignore case

foo = bar;
if(foo==bar)  // true


3 methods to compare string in java
==  equals() and compareTo()
The String compareTo() method compares values lexicographically and returns an integer value that describes
if first string is less than, equal to or greater than second string.

   String s1="Sachin";
   String s2="Sachin";
   String s3="Ratan";

   System.out.println(s1.compareTo(s2));//0
   System.out.println(s1.compareTo(s3));//1(because s1>s3)
   System.out.println(s3.compareTo(s1));//-1(because s3 < s1 )

---------------------------------
#Abstract class
 1. can have abstract and non abstract method
 2. dont support multiple inheritance
 3. can have static, nonstatic, final, nonfinal variables
 4. abstract keyword to declare class
 5. An abstract class can extend another Java class and implement multiple Java interfaces.
 6. An abstract classcan be extended using keyword "extends".
 7. A Java abstract classcan have class members like private, protected, etc.
 ex. Shape class with abstarct menthod area() which could be diffrent for different shape like sqare, circle, rect..

 #vs

 #interface
 1. only abstract method, Since Java 8, it can have default and static methods also.
 2. support multiple inheritance
 3. have only static and final variable
 4. interface keyword to declare interface
 5. An interface can extend another Java interface only.
 6. An interface classcan be implemented using keyword "implements".
```

```
7. Members of a Java interface are public by default.
-----------------------------------
Java does not support multiple inheritence bcoz if more than one parent class have same method then its hard to
decide by compiler which one to choose.
----------------------
override - run time poly.  done using inheritance and interface.
class Car {
  void run()
   {
     System.out.println("car is running");
   }
}
class Test extends Car {
   void run()
   {
   System.out.println("Audi is running safely with 100km");
   }
   public static void main(String[] args)
   {
      Car b= new Test();    //upcasting
      b.run();
   }
}

overloading - compile time poly..
class Test {
   static int add(int a, int b)
   {
    return a+b;
   }
   static double add( double a, double b)
   {
    return a+b;
   }
   public static void main(String args[])
   {
   System.out.println(Test.add(11,11));
   System.out.println(Test.add(12.3,12.6));
   }
}
----------------------
method hiding:
In case of overriding if the super class method is final or static
then it can not be inherited in child class. if create same method with same signature in child class then
that method would be hide.
so can not override private or static method in java.
class Parent {
   private void display() {
      System.out.println("parent methosd");
   }
}
```

```
class Child  extends Parent {
    public void display() {
        System.out.println("child methosd");
    }
}
public class Hello {
    public static void main(String[] args) {
        Parent obj = new Child();
        obj.display(); // error The method display() from the type Parent is not visible
    }
}
------------------------
Servlet - Java servlet is a server side technologies to extends the capabilities of the web servers by
proving support for dynamic response and persistence.

the javax.servlet and javax.servlet.http package provide interface and classes for writing our own servlets.

All servelets ,ust implements the javax.servlet.Servlet interface which defines servlet lifecycle methods.

ServletConfig vs ServletContext
ServletConfig - for single servlet
ServletContext - represent whole web application

Stages/ lifecycle

1 - Loaded
2 - Instantiated
3 - Initialize
4 - Request
5  -Destruction

init, service, destroy are 3 apis

A Java servlet is a Java software component that extends the capabilities of a server.
Although servlets can respond to any types of requests, they most commonly implement
web containers for hosting web applications on web servers and thus qualify as a server-side servlet web API.
---------------------------
Request Dispacher: interface used to forward the request to another resource that can be HTML, JSP
or another servlet in the same application.

2 method

void forward()
void include()
---------------------------
int Vs Integer
Integer.parseInt("1") vs int.parseInt("1")

int - is a primitive type. Variables of type int store the actual binary value for the integer
Integer - Integer is a class with a single field of type int. This class is used where you need an int to be treated like any other object,
```

Integer.parseInt("1") is a call to the static method parseInt from class Integer (note that this method actually returns an int and not an Integer)

>> Note that every primitive type in Java has an equivalent wrapper class:
Wrapper classes inherit from Object class, and primitive don't.

byte has Byte
short has Short
int has Integer
long has Long
boolean has Boolean
char has Character
float has Float
double has Double
---
static method / block executed when class is loaded into the memory
normal block executed when object created.
---
why hibernate over jdbc?

remove boiler plate code
support inheritance, association, collections
implicitly provide transaction mgmt
caching for better performance
HQL hibernate query language is more object oreineted while in jdbc you have to write the native sql query

---
Collection - in Java is a framework that provides an architecture to store and manipulate the group of objects.
A Collection represents a single unit of objects, i.e., a group.

searching, sorting, insertion, manipulation, and deletion.

Java Collection framework provides many
interfaces (Set, List, Queue, Deque) and
classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet).

Iterator interface provides the facility of iterating the elements in a forward direction only.
3 mrthods
1 public boolean hasNext() It returns true if the iterator has more elements otherwise it returns false.
2 public Object next() It returns the element and moves the cursor pointer to the next element.
3 public void remove() It removes the last elements returned by the iterator. It is less used.

Iterable Interface ->
Collection Interface ->
List Interface
Set Interface
Queue Interface
Dequeue interface

LIST INTERFACE::

```
LIST INTERFACE::
four classes to instantiate
List <data-type> list1= new ArrayList();
List <data-type> list2 = new LinkedList();
List <data-type> list3 = new Vector();
List <data-type> list4 = new Stack();

The ArrayList class implements the List interface, uses a dynamic array to store the duplicate element of different
data types. The ArrayList class maintains the insertion order and is non-synchronized.


LinkedList implements the Collection interface. It uses a doubly linked list internally to store the elements.
It can store the duplicate elements. It maintains the insertion order and is non-synchronized. In LinkedList,
the manipulation is fast because no shifting is required


Vector uses a dynamic array to store the data elements. It is similar to ArrayList. However, It is synchronized
nd contains many methods that are not the part of Collection framework.


The stack is the subclass of Vector. It implements the last-in-first-out data structure, i.e., Stack.
The stack contains all of the methods of Vector class and also provides its methods like boolean push(), boolean peek(), boolean push(object o), which defines its properties.


SET INTERFACE::
Set Interface in Java is present in java.util package. It extends the Collection interface. It represents the unordered set
of elements which doesn't allow us to store the duplicate items. We can store at most one null value in Set.
Set is implemented by HashSet, LinkedHashSet, and TreeSet.
Set<data-type> s1 = new HashSet<data-type>();
Set<data-type> s2 = new LinkedHashSet<data-type>();
Set<data-type> s3 = new TreeSet<data-type>();

HashSet class implements Set Interface. It represents the collection that uses a hash table for storage. Hashing is used to
store the elements in the HashSet. It contains unique items.

LinkedHashSet class represents the LinkedList implementation of Set Interface. It extends the HashSet class and implements
Set interface. Like HashSet, It also contains unique elements. It maintains the insertion order and permits null elements.

Java TreeSet class implements the Set interface that uses a tree for storage. Like HashSet, TreeSet also contains unique
elements. However, the access and retrieval time of TreeSet is quite fast. The elements in TreeSet stored in ascending order.

QUEUE INTERFACE::
Queue interface maintains the first-in-first-out FIFO order. It can be defined as an ordered list that is used to hold the elements which
are about to be processed. There are various classes like PriorityQueue, Deque, and ArrayDeque which implements the Queue interface.
Queue<String> q1 = new PriorityQueue();
Queue<String> q2 = new ArrayDeque();


The PriorityQueue class implements the Queue interface. It holds the elements or objects which are to be processed by their priorities.
PriorityQueue doesn't allow null values to be stored in the queue.
```

```
ArrayDeque class implements the Deque interface. It facilitates us to use the Deque. Unlike queue, we can add or delete the elements from both the ends.
ArrayDeque is faster than ArrayList and Stack and has no capacity restrictions.

Questions:
List A = new ArrayList<>();   -> List is interface and A has access to all method inside List
VS
ArrayList A = new ArrayList<>(); -> ArrayList is class and A have access of this class only

ArrayList vs LinkedList:
araylist internally uses dynamic array to store the data, linkedlist used doubly link list to store the data
arraylist good for storage and access, linkedlist good for manipulation
arraylist act as a list only, linkedlist can act as a list and queue both.
-----------------------------------------
Working of HashMap: how hashmap work internally
HashMap<String, Int> hm=new HashSet<String, Int>();

hm.put('Run', 572);

1 - once you add any record, first it create hashcode for the the key 'Run'. calling hashcode() method  ex. 773737773
2 - calculate & find the bucket index to store the record in the List ( default 16(0-15) Node)  ex. index 4
3 - then it store the value.  index 4 -> "Run|773737773|572|null"   or
                              index 7 -> "Run1|4443434343|777|null"
4 - In case if for some key index calculated is same then it store in a link list manner.
index 4 -> "Run|773737773|572|15"-> "Run2|2222222222|3333|null"
   15-> reference for other record Run2.

hm.get('Run');
1 - create hashcode for key.
2 - compare hashcode in the list of buckets and fetch the record.
-----------------------------------------
Comparable and Comparator:
Comparable and Comparator both are interfaces and can be used to sort collection elements.

> Comparable sort on single element. Comparator based on multiple element.

> Comparable affects the original class, i.e., the actual class is modified. Comparator doesn't affect the original class.

> Comparable provides compareTo() method to sort elements. Comparator provides compare() method to sort elements.

> Comparable is present in java.lang package. Comparator is present in the java.util package.

> Collections.sort(List) &  Collections.sort(List, Comparator)

-----------------------------------------
stream: sequence of objects from a source, which supports aggregate operations like forEach, map, filter, reduce, find, match, and so on
Parallel Processing, Collectors, Statistics

Using stream, you can process data in a declarative way similar to SQL statements.

Collection interface has two methods to generate a Stream.
stream() – Returns a sequential stream considering collection as its source.
```

```
Collection interface has two methods to generate a Stream.
stream() – Returns a sequential stream considering collection as its source.
parallelStream() – Returns a parallel Stream considering collection as its source.


stream examples

List<Integer> numbers = Arrays.asList(6, 2, 2, 3, 7, 3, 5);
numbers.stream().map( i -> i*i).distinct().forEach(System.out::println);

List<String>strings = Arrays.asList("abc", "", "bc", "efg", "abcd","", "jkl");
int test = (int) strings.stream().filter(string -> string.isEmpty()).count(); //forEach(System.out::println);
System.out.println(test);

//limit the size of the stream
Random random = new Random();
random.ints().limit(10).forEach(System.out::println);

// sort the stream
Random random = new Random();
random.ints().limit(10).sorted().forEach(System.out::println);


Parallel Processing::
parallelStream is the alternative of stream for parallel processing
ex.
List<String> strings = Arrays.asList("abc", "", "bc", "efg", "abcd","", "jkl");
long count = strings.parallelStream().filter(string -> string.isEmpty()).count();
System.out.println(count);


Collectors::
Collectors are used to combine the result of processing on the elements of a stream.
Collectors can be used to return a list or a string.
List<String>strings = Arrays.asList("abc", "", "bc", "efg", "abcd","", "jkl");
List<String> filtered = strings.stream().filter(string -> !string.isEmpty()).collect(Collectors.toList());
System.out.println("Filtered List: " + filtered);  //Filtered List: [abc, bc, efg, abcd, jkl]

String mergedString = strings.stream().filter(string -> !string.isEmpty()).collect(Collectors.joining(", "));
System.out.println("Merged String: " + mergedString); //Merged String: abc, bc, efg, abcd, jkl


Statistics::
With Java 8, statistics collectors are introduced to calculate all statistics when stream processing is being done.
ex.
List<String> list = Arrays.asList("3", "6", "8", "14", "15");
// Using Stream mapToInt(ToIntFunction mapper)
// and displaying the corresponding IntStream
IntSummaryStatistics stats = list.stream().mapToInt(num -> Integer.parseInt(num)).summaryStatistics();
System.out.println("Highest number in List : " + stats.getMax());  //15
System.out.println("Lowest number in List : " + stats.getMin());  //3
```

```
System.out.println("Sum of all numbers : " + stats.getSum());   // 46
System.out.println("Average of all numbers : " + stats.getAverage()); //9.2

---------------------------
What is Classloader::
responsible for loading Java classes during runtime dynamically to the JVM, these Java classes aren't loaded into memory all at once, but when required by an application. Java classes are loaded by an instance of java.lang.ClassLoader
Java.lang.ClassLoader.loadClass() method is responsible for loading the class definition into runtime
---------------------------
What is Caching and how can you implement your Cache ?
Caching is a mechanism to enhance the performance of a system. It is a temporary memory (store frequently used data ) that lies between the application and the persistent database.
>Master data, List of products available in an eCommerce store.

types:
1 - In memery cache, memcache    (Redis )
2 - database caching. ex. first level cache of Hibernate or any ORM frameworks.
3 - Web server caching

in spring boot
@SpringBootApplication
@EnableCaching
public class SpringCacheApplication {
    public static void main(String[] args) {
        SpringApplication.run(SpringCacheApplication.class, args);
    }
}

@Cacheable(value="book", condition="#name.length < 50")
public Book findStoryBook (String name) {


Caching in hibernate:::
Hibernate provide 3 level of caching
1 - session cache - default
2 - second level cache  - caching objects across sessions, When this is turned on, objects will first be searched in the cache and if they are not found, a database query will be fired. Second level cache will be used when the objects are loaded using their primary key. This includes fetching of associations. Second level cache objects are constructed and reside in different memory locations.

3 - Query Cache
---------------------------
Convert string to Character Array
String a = "aasa";
char[] c = a.toCharArray();

Sort array
Arrays.sort(c);
---------------------------
war - web application resource.
ear - An enterprise archive file (EAR) is a Java archive (JAR) file format used to package modules as single archive files to ensure the coherent deployment of different modules to application servers

---------------------------
```

```
---------------------------
Which java class file will be called if same class is packed in two jar files?

If both jar files are in the same ClassLoader, for instance the Java classpath (-cp option), normally it should be the first file found in the jar list order.

an case of EAR file or in WEB-INF/lib or a WAR file, there is no warranty the container will load the same class between two startups. In that context, the only thing sure is that WEB-INF/classes is searched before WEB-INF/lib
-----------------------------
SpringBoot debugging::
To enable debugging, we would simply add the debug argument using the -D option:
java -jar myapp.jar -Dagentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=8000

With Maven, we can use the provided run goal to start our application with debugging enabled:
mvn spring-boot:run -Dagentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=8000
-----------------------------
Contructor:
called when initialize objet & have no return type
> 2 types   1.defualt, 2.parametirized
-----------------------------
What is singleton class and how can we make a class singleton?
Singleton class is a class whose only one instance can be created at any given time, in one JVM.
A class can be made singleton by
1) making its constructor private.
2) Write a static method that has return type object of this singleton class. Here, the concept of Lazy initialization in used to write this static method.

class Singleton
{
    // static variable single_instance of type Singleton
    private static Singleton single_instance = null;

    // variable of type String
    public String s;

    // private constructor restricted to this class itself
    private Singleton()
    {
        s = "Hello I am a string part of Singleton class";
    }

    // static method to create instance of Singleton class
    public static Singleton getInstance()
    {
        if (single_instance == null)
            single_instance = new Singleton();

        return single_instance;
    }

    public static void main(String args[])
    {
        // instantiating Singleton class with variable x
        Singleton x = Singleton.getInstance();
```

length : 81,182 lines : 1,973 Ln : 481 Col : 61 Sel : 0 | 0 Windows (CR LF) UTF-8 INS

TeX file

```
        // instantiating Singleton class with variable y
        Singleton y = Singleton.getInstance();

        // instantiating Singleton class with variable z
        Singleton z = Singleton.getInstance();

        // changing variable of instance x
        x.s = (x.s).toUpperCase();

        System.out.println("String from x is " + x.s);
        System.out.println("String from y is " + y.s);
        System.out.println("String from z is " + z.s);
        System.out.println("\n");

        // changing variable of instance z
        z.s = (z.s).toLowerCase();

        System.out.println("String from x is " + x.s);
        System.out.println("String from y is " + y.s);
        System.out.println("String from z is " + z.s);
    }
}
```

---

what is aggregation in java?

If a class have an entity reference, it is known as Aggregation. Aggregation represents HAS-A relationship.
Consider a situation, Employee object contains many informations such as id, name, emailId etc.
It contains one more object named address, which contains its own informations such as city, state, country, zipcode etc. as given below.

```
class Employee{
    int id;
    String name;
    Address address;//Address is a class
...
}
```

What is composition in Java?
Composition is again specialized form of Aggregation and we can call this as a "death" relationship.
It is a strong type of Aggregation. Child object dose not have their lifecycle and if parent object deletes all child object will also be deleted.

ex:
Let's take again an example of relationship between House and rooms. House can contain multiple rooms there is no independent life of room and any room can not belongs to two different house if we delete the house room will automatically delete.

JDBC driver:
JDBC Driver is a software component that enables java application to interact with the database

---

execute:
used to execute any SQL query and it returns TRUE if the result is an ResultSet such as running Select queries.
The output is FALSE when there is no ResultSet object such as running Insert or Update queries. We can use

```
getResultSet() to get the ResultSet and getUpdateCount() method to retrieve the update count.

executeQuery:
used to execute Select queries and returns the ResultSet.

executeUpdate: used to execute Insert/Update/Delete (DML) statements or DDL statements that returns nothing.
The output is int and equals to the row count for SQL Data Manipulation Language (DML) statements. For DDL statements, the output is 0
-------------------------
Hibernate
Object-relational mapping or ORM

advantages of Hibernate over JDBC?
>removes a lot of boiler-plate code that comes with JDBC API, the code looks more cleaner and readable.
>supports inheritance, associations and collections. These features are not present with JDBC API.

------------------------
exception handling
here are five keywords used to handle exceptions in java:

try
catch
finally
throw
throws

Can use try without catch?  Yes
super class for exception?  Class Exception.
-------------------------
Error vs Exception

Error-Ocurs at runtime, ex. OutOfMemory error
Exception-human error ex. FileNotFoundException, ClassNotFoundException

----------------------
Checked Exception:
The classes that extend Throwable class except RuntimeException and Error are known as checked exceptions.
Example: IOException, SQLException, ClassNotFoundException, FileNotFoundException etc

Unchecked Exception:
The classes that extend RuntimeException are known as unchecked exceptions
Example: ArithmeticException, NullPointerException, ArrayOutOfBoundException etc.
----------------------
Q5. What are the differences between throw and throws?
Throw is used to explicitly throw the exceptions.
Throws is used along the method to notify the calling instance that it might throw an exception.
Throw:
void myMethod() {
  //Throwing single exception using throw
  throw new ArithmeticException("An integer should not be divided by zero!!");
}
..
```

```
..
Throws:

//Declaring multiple exceptions using throws
void myMethod() throws ArithmeticException, NullPointerException{
  //Statements where exception might occur
}
----------------------
Exception Hierarchy

Object -> Throwable
            -> Exception
                -> Checked
                -> Unchecked

            -> Error
                -> VirtualMachineError
                -> AssertionError
----------------------
Can we write multiple catch blocks under single try block?

Yes
----------------------
Difference between system.exit() and return ?
System.exit() completely halts you program, but where a return will continue to work
----------------------
Regular expression

String line = "This order was placed for QT3000! OK?";
String pattern = "(.*)(\\d+)(.*)";

// Create a Pattern object
Pattern r = Pattern.compile(pattern);

// Now create matcher object.
Matcher m = r.matcher(line);
if (m.find( )) {
 System.out.println("Found value: " + m.group(0) );
 System.out.println("Found value: " + m.group(1) );
 System.out.println("Found value: " + m.group(2) );
}else {
 System.out.println("NO MATCH");
}
---------------------------
Twilio  - is a 3rd party application used to send SMS and make voice calls from our application.
It allows us to send the SMS and make voice calls programmatically.
-------------------------
A service shared by two controller through @autoWired are singleton

@Service
Class Data {
```

A service shared by two controller through @autoWired are singleton

```
@Service
Class Data {
..

Controller1 {
@AutoWired Data data;  //singleton , same thread shared by each instances

Controller2 {
@AutoWired Data data; //singleton

how to make it non Singleton.
ans: By making bean scope = Prototype.. default-Singleton.
```

Alternate of using @autowired to add service or repository

```
// @Autowired EmployeeRepository employeerepository

@Component
public class Controller{
    EmployeeRepository employeerepository;
    @Autowire
    public Controller(EmployeeRepository employeerepository){
        this.employeerepository = employeerepository;
    }
}
```

---

function interface: java8
interface having only one abstract method.

SPRING BOOT

Microservice: .an architecture that allows the developers to develop and deploy services independently.
Each service running has its own process and this achieves the lightweight model to support business applications.
Advantage: Easy deployment, Simple scalability, Compatible with Containers, Minimum configuration, Lesser production time

---

Configure HTTPS in spring boot

```
server.port: 443
server.ssl.key-store: keystore.p12
server.ssl.key-store-password: springboot
server.ssl.keyStoreType: PKCS12
server.ssl.keyAlias: tomcat
```

Scheduling
The @EnableScheduling annotation is used to enable the scheduler for your application.
This annotation should be added into the main Spring Boot application class file.

@SpringBootApplication (automatically add @Configuration, @EnableAutoConfiguration, @EnableWebMvc, @ComponentScan)
@EnableScheduling

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

The @Scheduled annotation is used to trigger the scheduler for a specific time period.

// @Scheduled(fixedRate = 1000)
// @Scheduled(fixedDelay = 1000, initialDelay = 1000)
@Scheduled(cron = "0 * 9 * * ?")
public void cronJobSch() throws Exception {
}
------------------------------
Database connection setup java::

in application.properties - spring.datasource.jndi-name=jdbc/VFSDB
------------------------------
Slf4j - Simple Logging Facade for Java provides a Java logging API by means of a simple facade pattern
Log4J
log level -> INFO, ERROR or WARN

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
private static final Logger log = LoggerFactory.getLogger(Application.class);
log.info("Customers found with findAll():");
---------------
Spring Boot Actuator
Spring Boot provides actuator to monitor and manage our application.
It has HTTP endpoints. when application is pushed to production,
you can choose to manage and monitor your application using HTTP endpoints. (like shutdown, trace, info..)

in pom.xml
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-actuator</artifactId>
    </dependency>
</dependencies>
---------------
import lombok.Data;
@Data
@Data is like having implicit @Getter, @Setter, @ToString, @EqualsAndHashCode and @RequiredArgsConstructor annotations on the
class (except that no constructor will be generated if any explicitly written constructor exists).
@Data generates all the boilerplate that is normally associated with simple POJOs (Plain Old Java Objects)
and beans: getters for all fields, setters for all non-final fields, and appropriate toString,
equals and hashCode implementations that involve the fields of the class, and a constructor that
initializes all final fields, as well as all non-final fields with no initializer that have been marked with @NonNull,
in order to ensure the field is never null.

@Where(clause = "ACTIVE_EXAMPLE_INDICATOR='Y'") to set global condition for entity
```

```
@EqualsAndHashCode - Lombok provides @EqualsAndHashCode annotation to generate equals() and hashCode() methods automatically.


DTO - its like model
A Data Transfer Object is an object that is used to encapsulate data, and send it from one subsystem of an application to another.
DTOs are most commonly used by the Services layer in an N-Tier application to transfer data between itself and the UI layer.
The main benefit here is that it reduces the amount of data that needs to be sent across the wire in distributed applications.
They also make great models in the MVC pattern.


@CrossOrigin - enable cross origin request only for specific method.
customize this behavior by specifying the value of one of the annotation attributes:
origins, methods, allowedHeaders, exposedHeaders, allowCredentials or maxAge

@Transactional
@RequestMapping(value = "/tests/tests", method = RequestMethod.GET)


@Validated annotation to validate the input

RequestMethod annotation method
method = RequestMethod.POST
method = RequestMethod.GET
method = RequestMethod.PUT
method = RequestMethod.DELETE


POJO - plain old java object
ex: this is pojo
class Point {
    private double x;
    private double y;
    public double getX() { return x; }
    public double getY() { return y; }
    public void setX(double v) { x = v; }
    public void setY(double v) { y = v; }
    public boolean equals(Object other) {...}
}
IDEs can do this for you, but the most elegant way is to use the annotations defined by Project Lombok:
import lombok.Data;

@Data
class Point {
    private double x;
    private double y;
}


JPA vs HiberNate::
JPA is the interface while Hibernate is the implementation.
```