1. Bagaimana cara mengimpor class **WebSocketCreatorFactory**? Tuliskan code caranya!

import spark.embeddedserver.jetty.websocket.WebSocketCreatorFactory;

1. Bagaimana cara mengimpor method **encode()** dari class **Base64**? Tuliskan code caranya!

import static spark.Base64.encode;

1. Bagaimana cara mengimplementasi interface **Filter** dan interface **Wrapper**? Tulislah sebuah class bernama **House**, lalu class **House** ini mengimplementasi kedua interface tersebut.

import spark.utils.Wrapper;

public class House implements Filter, Wrapper {

}

1. Bagaimana cara meng-extends class **Access**? Melanjutkan menulis class bernama **House** pada soal di atas, cobalah meng-extends class **Access**

public class House extends Access {

}

1. Sebutkan class apa saja yang dapat mengakses attribute **DEFAULT\_ACCEPT\_TYPE** milik class **Service**!

Attribute tersebut memiliki hak akses protected artinya hanya bisa di akses oleh kelas itu sendiri dan class satu package serta class turunannya

Class Access, CustomErrorPages, ExceptionHandler, ExcecptionHandlerImpl, ExceptionMapper, Experimental, Filter, FilterImpl, H[altException](https://github.com/perwendel/spark/blob/c697d981268c538dfeeb38786bd9eb0a03ab86f0/src/main/java/spark/HaltException.java)

1. Sebutkan class apa saja yang dapat mengakses method **getApplication()** milik class **SparkFilter**!

Class [FilterTools](https://github.com/perwendel/spark/blob/7486deb3c055da15e2cb94cead5007a4daeba2b0/src/main/java/spark/servlet/FilterTools.java), [SparkApplication](https://github.com/perwendel/spark/blob/7486deb3c055da15e2cb94cead5007a4daeba2b0/src/main/java/spark/servlet/SparkApplication.java), [SparkFilter](https://github.com/perwendel/spark/blob/7486deb3c055da15e2cb94cead5007a4daeba2b0/src/main/java/spark/servlet/SparkFilter.java)

method **getApplication()** memiliki hak akses protected artinya hanya bisa di akses oleh class itu sendiri dan class satu package serta class turunannya

1. Tuliskan bagaimana cara menginstansiasi class **Service**!

Service x = new Service();

1. Berdasarkan file **Session.java**, **pada baris 27** dan **28**,

Session(HttpSession session, Request request) {

Assert.notNull(session, "session cannot be null"); // baris 27

Assert.notNull(request, "request cannot be null"); // baris 28

}

berikan analisis bagaimana mungkin parameter pertama dari method **notNull()** dapat menerima tipe data yang berbeda! Pada **baris 27** menerima tipe data **HttpSession**, sedangakan pada **baris 28** menerima tipe data **Request**. Kenapa tidak error? Bagaimana mungkin?

Karena pada class Assert method notNull memiliki parameter yang memiliki tipe data Object dan String. Tipe data Object merupkana tipe data khusus yang bisa menampung beragam data, termasuk memiliki function/method sendiri.

1. Berdasarkan file **Service.java,** pada **baris 150**, apabila kita tambahkan keyword **static** pada method tersebut, akan muncul error pada **baris 154**. Berikan analisnya kenapa error!

Mungkin yang dimaksud baris 142

Terjadi error jika ditambahkan keyword static, karena dalam method tersebut sudah ada keyword this yang merepresentasikan objek dari class itu sendiri.

1. Berdasarkan file **HttpRequestWrapper.java**, pada **baris 38**, apa maksud dari **super(request)**?

Keyword super artinya merepresentasikan objek dari kelas induk, berarti super(request) artinya mengeksekusi konstuktor dengan value request dari kelas **HttpServletRequestWrapper** sebagai kelas induk

1. Berdasarkan file **HttpRequestWrapper.java**, pada **baris 51**, apa maksud dari **super.getRequest()**

Maksudnya mengeksekusi **method getRequest** dari kelas **HttpServletRequestWrapper** sebagai kelas induk

1. Berdasarkan file **InputStreamSerializer.java**, pada **baris 24**, apa tipe data dari hasil kode berikut ini? **element instanceof InputStream**

mungkin maksudnya baris 34

tipe data yang dikembalikan adalah Object

1. Berdasarkan file **InputStreamSerializer.java**, pada baris **40**, jelaskan bagaimana mungkin variable is dapat menerima nilai yang dikandung oleh variable **element** padahal variable element bertipe data **Object**!

Karena variabel element return valuenya bertipe data InputStream

1. Berdasarkan file **Utf8StringBuilder.java,** tidak ada deklarasi method bernama **checkState()** pada class **Utf8StringBuilder**, jelaskan bagaimana mungkin pada **baris 58** bisa memanggil method **checkState().** Jelaskan juga bagaimana mungkin memanggil method **checkState()** yang tidak ada pada kelas **Utf8StringBuilder** hanya dengan menulis seperti berikut ini tanpa instansiasi?

Method checkState() itu milik class Utf8Appendable, karena class Utf8StringBuilder mewarisi class Utf8Appendable maka bisa memanggil method checkState(), dan caranya menggunakan method overriding. Method Overriding dilakukan saat kita ingin membuat ulang sebuah method pada sub-class atau class anak

1. Berdasarkan file **CustomErrorPage.java**, baris 42:

Apa itu CustomErrorPages? Sebuah class

Apakah method getInstance() dipanggil secara static atau tidak? Iya, dipanggil secara static

Apa return datatype dari method getInstance()? CustomErrorPages

Apa tipe data dari customPage? Tipe datanya Object

Method containsKey() menerima parameter bertipe data apa? Tipe data Integer

Apa return datatype dari method containsKey()? Return datatype boolean

1. Berdasarkan file **CustomErrorPage.java**, baris **59** dan **61**, jelaskan apa yang dilakukan pada baris tersebut!

Pada baris 59 ada kondisi jika nilai customRenderer bertipe data String, maka nilai customPage sama dengan nilai customRendeerer

Pada baris 61 ada kondisi jika nilai cutomRenderer bertipe data Route maka akan dilakukan handling request

1. Coba cek class **ClassPathResource**, berdasarkan constructor-nya ada berapa cara untuk melakukan instansiasi? Tuliskan!

ClassPathResource x = new ClassPathResource(“abc”);

1. Berdasarkan file **ClassPathResource.java**, baris **249**, jelaskan apa yang dilakukan baris tersebut?

Jika value parameter object bertipe data ClassPathResource ,maka akan mengembalikan nilai true

1. Coba cek class **CustomErrorPage**, jelaskan kenapa **constructor**-nya **private**? Apa tujuannya?

Agar tidak bisa diakses kelas manapun selain kelas CustomErrorPage

1. Berdasarkan file **TemplateViewRouteImpl.java**, baris **85**, apa maksud dari:**(ModelAndView) object** Kenapa bisa demikian

………

1. Berdasarkan file **Routable.java**, baris **524**, apa tipe data dari connect?

Tipe data dari connect yaitu HttpMethod

1. Pada class **Spark**, kita ingin menuliskan pemanggilan method **raise()** milik **class NotSupportedException**. Agar tidak error, bagaimana cara menuliskan pemanggilannya?

import static spark.NotSupportedException.raise;

**NotSupportedException.raise(“abc”, “abc”);**

1. Apa hasil/keluaran dari baris berikut ini?

**(new BytesSerializer()).canProcess(new BytesSerializer());**

……

1. Berdasarkan file **FilterImpl.java**, baris **84**, jelaskan apa yang dilakukan pada baris tersebut!

this(path, acceptType);

Mengisi nilai parameter pada konstruktor FilterImpl yang mempunyai 2 parameter pada baris 78

1. Berdasarkan file **FilterImpl.java**, baris **66** hingga **76**, jelaskan dengan lengkap apa yang dilakukan pada setiap barisnya!

Pada methode create memiliki 3 parameter, jika nilai parameter acceptType null maka nilai variabel acceptType sama dengan nilai konstanta DEFAULT\_ACCEPT\_TYPE.

Lalu saat mengembalikan method FilterImpl kan mengoveride method handle dari class Filter yang memiliki 2 parameter request dan response