public class Main {  
 public static void main(String[] args) {  
VirtualThread virtualThread = new VirtualThread();  
 Virtual3 v3 = new Virtual3();  
 v3.r();  
virtualThread.Virtual();  
virtualThread.Virtual2();  
 }  
}  
 class VirtualThread {  
 public void Virtual() {  
 ThreadLocal<String> threadLocal = new ThreadLocal<>();  
 // ExecutorService exc = Executors.newFixedThreadPool(5);  
 threadLocal.set(Thread.*currentThread*().getName());  
  
 Thread t1 = new Thread(() -> {  
 threadLocal.set(Thread.*currentThread*().getName());  
 });  
  
 t1.start();  
 System.*out*.println(threadLocal.get());  
  
  
 }  
  
 public void Virtual2() {  
 ThreadLocal<String> threadLocal = new ThreadLocal<>();  
  
 ExecutorService exc = Executors.*newFixedThreadPool*(5);  
  
 exc.submit(() -> {  
 threadLocal.set(Thread.*currentThread*().getName());  
 threadLocal.remove();  
 });  
  
 for (int i = 0; i < 5; i++) {  
 exc.submit(() -> {  
 System.*out*.println(threadLocal.get());  
 });  
 }  
 }  
 }  
 class Virtual3 implements Runnable {  
 @Override  
 public void run() {  
 System.*out*.println("Virtual Thread");  
 }  
  
 public void r(){  
 for (int i = 0 ; i < 5; i++) {  
 Thread th = Thread.*ofVirtual*().start(this::run);  
 }  
 }  
 }

**THREAD POOL**

import org.apache.logging.log4j.LogManager;  
import org.apache.logging.log4j.Logger;  
import org.springframework.stereotype.Component;  
  
import java.util.ArrayList;  
import java.util.concurrent.\*;  
  
  
@Component  
public class ThreadPool {  
  
 private static final Logger *log* = LogManager.*getLogger*(ThreadPool.class);  
 ArrayList<String> arr = new ArrayList<>();  
  
 public ArrayList<String> getArr() {  
 return arr;  
 }  
  
 public void Thread() {  
  
 ThreadPoolExecutor exc = new ThreadPoolExecutor(2, 4, 1L, TimeUnit.*MINUTES*,  
 new ArrayBlockingQueue<>(2), new CustomThreadFactory(), new CustomAbortPolicy());  
 exc.allowCoreThreadTimeOut(true);  
  
 ScheduledExecutorService scheduledExecutorService = Executors.*newScheduledThreadPool*(2);  
  
 Future<String> future = scheduledExecutorService.schedule(() -> {  
 System.*out*.println("Scheduled Thread");  
 return "Scheduled Thread Future";  
 },5,TimeUnit.*SECONDS*);  
  
  
 try {  
 *log*.info(future.get());  
 } catch (InterruptedException e) {  
 throw new RuntimeException(e);  
 } catch (ExecutionException e) {  
 throw new RuntimeException(e);  
 }  
  
  
 for (int i = 0; i < 10; i++) {  
  
 exc.execute(() -> {  
 try{  
 Thread.*sleep*(5000);}  
 catch (Exception e){  
 e.printStackTrace();}  
  
 System.*out*.println();  
 //System.out.println(exc.getActiveCount());  
 System.*out*.println("task completed : " + exc.getCompletedTaskCount());  
 arr.add(Thread.*currentThread*().getName());  
  
 System.*out*.println(Thread.*currentThread*().getName());  
 System.*out*.println("Que : " + exc.getQueue().size());  
 System.*out*.println("Pool size : " +exc.getMaximumPoolSize());  
 // System.out.println(exc.getCorePoolSize());  
 System.*out*.println();  
  
 });  
  
 }  
  
 exc.shutdown();  
 try {  
 exc.awaitTermination(2,TimeUnit.*SECONDS*);  
 System.*out*.println(exc.isTerminated());  
 } catch (InterruptedException e) {  
 throw new RuntimeException(e);  
 }  
 exc.execute(() -> System.*out*.println("SKS"));  
 }  
  
  
}  
  
class CustomThreadFactory implements ThreadFactory {  
 @Override  
 public Thread newThread(Runnable r) {  
 Thread t = new Thread(r);  
 // t.setName("SKS\_THREAD");  
  
 t.setDaemon(false);  
 t.setPriority(Thread.*NORM\_PRIORITY*);  
 return t;  
 }  
}  
  
class CustomAbortPolicy implements RejectedExecutionHandler {  
 @Override  
 public void rejectedExecution(Runnable r, ThreadPoolExecutor executor) {  
 // System.out.println("rejectedExecution : "+r.toString());  
 System.*out*.println("rejectedExecution : "+executor.toString());  
 }  
}

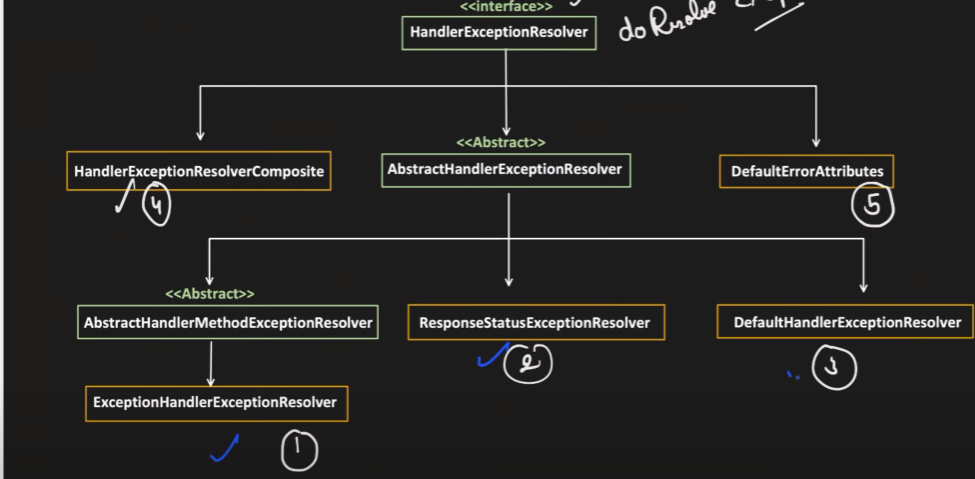
**ASYNC TRANSCTION**

**package com.example.scopebean.coding1;  
  
  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.scheduling.annotation.AsyncConfigurer;  
import org.springframework.scheduling.concurrent.ThreadPoolTaskExecutor;  
  
import java.util.concurrent.\*;  
import java.util.concurrent.atomic.AtomicInteger;  
  
@Configuration  
public class Config implements AsyncConfigurer {  
  
 private ThreadPoolExecutor PoolExecutor ;  
  
 @Override  
 public synchronized Executor getAsyncExecutor() {  
 if(PoolExecutor == null){  
 int min = 2;  
 int max = 5;  
 int que = 5;  
  
 PoolExecutor = new ThreadPoolExecutor(min , max,4, TimeUnit.*SECONDS*,new ArrayBlockingQueue<>(que), new CustomieThreadFactory(), new ThreadPoolExecutor.AbortPolicy());  
 }  
 return PoolExecutor;  
 }  
  
 @Bean("Pool")  
 public Executor getExecutor(){  
 int min = 2;  
 int max = 4;  
 int que = 5;  
  
 ThreadPoolTaskExecutor executor = new ThreadPoolTaskExecutor();  
 executor.setCorePoolSize(min);  
 executor.setMaxPoolSize(max);  
 executor.setQueueCapacity(que);  
 executor.setKeepAliveSeconds(60);  
 executor.setThreadNamePrefix("Async-");  
 executor.initialize();  
  
 return executor;  
 }  
  
 private static class CustomieThreadFactory implements ThreadFactory {  
  
 AtomicInteger count = new AtomicInteger(1);  
 @Override  
 public Thread newThread(Runnable r) {  
 Thread t = new Thread(r);  
 t.setName("SKS - " + count.getAndIncrement());  
 return t;  
 }  
 }  
}**

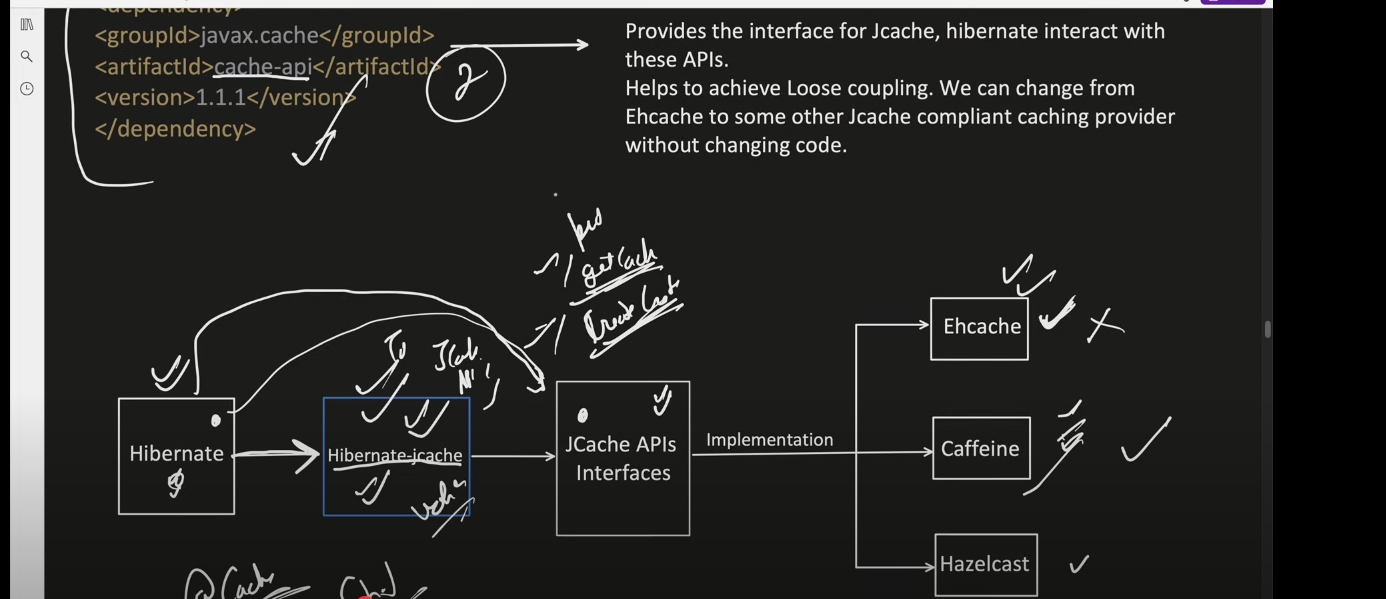
**package com.example.scopebean.coding1;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.scheduling.annotation.Async;  
import org.springframework.stereotype.Component;  
  
  
@Component  
public class AsyncTransc {  
  
 UserUtils userUtils;  
 @Autowired  
 public AsyncTransc(UserUtils userUtils) {  
 this.userUtils = userUtils;  
 }  
  
 @Async  
 public void User(){  
 System.*out*.println(Thread.*currentThread*().getName());  
 userUtils.update();  
 //return Thread.currentThread().getName();  
 }  
}  
  
  
@Component  
class UserUtils {  
  
 public void update(){  
 System.*out*.println(Thread.*currentThread*().getName());  
 System.*out*.println("update");  
 }  
}**

**package com.example.scopebean;  
  
  
import com.example.scopebean.coding.ASync;  
import com.example.scopebean.coding1.AsyncTransc;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
@RestController  
public class HomeController {  
 ASync aSync ;  
 AsyncTransc asyncTransc ;  
 public HomeController(ASync aSync , AsyncTransc asyncTransc) {  
 this.aSync = aSync;  
 this.asyncTransc = asyncTransc;  
 }  
 @GetMapping("/async")  
 public ResponseEntity<String> Async(){  
 aSync.Async();  
 System.*out*.println("inside " +Thread.*currentThread*().getName());  
 return ResponseEntity.*status*(HttpStatus.*ACCEPTED*).body("Async");  
  
 }  
  
 @GetMapping("/trans")  
 public ResponseEntity<String> ASyncTrans(){  
asyncTransc.User();  
 return ResponseEntity.*status*(HttpStatus.*ACCEPTED*).body("AsyncTransc");  
 }  
}**

**EXCEPTION HANDLER**

****

**L2 CACHE**

****

<dependency>  
 <groupId>org.ehcache</groupId>  
 <artifactId>ehcache</artifactId>  
 <version>3.10.8</version>  
</dependency>  
  
<!-- https://mvnrepository.com/artifact/org.hibernate.orm/hibernate-jcache -->  
<dependency>  
 <groupId>org.hibernate.orm</groupId>  
 <artifactId>hibernate-jcache</artifactId>  
 <version>7.1.0.Final</version>  
</dependency>  
  
<!-- https://mvnrepository.com/artifact/javax.cache/cache-api -->  
<dependency>  
 <groupId>javax.cache</groupId>  
 <artifactId>cache-api</artifactId>  
 <version>1.1.1</version>  
</dependency>

