QUESTION 1:

```
@GetMapping("/shoppingCars/{cartNumber}")
 public ResponseEntity<ShoppingCart> getShoppingCart(@PathVariable “cartNumber” String cartNumber)


 @PostMapping("/shoppingCars")
 public ResponseEntity<Product> addProductToShopingCart (
                                                     @RequestParam(value= "cartNumber") String cartNumber,
@RequestParam(vlaue = "productNumber") String productNumber,
@RequestParam(vlaue= "quantity") String quantity,
 @RequestParam(vlaue= "operation") String operation)



 @DeleteMapping("/shoppingCars")
 public ResponseEntity<?> removeProductToShopingCart (
                                                     @RequestParam(value= "cartNumber") String cartNumber,
@RequestParam(vlaue = "productNumber") String productNumber,
@RequestParam(vlaue= "quantity") String quantity,
 @RequestParam(vlaue= "operation") String operation)



@PutMapping("/shoppingCars")
 public ResponseEntity<Product> changeQuantityInShopingCart (
                                                     @RequestParam(value= "cartNumber") String cartNumber,
@RequestParam(vlaue = "productNumber") String productNumber,
@RequestParam(vlaue= "quantity")  String quantity,
 @RequestParam(vlaue= "operation")  String operation)



@DeleteMapping("/carts/{cartNumber}")
 public ResponseEntity<?> clearShoppingCart(@Pathvariable cartNumber)
```

Question :2

A. Each `JmsListener` method operates within its individual thread, implying that the `currentvalue` will be under a multithreaded environment. In scenarios like the question propose where a high volume of messages come, they will arrive and concurrently will attempts to alter this `currentvalue`,so the outcome could be erroneous calculations.
B. So we have to look how make the variable ‘currentvalue’ threat-safe, one way could be use synchronized keyword that will force to only one threat to get access at the time. Other aproach could be use a database , where we store tha variable and use the mechanisms like transactions or asolations to be in charge of the syncronisations, but this could be more expensive and complicated solution, but will depend of the requirements of the problem.

Question :3

```
@Component
public class StringCounter {

@Autowired
private FileReader fileReader;

public void countStringInFiles() {
   fileReader.countHowOfterStringAppearsInFilesOfDirectory("searchString", "directory");
}

}



@Component
public class FileReader {

@Autowired
private ApplicationEventPublisher publisher;

@Async
public void countHowOfterStringAppearsInFilesOfDirectory(String searchString, String directory) {
...
publisher.publishEvent(new FinishEvent(count));
}

}



@Component
public class CountProcessor {

@EventListener
@Async
public void onEvent(FinishEvent event) {
     process(event.getCount());
}

public void process(int count){
…
}

}

@Component public class Logger {

@EventListener
@Async
public void onEvent(FinishEvent event) {
     log("Count = "+event.getCount());
}


public void log(String string){
…
}
}
```

```
@Component
public class EmailSender {
@EventListener
@Async
public void onEvent(FinishEvent event) {
sendEmail("Count = "+event.getCount(), "receiver@gmail.com");
}

public void sendEmail(String message, String emailAddress) {
…
}

}
```