2)

* viele Interfaces für Logik, damit die konkreten Klassen nicht mehr gekoppelt sind

3)

* erstellen einer abstrakten Fabrik
* originale Factory leitet von abstrakter ab
* SimpleFileInput nicht als Singleton, da Datei übergeben wird (für Arbeit mit verschiedenen Dateien in gleiche Instanz müsste Interface etc. an Struktur geändert werden)
* AbstractFactory nicht als Singleton da diese nicht instanziiert wird

6)

* zwischenschalten der DTOs, diese sollen im TodoService statt der Todos aufgerufen werden. Ein DTO gibt beim Speichern aber ein Todo zurück (gespeichert werden nur Todos)
* Tests: Access von Packages nur im jeweiligen Aufgabenbereich/Schicht; Annotations dürfen nur in den Packages/Schicht aufgerufen werden

7)

* Viele Daten werden in aneinandergereihten Steps verarbeitet, als Batch-Job
* 2: auf verschiedene Channels weiterleiten (mit Strings als Namen)

8)

* Presenter kriegt score und view und wird sofort als Observer beim Score registriert
* schließen des Fensters: Abmelden des Observers
* Action Listener für incrementGues/HomeScore
* notifiyObserver für benachrichtigen aller Observer und update

9)

* @Column(length=1000)
* übrige Annotations
* getter und setter beim Todo
* im TodoService mit Dto arbeiten außer bei erstelle neues Todo
* todoRepo extenden (Methode nach Vorgabe, nichts weiter, aus Todo muss Dto gemacht werden in der Ausgabe)
* delete löschen, nach id finden und date hinzufügen

10)

* Nutzer:Todos ist One-to-Many
* @OnetoMany ein Todo hat mehrere Kommentare mit mappedBy Attribut = wie Variable in Gegenstück heißt
* @Autowired an den Konstruktoren
* Dto für Comment nötig und immer richtigen Nutzer rausziehen
* neuen Comment über Repo saven

12)

* GetMapping von alle Kontakte gefilter zu Name beinhaltet Filter aus Feld
* für change müssen Attribute von vorherigem Kontakt geholt werden, lösche einfach nach ID
* actionListener save
* save: neuen Kontakt anlegen mit PostRequest über Rest und ansonsten ändern mit nem PutRequest