Spring training

Hellgren, Feb 2021

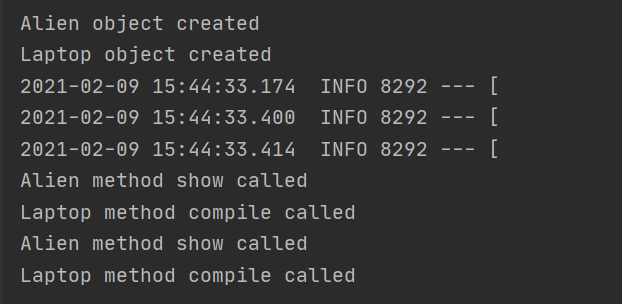
# SpringTeluskoBeanDemo

https:www.youtube.com/watch?v=K43qyHJXmWI

@Component make the class available as bean in spring container  
@Scope(value="prototype") this will apply prototype patter, object created when requested,  
 not present <=> singleton pattern, one created at start anyway, now new created  
public class Alien {  
 ---------- variables  
 private int id;  
 private String name;  
 @Autowired will make object search for laptop in spring container, search by type is default  
 @Qualifier("lap1") search by name instead of type  
 private Laptop laptop;  
 ---------- constructor(s)  
 public Alien() { System.*out*.println("Alien object created"); }  
 ---------- methods  
 public void show() { System.*out*.println("Alien method show called"); laptop.compile(); }  
}

@Component("lap1") object name specified to lap1, not needed if searched by type  
public class Laptop {  
 ---------- variables  
 private int id;  
 private String brand;  
 ---------- constructor(s)  
 public Laptop() { System.*out*.println("Laptop object created"); }  
 ---------- methods  
 public void compile() { System.*out*.println("Laptop method compile called"); }  
}

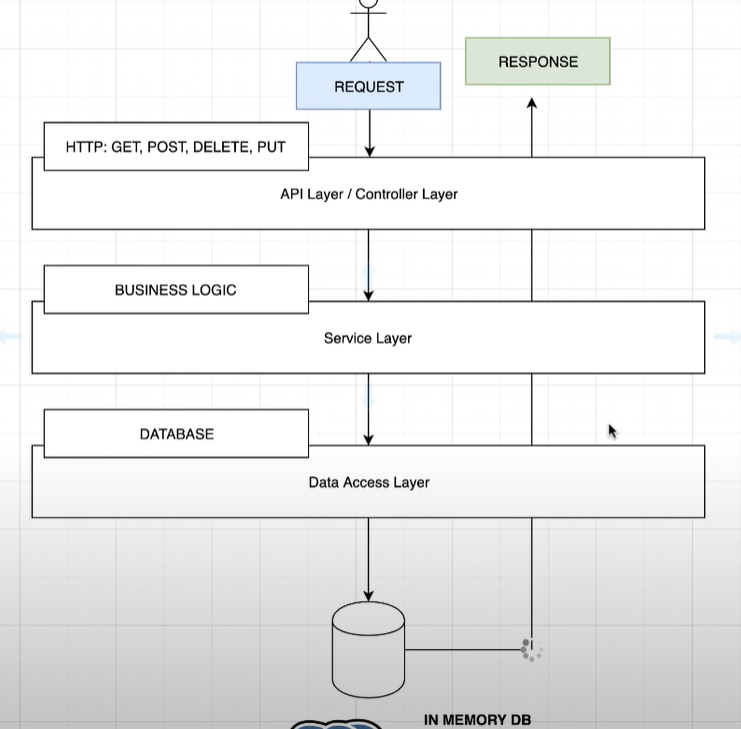
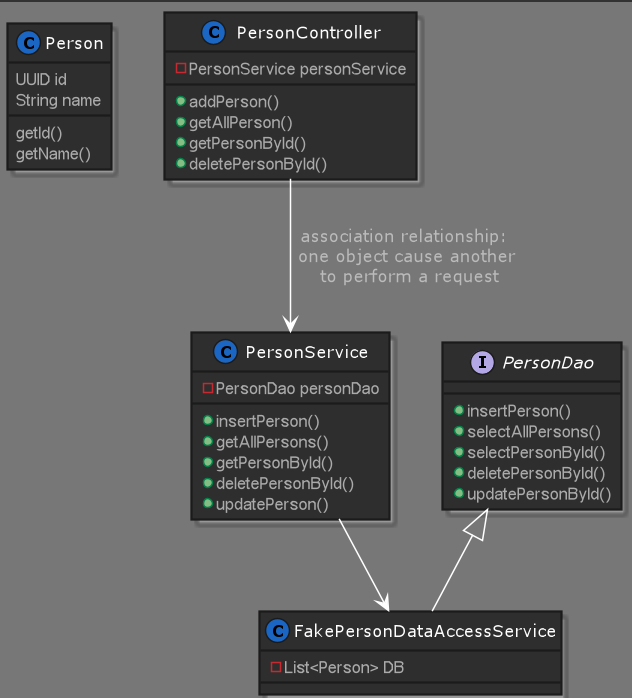
@SpringBootApplication  
@SpringBootApplication(exclude = {DataSourceAutoConfiguration.class })  
public class SpringApplication {  
 public static void main(String[] args) {  
 org.springframework.boot.SpringApplication.run(SpringApplication.class, args);  
 ConfigurableApplicationContext context=org.springframework.boot.SpringApplication.*run*(SpringApplication.class, args);  
 Alien a1 = context.getBean(Alien.class); a1.show();  
 Alien a2 = context.getBean(Alien.class); a2.show();  
 }  
}



How to reset version control: <https://stackoverflow.com/questions/16797260/remove-intellij-project-version-control>

# Spring Boot Tutorial for Beginners

https:www.youtube.com/watch?v=vtPkZShrvXQ

public class Person {  
 private final UUID id;  
 Need to have relevant dep. for ex. spring-boot-starter-validation, to make validation work  
 @NotNull(message="First name cannot be missing or empty")  
 @Size(min=2, message="First name must not be less than 2 characters")  
 private final String name;  
  
 JsonProperty matches json data to class fields  
 public Person(@JsonProperty("id") UUID id, @JsonProperty("name") String name) {  
 this.id = id; this.name = name;  
 }  
 public UUID getId() {  
 return id;  
 }  
 public String getName() {  
 return name;  
 }  
}

public interface PersonDao {  
 int insertPerson(UUID id, Person person);  
 default int insertPerson(Person person) {  
 default method implementation follows  
 UUID id= UUID.*randomUUID*(); return insertPerson(id,person); }  
 List<Person> selectAllPersons();  
 Optional<Person> selectPersonById(UUID id);  
 int deletePersonById(UUID id);  
 int updatePersonById(UUID id, Person person);  
}

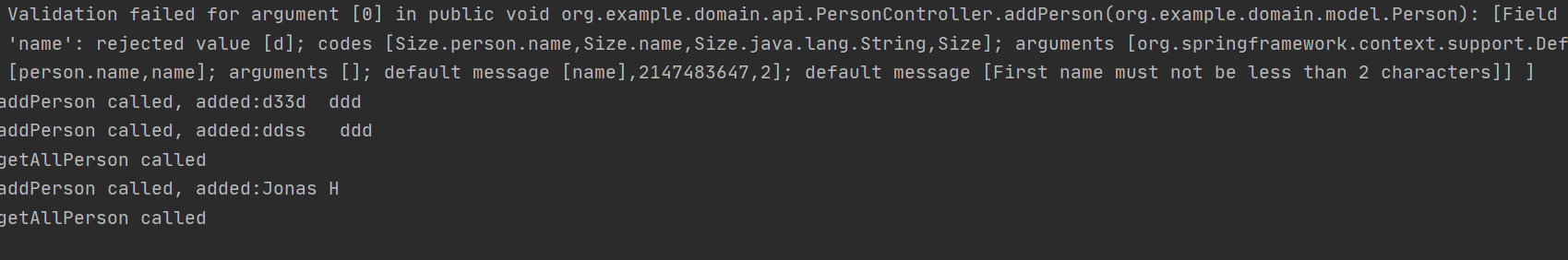
@RequestMapping("api/v1/person") client url to respond to  
@RestController express this is an object that defines endpoints clients can consume  
public class PersonController {  
 ------ Variables  
 @Autowired wires to personService object, it has the interface PersonDao  
 private final PersonService personService; reference to service  
 ------Constructor  
 public PersonController(PersonService personService) {  
 this.personService = personService;  
 }  
 ------ Methods  
 @PostMapping handles post requests from client  
 @Valid a triggers validations on person, @RequestBody put json body into object  
 public void addPerson(@Valid @NonNull @RequestBody Person person) {  
 System.*out*.println("addPerson called, added:"+person.getName());  
 personService.insertPerson(person);  
 }  
 @GetMapping handles get requests from client  
 public List<Person> getAllPerson() {  
 System.*out*.println("getAllPerson called");  
 return personService.getAllPersons();  
 }  
 @GetMapping(path = "{id}") handles get requests from client, with specific id included in request  
 public Person getPersonById(@PathVariable("id") UUID id) { take id from request and turn into UUID id  
 System.*out*.println("getPersonById called, found:"+personService.getPersonById(id)  
 .orElse(null).getName());  
 return personService.getPersonById(id)  
 .orElse(null); cen be improved by e.g. throw  
 }  
 @DeleteMapping(path = "{id}")  
 public void deletePersonById(@PathVariable("id") UUID id) {  
 personService.deletePerson(id);  
 }  
 @PutMapping(path = "{id}")  
 public void updatePersonById(@PathVariable("id") UUID id, @Valid @NonNull @RequestBody Person personToUpdate) {  
 personService.updatePerson(id,personToUpdate);  
 }  
}

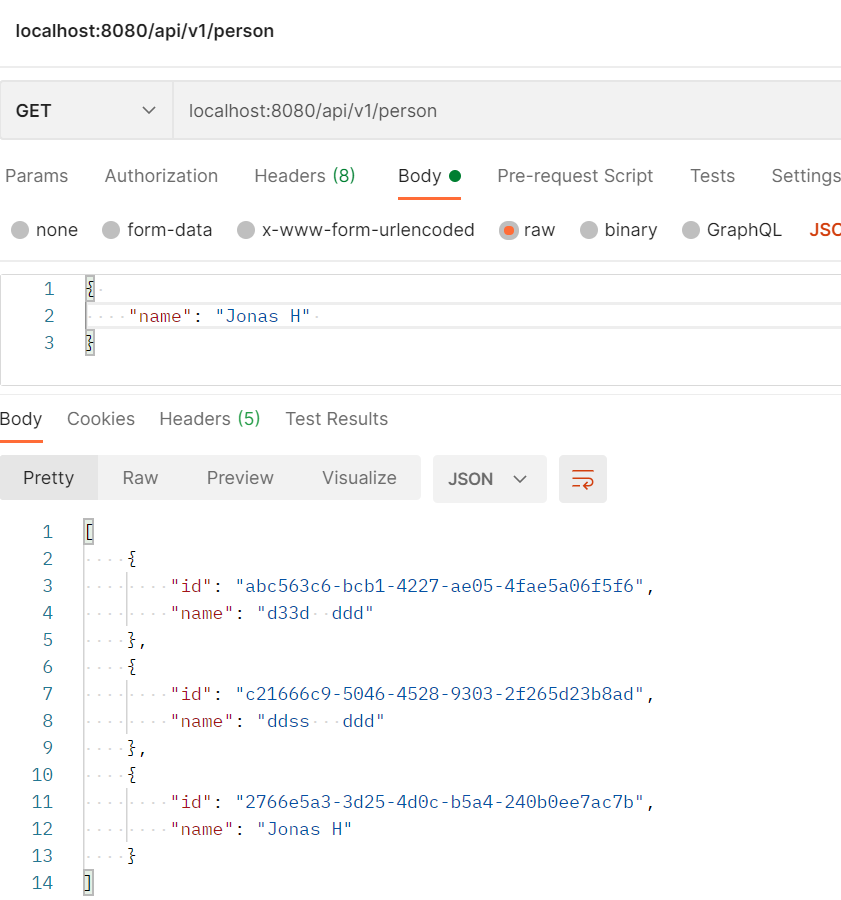
public class PersonDataAccessServicePostGres implements PersonDao {  
Not implemented, just to show easy to replace fake db  
 @Override

@Repository("fakeDao") can use @Component but rep more specific  
public class FakePersonDataAccessService implements PersonDao {  
 private static List<Person> *DB*=new ArrayList<>();  
 @Override  
 public int insertPerson(UUID id, Person person) {  
 *DB*.add(new Person(id, person.getName()));  
 return 1; assume always works  
 }  
 @Override  
 public List<Person> selectAllPersons() {  
 return *DB*;  
 }  
 @Override  
 public Optional<Person> selectPersonById(UUID id) {  
 return *DB*.stream() filter out first person with id id  
 .filter(person -> person.getId().equals(id))  
 .findFirst();  
 }  
 @Override  
 public int deletePersonById(UUID id) {  
 Optional<Person> personMaybe=selectPersonById(id);  
 if (personMaybe.isEmpty()) avoids null check by Optional  
 return 0; fail fast if not exists  
 *DB*.remove(personMaybe.get());  
 return 1;  
 }  
  
 @Override  
 public int updatePersonById(UUID id, Person updatePerson) {  
 return selectPersonById(id).  
 map(p -> { p is the return from selectP.  
 int indexOfPersonToUpdate = *DB*.indexOf(p);  
 if (indexOfPersonToUpdate >= 0) { replace corresponding post in DB if p exists  
 *DB*.set(indexOfPersonToUpdate, new Person(id, updatePerson.getName()));  
 return 1;  
 }  
 return 0;  
 })  
 .orElse(0);  
 }  
}

Some general comments:

* Can easily change data base type by @Qualifier("fakeDao".
* Need to have relevant dep. for ex. spring-boot-starter-validation, to make validation work

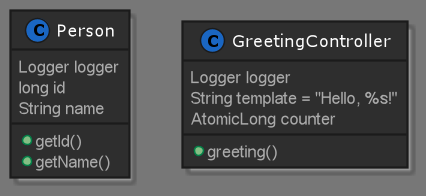




# Building a RESTful Web Service

https:spring.io/guides/gs/rest-service/

Simple REST controller that responds to get requests. Logging with debug and info level is performed. Classes described with uml diagram, uml code (\*.puml) viewable with PlantUml.

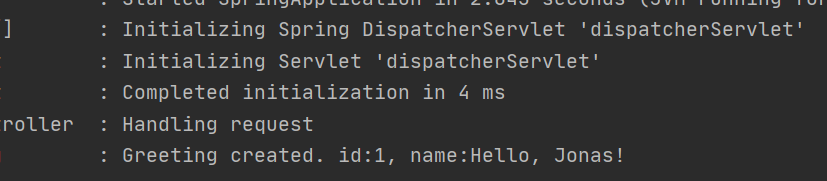


public class Greeting {  
Thanks to the Jackson JSON library we can marshal instances of type Greeting into JSON  
 private static Logger *logger* = LogManager.*getLogger*(Greeting.class);  
 private final long id;  
 private final String name;  
 public Greeting(long id, String name) {  
 this.id = id; this.name = name;  
 *logger*.info("Greeting created."+" id:" +id+", name:"+name);  
 }  
 Following getters are needed for spring to extract id and name field when sending as json  
 public long getId() { *logger*.debug("getId called"); return id; }  
 public String getName() { *logger*.debug("getContent called"); return name; }  
}

@RestController  
This code uses Spring @RestController annotation, which marks the class as a controller where every   
 method returns a domain object instead of a view. It is shorthand for including both   
 @Controller and @ResponseBody.  
public class GreetingController {  
 private static Logger *logger* = LogManager.*getLogger*(GreetingController.class);  
 private static final String *template* = "Hello, %s!";  
 private final AtomicLong counter = new AtomicLong();  
  
 @GetMapping("/greeting") HTTP GET requests to /greeting are mapped to the greeting() method  
 variable name below cant change name !!???  
 public Greeting greeting(@RequestParam(value = "name", defaultValue = "World") String name) {  
 The Greeting object must be converted to JSON. Thanks to Spring’s HTTP message converter   
 support, you need not do this conversion manually. Because Jackson 2 is on the   
 classpath, Spring’s MappingJackson2HttpMessageConverter is automatically chosen to   
 convert the Greeting instance to JSON.  
 *logger*.info("Handling request");  
 return new Greeting(counter.incrementAndGet(), String.*format*(*template*, name));  
 }  
}

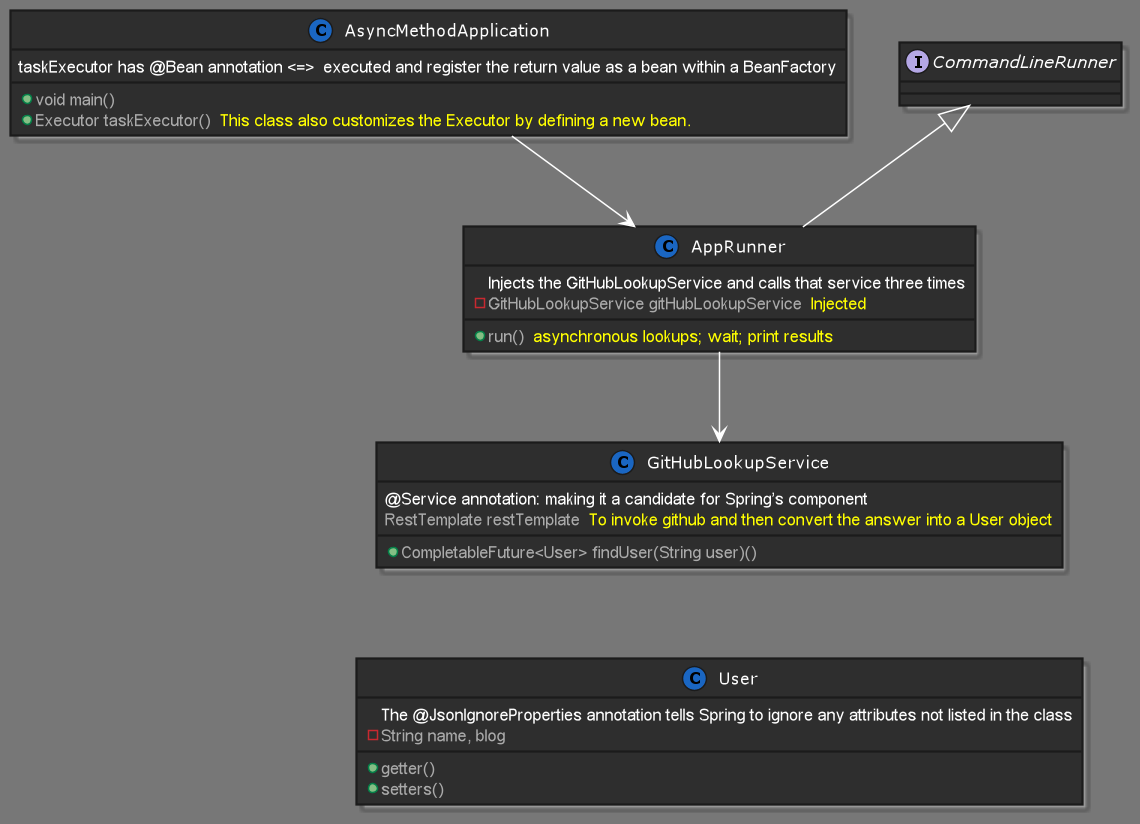
Following sets log level as info (application.properties..)

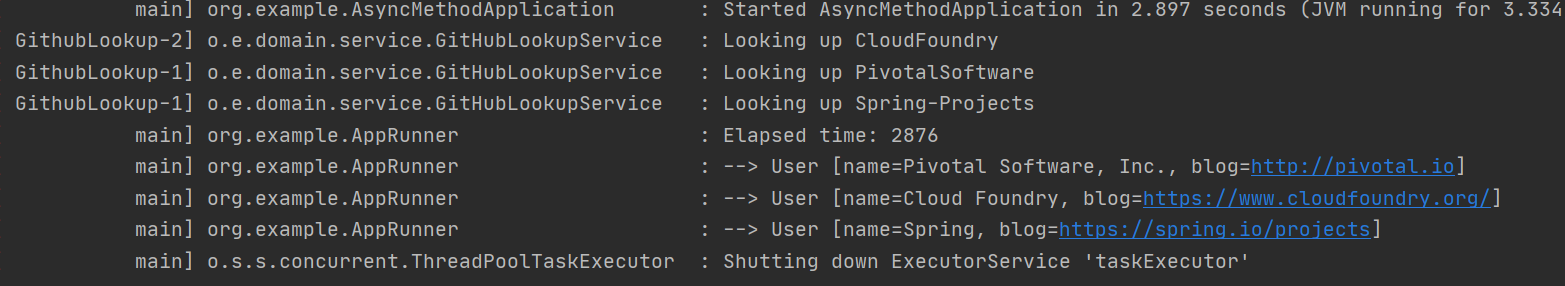
logging.level.root=info



# Creating Asynchronous Methods

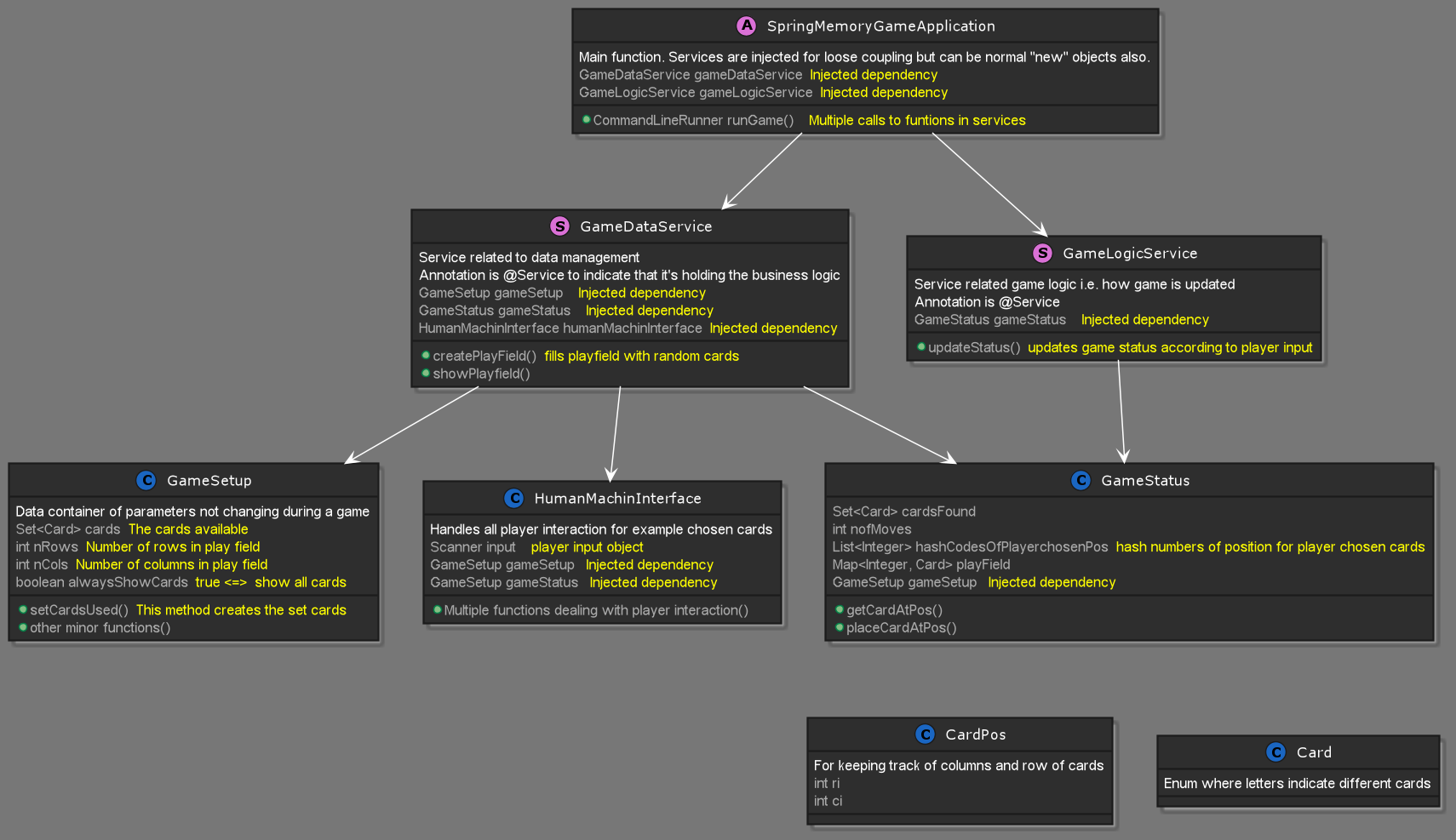
<https://spring.io/guides/gs/async-method/>

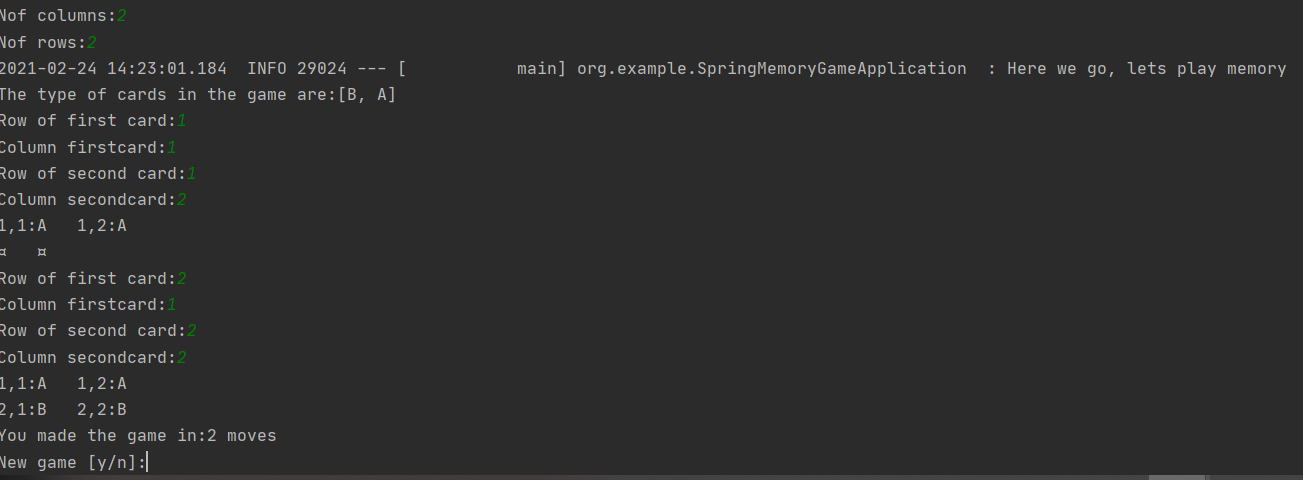




# Memory game

Classical memory game implemented in spring boot. Loose coupling between classes thanks to dependency injection,





<https://spring.io/guides/gs/spring-boot-docker/>