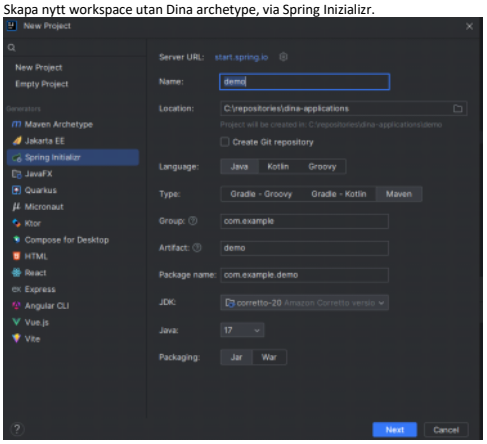
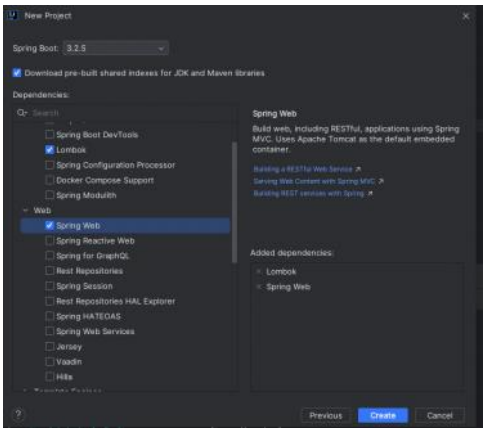
Skapa projekt med Java20

den 16 maj 2024 14:47

Skapa nytt workspace utan Dina archetype, via Spring Inizializr.





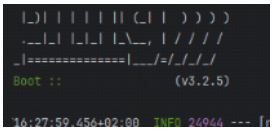
Dvs spring-boot-starter-web.

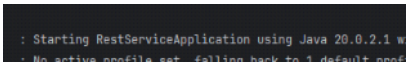
Detta är det enda som behövs, men man brukar också numera ta in följande två:
spring-boot-devtools
spring-boot-starter-actuator

Med spring-boot-devtools kan man göra hot-reload som man kan i frontend, räcker att spara filen.

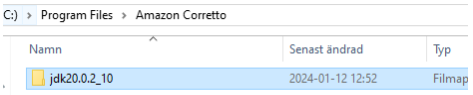
Spring boot 3.2.5 är rätt ny.
Min maven är: maven 3.9.6

JAVA_HOME	C:\Program Files\Amazon Corretto\jdk20.0.2_10
MAVEN_HOME	C:\Program Files\apache-maven-3.9.6\bin

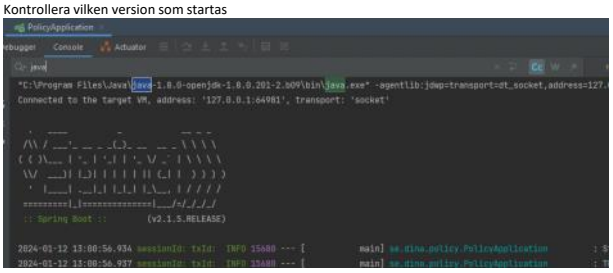




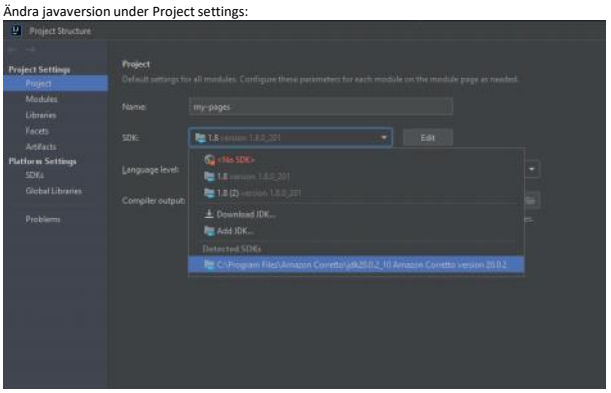
Här ligger min java 20

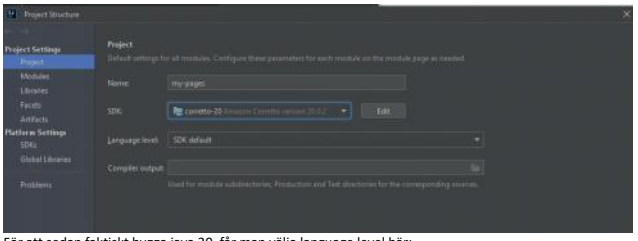


Kontrollera vilken version som startas

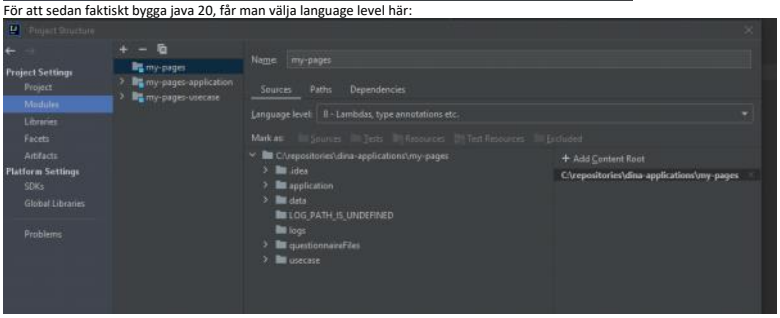


Ändra javaversion under Project settings:





För att sedan faktiskt bygga java 20, får man välja language level här:



Lab java 20

den 29 januari 2024 10:33

Saker att titta närmare på
https://advancedweb.hu/a-categorized-list-of-all-java-and-jvm-features-since-jdk-8-to-21/
New language features since Java 8 to 21 - Advanced Web Machinery

- Switch expressions
- Helpful NPExceptions
- Multi-line String literal called Text Blocks """. En sträng kan numera formatteras med radbryt, tabbar, citattecken, mm.
- Pattern matching i conditions med instanceof
- Record Classes

Saker som jag inte tycker är så intressant
- Local variable type inference (Var keyword)
- Try-with-resources (vid resurser, alltså BufferedReaders etc) Har funnits sedan före java 7 och förbättrats.
- Diamond operators för anonymous inner classes
- Allow private methods in interfaces (Sedan java 8 kan man lägga till default metoder i interface. Med java 9 kan dessa anropa andra privata metoder)
- Sealed classes

Switch statement => Switch expression
The Evolution Of Switch Statement From Java 7 to Java 17 | by Java Techie | Medium
-> Java 7: Integer
Java 8: String, enum

Java 12 - preview (tillgängligt i java 14):
- Ordet "break" ändrade till att betyda "return" (detta ersattes sedan med "yield" i java 13)
- Arrow operator / Enhanced switch (Det är denna som föreslås av intellij)
- Syntaxen för att flera värden returnerar samma värde är annorlunda

Med andra ord:
- Tidigare enbart switch statements, nu finns även Switch expressions. Dvs undviker subtila buggar med missade breaks
- Varje sats har sitt eget scope, med arrowfunktion. Kan vid behov returnera mha yield.
- Man behöver bara definiera "default" för primitiver. För enums märker man om det behövs

Java 17 - preview features (tillgängligt först i java 21):
Pattern matching:

```
return switch (obj) {
    case Integer i -> "It is an integer";
    case String s -> "It is a string";
    case Employee s -> "It is a Employee";
    default -> "It is none of the known data types";
};
```

Detta är en preview feature i java 17. Vill man absolut använda den kan man göra så här:

4. Patterns for Switch

Pattern matching for instanceof became a permanent feature in Java SE 16.
With Java 17, the application of pattern matching now also expands to *switch* expressions.
However, it is still a preview feature, so we need to enable preview to use it:

```
java --enable-preview --source 17 PatternMatching.java
```

Guarded patterns:

```
return switch (obj) {
    case Integer i -> "It is an integer";
    case String s -> "It is a string";
    case Employee employee && employee.getDept().equals("IT") -> "IT
Employee";
    default -> "It is none of the known data types";
};
```

Null cases:

```
case null -> "It is a null object";
```

Var keyword
Introducerades i java 10, förbättrades i java 11
var msg = "hello!";
Detta är inte som i java script "dynamic typing"
Typen bestäms istället i compile time.

Fördelar
?

Nackdelar
- Även om man själv kan få hjälp från IDE, försvårar detta läsbarhet vid kodgranskning mm
- Åtminstone ena sidan måste deklarerara typerna i en mapp, om man vill att det ska vara annat än Object
- Primitiva typer

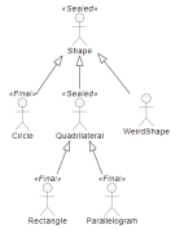
Helpful Nullpointers
Bättre felmeddelanden from Java 15. Numera får man:

```
java.lang.NullPointerException: Cannot invoke "se.dina.lab.java20Lek.ObjectWithDay.getDay()" because "objectWithDay" is null

    at se.dina.lab.java20Lek.NullPointers.proJava7(NullPointers.java:5)
    at se.dina.lab.unit.NullPointersTest.proJava7(NullPointersTest.java:11) <29 internal lines>
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1511) <9 internal lines>
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1511) <28 internal lines>
```

RecordClasses
Immutable data classes
- Kan ha speciella konstruktorer, men inte så mycket mer. Kan ha statiska metoder och instance metoder
- Responsobjekt och requestobjekt skulle kunna vara sånt.
- Alla våra dataobjekt som tex PolicyOption, de har ju lite annat också i sig. Detta skulle väl itne fungera?
- Man kan även ha dessa RecordClasses som local record classes inuti en klass för intemediate transformations.
    - Ex: ClaimRegistrationData objektet

Det står att detta enablar local enums och även interfaces ?

Sealed classes



Ser inte riktigt vitsen...
Men det står "consider using sealed classes over enums". Men hur och varför?
En enum kräver att alla värden finns i samma fil, och inte om det är någon konstant som representerar ett individuellt meddelande etc. Men ser ändå inte riktigt behovet.

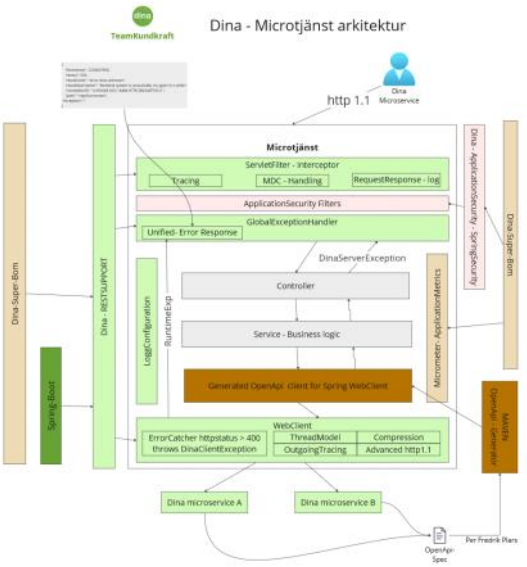
REST i nytt intellijprojekt utan dina koppling

den 16 maj 2024 17:15

Att göra:
Få till ett eget projekt som har samma innehåll/funktion som lab-java20, men utan dina-archetype.

Det enda som behövs, bortsett från spring-boot-starter-web är att annotera controllern med @RestController

WebClient är inte det jag behöver nu, inte heller OpenApi:



REST anrop

den 16 maj 2024 17:18

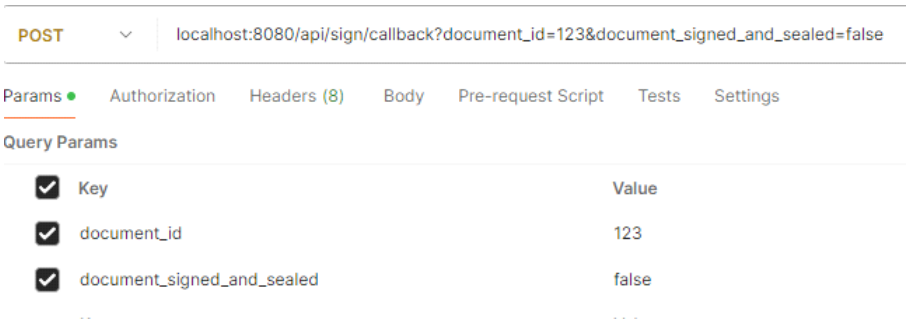
Använd UriComponentsBuilder om man har tillgång till spring (dvs i en applikation, men inte i tiaAdaptern)
Guide to UriComponentsBuilder in Spring | Baeldung

Uiconfig borde vara ett GET anrop men är en POST. Det är troligen pga att det har en request body.
Why an HTTP Get Request Shouldn't Have a Body | Baeldung on Computer Science

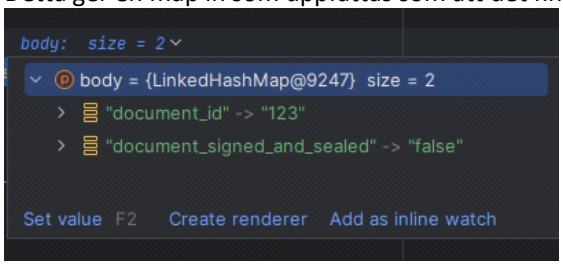
REST/Postman - post med map i body (third-party)

den 22 maj 2024 16:37

```
@PostMapping(value = "/api/sign/callback", consumes = MediaType.APPLICATION_FORM_URLENCODED_VALUE)
void signCallback(@RequestParam Map<String, String> body) {
    callbackService.handleHealthDeclarationSigned(body);
}
```



Alltså ett POST-anrop med ***två*** requestparameters.
Detta ger en map in som uppfattas som att det finns två saker i mappen:



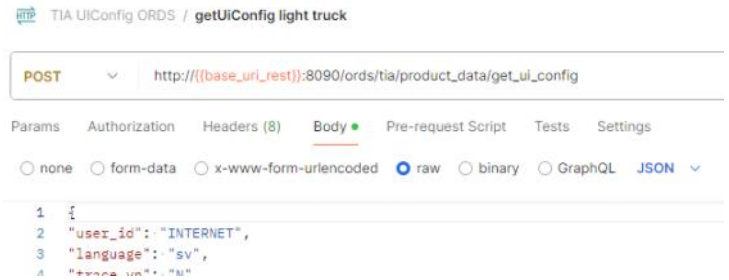
Och dessutom behövdes rätt content-type, dvs

Postman

den 21 maj 2024 07:09

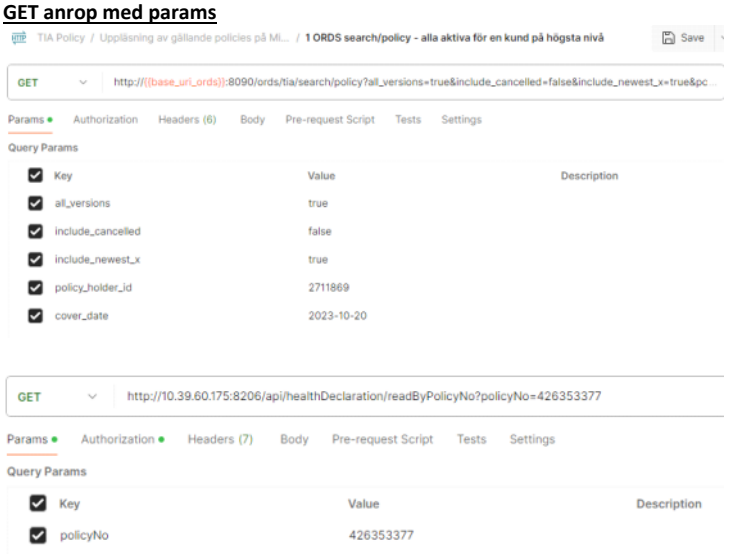
Ex på POST anrop med body

Sparat undan två kollektioner - lab och båtregistret

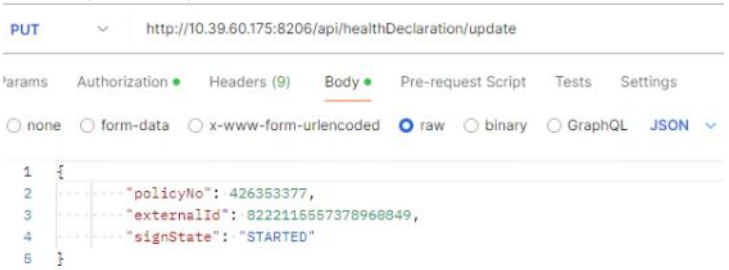


(Varför är detta inte GET???)

GET anrop med params



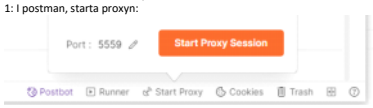
PUT anrop med body



Postman vs network

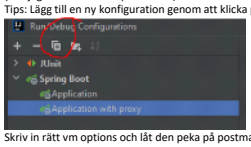
den 2 november 2023 09:39

Istället för network, använd postman!!
1: I postman, starta proxyn:



- Denna sägs ta typ alla anrop som görs på min dator => Ha den inte på när man inte behöver den. Kan pausas eller stoppas.
- För att den ska ta en viss applikation så måste man köra applikationen med speciella vm parametrar:
  -Dhttp.proxyHost=localHost -Dhttp.proxyPort=5559 -Dhttp.nonProxyHosts=

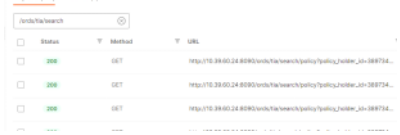
(Dvs jag behöll förvalda porten här)
Tips: Lägg till en ny konfiguration genom att klicka på Copy configuration:



Skriv in rätt vm options och låt den peka på postman.

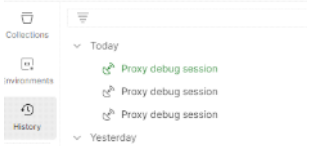
Proxy sessionen
Det finns mycket att grotta i här, men några basics:
Kan clearа alla tidigare anrop

Kan filtrera valda endpoints

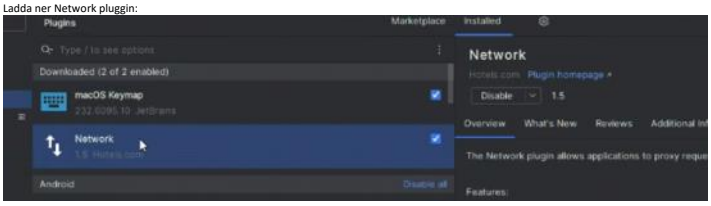


Klicka på ett av anropen och den öppnas i en ny flik.
Kan köras och sparas.

Annat:
Under history hittas sessionerna



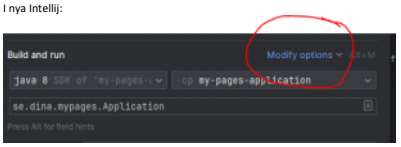
Ladda ner Network pluggin:

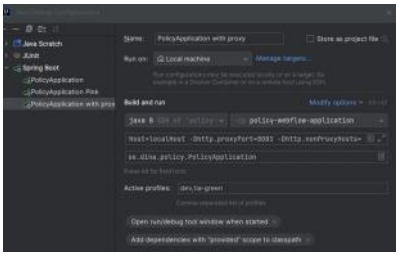


Välj lämplig port i settings i Network, tex 8083

I alla Intellij workspace man vill ha detta i, fyll i VM Options:
-Dhttp.proxyHost=localHost
-Dhttp.proxyPort=8083
-Dhttp.nonProxyHosts=

I nya Intellij:

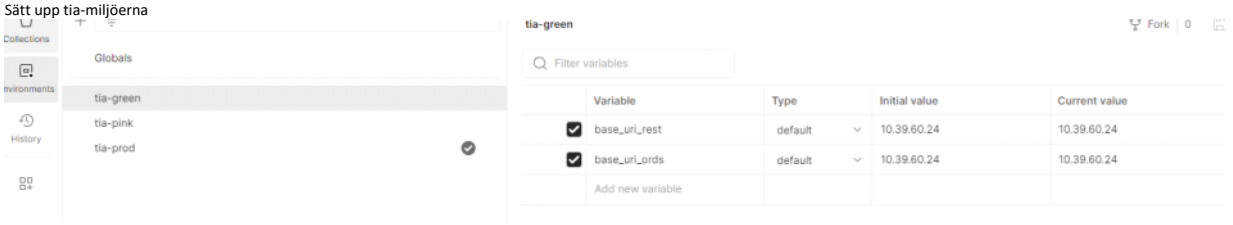




Postman - konfigurera env mm

den 2 november 2023 10:28

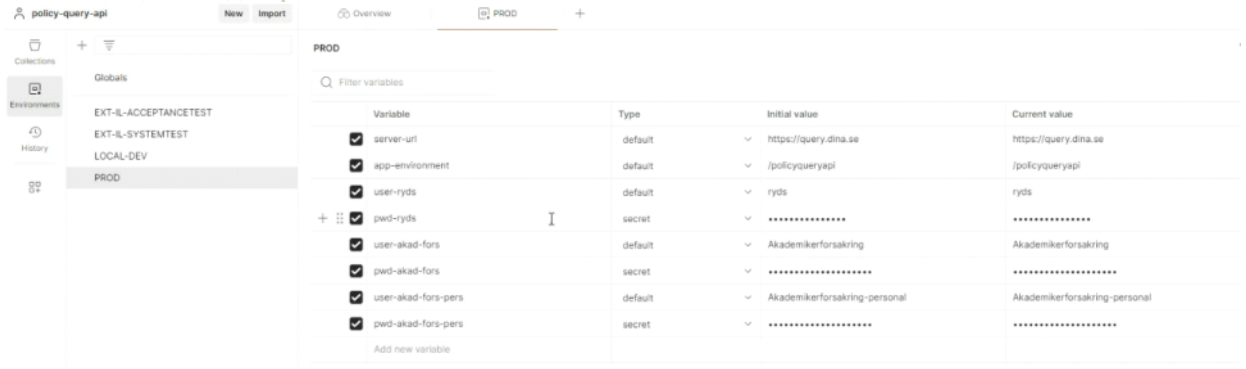
Sätt upp tia-miljöerna

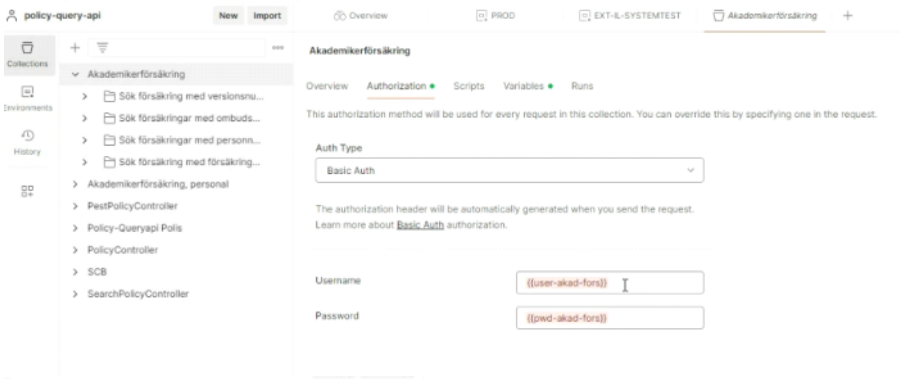


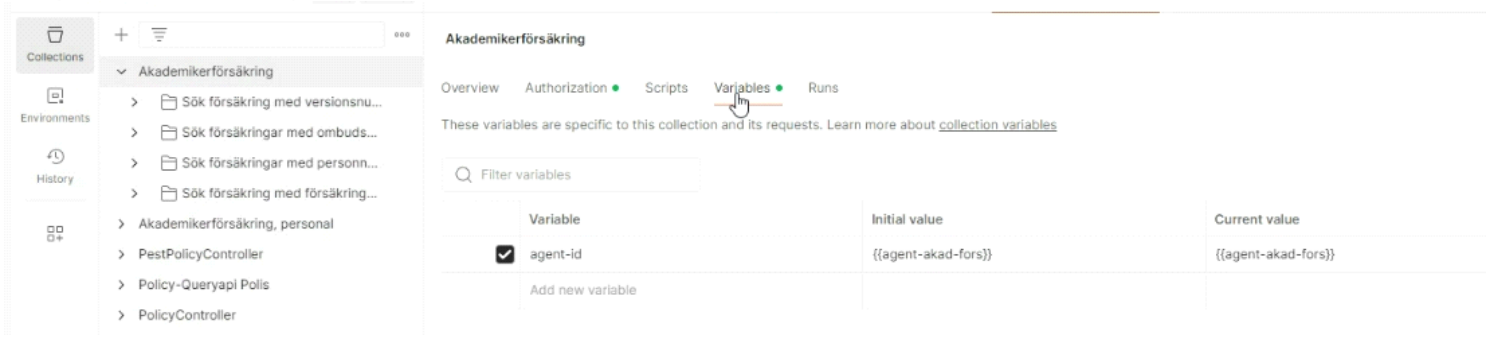
Ange rätt variabel:

Då står det så här i urlen:

Byt miljö genom att byta environment:







Swagger vs OpenAPI

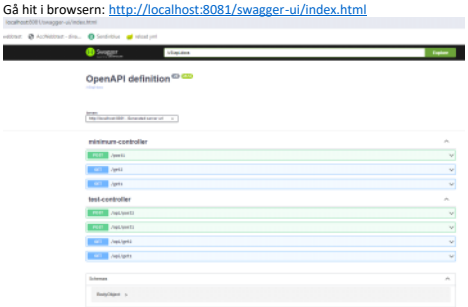
den 10 februari 2022 14:43

Se Swagger and Spring Boot Microservices - John Dobie (Titta mer på denna!)

För att lägga till stöd för swagger, behövs enbart denna dependency:

```
<dependency>
	<groupId>org.springdoc</groupId>
	<artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
	<version>2.1.0</version>
</dependency>
```

Gå hit i browsern: http://localhost:8081/swagger-ui/index.html



De annoteringar jag använt sedan tidigare fungerar med ovan dependency. Mer info för att lägga till lämpliga annoteringar, se mer på länken ovan.

Om jag klickar in på något, blir länken: http://localhost:8081/swagger-ui/index.html#/minimum-controller/post1

Both Swagger and OpenAPI are widely used to define and document RESTful APIs.

OpenAPI is built on the foundation of Swagger, incorporating Swagger Specification as its starting point.

Swagger and OpenAPI are closely related, but they are not the same thing. Swagger is the predecessor to OpenAPI and refers specifically to version 2.0 of the specification. When OpenAPI Specification 3.0 was released, the term "Swagger" was phased out, and the specification became known as OpenAPI. So, OpenAPI is the current version and the successor to Swagger.

Policy-query-api:
Skippa v1, för mina behov, vi har inga externa parter för apit.
Lokalt: http://localhost:8080/v1/swagger-ui.html

Confluence
Bas-funktionalitet
För att lägga till swagger behöver endast pom och yml-fil uppdateras.
Pom-filen

```
<dependency>
	<groupId>org.springdoc</groupId>
	<artifactId>springdoc-openapi-ui</artifactId>
	<version>1.5.9</version>
</dependency>
<dependency>
	<groupId>org.springdoc</groupId>
	<artifactId>springdoc-openapi-data-rest</artifactId>
	<version>1.5.9</version>
</dependency>
```

application yml-filen

```
springdoc:
  api-docs.path: /api-docs
  swagger-ui.path: /swagger-ui.html
```

Nu finns ett bas-gränssnitt som kan nås lokalt och i test. Ex:
http://localhost:8082/swagger-ui.html
http://10.39.60.182:8201/swagger-ui.html
Mer avancerad funktionalitet
För att få lite utförligare beskrivningar kan man lägga till diverse annoteringar
Ex:
@Operation för att beskriva endpointen
@Requestbody för beskrivning av requestBodyn
@Parameter för beskrivning av en ingående parameter
(Googla för exakt syntax, eller titta i någon av de nyare applikationerna, som policy-query-api, eller questionnaire-datastore-api.)

Backend tips and trix - TODO

den 20 maj 2024 08:54

CompletableFutures

```
asyncClaimSender.generateClaim(claimRegistrationData, policy, questionnair
        .thenApply(claimRegistrationInfo -> claimEmailHandler.sendThanksEm
                claimRegistrationInfo, questionnaireLayout, policyHolder))
        .thenApply(claimRegistrationInfo -> handleAttachedFilesAndQuestion
                claimRegistrationInfo, sessionStateId, paths, policy, poli
        .thenApply(claimRegistrationInfo -> generateCaseItem(
                claimRegistrationInfo, paths, policy, companyNumber, polic

        .thenAccept(claimRegistrationInfo -> claimEmailHandler.makeSureCla
```

AsyncClaimSender.java × ClaimService.java

```
import ...

7 usages  marost1 +4
@Slf4j
@RequiredArgsConstructor
@FieldDefaults(level = AccessLevel.PRIVATE, makeFinal = true)
public class AsyncClaimSender {
    Executor executor;
    ClaimNotificationEventBuilder claimNotificationEventBuilder;
    AuthorizationToken token;
    ClaimRepository claimRepository;

    4 usages  marost1 +2
    CompletableFuture<ClaimRegistrationData> generateClaim(
            ClaimRegistrationData claimRegistrationData, Policy poli
            LocalDateTime incidentDate, PolicyLineObject policyLineO
        return CompletableFuture.supplyAsync(() -> getClaimFromTia(
                claimRegistrationData,
                policy,
                questionnaireLayout,
                incidentDate,
                policyLineObject,
                nearbyClaimEvents), executor);
    }

    11 usages  marost1 +2
    ClaimRegistrationData getClaimFromTia(
```

Factory

```
QuestionnaireInformation.QuestionnaireAnswers questionnaireAnswers = QuestionnaireInformationFactory
        .getQuestionnaireInformation(questionnaireLayout.getQuestionnaireType())
        .makeQuestionnaireAnswers(questionnaireLayout, objectClaimInformation);
```

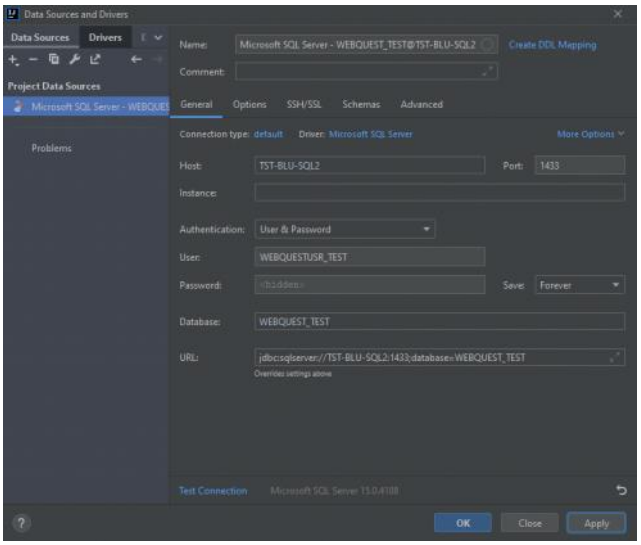
```
public class QuestionnaireInformationFactory {
    marost1 +2
    static public QuestionnaireInformation getQuestionnaireInformation(QuestionnaireType type) {
        switch (type) {
            case LIVING_CLAIM_THEFT:
                return new TheftQuestionnaireInformation();
            case LIVING_CLAIM_LOST:
                return new LostQuestionnaireInformation();
            case LIVING_CLAIM_BROKEN:
                return new BrokenQuestionnaireInformation();
            case LIVING_CLAIM_THUNDER:
                return new ThunderQuestionnaireInformation();
            case LIVING_CLAIM_NATURE_DAMAGE:
                return new NatureDamageQuestionnaireInformation();
            case LIVING_CLAIM_LEAKAGE_DAMAGE:
                return new LeakageDamageQuestionnaireInformation();
            case LIVING_CLAIM_LUGGAGE_DELAY:
                return new LuggageQuestionnaireInformation();
            case LIVING_CLAIM_MEANS_OF_TRANSPORTATION_DELAY:
                return new TransportQuestionnaireInformation();
            case MOTOR_CLAIM_SINGLE:
                return new MotorSingleQuestionnaireInformation();
            case MOTOR_CLAIM_COLLISION:
                return new MotorCollisionQuestionnaireInformation();
            case MOTOR_CLAIM_MACHINERY_BREAKDOWN:
                return new MotorMachineryBreakdownQuestionnaireInformation();
            default:
                throw new IllegalStateException("Claim type not implemented:" + type);
        }

    }
}
```

Db via Intellij

den 21 februari 2022 13:58

Questionnaire datastore databasen WEQBQUEST_TEST

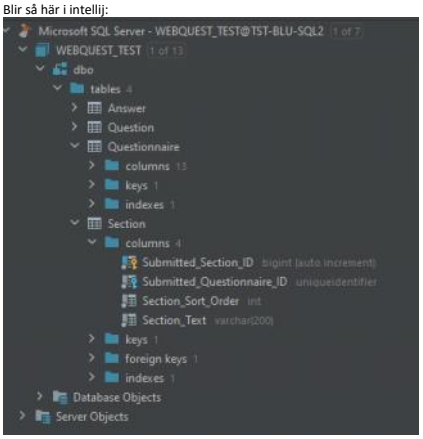
Inifrån Database fliken i IntelliJ:
Välj + -> Datasource -> MicrosoftSQL Server
Fyll i:
Server: TST-BLU-SQL2
Finns i keepass under Test/windows
Databas host: TST-BLU-SQL2
Port: 1433
Databasanvändare: WEBQUESTUSR_TEST.



Urlen ska anges i applikationsyml filen.

Dubbelklicka på en tabell för att se de 500 första
För att köra en sql fråga: New query console

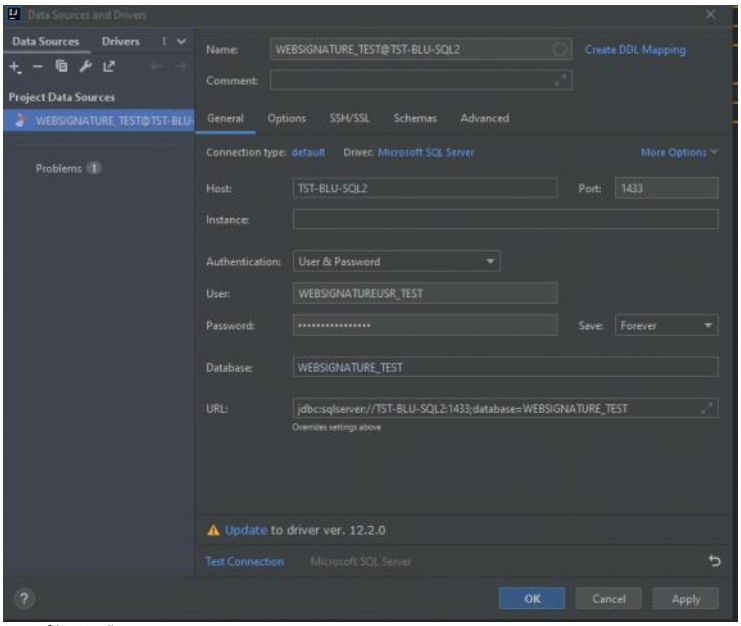
Blir så här i intellij:



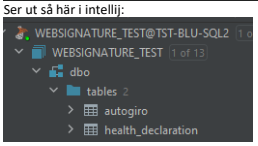
Websignature

jdbc:sqlserver://10.39.60.81:1433;database=WEBSIGNATURE_TEST;encrypt=false;
driver-class-name: com.microsoft.sqlserver.jdbc.SQLServerDriver
username: WEBSIGNATUREUSR_TEST

Men jag kopplade upp pss som för webquest => Bör kanske byta ut detgula??



Ser ut så här i intellij:



Maven tips

den 18 mars 2022 10:57

För att hitta senaste dependency versionen:

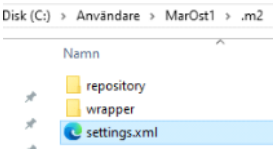
- För en viss dependency: Googla…
- För alla i ett projekt: skriv mvn versions:display-dependency-updates

Maven:

Skapa .m2-katalog under Användare

Lägg in settings.xml i .m2-katalogen:

Sätt MAVEN_HOME (bin-katalogen) och JAVA_HOME (tex: C:\Program Files\Java\jdk1.8.0_151) som miljövariabler under Användarvariabler samt i Path



Static final variabler

den 10 januari 2023 16:28

Hur döper man egentligen static final variabler? Varför gör jag dem i *lowerCase* och inte *UPPER_CASE_WITH_CAMEL_CASE*?

Summary

In conclusion, the convention you choose for `static final` variables is going to be your preference, but I strongly believe that the context of use should heavily weigh on your design decision. My personal recommendation would be to follow one of the two methodologies:

Methodology 1: Evaluate Context and Intention `[highly subjective; logical]`

- If it's a `private` variable that should be indistinguishable from a `private` instance variable, then name them the same. *all lowercase*
- If it's intention is to serve as a type of loose `enum` style block of `static` values, then name it as you would an `enum`. *pascal case: initial-cap each word*
- If it's intention is to define some publicly accessible, constant, and static property, then let it stand out by making it *all uppercase*

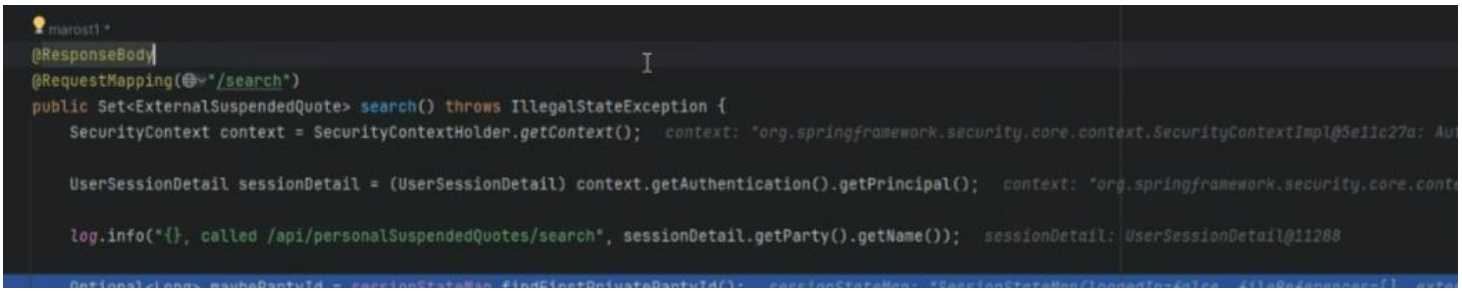
Methodology 2: Private vs Public `[objective; logical]`

Methodology 2 basically condenses its context into visibility, and leaves no room for interpretation.

- If it's `private` or `protected` then it should be *all lowercase.*
- If it's `public` or `package` then it should be *all uppercase.*

Security context

den 11 mars 2024 13:55



Tips för Commit (Amend, squash, cherry-pick)

den 5 juli 2019 08:34

Amend commit
Använd där det går.

Ctrl-k commit
Ctrl-shift-k push

Squash mha interactive rebase

1. Stå på den ***äldsta*** committen.
2. Högerklicka och välj "interactive rebase from here".
3. Välj "squash" på alla ***utom den översta***. Skriv ny kommentar.

Cherry-pick

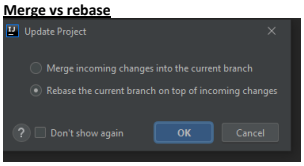
Stå i den branch du vill få in ändringarna i.
Högerklicka på committen och välj cherry-pick.
Nu blir det committat till min branch.

Att squasha committs med pushat kod (Gideon)

Markera de båda committsen, och squasha ihop med lämpligt meddelande.
Force-pusha detta så att det ändras på mastern (jag menar väl origin?!).

Tips: Om man tar två närliggande commits är detta enkelt, men om man tar en äldre commit kan det bli trassel/merge.

Merge vs rebase



Vi använder Rebase. Då tar man in remoten och lägger mina ändringar över.

Dvs vi gör egentligen git pull --rebase

git pull gör tvärt om, dvs lägger in och mergar alla saker från remoten ovanpå mina egna grejer. Men med rebase kommer remote branchens ändringar under mina.

De alternativ man får när man väljer gröna pilen "Git Update" inifrån IntelliJ är:

- Merge
- Rebase
- Branch Default

Jag har valt Rebase som default. Using stash.

(shelve är typ samma sak, men IntelliJs…)

Pss för amend

Använda yml-info i annoteringar

den 10 juni 2024 06:27

```
@Slf4j
@Component
@ConditionalOnProperty(name = "ecm-retry-task.active", havingValue = "true")
public class EcmRetryTask {
```

```
@Scheduled(cron = "..." )
// @Scheduled(cron = "${asr-retry-task.retry-frequency}" )
public void startScheduler() {
```

ObjectMapper och LocalDateTime

den 10 juni 2024 07:35

För att det ska fungera behöver man ta in en speciell object mapper.

```
public void saveASRFileForLaterRetry(ClaimRegistrationD
    String pathName = getPath() + fileName;

    log.info("Save ASR file to " + pathName);
    ObjectWriter writer = mapper.writer();
    writer.writeValue(new File(pathName), claimData);
}
```

Annars hade man kunnat göra typ så här:

```
public void saveASRFileForLaterRetry(ClaimRegistrationD
    String pathName = getPath() + fileName;

    log.info("Save ASR file to " + pathName);
    ObjectMapper mapper1 = new ObjectMapper();
    ObjectWriter writer = mapper1.writer();
    writer.writeValue(new File(pathName), claimData);
}
```

Men då sparas incident Date så här:

```
"incidentDate": {
    "dayOfMonth": 28,
    "dayOfWeek": "THURSDAY",
    "dayOfYear": 88,
    "year": 2024,
    "month": "MARCH",
    "monthValue": 3,
    "hour": 0,
    "minute": 0,
    "second": 0,
    "nano": 0,
    "chronology": {
        "id": "ISO",
        "calendarType": "iso8601"
    }
},
```

Eftersom incidentDate är en LocalDateTime.
Detta går inte att deserialisera med den vanliga object mappern.

I Application.java, speca därför en speciell Object mapper som hanterar detta:

```
@Bean("asrObjectMapper")
public ObjectMapper getAsrObjectMapper() {
    return  new Jackson2ObjectMapperBuilder()
            .dateFormat(new StdDateFormat())
            .featuresToDisable(SerializationFeature.WRITE_DATES_AS_TIMESTAMPS)
            .build();
}
```

Använd den för att få incident Date att sparas som en vanlig sträng istället:

```
"submittedFrom": "Web",
"incidentDate": "2024-04-18T00:00:00",
```

Måste då specas i bönorna som sparar och läser:

```
@Bean
AsrRetryService asrRetryService(
        @Qualifier("asrObjectMapper") ObjectMapper asrObjectMapper) {
    return new AsrRetryService(
            new File(useCaseProperties.getFileStorageArea().toString()),
            new RetryFileService(asrObjectMapper),
            asrClient(),
            mailService
    );
}
```

Dvs:

```
public class RetryFileService {
    private ObjectMapper mapper;
```

```
public class QuestionnaireFileService {
    private final Path fileStorageArea;
    private final ServiceProperties serviceProperties;
    private final ObjectMapper mapper;
```

Andra jackson tips

den 10 juni 2024 08:20

```java
@Value
@Builder
@Jacksonized
@JsonIgnoreProperties(ignoreUnknown = true)
public class CuckooSummaryDto {
    Info info;
    List<Signature> signatures;
    Target target;
}
```

```java
@Value
@Builder
@Jacksonized
public class File {
    String md5;
    String name;
    int size;
    String type;
}
```

ConstraintValidators

den 10 juni 2024 08:24

I restanrop, kan man speca egna validatorer:

```
public ExternalPolicyMotor getPolicyOptionsForCompanyMotor(
        @Age(min = 16, max = 200)
        @RequestParam("orgNr") CompanyRegistrationNumber orgNr,
        @DateWithinDaysRange(max = 200)
        @RequestParam("coverDate")
        @DateTimeFormat(iso = DateTimeFormat.ISO.DATE) LocalDate startDate,
        @RequestParam("vehicleRegNumber") VehicleRegistrationNumber vehicleRe
        @CheckExperiment
        @RequestParam("experimentId") String experimentId,
        @RequestParam("policyContentType") PolicyContentType policyContentTyp
        @RequestParam("agent") String agent,
        @RequestParam("ownerCompany") String ownerCompany) throws Integration
```

Ex 1 - en speciell validator

Constraint interface

```
@Target({ FIELD, METHOD, PARAMETER })
@Retention(RUNTIME)
@Constraint(validatedBy = DateWithinDaysRangeValidator.class)
@Documented
public @interface DateWithinDaysRange {
    Michael Svensson
    String message() default "";
    Michael Svensson
    Class<?>[] groups() default {};
    Michael Svensson
    Class<? extends Payload>[] payload() default {};
    Michael Svensson
    int min() default 0;
    Michael Svensson
    int max();
```

Validatorn:

```
public class DateWithinDaysRangeValidator implements ConstraintValidator<DateWithinDaysRange, LocalDate> {

    3 usages
    private DateWithinDaysRange range;

    Michael Svensson
    @Override
    public void initialize(DateWithinDaysRange constraint) { range = constraint; }

    Michael Svensson +1
    @Override
    public boolean isValid(LocalDate startDate, ConstraintValidatorContext context) {
        LocalDate firstValidStartDate = LocalDate.now().minusDays(range.min());
        LocalDate lastValidStartDate = LocalDate.now().plusDays(range.max());
        if (isDateNotWithinRange(startDate, firstValidStartDate, lastValidStartDate)) {
            HibernateConstraintValidatorContext hibernateContext = context.unwrap(HibernateConstraintValidatorContext.class);

            hibernateContext.disableDefaultConstraintViolation();
            hibernateContext.addExpressionVariable( s: "firstValidDate", firstValidStartDate) HibernateConstraintValidatorCo...
                    .addExpressionVariable( s: "lastValidDate", lastValidStartDate)
                    .buildConstraintViolationWithTemplate( s: "Startdatum måste vara mellan ${firstValidDate} och ${lastValidDate}")
                    .addConstraintViolation();

            return false;
        }
        return true;
    }
    1 usage  new *
    private static boolean isDateNotWithinRange(LocalDate candidate, LocalDate fromDate, LocalDate toDate) {
        return candidate.isBefore(fromDate) || candidate.isAfter(toDate);
    }
```

Ex 2 - via pattern

```
public void selectEmail(@RequestHeader(value="Sub-Session-Id") UUID webFlowSessionId, @EmailAddress @RequestBody String email) {
    WebFlowState webFlowState = webFlowStateMap.getWebFlowState(webFlowSessionId);
    webFlowState.setEmail(email);
}
```

```
2 usages  A marost
@Target({ FIELD, PARAMETER, ANNOTATION_TYPE })
@Retention(RUNTIME)
@Documented
@Constraint(validatedBy = {})
@NotNull
@Pattern(regexp = "^[a-zA-Z0-9!#$%&'*+\\/=?^_`{|}~-](
@ReportAsSingleViolation
public @interface EmailAddress {
    // The regexp is inspired from RFC5322, so we use
    A marost
    String message() default "Felaktig e-postadress";
    A marost
    Class<?>[] groups() default {};
    A marost
    Class<? extends Payload>[] payload() default {};
```

Här kan man ha lite olika annoteringar:

```
@Target({ FIELD, PARAMETER, ANNOTATION_TYPE })
@Retention(RUNTIME)
@Documented
@Constraint(validatedBy = {})
@NotNull
@Length(min = 1, max = 55)
@Pattern(regexp = "^[.\\p{L}0/60-9 ]+$")
@ReportAsSingleViolation
```

```
@Target({ FIELD, METHOD, PARAMETER, ANNOTATION_TYPE })
@Retention(RUNTIME)
@Documented
@Constraint(validatedBy = {})
@Length(max = 35, message = "Objekt id är för långt")
@Pattern(regexp = "^[A-Z0-9_]+$", message = "Objekt id innehåller ogiltiga tecken")
```

Caches

den 10 juni 2024 09:06

Detta är enbart en concurrentHashMap:

```
3 usages
private ConcurrentHashMap<Long, ExternalSweBoat> boatCache = new ConcurrentHashMap<>();
8 usages
```

```
2 usages
private final List<ExternalActivity> activityCache = new ArrayList<>();
1 usage
```

Databas/CrudRepository - TODO Gör egna exempel

den 14 juni 2024 07:15

```java
import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;

no usages  ▲ marost1
@Repository
public interface SectionRepository extends CrudRepository<DbSection, Integer> {
}
```

```java
@Repository
public interface QuestionnaireRepository extends CrudRepository<DbQuestionnaire, UUID> {
    7 usages  ▲ marost1
    List<DbQuestionnaire> findByClaimNo(Long claimNo);
}
```

```java
DbQuestionnaire savedQuestionnaire = questionnaireRepository.save(dbQuestionnaire);
```

```java
Optional<DbQuestionnaire> dbQuestionnaireOptional = questionnaireRepository.findById(uuid);
```

```java
allUuidsForClaimNo.forEach(uuid -> {
    questionnaireRepository.deleteById(uuid);
});
```

```java
return questionnaireRepository.count() > 0;
```

```java
List<DbQuestionnaire> allForClaimNo = questionnaireRepository.findByClaimNo(claimNo);
```

```java
@Entity
@Table(schema = "dbo", name="Questionnaire")
@Builder
@AllArgsConstructor
@NoArgsConstructor
@Getter
@Setter
@FieldDefaults(level = AccessLevel.PRIVATE)
public class DbQuestionnaire {
    @Id
    @Column(nullable = false, name = "Submitted_Questionnaire_ID")
    UUID submittedQuestionnaireId;

    @Column(nullable = false, name = "Party_ID")
    Long partyId;

    @Column(nullable = false, name = "Policy_No")
    Long policyNo;

    @Column(nullable = false, name = "Claim_No")
    Long claimNo;

    @Column(nullable = false, name = "Object_No")
    Long objectNo;

    @Column(nullable = false, name = "Social_Security_Number")
    String socialSecurityNo;

    @Column(nullable = false, name = "Incident_Date")
    String incidentDate;

    @Column(nullable = false, name = "Questionnaire_Type")
    @Enumerated(EnumType.STRING)
    QuestionnaireType questionnaireType;

    @Column(nullable = false, name = "Questionnaire_Version")
    int version;

    @Column(nullable = false, name = "Finalized")
    boolean finalized;

    @Column(nullable = false, name = "Area")
    @Enumerated(EnumType.STRING)
    QuestionnaireArea area;

    @Column(nullable = false, name = "Created_By")
    String createdBy;

    @Column(nullable = false, name = "Last_Update_Date")
    String lastUpdateDate;

    @OneToMany(mappedBy = "dbQuestionnaire", cascade = CascadeType.ALL)
    List<DbSection> sectionList;
}
```

```java
@Entity
@Table(schema = "dbo", name="Section")
@Builder
@AllArgsConstructor
@NoArgsConstructor
@Getter
@Setter
@FieldDefaults(level = AccessLevel.PRIVATE)
public class DbSection {

    @Id
    @Column(nullable = false, name = "Submitted_Section_ID")
    @GeneratedValue(strategy= GenerationType.IDENTITY)
    Long submittedSectionId;

    @ManyToOne
    @JoinColumn(nullable = false, name = "Submitted_Questionnaire_ID")
    DbQuestionnaire dbQuestionnaire;

    @Column(nullable = false, name = "Section_Sort_Order")
    Integer sectionSortOrder;

    @Column(nullable = false, name = "Section_Text")
    String sectionText;

    @OneToMany(mappedBy = "dbSection", cascade = CascadeType.ALL)
    List<DbQuestion> questionsList;
}
```

```java
@Entity
@Table(schema = "dbo", name="Answer")
@Builder
@AllArgsConstructor
@NoArgsConstructor
@Getter
@Setter
public class DbAnswer {
    @Id
    @Column(nullable = false, name = "Submitted_Answer_ID")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    Integer submittedAnswerId;

    @ManyToOne
    @JoinColumn(nullable = false, name="Submitted_Question_ID", referencedColumnName = "Submitted_Question_ID")
    DbQuestion dbQuestion;

    @Column(nullable = false, name = "Answer_ID") String answerId;
    @Column(name = "Answer_Label") String answerLabel;
    @Column(name = "Answer_Text") String answerText;
    @Column(name = "I01") String s01;
    @Column(name = "N01") String n01;
    @Column(name = "B01") Boolean b01;
    @Column(name = "D01") String d01;
    @Column(name = "T01") String t01;
}
```

```java
@Entity
@Table(schema = "dbo", name="Question")
@Builder
@AllArgsConstructor
@NoArgsConstructor
@Getter
@Setter
@FieldDefaults(level = AccessLevel.PRIVATE)
public class DbQuestion {
    @Id
    @Column(nullable = false, name = "Submitted_Question_ID")
    @GeneratedValue(strategy= GenerationType.IDENTITY)
    Long submittedQuestionId;

    @ManyToOne
    @JoinColumn(nullable = false, name = "Submitted_Section_ID", referencedColumnName = "Submitted_Section_ID")
    DbSection dbSection;

    @Column(nullable = false, name = "Question_ID")
    String questionId;

    @Column(nullable = false, name = "Question_Type")
    @Enumerated(EnumType.STRING)
    QuestionType questionType;

    @Column(nullable = false, name = "Question_Sort_Order")
    int questionSortOrder;

    @Column(name = "Question_Reference")
    String questionReference;

    @Column(name = "Unit")
    String unit;

    @Column(nullable = false, name = "Question_Text")
    String questionText;

    @OneToMany(mappedBy = "dbQuestion", cascade = CascadeType.ALL)
    List<DbAnswer> answersList;
}
```

Java 21

den 28 juni 2024 07:21

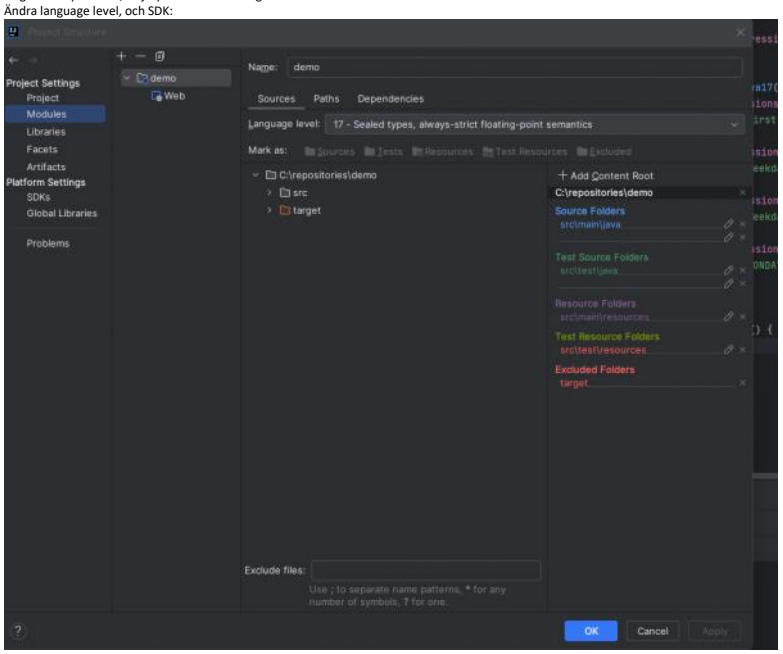
Language level

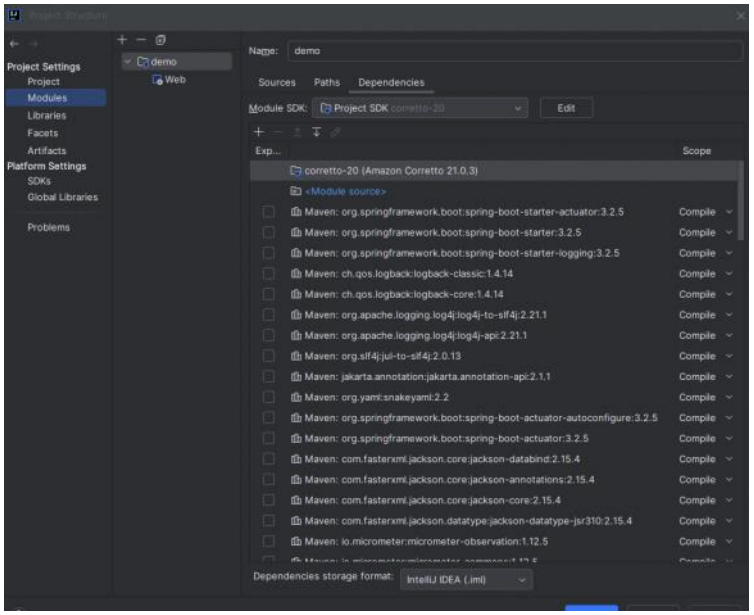
den 28 juni 2024 08:29

Kan inte köra mina unittester med java21 grejer:

```
C:\repositories\demo\src\main\java\com\example\demo\Java21Things\SwitchExpressions.java:48:18
java: patterns in switch statements are not supported in -source 17
  (use -source 21 or higher to enable patterns in switch statements)
```

Högerklicka på demo, välj Open ModuleSettings
Ändra language level, och SDK:



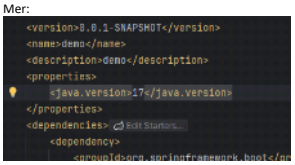


Här ser man vilken java man kör med:

OBS: Räckte inte att bara ändra detta för att demo skulle köra sina tester med java 21.

- Jag får inte kompileringsfel/varningar, så language level är 21
- Jag kan starta applikationen med java 21
- Varför kan jag inte köra unittester med java 21?

Mer:



Räckte inte ändå. Men när jag startade om datorn fungerade allt!

List.of() vs andra alternativ

den 24 juni 2024 07:23

Olika syntax beroende på 1 eller flera i listan

I unittester brukar jag använda Arrays.asList(1) och Arrays.asList(1, 2) för att få samma syntax i alla testfallen, men får då alltid varning på den första.
Om man använder List.of() får man ingen varning på varken List.of(1) och List.of(1,2) => Använd denna i testerna.

Immutable?

- Varken Arrays.asList() och List.of() går att ändra storlek på
- Arrays.asList() kan ändra innehåll mha set metod. Det går inte för List.of().
- Arrays.asList() har en specifik array i bakgrunden som man ändrar

Arrays.asList() är MODIFIABLE, uppbackad av en array.
List.of() är IMMUTABLE
ArrayList som är MUTABLE => Bör aldrig användas normalt sett…

Varningar i IDEt

Man får inga varningar i Arrays.asList att man kommer på ett exception. Det får man på båda fallen för List.of.

Läs mer här: https://medium.com/@mgm06bm/list-of-vs-arrays-aslist-7e2f7af64361

Enhanced Switch

den 28 juni 2024 08:24

https://medium.com/@javatechie/the-evolution-of-switch-statement-from-java-7-to-java-17-4b5eee8d29b7

Fram till och med Java 17

- Tidigare enbart switch statements, nu finns även Switch expressions. Dvs undviker subtila buggar med missade breaks.
- Varje sats har sitt eget scope, med arrowfunktion. Kan vid behov returnera mha yield.
- Man behöver bara definiera "default" för primitiver. För enums märker man om det behövs

I java 21

Pattern matching

- Göra olika saker beroende på vad det är för typ
- Kan mha scope göra sub-logik beroende på värde på det som skickas in. => Kan även skrivas som guarded pattern

```
public String conditionalResultsWithoutGuardedPatterns(Object o) {
    return switch (o) {
        case String s -> {
            if (!s.isEmpty()) {
                yield s + " is a string";
            } else {
                yield "string is empty";
            }
        }
        case ObjectWithDay day -> {
            if (day.getDay().equals(Days.MONDAY)) {
                yield "Its Monday";
            } else {
                yield day.getDay() + " is another day";
            }
        }
        default -> "Something else";
    };
}
```

Guarded patterns:

```
public String conditionalResultsWithGuardedPatterns(Object o) {  3 usages
    return switch (o) {
        case String s when !s.isEmpty() -> s + " is a string - guarded pattern";
        case ObjectWithDay day when day.getDay().equals(Days.TUESDAY) -> "Its Tuesday";
        case ObjectWithDay day -> day.getDay() + " is another day";
        default -> "Something else";
    };
}
```

Null cases

Kan man numera ha

```java
public String nullAsACase(Days day) {  no usages
    return switch (day) {
        case MONDAY -> {
            System.out.println("Inside case 1");
            yield  "MONDAY";
        }
        case TUESDAY, WEDNESDAY, THURSDAY, FRIDAY -> "Other weekday";
        case SATURDAY, SUNDAY -> "Weekend";
        case null -> "Its null";
    };
}
```

Raw string literals / template strings TODO

den 9 juli 2024 06:35

Se tex:
https://openjdk.org/jeps/326

Java functional programming - Amigoscode på Youtube

den 10 oktober 2024 08:08

Helenas länk: https://youtu.be/VRpHdSFWGPs

Grundläggande functional interfaces

den 10 oktober 2024 10:28

Imparative programming = for loopar etc, pre-java 8
Declarative programming = Streams with collections, functional programming, java 8->

Se dokumentation:
https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/function/package-summary.html

Kan använda functional interfaces för tex
Assignment, method invocation och cast contexts

```
// Assignment context
Predicate<String> p = String::isEmpty;

// Method invocation context
stream.filter(e -> e.getSize() > 10)...

// Cast context
stream.map((ToIntFunction) e -> e.getSize())...
```

Exempel på functional interfaces
Basic function shapes
Function<T,R> (unary function from T to R). Takes an input and produces a result
Consumer<T> (unary function from T to void) Takes an input and returns no result
Predicate<T> (unary function from T to boolean) Represents a predicate (boolean-valued function) of one argument
Supplier<T> (nullary function to R (menar väl T???)) Represents a supplier of results
Andra exempel
BiFunction<T,U,R> (binary function from T and U to R)

Binary function tar två argument
Unary function tar ett argument
Nullary function tar inga argument

Functions
Man kan anropa en Function genom att göra .apply.
Man kan chaina ihop funktioner med .apply och .andThen

```
        System.out.println(addBy1AndThenMultiplyBy10.apply( t: 4));
    }
    static Function<Integer, Integer> incrementByOneFunction = number -> number + 1;  2 usages
    static Function<Integer, Integer> multiplyBy10Function = number -> number * 10;  2 usages
    static Function<Integer, Integer> addBy1AndThenMultiplyBy10 = incrementByOneFunction.andThen(multiplyBy10Function);
```

Man kan även göra BiFunctions:

```
        System.out.println(incrementByOneAndMultiplyBiFunction.apply( t: 4,  u: 100));
    }


    static BiFunction<Integer, Integer, Integer>  incrementByOneAndMultiplyBiFunction =  1 usage
            (numberToIncrementByOne, numberToMultiplyBy) -> (numberToIncrementByOne + 1) * numberToMultiplyBy;
```

Functions: .apply
Kan chainas ihop med .andThen

Consumers: .accept
Kan chainas ihop med .andThen

Predicates: .test
Kan chainas ihop med .and, .or och .negate

Suppliers: .get

Consumers
Consumer använder .accept och .andThen:

```
    greetCustomerConsumer.accept(maria);
}

static Consumer<Customer> greetCustomerConsumer = customer ->  1 usage
        System.out.println("Hello " + customer.customerName +
                ", thanks for registering phone number " + customer.customerPhoneNumber);
```

Ex med BiConsumer:

```
    greetCustomerConsumerV2.accept(maria,  u: false);
}

static BiConsumer<Customer, Boolean> greetCustomerConsumerV2 = (customer, showPhoneNumber) ->
        System.out.println("Hello " + customer.customerName +
                ", thanks for registering phone number "
                + (showPhoneNumber ? customer.customerPhoneNumber : "****"));
```

Men hur och när skulle jag ha använt detta sätt att skriva? (Mer än möjligen i validatorer)
I strömmar används de ju inte "löst"...

Och det är också det han säger!

Predicates
Predicate använder .test, kan chainas ihop med .and och .or

```
        System.out.println(isPhoneNumberValidPredicate.test( t: "07000000000"));
        System.out.println(isPhoneNumberValidPredicate.test( t: "0700000000"));
        System.out.println(isPhoneNumberValidPredicate.test( t: "17000000000"));
    }
    static Predicate<String> isPhoneNumberValidPredicate = phoneNumber ->  3 usage
            phoneNumber.startsWith("07") && phoneNumber.length() == 11;
```

```
    System.out.println("combine predicates");
    System.out.println(isPhoneNumberValidPredicate.and(containsNumber3).test( t: "07003000000"));
    System.out.println(isPhoneNumberValidPredicate.and(containsNumber3).test( t: "0700300000"));
    System.out.println(isPhoneNumberValidPredicate.and(containsNumber3).test( t: "17003000000"));
    System.out.println(isPhoneNumberValidPredicate.and(containsNumber3).test( t: "07005000000"));
}

static Predicate<String> containsNumber3 = phoneNumber -> phoneNumber.contains("3");  4 usages
static Predicate<String> isPhoneNumberValidPredicate = phoneNumber ->  7 usages
        phoneNumber.startsWith("07") && phoneNumber.length() == 11;
```

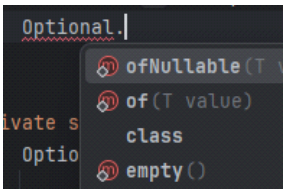
Detta borde jag verkligen ha utnyttjat i validator-situationer, eller?

Supplier

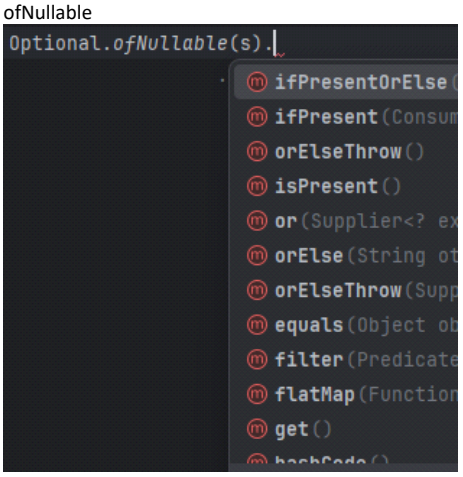
```
        System.out.println(getDBConnectionUrlSupplier.get());
    }

    static Supplier<String> getDBConnectionUrlSupplier = () -> "jdbc://localhost:5432/users";
```

Optionals

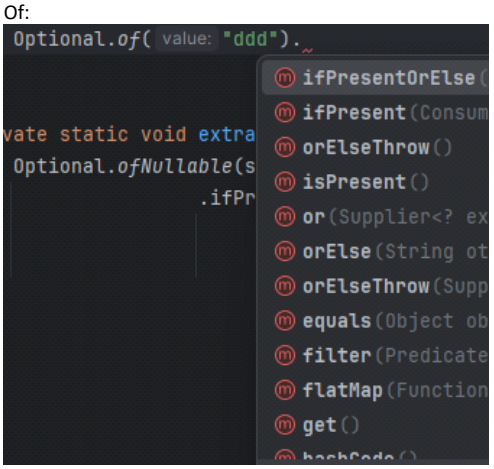
den 10 oktober 2024 10:31



ofNullable



Of:



Har typ samma metoder oavsett om det är Nullable eller inte.

Om man anropar en non-nullable optional med null får man en NP när man kör. Men, hur ska man komma till else fallet??

```java
public static void main(String[] args) {
    nullableOptional( s: "a@bb.cc");
    nullableOptional( s: null);
    nonNullableOptional( s: "a@bb.cc");
    nonNullableOptional( s: null); // Ger NP när man kör
}

private static void nullableOptional(String s) {  2 usages
    Optional.ofNullable(s)
                    .ifPresentOrElse(
                            email -> System.out.println("Sending email to " + email),
                            () -> System.out.println("Cannot send email"));
}

private static void nonNullableOptional(String s) {  2 usages
    Optional.of(s)
            .ifPresentOrElse(
                    email -> System.out.println("Sending email to " + email),
                    () -> System.out.println("Cannot send email")); // Hur kan man någonsin komma hit??
}
```

Knasigt exempel pga strings? Se hellre min egen genomgång av Optionals

Higher order function

den 11 oktober 2024 10:46

Definition:

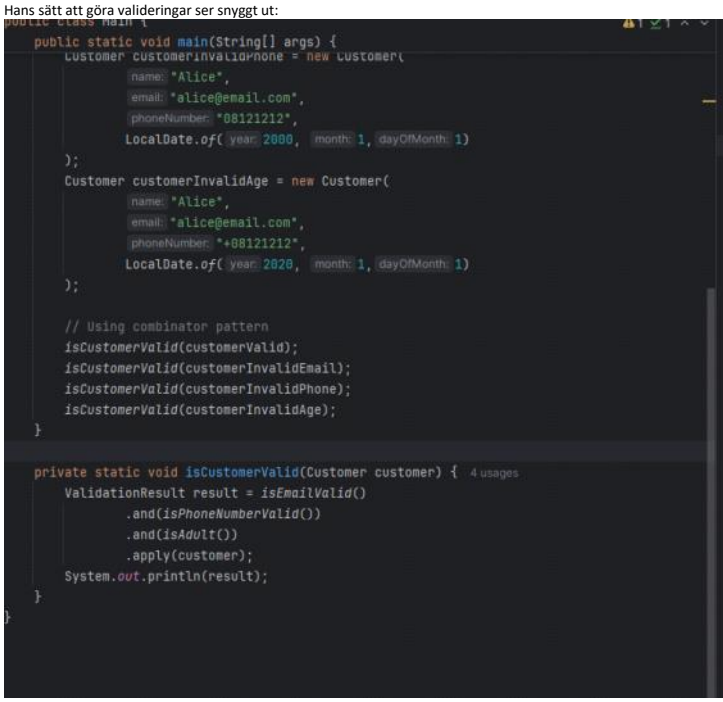
- Returnerar en funktion som resultat - tex combinatory pattern exemplet
- Tar en/flera funktioner som argument - tex callback exemplet

Combinator pattern - snyggare valideringar

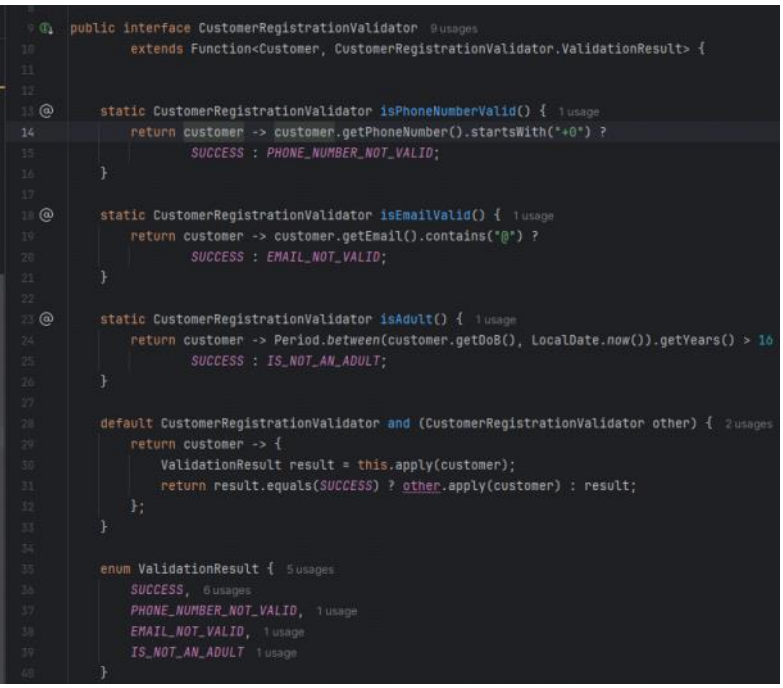
den 10 oktober 2024 11:52

Validatorn är ett interface av typen Functions, och metoderna returnerar alltså Functions.

Hans sätt att göra valideringar ser snyggt ut:



```
public class Main {
    public static void main(String[] args) {
        Customer customerInvalidPhone = new Customer(
                name: "Alice",
                email: "alice@email.com",
                phoneNumber: "08121212",
                LocalDate.of( year: 2000,  month: 1, dayOfMonth: 1)
        );
        Customer customerInvalidAge = new Customer(
                name: "Alice",
                email: "alice@email.com",
                phoneNumber: "+08121212",
                LocalDate.of( year: 2020,  month: 1, dayOfMonth: 1)
        );

        // Using combinator pattern
        isCustomerValid(customerValid);
        isCustomerValid(customerInvalidEmail);
        isCustomerValid(customerInvalidPhone);
        isCustomerValid(customerInvalidAge);
    }

    private static void isCustomerValid(Customer customer) {  4 usages
        ValidationResult result = isEmailValid()
                .and(isPhoneNumberValid())
                .and(isAdult())
                .apply(customer);
        System.out.println(result);
    }
}
```



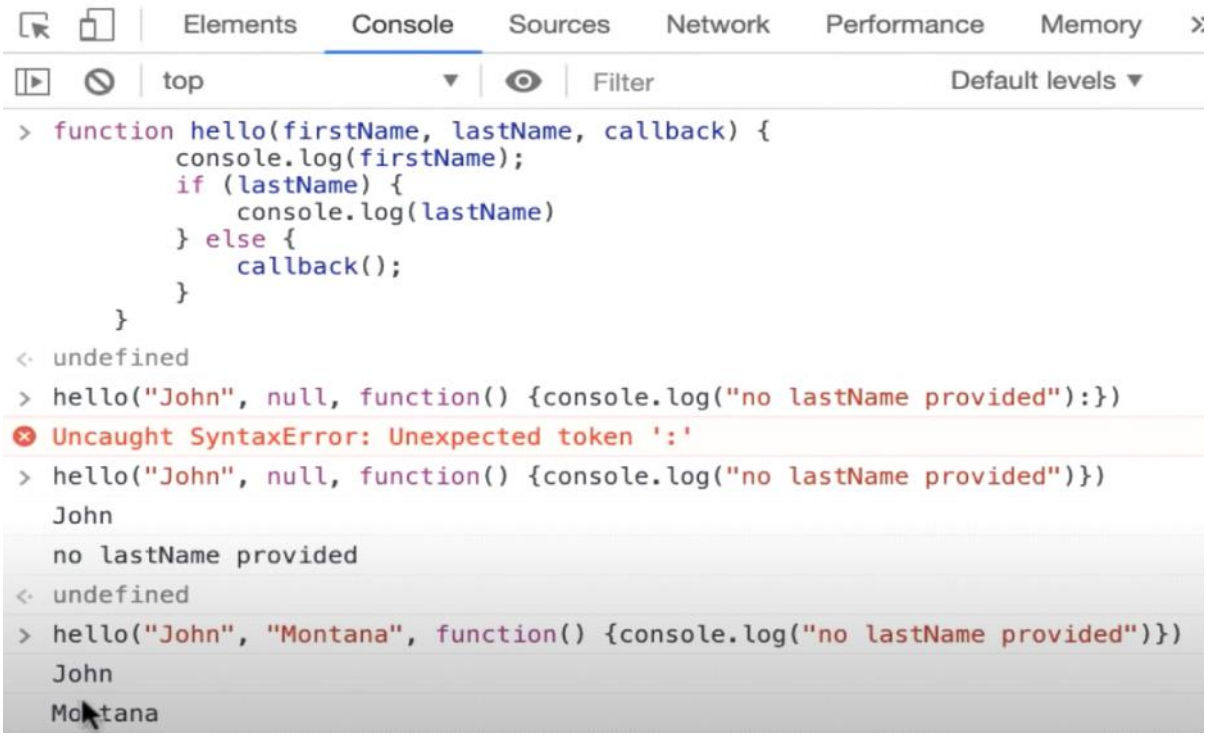
```
public interface CustomerRegistrationValidator  9 usages
        extends Function<Customer, CustomerRegistrationValidator.ValidationResult> {


    static CustomerRegistrationValidator isPhoneNumberValid() {  1 usage
        return customer -> customer.getPhoneNumber().startsWith("+0") ?
                SUCCESS : PHONE_NUMBER_NOT_VALID;
    }

    static CustomerRegistrationValidator isEmailValid() {  1 usage
        return customer -> customer.getEmail().contains("@") ?
                SUCCESS : EMAIL_NOT_VALID;
    }

    static CustomerRegistrationValidator isAdult() {  1 usage
        return customer -> Period.between(customer.getDoB(), LocalDate.now()).getYears() > 16 ?
                SUCCESS : IS_NOT_AN_ADULT;
    }

    default CustomerRegistrationValidator and (CustomerRegistrationValidator other) {  2 usages
        return customer -> {
            ValidationResult result = this.apply(customer);
            return result.equals(SUCCESS) ? other.apply(customer) : result;
        };
    }

    enum ValidationResult {  5 usages
        SUCCESS,  6 usages
        PHONE_NUMBER_NOT_VALID,  1 usage
        EMAIL_NOT_VALID,  1 usage
        IS_NOT_AN_ADULT  1 usage
    }
```

Men, fattar faktiskt inte alls hur koden fungerar…

- Interface men med metoder som har implementation
- Metoderna returnerar ju en enum, hur kan då retur typen vara CustomerRegistrationValidator?
- Hur fungerar egentligen default metoder?

Callbacks

den 11 oktober 2024 09:55



Man har ofta callbacks i javascript och så här gör man motsvarande i java:

```java
public class Main {
    public static void main(String[] args) {
        hello( firstName: "John1",  lastName: "Montana1",  callback: null);
        hello( firstName: "John2",  lastName: "Montana2", value -> System.out.println("lastName not provided for " + value));
        hello( firstName: "John3",  lastName: null, value -> System.out.println("lastName not provided for " + value));
    }

    static void hello(String firstName, String lastName, Consumer<String> callback) {  3 usages
        System.out.println(firstName);
        if (lastName != null) {
            System.out.println(lastName);
        } else {
            callback.accept(firstName);
        }
    }
}
```

```
John1
Montana1
John2
Montana2
John3
lastName not provided for John3
```

Alt:

```
    hello2( firstName: "John4",  lastName: null, () -> System.out.println("lastName not provided"));
}

static void hello2(String firstName, String lastName, Runnable callback) {  1 usage
    System.out.println(firstName);
    if (lastName != null) {
        System.out.println(lastName);
    } else {
        callback.run();
    }
}
```

Jplanes - mer Lambda validators with Java 8

den 11 oktober 2024 11:01

Helenas länk: https://medium.com/@jplanes/lambda-validations-with-java-8-86aa8143bd9f

Samma princip som det som förklaras i Amigoscode, men det är fortfarande rätt obegripligt, även om det "blir snyggt". Känns som liknande de dsler vi skrev på sjm , dvs returnerar ut funktion/sig själv- flyttat som går att jobba vidare på.

Collections, Optionals and streams

den 15 oktober 2024 07:56

Tips från IDEt:

```
List<Person> females2 = people.stream()
        .filter(person -> FEMALE.equals(person.gender))
        .collect(Collectors.toList());
females2.forEach(System.out::println);
```

Collect(Collectors.toList() Kan numera ersättas med toList():

```
List<Person> females2 = people.stream()
        .filter(person -> FEMALE.equals(person.gender))
        .toList();
females2.forEach(System.out::println);
```

Tankar från i våras

den 20 maj 2024 08:47

Reverse (Den här förstår jag inte hur jag ska använda)

```
Collections.reverse(onePolicyVersionPerUniquePolicyNo);
```

UnmodifiableMap

```
private static Map<String, String> initializeDocumentTypeIdDescriptionResolveLib(){
    Map<String, String> titleMap = new HashMap();

    titleMap.put("R2D01", "Försäkringsbrev fordon");
    //...

    return Collections.unmodifiableMap(new HashMap<>(titleMap));
}
```

Matcher med regex:

```
public static Pattern FILE_REFERENCE_PATTERN = Pattern.compile( regex: "^[A-F0-9]{5,50}$");
```

```
1 usage   gidodu1
private boolean verifyFileReferenceSyntax(String fileReference) {
    return FILE_REFERENCE_PATTERN.matcher(fileReference).matches();
}
```

Flatmaps vs maps

den 21 oktober 2024 07:36

Maps vs flatmaps - varför använder jag aldrig flatmaps? Har jag behövt det?

- Optionals
- Streams

Optionals

```
// Optionals
public static Optional<String> methodReturningOptional(String str) {  4 usages
    // complicated method returning an optional:
    return Optional.ofNullable(str)
            .map(String::toUpperCase);
}

public static Optional<Optional<String>> nestedStructureWithMapInOptional(String str) {  2 usages
    return Optional.ofNullable(str)
            .map(MapsAndFlatMaps::methodReturningOptional); // .map(s -> methodReturningOptional(s));
}

public static Optional<String> nestedStructureWithFlatMapInOptional(String str) {  2 usages
    return Optional.ofNullable(str)
            .flatMap(MapsAndFlatMaps::methodReturningOptional); // .map(s -> methodReturningOptional(s));
}
```

Streams

```
public static List<String> moreComplexStreamExample(String param1, String param2, String param3, String param4) {
    List<String> stringsOfUpperCase1 = makeListAndConvertToUppercase(param1, param2);
    List<String> stringsOfUpperCase2 = makeListAndConvertToUppercase(param3, param4);
    List<List<String>> aNestedList = createANestedList(stringsOfUpperCase1, stringsOfUpperCase2);

    //Gör nu ännu en nivå av nästling:
    List<List<List<String>>> aNestedList1 = List.of(aNestedList);

    return aNestedList1.stream()
            .flatMap(Collection::stream) // flat map nästlar bara upp en nivå i taget!
            .flatMap(Collection::stream) // flat map nästlar bara upp en nivå i taget!
            .toList();
}
```

Gamla exempel:

```
public Set<ExternalClaim> getClaimsForAllCustomers() {
    UserSessionDetail userSessionDetail = (UserSessionDetail) Se

    return userSessionDetail.getParties().values().stream() Stre
            .flatMap(Collection::stream) Stream<Party>
            .flatMap(party -> getClaimsForCustomer(party.getPart
            .collect(Collectors.toSet());
}
```

```java
private Stream<Policy> getAllPolicies() {
    return Stream.of(
            sessionStateMap.getPolicies(),
            sessionStateMap.getFuturePolicies(),
            sessionStateMap.getQuotes(),
            sessionStateMap.getCustomerAgreements()
    ).flatMap(Collection::stream);
}
```

```java
return (Person) customerInfo.getParties(registrationNumbe
        .stream() Stream<List<...>>
        .flatMap(Collection::stream) Stream<Party>
        .filter(party -> party instanceof Person)
        .findFirst() Optional<Party>
        .orElseThrow(() -> new IllegalStateException("No
```

Comparators

den 21 oktober 2024 07:36

Comparators

```
public static List<Policy> getAllPoliciesWithPolicyNosInDescendingOrder(List<Policy> policies) {  1 usage
    return policies.stream()
            .sorted(Comparator.comparing(Policy::getPolicyNo).reversed())
            .toList();
}

public static List<Policy> getAllPoliciesWithPolicyNosInDescendingOrder2(List<Policy> policies) {  1 usage
    return policies.stream()
            .sorted(Collections.reverseOrder(Comparator.comparing(Policy::getPolicyNo)))
            .toList();
}
```

Max:

```
public static Policy getPolicyWithHighestPolicyNo2(List<Policy> policies) {  1 usag
    return policies.stream()  Stream<Policy>
            .max(Comparator.comparing(Policy::getPolicyNo))  Optional<Policy>
            .orElseThrow(() -> new IllegalArgumentException("no valid policy"));
}
```

Gamla exempel:

```
public Optional<PolicyDocument> getPolicyDocumentByType(  7 usages  gidodu1
        DinaJsonWebToken defaultAuthorizationToken
) {
    List<PolicyDocument> policyDocuments = this.getPolicyDocuments(
            partyId,
            currentPolicy.getPolicyNo(),
            defaultAuthorizationToken
    ).stream()
            .filter(
                    policyDocument -> filterOnDocumentTypeIds.contains(policyDocument.getDocumentTypeId()))
            .sorted(Comparator.comparing(PolicyDocument::getPolicySeqNo).reversed())
            .collect(Collectors.toList());

    return getPolicyDocumentByNewestSequence(currentPolicy, policyDocuments).stream()
            .max(Comparator.comparing(PolicyDocument::getPolicySeqNo));
}

public static long getNewestPolicySequence(Policy policy) {  3 usages  gidodu1
    // preference on newest policy-seq to reflect the newest changes
    return Optional.ofNullable(policy.getNextPolicySeqNo())
            .orElse(policy.getPolicySeqNo());
}
```

Collectors (groupingBy, partitioningBy etc)

den 21 oktober 2024 07:37

Collectors

- GroupingBy
- PartitioningBy
- toList(), toSet() etc

```
public static Map<Result, List<ResultObject>> groupByResult(List<ResultObject> results) {  2 usages
    return results.stream()
            .collect(Collectors.groupingBy(ResultObject::getResult));
}

public static Map<Boolean, List<ResultObject>> partitioningByResult(List<ResultObject> results) {
    return results.stream()
            .collect(Collectors.partitioningBy(r -> r.getResult().equals(Result.SUCCESS)));
}
```

Gamla exempel:

```
@   public List<BinaryFile> resolveFileAnalysisTasks(  no usages  new *
            ClaimRegistrationData claimRegistrationData,
            List<CompletableFuture<AntivirusResult>> fileAnalysisTasks
    ) {

        Map<Status, List<AntivirusResult>> fileAnalysisResult = fileAnalysisTasks.stream() Stream<CompletableFuture<...>>
                .map(CompletableFuture::join) Stream<AntivirusResult>
                .collect(Collectors.
                        groupingBy(
                                result -> result.isPassed() ? Status.PASSED : Status.FAILED)
                );

        List<AntivirusResult> filesWithMaliciousBehavior = fileAnalysisResult.get(Status.FAILED);

        if (!filesWithMaliciousBehavior.isEmpty()) {
            log.error(
                    "Unexpected result from antivirus analysis - no results received for {} files, although we know there should be.",
                    fileAnalysisResult.size()
            );

            claimRegistrationData.setFilesWithVirus(
                    filesWithMaliciousBehavior.size());


            claimRegistrationData.setVirusCheckFailure(true);
        }
```

```
Map<Long, List<Policy>> allVersionsPerUniquePolicyNo = allPolicyVersions.stream()
        .collect(Collectors.groupingBy(Policy::getPolicyNo));
```

Optionals

den 21 oktober 2024 07:54

För att returnera ett defaultvärde om Null:

```
public static Integer getPolicyNoOrDefault(Policy policy) {  2 usages
    return Optional.ofNullable(policy.getPolicyNo()).orElse( other: 0);
}

public static Integer getPolicyNoOrDefaultWithoutOptionals(Policy policy) {  1 us
    // så här skulle jag skrivit...
    return policy.getPolicyNo() != null ? policy.getPolicyNo() : 0;
}
```

Gammalt exempel:

```
public Optional<PolicyDocument> getPolicyDocumentByType(  7 usages  ▲ gidodul
        DinaJsonWebToken defaultAuthorizationToken
) {
    List<PolicyDocument> policyDocuments = this.getPolicyDocuments(
            partyId,
            currentPolicy.getPolicyNo(),
            defaultAuthorizationToken
    ).stream()
            .filter(
                    policyDocument -> filterOnDocumentTypeIds.contains(policyDocument.getDocumentTypeId()))
            .sorted(Comparator.comparing(PolicyDocument::getPolicySeqNo).reversed())
            .collect(Collectors.toList());

    return getPolicyDocumentByNewestSequence(currentPolicy, policyDocuments).stream()
            .max(Comparator.comparing(PolicyDocument::getPolicySeqNo));
}

public static long getNewestPolicySequence(Policy policy) {  3 usages  ▲ gidodul
    // preference on newest policy-seq to reflect the newest changes
    return Optional.ofNullable(policy.getNextPolicySeqNo())
            .orElse(policy.getPolicySeqNo());
}
```

I strömmar, när man får tillbaka en optional, eller tom

Alt 1 hantering av en eventuell Optional, både då den finns och den inte finns:

```
Stream.of(list1, list2) Stream<List<...>>
        .flatMap(Collection::stream) Stream<Policy>
        .map(Policy::getPolicyNo) Stream<Integer>
        .filter(p -> p != null && p > 3)
        .findFirst() Optional<Integer>
        .ifPresent(p -> System.out.println("3 - there are at least one policyNo above 3: " + p));

// Om man vill skriva ut något även om det inte finns:
Stream.of(list1, list2) Stream<List<...>>
        .flatMap(Collection::stream) Stream<Policy>
        .map(Policy::getPolicyNo) Stream<Integer>
        .filter(p -> p != null && p > 3)
        .findFirst() Optional<Integer>
        .ifPresentOrElse( p-> System.out.println("4 - there are at least one policyNo above 3: " + p),
                () -> System.out.println("There are no policyNos above 3"));

// Det består av två delar:
Consumer<Integer> consumer = p -> System.out.println("4 - there are at least one policyNo above 3: " + p);
Runnable runnable = () -> System.out.println("There are no policyNos above 3");
Stream.of(list1, list2) Stream<List<...>>
        .flatMap(Collection::stream) Stream<Policy>
        .map(Policy::getPolicyNo) Stream<Integer>
        .filter(p -> p != null && p > 3)
        .findFirst() Optional<Integer>
        .ifPresentOrElse(consumer, runnable);
```

På riktigt vill man troligen först returnera Optional, och sedan hantera den, typ:

```
Optional<Integer> firstOptional = Stream.of(list1, list2) Stream<List<...>>
        .flatMap(Collection::stream) Stream<Policy>
        .map(Policy::getPolicyNo) Stream<Integer>
        .filter(Objects::nonNull)
        .filter(p -> p > 3)
        .findFirst();

// Om man vill skriva ut något även om det inte finns:
firstOptional.ifPresentOrElse(
        p-> System.out.println("4 - there are at least one policyNo above 3: " + p),
        () -> System.out.println("There are no policyNos above 3"));
```

Gammalt exempel:

```
cuckooTaskDto
        .map(CuckooTaskDto::getTask) Optional<Task>
        .map(Task::getErrors) Optional<List<...>>
        .ifPresent(errors -> {
            if (!errors.isEmpty()) {
                log.warn("response from cuckoo get task endpoint contains error: {}", errors);
            }
        });
```

Om man vill kasta exception när det inte finns någon Optional:

```
public void validateThereAreCertainPolicyNosOrElseThrowException(List<Policy> list1, List<Policy> list
    Optional<Integer> firstOptional = Stream.of(list1, list2) Stream<List<...>>
            .flatMap(Collection::stream) Stream<Policy>
            .map(Policy::getPolicyNo) Stream<Integer>
            .filter(Objects::nonNull)
            .filter(p -> p > 3)
            .findFirst();

    firstOptional.orElseThrow(() -> new IllegalArgumentException("There are no policyNos above 3"));
}
```

Gammalt exempel:

```
return cuckooTaskDto
        .orElseThrow(
                () -> new CuckooClientException(String.format("unable to fetch taskId: %d", taskId)
                )
        );
```

EnumSet.of

den 21 oktober 2024 07:59

Man kan filtrera ut subset av en enum genom EnumSet.of

```
public static List<DALPolicy> removeUnwantedObjectTypes(List<DALPolicy> policies) {  1 usage

    return policies.stream()
            .filter(p -> EnumSet.of(PRIVATE_MAIN_BUILDING, PRIVATE_PROPERTY_PERSONAL).contains(p.getObjectType()))
            .toList();

}
```

```
// Hur verifiera resultatet på snyggt vis?

// Partitioning by det förbjudna
List<DALPolicy> forbiddenPolicies = result.stream() Stream<DALPolicy>
        .collect(Collectors.partitioningBy(p -> p.getObjectType().equals(ObjectType.PRIVATE_SMALL_BUILDING))) Map<Boo
        .get(true);
assertEquals( expected: 0, forbiddenPolicies.size());

// Gruppera på alla ObjectTypes och förvänta sig att det förbjudna är null
Map<ObjectType, List<DALPolicy>> sortResultByObjectType = result.stream()
        .collect(Collectors.groupingBy(DALPolicy::getObjectType));
assertNull(sortResultByObjectType.get(ObjectType.PRIVATE_SMALL_BUILDING));

// Hämta ut den förbjudna och om den finns, logga
Optional<DALPolicy> forbiddenPolicyOptional = result.stream()
        .filter(p -> p.getObjectType().equals(ObjectType.PRIVATE_SMALL_BUILDING)).findAny();
forbiddenPolicyOptional
        .ifPresent(p -> System.out.println("Om fel resultat, är följande policy invalid: " + p.getPolicyNo() ));

// Hämta ut den förbjudna och logga även om allt är bra
Optional<DALPolicy> forbiddenPolicyOptional2 = result.stream()
        .filter(p -> p.getObjectType().equals(ObjectType.PRIVATE_SMALL_BUILDING)).findAny();
forbiddenPolicyOptional2
        .ifPresentOrElse(p -> System.out.println("Om fel resultat, är följande policy invalid: " + p.getPolicyNo()),
                () -> System.out.println("Everything is fine!"));
```

Gammalt exempel:

```
@Override
public List<AttributeOption> selectArea(PolicyOption policyOption, String value) {
    List<AttributeOption> areas = policyOption.getPolicyLineOptions().stream() Stream<PolicyLineOption>
            .map(PolicyLineOption::getPolicyLineObjectOptions) Stream<List<...>>
            .flatMap(Collection::stream) Stream<PolicyLineObjectOption>
            .filter(p -> EnumSet.of(PRIVATE_PROPERTY_PERSONAL, PRIVATE_MAIN_BUILDING).contains(p.getObjectType()))
            .map(PolicyLineObjectOption::getAttributeOptions) Stream<List<...>>
            .flatMap(Collection::stream) Stream<AttributeOption>
            .filter(attributeOption -> attributeOption.getName().equals("AREA1"))
            .collect(Collectors.toList());

    areas.forEach(attributeOption -> attributeOption.createOrReplaceItemsWithNewValue(value));

    return areas;
}
```

Predikat

den 22 oktober 2024 07:06

AnyMatch, AllMatch, NoneMatch tar predikat:

```
public class Predicates {
    public static boolean anyMatch(List<ResultObject> results) {
        return results.stream()
                .anyMatch(r -> r.getResult().equals(Result.SUCCESS));
    }
    public static boolean allMatch(List<ResultObject> results) {
        return results.stream()
                .allMatch(r -> r.getResult().equals(Result.SUCCESS));
    }
    public static boolean noneMatch(List<ResultObject> results) {
        return results.stream()
                .noneMatch(r -> r.getResult().equals(Result.SUCCESS));
    }
}
```

Gammalt exempel:

```
private final static Predicate<ClaimCase> claimNotOpenedInErrorAndClosed = (claimCase) -> !OPENED_IN_ERROR_AND_CLOSED.equals(claimCase.getStatus());
```

```
private List<ClaimEvent> getRelevantClaimEvents(List<ClaimEvent> claimEvents) {
        return claimEvents.stream()
                .filter(claimEvent -> !claimEvent.getIncidentDate().toLocalDate().isBefore(STARTDATE_FOR_CLAIMS_OVERVIEW))
                .filter(claimEvent -> claimEvent.getClaimCases().stream()
                        .anyMatch(claimNotOpenedInErrorAndClosed))
                .filter(claimEvent -> !ClaimEventType.UNKNOWN.equals(claimEvent.getClaimEventType()))
                .collect(Collectors.toList());
}
```

Opterationer tillgängliga på Predicate - kan nästlas ihop:

negate(), and() och or():

```
public static List<ResultObject> getAllFailuresUsingNegate(List<ResultObject> results) {
    Predicate<ResultObject> predicate = resultObject -> Result.SUCCESS.equals(resultObject.getResult());
    return results.stream()
            .filter(predicate.negate())
            .toList();
}

public static List<ResultObjectWithReason> getAllFailuresUsingAnd(List<ResultObjectWithReason> results) {
    Predicate<ResultObjectWithReason> predicate1 = resultObject -> Result.FAILURE.equals(resultObject.getResult());
    Predicate<ResultObjectWithReason> predicate2 = resultObject -> resultObject.getReason().contains("error");
    return results.stream()
            .filter(predicate1.and(predicate2))
            .toList();
}

public static List<ResultObjectWithReason> getAllFailuresUsingOr(List<ResultObjectWithReason> results) {
    Predicate<ResultObjectWithReason> predicate1 = resultObject -> Result.FAILURE.equals(resultObject.getResult());
    Predicate<ResultObjectWithReason> predicate2 = resultObject -> resultObject.getReason().contains("error");
    Predicate<ResultObjectWithReason> predicate3 = resultObject -> resultObject.getReason().contains("not");
    return results.stream()
            .filter(predicate1.and(predicate2.or(predicate3)))
              .filter(predicate1.and(predicate2).or(predicate3)) // Även detta fungerar - hur kommer det sig?
            .toList();
}
```

Gamla exempel:

```
private Predicate<Policy> matchesCompanyPolicies() {
    return policy -> policiesService.DISALLOWED_COMPANY_POLICY_TYPES.contains(policy.getPolicyType());
}
```

```
1 usage   marost1
private List<Policy> filterDisallowedPolicies(Collection<Policy> policies) {
    // Negate means that we do the opposite of the predicate
    return policies.stream().filter(matchesCompanyPolicies().negate()).collect(Collectors.toList());
}
```

```
Predicate<Policy> policyFilterPredicate = isPolicyStatusPredicate(PolicyStatus.SUSPENDED_QUOTE)
        .and(isProductType(PolicyType.PERSONAL))
        .and(isNewerThanDaysOld(90L))
        .and(isDal().negate());
```

```
static Predicate<Policy> isAwaitingCustomerAgreement() {
    return isDal(). // This is the important check to do
            and(isDalCreationDateSet() // This is to double-c
                    .and(idDalConversionDateSet().negate()));
}
```

```
private void updateLengthValidationStatus(List<ExternalSweBoat> boats) {
    boats.stream()
            .filter(lengthLowerThanAllowedValue.or(lengthHigherThanAllowedValue))
            .forEach(boat -> boat.invalidBoatLength = true);
}
```

```
1 usage
private final static Predicate<ExternalSweBoat> lengthHigherThanAllowedValue = boat -> boat.length > BOAT_LENGTH_MAX_VALUE.doubleValue();
```

Ett predikat kan vara komplext och bestå av flera saker

Tex en switch sats, där exception kan kastas

```
private static Predicate<Result> successOrFailurePredicate(KindOfType type) {  1 usage
    switch (type) {
        case TYPE1 -> {
            System.out.println("Type1 -> SUCCESS");
            return r -> r.equals(Result.SUCCESS);
        }
        case TYPE2 -> {
            System.out.println("Type2 -> FAILURE");
            return r -> r.equals(Result.FAILURE);
        }
        default -> {
            System.out.println("unexpected new type - not yet implemented");
            throw new IllegalArgumentException("new type - not yet implemented: " + type);
        }
    }
}

public static List<Result> getResultsDependingOnType(List<ResultObjectWithReason> results, KindOfType type)
    return results.stream() Stream<ResultObjectWithReason>
            .map(ResultObjectWithReason::getResult) Stream<Result>
            .filter(successOrFailurePredicate(type))
            .toList();
}
```

Tex switch sats som returnerar en optional

```
private static Predicate<Result> successOrFailurePredicateUsingSwitchAndOptional(KindOfType type) {  1 usage
    Optional<Predicate<Result>> predicate = Optional.empty();
    switch (type) {
        case TYPE1 -> predicate = Optional.of(r -> r.equals(Result.SUCCESS));
        case TYPE2 -> predicate = Optional.of(r -> r.equals(Result.FAILURE));
    }
    return predicate.orElseThrow(
            () -> new IllegalArgumentException("new type - not yet implemented: " + type));
}

public static List<Result> getResultsDependingOnType2(List<ResultObjectWithReason> results, KindOfType type) {
    return results.stream() Stream<ResultObjectWithReason>
            .map(ResultObjectWithReason::getResult) Stream<Result>
            .filter(successOrFailurePredicateUsingSwitchAndOptional(type))
            .toList();
}
```

Gamla exempel:
Komplicerat predikat med switch sats

```
@NotNull
private static Predicate<ClaimEvent> claimEventsOfApplicableEventTypePredicate(QuestionnaireType questionnaireType) {
    switch (questionnaireType) {
        case MOTOR_CLAIM_SINGLE:
            return claimEvent -> claimEvent.getClaimEventType().equals(ClaimEventType.COLLISION)
                    || claimEvent.getClaimEventType().equals(ClaimEventType.VANDALISM)
                    || claimEvent.getClaimEventType().equals(ClaimEventType.SALVAGE)
                    || claimEvent.getClaimEventType().equals(ClaimEventType.UNKNOWN);
        case MOTOR_CLAIM_COLLISION:
            return claimEvent -> claimEvent.getClaimEventType().equals(ClaimEventType.COLLISION)
                    || claimEvent.getClaimEventType().equals(ClaimEventType.SALVAGE)
                    || claimEvent.getClaimEventType().equals(ClaimEventType.UNKNOWN);
        case MOTOR_CLAIM_MACHINERY_BREAKDOWN:
            return claimEvent ->
                    claimEvent.getClaimEventType().equals(ClaimEventType.SALVAGE)
                    || claimEvent.getClaimEventType().equals(ClaimEventType.MACHINERY_BREAKDOWN);
        default:
            // Not possible to reach/test
            throw new IllegalStateException("Not yet implemented");
    }
}
```

```
List<ClaimEvent> allApplicableClaimEvents = allExistingClaimEvents.stream()
        .filter(claimEventOfApplicableDatesPredicate(applicableDates))
        .filter(claimEventsOfApplicableEventTypePredicate(layout.getQuestionnaireType()))
        .filter(claimEventOfApplicablePolicyLineObjectPredicate(policyLineObject))
        .collect(toList());
```

Komplicerat predikat, switch case, optional, kastar exception:

```
private final static Predicate<Policy> homePolicies = (Policy policy) -> policy.getPolicyType().equals(PolicyType.HOME);
private final static Predicate<Policy> HSBPolicies = (Policy policy) -> policy.getPolicyType().equals(PolicyType.HSB);
private final static Predicate<Policy> livingPolicies = homePolicies.or(HSBPolicies);
private final static Predicate<Policy> motorPolicies = (Policy policy) -> policy.getPolicyType().equals(PolicyType.MOTOR);
```

```
public Predicate<Policy> getPolicyTypeFilter(PolicyType policyType) {
    Optional<Predicate<Policy>> policyPredicate = Optional.empty();
    switch (policyType) {
        case HOME:
            policyPredicate = Optional.of(livingPolicies);
            break;
        case MOTOR:
            policyPredicate = Optional.of(motorPolicies);
            break;
    }
    return policyPredicate.orElseThrow(()-> new IllegalStateException("
}
```

```
Predicate<Policy> policyFilter = sessionStateMap.getPolicyTypeFilter(policyType);

Set<Policy> policiesOfSpecifiedPolicyType = validPoliciesOnDate.stream()
        .filter(policyFilter)
        .collect(Collectors.toSet());
```

Komplicerat predikat som även kastar exception

```
private Predicate<PolicyDocument> hasAttachmentStartingWith(String filename) {
    return policyDocument -> {
        List<BinaryFile> attachments = policyDocument.getAttachments();
        Boolean hasCorrectName = attachments.stream() Stream<BinaryFile>
                .findFirst() Optional<BinaryFile>
                .map(attachment -> attachment.getName().toLowerCase().startsWith(filename))
                .orElseThrow(IllegalStateException::new);
        return hasCorrectName;
    };
}
```

```
public List<PolicyDocument> filterAdviceDocumentsOnAttachmentName(List<PolicyDocument> fileReferences) {
    return fileReferences.stream()
            .map(ref -> getPolicyDocument(ref.getId()).withDocumentType(ADVICE).withCreatedDate(ref.getCreatedDate()))
            .filter(hasAttachmentStartingWith(ADVICE_FILE_NAME))
            .collect(Collectors.toList());
}
```

Vanliga predikat i filtrering, olika slag:

```
private Predicate<Policy> customerAgreementsWithoutReferrals() {
    return policy -> policy.getReferToUnderwriter() == 0;
}

1 usage  ▲ PetSam1
private Predicate<Policy> getSuspendedPolicies() {
    return policy -> policy.getPolicyStatus().equals(SUSPENDED_POLICY);
}

1 usage  ▲ PetSam1
private Predicate<Policy> getSuspendedQuotes() {
    return policy -> policy.getPolicyStatus().equals(SUSPENDED_QUOTE);
}

1 usage  ▲ PetSam1
private Predicate<Policy> isValidCoverStartDate() {
    return policy -> policy.getCoverStartDate().isAfter(LocalDate.now()) || policy.getCoverStartDate().isEqual(LocalDate.now());
}
```

```
private void updateSessionStateMapWithCustomerAgreements(List<Policy> allCustomerAgreements) {
    Set<Policy> filteredCustomerAgreements = allCustomerAgreements.stream()
            .filter(customerAgreementsWithoutReferrals()).collect(Collectors.toSet());
    Set<Policy> suspendedPolicies = filteredCustomerAgreements.stream()
            .filter(getSuspendedPolicies())
            .collect(Collectors.toSet());
    Set<Policy> validSuspendedQuotes = filteredCustomerAgreements.stream()
            .filter(getSuspendedQuotes())
            .filter(isValidCoverStartDate())
            .collect(Collectors.toSet());
    this.sessionStateMap.getCustomerAgreements().addAll(suspendedPolicies);
    this.sessionStateMap.getCustomerAgreements().addAll(validSuspendedQuotes);
}
```

Andra smarta tips

Sortera med en separat Comparator:

```
private final static Predicate<ResultObjectWithReason> isFailurePredicate = resultObject -> Result.FAILURE.equals(resultObject.getResult());
private final static Comparator<ResultObjectWithReason> reasonComparator = Comparator.comparing(ResultObjectWithReason::getReason);  1 usage

public static List<ResultObjectWithReason> getFailuresSortedByReason(List<ResultObjectWithReason> results) {  2 usages
    return results.stream()
            .filter(isFailurePredicate)
            .sorted(reasonComparator)
            .toList();
}
```

Mappa om mha function eller bifunction:

```
private final static Function<ResultObjectWithReason, String> getActualErrorMessage = r -> r.getReason().split( regex: "[")[1];  2 usages
public static List<String> getFailuresWithModifiedErrorMessage(List<ResultObjectWithReason> results) {  1 usage
    return results.stream()  Stream<ResultObjectWithReason>
            .filter(isFailurePredicate)
            .map(getActualErrorMessage)  Stream<String>
            .toList();
}

private final static BiFunction<ResultObjectWithReason, String, String> formatErrorMsg = (r, s) -> s.concat(getActualErrorMessage.apply(r));
public static List<String> getFailuresWithModifiedErrorMessage2(List<ResultObjectWithReason> results) {  1 usage
    return results.stream()  Stream<ResultObjectWithReason>
            .filter(isFailurePredicate)
            .map(r -> formatErrorMsg.apply(r,  s: "The following error occurred: "))  Stream<String>
            .toList();
}
```

Se gamla exempel:

```
private final static Comparator<SearchResult> bySearchName = Comparator.comparing(SearchResult::getSearchName);
1 usage
private final static BiFunction<String, String, String> concatenation = (s1, s2) -> s1.concat( str: " ").concat(s2);
1 usage
private final static BiFunction<Long, ExternalSweBoat, SearchResult> creationOfSearchResults = (id, boat) -> new SearchResult( id: "" + id, concatenation.apply(boat.getBrandName(), boat.getModelName()));
8 usages
```

```
private void createSearchResults(List<ExternalSweBoat> boats) {
    Map<String, SearchResult> searchResults = boats.stream()  Stream<ExternalSweBoat>
            .map(boat -> creationOfSearchResults.apply(boat.id, boat))  Stream<Searc
            .collect(Collectors.toMap(SearchResult::getId, Function.identity()));
    this.entrySetWithBoatBrandsAndModels = searchResults.entrySet();
}
```

```
public List<SearchResult> searchForBoatBrandsAndModels(String searchString) {
    final String finalSearchString = searchString.toLowerCase();
    if(entrySetWithBoatBrandsAndModels != null) {
        return entrySetWithBoatBrandsAndModels.stream()  Stream<Entry<_,>>
                .map(entry -> entry.getValue())  Stream<SearchResult>
                .filter(searchResult -> searchResult.getSearchName().contains(finalSearchString))
                .sorted(bySearchName)
                .collect(Collectors.toList());
    }
    return null;
}
```