P1C4 - Prenez en main Spring Boot

Vérifiez votre travail

Le point d'entrée. Une annotation de plus suffit à activer les records de configuration :

```
@SpringBootApplication
@ConfigurationPropertiesScan
public class HamacApplication {

    public static void main(String[] args) {
        SpringApplication.run(HamacApplication.class, args);
    }
}
```

La configuration. `application.properties` contient désormais :

```
spring.jpa.open-in-view=false
spring.jpa.defer-datasource-initialization=true
logging.level.org.hibernate.SQL=debug
hamac.leave.fixed-term-days-per-full-month=2.08
```

La politique des CDD et son test. La règle reste dans le code, sa valeur vient de la configuration :

```
@Component
final class FixedTermContractPolicy implements LeaveEntitlementPolicy {

    private static final long DAYS_PER_MONTH = 30;

    private final double daysPerFullMonth;

    FixedTermContractPolicy(LeaveProperties properties) {
        this.daysPerFullMonth = properties.fixedTermDaysPerFullMonth();
    }

    @Override
    public boolean supports(ContractType type) {
        return type == ContractType.FIXED_TERM;
    }

    @Override
    public double computeEntitlement(Employee employee, LocalDate today) {
        var daysSinceHire = ChronoUnit.DAYS.between(employee.getHireDate(), today);
        var fullMonths = Math.floorDiv(daysSinceHire, DAYS_PER_MONTH);
        return fullMonths * daysPerFullMonth;
    }
}
```

Le test unitaire ne compile plus tel quel, puisque la politique demande maintenant un paramètre. C'est l'injection par constructeur qui vous le signale, et c'est exactement ce que vous voulez. Le test fournit lui-même la valeur, sans démarrer Spring :

```
var policy = new FixedTermContractPolicy(new LeaveProperties(2.08));
```

Le service. Il est devenu un bean transactionnel, et il ne filtre plus en mémoire :

```
@Service
```

```
class LeaveService {

    private final EmployeeRepository employeeRepository;
    private final LeaveRequestRepository leaveRequestRepository;
    private final LeavePolicyResolver policies;
    private final LeaveNotifier notifier;

    LeaveService(EmployeeRepository employeeRepository,
                 LeaveRequestRepository leaveRequestRepository,
                 LeavePolicyResolver policies,
                 LeaveNotifier notifier) {
        this.employeeRepository = employeeRepository;
        this.leaveRequestRepository = leaveRequestRepository;
        this.policies = policies;
        this.notifier = notifier;
    }

    public Optional<Double> getBalance(long employeeId) {
        return employeeRepository.findById(employeeId).map(this::computeBalance);
    }

    @Transactional
    public ApprovalResult approve(long requestId) {
        var found = leaveRequestRepository.findById(requestId);
        if (found.isEmpty()) {
            return ApprovalResult.NOT_FOUND;
        }
        var request = found.get();
        var employee = employeeRepository.findById(request.getEmployeeId()).orElseThrow();

        if (computeBalance(employee) < request.getWorkingDays()) {
            request.setStatus("REFUSED");
            leaveRequestRepository.save(request);
            return ApprovalResult.INSUFFICIENT_BALANCE;
        }

        request.setStatus("APPROVED");
        employee.setDaysTaken(employee.getDaysTaken() + request.getWorkingDays());
        leaveRequestRepository.save(request);
        employeeRepository.save(employee);

        notifier.notify(employee, "Votre demande de congés est approuvée.");
        return ApprovalResult.APPROVED;
    }

    private double computeBalance(Employee employee) {
        var type = ContractType.fromCode(employee.getContractType());
        return policies.policyFor(type).computeEntitlement(employee, LocalDate.now())
                - employee.getDaysTaken();
    }
}
```

La classe n’est plus `final`, pour que Spring puisse appliquer la transaction, et elle n’est plus `public`, puisque seul le contrôleur du même package l’utilise. Son constructeur ne porte aucune annotation : c’est le seul, Spring l’utilise.

Le repository des demandes.

```
interface LeaveRequestRepository extends Repository<LeaveRequest, Long> {

    Optional<LeaveRequest> findById(Long id);

    LeaveRequest save(LeaveRequest request);
}
```

Pas de `@Repository` : Spring Data la détecte seul.

Le contrôleur. Son constructeur se réduit à une affectation, comme vous l'avez vu plus haut. Les méthodes `getBalance` et `approve` ne changent pas.

Le test de la notification. Dans `LeaveApprovalTests`, une doublure remplace le vrai notificateur pour tous les tests de la classe :

```
@MockitoBean
LeaveNotifier notifier;

@Test
void notifiesEmployeeWhenRequestIsApproved() {
    assertThat(mvc.get().uri("/api/leave/approve?id=1")).hasStatusOk();

    verify(notifier).notify(any(Employee.class), eq("Votre demande de congés est approuvée."));
}
```

`@MockitoBean` remplace dans le conteneur le bean de type `LeaveNotifier` par une doublure Mockito, qui enregistre les appels au lieu d'envoyer des messages. Les tests n'envoient plus de vraies notifications, et vous pouvez enfin vérifier qu'une approbation en déclenche une. C'est la seule exception à la règle de l'injection par constructeur : dans une classe de test, `@MockitoBean` s'applique à un champ. Dans les tutoriels plus anciens, vous trouverez `@MockBean` : cette annotation n'existe plus dans Spring Boot 4.

La distinction attendue est la suivante : déplacer les `new` du contrôleur vers une autre classe aurait simplement **déplacé l'assemblage** ; en laissant le conteneur lire les constructeurs, vous l'avez **rendu déclaratif**. Chaque classe déclare ce dont elle a besoin, et Spring se charge de le fournir. Pour ajouter les stages, une classe annotée `@Component` suffira.

Erreurs fréquentes

- **Laisser** `LeaveService` **en** `final` : avec `@Transactional`, l'application ne démarre plus, car Spring ne peut pas créer la sous-classe qui gère la transaction.
- **Ajouter** `@Autowired` **ou un second constructeur** : avec un constructeur unique, l'annotation est inutile ; avec deux constructeurs, Spring ne sait plus lequel utiliser.
- **Placer** `@Transactional` **sur le contrôleur** : la transaction fonctionne, mais le contrôleur décide à nouveau de ce qui forme un tout métier.
- **Oublier** `@ConfigurationPropertiesScan` : le message indique alors qu'aucun bean de type `LeaveProperties` n'a été trouvé.
- **Remplacer** `findAll()` **par** `findAll().stream().filter(…).findFirst()` **dans une autre méthode** : le nom change, mais la base renvoie toujours toute la table.
- **Tout faire en un seul commit** : si un test échoue, vous ne savez plus si la cause est l'injection, la configuration ou l'accès aux données.