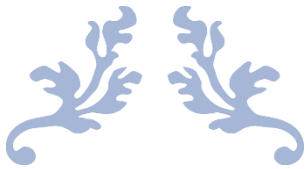
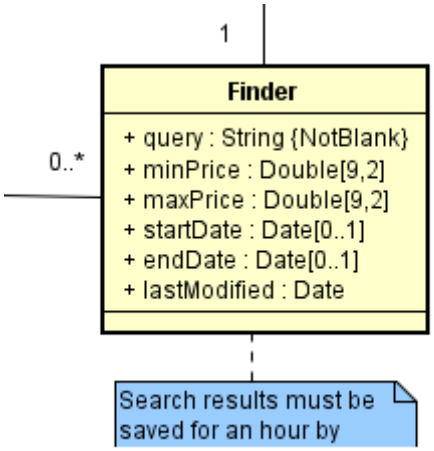


CHANGELOG D05

Cambios con respecto al entregable anterior

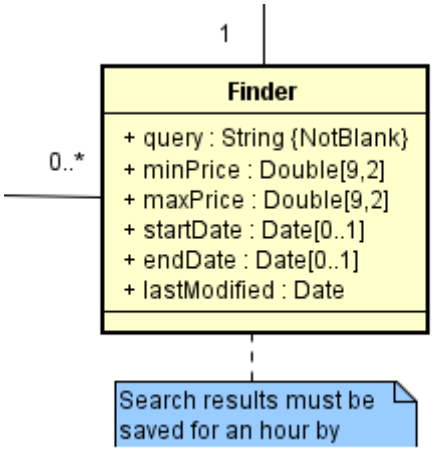
1. Cambios en el Modelo Conceptual

Para esta entrega nos ha sido necesario añadir una nueva propiedad a la tabla Finder, que modela el buscador de los Handy Workers. Este cambio ha sido añadir una nueva propiedad, lastModified, que marca la última actualización del Finder, y define si es necesario cargar los resultados de nuevo o debemos utilizar los anteriores.



2. Cambios en el modelo de dominio

Los cambios realizados en el modelo de dominio son los mismos que en el conceptual, sólo se ha añadido esta nueva propiedad de la que hablamos anteriormente.



3. Cambios en el modelo Java

En el modelo Java hemos tenido que cambiar varias clases, cuyos añadidos detallamos a continuación:

- Clase AdministratorService:
  En esta clase hemos creado un método para poder generar el ticker de manera general, además de los métodos relacionados con el A-Level de la entrega anterior:

```
public String generateTicker() {
        byte[] array = new byte[6]; // length is bounded by 7
        new Random().nextBytes(array);
        String generatedString = new String(array, Charset.forName("UTF-8"));

        return generatedString.toUpperCase();
    }
    public List<String> getBadWords(){
            SystemConfig configuration =
administratorRepository.returnSystemConfig().get(0);
            return configuration.getBadWords();
    }

    public List<String> addBadWord(String word){
            SystemConfig configuration =
administratorRepository.returnSystemConfig().get(0);
            configuration.getBadWords().add(word);
            return configuration.getBadWords();
    }

    public void deleteBadWord(String word){
            SystemConfig configuration =
administratorRepository.returnSystemConfig().get(0);
            configuration.getBadWords().remove(word);
    }

    public List<String> getGoodWords(){
```

```
		SystemConfig configuration =
administratorRepository.returnSystemConfig().get(0);
		return configuration.getBadWords();
	}

	public List<String> addGoodWord(String word){
		SystemConfig configuration =
administratorRepository.returnSystemConfig().get(0);
		configuration.getGoodWords().add(word);
		return configuration.getGoodWords();
	}

	public void deleteGoodWord(String word){
		SystemConfig configuration =
administratorRepository.returnSystemConfig().get(0);
		configuration.getGoodWords().remove(word);
	}

	public Map<Actor, Score> calculateAllActorsScores(){ //void
		Map<Actor, Score> allscores = new HashMap<Actor, Score>();
		List<Endorsement> endorsements = (List<Endorsement>) es.findAll();

		for(Endorsement e: endorsements){
			Double newscore = e.calculateScore().getNumericScore();
			if(allscores.containsKey(e.getEndorser())){
				Double oldscore =
allscores.get(e.getEndorser()).getNumericScore();
				Double oldplusnew = (oldscore + newscore)/2;

				Score putscore = new Score();
				putscore.setNumericScore(oldplusnew);
```

```
                    allscores.put(e.getEndorser(), putscore);

                } else {

                    Score putscore = new Score();
                    putscore.setNumericScore(newscore);

                    allscores.put(e.getEndorser(), putscore);

                }
            }

            return allscores;
        }

}
```

- Clase customerService:
  En esta clase ha sido necesario crear los métodos relacionados con el A-Level de la entrega anterior:

```
public Collection<Endorsement> getEndorsments(){
            Assert.isTrue(!actorservice.isActualActorBanned());

            Collection<Endorsement>  res = new ArrayList<>();
            Customer actual =
uas.getCustomerByUserAccount(LoginService.getPrincipal());
            Assert.isTrue(!actual.getIsBanned());

            Collection<Endorsement> prueba =endorsementService.findAll();
```

```
for(Endorsement a :endorsementService.findAll()){
		if(a.getSender().equals(actual)){
			res.add(a);
		}

}

		return res;
		}
		public Endorsement getEndorsement(int id){
			Assert.isTrue(!actorservice.isActualActorBanned());
			Customer actual =
uas.getCustomerByUserAccount(LoginService.getPrincipal());

			Endorsement  res =null;

			for(Endorsement a :endorsementService.findAll()){
				if(a.getSender().equals(actual)&& a.getId()==id){
					res=a;
				}

			}
			return res;
			}
public  Endorsement createEndorsement(HandyWorker handy,Endorsement endor,FixUpTask
n ){
		Assert.isTrue(!actorservice.isActualActorBanned());

		Customer actual = uas.getCustomerByUserAccount(LoginService.getPrincipal());
		checkAuthority();
```

```
Endorsement  res = new Endorsement();


        if
(handyWorkerRepository.getFixUpTaskByHandyWorker(handy.getUserAccount()).contains(n)){
            res.setComments(endor.getComments());
            res.setEndorser(handy);
            res.setMoment(LocalDate.now().toDate());
            res.setSender(actual);



        }
         Endorsement  result = endorsementService.save(res);
        return result;



}
public Endorsement updateEndorsement(Endorsement endor){
   Assert.isTrue(!actorservice.isActualActorBanned());
        checkAuthority();
   Assert.notNull(endor);
        Endorsement  result =this.endorsementService.save(endor);
        return result;
}
public void  deleteEndorsement(Endorsement endor){
        Assert.isTrue(!actorservice.isActualActorBanned());
        this.endorsementService.delete(endor);
        }


}
```

- Clase SponsorService:

  Creados los métodos del A-Level de la entrega anterior:

```
public Sponsor create(){
		final Sponsor result = new Sponsor();
		final UserAccount user = new UserAccount();
		result.setProfiles(new ArrayList<Profile>());
		final Authority a = new Authority();
		a.setAuthority(Authority.SPONSOR);
		final Collection<Authority> r = new ArrayList<Authority>();
		r.add(a);
		user.setAuthorities(r);
		result.setUserAccount(user);

		return result;
	}
	public Collection<Sponsor> findAll(){
		return sponsorRepository.findAll();
	}

	public Sponsor findOne(int sponsorId){
		return sponsorRepository.findOne(sponsorId);
	}

	public Sponsor save(Sponsor sponsor){
		return sponsorRepository.save(sponsor);
	}

	public void delete(Sponsor sponsor){
		sponsorRepository.delete(sponsor);
	}
```

```
public void checkAuthority() {
        Assert.isTrue(!actorService.isActualActorBanned());
        UserAccount ua;
        ua = LoginService.getPrincipal();
        Assert.notNull(ua);
        final Collection<Authority> auth = ua.getAuthorities();
        final Authority a = new Authority();
        a.setAuthority(Authority.SPONSOR);
        Assert.isTrue(auth.contains(a));
}

public Sponsor register(final Sponsor sp, final String username, final String password) {

        this.checkAuthority();

        Assert.notNull(username);
        Assert.notNull(password);
        Assert.notNull(sp);
        Assert.notEmpty(sp.getProfiles());



        final Sponsor result = this.create();
        result.setAddress(sp.getAddress());
        result.setMiddleName(sp.getMiddleName());
        result.setName(sp.getName());
        result.setPhone(sp.getPhone());
        result.setPhoto(sp.getPhoto());
        result.setProfiles(sp.getProfiles());
        result.getUserAccount().setPassword(password);
        result.getUserAccount().setUsername(username);
```

```
		final MessageBox in = actorService.createNewMessageBox(username, "-in");
		final MessageBox out = actorService.createNewMessageBox(username, "-
out");
		final MessageBox trash = actorService.createNewMessageBox(username, "-
trash");
		final MessageBox spam = actorService.createNewMessageBox(username, "-
spam");

		final Collection<MessageBox> msboxes = new ArrayList<MessageBox>();
		msboxes.add(in);
		msboxes.add(out);
		msboxes.add(trash);
		msboxes.add(spam);

		result.setMessageBoxes(msboxes);
		final Sponsor x = this.save(result);

		return x;

	}

	public Collection<Tutorial> findAllTutorials(){ //inutilidad
		this.checkAuthority();
		Collection<Tutorial> res = tutorialService.findAll();
		return res;
	}

	public Map<HandyWorker, Collection<Tutorial>> findTutorialsByHandyWorker(){
		this.checkAuthority();
		Map<HandyWorker, Collection<Tutorial>> res = new HashMap<HandyWorker,
Collection<Tutorial>>();
		Collection<HandyWorker> hws = handyWorkerService.findAll();
```

```
            for(HandyWorker h: hws){
                res.put(h, putTutorialInHandyWorker(h));
          }

            return res;
      }

private Collection<Tutorial> putTutorialInHandyWorker(HandyWorker h) {
            Collection<Tutorial> res = new ArrayList<Tutorial>();
            for(Tutorial t: this.tutorialService.findAll()){
                  if(t.getHandyworker()==h){
                        res.add(t);
                  }
            }
            return res;
      }
```

- Clase HandyWorkerService:
  Generados los métodos del A-Level de la entrega anterior:

```
                  public Collection <Endorsement> listEndorsements(){
                        Assert.isTrue(!actorservice.isActualActorBanned());
                        final HandyWorker actual =
      this.uas.getHandyByUserAccount(LoginService.getPrincipal());
                        final Authority a = new Authority();
                        a.setAuthority(Authority.HANDYWORKER);

            Assert.isTrue(actual.getUserAccount().getAuthorities().contains(a));

                        Collection<Endorsement> res = new ArrayList<>();
                        for(Endorsement endor: this.endorsementService.findAll()){
                              if(
      (endor.getEndorser().equals(actual))||(endor.getSender().equals(actual))){
                                    res.add(endor);
                              }
                        }

                        return res;
```

```
        }

        public Endorsement showEndorsement(int id){

                Assert.isTrue(!actorservice.isActualActorBanned());
            final HandyWorker actual =
this.uas.getHandyByUserAccount(LoginService.getPrincipal());
                final Authority a = new Authority();
                a.setAuthority(Authority.HANDYWORKER);

        Assert.isTrue(actual.getUserAccount().getAuthorities().contains(a));

            Endorsement  res =null;

            for(Endorsement endor :endorsementService.findAll()){
               if(
(endor.getEndorser().equals(actual))||(endor.getSender().equals(actual))&&
endor.getId()==id){
                  res=endor;
               }


            }
            return res;
          }

        public  Endorsement createEndorsement(Customer
customer,Endorsement endor,FixUpTask n){

            Assert.isTrue(!actorservice.isActualActorBanned());
          checkAuthority();
          HandyWorker actual =
uas.getHandyByUserAccount(LoginService.getPrincipal());
          Endorsement  res = new Endorsement();

          if (customer.getFixUpTasks().contains(n)){
            res.setComments(endor.getComments());
            res.setEndorser(customer);
            res.setMoment(LocalDate.now().toDate());
            res.setSender(actual);


          }
           Endorsement  result =    endorsementService.save(res);
          return result;
```

```
}
public Endorsement updateEndorsement(Endorsement endor){
   Assert.isTrue(!actorservice.isActualActorBanned());
   checkAuthority();
   Assert.notNull(endor);
   Endorsement  result =this.endorsementService.save(endor);
   return result;
}
public void  deleteEndorsement(Endorsement endor){
   Assert.isTrue(!actorservice.isActualActorBanned());
   this.checkAuthority();
   this.endorsementService.delete(endor);
   }
```