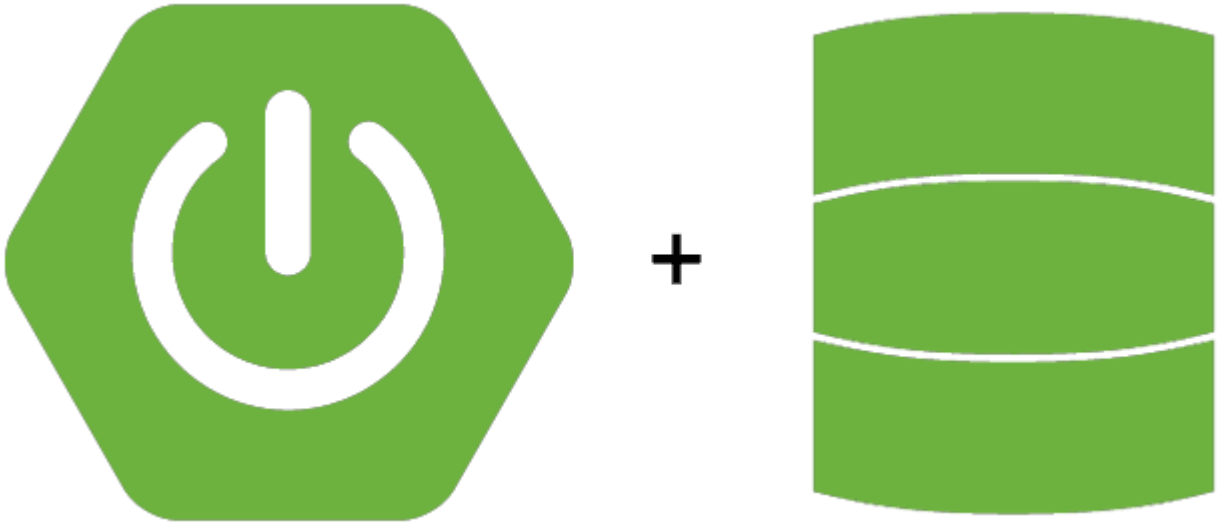
SPRING DATA JPA – CrudRepository



UP ASI
Bureau E204

Plan du Cours

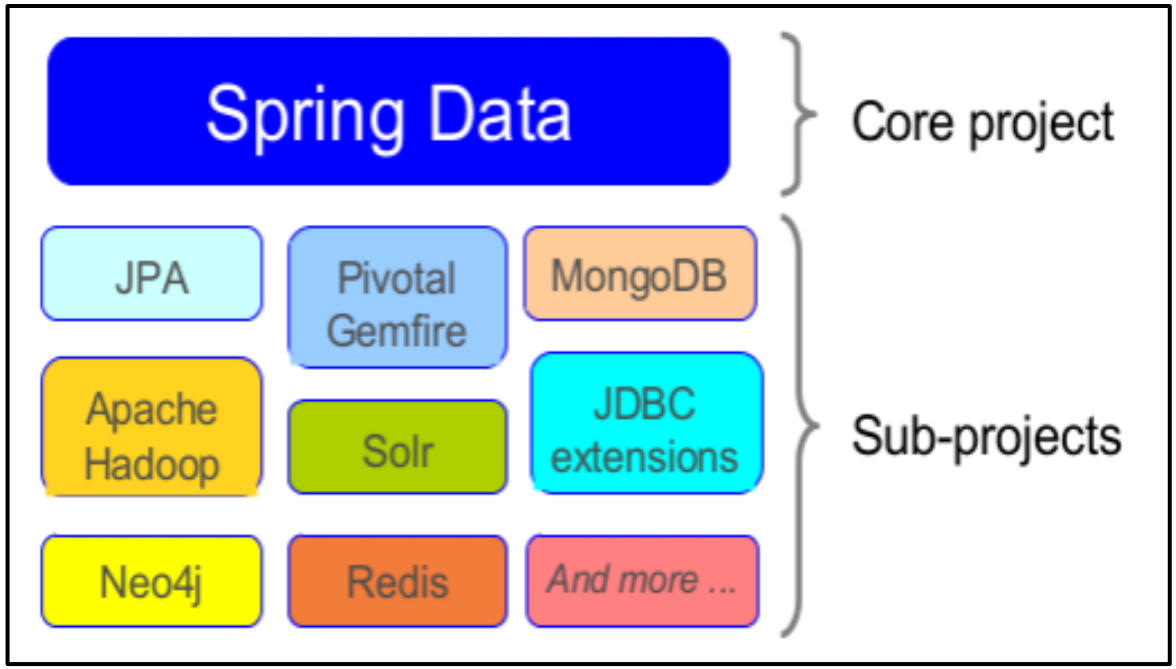
- Spring Data
- Spring Data JPA

- CRUD Repository interface
- Créer et Utiliser un Repository
- Keywords
- JPQL

- Exercice

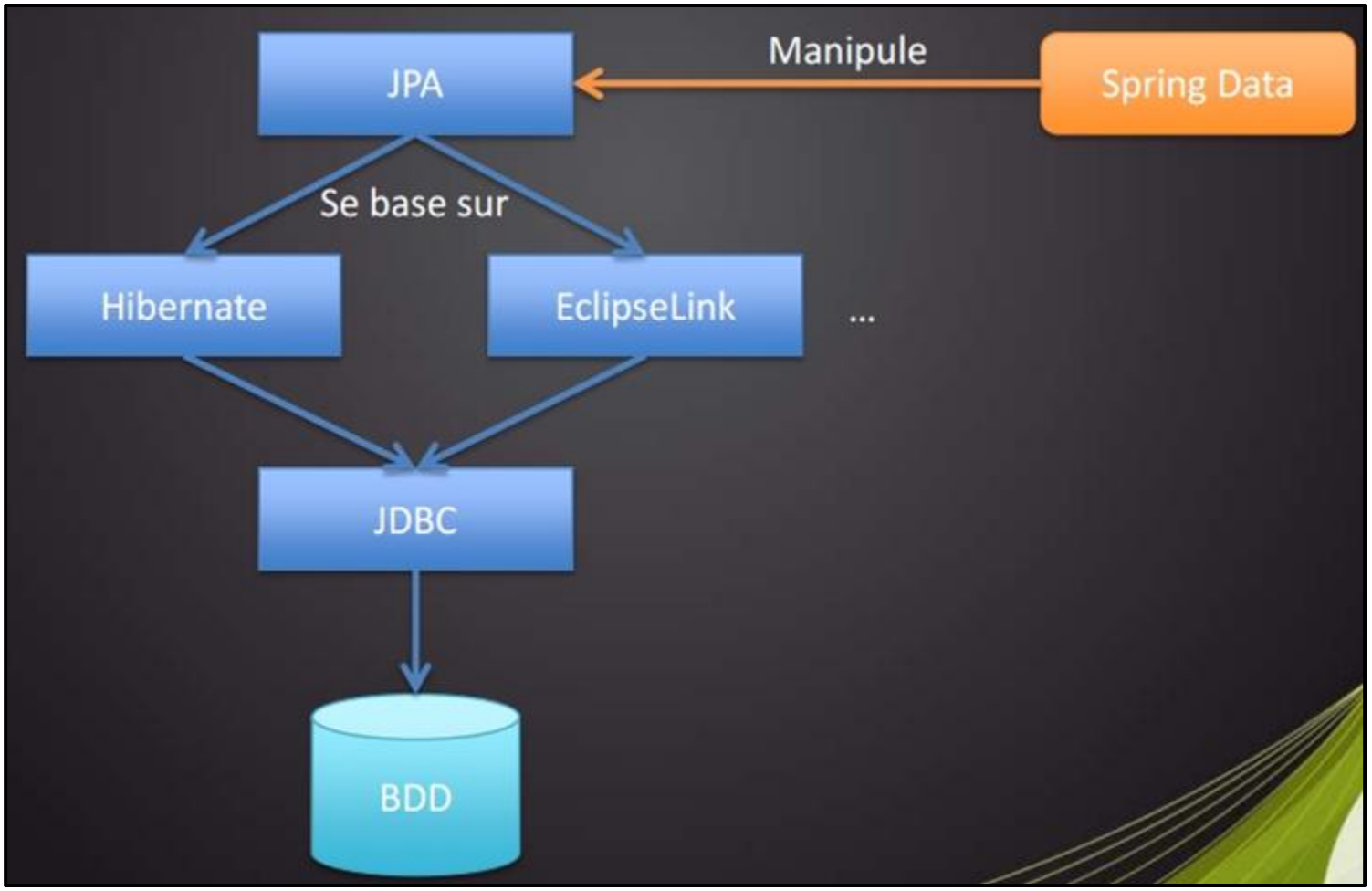
SPRING DATA

- C'est un module Spring qui a pour but de :
  - Faciliter l'écriture des couches d'accès aux données.
  - Offrir une abstraction commune pour l'accès aux données quelle que soit la source de données (SQL ou NoSQL).



SPRING DATA JPA

- Spring Data JPA est un sous projet du projet Spring Data.



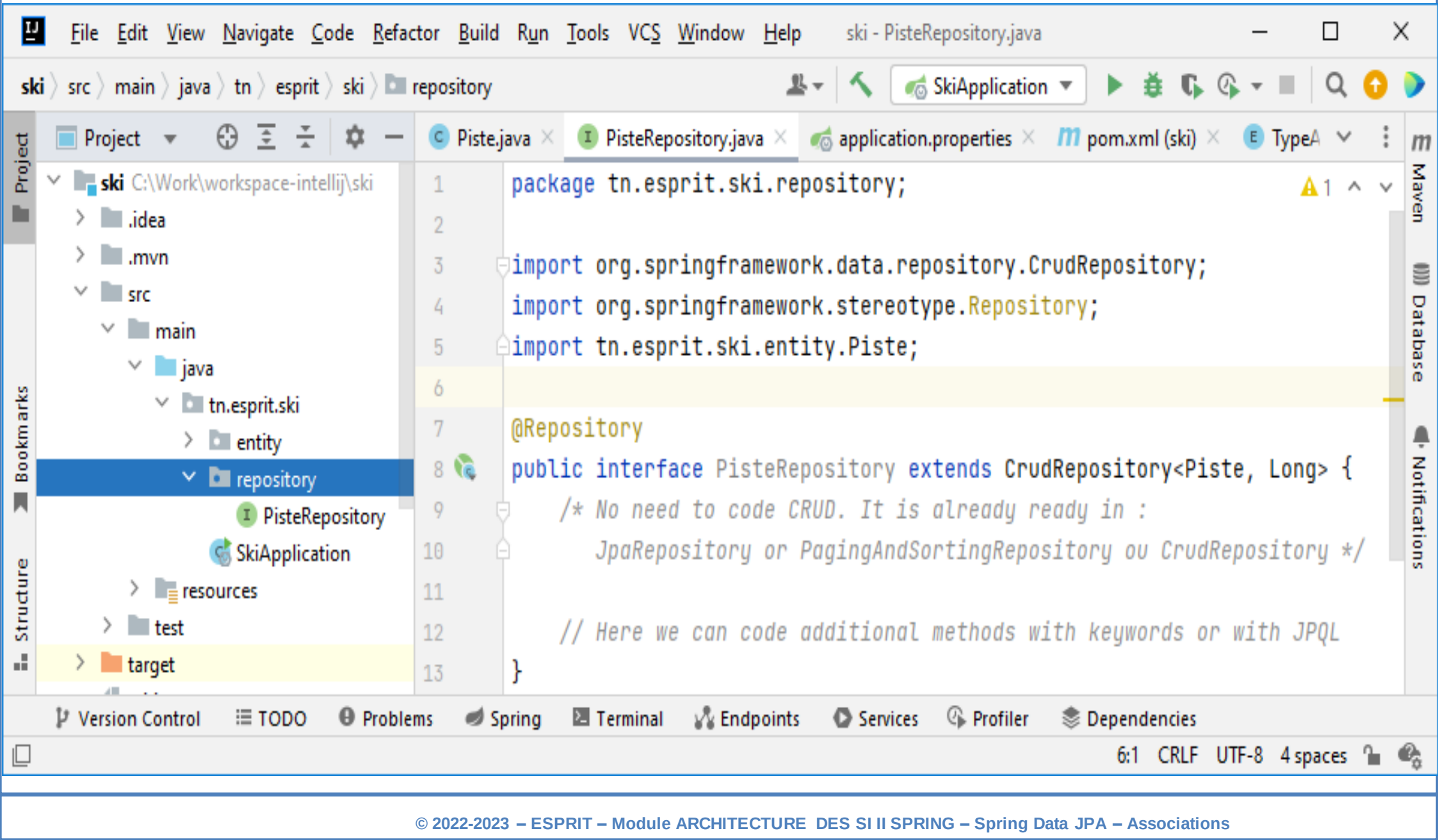
SPRING DATA JPA

- Code répétitif avec les DAO (Data Access Object) avant Spring Data JPA:

```
public class ProjetDAO {

public Projet findById (Long id) {
//code ...
}

public List<Projet>findAll{
//code ...
}

}
```

```
public class EntrepriseDAO{

public Entreprise findById (Long id) {
//code ...
}

public List<Entreprise>findAll{
//code ...
}

}
```

Exemple Repository de l'Entité PisteRepository



SPRING DATA JPA

- Avec Spring Data JPA, il vous suffit de définir une interface qui permet de manipuler une entité, en étendant l'interface CrudRepository <T, ID> et de déclarer les méthodes pour manipuler cette entité.
- Spring Data JPA créera une classe qui implémente cette interface pour vous.
- Exemple :

```
public interface ProjetRepository extends JpaRepository<Projet, Long> {..}
```

- L'interface **ProjetRepository** étend l'interface **CrudRepository<Projet, Long>**. Elle contient les méthodes pour manipuler l'entité Projet.
- Spring Data JPA va automatiquement créer une classe qui implémente cette interface au moment de l'exécution de la l'application.

KEYWORDS

- Spring Data JPA va créer le code pour les méthodes que tu souhaites utiliser. Il suffit de lui indiquer les méthodes à utiliser :

Keyword	Sample	Equivalent to
GreaterThan	findBydateDebutGreaterThan(Date dateC);	Select p from ProjetDetail p where p.dateDebut > :dateC
LessThan	findBydateDebutLessThan(Date dateN);	Select p from ProjetDetail p where p.dateDebut < :dateN
Between	findBydateDebutBetween(Date dFrom, Date dTo);	Select p from ProjetDetail p where p.dateDebut between :dFrom and :dTo
IsNotNull, NotNull	findByCout_provisoireNotNull();	Select p from ProjetDetail p where p.cout_provisoire is not null
IsNull, Null	findByDescriptionNull();	Select p from ProjetDetail p where p.description is null
Like	findByTechnologieLike(String technologie);	Select p from ProjetDetail p where p. technologie like :technologie
(No keyword)	findByTechnologie (String technologie);	Select p from ProjetDetail p where p. technologie= :technologie
(No keyword)	findOne(ID primaryKey);	Select p from ProjetDetail p where p.id =:primaryKey
(No keyword)	Long count();	Select count(*) From Projet;

KEYWORDS

```
@Repository
public interface ProjetDetailRepository
extends JpaRepository<ProjetDetail, Long>
{
```

```
@Entity
@Table(name = "T_PROJET_DETAIL")
public class Projet_Detail implements Serializable {
        private static final long serialVersionUID = 1L;
        @Id
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        @Column(name = "PD_ID")
        private Long id; // Identifiant projet detail (Clé
        @Column(name = "PD_DESCRIPTION")
        private String description;
        @Column(name = "PD_TECHNOLOGIE")
        private String technologie;
        @Column(name = "PD_COUT_PROVISOIRE")
        private Long cout_provisoire;
        @Temporal(TemporalType.DATE)
        private Date dateDebut;
        @OneToOne(mappedBy="projetDetail")
        private Projet projet;
}
```

```
// SELECT * FROM ProjetDetail WHERE technologie LIKE '%in%';
List<ProjetDetail> findByTechnologieLike(String technologie);
List<ProjetDetail> findByTechnologieContains(String technologie);
List<ProjetDetail> findByTechnologieContaining(String technologie);}
```

KEYWORDS

- Afficher la liste des projets qui ont une technologie précise.

```
@Repository
public interface ProjetRepository extends JpaRepository<Projet, Long> {
List<Projet> findByProjetDetailTechnologieContains(String technologie);
}
```

association

attribut dans la table Projet Detail

```
@Entity
@Table(name = "T_PROJET")
public class Projet implements Serializable {
        private static final long serialVersionUID = 1L;

        @Id
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        @Column(name = "PROJET_ID")
        private Long id; // Identifiant projet (Clé primair

        @Column(name = "PROJET_SUJET")
        private String sujet;

        @OneToOne
        private Projet_Detail projetDetail;

}
```

KEYWORDS

- Afficher la liste des projets d'une équipe.

```java
@Entity
@Table(name = "T_PROJET")
public class Projet implements Serializable {
	private static final long serialVersionUID = 1L;

	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	@Column(name = "PROJET_ID")
	private Long id; // Identifiant projet (Clé primaire)

	@Column(name = "PROJET_SUJET")
	private String sujet;

	@ManyToMany(mappedBy="projets", cascade = CascadeType.ALL)
	private Set<Equipe> equipes;
}
```

```java
@Repository
public interface ProjetRepository
extends JpaRepository<Projet, Long> {
List<Projet> findByEquipesIdEquipe(Long equipeId);
}
```

KEYWORDS

- Afficher la liste des projets d'une équipe dont la description est non nulle.

```
@Repository
public interface ProjetRepository extends JpaRepository<Projet, Long> {
List<Projet>
findByEquipesIdEquipeAndProjetDetailDescriptionNotNull(Long equipeId);
}
```

KEYWORDS

```
@Repository
public interface ProjetRepository extends JpaRepository<Projet, Long> {
```

- Afficher la liste des projets par équipe et entreprise.

```
List<Projet> findByEquipesIdEquipeAndEquipesEntrepriseIdEntreprise
(Long equipeId, Long entrepriseId);
```

- Afficher la liste des projets par la spécialité d'une équipe et l'adresse de l'entreprise.

```
List<Projet>
findByEquipesSpecialiteContainsAndEquipesEntrepriseAdresseContains
(String specialite, String adresse);
}
```

JPQL

- **JPQL** = Java Persistence Query Language
- JPQL peut être considéré comme une version orientée objet de SQL.
- En JPQL, on sélectionne des objets d'une entité (une classe annotée par @Entity) en utilisant les noms des attributs et le nom de la classe **de l'entité** en question et non plus ceux de la table de base de données et ses colonnes.

JPQL : SELECT

- Ces méthodes permettent de récupérer les entreprises avec une adresse donnée :
- **JPQL :**

```
@Query("SELECT e FROM Entreprise e WHERE e.adresse =:adresse")
List<Entreprise> retrieveEntreprisesByAdresse(@Param("adresse") String adresse);
```

C'est équivalent à :

```
@Query("SELECT e FROM Entreprise e WHERE e.adresse = ?1")
List<Entreprise> retrieveEntreprisesByAdresse(String adresse);
```

Supposons que nous avons mapper l'entité Entreprise avec **la table associé T_Entreprise**

- **Native Query (SQL et non JPQL) :**

```
@Query(value = "SELECT * FROM T_Entreprise e WHERE e.entreprise_adresse = :adresse",
nativeQuery = true)
List<Entreprise> retrieveEntreprisesByAdresse(@Param("adresse") String adresse);
```

C'est équivalent à :

```
@Query(value = "SELECT * FROM T_Entreprise e WHERE e.entreprise_adresse = ?1" ,
nativeQuery = true)
List<Entreprise> retrieveEntreprisesByAdresse( String adresse);
```

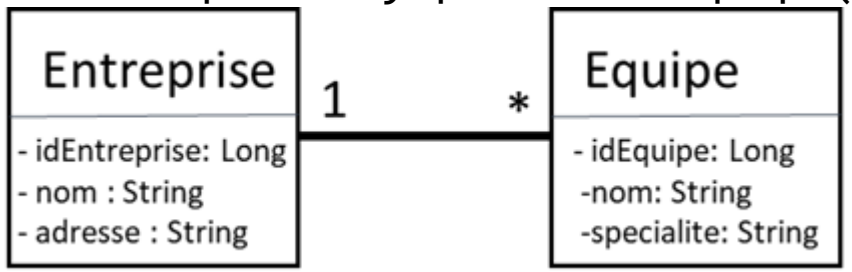
JPQL : SELECT

- Ces méthodes permettent de récupérer les entreprises qui ont une équipe avec une spécialité donnée :
- **JPQL :**

```
@Query("SELECT entreprise FROM Entreprise entreprise , Equipe equipe where
entreprise.id = equipe.entreprise.id and equipe.specialite =:specialite")

List<Entreprise> retrieveEntreprisesBySpecialiteEquipe(@Param("specialite")
String specialite);
```

- **Native Query (SQL et non JPQL) :**

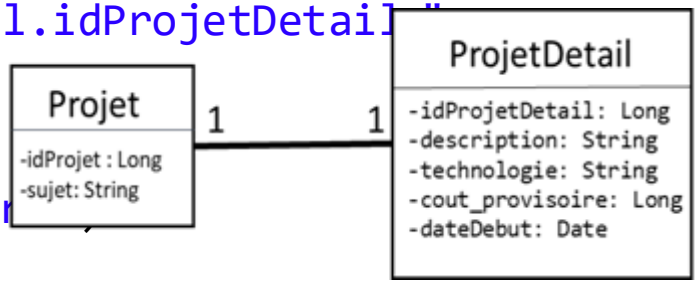
```
@Query(value = "SELECT * FROM T_ENTREPRISE entreprise INNER JOIN T_EQUIPE equipe
ON entreprise.ENTREPRISE_ID = equipe.ENTREPRISE_ENTREPRISE_ID where
equipe.EQUIPE_SPECIALITE =:specialite", nativeQuery = true)

List<Entreprise> retrieveEntreprisesBySpecialiteEquipe(@Param("specialite")
String specialite);
```



JPQL : SELECT

- Cette méthode permet d'afficher les projets qui ont un coût supérieur à coût donné dt et une technologie donnée.
- **JPQL :**

```
@Query("SELECT projet FROM Projet projet, ProjetDetail detail where "
        + "detail.idProjetDetail = projet.projetDetail.idProjetDetail "
        + "and detail.technologie =:technologie "
        + "and detail.cout_provisoire >:cout_provisoire")
List<Projet> retrieveProjetsByCoutAndTechnologie(@Param("technologie") String technologie,
        @Param("cout_provisoire") Long cout_provisoire);
```



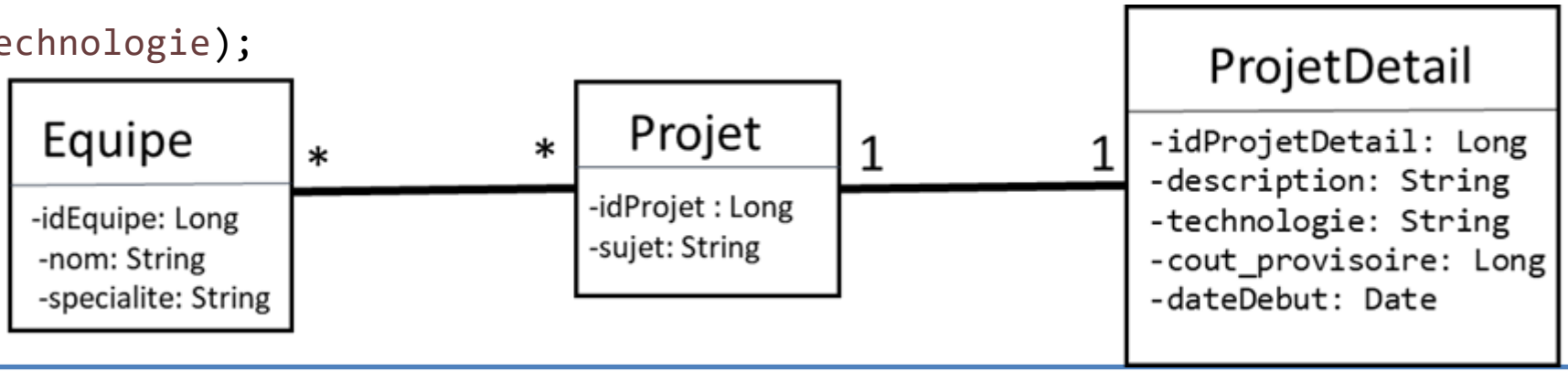
- **Native Query (SQL et non JPQL) :**

```
@Query(value = "SELECT * FROM T_PROJET projet INNER JOIN T_PROJET_DETAIL detail ON
detail.PD_ID =   projet.PROJET_DETAIL_PD_ID WHERE detail.PD_TECHNOLOGIE = ?1 and
detail.PD_COUT_PROVISOIRE > ?2", nativeQuery = true)
    List<Projet> retrieveProjetsByCoutAndTechnologie(@Param("technologie") String technologie,
        @Param("cout_provisoire") Long cout_provisoire);
```

JPQL : SELECT

- Cette méthode permet d'afficher les équipes qui travaillent sur une technologie donnée dont le projet n'a pas encore commencé.
- **JPQL :**

```
@Query("SELECT equipe FROM Equipe equipe"
        + " INNER JOIN equipe.projets projet"
        + " INNER JOIN ProjetDetail detail"
        + " ON detail.idProjetDetail = projet.projetDetail.idProjetDetail"
        + " where detail.dateDebut > current_date"
        + " and detail.technologie =:technologie")
List<Equipe> retrieveEquipesByProjetTechnologie(@Param("technologie")
String technologie);
```



JPQL : UPDATE

- Si nous souhaitons faire un **UPDATE, DELETE et INSERT**, nous devons ajouter l'annotaion **@Modifying** pour activer la modification de la base de données.

- Cette méthode permet de mettre à jour l'adresse de l'entreprise.

- **JPQL :**

```
@Modifying

@Query("update Entreprise e set e.adresse = :adresse where e.idEntreprise =
:idEntreprise")

int updateEntrepriseByAdresse(@Param("adresse") String adresse,
@Param("idEntreprise")

Long idEntreprise);
```

JPQL : DELETE

- Cette méthode permet de supprimer les entreprises qui ont une adresse donnée :

- **JPQL :**

```
@Modifying
@Query("DELETE FROM Entreprise e WHERE e.adresse= :adresse")
int deleteEntreprisebyadresse(@Param("adresse") String adresse);
```

C'est équivalent à :

```
@Modifying
@Query("DELETE FROM Entreprise e WHERE e.adresse= ?1")
int deleteFournisseurByCategorieFournisseur(String adresse);
```

JPQL : INSERT

- Cette méthode permet d'insérer des projets dans la table T_Projet:

- **JPQL :** Nous utilisons Spring Data JPA. Or INSERT ne fait pas partie des spécifications JPA. Donc, nous sommes obligés d'utiliser les Natives Query pour le INSERT.
- **Pas de JPQL pour les requêtes INSERT.**

- **Native Query (SQL et non JPQL) :**

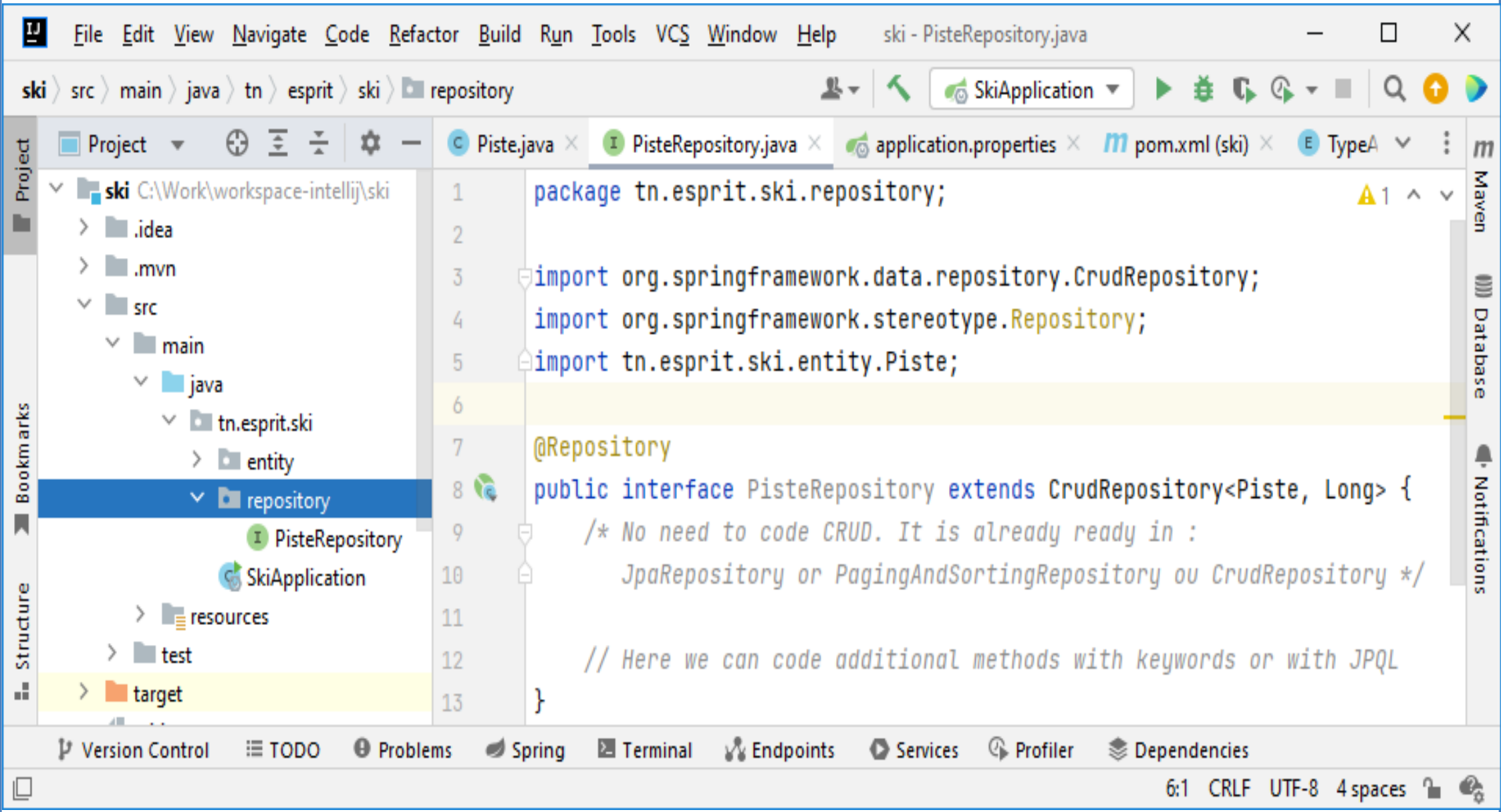
```
@Modifying
@Query(value = "INSERT INTO T_Projet(projet_sujet) VALUES (:projetsujet)",
   nativeQuery = true)
void insertProjet(@Param("projetsujet") String projetsujet);
```

Travail à faire

Exemple :

```
package tn.esprit.ski.repository;

import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;
import tn.esprit.ski.entity.Piste;

@Repository
public interface PisteRepository extends CrudRepository<Piste, Long> {
    /* No need to code CRUD. It is already ready in :
       JpaRepository or PagingAndSortingRepository ou CrudRepository */

    // Here we can code additional methods with keywords or with JPQL
}
```

Exemple Repository de l'entité Piste



Travail à faire

Partie 3 Spring Data JPA – CrudRepository + Kywords + JPQL

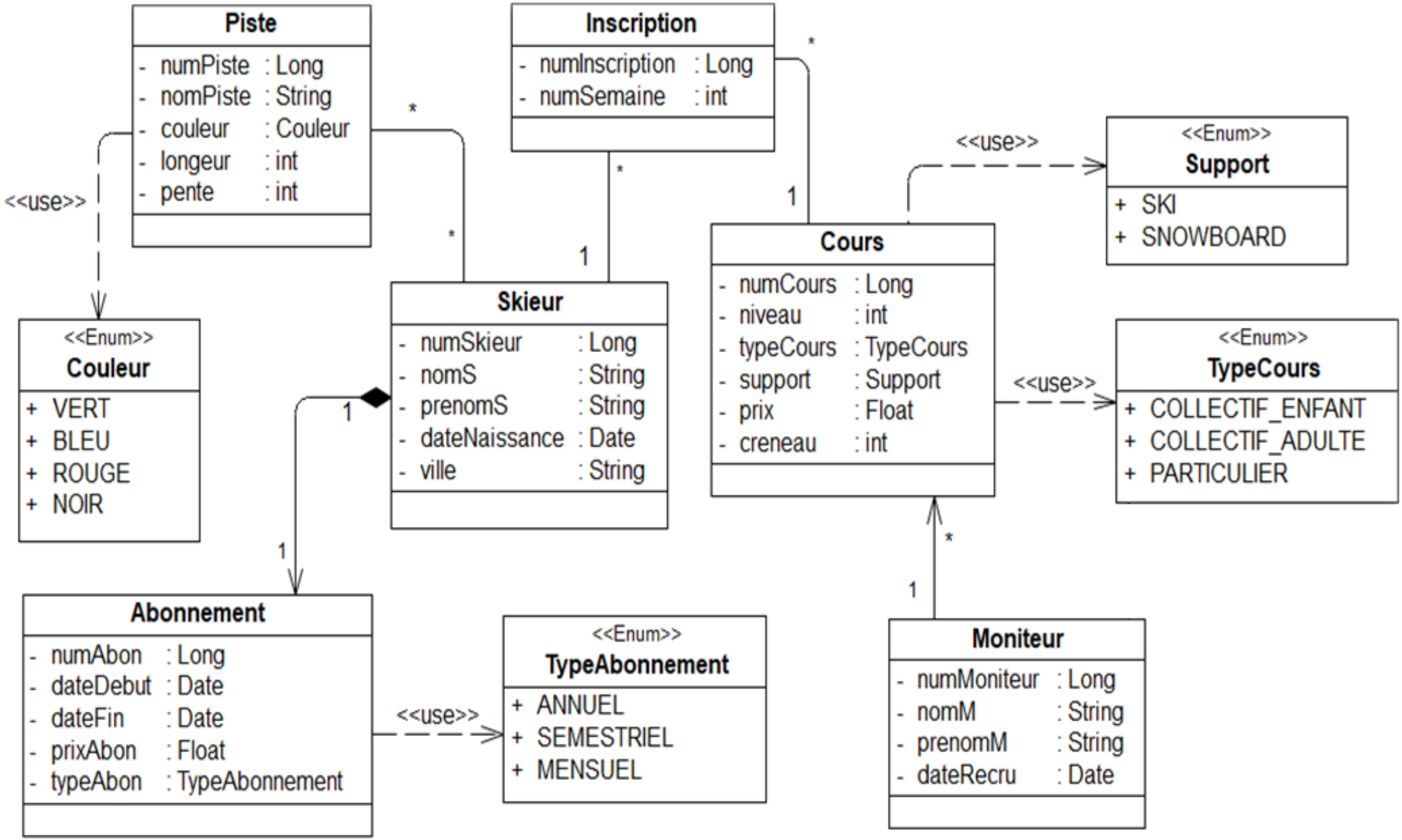
Dans l'étude de cas Station de Ski et après avoir créer les entités et leur associations :

- Créer le package repository
- Coder le repository de chaque entité
- Créer une méthode avec keyword : **liste des pistes avec une pente inférieure à une valeur donnée et une longueur supérieure à une valeur donnée.**
- Créer une méthode avec JPQL :**liste des pistes qui ont des skieurs inscrits dans cours de type SNOWBOARD.**

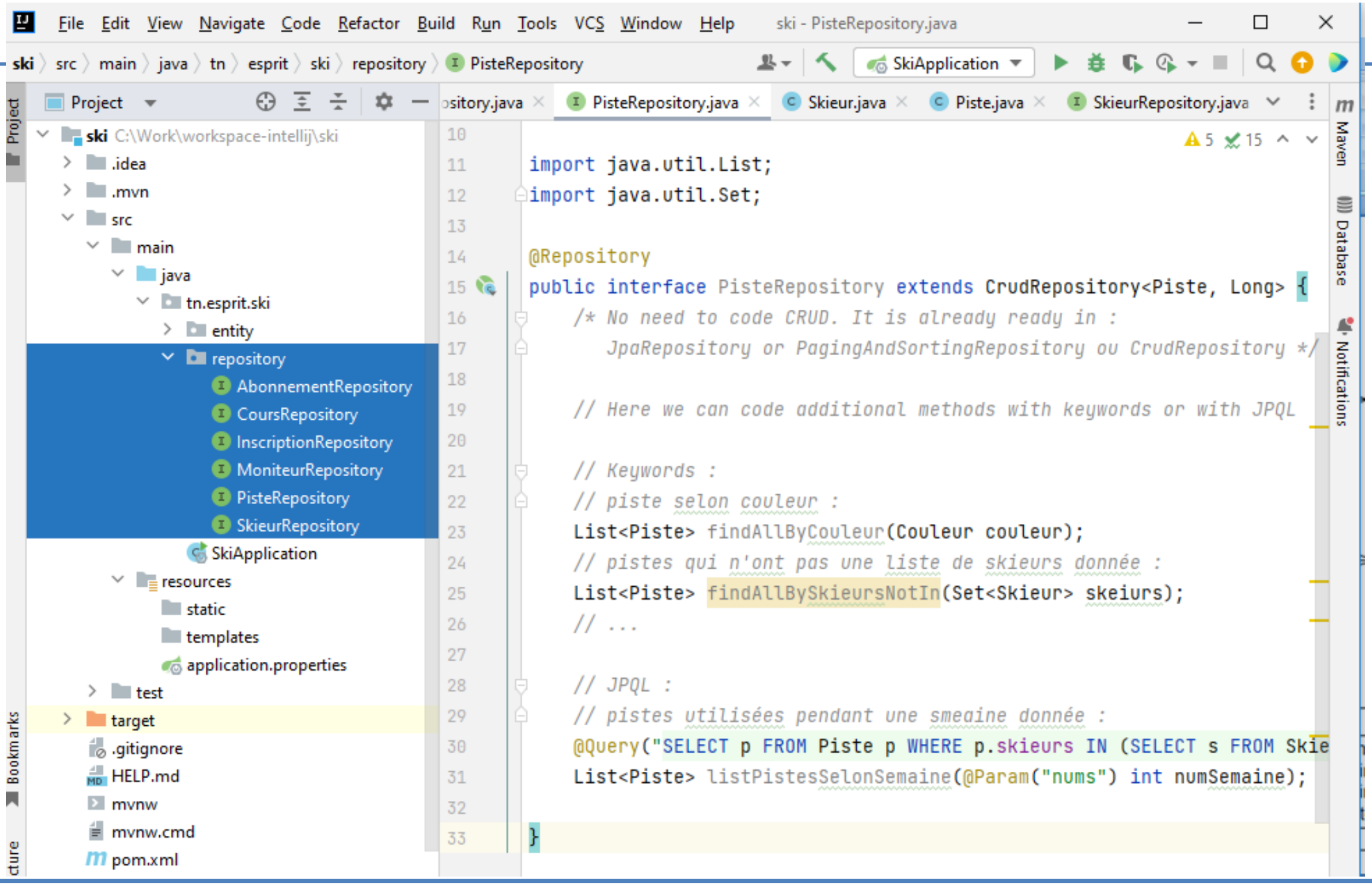
- **Attention : si la syntaxe des méthodes (Keyword ou JPQL) est fausse, alors le projet ne démarrera pas correctement (voir a console si des erreurs).**

(voir diagramme page suivante)

Diagrammes de Classes



Résultat


File Edit View Navigate Code Refactor Build Run Tools VCS Window Help
ski - PisteRepository.java
ski src main java tn esprit ski repository PisteRepository
SkiApplication
Project
ski C:\Work\workspace-intellij\ski
.idea
.mvn
src
main
java
tn.esprit.ski
entity
repository
AbonnementRepository
CoursRepository
InscriptionRepository
MoniteurRepository
PisteRepository
SkieurRepository
SkiApplication
resources
static
templates
application.properties
test
target
.gitignore
HELP.md
mvnw
mvnw.cmd
pom.xml
PisteRepository.java
Skieur.java
Piste.java
SkieurRepository.java
import java.util.List;
import java.util.Set;
@Repository
public interface PisteRepository extends CrudRepository<Piste, Long> {
/* No need to code CRUD. It is already ready in :
JpaRepository or PagingAndSortingRepository ou CrudRepository */
// Here we can code additional methods with keywords or with JPQL
// Keywords :
// piste selon couleur :
List<Piste> findAllByCouleur(Couleur couleur);
// pistes qui n'ont pas une liste de skieurs donnée :
List<Piste> findAllBySkieursNotIn(Set<Skieur> skeiurs);
// ...
// JPQL :
// pistes utilisées pendant une smeaine donnée :
@Query("SELECT p FROM Piste p WHERE p.skieurs IN (SELECT s FROM Skie
List<Piste> listPistesSelonSemaine(@Param("nums") int numSemaine);
}
Maven
Database
Notifications
Bookmarks