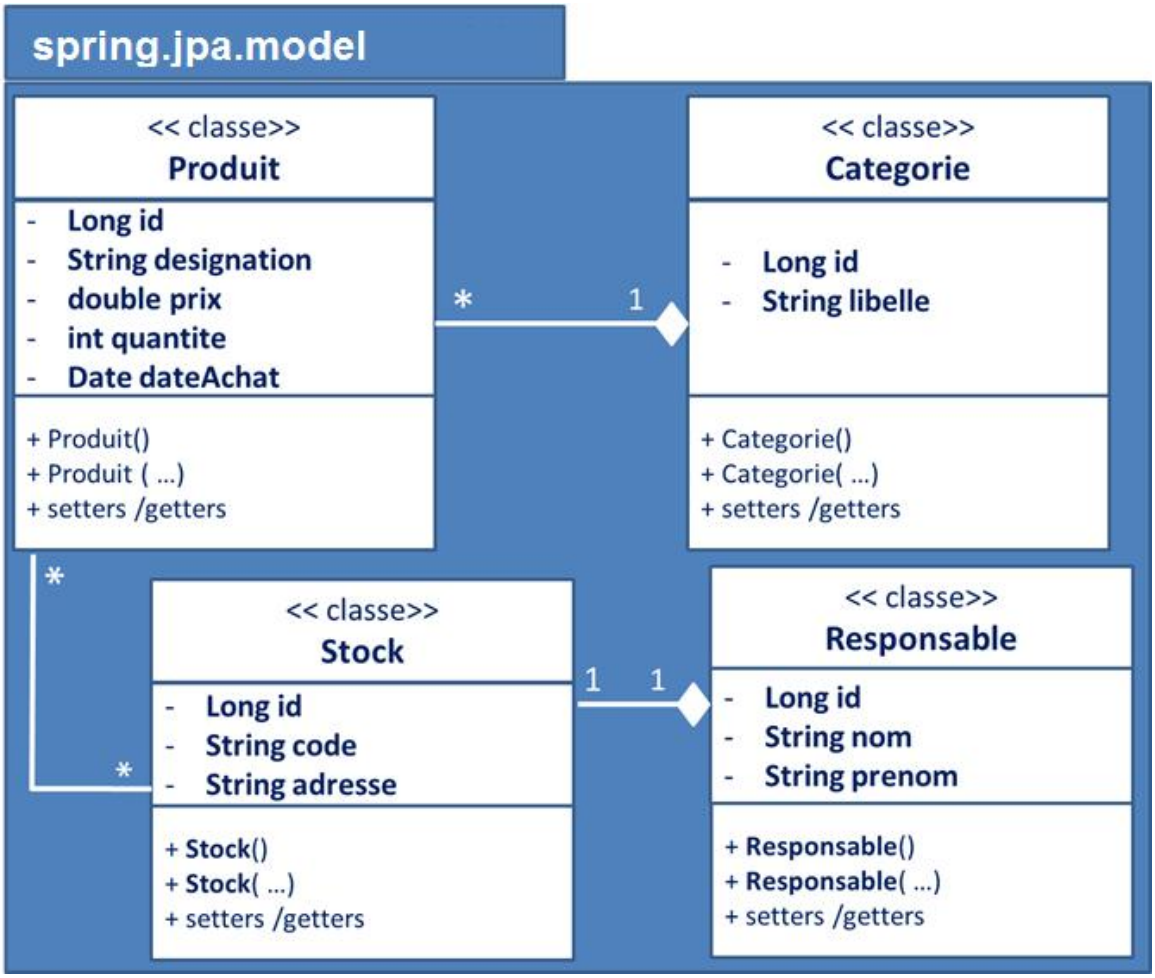
Atelier Spring -03-
JPA –Partie08

Objectifs

- **JPA et Spring Data**
  - Manipuler les relations entre les tables
  - **@OneToOne**
  - Comportement de cascade
  - Mode de récupération de collection

Voici un modèle conceptuel qui modélise une relation **1-1** une classe «**Responsable**» et une classe «**Stock**» :



1. Prendre une copie du projet «**jpa-spring-data-spring-boot-2-relations-2**» et le nommer «**jpa-spring-data-spring-boot-2-relations-3**»

2. Créer une nouvelle entité «**Responsable**» ayant le code suivant :

```
package spring.jpa.model;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.Id;
import jakarta.persistence.OneToOne;

@Entity
public class Responsable {
	@Id
	@GeneratedValue
	private Long id;

	@Column(length = 50)
	private String nom;

	@Column(length = 50)
	private String prenom;

	@OneToOne (mappedBy = "responsable")
	private Stock stock;

	public Responsable(String nom, String prenom) {
		super();
		this.nom = nom;
		this.prenom = prenom;
	}

	public Long getId() {
		return id;
	}
	public void setId(Long id) {
		this.id = id;
	}
	public String getNom() {
		return nom;
	}
	@Override
	public String toString() {
		return "Responsable [id=" + id + ", nom=" + nom + ", prenom=" +
prenom + ", stock=" + stock + "]";
	}
```

```
	public Responsable(String nom, String prenom, Stock stock) {
		super();
		this.nom = nom;
		this.prenom = prenom;
		this.stock = stock;
	}
	public Responsable() {
		super();
		// TODO Auto-generated constructor stub
	}
	public void setNom(String nom) {
		this.nom = nom;
	}
	public String getPrenom() {
		return prenom;
	}
	public void setPrenom(String prenom) {
		this.prenom = prenom;
	}
	public Stock getStock() {
		return stock;
	}
	public void setStock(Stock stock) {
		this.stock = stock;
	}
}
```

3. Remplacer le code de l'entité «**Stock**» par le code suivant :

```
package spring.jpa.model;

import java.util.ArrayList;
import java.util.Collection;
import jakarta.persistence.CascadeType;
import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.Id;
import jakarta.persistence.ManyToMany;
import jakarta.persistence.OneToOne;

@Entity
public class Stock
{
	@Id
	@GeneratedValue
	private Long id;

	@Column(length = 50)
	private String code;

	@Column(length = 50)
	private String adresse;
```

```
	@ManyToMany (mappedBy = "stocks", cascade = CascadeType.REMOVE)
	private Collection<Produit> produits = new ArrayList<Produit>();

	@OneToOne (cascade= {CascadeType.PERSIST})
	private Responsable responsable;

	public String getCode() {
		return code;
	}

	public Responsable getResponsable() {
		return responsable;
	}

	public void setResponsable(Responsable responsable) {
		this.responsable = responsable;
	}

	public Stock(String code, String adresse, Responsable responsable) {
		super();
		this.code = code;
		this.adresse = adresse;
		this.responsable = responsable;
	}
	public void setCode(String code) {
		this.code = code;
	}

	public Long getId() {
		return id;
	}
	public void setId(Long id) {
		this.id = id;
	}
	public Collection<Produit> getProduits() {
		return produits;
	}
	public void setProduits(Collection<Produit> produits) {
		this.produits = produits;
	}
	@Override
	public String toString() {
		return "Stock [id=" + id + ", code=" + code + ", adresse=" + adresse + "]";
	}
	public Stock() {
		super();
		// TODO Auto-generated constructor stub
	}
	public Stock(String code, String adresse) {
		super();
		this.code = code;
		this.adresse = adresse;
	}
	public String getAdresse() {
		return adresse;
	}
	public void setAdresse(String adresse) {
		this.adresse = adresse;
	}
}
```

- Remarquer la mise en place d'une relation **1-1** via l'instruction suivante dans l'entité «**Stock**» (**Entité maître**):

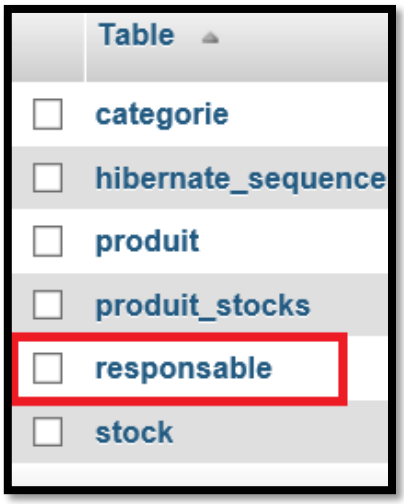
```
@OneToOne (cascade= {CascadeType.PERSIST})
private Responsable responsable;
```

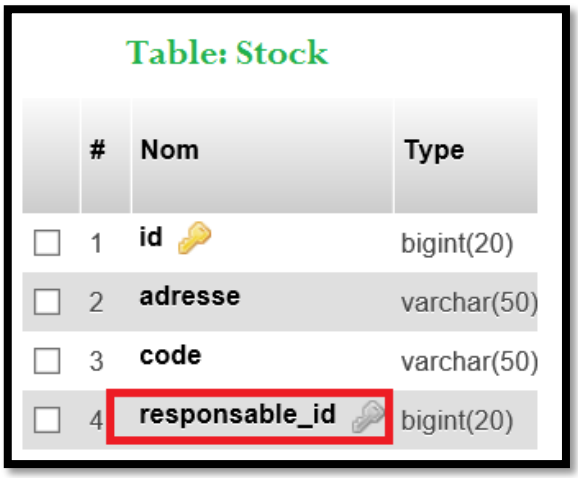
- L'attribut «**responsable**» est un attribut de relation avec l'entité «**Responsable**»
- **cascade= {CascadeType.PERSIST}** : pour réaliser l'insertion par cascade.
- De l'autre côté, une relation **1-1** est déclarée dans l'entité «**Responsable**» via l'instruction :

```
@OneToOne (mappedBy = "responsable")
private Stock stock;
```

- **Il s'agit bien d'une relation bidirectionnelle**
- L'attribut « **mappedBy** » dans l'annotation « **@OneToOne** » permet de référencer la relation dans l'entité « **Stock** ».

4. Lancer l'exécution du projet et remarquer la génération d'une clé étrangère « **responsable_id** » dans la table «**Stock**».





5. Passer, maintenant, pour réaliser des traitements sur les données. Créer une interface «**ResponsableRepository**» qui hérite de l'interface «**JpaRepository**» ayant le code suivant :

```
package spring.jpa.repository;

import org.springframework.data.jpa.repository.JpaRepository;

import spring.jpa.model.Responsable;

public interface ResponsableRepository extends JpaRepository<Responsable, Long> {}
```

6. Prendre une nouvelle version de la classe « **JpaSpringBootApplication** » pour réaliser **uniquement** les traitements suivants :

✓ Déclarer deux références pour «**StockRepository**» et «**ResponsableRepository**» (comme des attributs en dehors de la méthode « **main** »):

```
   //Déclaration d'un repository pour gérer les stocks
     static StockRepository stockRepos;
    //Déclaration d'un repository pour gérer les responsables
     static ResponsableRepository responsableRepos;
```

✓ Référencer le contexte (dans la méthode «**main**»)

```
// référencer le contexte
         ApplicationContext contexte =
SpringApplication.run(JpaSpringBootApplication.class, args);
```

✓ Récupérer des implémentations par injection de dépendance

```
// Récupérer une implémentation de l'interface "StockRepository" par injection de dépendance
stockRepos = contexte.getBean(StockRepository.class);
// Récupérer une implémentation de l'interface "ResponsableRepository" par injection de dépendance
responsableRepos = contexte.getBean(ResponsableRepository.class);
```

✓ Insérer un responsable («**Ali**», «**Ben Saleh**») et lui affecter à un stock («**1**», «**Tunis**»)

```
          Responsable r1 = new Responsable ("Ben Saleh", "Ali");
          Stock s1 = new Stock("1", "Tunis",r1);
          stockRepos.save(s1);
```

✓ Insérer un responsable («**Omar**», «**Ben Ahmed**») et lui affecter à un stock («**2**», «**Sousse** »)

```
          Responsable r2 = new Responsable ("Ben Ahmed", "Omar");
          Stock s2 = new Stock("2", "Sousse",r2);
          stockRepos.save(s2);
```

✓ Insérer un responsable («**Samira**», «**Sallemi**») et lui affecter à un stock (« **3**», «s**fax**»)

```
Responsable r3 = new Responsable ("Sallemi", "Samira");
Stock s3 = new Stock("3", "Sfax",r3);
stockRepos.save(s3);
```

✓ Insérer un responsable (« **Zied** » , « **Zouari** ») sans lui affecter un stock

```
Responsable r4 = new Responsable ("Zouari", "Zied");
responsableRepos.save(r4);
```

✓ Insérer un stock (« **4** », « **Gabes** ») sans lui affecter un responsable

```
Stock s4 = new Stock("4", "Gabes");
stockRepos.save(s4);
```

✓ Afficher la liste des responsables (chaque responsable avec le stock qui lui est associé s'il existe)

```
//f)    Afficher la liste des responsables (chaque responsable avec le stock qui lui est associé s'il existe)
System.out.println("***********************************************");
System.out.println("Afficher tous les responsables avec leurs stocks");
Collection <Responsable> lr = responsableRepos.findAll();
for (Responsable r : lr)
{
System.out.println(r);
}
System.out.println("***********************************************");
```

✓ Lancer l'exécution du programme pour avoir l'affichage suivant :

```
***********************************************
Afficher tous les responsables avec leurs stocks
Responsable [id=2, nom=Ben Saleh, prenom=Ali, stock=Stock [id=1, code=1, adresse=Tunis]]
Responsable [id=4, nom=Ben Ahmed, prenom=Omar, stock=Stock [id=3, code=2, adresse=Sousse]]
Responsable [id=6, nom=Sallemi, prenom=Samira, stock=Stock [id=5, code=3, adresse=Sfax]]
Responsable [id=7, nom=Zouari, prenom=Zied, stock=null]
***********************************************
```

✓ Remarquer que l'insertion des responsables « **Ben Saleh** », « **Ben Ahmed** » et « **Sallemi** » est réalisée suite à l'insertion de leurs stocks correspondants (par cascade) , alors que le responsable « **Zouari** » est inséré en utilisant le repository correspondant (**ResponsableRepository**)

✓ Rendre «**Ali Ben Saleh**» responsable du stock de «**Gabes**»

```
s1.setResponsable(null);
stockRepos.save(s1);
s4.setResponsable(r1);
stockRepos.save(s4);
```

✓ Rendre « **Zied Zouari** » responsable du stock de « **Tunis** »

```
s1.setResponsable(r4);
stockRepos.save(s1);
```

✓ Afficher la liste Afficher la liste des responsables (chaque responsable avec le stock qui lui est associé s'il existe) puis lancer l'exécution :

```
*********************************************
Afficher tous les responsables avec leurs stocks
Responsable [id=2, nom=Ben Saleh, prenom=Ali, stock=Stock [id=8, code=4, adresse=Gabes]]
Responsable [id=4, nom=Ben Ahmed, prenom=Omar, stock=Stock [id=3, code=2, adresse=Sousse]]
Responsable [id=6, nom=Sallemi, prenom=Samira, stock=Stock [id=5, code=3, adresse=Sfax]]
Responsable [id=7, nom=Zouari, prenom=Zied, stock=Stock [id=1, code=1, adresse=Tunis]]
*********************************************
```