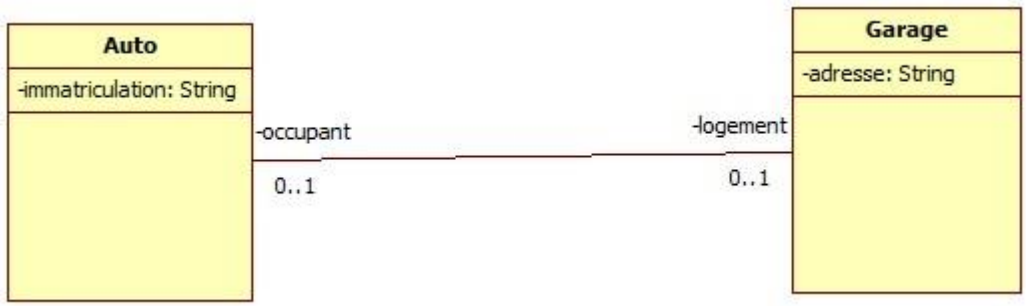
Règles de traduction en JAVA des principales notations UML pour le diagramme de classes

Protection/visibilité

- - : private
- ~ ou aucun : package (donc aucun en java)
- # : protected
- + : public

Association UN-UN :




```
import java.util.Objects;

public class Auto {

    private String immatriculation;
    private Garage logement;

    public Auto(String immatriculation) {
        this.immatriculation = immatriculation;
    }

    public String getImmatriculation() {
        return immatriculation;
    }

    public void setImmatriculation(String immatriculation) {
        this.immatriculation = immatriculation;
    }

    public Garage getLogement() {
        return logement;
    }

    public void setLogement(Garage logement) {
        this.logement = logement;
    }

    @Override
    public int hashCode() {
        int hash = 5;
```

```
        hash = 83 * hash + Objects.hashCode(this.immatriculation);
        return hash;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        }
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
        final Auto other = (Auto) obj;
        if (!Objects.equals(this.immatriculation,
other.immatriculation)) {
            return false;
        }
        return true;
    }
 }
```

```
import java.util.Objects;


public class Garage {

    private String adresse;
    private Auto occupant;

    public Garage(String adresse) {
        this.adresse = adresse;
     }

    public String getAdresse(){
        return adresse;
    }

    public void setAdresse(String adresse) {
        this.adresse = adresse;
    }

    public Auto getOccupant() {
        return occupant;
    }
```

```
    public void setOccupant(Auto occupant) {
        this.occupant = occupant;
    }

    @Override
    public int hashCode() {
        int hash = 3;
        hash = 29 * hash + Objects.hashCode(this.adresse);
        return hash;
    }

    @Override public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        }
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
        final Garage other = (Garage) obj;
        if (!Objects.equals(this.adresse, other.adresse)) {
            return false;
        }
        return true;
    }

}
```

La différence entre la multiplicité 0..1 et 1 est que dans le cas 0..1 on accepte la valeur null pour le partenaire alors que avec 1 on considère que celui-ci doit avoir une valeur effective.

Auto 1 occupant-----0..1 logement Garage :

➔

- La variable d'instance occupant d'un garage ne peut pas être nulle
- La variable d'instance logement d'une auto peut être nulle

```
package parking;
public class Garage {
protected Auto occupant;
  public void setOccupant(Auto occupant) {
      if(occupant!=null) ocuppant.logement=this;
      else if(this.occupant!=null) this.occupant.logement=null;
      this.occupant = occupant;
  }
}
```

```
package parking;
class Auto {
  protected Garage logement;
  protected Garage logement;
public void setLogement(Garage logement) {
    if(logement!=null) this.logement.occupant=this;
    else if(this.logement!=null) this.logement.occupant=null
    this.logement = logement;
}
```


Nous permet de faire : monAuto.setLogement(monGarage); OU monGarage.setOccupant(monAuto); (un des 2 suffit)
monAuto.setLogement(null); OU monGarage.setOccupant(null);

En mettant protected à des variables de classes différentes mais du même package, on peut faire occupant.logement = this par exemple, alors qu'avec uniquement des variables privées impossible (obligation de passer par les accesseurs)

Exemple de code :

```
Auto monAuto= new Auto(« ABC123 »);
Garage monGarage=new Garage(« Mons »);
monAuto.setLogement(monGarage) ;
monGarage.setOccupant(monAuto) ;
Garage tonGarage=new Garage(« BXL »);
Auto tonAuto=new Auto(« XWE234 »)
tonAuto.setLogement(tonGarage) ;
tonGarage.setOccupant(tonAuto) ;
```

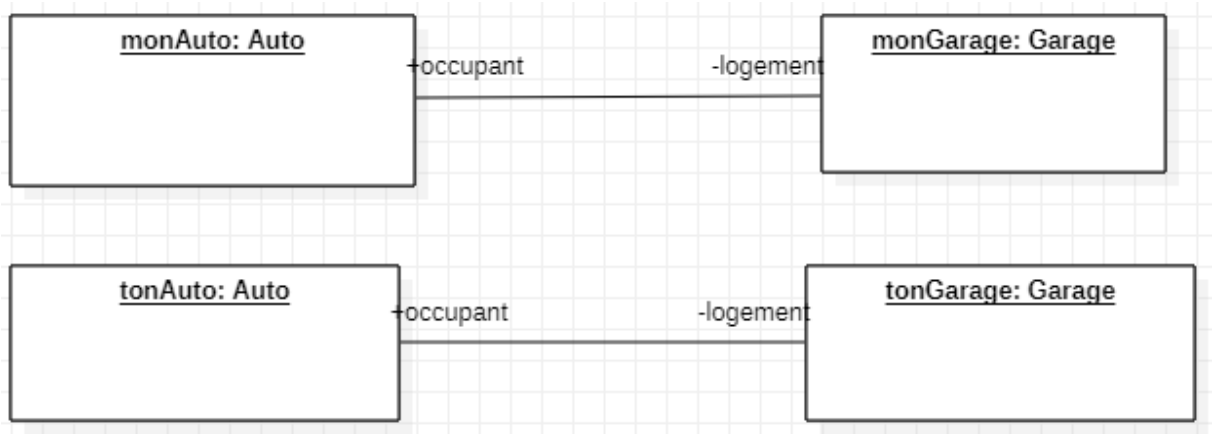
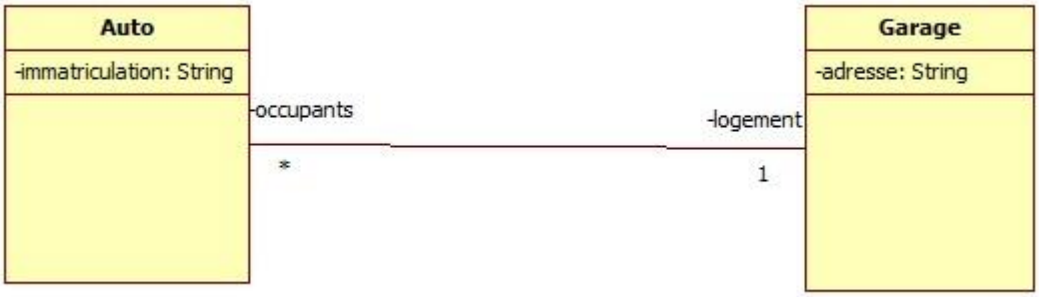


Diagramme d'objets

Association un à plusieurs



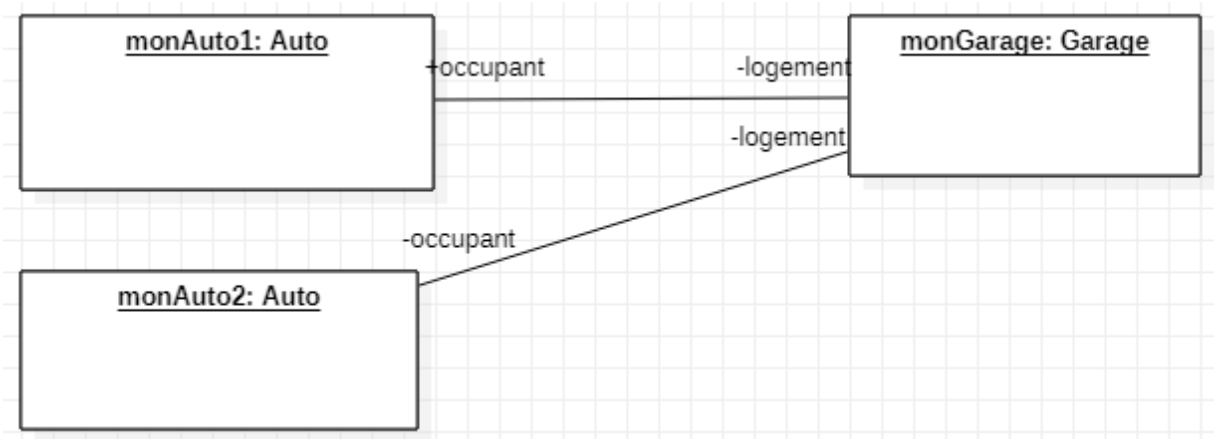
```
import java.util.ArrayList;
import java.util.Objects;
import java.util.List;

public class Garage {

   private String adresse;
   private List<Auto> occupants=new ArrayList<>();

   public Garage(String adresse) {
      this.adresse = adresse;
    }

   public String getAdresse() {
      return adresse;
   }

   public void setAdresse(String adresse) {
      this.adresse = adresse;
   }

public List getOccupants() {
      return occupants;
   }
 public void setOccupants(List<Auto> occupants) {
      this.occupants=occupants ;
   }

  @Override
   public int hashCode() {
      int hash = 3;
      hash = 29 * hash + Objects.hashCode(this.adresse);
      return hash;
   }

   @Override
   public boolean equals(Object obj) {
      if (this == obj) {
         return true;
      }
      if (obj == null) {
         return false;
      }
      if (getClass() != obj.getClass()) {
         return false;
      }
```

```
    final Garage other = (Garage) obj;
    if (!Objects.equals(this.adresse, other.adresse)) {
       return false;
    }
    return true;
  }
}
```

```
Classe Auto : inchangée
```

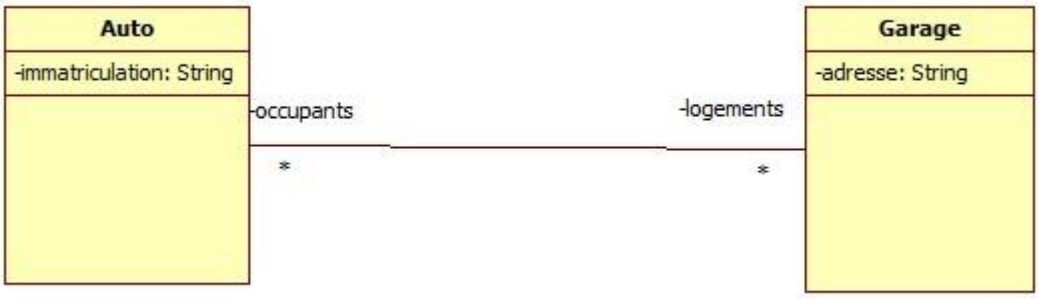
La différence entre la multiplicité *(ou 0..*) et 1..* est que le premier cas on accepte que la liste soit vide alors que dans le second elle doit contenir au moins un élément.
Exemple de code :

```
Garage monGarage=new Garage(« Mons ») ;
Auto monAuto1=new Auto(« ABC123 ») ;
monGarage.getOccupants().add(monAuto1) ;
monAuto1.setLogement(monGarage) ;
Auto monAuto2=new Auto(« FVR123 ») ;
monGarage.getOccupants().add(monAuto2) ;
monAuto2.setLogement(monGarage) ;
monGarage.getOccupants().remove(monAuto1) ;
monAuto1.setLogement(null) ;
```



RQ : l'utilisation des Sets combinés aux méthodes hashcode et equals de la classe Auto permet d'empêcher l'encodage de plusieurs autos identiques au sein de la même collection.

Association plusieurs à plusieurs



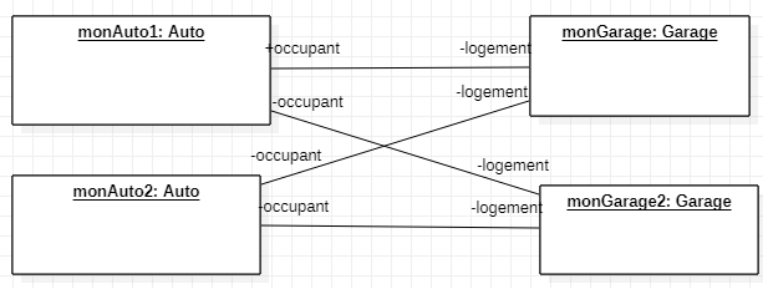
Classe Garage inchangée

```
import java.util.ArrayList;
import java.util.Objects;
import java.util.List;


public class Auto {

    private String immatriculation;
    private List<Garage> logements=new ArrayList<>();

    public Auto(String immatriculation) {
        this.immatriculation = immatriculation;
    }

    public String getImmatriculation() {
        return immatriculation;
    }

    public void setImmatriculation(String immatriculation) {
        this.immatriculation = immatriculation;
    }

    public List<Garage> getLogements() {
        return logement;
    }

    public void setLogements(List<Garage> logements) {
        this.logements=logements ;
    }


    @Override
    public int hashCode() {
        int hash = 5;
```

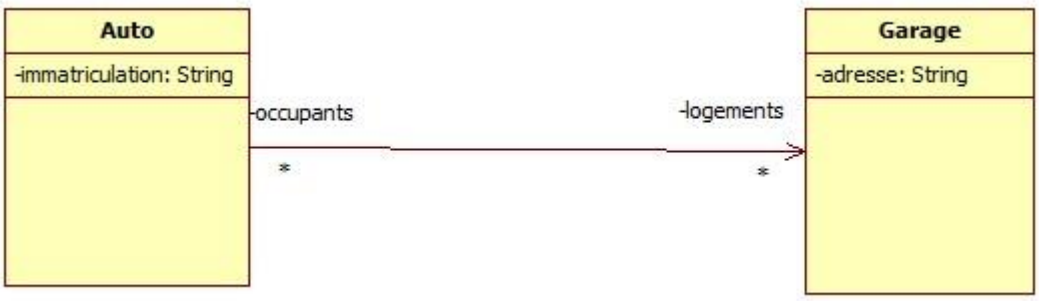
```
        hash = 83 * hash + Objects.hashCode(this.immatriculation);
        return hash;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        }
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
        final Auto other = (Auto) obj;
        if (!Objects.equals(this.immatriculation,
other.immatriculation)) {
            return false;
        }
        return true;
    }

}
```

Exemple de code

```
Garage monGarage1=new Garage(« Mons ») ;
Garage monGarage2=new Garage(« BXL ») ;
Auto monAuto1=new Auto(« ABC123 »);
Auto monAuto2=new Auto(« FVR123 ») ;
monGarage1.getOccupants().add(monAuto1) ;
monAuto1.getLogements().add(monGarage1) ;
monGarage1.getOccupants().add(monAuto2) ;
monAuto2.getLogements().add(monGarage1) ;
monGarage2.getOccupants().add(monAuto1) ;
monAuto1.getLogements().add(monGarage2) ;
monGarage2.getOccupants().add(monAuto2) ;
monAuto2.getLogements().add(monGarage2) ;
monGarage1.getOccupants().remove(monAuto1) ;
monAuto1.getLogements().remove(monGarage1) ;
```



Navigabilité

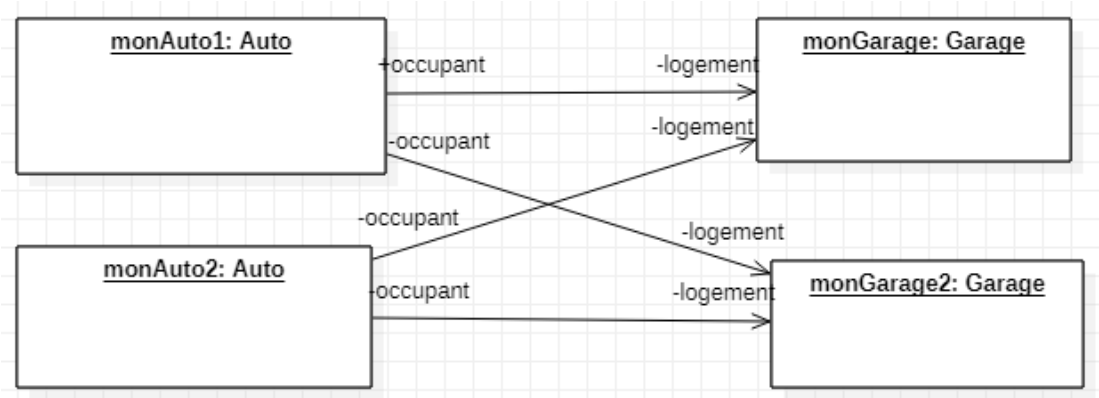


La classe Auto conserve sa référence à la classe Garage (simple ou Set), la classe Garage n'a plus aucune référence vers la classe Auto.

```
public class Garage {
     private String adresse;
     public Garage(String adresse) {
       this.adresse = adresse;
     }
      public String getAdresse() {
           return adresse;
     }
     public void setAdresse(String adresse) {
        this.adresse = adresse;
    }
      //etc + hashcode et equals
}
```

La navigabilité réduite facilite la gestion du modèle(moins de redondances à gérer) mais complexifie la recherche d'information.

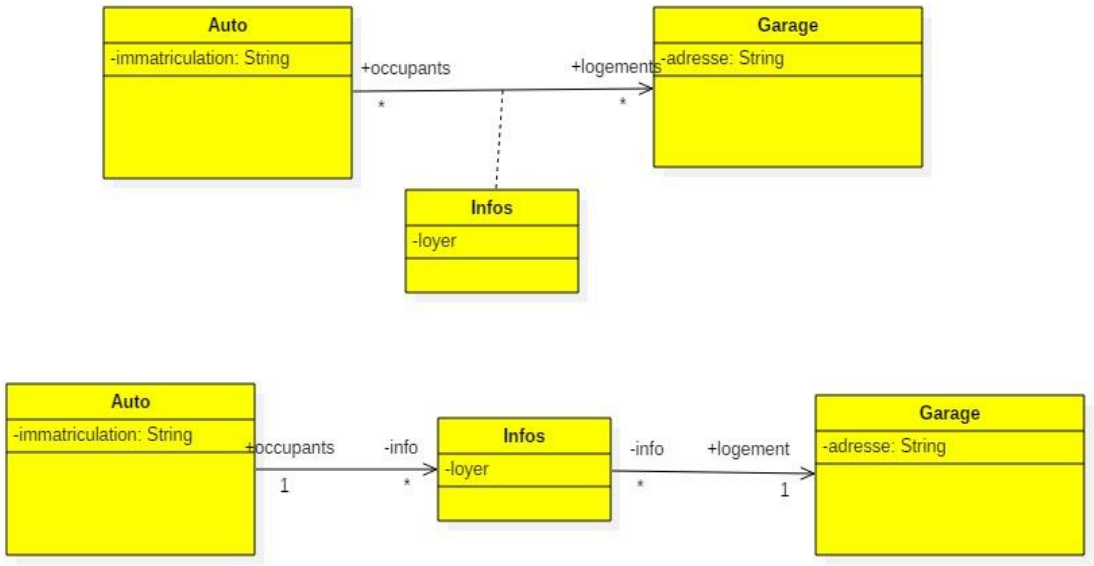
```
Garage monGarage1=new Garage(« Mons ») ;
Garage monGarage2=new Garage(« BXL ») ;
Auto monAuto1=new Auto(« ABC123 »);
Auto monAuto2=new Auto(« FVR123 ») ;
monAuto1.getLogements().add(monGarage1);
//ou setLogement(monGarage1) pour version Auto *-1 Garage
monAuto2.getLogements().add(monGarage1) ;
monAuto1.getLogements.add(monGarage2) ;
monAuto2.getLogements.add(monGarage2) ;
monAuto1.getLogements().remove(monGarage1) ;
//ou setLogement(null) pour version Auto *-1 Garage
```



Classes d'association

Nous nous intéressons principalement ici aux associations plusieurs à plusieurs navigables dans un seul sens.

Plusieurs solutions sont possibles, la plus simple est de dégrader le modèle en trois classes « classiques » comme dans le schéma ci-dessous.




```
import java.util.Objects;

public class Infos {
 private double loyer;
 private Garage logement;

    public Infos(Garage logement,double loyer) {
        this.logement = logement;
        this.loyer=loyer ;
    }
```

```
    public double getLoyer() {
        return loyer;
    }

    public void setLoyer(double loyer) {
        this.loyer = loyer;
    }

    public Garage getLogement() {
        return logement;
    }


    @Override
    public int hashCode() {
        int hash = 5;
        hash = 43 * hash + Objects.hashCode(this.logement);
        return hash;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        }
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
        final Infos other = (Infos) obj;
        if (!Objects.equals(this.logement, other.logement)) {
            return false;
        }
        return true;
    }
}
```

```
…
public class Auto {

    private String immatriculation;
    private List<Infos> info=new ArrayList<>();

    public Auto(String immatriculation) {
        this.immatriculation = immatriculation;
    }

    public String getImmatriculation() {
        return immatriculation;
```

```
    }

    public void setImmatriculation(String immatriculation) {
        this.immatriculation = immatriculation;
    }
    public List<Infos> getInfo() {
        return info;
    }
…
```

Au point de vue code, la classe Garage reste inchangée.

```
Garage monGarage1=new Garage(« Mons ») ;

Garage monGarage2=new Garage(« BXL ») ;

Auto monAuto=new Auto(« ABC123 »);

Infos info1 = new Info(monGarage1,200.0);

Infos info2 = new Info(monGarage2,300.0);

monAuto.getInfo().add(info1) ;

monAuto.getInfo().add(info2) ;

…
monAuto1.getInfos().remove(info1) ;
```