EPISEN

Ing2 2022-2023

Module ISI

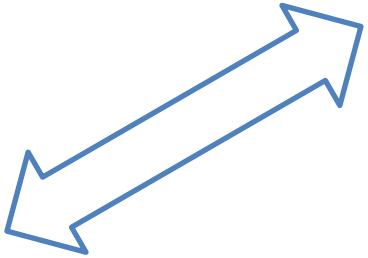
GÉNÉRATION DE CODE

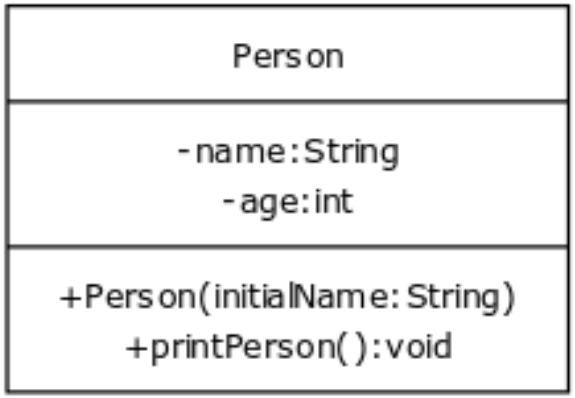
Génération de code

- Dans ce cours, on va s'intéresser à la manière dont on peut produire du code Java à partir d'un modèle UML.
- Le support principal de cette activité est le **diagramme de classes**.
- Certains outils d'édition de diagrammes UML proposent des générateurs automatiques de code, selon les principes que nous allons voir.
- **Attention :** le code ainsi produit n'est pas encore exécutable !
- Le diagramme de classes (statique) permet de générer un « squelette » de code, dont il faut encore spécifier le comportement dynamique.
- Ce comportement dynamique peut se déduire des diagrammes dynamiques.
- **Attention :** différents outils d'édition UML vont produire du code différent à partir du même modèle.

DU DIAGRAMME DE CLASSES AU CODE

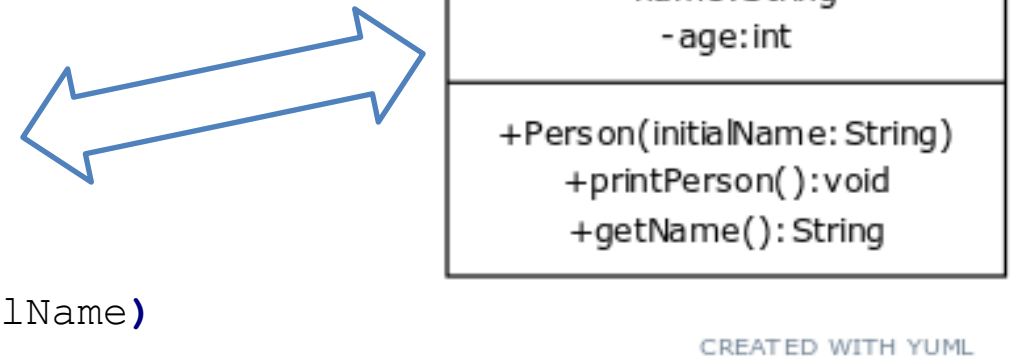
Classe, attributs et opérations





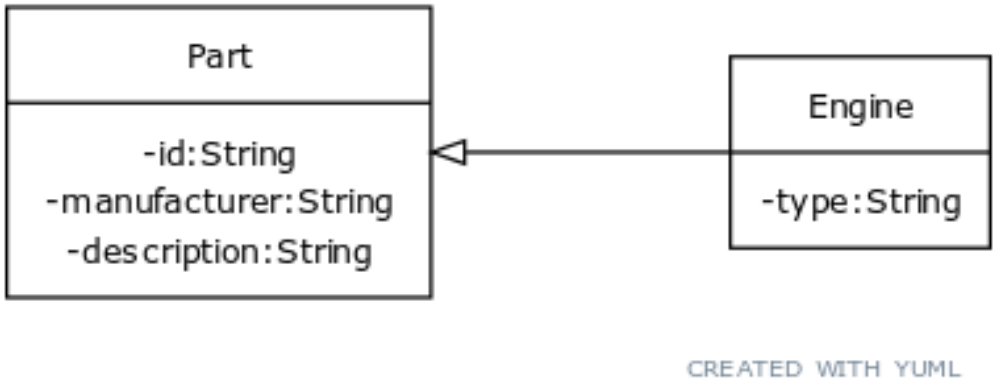

```
public class Person {
 private String name;
 private int age;
 public Person(String initialName) {
  this.name = initialName;
  this.age = 0;
 }
 public void printPerson() {
  System.out.println(this.name + ", age " + this.age + " years");
 }
}
```

Classe, attributs et opérations



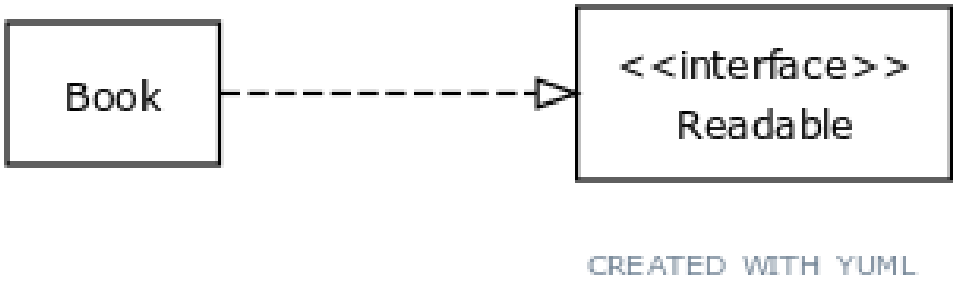

```
public class Person {
 private String name;
 private int age;
 public Person(String initialName)
{
   this.name = initialName;
   this.age = 0;
 }
 public void printPerson() {
   System.out.println(this.name + ",
age " + this.age + " years");
 }
 public String getName() {
   return this.name;
 }
}
```

Généralisation




```
public class Part{
	private String id;
	private String manufacturer;
	private String description;
}
public class Engine extends Part{
	private String type;
}
```

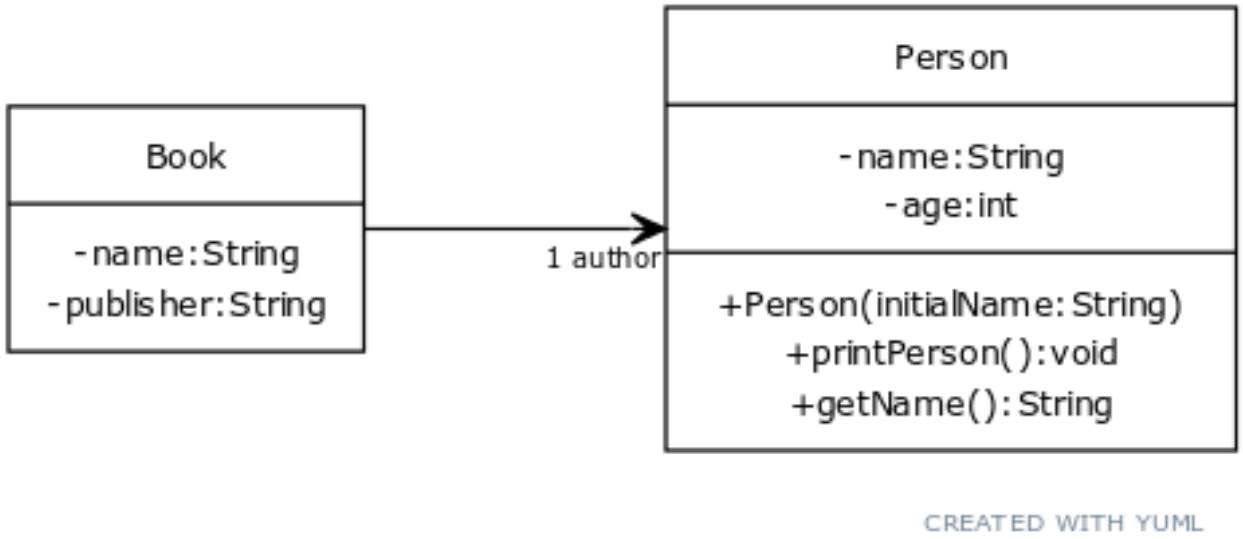
Réalisation




```
public interface Readable{
}
public class Book implements Readable{
}
```

Associations

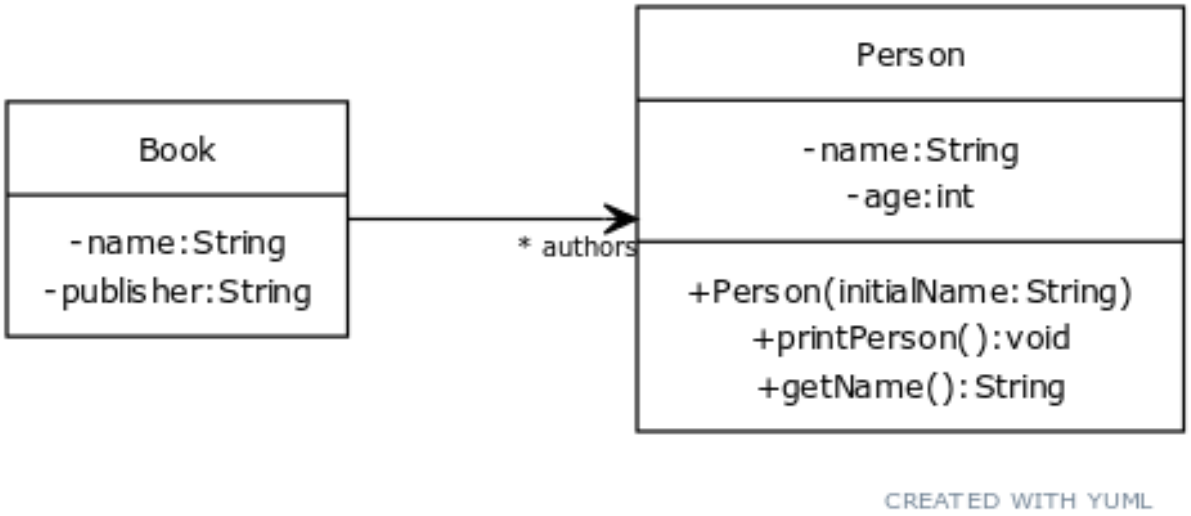
Unidirectionnelle et singulière




```java
public class Book {
 private String name;
 private String publisher;
 private Person author;
 // constructors and methods
}
```

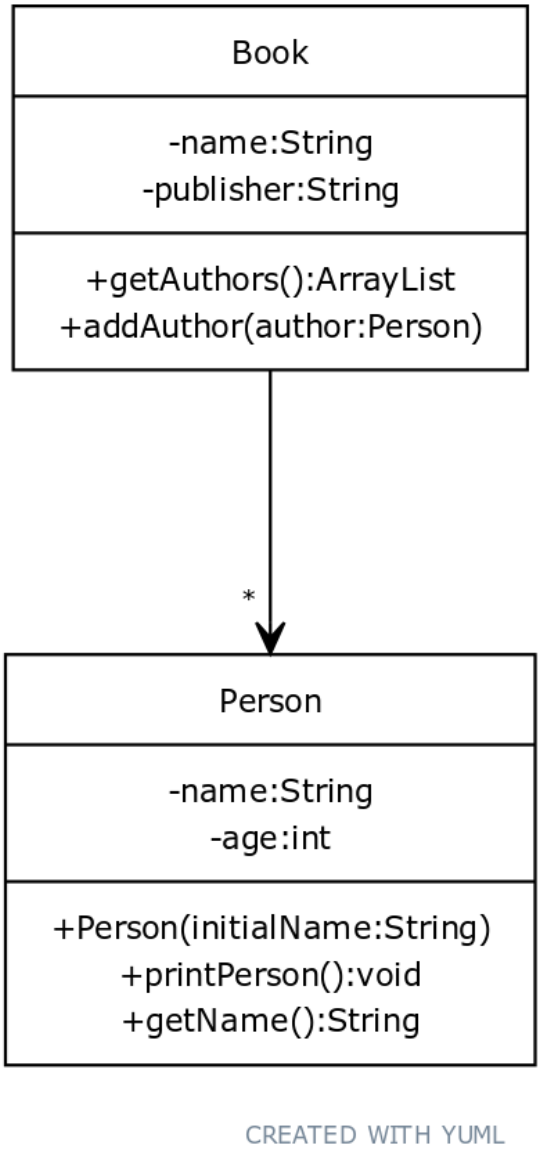
Associations

Unidirectionnelle et multiple




```java
public class Book {
 private String name;
 private String publisher;
 private ArrayList<Person> authors;
 // constructors and methods
}
```

Associations

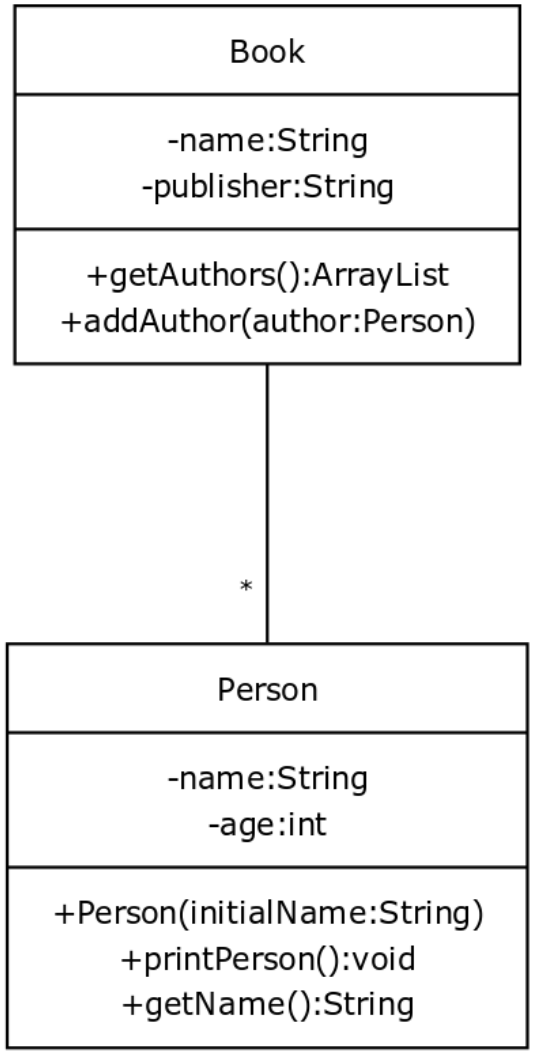


Unidirectionnelle et multiple

```
public class Book {
 private String name;
 private String publisher;
 private ArrayList<Person> authors;
 // constructor
 public ArrayList<Person> getAuthors() {
  return this.authors;
 }
 public void addAuthor(Person author) {
  this.authors.add(author);
 }
}

public class Person {
 private String name;
 private int age;
 public Person(String initialName) {…}
 public void printPerson() {…}
 public String getName() {
  return this.name;
 }
}
```

Associations

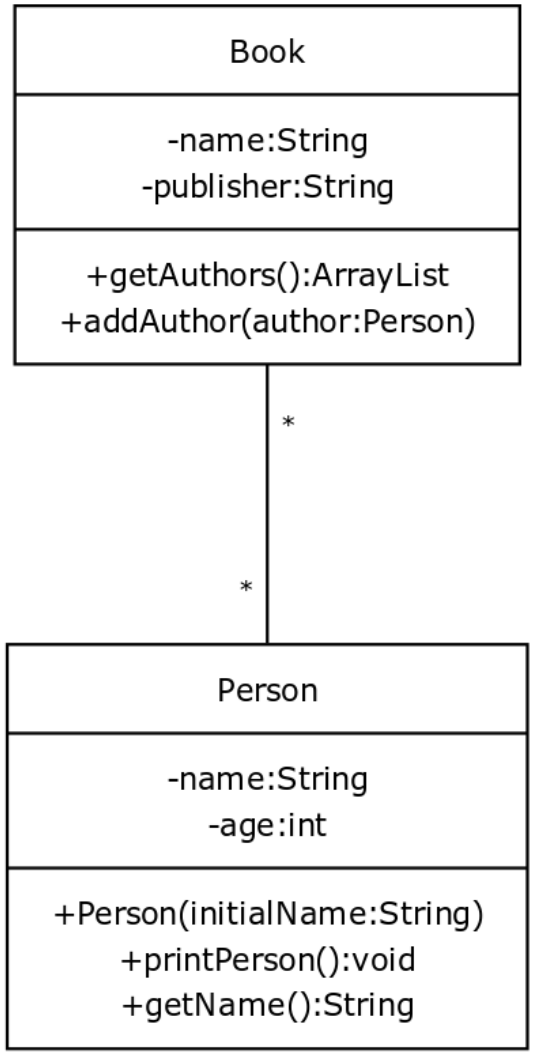


Bidirectionnelle

```java
public class Book {
 private String name;
 private String publisher;
 private ArrayList<Person> authors;
 // constructors and methods
}

public class Person {
 private String name;
 private int age;
 private Book book;
 // constructors and methods
}
```

Associations



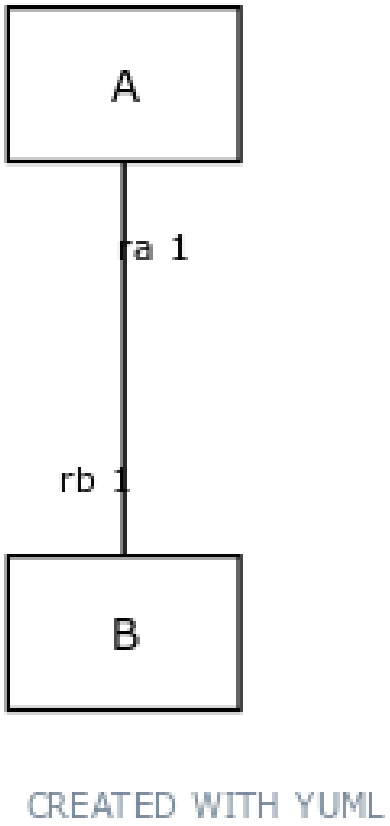
Bidirectionnelle

```java
public class Book {
 private String name;
 private String publisher;
 private ArrayList<Person> authors;
 // constructors and methods
}

public class Person {
 private String name;
 private int age;
 private ArrayList<Book> books;
 // constructors and methods
}
```

Associations

Forcer l'unicité




```
public class A {
 private B rb;
 public void addB( B b ) {
  if( b != null ){
   if ( b.getA() != null ) {
 // si b est déjà connecté à un autre A
    b.getA().setB(null);
 // cet autre A doit se déconnecter
   }
   this.setB( b );
   b.setA( this );
  }
 }
 public B getB() {
  return( rb );
 }
 public void setB( B b ) {
  this.rb=b;
 }
}
```

Agrégations et compositions

- Comme des associations normales !
- On voit donc que ces notions sont « sémantiques » et non « syntaxiques » : il n'y a pas de différences dans le code, mais il y en a une au niveau conceptuel.