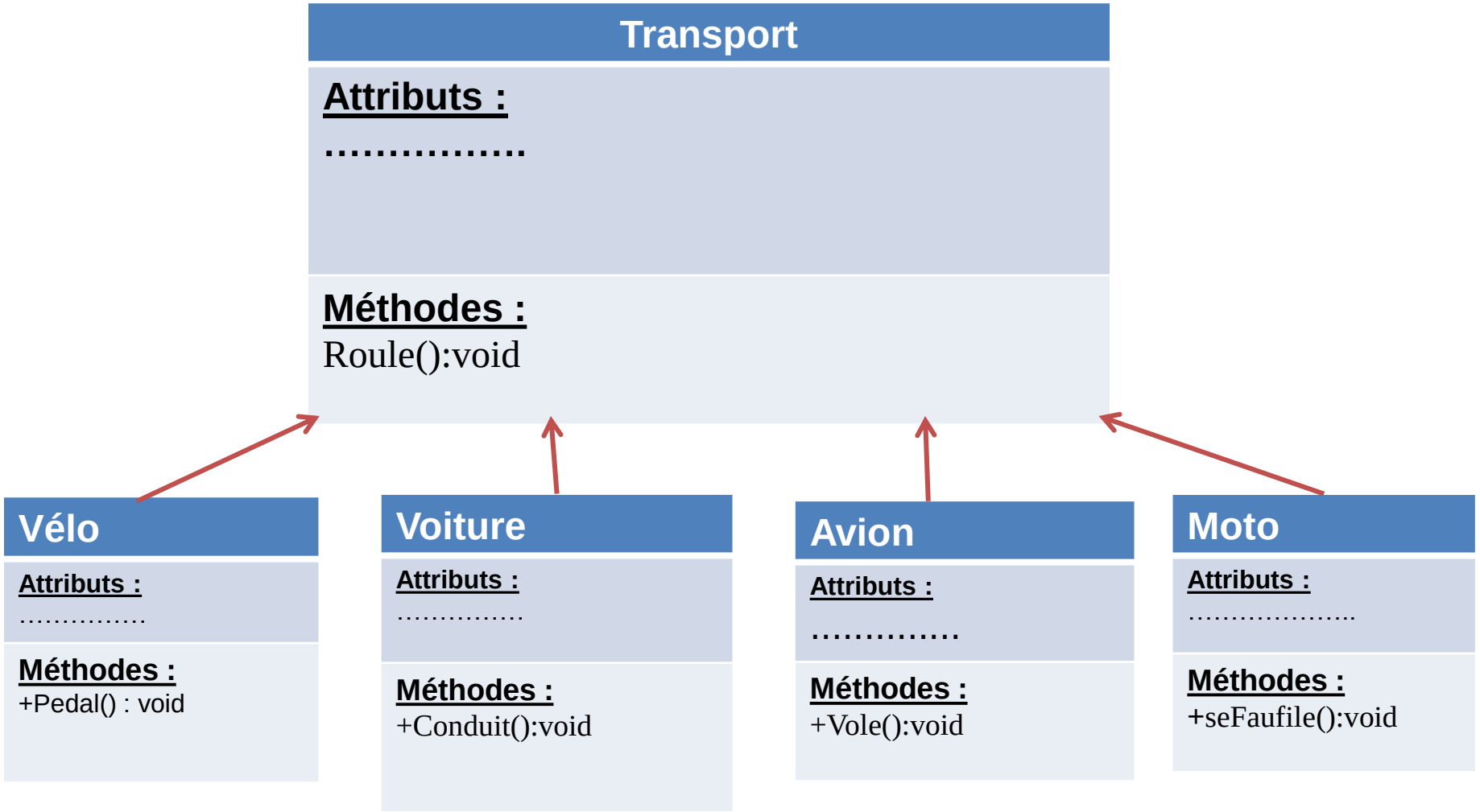
المدرسة الوطنية للعلوم التطبيقية –أكادير
ⵜⵉⵎⵙⵙⵓⵔⵉⵏ ⵜⵏⵏⴰⵙⵉⵢⵜ ⵏ ⵜⴽⵔⵉⵙⵜ ⵜⵉⵏⵉⵙⵢⵜ – ⴰⴳⴰⴷⵉⵔ.
ÉCOLE NATIONALE DES SCIENCES APPLIQUÉES - AGADIR

INTERFACES ET CLASSES ABSTRAITES

Abdellaziz Walid
a.walid@uiz.ac.ma
Département G-Info
2019-2020

Motivation : Quel est le point commun entre ces différents objets

Diagramme de classe



Implémentation java


```
public  class Transport {
     public  void roule {}
}

 public  class Voiture  extends Transport {
     public  void conduit {}
}

 public  class Avion  extends Transport {
     public  void vole {}
}

 public  class Moto  extends Transport {
     public  void seFaufile {}
}

 public  class Velo  extends Transport {
     public  void pedale {}
}
```

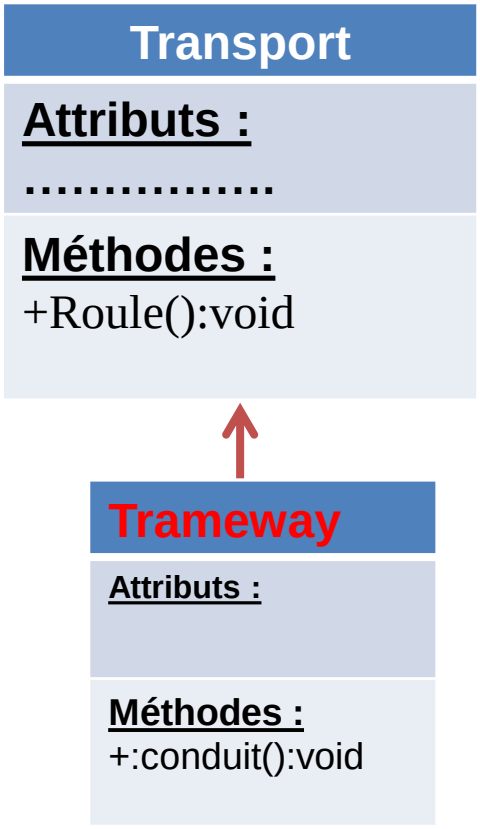
Ajout d'un nouveau service : station de service

Classe StationService et son implémentation Java

StationService
Attributs : ……
Méthodes : +faireLePlein(Transport transport) :void

```
public  class StationService {
     public  void faireLePlein(Transport transport) {
         if (transport  instanceof Velo) {
             // ne pas faire le plein
        }  else {
             // faire le plein
        }
    }
}
```

Problème : ajout d'un nouveau moyen de transport



Problème : ajout d'un nouveau moyen de transport

Supposons qu'un autre développeur reprenne le code précédent, et sans connaître l'implémentation de la classe StationService, écrive une autre extension de Transport (Trameway).

```
public  class  Trameway extends Transport {
    public  void conduit() ;
}
```

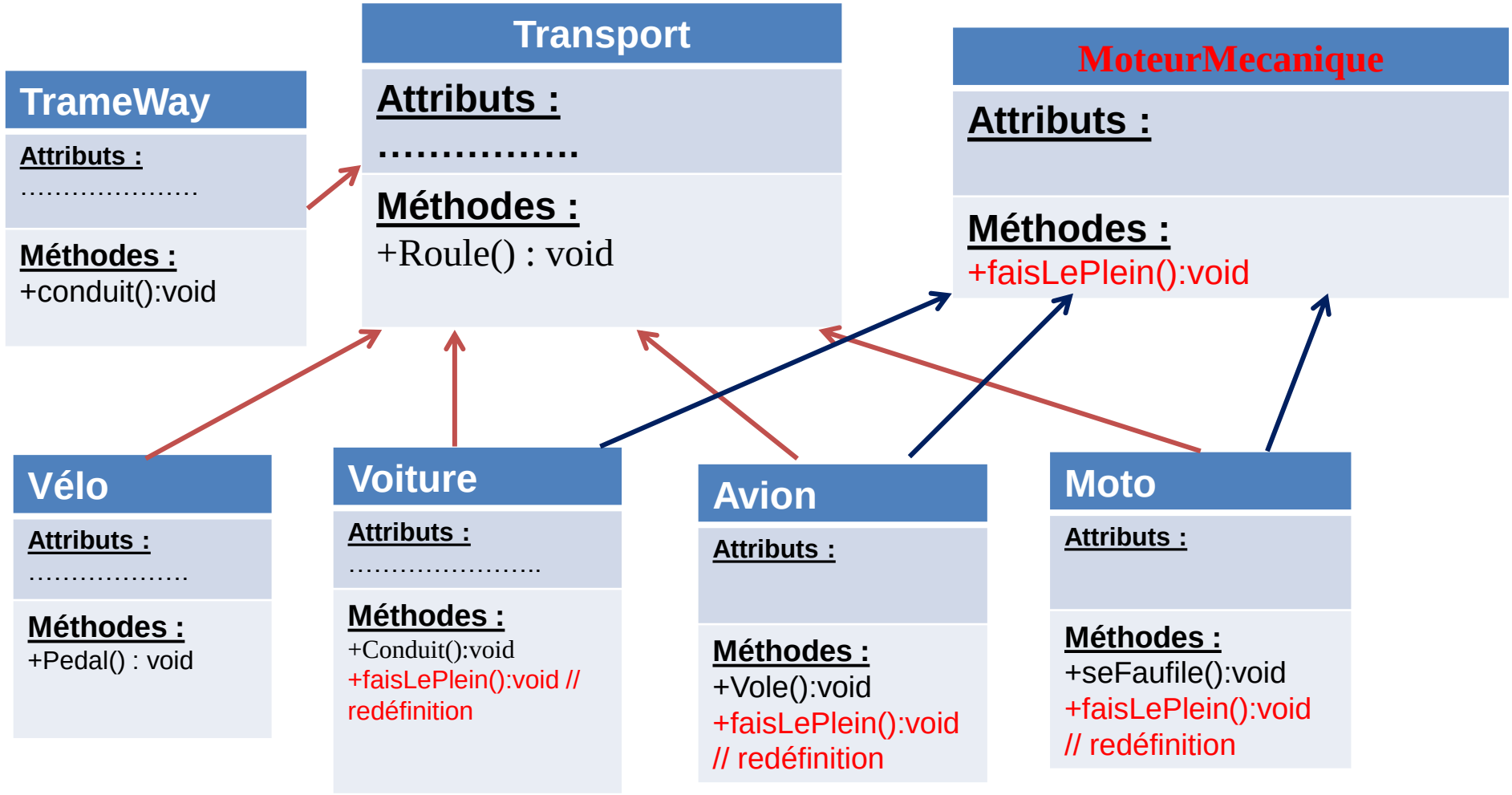
Transport T1= new Trameway();
StationService S1 = new StationService();
S1.faireLePlein(T1);

Si un trameway se présente à la station service, notre système aura un problème.

```
public  class StationService {
    public  void faireLePlein(Transport transport) {
        if (transport  instanceof Velo) {
            // ne pas faire le plein
        } else {
            // faire le plein
        }
    }
}
```

Erreur catastrophique

Comment résoudre ce problème?



Comment résoudre ce problème?


```
public   class   MoteurMecanique{
      public  void faisLePlein() ;
}
```

```
public  class StationService {
     public  void faireLePlein( MoteurMecanique moteur ) {

          moteur.faireLePlein()

     }
}
```

Problème :
Héritage multiple de classes: les sous classes (Avion, Voiture, Moto) héritent de deux classes (Transport et MoteurMecanique). Alors que l'héritage multiple de classes n'est pas supporté par Java

Pourquoi l'héritage multiple de classes n'est pas supporté par Java

Problème de l'héritage multiple de classes :

- Si on hérite de deux méthodes ayant même signature dans deux super classes, quelle code choisir ?
- L'héritage multiple est compliqué à gérer que ce soit pour le programmeur ou pour le compilateur.

Solution de Java:

- Il n'y a pas d'héritage multiple de classes en java.
- Java définit des interfaces et permet à une classe d'implémenter plusieurs interfaces.
- La notion d'interface est absolument centrale en Java, et massivement utilisée dans le design des API du JDK et de Java EE.
- Tout bon développeur Java doit absolument maîtriser ce point parfaitement.

Concept : interface

- Une interface est un type, au même titre qu'une classe, mais abstrait et qui donc ne peut être instancié (par appel à new plus constructeur).

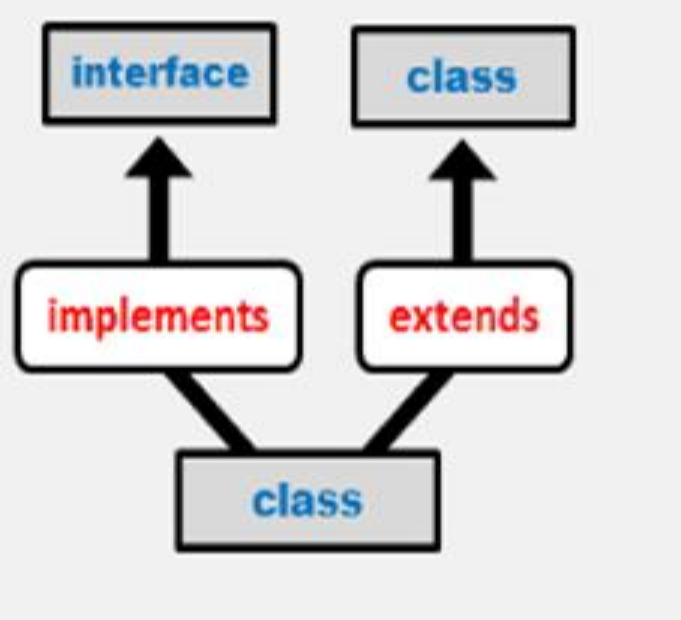
- Une interface décrit un ensemble de signatures de méthodes, sans implémentation, qui doivent être implémentées dans toutes les classes qui implémentent l'interface.

- L'utilité du concept d'interface réside dans le regroupement de plusieurs classes, tel que chacune implémente un ensemble commun de méthodes, sous un même type.

Caractéristiques de l'interface

- Une interface possède les caractéristiques suivantes :
  - elle contient des signatures de méthodes ;
  - elle ne peut pas contenir de variable (attributs) ;
  - une interface peut hériter d'une autre interface ou plusieurs interfaces (avec le mot-clé extends) ;
- Une classe concrète peut implémenter plusieurs interfaces.
- Une classe concrète qui implémente une interface doit nécessairement fournir directement ou indirectement (dans une de ses super-classes) une implémentation pour toutes les méthodes abstraites de cette interface.
- La liste des interfaces implémentées doit alors figurer après le mot-clé implements placé dans la déclaration de classe, en séparant chaque interface par une virgule.

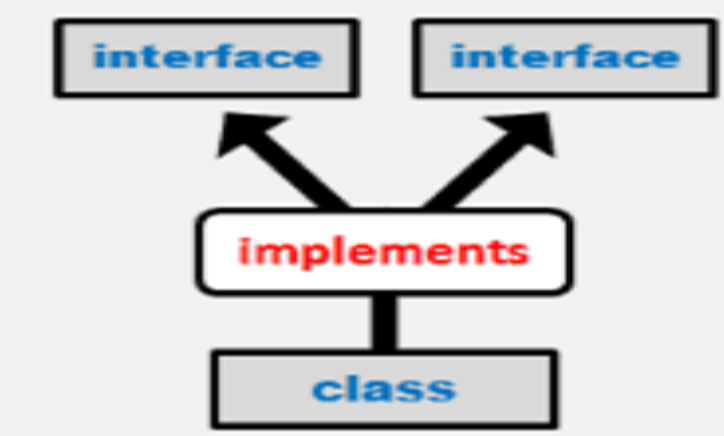
Interfaces




```
public class A extends B implements C {
  // corps de la classe
}
```

Classes : A et B C:interface

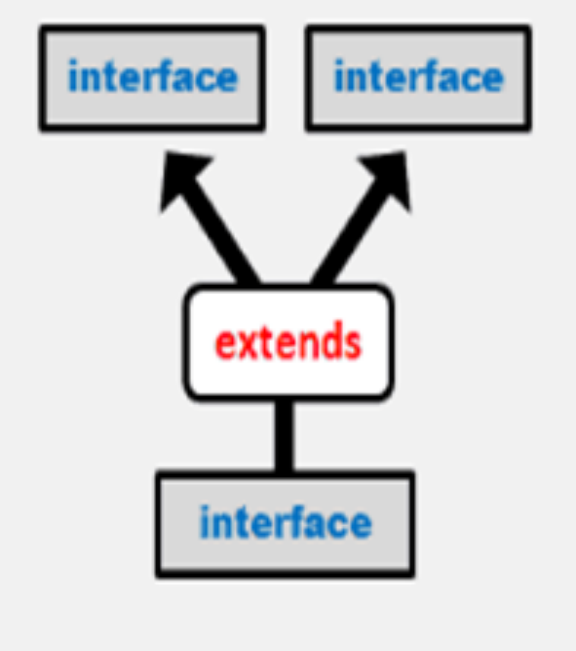
Interfaces




```
public  interface A {
      public   void a() ;


}

 public  interface B {
      public   void b() ;


}
```

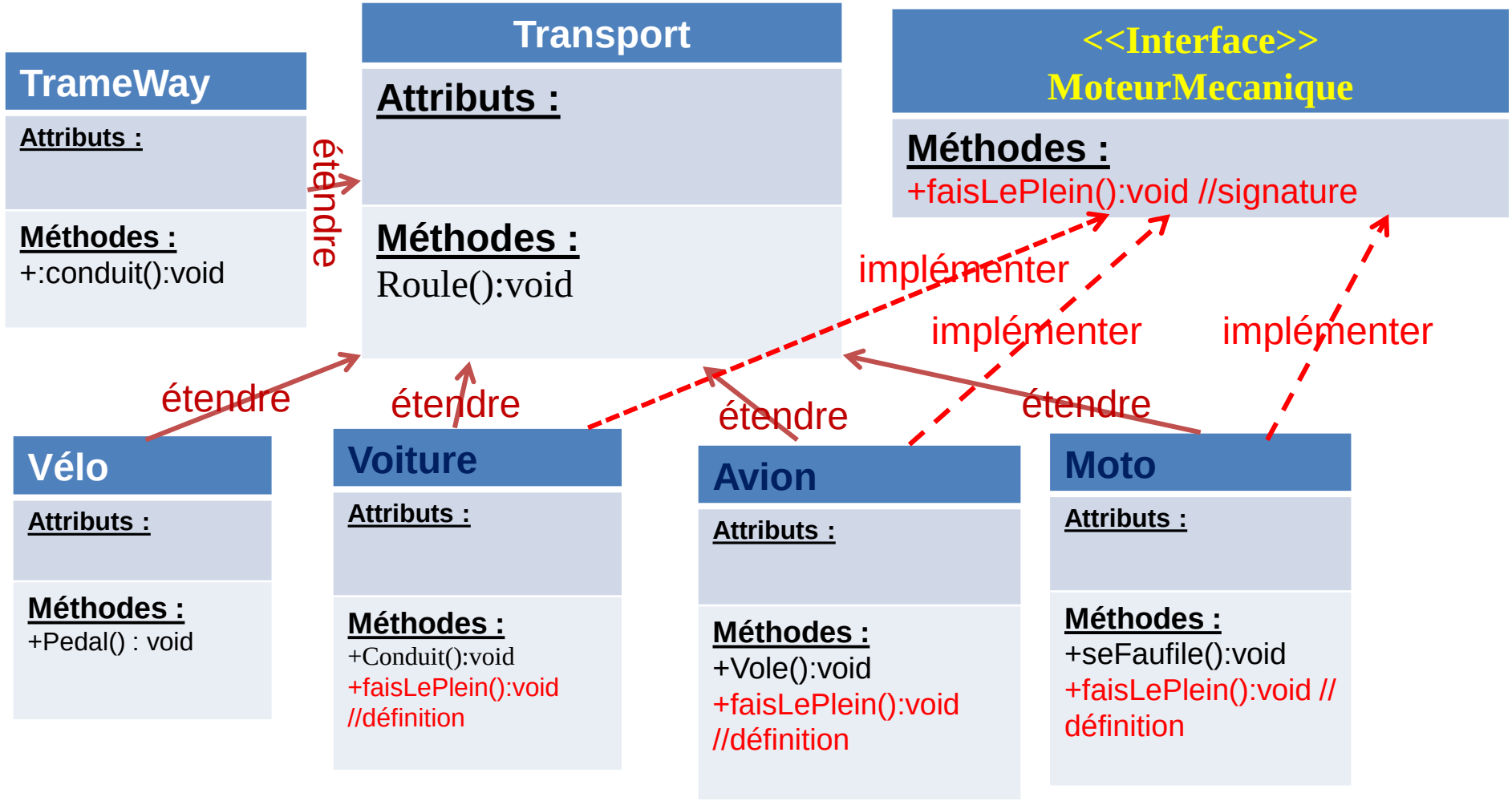
```
 public  class C  implements A, B {
     // corps de la classe
}
```

Interfaces : héritage multiple de type




```
public interface S   extends  Serializable, Comparable, CharSequence {
        // interface
}
```

Solution du problème: Les interfaces



Solution du problème : Implémentation Java


```
public  class Transport {  // une instance de Transport ne sait pas toujours
                           // faire le plein
    public  void roule() {}
}
```

```
public  interface  MoteurMecanique  {  // notre interface
    public  void faisLePlein() ;
}
```

signature

Cette fois les instances des classes qui implémentent l'interface MoteurMecanique peuvent invoquer la méthode faireLePlein.

```
public  class StationService {
    public  void faireLePlein( MoteurMecanique moteur ) {

        moteur. faisLePlein()

    }
}
```

```
public  class Voiture  extends Transport  implements  MoteurMecanique {
    public  void conduit() {}
    public  void faisLePlein() {}
}

public  class Avion  extends Transport  implements  MoteurMecanique  {
    public  void vole() ;
    public  void faisLePlein() {}
}

public  class Moto  extends Transport  implements  MoteurMecanique  {
    public  void seFaufile() ;
    public  void faisLePlein() {}
}

public  class Velo  extends Transport {  // ne sait pas faire le plein
    public  void pedale() ;
}

public  class Trameway  extends Transport {  // ne sait pas faire le plein
    public  void conduit() {}
}
```

implémentation

implémentation

implémentation

Evolution des interfaces dans Java

Java 7

- Méthode abstraite (signature)
- Constantes

Java 8

- Méthode abstraite (signature)
- Constantes
- Méthodes statiques et par défaut

Java 9

- Méthode abstraite (signature)
- Constantes
- Méthodes statiques et par défaut
- Méthodes privées
- Méthodes statiques privées

Définition de constantes dans les interfaces


```
public  interface Constantes {  // dans le fichier Constantes.java
     public  static  final  double G =  9.81 ;
}

 public  class ChampGravitationnel  // dans le fichier ChampGravitationnel.java
 implements Constantes {
     private  double vitesse ;

     public  double calculeVitesse(double temps) {
         return G*temps ;
     }
}
```

Méthode par défaut

- Une méthode par défaut permet d'écrire une méthode dans une interface, en fixant sa signature et en donnant une implémentation.
- Si une interface possède des méthodes par défaut, toute classe concrète qui l'implémente est libre de fournir sa propre implémentation de cette méthode par défaut, ou d'hériter celle de l'interface.

A partir de java 8 : Héritage multiple d'implémentation avec les méthodes par défaut

```
public  interface A {
     default  void a() {
         // implémentation
    }
}

 public  interface B {
     default  void a() {
         // implémentation
    }
}

 public  class C  implements A, B {  // !!! erreur de compilation !!!
     // corps de la classe
}
```

Problème : quelle implémentation de la méthode a() va-t-elle être invoquée ? Celle de A ? Celle de B ?

Solutions:

- Créer une implémentation concrète de la méthode a() dans la classe C qui aura la priorité sur la méthode par défaut.
- Décider qu'une des deux interfaces, par exemple A étend la seconde. Dans ce cas, A devient plus spécifique que B, et l'on dit que c'est l'implémentation la plus spécifique qui a la priorité.

Méthode par défaut

Notons qu'une implémentation peut appeler explicitement une implémentation par défaut grâce à la syntaxe suivante :

```
public  interface A {
     default  void a() {
         // implémentation
    }
}

 public  class C  implements A {
     public  void a() {
         return A.super.a() ;  // appelle la méthode par défaut de A
    }
}
```

Méthode d'interface statique

- En plus de pouvoir déclarer des méthodes *default* dans des interfaces, Java 8 nous permet de définir et d'implémenter des méthodes *static* dans des interfaces .
- Comme les méthodes *static* n'appartiennent pas à un objet particulier, elles doivent être appelées à l'aide **du nom d'interface précédant le nom de méthode.**
- Définir une méthode *static* au sein d'une interface est identique à en définir une dans une classe.
- Exemple :

```
public interface Vehicle {

   //regular/default interface methods

    static int getHorsePower(int rpm, int torque) {
        return (rpm **  torque)/5252;
    }
}
```

Exemple : Méthode par défaut et méthode statique


```java
public interface CustomInterface {

    public abstract void method1();

    public default void method2() {
        System.out.println("default method");
    }

    public static void method3() {
        System.out.println("static method");
    }
}

public class CustomClass implements CustomInterface {

    @Override
    public void method1() {
        System.out.println("abstract method");
    }

    public static void main(String[] args){
        CustomInterface instance = new CustomClass();
        instance.method1();
        instance.method2();
        CustomInterface.method3();
    }
}
```

Résultats :

abstract method
default method
static method

Méthode privée et méthode statique privée

- Le Java SE 9 a apporté une petite évolution sur la mise en œuvre d'interface : il est maintenant possible d'y définir :
  - Des méthodes privées concrètes.
  - Des méthodes statiques privées concrètes.

- Une méthode par défaut peut invoquer les méthodes privées et les méthodes statiques privées implémentées dans la même interface que celle de la méthode par défaut.

- Une méthode statique d'une interface peut invoquer seulement les méthodes privées statique du même interface.

Méthode privée et méthode statique privée


```java
public interface CustomInterface {

    public abstract void method1();

    public default void method2() {
        method4();  //private method inside default method
        method5();  //static method inside other non-static method
        System.out.println("default method");
    }

    public static void method3() {
        method5(); //static method inside other static method
        System.out.println("static method");
    }

    private void method4(){
        System.out.println("private method");
    }

    private static void method5(){
        System.out.println("private static method");
    }
}

public class CustomClass implements CustomInterface {

    @Override
    public void method1() {
        System.out.println("abstract method");
    }

    public static void main(String[] args){
        CustomInterface instance = new CustomClass();
        instance.method1();
        instance.method2();
        CustomInterface.method3();
    }
}
```

Résultats :

abstract method
private method
private static method
default method
private static method
static method

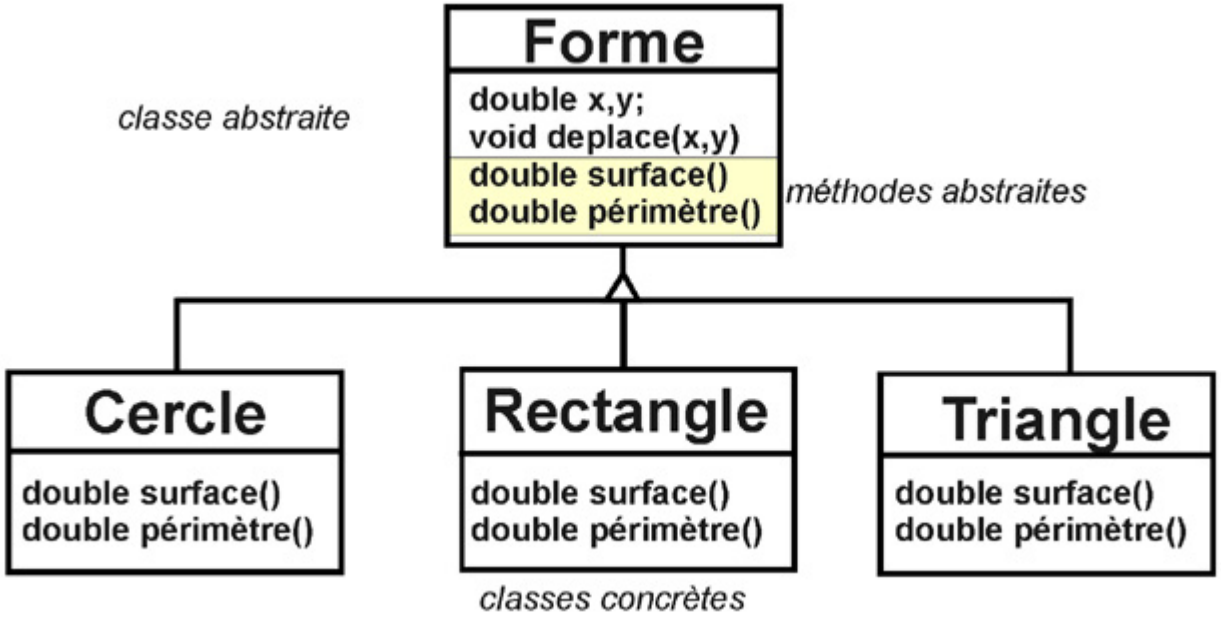
Synthèse

- Interface : Un ensemble d'exigences pour les classes.
- Toutes les méthodes d'une interface sont automatiquement publiques. (pas nécessaire de fournir le mot clé public)
- Avant Java 8, les méthodes n'étaient jamais implémentées dans les interfaces.
- Les méthodes par défaut (à partir de Java SE 8) sont héritées comme les méthodes ordinaires et implémentées dedans les interfaces.
- Les méthodes privées (à partir de Java SE 9) peuvent être invoquées seulement dedans les méthodes par défaut de l'interface où elles appartiennent.

Motivation : besoin de méthode abstraite dedans des classes = classes abstraites

Peut-on calculer et afficher la surface d'une forme sans savoir de quelle forme il s'agit ?

Réponse : Non

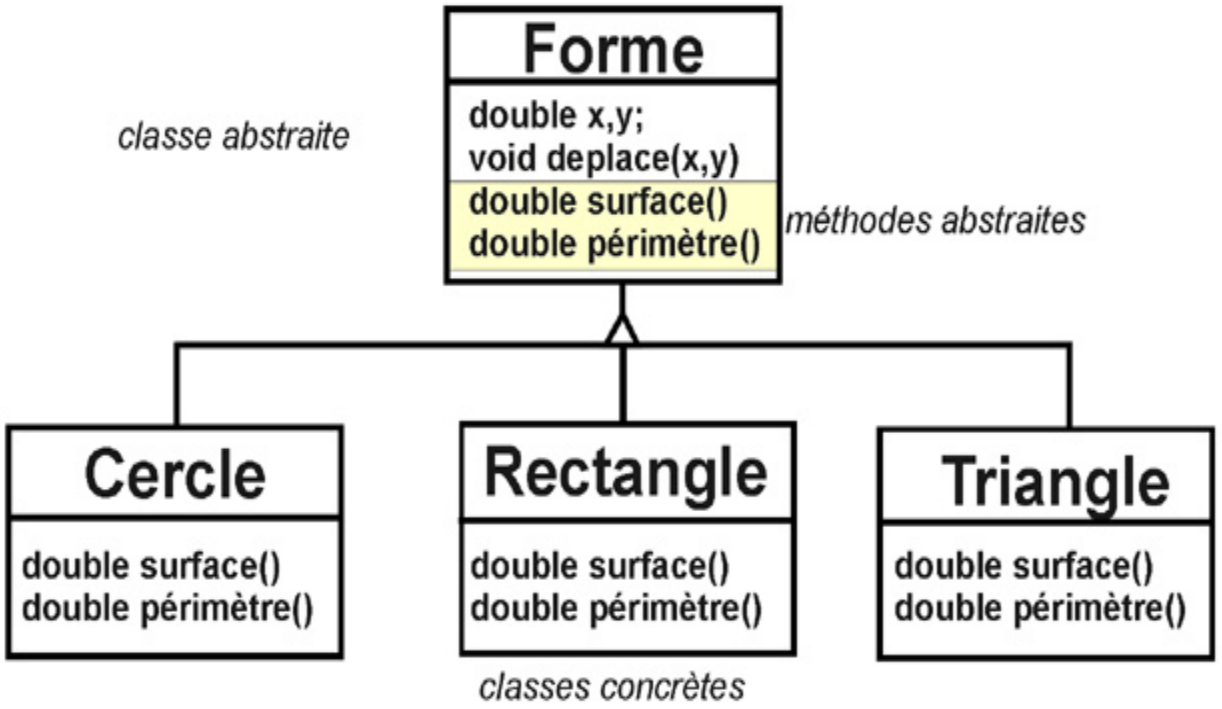


→ La classe forme est une classe abstraite car elle comporte deux méthodes abstraites c.-à-d. sans implémentation(signature)

Classe abstraite

- C’est une classe qu’on ne peut pas directement instancier car certaines de ses méthodes ne sont pas implémentées.
- Une classe abstraite peut donc contenir des variables (attributs), constructeurs, des méthodes implémentées et des signatures de méthode à implémenter.
- Une classe abstraite peut implémenter (partiellement ou totalement) des interfaces et peut hériter d’une classe ou d’une classe abstraite.
- La principale différence entre interface et classe abstraite réside dans le fait que les classes abstraites peuvent avoir des constructeurs, des états et des comportements.

Exemple : Forme est une classe abstraite



Classe abstraite : implémentation Java

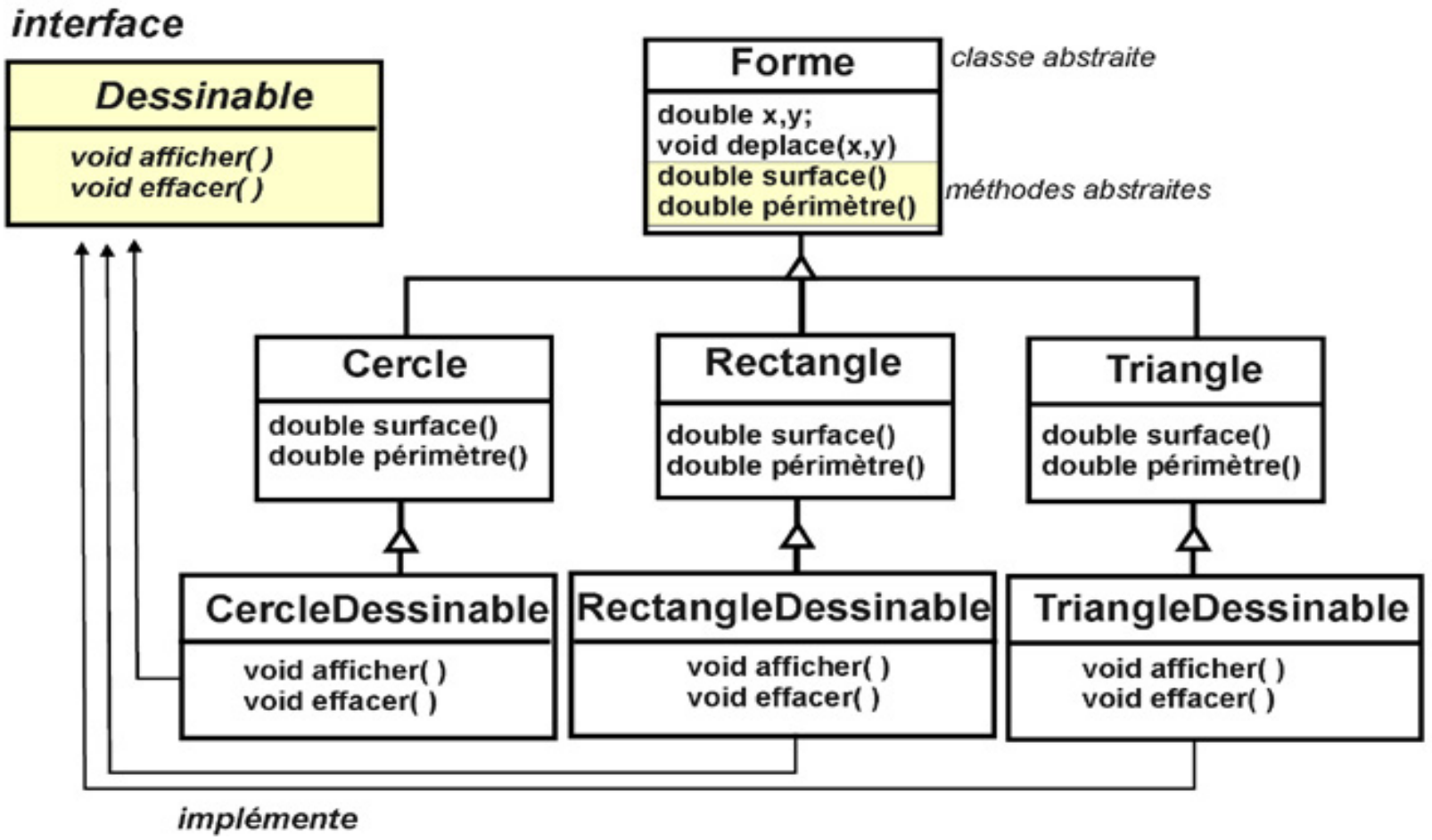

```
pubic abstract class Forme {
```
définition de la classe de base comme étant abstraite

```
  double x,y; // position de la forme
```
attributs

```
  public void deplace(double x, double y)
  {
    this.x += x; this.y +=y;
  }
  ...
```
méthode "concrète"

```
  public abstract double surface();
  public abstract double périmètre();
}
```
méthodes "abstraites"

Exemple de synthèse : Interface, Classe abstraite et Classe



Merci de votre attention