JAVA – HÉRITAGE

Wijin

v1.0 2024

Plan

- Introduction à l'héritage
- La protection avec le mot clé final
- Mise en œuvre de l'héritage
- Classes abstraites et polymorphisme
- Visibilité des membres avec l'héritage

Java - Introduction à l'héritage

Java – Introduction à l'héritage

- **Héritage** : notion indissociable du langage Java et de la programmation orientée objet
  - Est-ce que ma classe est une sorte d'autre classe, plus **générique** ?
  - Exemple : est-ce qu'un Smartphone est une sorte de téléphone ?

Java – Introduction à l'héritage

- **Hypothèse** : un Smartphone est une sorte de Téléphone
- 2 classes : **Smartphone** et **Telephone**
- L'héritage s'exprime de cette manière :

```
class Smartphone extends Telephone {
…
}
```

Smartphone hérite de Telephone

Java – Introduction à l'héritage

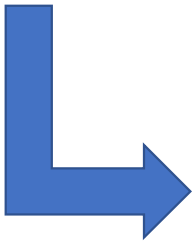
- Les types génériques en Java héritent **TOUS** d'une classe nommée **Object**

- Seuls les types de bases n'héritent pas de la classe **Object**

- **Conséquence** : la création d'une classe qui n'hérite de rien explicitement, hérite quand même de la classe **Object**

Java – Introduction à l'héritage

```java
package heritage;

public class Livre {

    private String titre;

    public Livre(String titre) {
        super();
        this.titre = titre;
    }

}
```



Equivalent

L'instruction « ***extends Object*** »
n'est pas nécessaire
car implicite !

```java
package heritage;

public class Livre extends Object {

    private String titre;

    public Livre(String titre) {
        super();
        this.titre = titre;
    }

}
```

Java – Introduction à l'héritage

Test dans Eclipse : saisir « ***this.*** » et regarder les propriétés et méthodes disponible pour l'objet en cours…

```java
package heritage;

public class Livre {

    private String titre;

    public Livre(String titre) {
        super();
        this.titre = titre;
    }

}
```

Java – Introduction à l'héritage

- Avec l'héritage, on peut **redéfinir** les méthodes héritées

```java
package heritage;

public class Livre extends Object {

    private String titre;

    public Livre(String titre) {
        super();
        this.titre = titre;
    }

    @Override
    public String toString() {
        return "Objet de type Livre avec le titre " + this.titre;
    }

}
```

Redéfinition de la méthode ***toString*** de la classe **Object**

Java – Introduction à l'héritage

- L'annotation « ***@Override*** » n'est pas obligatoire
- C'est une **bonne pratique** :
  - Pour indiquer aux **développeurs** qu'il s'agit d'une **méthode redéfinie**
  - Pour indiquer au **compilateur** qu'il s'agit d'une **méthode redéfinie**
    - Le compilateur va vérifier que la redéfinition est correcte !

Java – Introduction à l'héritage

- Ce qui précède est appelée **l'héritage naturel**
- Il est possible également d'utiliser une **méthode** de la classe héritée :
  - Mot-clé **super**
  - A faire suivre du **nom de la méthode** qui nous intéresse dans la classe héritée

Java – Introduction à l'héritage

```java
package heritage;

public class Livre extends Object {

    private String titre;

    public Livre(String titre) {
        super();
        this.titre = titre;
    }

    @Override
    public String toString() {
        return super.toString() + " Objet de type Livre avec le titre " + this.titre;
    }

}
```

Appel de la méthode ***toString*** de la classe héritée (ici : **Object**)

Concaténation avec autre chose

Java – Introduction à l'héritage

- Synthèse

  - **this** permet de référencer les propriétés et méthodes de l'objet en cours et de l'objet dont on hérite (à condition d'être accessible : **public** ou **protected**)

  - **super** permet d'accéder aux propriétés et méthodes de l'objet dont on hérite (à condition d'être accessible : **public** ou **protected**)

Java - La protection avec le mot clé final

Java - La protection avec le mot clé final

- L'héritage peut être **bloqué** en Java !
- Utilisation du mot clé **final** sur la classe
  - Cela provoque un **blocage complet** : aucune autre classe ne pourra hériter d'une classe définie avec **final**

Java - La protection avec le mot clé final

```java
package heritage;

final class Compte {
    private Double solde;
    private String proprietaire;

    public Compte(Double solde, String proprietaire) {
        super();
        this.solde = solde;
        this.proprietaire = proprietaire;
    }

    protected String getElements() {
        return "Propriétaire = " + this.proprietaire + ", solde = " + this.solde;
    }
}

class CompteBancaire extends Compte {
    private String nomBanque;

    public CompteBancaire(Double solde, String proprietaire, String nomBanque) {
        super(solde, proprietaire);
        this.nomBanque = nomBanque;
    }

    @Override
    public String getElements() {
        return super.getElements() + ", nom de la banque = " + this.nomBanque;
    }
}
```

Classe **Compte** définie avec **final**

Impossible d'hériter de la classe **Compte** (erreur de compilation) !

Java - La protection avec le mot clé final

- Il est possible de **limiter** le blocage de l'héritage
  - Limitation à une méthode en utilisant le mot clé **final** sur la méthode
  - Limitation à une propriété en utilisant le mot clé **final** sur la propriété (lecture seule !)

Java - La protection avec le mot clé final

```java
package heritage;

public class Compte {
    private final Double solde;
    private String proprietaire;

    public Compte(Double solde, String proprietaire) {
        super();
        this.solde = solde;
        this.proprietaire = proprietaire;
    }

    public final String getElements() {
        this.solde = 100.0;
        return "Propriétaire = " + this.proprietaire + ", solde = " + this.solde;
    }
}

class CompteBancaire extends Compte {
    private String nomBanque;

    public CompteBancaire(Double solde, String proprietaire, String nomBanque) {
        super(solde, proprietaire);
        this.nomBanque = nomBanque;
    }

    @Override
    public String getElements() {
        return super.getElements() + ", nom de la banque = " + this.nomBanque;
    }
}
```

Propriété *solde* déclarée avec **final**

Méthode *getElements()* déclarée avec **final**

Modification de la propriété *solde* impossible ! (erreur de compilation)

Redéfinition de la méthode *getElements()* impossible ! (erreur de compilation)

Java - Mise en œuvre de l'héritage

Java – Mise en œuvre de l'héritage

```java
public class Compte {
    private  Double solde;
    private String proprietaire;

    public Compte(Double solde, String proprietaire) {
        super();
        this.solde = solde;
        this.proprietaire = proprietaire;
    }

    public String getElements() {
        return "Propriétaire = " + this.proprietaire + ", solde = " + this.solde;
    }
}

class CompteBancaire extends Compte {
    private String nomBanque;

    public CompteBancaire(Double solde, String proprietaire, String nomBanque) {
        super(solde, proprietaire);
        this.nomBanque = nomBanque;
    }
}
```

La classe **Compte** dispose d'un constructeur à 2 paramètres

La classe **CompteBancaire** hérite de Compte

Appel du constructeur de la classe héritée (**Compte**)

Java – Mise en œuvre de l'héritage

- **Héritage** : si une méthode est déclarée dans la classe héritée (classe mère), et qu'elle n'est **pas redéfinie** dans la classe qui hérite (classe fille), alors cette méthode est **accessible** depuis un objet de la classe qui hérite (classe fille)

  - Si la **visibilité** le permet !

Java – Mise en œuvre de l'héritage

```java
public class Compte {
    private  Double solde;
    private String proprietaire;

    public Compte(Double solde, String proprietaire) {
        super();
        this.solde = solde;
        this.proprietaire = proprietaire;
    }

    public String getElements() {
        return "Propriétaire = " + this.proprietaire + ", solde = " + this.solde;
    }
}

class CompteBancaire extends Compte {
    private String nomBanque;

    public CompteBancaire(Double solde, String proprietaire, String nomBanque) {
        super(solde, proprietaire);
        this.nomBanque = nomBanque;
    }
}
```

Méthode accessible depuis les objets de la classe **CompteBancaire**

Java – Mise en œuvre de l'héritage

```java
package heritage;

public class TestComptes {

    public static void main(String[] args) {
        CompteBancaire cb = new CompteBancaire(100.0, "Dupont", "BNP");
        System.out.println(cb.getElements());
    }
}

```

Méthode accessible depuis l'objet de type **CompteBancaire**

Java – Mise en œuvre de l'héritage

```java
public class Compte {
    private  Double solde;
    private String proprietaire;

    public Compte(Double solde, String proprietaire) {
        super();
        this.solde = solde;
        this.proprietaire = proprietaire;
    }

    public String getElements() {
        return "Propriétaire = " + this.proprietaire + ", solde = " + this.solde;
    }
}

class CompteBancaire extends Compte {
    private String nomBanque;

    public CompteBancaire(Double solde, String proprietaire, String nomBanque) {
        super(solde, proprietaire);
        this.nomBanque = nomBanque;
    }

    @Override
    public String getElements() {
        return super.getElements() + ", nom de la banque = " + this.nomBanque;
    }
}
```

Redéfinition de *getElements()* avec utilisation de ***super*** pour appeler la méthode de la classe héritée

Java – Mise en œuvre de l'héritage

```java
package heritage;

public class TestComptes {

    public static void main(String[] args) {
        CompteBancaire cb = new CompteBancaire(100.0, "Dupont", "BNP");
        System.out.println(cb.getElements());
    }
}

```

Appel de *getElements()* de **Compte** dans la méthode *getElements()* de **CompteBancaire**

Java – Mise en œuvre de l'héritage

- **Uniquement de l'héritage simple** en Java !
  - Une classe ne peut hériter directement **que d'une seule classe** : pas d'héritage multiple !
  - Par contre, il est possible de constituer un **chainage** avec de l'héritage simple

Java - Classes abstraites et polymorphisme

Java – Classes abstraites et polymorphisme

- Dans une classe, il est possible de déclarer des méthodes **sans code associé**
  - Parce qu'il n'est **pas possible** de fournir un code pour ce **niveau d'abstraction**
  - Parce qu'on veut mettre en œuvre le **polymorphisme**

Java – Classes abstraites et polymorphisme

- Exemple
  - 1 Classe **Voiture** avec une méthode *rouler()*
  - 1 Classe **Camion** avec une méthode *rouler()*
  - 1 Classe **Vehicule** avec une méthode *rouler()* sans code !
    - On ne sais pas dire comment le véhicule roule, c'est trop abstrait à ce niveau !

Java – Classes abstraites et polymorphisme

- Exemple
  - La méthode *rouler()* est définie dans **Vehicule**, sans code, mais on peut l'appeler !

```java
package heritage;

public abstract class Vehicule {

    protected Boolean type;

    public Vehicule(Boolean type) {
        this.type = type;
    }

    public abstract void rouler();

}
```

La classe doit être abstraite également : utilisation du mot clé **abstract** sur la classe !

Utilisation du mot clé **abstract** sur la méthode *rouler()* sans code !

Java – Classes abstraites et polymorphisme

- Exemple

```java
package heritage;

public class Voiture extends Vehicule {

    public Voiture(Boolean type) {
        super(type);
    }

    @Override
    public void rouler() {
        System.out.println("La voiture roule.");
    }
}
```

La classe **Voiture** hérite de **Vehicule**

Implémentation de la méthode *rouler()* pour la classe **Voiture**

```java
package heritage;

public class Camion extends Vehicule {

    public Camion(Boolean type) {
        super(type);
    }

    public void rouler() {
        System.out.println("Le camion roule!");
    }

}
```

La classe **Camion** hérite de **Vehicule**

Implémentation de la méthode *rouler()* pour la classe **Camion**

Java – Classes abstraites et polymorphisme

- Exemple

```java
package heritage;

public class Garage {

    public static void main(String[] args) {

        Vehicule vehicule1 = new Voiture(true);
        vehicule1.rouler();

        Vehicule vehicule2 = new Camion(false);
        vehicule2.rouler();

    }

}
```

Création de 2 objets de type **Vehicule** : 1 compatible avec **Voiture** et 1 compatible avec **Camion**

Appel de la méthode *rouler()* possible car *rouler()* a été défini dans **Vehicule**

A l'exécution, c'est la méthode définie sur l'objet réel qui est appelée !
➔ **Polymorphisme**

Java – Classes abstraites et polymorphisme

- **Polymorphisme**
  - Fournir une **interface unique** à des entités pouvant avoir **différents types**

Java – Classes abstraites et polymorphisme

- Autre exemple de polymorphisme

```java
package heritage;

public abstract class Vehicule {

    protected Boolean type;

    public Vehicule(Boolean type) {
        this.type = type;
    }

    public void demarrer() {
        this.rouler();
    }

    public abstract void rouler();

}
```

Ajout d'une méthode concrète *demarrer()*

rouler() ne sera jamais exécutée sur **Vehicule**, mais sur une classe concrète (**Voiture** ou **Camion**)

Java – Classes abstraites et polymorphisme

- Autre exemple de polymorphisme

```java
public class Voiture extends Vehicule {

    public Voiture(Boolean type) {
        super(type);
    }

    @Override
    public void rouler() {
        System.out.println("La voiture roule.");
    }

}
```

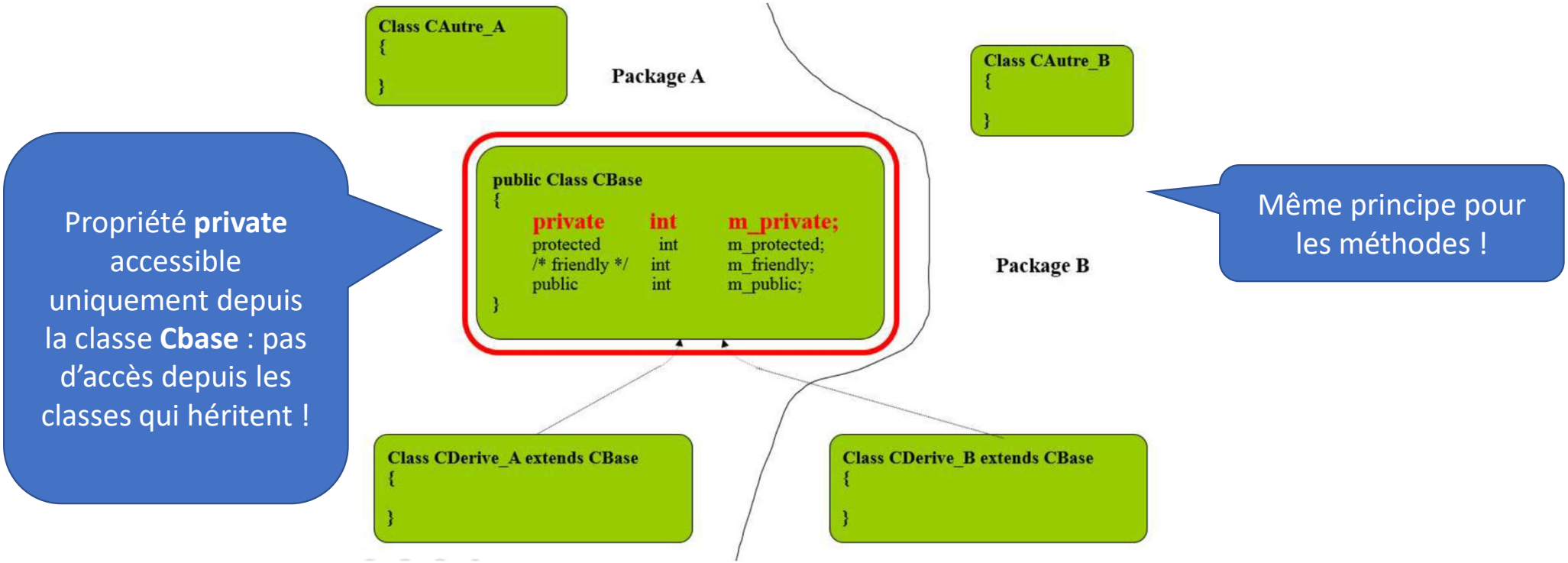
Définition de la méthode *rouler()* pour **Voiture**

```java
public class Garage {

    public static void main(String[] args) {

        Vehicule vehicule1 = new Voiture(true);
        vehicule1.demarrer();
    }

}
```

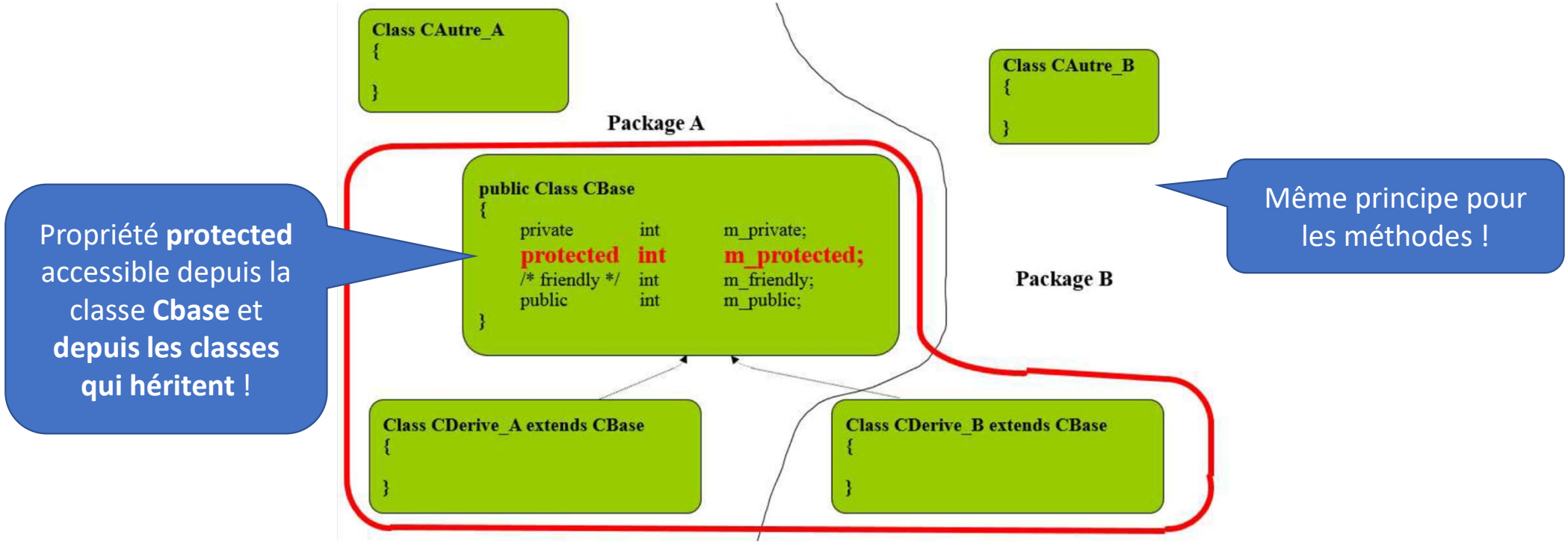
Création d'un objet de type **Vehicule** (avec le constructeur de **Voiture**) et appel de *demarrer()*

Java – Visibilité des membres avec l'héritage

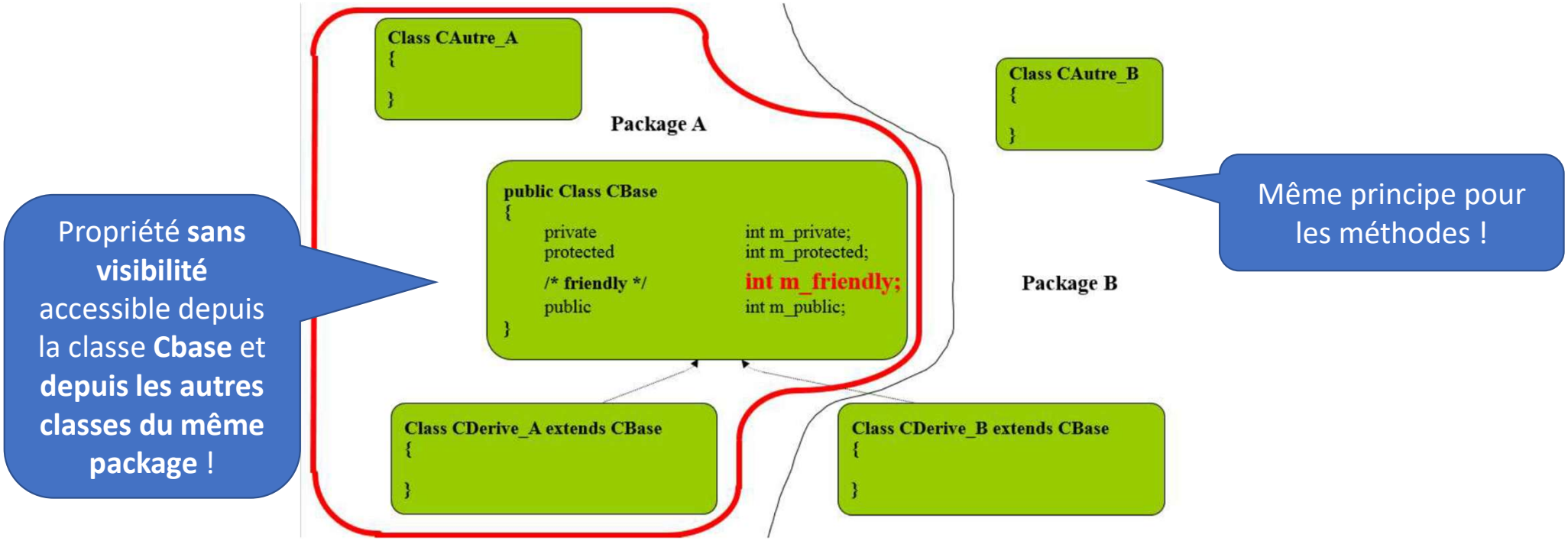
Java – Visibilité des membres avec l'héritage



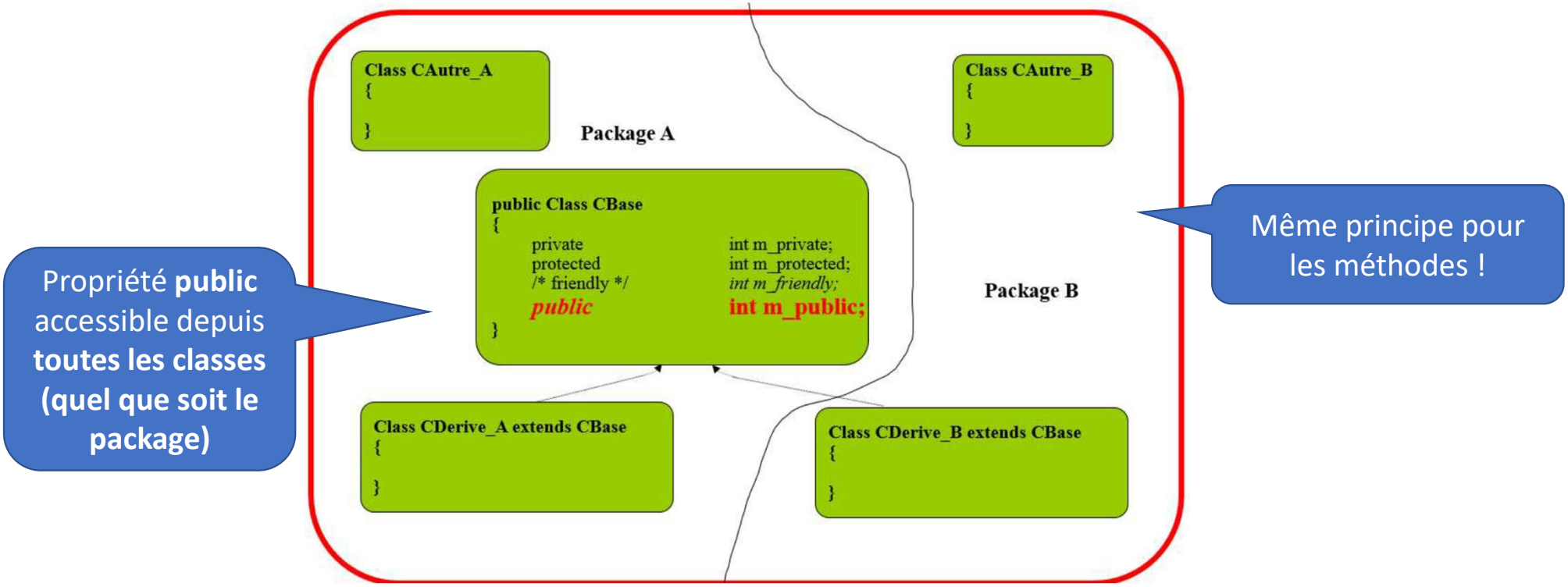
Java – Visibilité des membres avec l'héritage



Java – Visibilité des membres avec l'héritage



Java – Visibilité des membres avec l'héritage



Java - Quiz

Java – Quiz

- L’héritage multiple est possible en Java ?
  - **Vrai**
  - **Faux**

Java – Quiz

- Une propriété **private** est visible :
  - Uniquement depuis les classes « **static** »
  - Depuis la classe dans laquelle elle a été définie, ainsi que dans les classes qui en héritent
  - Uniquement depuis la classe dans laquelle elle a été définie

Java – Quiz

- Quelle est la syntaxe pour définir qu'une classe B hérite d'une classe A ?

  - class B extends A
  - class B : A
  - class B implements A

Java – Quiz

- Comment, depuis le constructeur d'une classe, peut-on appeler le constructeur de la classe dont on hérite, et qui attend deux String en paramètre ?
  - parent(''chaine1'', ''chaine2'')
  - ClassSuper(''chaine1'', ''chaine2'')
  - super(''chaine1'', ''chaine2'')
  - this(''chaine1'', ''chaine2'')

Java – Quiz

- Comment appelle-t-on en Java le fait que 2 méthodes ayant le même nom soient déclarées à des niveaux d'héritage différents ?
  - L'overcraft
  - L'overload
  - La surcharge
  - La redéfinition

Java – Quiz

- Qu’est-ce que le **polymorphisme** en Java ?
  - Le fait qu’une méthode appelée dépende de la classe dans laquelle elle a été déclarée
  - Le fait qu’un même appel de méthode peut avoir des comportements différents
  - Le fait qu’un même appel de méthode déclenche un traitement partagé

Java – Quiz

- Comment faire pour qu'une propriété d'une classe soit visible de cette classe ainsi que des classes qui en héritent uniquement ?

  - On utilise le mot clé **protected** sur la propriété

  - C'est le comportement par défaut, donc rien de particulier

  - On utilise le mot clé **default** sur la propriété

  - On utilise le mot clé **public** sur la propriété

Java – Quiz

- Il est possible de faire un **new** sur une classe **abstraite**
  - **Vrai**
  - **Faux**

Java – Quiz

- Une méthode déclarée **abstract** oblige de déclarer la classe en **abstract**

  - **Vrai**
  - **Faux**

MERCI POUR VOTRE ATTENTION

Faites-moi part de vos remarques concernant le cours afin qu'il soit amélioré pour les prochaines sessions : nicolas.sanou@wijin.tech