# Cours de Java

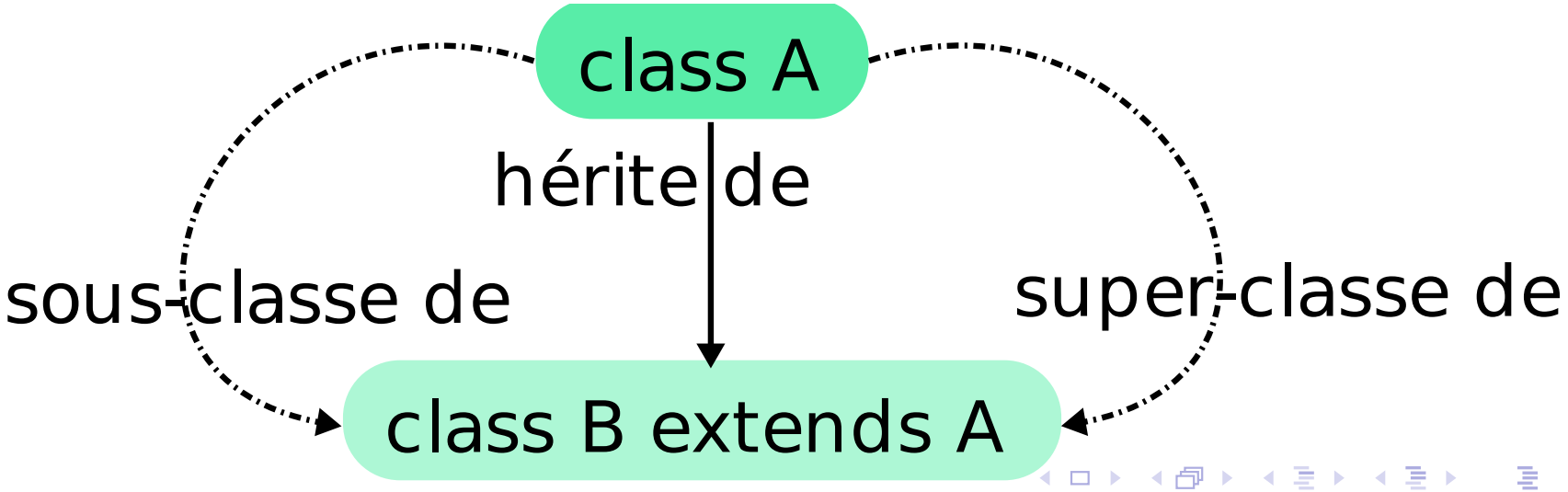
Paul Bedaride

Formation Continue

23 mai 2008

# L'héritage (Définition)

- Méthodologie de dévellopement
- Factorisation du code
- Si une classe B hérite d'une classe A, alors :
  - A est la super-classe de B
  - B est la sous-classe de A
  - B récupére les caractéristiques de A
    (i.e., B hérite de l'ensemble des méthodes et variables non privés de A)
- Une classe ne peut hériter que d'une seule classe
- On peut répéter ce mécanisme autant de fois qu'on veut



# L'héritage (Exemple)

```
public class Bloc {
  static int nbPieces = 1;
  protected int numero;
  protected Cube sur;
  public void vaEn(Cube dest) {
    ... // code 1
  }
  public void quiJeSuis() { ... // code 2 }
  public int numero() {return numero;}
  public Cube quiMePorte() {return sur;}
  public Cube quiMePorte(Cube porte) {return sur = porte;}
}
```

```
public class Cube extends Bloc {
  Bloc porte;
  Cube(){
    porte = null;
    numero = Bloc.nbPieces++;
  }
  public void quiJeSuis() {
    ... // code 3
  }
  public Bloc quiJePorte() {return porte;}
  public Bloc quiJePorte(Bloc porte)
    {return this.porte = porte;}
  public void vaEn(Cube dest) {
    ... // code 4
    super.vaEn(dest);
  }
}
```

```
public class Pyramide extends
    ↪ Bloc {
  Pyramide() {
    numero = Bloc.nbPieces++;
  }
  public void quiJeSuis() {
    ... // code 5
  }
}
```

# Les classes abstraites

- Classe abstraite : Définition de classes incomplètes
- Une sous-classe d'une classe abstraite ne redéfinissant pas toutes les méthodes abstraite est abstraite

```
abstract class Bloc {
  static int nbPieces = 1; // variable de classe
  ...
  abstract public void quiJeSuis();
  ...
}
```

- Différent de classe non instanciable

```
public class Bloc {
  protected static int nbPieces = 1;
  ...
  private Bloc() { // constructeur prive empeche l'instanciation
    ...
  }
  ...
}
```

# Les interfaces

- Une interface définie un prototype de classe
- Une classe peut implémenter plusieurs interfaces
  Cela permet de faire une sorte d'héritage multiple
- Une interface définie des constantes et des méthodes publiques non implémentés

```
interface Deplacable {
  public void vaEn(Porteur dest);
}
interface Porteur {
  public Deplacable quiJePorte();
  public Deplacable quiJePorte( →
      ↪ Deplacable porte);
}
abstract class Piece {
  protected static int nbPieces = 1;
  protected int numero;
  public int numero()
    {return numero;}
  abstract public void quiJeSuis();
}
```

```
abstract class Bloc extends Piece implements →
    ↪ Deplacable {
  protected Porteur sur;
  public void vaEn(Porteur dest) { ... }
  public Porteur quiMePorte() { return sur }
  public Porteur quiMePorte()
    { return sur = porte; }
}
class Pyramide extends Bloc{
  Pyramide() {...};
  public void quiJeSuis() {...};
}
class Cube extends Bloc implements Porteur {
  Cube() {...};
  public Bloc quiJePorte() {...};
  public Bloc quiJePorte() {...};
  public void quiJeSuis() {...};
}
```

# Le masquage

- Masquage de variable :

Dans la class `B` entier `a` est caché par caractère `a` mais accessible pas `super.a`

L'entier `b` est accessible par `A.b`

```
class A {
  int a;
  static intb;
  ...
}
```

```
class B extends A {
  char a;
  static chat b;
  ...
}
```

- Redéfinition de méthode :
  - Si une sous-classe définie une méthode ayant la même signature qu'une méthode de la super-classe.
  - Méthode de la super-classe accessible via `super`
- La surcharge de méthode :
  - plusieurs méthodes ayant le même nom mais des signature différentes
  - Exemple :

```
Deplacable quiJePorte();
Deplacable quiJePorte(Deplacable porte);
```

# Le polymorphisme

- Type static : type d'un objet dans le code
- Type dynamique : type d'un objet à l'execution
- Exemple :
  `Bloc sur = new Cube();`
  Le type static est `Bloc` et le type dynamique `Cube`
- On ne peut appeller que les méthodes du type static sur un objet
- C'est la méthode du type dynamique qui est appellé ou celle d'une de c'est super classe si la méthode n'a pas été redéfinie
- Exemples d'appels :

```
sur.quiMePorte() // appel la methode quiMePorte() de la class Bloc
sur.quiJeSuis() // appel la methode quiJeSuis() de la class Cube
sur.quiJePorte() // Erreur car la methode quiJePorte() n'existe pas →
    ↪ dans la class Bloc
```