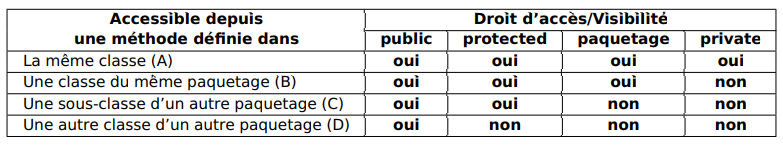
**public** : accessible depuis toutes les classes ;

类的访问

**private** : accessible seulement de la classe et d’aucune autre ;

**protected** : accessible du paquetage et des sous-classes (cf héritage).

absence de modifieur de droit d’accès : droit d’accès de paquetage, accessible depuis toutes les classes du même paquetage ;

**Règle** : Un attribut devrait toujours être déclaré **private** pour respecter :

(1)le **principe de la protection en écriture** ; (2) le **principe d’accès uniforme**.

L’attribut doit être **privé (private)** une méthode appelée setXxxx. C’est un **modifieur. /** une méthode appelée getXxx() par convention. C’est un **accesseur**.

>

Modularité有利于la réutilisation(Tout devrait être privé par défaut) & l’évolution(être changées sans impact sur les utilisateurs du module).

**概念**: Encapsulation : regrouper des informations logiquement liées. Masquage d’information : certaines informations sont cachées **(private)**.

>

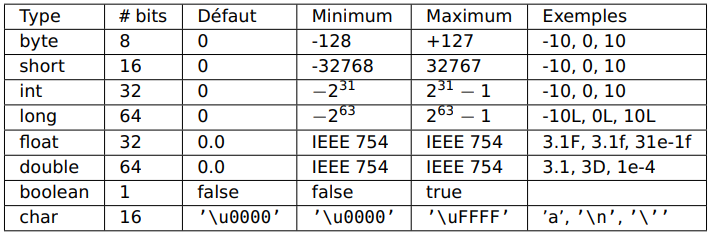
类属性方法

attributs et méthodes(d’instance) : toujours appliqués à un objet (éventuellement this).

attributs et méthodes de classe : appliqués à une classe, non à un objet. **static** indique qu’un attribut ou une méthode est **de classe.**

**类属性：**un attribut non spécifique à un objet donné mais commun à et partagé par tous les objets de la classe.**等效于范围为类的全局变量**

● est accessible de n’importe où (variable globale) ; ● **MAIS** est souvent un frein pour l’évolution de l’application.

**类方法：**● est accessible de partout (à partir de sa classe), ● permet de conserver le caractère symétrique de certaines méthodes (somme, égalité...),

● pas de récepteur (pas de this)

>

Le mot-clé **final** signifie que la variable ne pourra être réinitialisée ⇒ c’est donc une constante !

>

UML

Ordre des rubriques :1) nom de la classe ; 2)attributs ; 3)opérations ; 4)constructeurs

Droits d’accès : **public : +** ; **privé : −**; **protected : #** ; **paquetage : (rien)** ; **membre de classe : 下划线**

>

**Fabrique statique**优缺点

Avantages : ● les méthodes de classe ont un nom. ● une méthode de classe n’est pas obligée de créer un nouvel objet à chaque appel.

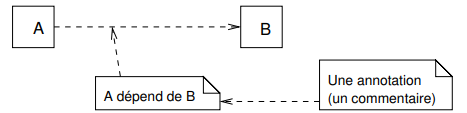
● la méthode peut retourner un objet d’un sous-type (à venir)

Inconvénients : ● pas de différence syntaxique entre fabrique statique et méthodes de classe ; ● déviation de la norme (les constructeurs !)

>

►依赖关系**(Dependency**) :Il y a relation de dépendance entre une classe A et une classe B si la classe A fait référence à la classe B dans son texte.

关 系 的 图 示

**语法**：类B作为类A的方法的参数（或者局部变量）存在

**符号**：由类A指向类B的**带箭头虚线**表示

►关联关系(**Association**) :

比依赖关系强，必然的，长期的，强烈的；分为单向关联（只是班级中增加了学生）、双向关联（在学生中也添加班级属性）；分为一对一（学生和学生证）、一对多（班级和学生）、多对多关联（学生和课程）；有两个类的关联（客户和订单、订单和商品）、还有一个类和自身关联（领导也是员工）

**Définition :** Une relation d’association est une relation entre deux classes qui traduit un couplage faible : chaque classe pourrait être considérée indépendamment de l’autre. Les objets des deux classes sont relativement indépendants et leurs durées de viene sont pas liées.

**语法**：类B作为成员变量形成存在于类A中

**符号**：●由类A指向类B的**带箭头实线**表示；双向关联可以取消两个箭头

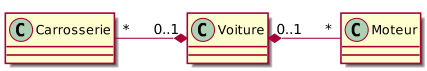
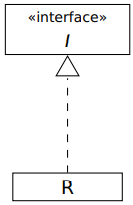
●在关联的两端可以标注关联双方的角色和多重性标记(数量)[(min/0)…(max/\*)]

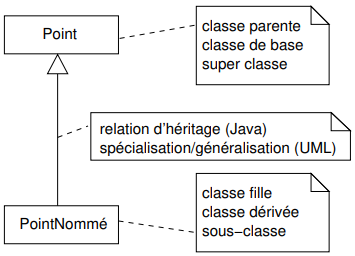
►聚合关系**(Aggregation)**：关联关系的一种特例；整体和部分的关系；整体部分可分离，整体的生命周期和部分的生命周期不同，公司与员工的关系

**Définition :** ses objets (le tout) agrégent d’autres objets (les parties).语法：同关联关系

符号：**空心菱形加实线箭头**

►组合关系**(Composition)**：关联关系的一种特例；整体和部分关系、整体部分不可分离、比聚合更强；整体的生命周期和部分的生命周期相同，同生共死。

**语法**：同关联关系

**符号：实心菱形加实线箭头**

►实现关系**(Realization)**：**类和接口之间的关系**；

一个类可以实现多个接口，实现所有接口的功能

语法：**implements**

符号：实现用一条**带空心三角箭头的虚线**表示，从类指向实现的接口

►继承关系（泛化关系 **Generalization**）

类和子类的关系，接口和子接口的关系；

一个类（称为子类）继承另外的一个类（称为父类）的功能，并可以增加它自己的新功能

**En Java, hériter c’est :**

●**définir un sous-type** : PointNommé est un sous-type de Point

●récupérer dans PointNommé (sous-classe) les éléments de Point (super-classe)

语法：**extends**

符号：一条**带空心三角箭头的实线**，从子类指向父类，或者子接口指向父接口。

►几种关系的比较：

**association** : couplage faible correspondant à une relation symétrique entre objets relativement indépendants (durées de vie non liées) ;

**agrégation** : association non symétrique avec couplage plus fort. Relation tout et parties ; Relation de subordination : propagation des opérations de A vers B ; On ne peut pas parler de A sans parler de B

**composition** : agrégation forte (par valeur). La durée de vie des objets « partie » est liée à celle du « tout ». Pas de partage possible.>

►**Interface** : Une interface correspond à une spécification. Elle définit un type (son nom) et la spécification des opérations qui peuvent lui être appliquées (méthodes sans code). On appelle méthode abstraite une méthode dont le code n’est pas donnée. Une interface correspond à une **vue utilisateur**/ **diagramme d’analyse**.

接口

►**语法约束**：Dans une interface, tout est **public(充要) ;** il ne contient pas de **code ;** il ne contient que des **méthodes d’instance,** pas de **méthode de classe (static)** ; ne peut contenir **ni constructeur, ni initialiseur** ; **tout attribut** est **de classe et constant(充要)**

►**实现**：On appelle réalisation d’une interface une classe qui s’engage (mot-clé **implements** en Java) à définir les opérations spécifiées dans cette interface.

►**Dans une réalisation, on peut** (bien sûr) : ajouter de nouvelles méthodes, définir des attributs, définir des constructeurs, utiliser les droits d’accès...

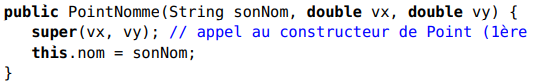
►如果类C实现I接口，则类型C是类型I的子类型 (**Sous-type**)

>

继承

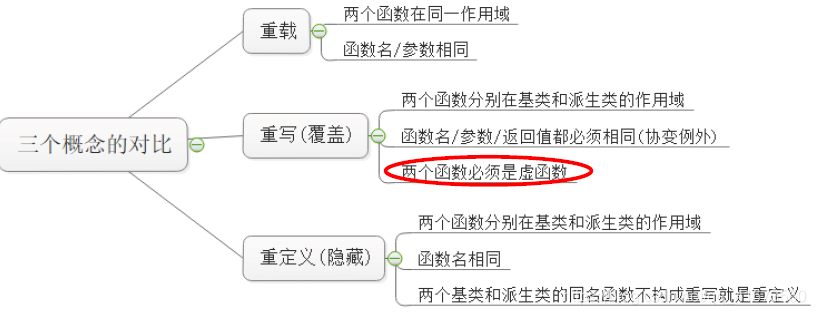
►En Java**, hériter** c’est **:**

●**définir un sous-type** : PointNommé est un sous-type de Point ●récupérer dans PointNommé (sous-classe) les éléments de Point (super-classe)

►Dans la sous-classe, on peut ajouter : ● de **nouveaux attributs** ● de **nouvelles méthodes**.( s’entend au sens de la ***surcharge***.具有重载的含义)►Héritage et constructeurs : Tout constructeur d’une sous-classe doit **obligatoirement** appeler un des constructeurs de la super-classe.►**语法**：**super** suivi des paramètres effectifs pour appeler un constructeur de la super-classe.

►**重载**(Overload/Surcharge) & **重定义**(Redefining) & **重写**(Override)

**Surcharge** : deux méthodes différentes qui ont le même nom (et donc des signatures différentes)

 **Redéfinition** : méthode de la super-classe définie dans la sous-classe (mêmes signatures !) @**Override** : exprime l’intention du programmeur de redéfinir une méthode de la super-classe et permet donc au compilateur de contrôler qu’il s’agit bien d’une redéfinition !Intérêt : Le compilateur vérifie qu’il s’agit bien d’une redéfinition

抽象

►**抽象类Une classe abstraite** est une classe qui ne permet pas de créer d’objet.►**语法**：On utilise le modifieur **abstract** devant class.

►**抽象方法Une méthode abstraite** (ou retardée) est une méthode dont le code n’est pas

►donné. On doit utiliser le mot-clé abstract (dans les modifieurs)

►**abstract** est incompatible(不兼容) avec final ou static.

►Q : Une classe abstraite ne permet pas de créer d’objets. À quoi sert le constructeur ?

►A: Il sert toujours à initialiser l’objet. Il sera appelé quand on créera

un objet à partir d’une sous-classe car le constructeur d’une

sous-classe doit obligatoirement appeler unconstructeur de la super-classe.

►接口和抽象类都是继承树的上层，他们的共同点如下：

1) 都是上层的抽象层。

2) 都不能被实例化

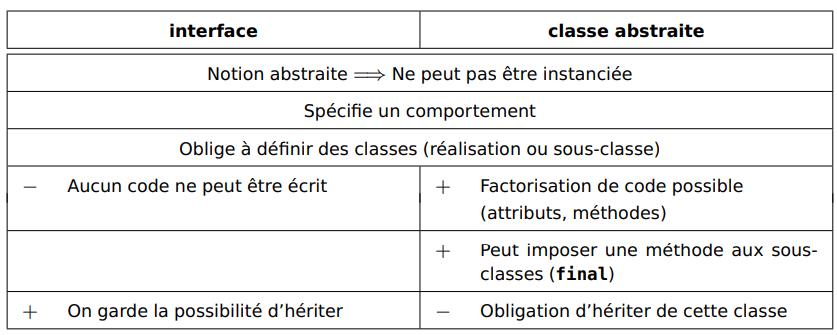
3) 都能包含抽象的方法，这些抽象的方法用于描述类具备的功能，但是不比提供具体的实现。

他们的区别如下：

1) 在抽象类中可以写非抽象的方法，从而避免在子类中重复书写他们，这样可以提高代码的复用性，这是抽象类的优势；接口中只能有抽象的方法。

抽象

2) 一个类只能继承一个直接父类，这个父类可以是具体的类也可是抽象类；但是一个类可以实现多个接口。

>

异常

● L’opérateur **throw** lève(抛出) objet exception : e.g.throw new Exception(<paramètres effectifs>);

● placer les instructions susceptibles de la lever dans un bloc **try**。将可能引发该语句的语句放在try块中.

● définir un gestionnaire d’exception (**catch**) pour le type d’exception considéré. 为所考虑的异常类型定义一个异常处理程序catch

►**le mécanisme d’exception** repose sur **3 phases (处理异常的3步)** :

1) une exception est levée(抛出异常) quand un problème ou anomalie est détecté ; 2) l’exception est propagée(传播异常) : l’exécution séquentielle du programme est interrompue et le flot de contrôle est transféré aux gestionnaires d’exception ;

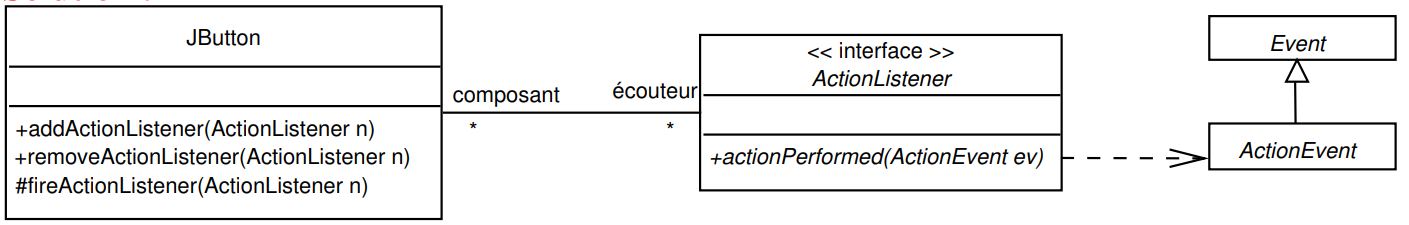
3) L’exception est récupérée par un gestionnaire d’exception(获取异常). L’exécution « normale » reprend avec les instructions qui suivent le gestionnaire d’exception. Une exception non récupérée provoque l’arrêt du programme (programme non robuste).►On peut spécifier les exceptions propagées par une méthode avec **throws** et plusieurs exception**s** : **throws E1, E2, E3**

►**throw** is a statement that throws an exception. **throws** is a statement that a method may throw an exception.

>

JAVA SWINGS

►**L’interface graphique** est composée de : **JLabel** ; **JButton** ; **le modèle**

►Un diagramme de classe UML :

►**顶层容器**属于窗口类组件，可以独立显示，一个图形界面至少需要一个窗口 :

**JFrame** : 一个普通的窗口（绝大多数 Swing 图形界面程序使用 JFrame 作为顶层容器）

►**中间容器**充当基本组件的载体，不可独立显示。最顶层的一个中间容器必须依托在顶层容器（窗口）内

**JPanel :** 一般轻量级面板容器组件 ; **JScrollPane :** 带滚动条 ; **JSplitPane** : 分隔面板

►**基本组件**是直接实现人机交互的组件 :

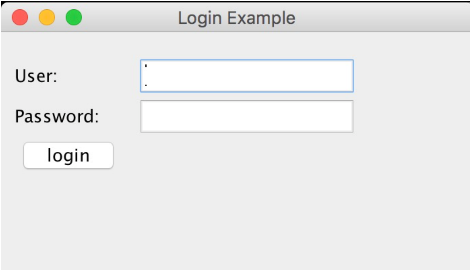
JLabel :标签 JButton :按钮 JCheckBox :复选框 JTextField :文本框 JTextArea :文本区域(多行)

►把 Swing 的各种组件(JComponent)添加到面板容器中(JPanel)，需要给面板容器指定**布局管理器(LayoutManager)**，明确容器(Container)内的各个组件之间的排列布局方式。**public void setBounds(int x, int y, int width, int height)**

**BorderLayout** :边界布局，把Container按方位分为 5 个区域（东、西、南、北、中），每个区域放置一个组件。

**FlowLayout**:流式布局，按组件加入的顺序，按水平方向排列，排满一行换下一行继续排列

**GridLayout** :网格布局，把Container按指定行列数分隔出若干网格，每一个网格按顺序放置一个控件。

►例子：

**import** javax.swing.JButton;

**import** javax.swing.JFrame;

**import** javax.swing.JLabel;

**import** javax.swing.JPanel;

**import** javax.swing.JPasswordField;

**import** javax.swing.JTextField;

**public** **class** SwingLoginExample {

**public** **static** **void** main(String[] args) {

*// 创建 JFrame 实例*

JFrame frame = **new** JFrame("Login Example"); *// Setting the width and height of frame*

frame.setSize(350, 200);

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

*/\* 创建面板，这个类似于 HTML 的 div 标签*

*\* 我们可以创建多个面板并在 JFrame 中指定位置。面板中我们可以添加文本字段，按钮及其他组件。*

*\*/*

JPanel panel = **new** JPanel();

*// 添加面板*

frame.add(panel);

*/\**

*\* 调用用户定义的方法并添加组件到面板*

*\*/*

placeComponents(panel);

*// 设置界面可见*

frame.setVisible(**true**);

}

**private** **static** **void** placeComponents(JPanel panel) {

*/\* 布局部分我们这边不多做介绍*

*\* 这边设置布局为 null*

*\*/*

panel.setLayout(**null**);

*// 创建 JLabel*

JLabel userLabel = **new** JLabel("User:");

*/\* 这个方法定义了组件的位置。*

*\* setBounds(x, y, width, height)*

*\* x 和 y 指定左上角的新位置，由 width 和 height 指定新的大小。*

*\*/*

userLabel.setBounds(10,20,80,25);

panel.add(userLabel);

*//创建文本域用于用户输入*

JTextField userText = **new** JTextField(20);

userText.setBounds(100,20,165,25);

panel.add(userText);

*// 输入密码的文本域*

JLabel passwordLabel = **new** JLabel("Password:");

passwordLabel.setBounds(10,50,80,25);

panel.add(passwordLabel);

*/\**

*\*这个类似用于输入的文本域*

*\* 但是输入的信息会以点号代替，用于包含密码的安全性*

*\*/*

JPasswordField passwordText = **new** JPasswordField(20);

passwordText.setBounds(100,50,165,25);

panel.add(passwordText);

*// 创建登录按钮*

JButton loginButton = **new** JButton("login");

SimpleListener ourListener = new SimpleListener();// 建立一个actionlistener让两个按钮共享

loginButton.addActionListener(ourListener);

loginButton.setBounds(10, 80, 80, 25);

panel.add(loginButton);

}

private class SimpleListener implements ActionListener {

//利用该内部类来监听所有事件源产生的事件。便于处理事件代码模块化

public void actionPerformed(ActionEvent e) {

// 利用getActionCommand获得按钮名称

// 也可以利用getSource()来实现

// if (e.getSource() == loginButton)

String buttonName = e.getActionCommand();

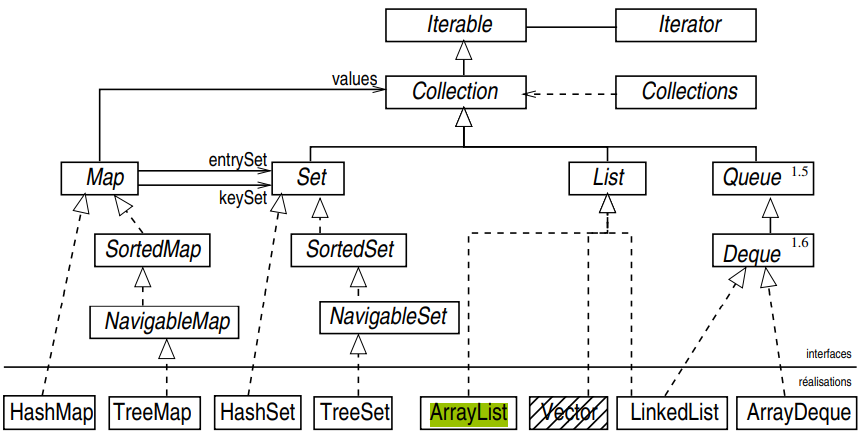
if (buttonName.equals("login "))

　　 JOptionPane.showMessageDialog(frame," loginButton被点击");

}

>

数据结构

►**Principales collections :**

**Collection** : le type le plus général des collections.

**List** : les éléments ont une position (numéro d’ordre, rang, index...)

opérations relatives à la position

**Set** : ensemble au sens mathématique :

pas de double (ensemble mathématique), pas de position

SortedSet : éléments munis d’une relation d’ordre (éléments ordonnés)

NavigableSet : SortedSet avec des opérations pour trouver un élément proche

**Queue** : file (d’attente), avec politique **FIFO** ou autre.

**Deque** : (Double Ended Queue) :

接口

opérations sur les deux extrémités de la file (File, Pile)

**Map** : tableau associatif (accès aux éléments par une clé)

Attention, ce n’est pas un sous-type de collection !

实现

sous-types : SortedMap et NavigableMap

►List&ArrayList:

List is an interface, and ArrayList is an implementation class of the List interface.

The ArrayList class inherits and implements the List interface.

Therefore, the List interface cannot be constructed, that is, we cannot create an instance object, but we can create an object reference to the List interface as follows, and the instance object of the ArrayList implementation class serves as this pointer to the List. The object reference of the interface.