

Nfp121

Programmation Avancée

Des Cours , des Exercices dirigés, des devoirs hebdomadaires

Cours 1h30 + 0h30 Présentation devoirs

Eds 1h30 centrés sur la réponse aux devoirs et compléments

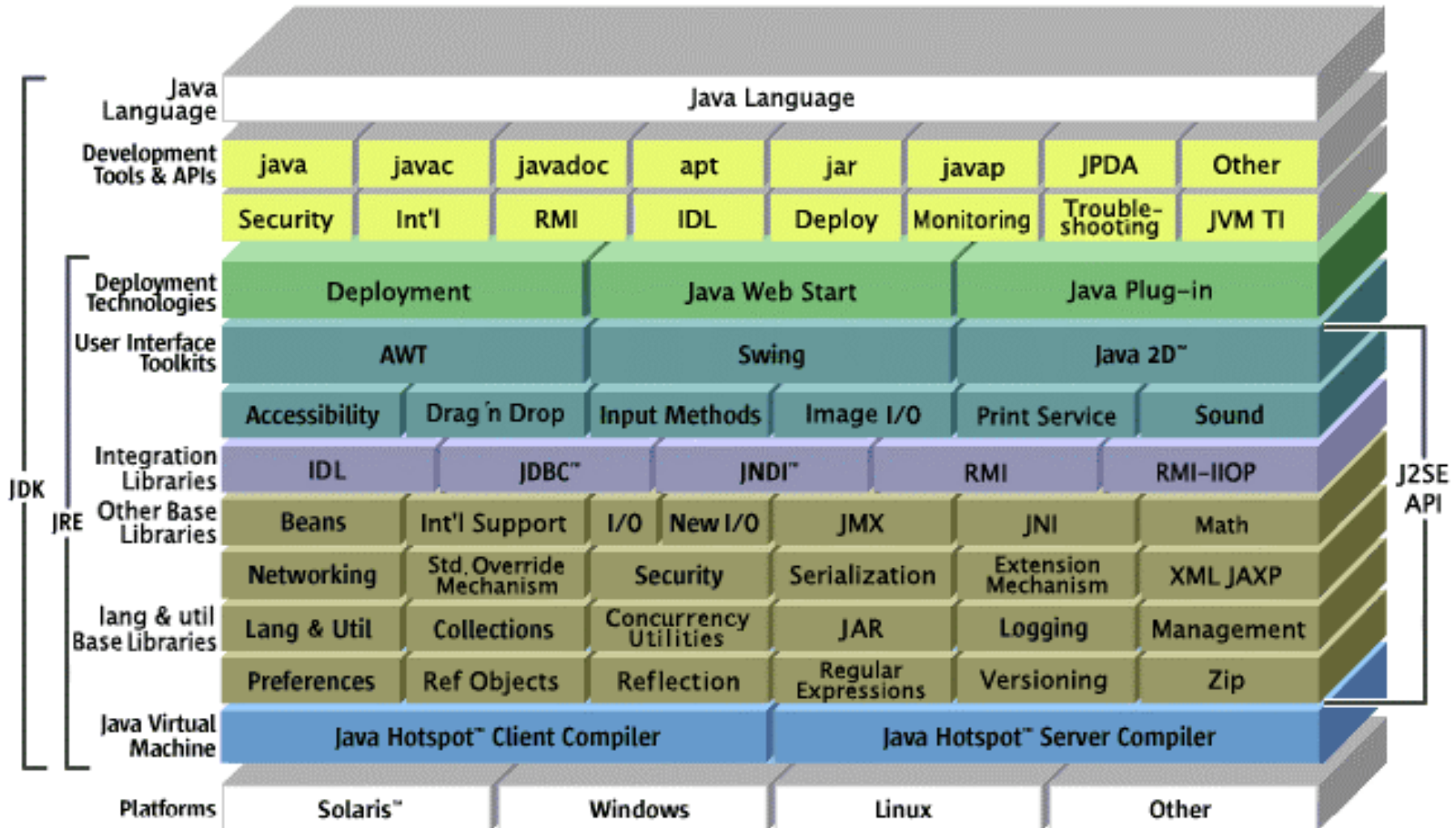
L'évolution de Java

conséquences sur le cours

- Java 1.0
 - 212 classes , 8 packages
- Java 1.1
 - 504 classes , 23 packages
- Java 1.2
 - 1520 classes, 59 packages
- Java 1.3
 - 1842 classes, 76 packages
- Java 1.4
 - 2991 classes, 135 packages
- Java 1.5
 - 3562 classes, 166 packages

Java 2

Java™ 2 Platform Standard Edition 5.0



Le programme du Cours

- 1. Structure des applications objets avec JAVA
- 2. Types primitifs, Aspects impératifs du langage
- 3. Exceptions, Interfaces, Types et Classes
- 4. Programmation événementielle
- 5. Généricité et Collections:
- 6. Structures de données et Patterns de conception
- 7. Introspection et réflexivité en Java:
- 8. Programmation des Entrées/Sorties:
- 9. Java et XML, persistance:
- 10. Programmation par Contrat:
- 11. Validation des programmes: ESC/Java
- 12. UML2 et le langage OCL
- 13. Méthodes de développement et langage UML
- 14) Programmation concurrente

Programmation
Orientée
Objets

et

langage Java

L'esprit de ce premier cours

- Introduire Java à travers la vision Objet
- En particulier dans son rapport à UML
- Lister les aspects Java correspondant aux concepts objets

Programmation procédurale

```
Construire(Maison m){  
    creuser(fouilles);  
    commander(béton)  
    couler(fouilles);  
    commander(parpaings);  
    while(!fini(sous-sol)) {poser(parpaings);}  
    commander(ourdis);  
    while(...  
    commander(béton);  
    couler(plancher);  
    ...  
}
```

Programmation non structurée

```
Construire(Maison m){
    creuser(fouilles);
    commander(béton)
    couler(fouilles);
    commander(parpaings);
    while(!fini(sous-sol)) {poser(parpaings);}
    commander(ourdis);
    while(....
    commander(béton);
    couler(plancher);
    ...
}
```

```
commander(Béton b){
    ...
}
couler(Fouilles f){
    ...
}
commander(Parpaings p){
    ...
}
couler(Plancher p) {
    ...
}
```

Structuration par les fonctions

```
package commandes;
```

```
public commander(béton){.....}  
... commander(Parpaings p){.....}  
... commander(Ourdis o ) {.....}  
... commander(Béton b) ){.....}  
...
```

```
package coulages;
```

```
public couler(Fouilles f){.....}  
... couler(Plancher p) ){.....}  
...
```

```
package construction;
```

```
import commandes.*;
```

```
import coulages.couler;
```

```
...
```

```
construire(Maison m){
```

```
    creuser(fouilles); commander(béton)
```

```
    couler(fouilles); commander(parpaings);
```

```
    while(!fini(sous-sol)){poser(parpaings);};
```

```
..}
```

Visibilité

Portée

Protection

Prog. Structurée Descendante

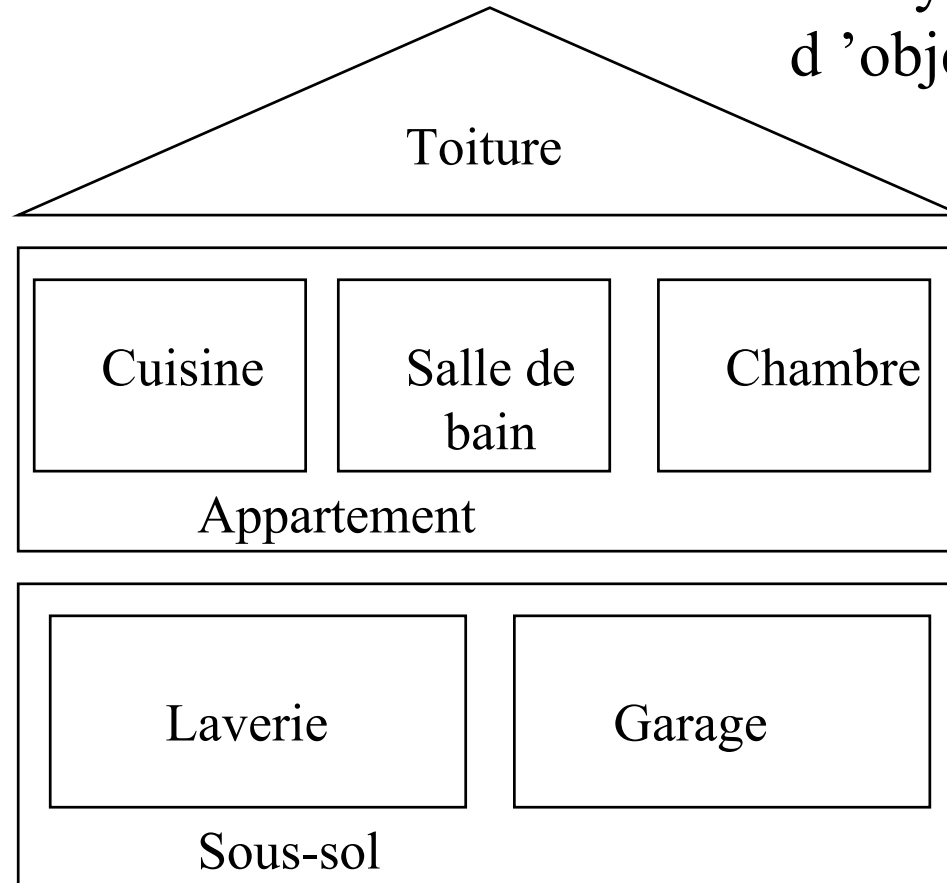
```
Construire(Maison m){  
    passerLesCommandes();  
    construireLeSousSol();  
    construireAppartement();  
    couvrir();  
    ....  
    creuser();  
}  
  
passerLesCommandes(){  
    commander(Béton b){  
        ...  
    }  
    couler(Fouilles f){  
        ...  
    }  
    commander(Parpaings p){  
        ...  
    }  
}  
  
BEGIN  
    commander(béton);  
    commander(parpaings);  
    ...  
END;
```

imbrication

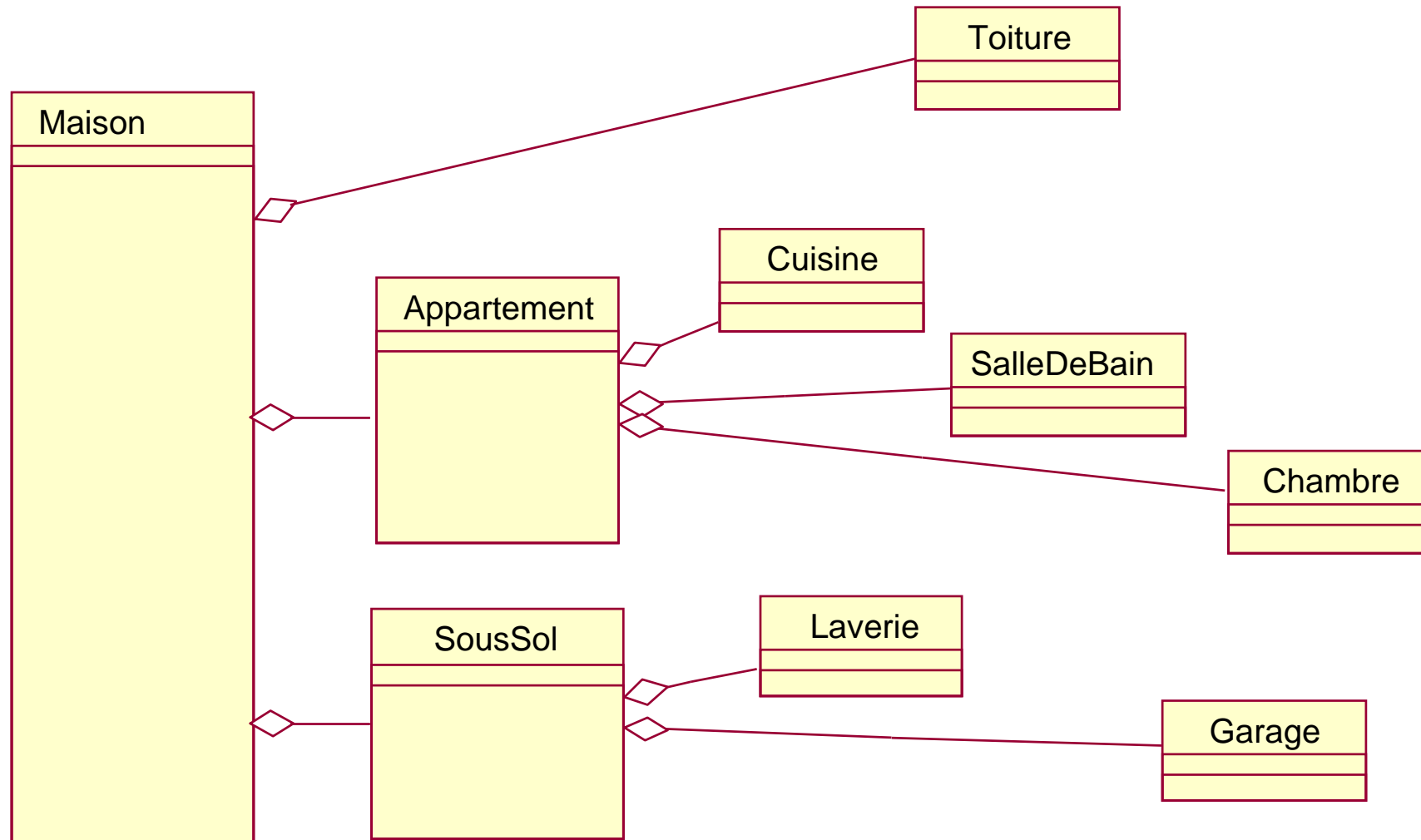
Structuration par les objets

Plan de la Maison

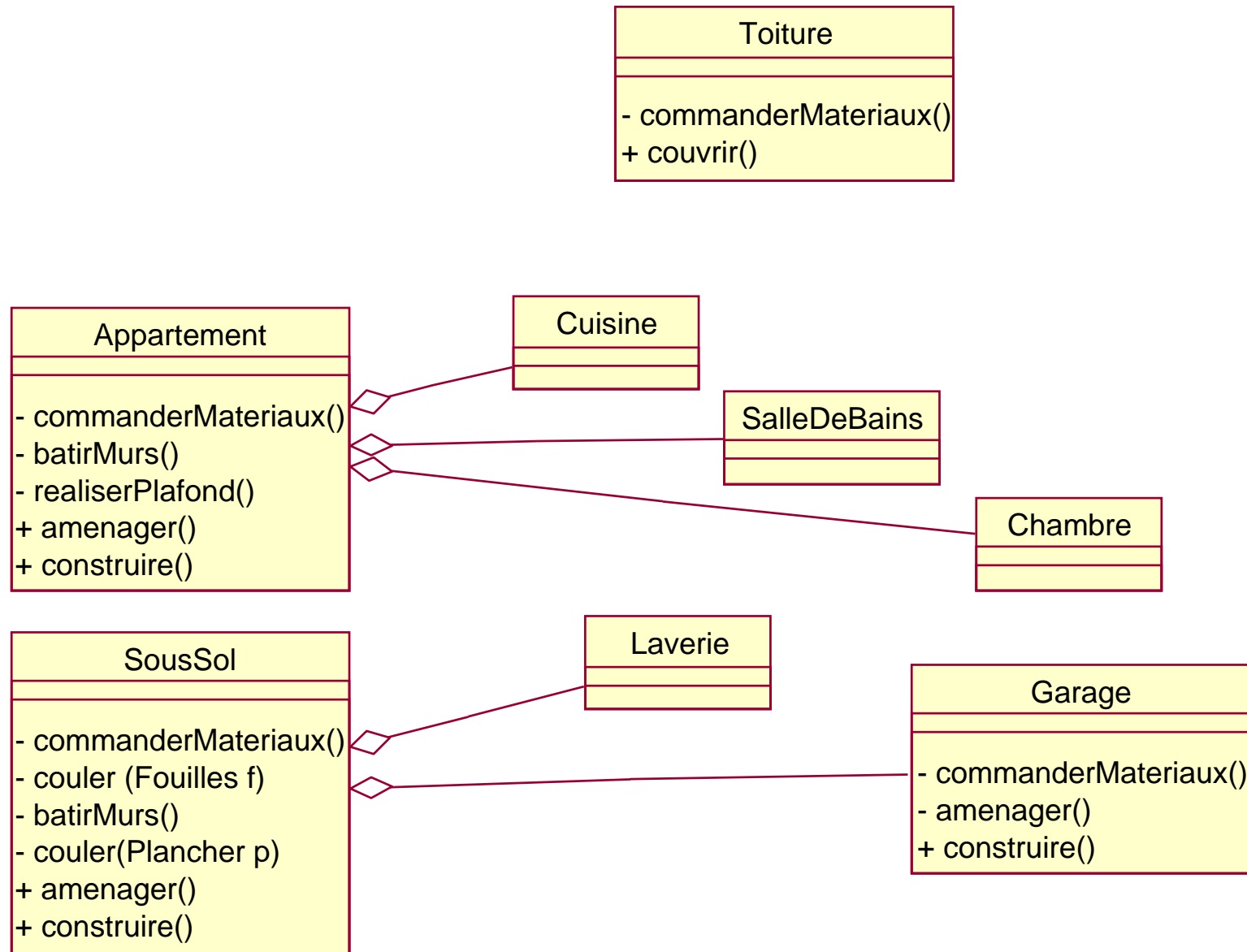
Analyser les classes
d'objets du réel



Plan du Logiciel (modélisation)



Structurer les fonctions par les classes



Des Objets du réel aux classes Java

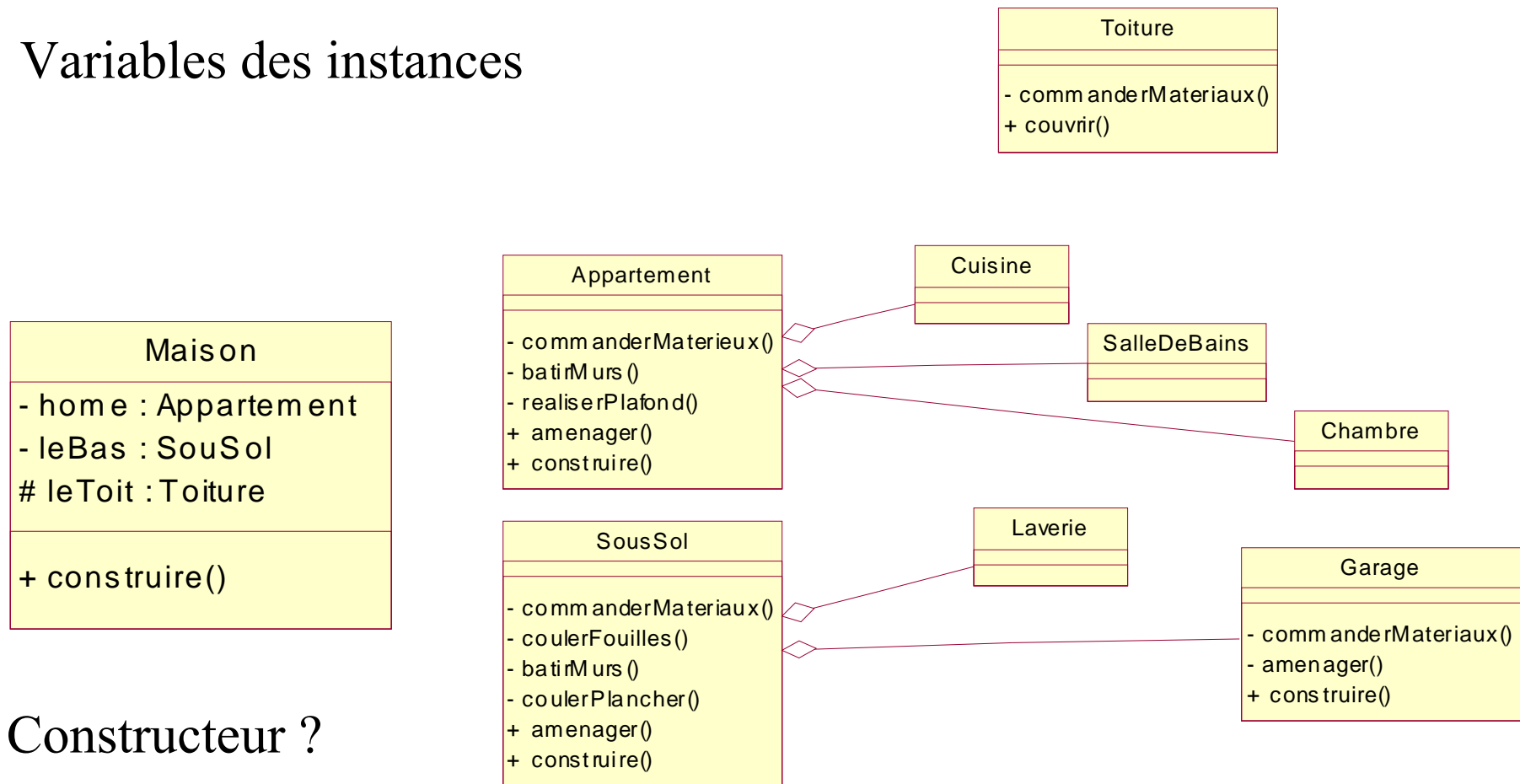
```
public class Appartement {  
    ...  
    private void commanderMateriaux() {  
        ...  
    }  
  
    private void batirMurs() {  
        ...  
    }  
  
    private void realiserPlafond() {  
        ...  
    }  
  
    public void amener() {  
        ...  
    }  
  
    public void construire() {  
        commanderMateriaux();  
        batirMurs();  
        realiserPlafond();  
    }  
}
```

Appartement
- commanderMateriaux() - batirMurs() - realiserPlafond() + amener() + construire()

Conventions sur les identificateurs
de classe
de variables

Attributs (propriétés des objets)

Variables des instances



Constructeur ?

Visibilité des attributs

Maison
- home : Appartement - leBas : SouSol # leToit : Toiture
+ Maison()

```
// Source file: Maison.java
```

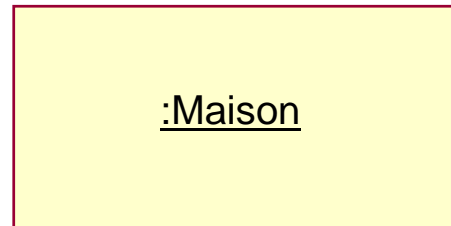
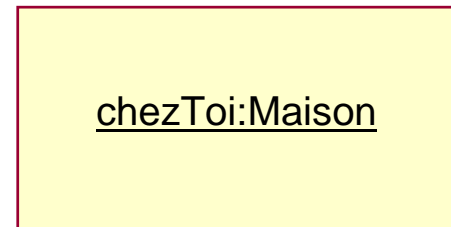
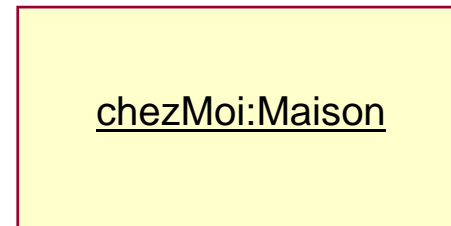
```
public class Maison {  
    private Appartement home;  
    private SouSol leBas;  
    protected Toiture leToit;  
  
    public Maison() {  
        ...  
    }  
}
```

Constructeur des instances

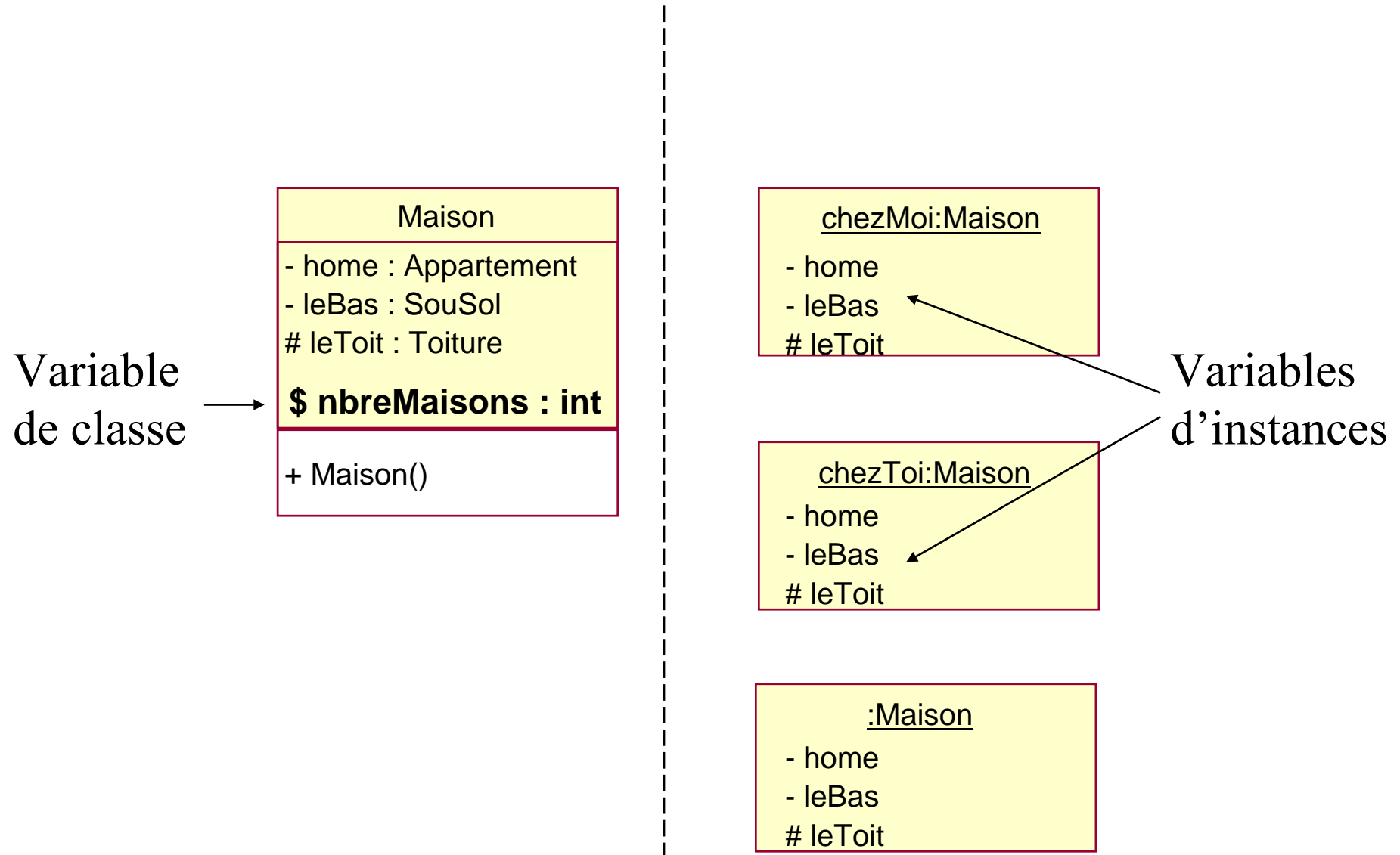
Classe

versus

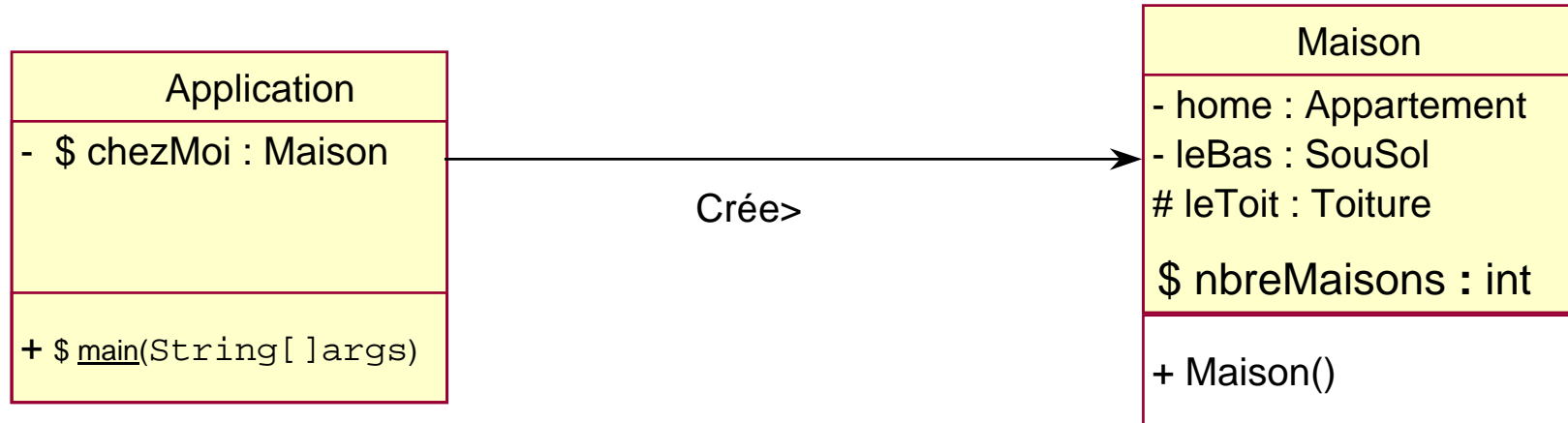
Objets (instances)



Attributs de Classe



Constructeur d 'objet

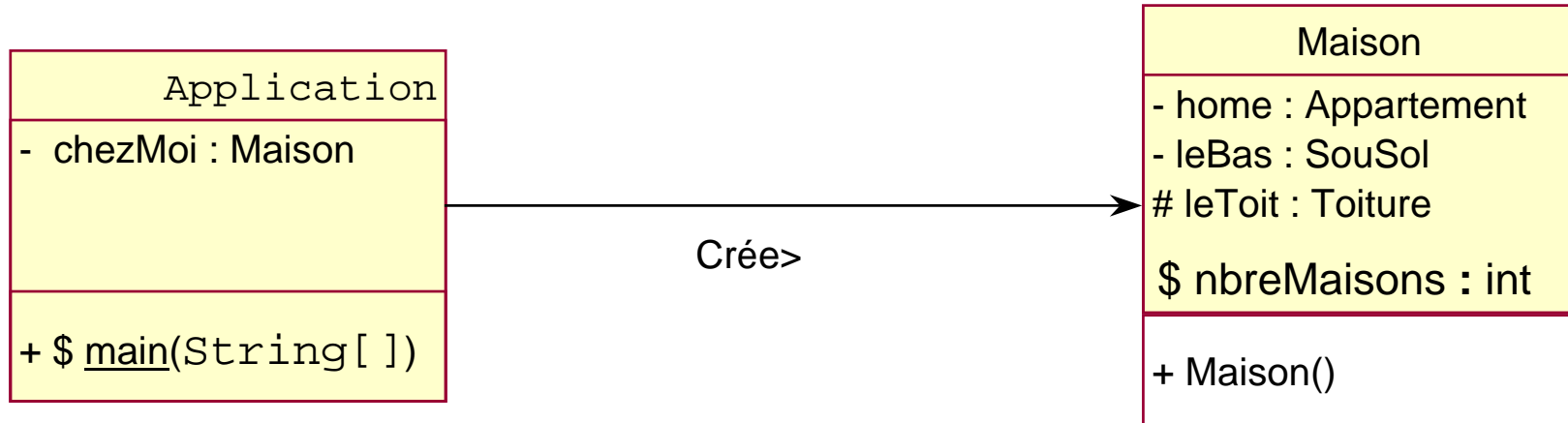


```
public class Application{
    private static Maison chezMoi;           { //variable de classe

    //programme principal
    public static void main(String[] args) { //méthode de classe

        chezMoi= new Maison();
    }
}
```

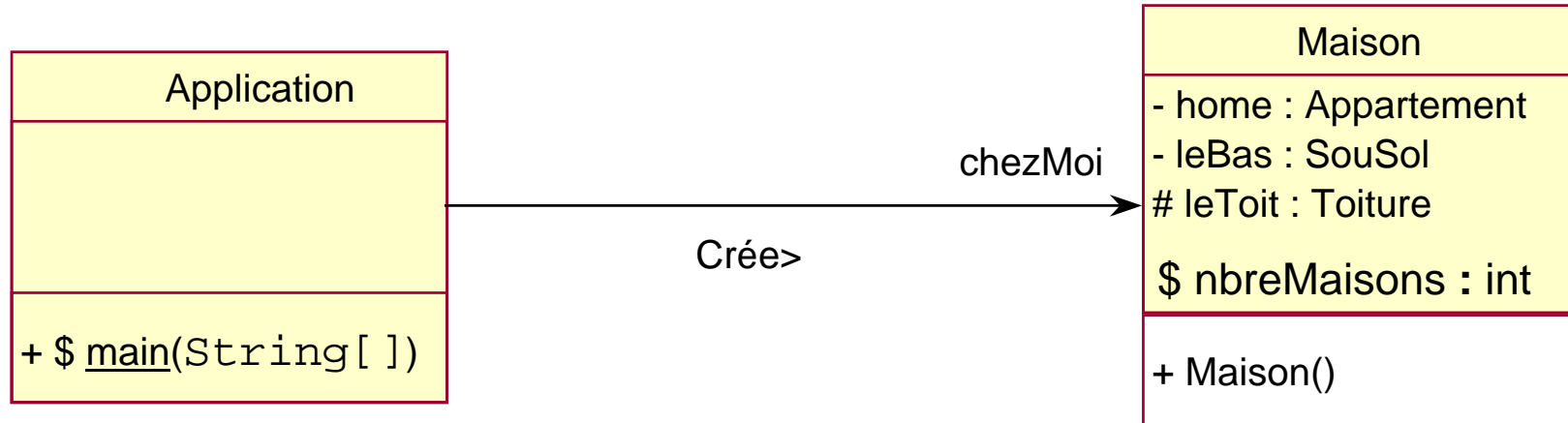
Interdit !



```
public class Application{
    private Maison chezMoi; //variable d'instance

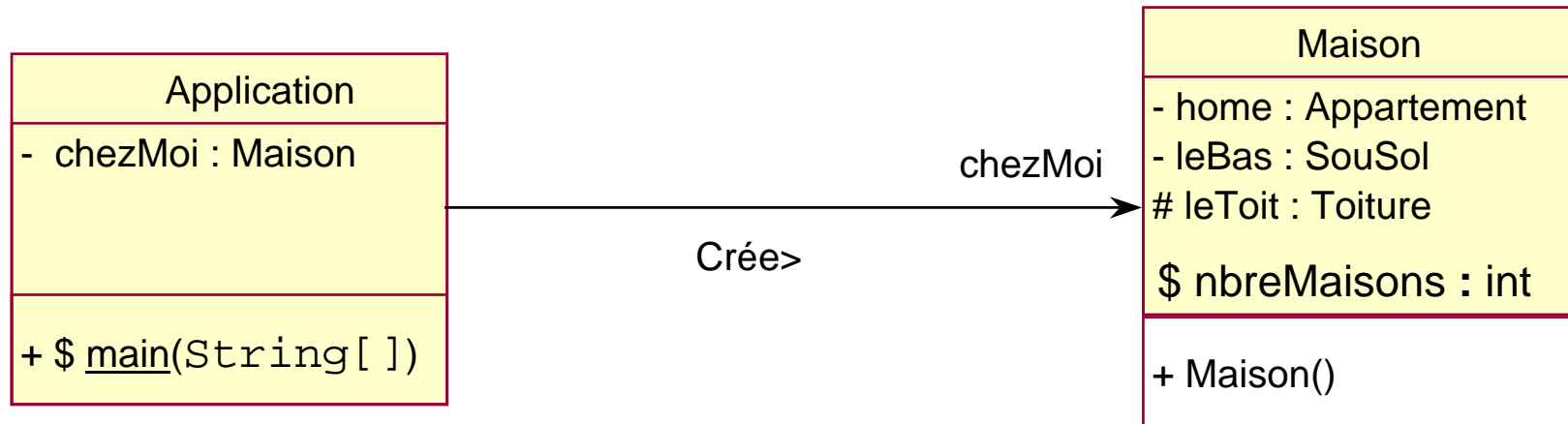
    public static void main(String[] args) { //méthode de classe
        chezMoi= new Maison(); interditt!!!!!!
    }
}
```

Variables locales



```
public class Application{  
  
    public static void main(String[] args) {  
        Maison chezMoi;                //variable locale  
  
        chezMoi= new Maison();  
        ...  
    }  
}
```

Méthode statique et d'instance



```
public class Application{
    private Maison chezMoi;           //variable d'instance

    public static void main(String[] args) { //méthode de classe
        new Application().ProgPrincipal(args);
    }

    public void ProgPrincipal(String[] args) { //méthode d'instance
        chezMoi= new Maison();
    }
}
```

Exemples de méthodes statiques

Integer

\$ parseInt (s : String) : int

Long

\$ parseLong (s : String) : long

Float

\$ valueOf (s : String) : Float
floatValue() : float

String

equals (anObject : Object) : boolean

charAt (index : int) : char

length () : int

indexOf (ch : int) : int

toCharArray () : char[]

Appel des méthodes d'instance

Maison
- home : Appartement - leBas : SouSol # leToit : Toiture
\$ nbreMaisons : int = 0
+ Maison()

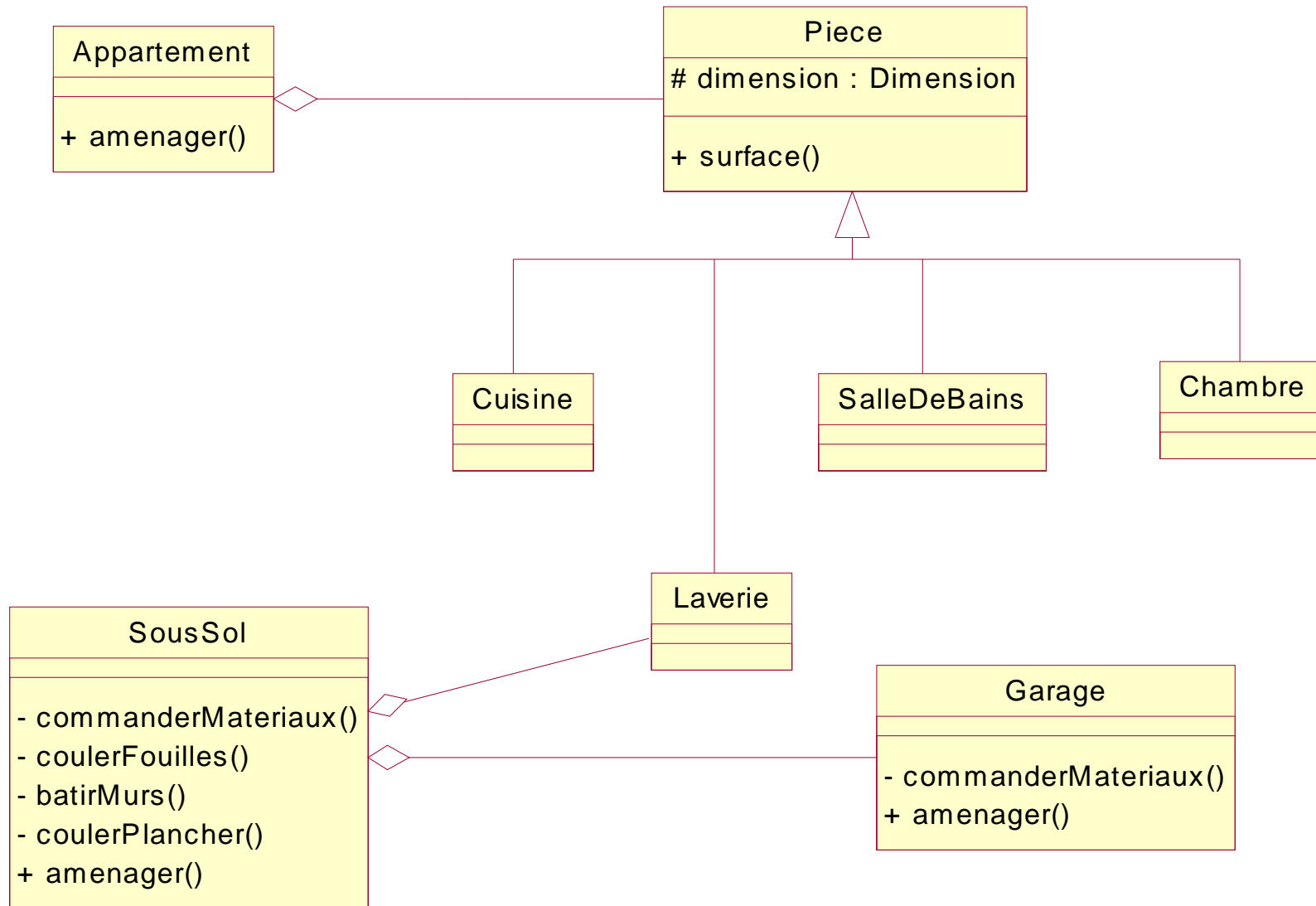
```
// Source file: Maison.java

public class Maison {
    private Appartement home;
    private SouSol leBas;
    protected Toiture leToit;
    static int nbreMaisons=0;

    public Maison() {
        Maison.nbreMaisons++;
        leBas= new SousSol();
        home= new Appartement();
        leToit= new Toiture();

        leBas.amenager();
        home.amenager();
        leToit.couvrir();
    }
}
```

Héritage



Java et l'héritage: la syntaxe

```
public class Piece {  
    protected Dimensions dimension = new Dimension();  
    public int surface() { return dimension.larg * dimension.long ; }  
}
```

```
public class Cuisine extends Piece {  
}
```

....

```
public class Chambre extends Piece {  
}
```



Typage et héritage

```
public class Appartement {  
    protected Piece[] piece;    // nous reverrons les tableaux plus tard  
    public Appartement() {  
        piece = {new Cuisine(), new salleDeBain(),  
                new Chambre(), new Chambre() };  
    }  
    s=piece[1].surface();  
    s=piece[2].surface();  
}
```

Initialisateur d'attribut de classe

```
public class Maison {
    private Appartement home;
    private SousSol leBas;
    protected Toiture leToit;
    static int nbreMaisons;

    static {
        nbreMaisons=0;        //par défaut
    }

    public Maison() {
        leBas= new SousSol();
        home= new Appartement();
        leToit= new Toiture();

        leBas.amenager();
        home.amenager();
        leToit.couvrir();
    }
}
```

Initialisateur d'attributs

```
public class Appartement {  
    protected Piece[] piece;  
    { // initialisateur d'instance  
        piece = {new Cuisine(), new salleDeBain(),  
                new Chambre(), new Chambre() };  
    }  
    s=piece[1].surface();  
    s=piece[2].surface();  
}
```

Constantes « blanches »

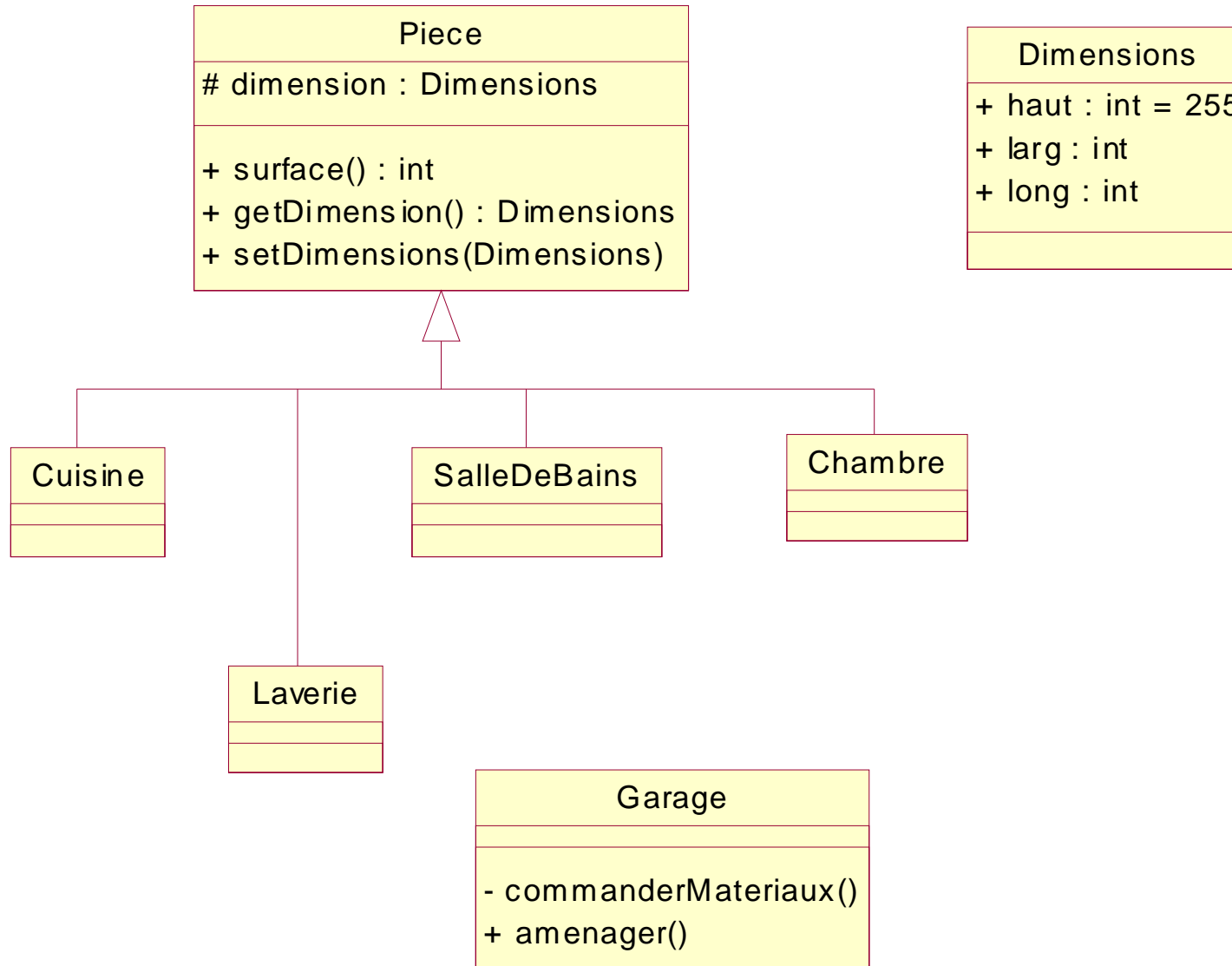
```
public class Maison {
    private Appartement home;
    private SousSol leBas;
    protected Toiture leToit;
    static int nbreMaisons;
    static final int maxMaisons;

    static {
        maxMaisons =100;
    }

    public Maison() {
        leBas= new SousSol();
        home= new Appartement();
        leToit= new Toiture();

        leBas.amenager();
        home.amenager();
        leToit.couvrir();
    }
}
```

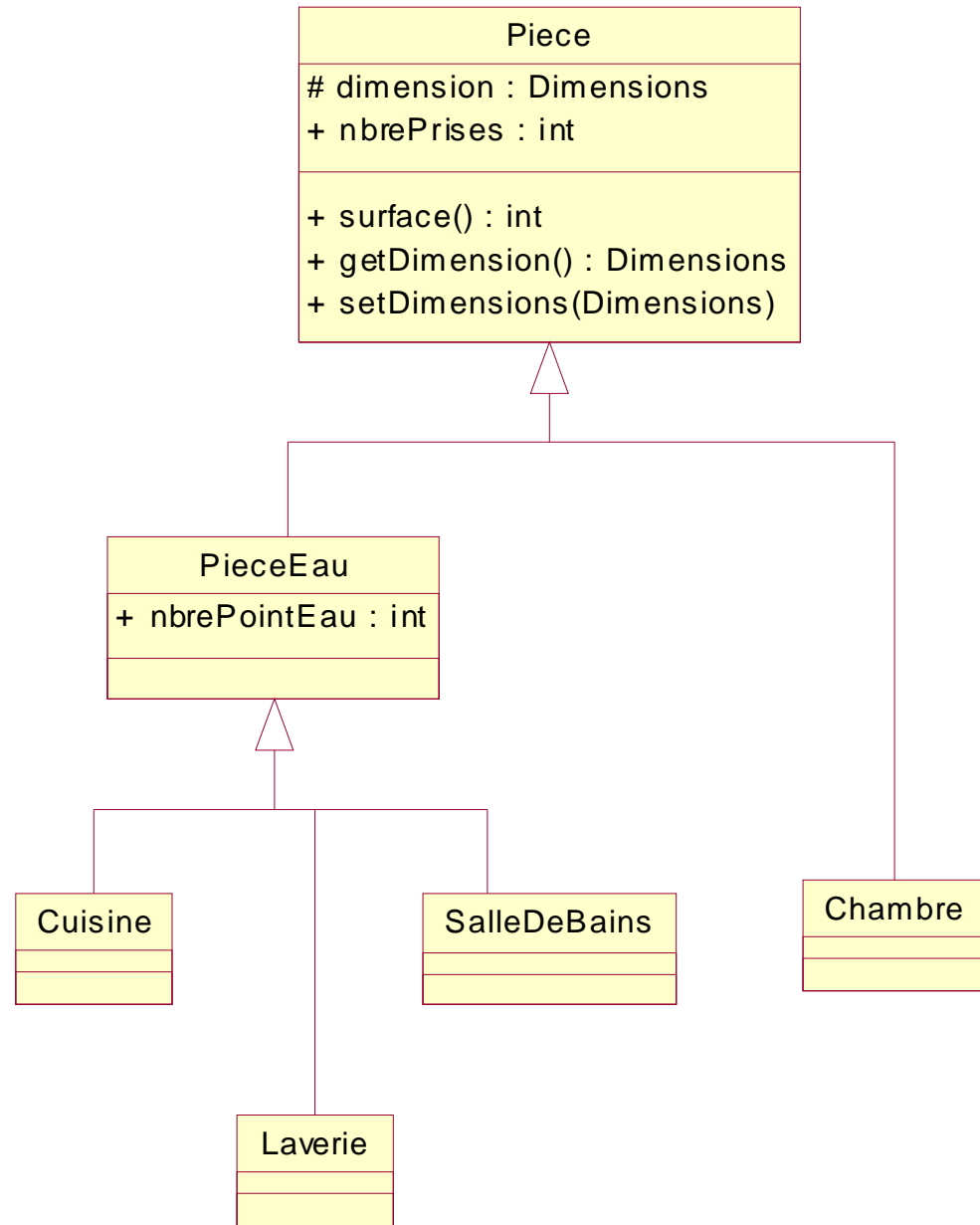
Accesseurs



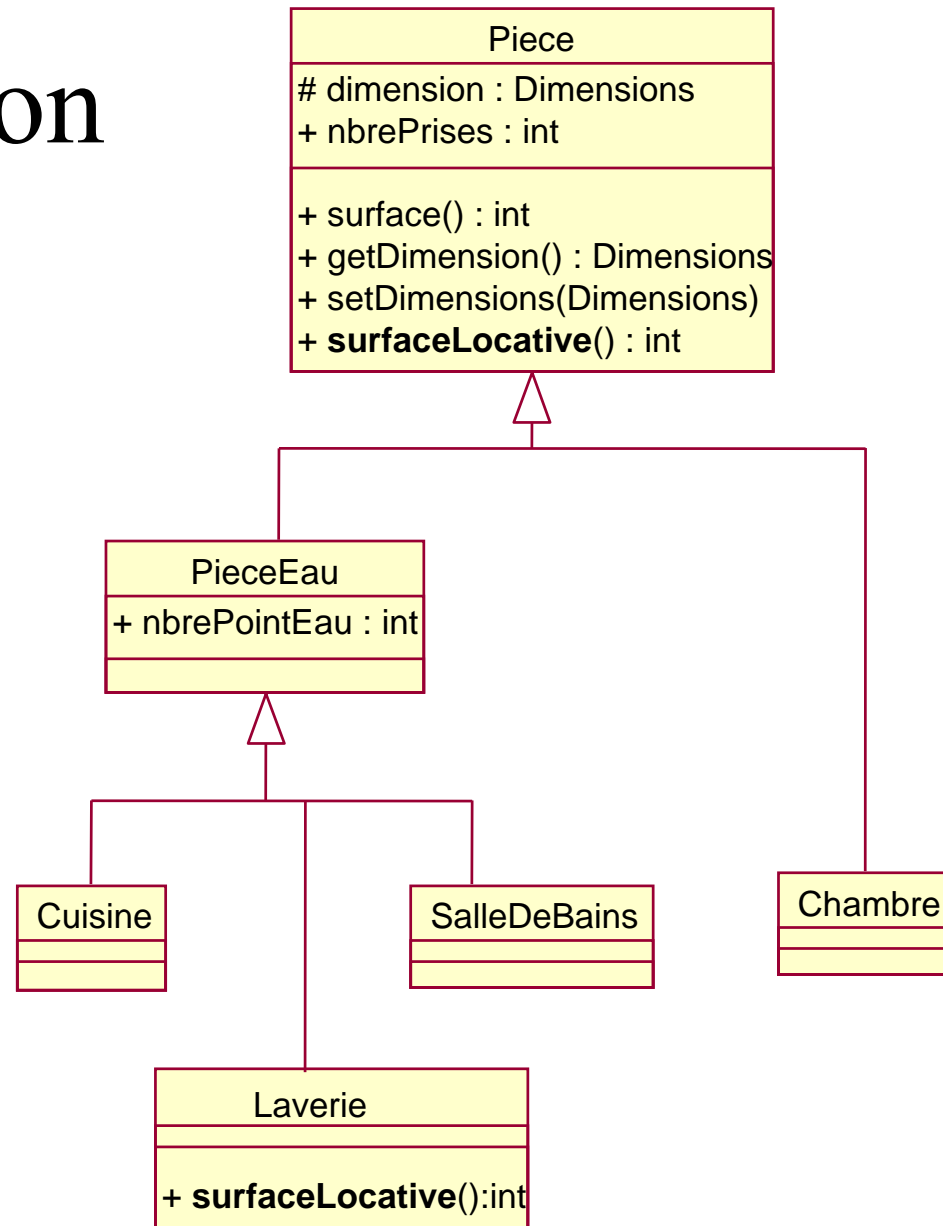
« Pattern » Accesseurs

```
public class Piece {  
    protected Dimensions dimension = new Dimensions();  
    public void setDimension(Dimensions dimension){this.dimension=dimension;}  
    public Dimensions getDimension(){return dimension;}  
    public int surface(){ return dimension.larg * dimension.long ; }  
}  
public class Cuisine extends Piece{  
  
}  
....  
public class Chambre extends Piece{  
  
}
```

Arbre d'héritage



spécialisation



Méthode spécialisée

Spécialisation java

```
public class Piece {
    protected Dimensions dimension = new Dimension();
    public setDimension(Dimensions dimension){this.dimension=dimension;}
    public Dimensions getDimension(){return dimension;}
    public int surface(){ return dimension.larg * dimension.long ; }
    public int surfaceLocative(){ return surface() ; }
}

....

public class PieceEau extends Piece{
    public int nbrePointEau;
}

public class Laverie extends PieceEau {
    public int surfaceLocative(){ return super.surfaceLocative()/2 ; }
}
```

Appel des méthodes spécialisées

```
public class SousSol {  
    protected Piece rangement;           //c'est une pièce  
    protected Garage garage = new Garage();  
  
    ...  
    rangement = new Laverie();           //en fait cette pièce est une laverie  
}  
  
int sl= leBas. rangement .surfaceLocative();  
int pe= leBas. rangement .nbrePointEau;
```

Constructeur par défaut

```
public class PieceEau extends Piece {  
    public int nbrePointEau;  
    public PieceEau() {}  
}
```

```
public class PieceEau extends Piece {  
    public int nbrePointEau;  
    // le constructeur d'arité nulle existe par défaut en l'absence de tout autre  
}
```

Constructeurs des super-classes

```
public class PieceEau extends Piece{  
    public int nbrePointEau;  
    public PieceEau(){  
        super();  
    }  
}
```

```
public class PieceEau extends Piece{  
    public int nbrePointEau;  
    public PieceEau(){ //appel de super() par défaut  
}
```

surcharge

```
public class PieceEau {  
    public int nbrePointEau;  
  
    public PieceEau() {}  
  
    public PieceEau(int nbrePointEau) {  
        this.nbrePointEau= nbrePointEau;  
    }  
  
    public PieceEau(int nbrePointEau, Dimensions dimension) {  
        this.dimension= dimension;  
        this.nbrePointEau= nbrePointEau;  
    }  
}
```

Appel de super()

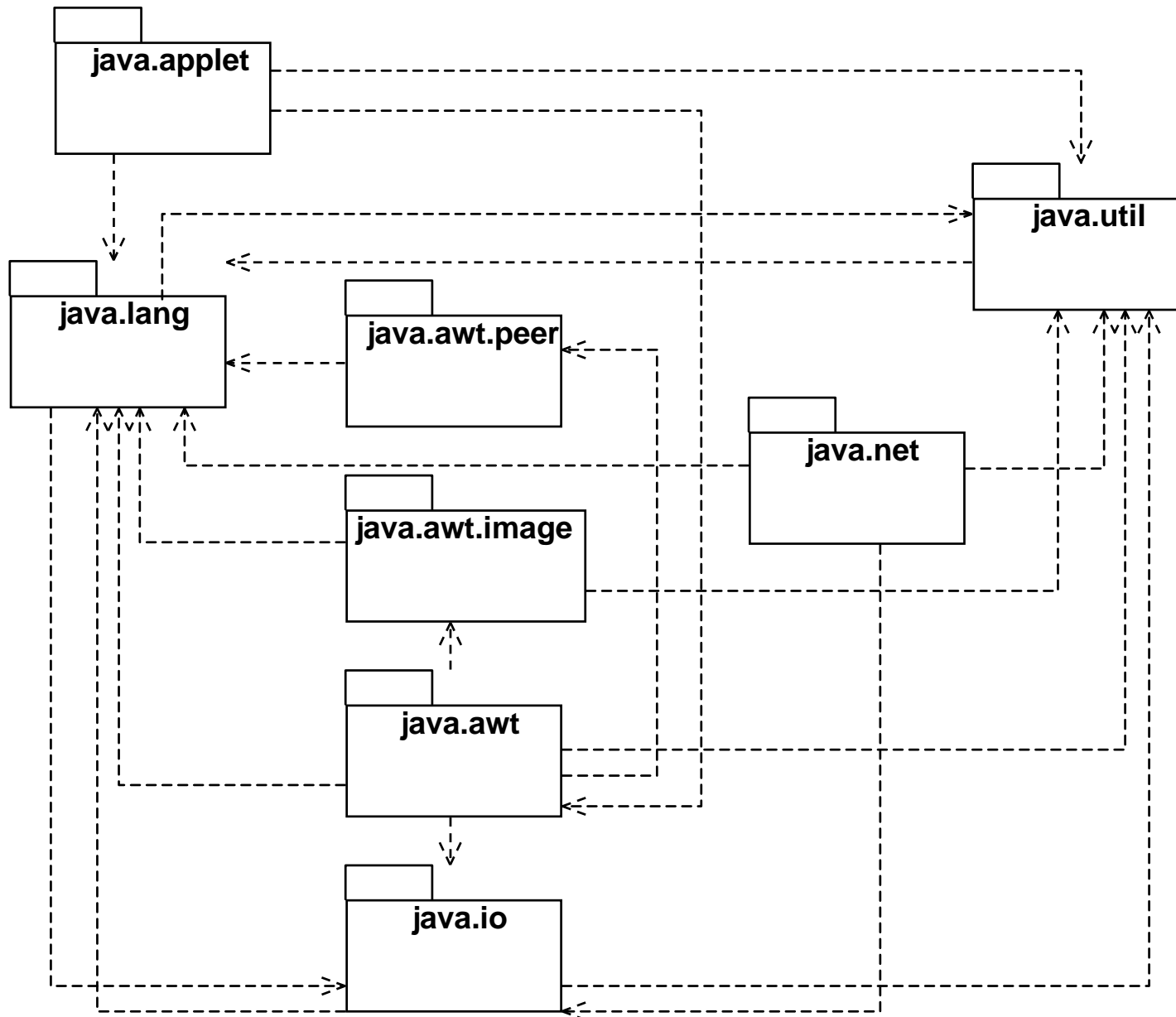
```
public class Piece {  
    protected Dimensions dimension ;  
    public Piece(Dimensions dimension ){  
        this.dimension = dimension;  
    }  
    ....  
}
```

```
public class PieceEau {  
    public int nbrePointEau;  
    public PieceEau(int nbrePointEau, Dimensions dimension){  
        super(dimension); //appel du constructeur de superclasse de même arité  
        this.nbrePointEau= nbrePointEau;  
    }  
    ...  
}
```

Appel de this()

```
public class PieceEau {  
    public int nbrePointEau;  
  
    public PieceEau(){  
        this(1);           //appel du constructeur de même classe d'arité 1  
    }  
  
    public PieceEau(int nbrePointEau){  
        this.nbrePointEau= nbrePointEau;  
    }  
}
```

De l'usage des packages



Packages et Classes

