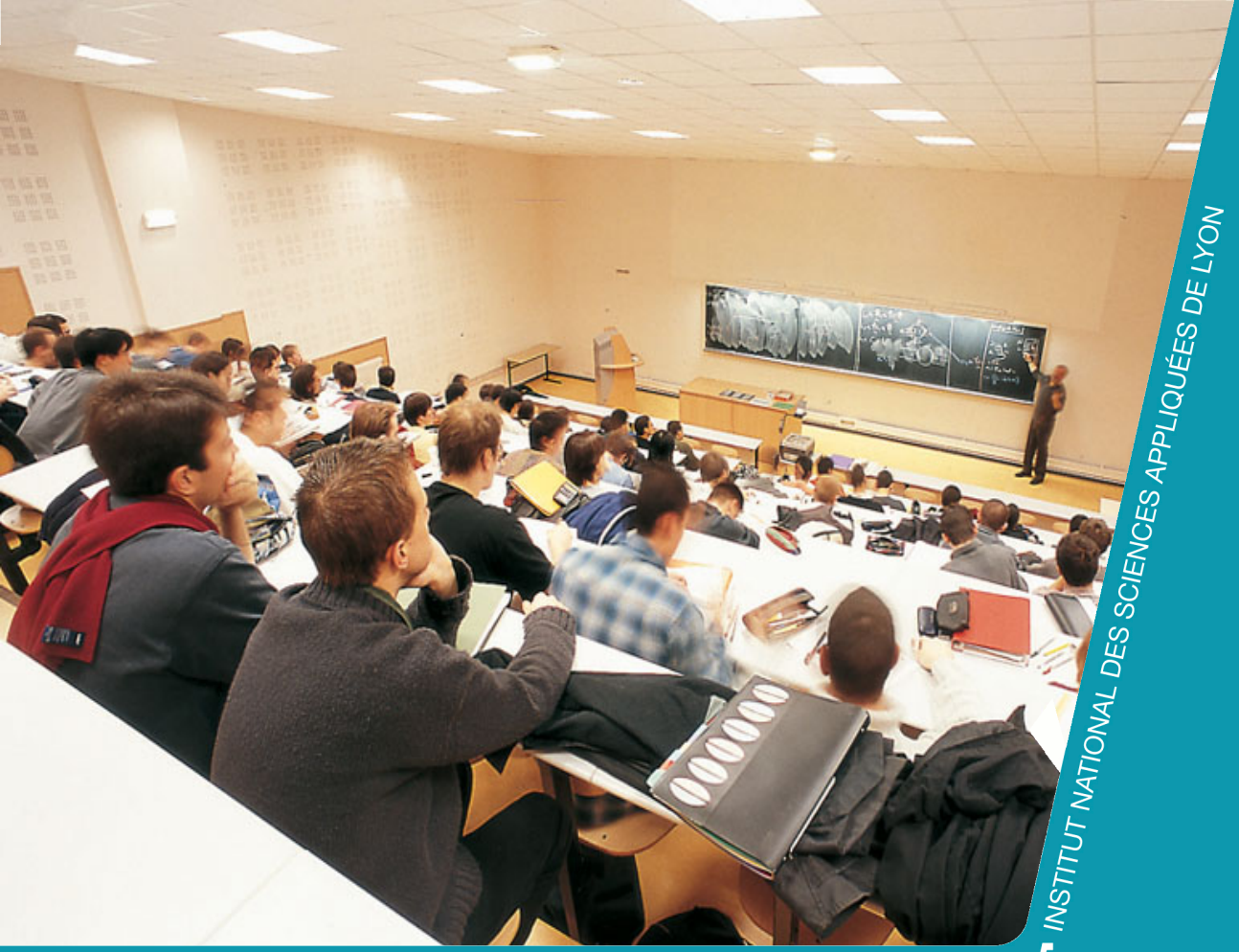


INSTITUT NATIONAL DES SCIENCES APPLIQUÉES DE LYON

FORMATION

Introduction à la programmation orientée objet sous Java

membre de
UNIVERSITÉ DE LYON

INSA LYON

Juste au cas où ….

- Qui suis-je ? Et comment me joindre ?
- Si vous avez des difficultés :
  - Internet est vaste : **soyez autonomes**.
- Si ça ne répond pas à vos questions :
  - Envoyez un mail à votre enseignant de TD

    (prénom.nom@insa-lyon.fr)
- Si ça contredit le cours :
  - Contactez moi, je vous expliquerai comment éluder la question

Organisation de l'année

- Semestre 1 : ***Initiation à la programmation orientée objet***
  - Cours : 14h
  - TD/TP : 24h
  - Évaluation :
    - 1 interrogation de 2h
    - 1 DS de 3h
    - QCMs …. peut être …. ou pas …. suivant notre humeur ….

- Semestre 2 : ***Projet de programmation*** *(par groupes de 4)*
  - Suivis de projet : 20h
  - Évaluation : rendus divers + soutenance

Mises en garde ...

- Cette année, nous utilisons les acquis de l'an dernier, mais nous recommençons **quelque chose de totalement nouveau** !
- Un étudiant qui était en difficulté en informatique l'an dernier pourra s'en sortir cette année s'il **suit** !
- **Ne ratez pas le virage !**
- Posez des **questions** *(en cours et en TD)*

- Point important : **Prenez des notes !**

Programmation Orientée Objet sous Java
Contenu du cours

- Cours 1
  - Rappels
  - Principes généraux de l'orienté objet :
    - *Classes, attributs, méthodes*
    - *Création d'objets, constructeurs*
    - *Variables et méthodes de classe*
    - *This : l'objet courant*
    - *Surcharge de méthodes*
    - *Visibilité*
  - Bons usages
    - *Accesseurs et encapsulation*

Version simplifiée vue l'année dernière ...

- Structure générale :
  - 1 **Classe** = 1 fichier **.java** *(code source)* ou **.class** *(bytecode)*
  - 1 **Classe** = ensemble de **méthodes**
  - **Programme** = des **Classes** dont une contient un **main**
- Déroulement
  - L'exécution d'une méthode se fait **de haut en bas**
  - L'exécution d'un programme consiste à exécuter son **main**
- Commandes pour l'exécution
  - javac : **compilation** d'un code source en bytecode
  - java : machine virtuelle qui **interprète** du bytecode

Convention de syntaxe en JAVA

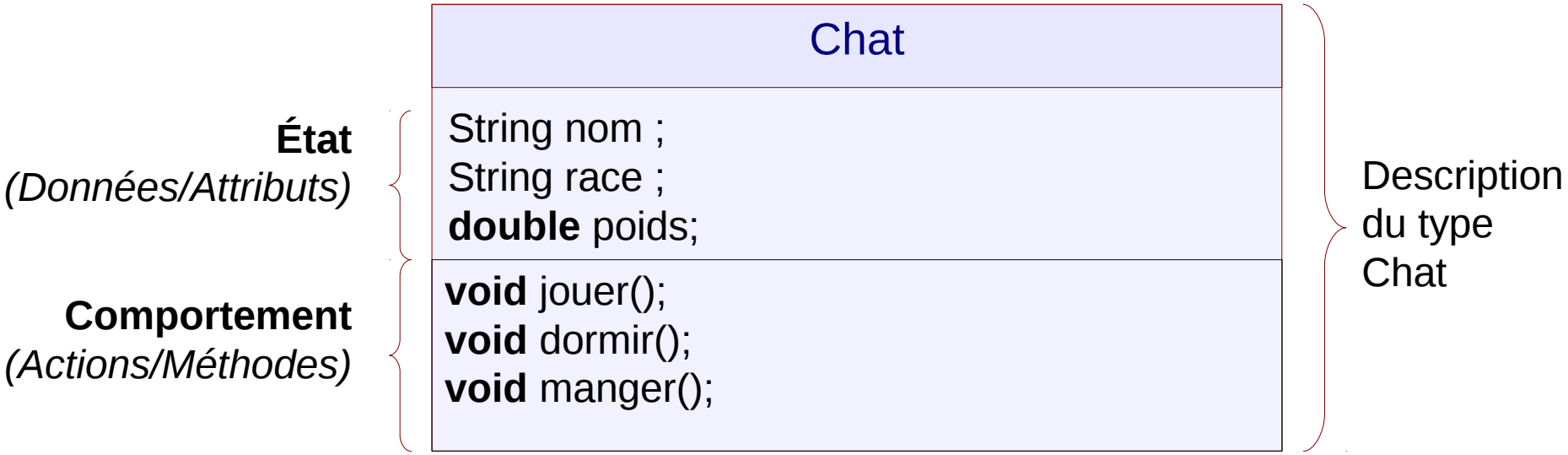
Type	Commentaire	Exemples
Classes	Les noms de classes devraient être des noms communs. Ils doivent être **capitalisés** et commencer par une **majuscule**.	FenetrePrincipale, SelecteurDeFichiers
Méthodes	Les noms de méthodes devraient être, ou contenir des verbes. Ils doivent être **capitalisés** et commencer par une **minuscule**.	fermeLaFenetre(), donnerDeLArgent(), manger(),
Variables	Les noms des variables doivent être expressifs. Il faut limiter autant que possible les noms de variables à 1 caractère. Ils ont les **mêmes règles de syntaxe que les méthodes**.	nombrePopcorn, tailleAppartement
Constantes	Les noms de constantes doivent être **écrits en majuscules**. S'ils sont composés de plusieurs mots, alors la barre de soulignement ('_') est utilisée comme séparateur.	LARGEUR_MAX, ID_COURANT

Principes de l'orienté objet
Second point de vue

- Orienté objet : création possible de types
  - classe = type *(Générateur d'objets / moule)*
  - objet = variable ayant une classe comme type
- Un objet est caractérisé par :
  - des **données** *(appelées **attributs**)*
  - des **actions** *(**méthodes**)*
- On distingue :
  - La description des objets : **CLASSE** (des chats, des voitures)
  - Les objets de chaque type : **INSTANCE** (ce chat, cette voiture)

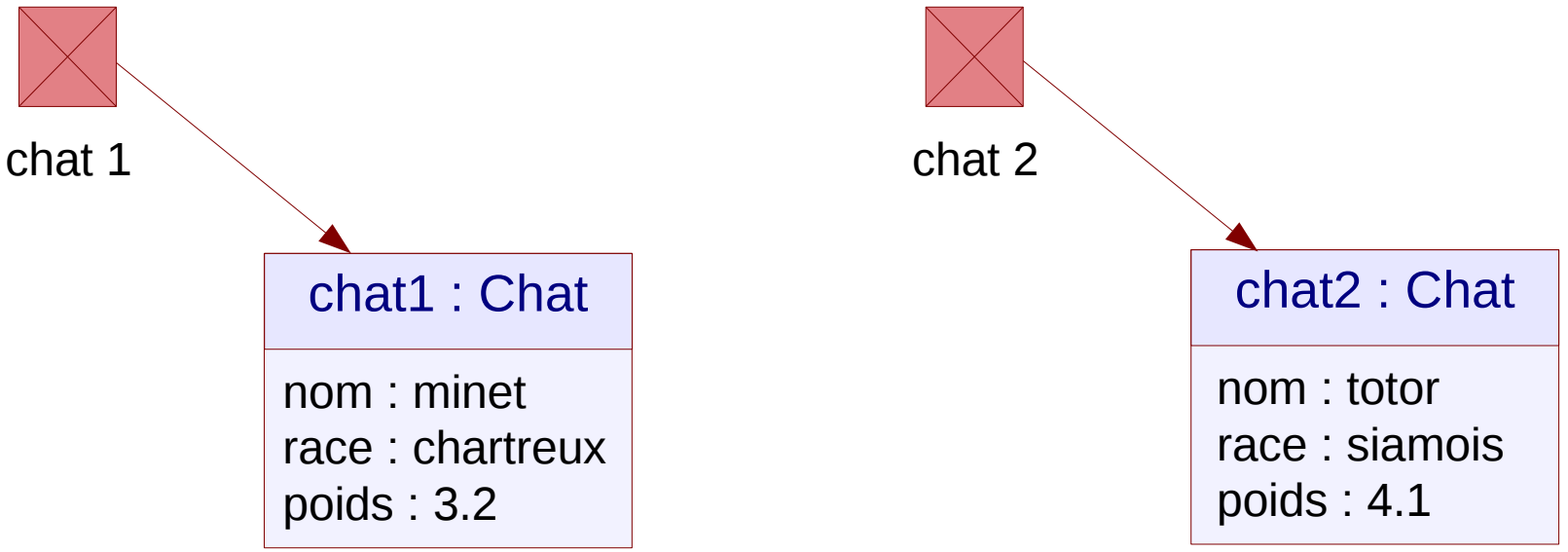
Qu'est ce qu'un objet décrivant un chat ?
Notation UML

- Objet : **type** contenant actions+données
- Class = déclaration du type *(moule / modèle / mode d'emploi)*



Qu'est ce qu'un objet décrivant un chat ?
Notation UML

- Instance = un élément/objet du type donné *(moulage)*



Qu'est ce qu'un chat en Java ?

Attributs
(données)

Méthodes
(actions)

```
public class Chat {
    String nom ;
    String race ;
    double poids ;
    public void manger(){
        double p = 0.5+(2*Math.random()) ;
        System.out.println(nom+" mange et prend "+p+"kilos.") ;
        poids = p+poids;
    }
    public void dormir(){
        System.out.println(nom+" dort.") ;
    }
    public void jouer(){
        double p = 0.1+(0.2*Math.random()) ;
        System.out.println(nom+" joue et perd "+p+"kilos.") ;
        poids = p-poids;
    }
}
```

Qu'est ce qu'un chat en Java ?

Attributs
(données)

Méthodes
(actions)

```
public class Chat {
    String nom ;
    String race ;
    double poids ;
    public void manger(){
        double p = 0.5+(2*Math.random()) ;
        System.out.println(nom+" mange et prend "+p+"kilos.") ;
        poids = p+poids;
    }
    public void dormir(){
        System.out.println(nom+" dort.") ;
    }
}
```

Attribut :

- Variable contenue dans une classe
- Accessible depuis toutes les méthodes de la classe
- Durée de vie = durée de vie de l'objet le contenant

Qu'est ce qu'un chat en Java ?

Attributs
(données)

Méthodes
(actions)

```
public class Chat {
    String nom ;
    String race ;
    double poids ;
    public void manger(){
        double p = 0.5+(2*Math.random()) ;
        System.out.println(nom+" mange et prend "+p+"kilos.") ;
```

Et **l'initialisation** des attributs,
comment la fait on ?

```
    }
    public void jouer(){
        double p = 0.1+(0.2*Math.random()) ;
        System.out.println(nom+" joue et perd "+p+"kilos.") ;
        poids = p-poids;
    }
}
```

Constructeur : initialisation des attributs

Attributs *(données)*

Constructeur *(Même nom que la classe)*

Méthodes *(actions)*

```
public class Chat {
    String nom ;
    String race ;
    double poids ;

    public Chat(String unNom, String uneRace, double unPoids){
        nom=unNom ;
        race=uneRace ;
        poids=unPoids;
    }

    public void manger(){  (...)  }
    public void dormir(){  (...)  }
    public void jouer(){  (...)  }
}
```

Attention !
Le constructeur est une méthode particulière.
Il n'a pas de type de retour déclaré !

Structure générale d'une classe

```
public class NomDeClasse {
    // Déclaration des attributs (Variables globales)
    String nom ;
    (...)

    // Définition des constructeurs
    public nomDeClasse(String unNom, ...){ (...) }
    (...)

    // Définition des méthodes
    public void dormir(){  (...)  }
    (...)
}
```

Instanciation d'un objet

- Constructeur :
  - Action qui n'est exécutée que lors de l'instanciation d'un objet
  - Rôle : initialisation des données d'un objet que l'on crée
  - Appel : via le mot clef **new**

Déclaration

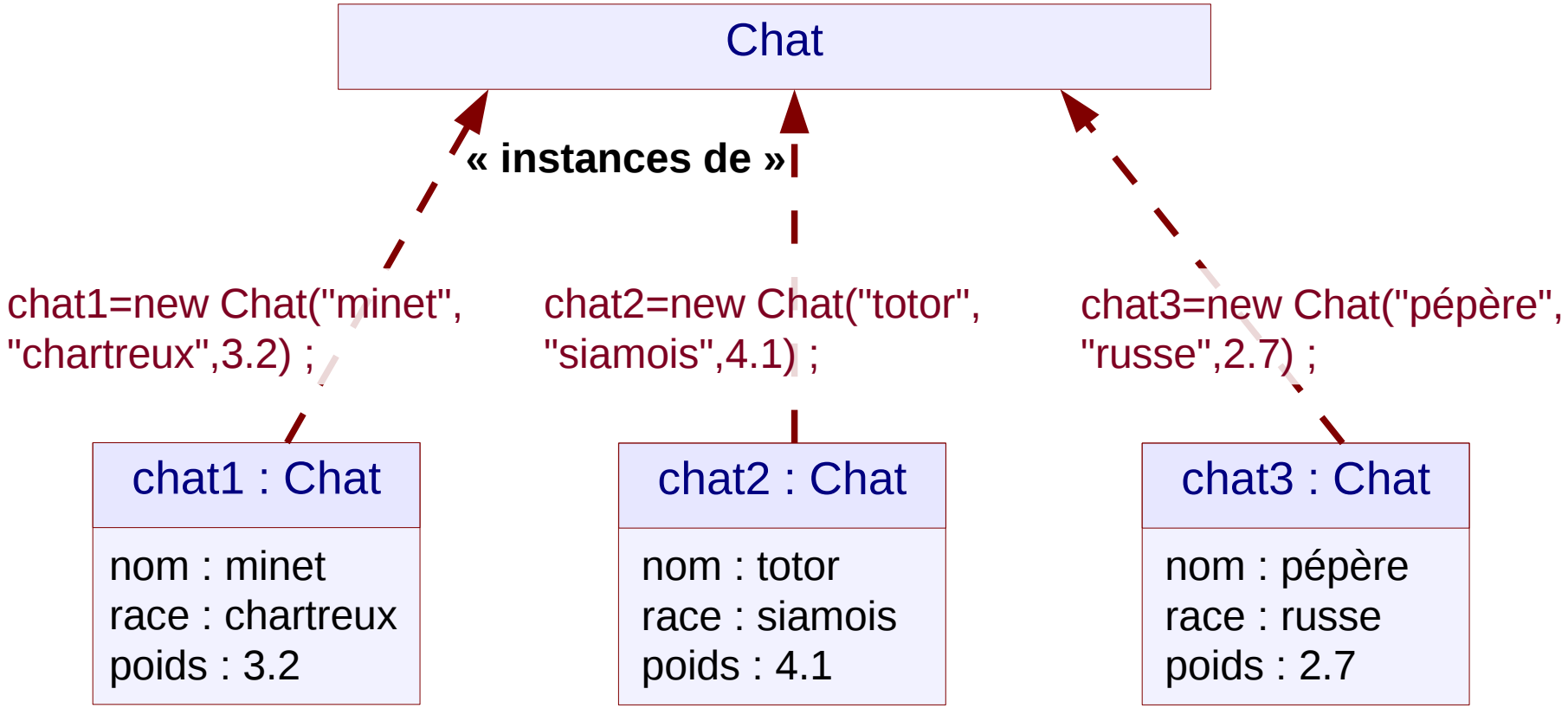
Instanciation
*(Appel du constructeur via **new**)*

```
public class VieDeChat {
    public static void main(String[] args) {
        Chat monChat ;
        monChat = new Chat("Félix", "Siamois", 4.2) ;

        monChat.manger();
        monChat.dormir();
        monChat.jouer();
    }
}
```

Instanciation d'un objet

- Chaque objet chat instance de la classe **Chat** possédera son propre **nom**, sa propre **race** et son propre **poids**



Instanciation d'un objet

- Création d'un objet à partir d'une **autre** classe:

  MaClasse monObjet= **new** MaClasse (liste_de_paramètres);

```
public class VieDeChat {
        public static void main(String[] args) {
                Chat monChat ;
                monChat = new Chat("Félix", "Siamois", 4.2) ;

                monChat.manger();
                MonChat.poids++ ;
                monChat.dormir();
                monChat.jouer();
        }
}
```

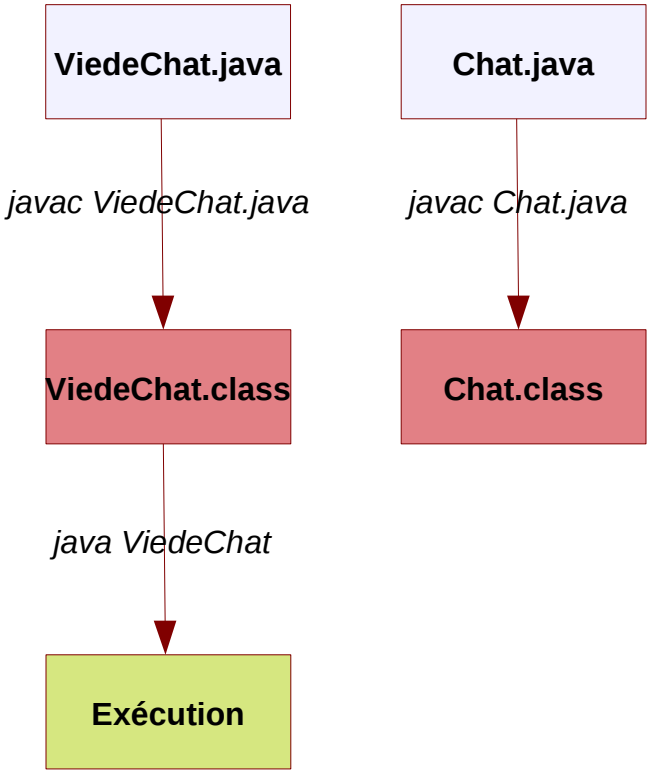
```
public class Chat {
        String nom ;
        String race ;
        double poids ;

        public Chat(String unNom, String uneRace, double
unPoids){
                nom=unNom ;
                race=uneRace ;
                poids=unPoids;
        }

        public void manger(){  (...)  }
        public void dormir(){  (...)  }
        public void jouer(){  (...)  }
}
```

Compilation séparée

- Un programme peut-être décomposé en de multiples fichiers sources (.java). Chaque fichier est **compilé séparément**.
- Puis, **à l'exécution**, la fonction **main** est recherchée et la machine virtuelle est démarrée sur cette fonction.
- Les diverses **classes externes** (à la classe contenant le main) utilisées **sont chargées** au fur et à mesure de leur utilisation. Tout se fait **dynamiquement**.



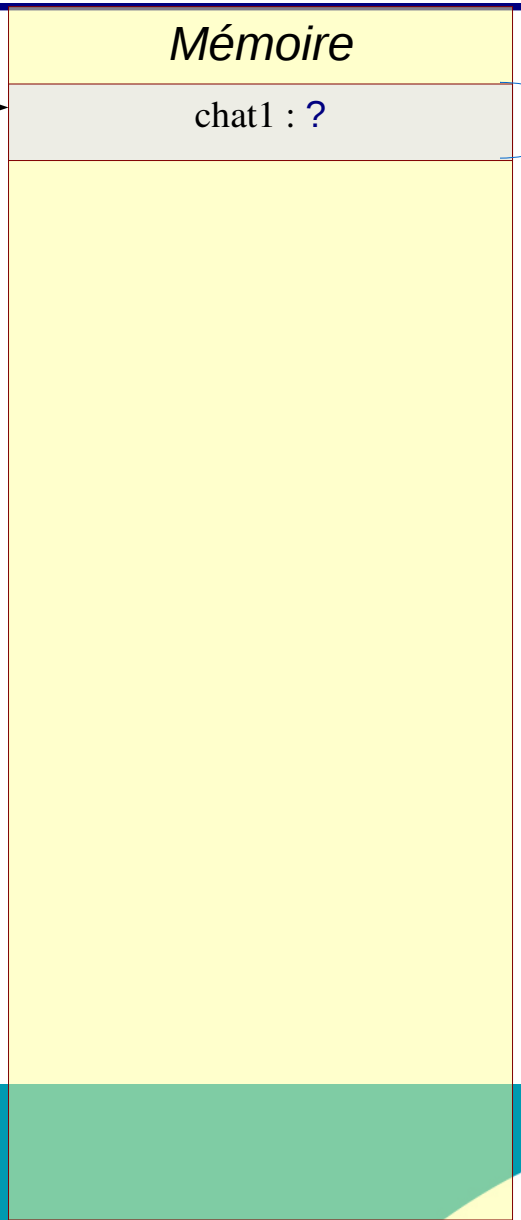
Instanciation d'un objet

- L'appel du constructeur déclenche **trois** étapes:
  - Création d'une **zone mémoire** capable de contenir les attributs
  - **Initialisation** des attributs
  - Renvoi d'une **référence** à cette zone mémoire *(**l'adresse**)*
- Désormais l'objet existe et sera accessible via cette référence.

Instanciation d'un objet

Déclaration d'une variable de type objet

Chat chat1;



null : valeur contenue dans variable de type non primitif qui n'est pas initialisée.

Ex : **int**[] t ; *// t vaut null à ce moment là*

Instanciation d'un objet

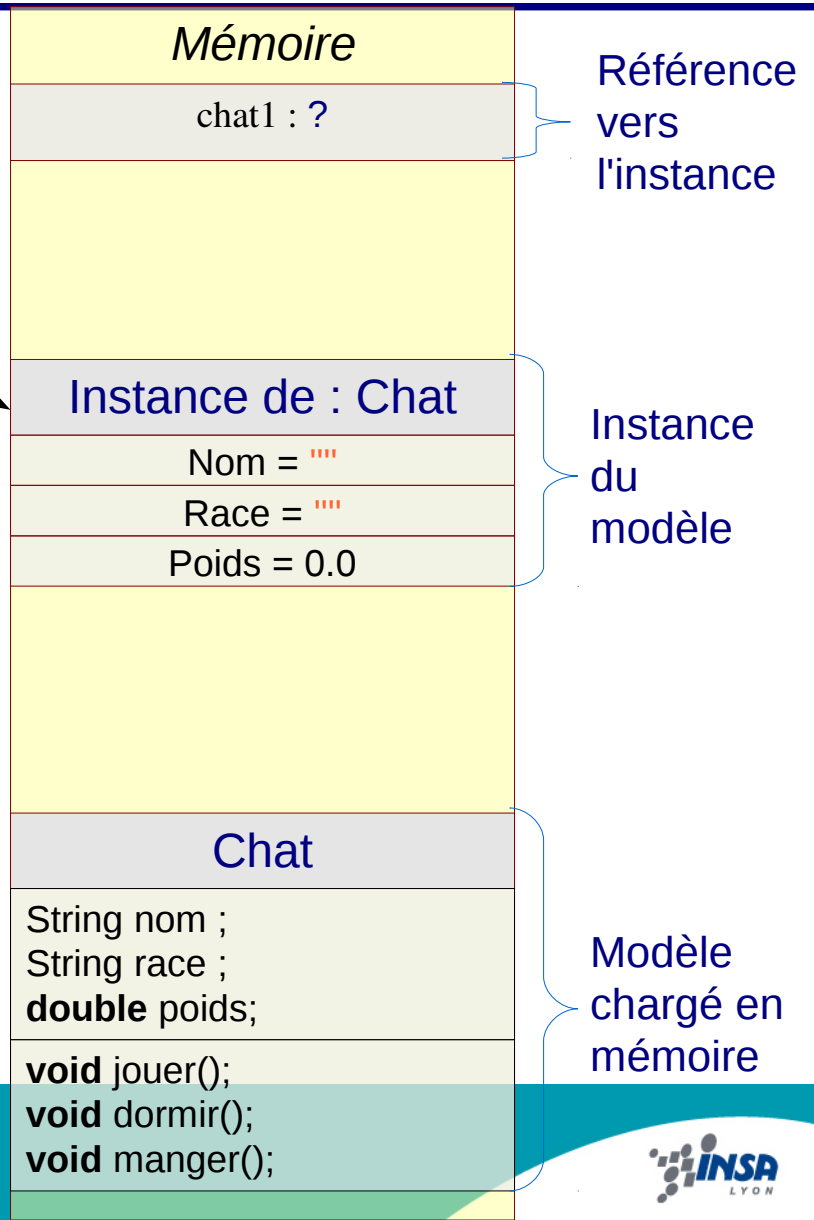
1/3 : Réservation de l'espace mémoire

```
Chat chat1;

chat1 = new Chat("Minette","Siamois",4.5);
```

La classe associée est chargée en mémoire.

L'instance est créée en mémoire avec des attributs initialisés à « 0 »



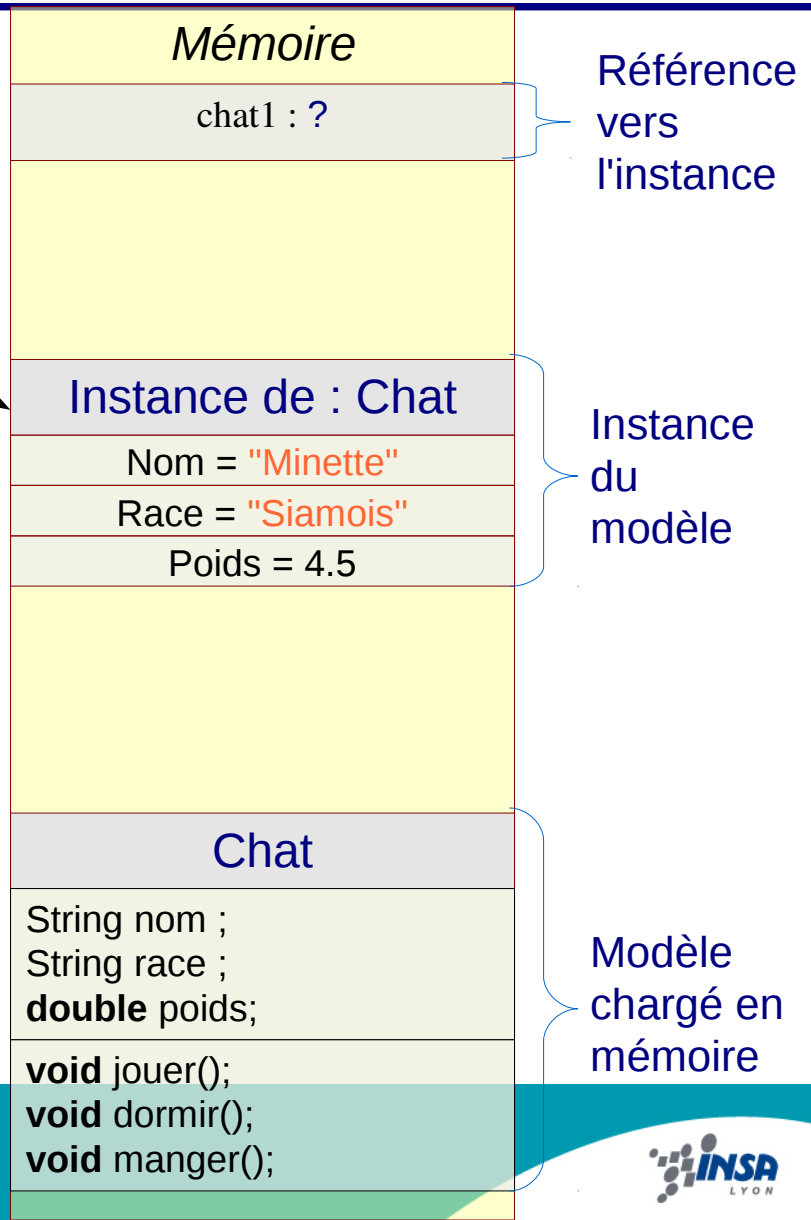
Instanciation d'un objet

2/3 : Exécution du constructeur appelé

```
Chat chat1;

chat1 = new Chat("Minette","Siamois",4.5);
```

Initialisation des attributs

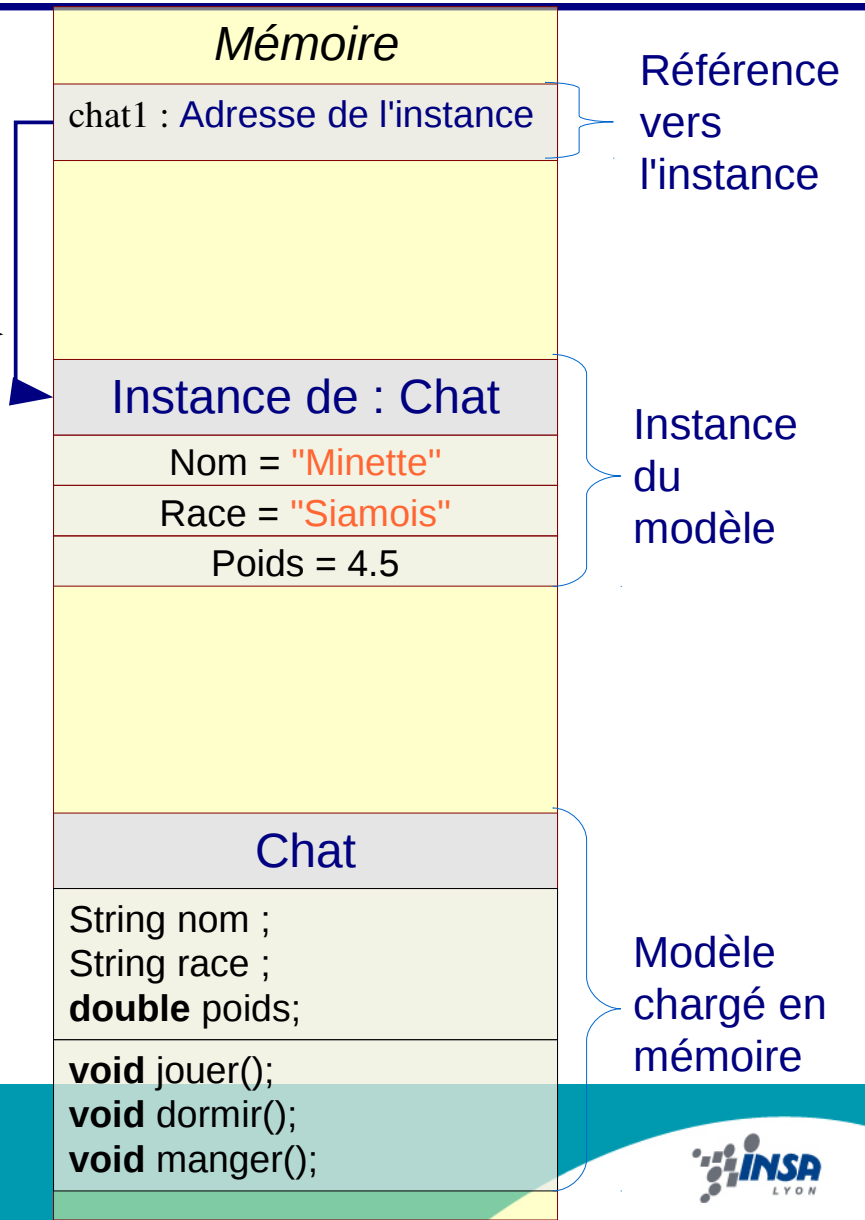


Instanciation d'un objet

3/3 : Renvoie de la référence vers l'instance créée

```
Chat chat1;

chat1 = new Chat("Minette","Siamois",4.5);
```



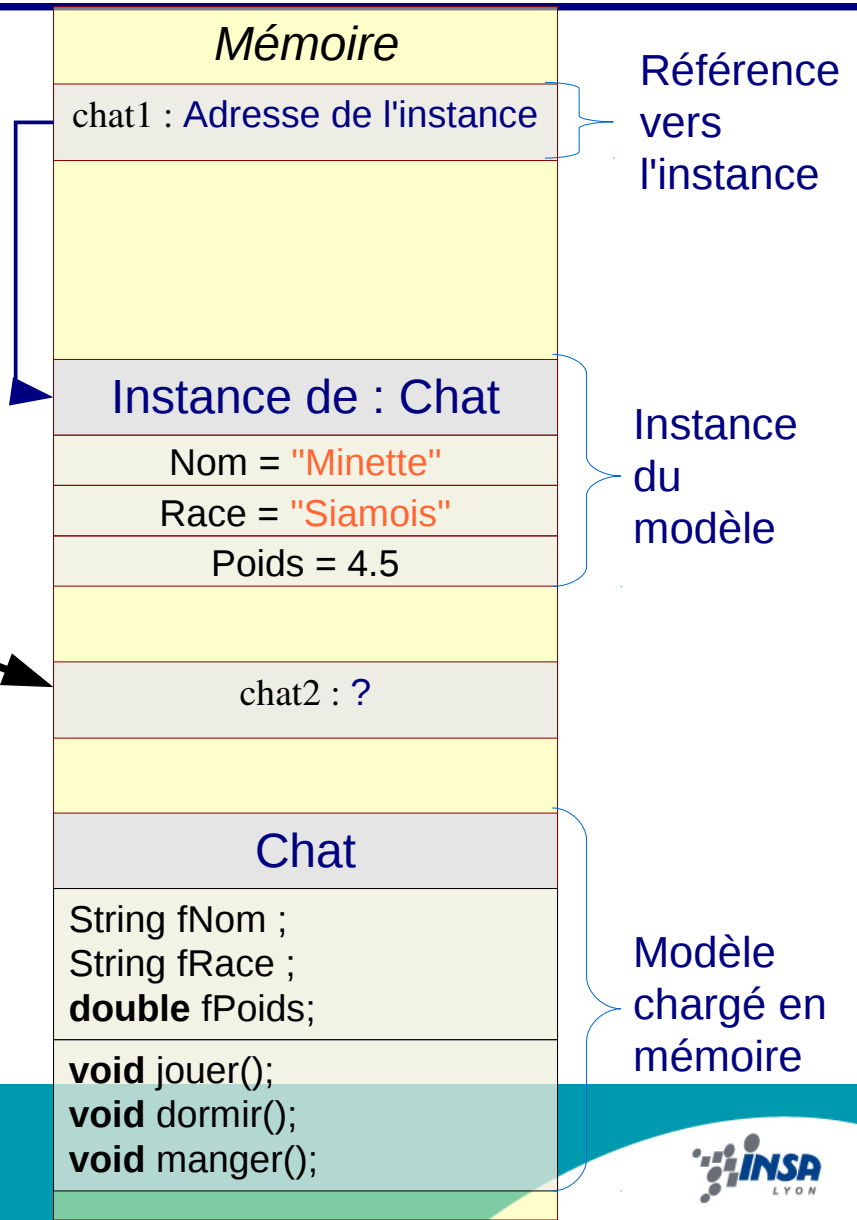
La référence vers l'instance créée est renvoyée en résultat du new.

Rappel : Un objet est un type non primitif

```
Chat chat1;

chat1 = new Chat("Minette","Siamois",4.5);

Chat chat2;
```



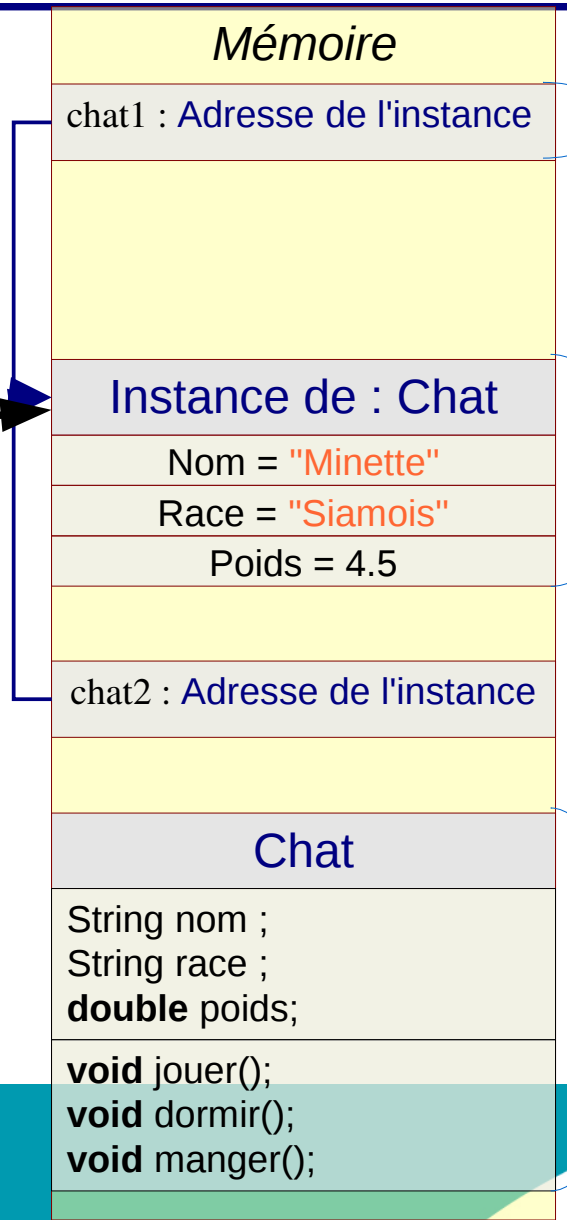
Rappel : Un objet est un type non primitif

```
Chat chat1;

chat1 = new Chat("Minette","Siamois",4.5);

Chat chat2;

chat2=chat1;
```



Affectation : *copie de l'adresse*

Maintenant : *Modifier **chat2** modifie **chat1***

Structure des objets

Un objet est constitué d'une partie **statique** et d'une partie **dynamique**

- Partie dynamique
  - son contenu varie d'une instance à une autre
  - son contenu peut varier durant la vie de l'objet
- Partie statique
  - son contenu ne varie pas d'une instance à une autre
  - un seul exemplaire pour l'ensemble des instances d'une classe

Et **static** dans tout ça ?

Cas des attributs

```
Chat chat1;
```

```
public class Chat {
    String nom ;
    String race ;
    double poids ;
    static int effectif=0;

    public Chat(String unNom, String uneRace, double unPoids){
        nom=unNom ;
        race=uneRace ;
        poids=unPoids ;
        effectif=effectif+1;
    }
```

Static = partagé par toutes les instances

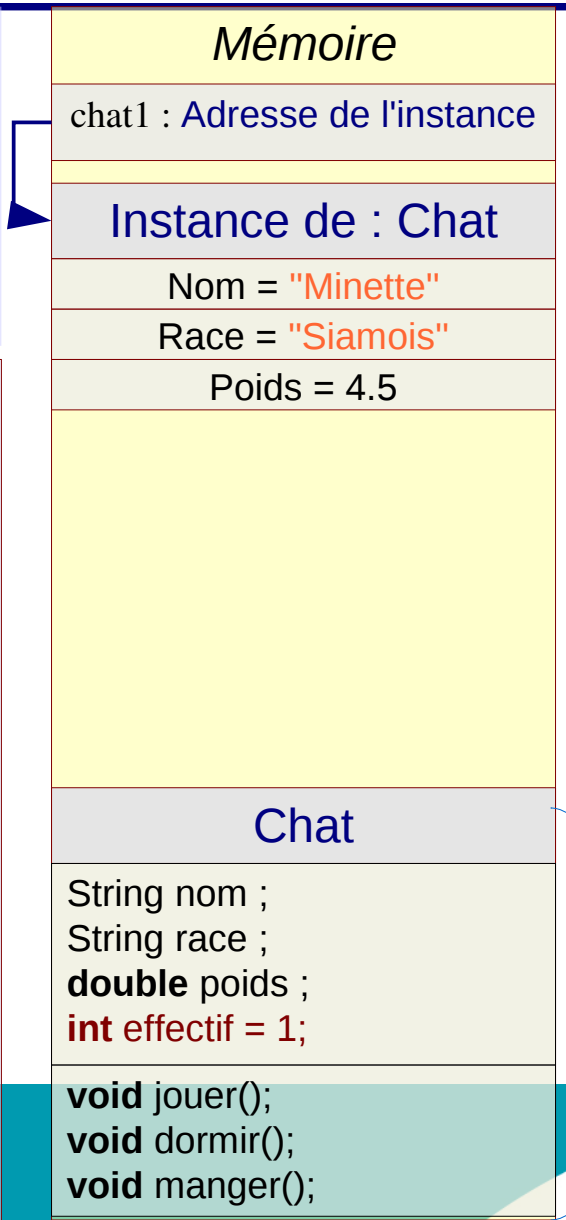
Et **static** dans tout ça ?

Cas des attributs

```
Chat chat1;
chat1 = new Chat("Minette","Siamois",4.5);
```

```
public class Chat {
    String nom ;
    String race ;
    double poids ;
    static int effectif=0;

    public Chat(String unNom, String uneRace, double unPoids){
        nom=unNom ;
        race=uneRace ;
        poids=unPoids ;
        effectif=effectif+1;
    }
```



Static = partagé par toutes les instances

Et **static** dans tout ça ?

Cas des attributs

```
Chat chat1;
chat1 = new Chat("Minette","Siamois",4.5);
Chat chat2;
```

```
public class Chat {
    String nom ;
    String race ;
    double poids ;
    static int effectif=0;

    public Chat(String unNom, String uneRace, double unPoids){
        nom=unNom ;
        race=uneRace ;
        poids=unPoids ;
        effectif=effectif+1;
    }
```

Static = partagé par toutes les instances

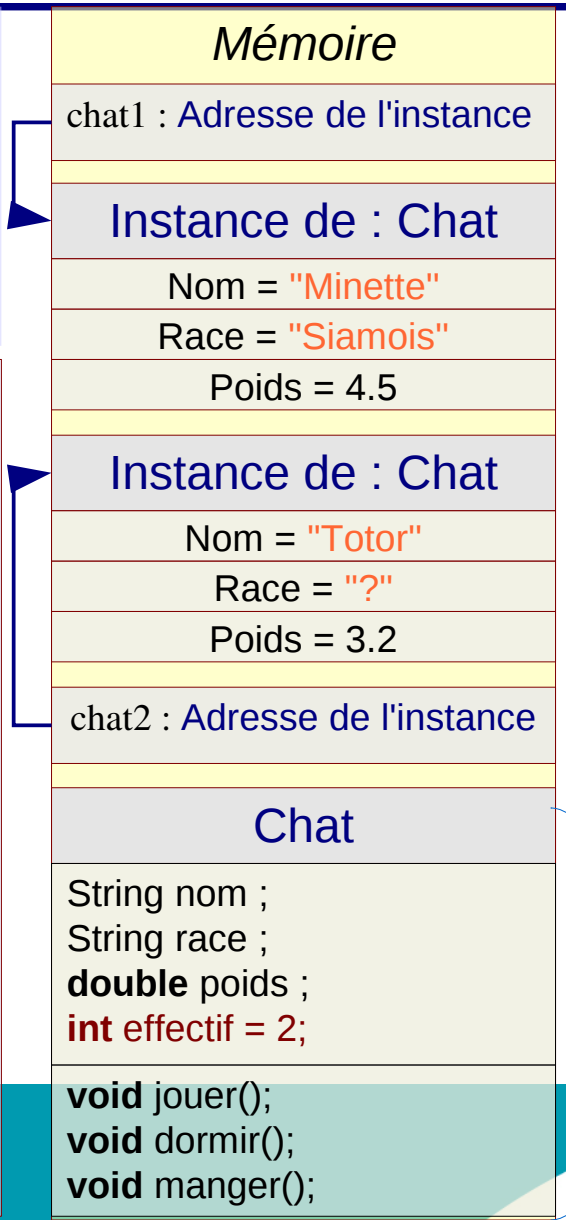
Et **static** dans tout ça ?

Cas des attributs

```
Chat chat1;
chat1 = new Chat("Minette","Siamois",4.5);
Chat chat2;
chat2 = new Chat("Totor","?",3.2);
// chat2.effectif == chat1.effectif
```

```
public class Chat {
    String nom ;
    String race ;
    double poids ;
    static int effectif=0;

    public Chat(String n, String r, double p){
        nom=unNom ;
        race=uneRace ;
        poids=unPoids ;
        effectif=effectif+1;
    }
    (...)
```



Static = partagé par toutes les instances

Et **static** dans tout ça ?

Cas des méthodes

- Attribut **static** :
  - Attribut partagé entre toutes les instances d'une classe

- Méthode **static** :
  - Méthode pouvant fonctionner sans instance de sa classe
  - Méthode n'accédant qu'à des attributs static ou ses paramètres
  - Exemples connus :

    Math.random() : génération d'un nombre aléatoire

    *Integer.parseInt(s) : conversion d'une String en **int***

    main(args) : point de départ des programmes

- Conséquence :
  - Dorénavant, plus aucune méthode ne sera **static** (sauf main)

this : pointeur vers l'objet courant

- Appel d'une méthode depuis l'extérieur de l'objet :

  MonObjet o = new MonObjet() ;

  o.methode(params) ;

- Appel d'une méthode depuis l'objet la contenant :

  methode(params) ;

  ou bien

  this.methode(params) ;

- **this** :

  Mot clef caractérisant l'objet courant

  Permet de lever des ambiguïtés

Exemples d'utilisation de **this**

```
public class Chat {
    String nom ;
    String race ;
    double poids ;

    /** Constructeur de chat initialisant les attributs */
    public Chat(String unNom, String race, double unPoids){
        nom=unNom ;          // this est facultatif ici
        this.race=race ;  // this est obligatoire ici, sinon ambiguïté
                          // car non respect de la convention de codage
        this.poids=unPoids ;    // this est facultatif ici
    }
    (...)
}
```

Surcharge de méthodes

- Au sein d'une même classe
- Même **nom**, même **visibilité** et même **type de retour**
- **Type ou nombre de paramètres** nécessairement **différents**

- Ex : Méthode existante dans le package System

  public void println(**int** x)

  public void println(**double** x)

  ~~**public int** println(**double** x)~~

  ~~**public void** println(**double** y)~~

Exemple de surcharge de constructeur

```
public class Chat {
    String nom ;
    String race ;
    double poids ;

    public Chat(String unNom, String uneRace, double unPoids){
        this.nom = unNom ;
        this.race = uneRace ;
        this.poids = unPoids ;
    }
    public Chat(String unNom, String uneRace){
        this.nom = unNom ;
        this.race = uneRace ;
        this.poids = 0.0 ;
    }
}
```

Exemple de surcharge de constructeur

```
public class Chat {
    String nom ;
    String race ;
    double poids ;

    public Chat(String unNom, String uneRace, double unPoids){
        this.nom = unNom ;
        this.race = uneRace ;
        this.poids = unPoids ;
    }
    public Chat(String unNom, String uneRace){
        this (unNom, uneRace, 0.0) ;
        // définir les constructeurs en cascade est plus sûr
    }
}
```

Exemple de surcharge de constructeur

```
public class Chat {
    String nom ;
    String race ;
    double poids ;

    public Chat(String unNom, String uneRace, double unPoids){
        this.nom = unNom ;
        this.race = uneRace ;
        this.poids = unPoids ;
    }
    public Chat(String unNom, String uneRace){
        this (unNom, uneRace, 0.0) ;
    }
    public Chat(String unNom, String uneRace){
        this (unNom, uneRace, Math.random()*20) ;
    }
}
```

Exemple de surcharge de constructeur

```
public class Chat {
    String nom ;
    String race ;
    double poids ;

    public Chat(String unNom, String uneRace, double unPoids){
        this.nom = unNom ;
        this.race = uneRace ;
        this.poids = unPoids ;
    }
    public Chat(String unNom, String uneRace){
        this (unNom, uneRace, 0.0) ;
    }
    public Chat(String unNom, String uneRace){
        this (unNom, uneRace, Math.random()*20) ;
    }
}
```

INTERDIT : pas deux fois la même signature

Exemple de surcharge de constructeur

```
public class Chat {
    String nom ;
    String race ;
    double poids ;

    public Chat(String unNom, String uneRace, double unPoids){
        this.nom = unNom ;
        this.race = uneRace ;
        this.poids = unPoids ;
    }
    public Chat(String unNom, String uneRace){
        this (unNom, uneRace, 0.0) ;
    }
    public Chat(String unNom, String uneRace){
        this (unNom, uneRace, Math.random()*20) ;
    }
}
```

INTERDIT : pas deux fois la même signature

Erreur à la compilation :

Chat.java:16: Chat(java.lang.String,java.lang.String) is already defined in Chat

public Chat(String unNom, String uneRace){

Bilan sur les principes de base des objets

- Définitions :
  - **Classe** = type *(c'est un moule)*
  - **Objet** = instance *(un élément généré à partir du moule)*
- Classe définie par :
  - **Attributs** (données) et **méthodes** (actions)
  - **Constructeur** : méthode particulière appelée par **new**
- **Static** :
  - Attributs partagés entre toutes les instances
  - Méthodes pouvant s'exécuter dans un contexte **static**
- **Surcharge** : multiples définitions de méthodes

Encapsulation

- Bon usage :
  - Les données d'un objet doivent être **privées**, c'est à dire **protégées** et accessibles (et surtout modifiables) uniquement au travers de méthodes prévues à cet effet
  - en JAVA, possible lors de leur définition d'agir sur la **visibilité** (accessibilité) des **membres** (attributs et méthodes) d'une classe vis à vis des autres classes
  - plusieurs niveaux de visibilité peuvent être définis en précédant d'un modificateur (**private, public, protected, -**)
  - Attention, le **niveau par défaut** est public

Encapsulation

Visibilité des membres d'une classe (en Java)

	public	**private**
classe	La classe peut être utilisée dans n'importe quelle classe	(Ce cas sera traité plus tard)
attribut	Attribut accessible directement depuis le code de n'importe quelle classe	Attribut accessible uniquement dans le code de la classe qui le définit (donc interdit en lecture/écriture en dehors du code de la classe)
méthode	Méthode pouvant être invoquée depuis code de n'importe quelle classe	Méthode utilisable uniquement dans le code de la classe qui la définit

- On verra les autres niveaux de visibilité (-, protected) en détail lorsque les notions d'héritage et de packages auront été abordées

Encapsulation
Cas par défaut

MonAnimalerie

```
public class MesAnimaux {

   public static void main(String[] args) {
      Chat chat1=new Chat("minet","siamois",1.2);
      chat1.poids = -20;
      //accès à l'attribut depuis l'extérieur
   }
}
```

Utilisation depuis une autre classe

Chat

Déclaration **sans** visibilité déclarée

```
public class Chat {
      String nom ;
      String race ;
      double poids ;

      public Chat(String unNom, String uneRace, double unPoids){
            this.nom = unNom ;
            this.race = uneRace ;
            this.poids = unPoids ;
      }
      public Chat(String unNom, String uneRace){
            this (unNom, uneRace, 0.0) ;
      }
}
```

- Conséquence : **toute** autre méthode de la classe et n'importe quelle autre classe peuvent avoir accès aux **attributs non protégés**

Encapsulation
Visibilité des attributs

MonAnimalerie

```
public class MesAnimaux {

   public static void main(String[] args) {
      Chat chat1=new Chat("minet","siamois",1.2);
      chat1.poids = -20;
      //accès à l'attribut depuis l'extérieur
   }
}
```

Utilisation depuis une autre classe

Chat

Encapsulation

```
public class Chat {
      private String nom ;
      private String race ;
      private double poids ;

      public Chat(String unNom, String uneRace, double unPoids){
            this.nom = unNom ;
            this.race = uneRace ;
            this.poids = unPoids ;
      }
      public Chat(String unNom, String uneRace){
            this (unNom, uneRace, 0.0) ;
      }
}
```

- **Erreur à la compilation** :

```
Mon Animalerie.java:5: poids has private access in Chat
      chat1.poids = -20;
            ^
```

- Conséquence : Les attributs sont donc bien **protégés en écriture** (ici, seule une méthode de la classe Chat peut les modifier)

Encapsulation

Modifier un attribut en contrôlant que sa valeur est correcte

MonAnimalerie

```
public class MesAnimaux {

    public static void main(String[] args) {
        Chat chat1=new Chat("minet","siamois",1.2);
        chat1.poids = -20;
        //le poids sera mis à zéro, car il n'est pas dans la fourchette fixée
    }
}
```

Utilisation depuis une autre classe

Chat

```
public class Chat {
    private String nom ;
    private String race ;
    private double poids ;

    public Chat(String unNom, String uneRace, double unPoids){
        this.nom = unNom ;
        this.race = uneRace ;
        setPoids(unPoids) ;
    }
    public void setPoids(double unPoids){
        if(unPoids >0 && unPoids < 20)
            this.poids = unPoids ;
        else
            this.poids = 0,0 ;
    }

    public double getPoids(){
        return this.poids ;
    }
}
```

- On écrit une méthode permettant cet accès AVEC contrôle des nouvelles valeurs : **mutateur (setter)**
- Réciproquement, pour **accéder** à la valeur il faut passer par une méthode de type **accesseur (getter)**

Encapsulation
Intérêt

- Accès au données ne se fait qu'au travers des méthodes publiques.

  Un objet ne peut être utilisé que de la manière prévue à la conception de sa classe, sans risque d'utilisation incohérente = **robustesse du code**

MonAnimalerie

```
public class MesAnimaux {

   public static void main(String[] args) {
      Chat chat1=new Chat("minet","siamois",1.2);
      chat1.poids = -20;
      //le poids sera mis à zéro, car il n'est pas dans la fourchette fixée
   }
}
```

Impossible d'avoir un chat dans un état incohérent (poids > 0 et poids < 20)

Le setteur se charge de le vérifier

Chat

```
public class Chat {
      private String nom ;
      private String race ;
      private double poids ;

      public Chat(String unNom, String uneRace, double unPoids){...}

      public void setPoids(double unPoids){
            if(unPoids >0 && unPoids < 20)
                  this.poids = unPoids ;
            else
                  this.poids = 0,0 ;}

      public double getPoids(){...}
}
```

Encapsulation
Méthodes privées

Chat

```
public class Chat {
        private String nom ;
        private String race ;
        private double poids ;
        private double taille ;

        public Chat(String unNom, String uneRace, double
unPoids, double uneTaille){
                this.nom = unNom ;
                this.race = uneRace ;
                this.poids = unPoids ;
                this.taille=uneTaille ;
        }
        private double imc(){
                return this.poids/(this.taille*this.taille) ;
        }
        public String bilanSante(){
                if (this.imc()>25
                        return "Votre chat est en surpoids" ;
                else
                        return "Votre chat est en bonne santé" ;
        }
}
```

- Une classe peut définir des méthodes privées à **usage interne**
- Une méthode privée ne peut plus être invoquée **en dehors du code de la classe** où elle est définie

MonAnimalerie

```
public class MesAnimaux {

    public static void main(String[] args) {
        Chat chat1=new  Chat("minet","siamois",1,2);
        ...
        chat1.imc()
        // produit une erreur à la compilation
    }
}
```