

M2103 – Bases de la Programmation Orientée Objets



Java – Cours 2

Objets & Classes

Qu'est-ce qu'un Objet ?

- Le monde 'informatique' consiste en des objets, caractérisés par :

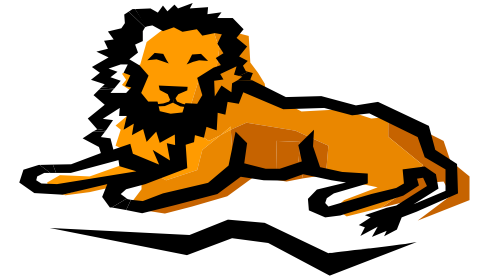
- Attributs

nom : Rusty
couleur : marron
largeur : 40 cm
hauteur : 45 cm



objetChien

largeur : 2m
hauteur : 1m



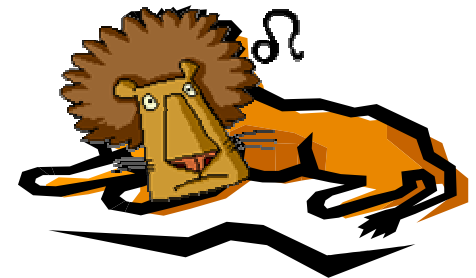
objetLion

- Comportements (ou facultés)

court
poursuit
aboie



court
chasse
rugit



Chaque Objet a...

- Un identificateur unique

chien321



Rusty
marron
40 cm
45 cm

Sprint
marron clair
1 m
75 cm



sprint

Scott
noir
30 cm
20 cm



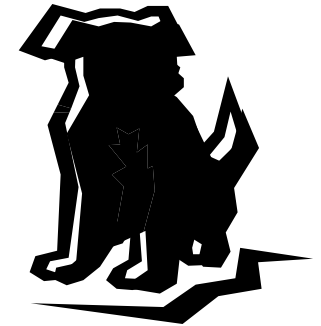
scot9

3 objets Chien

Chien

nom
couleur
largeur
hauteur

court
poursuit
aboie



1
classe
Chien

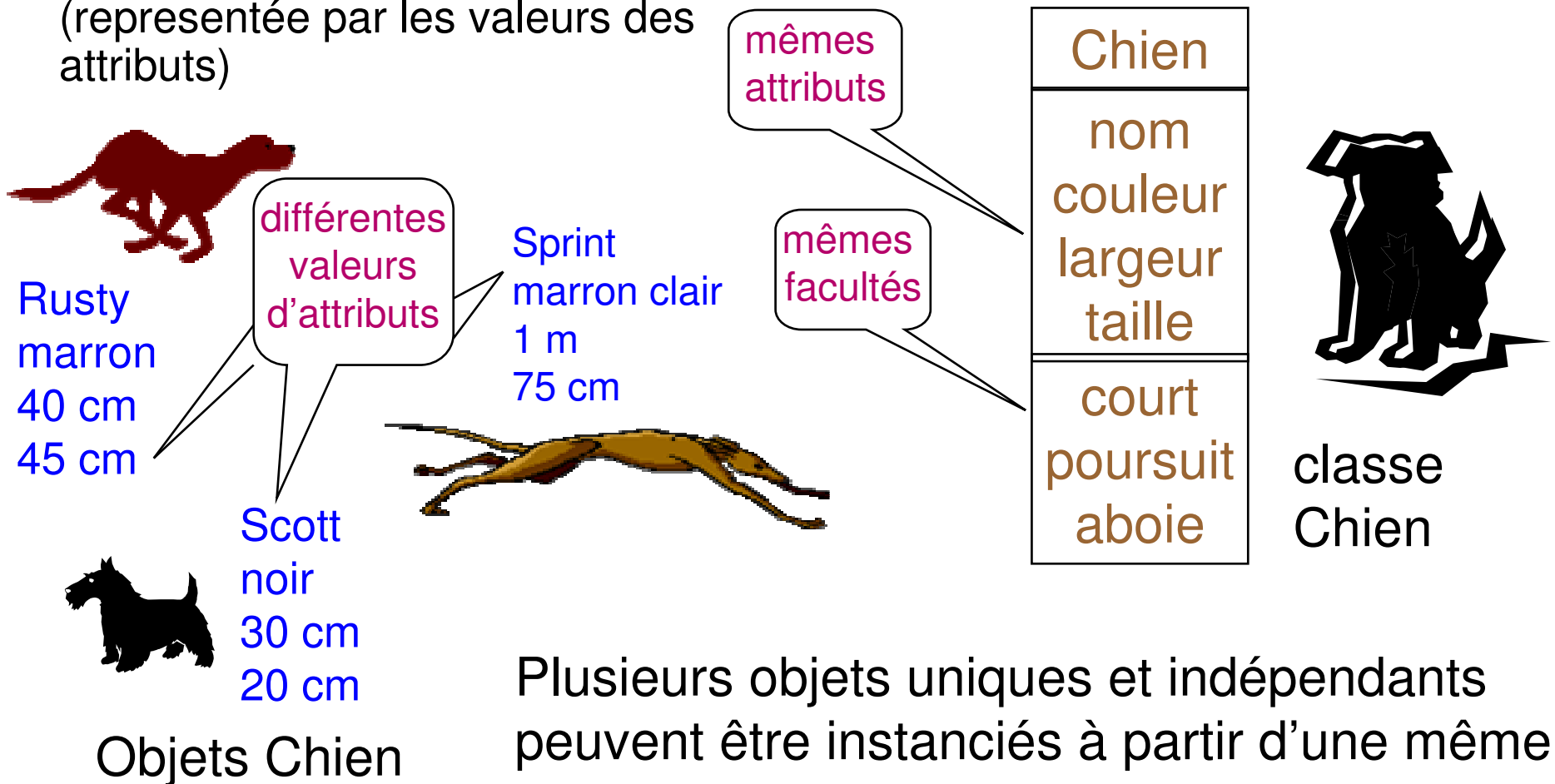
Les Classes sont :

- Un ensemble d'objets avec les mêmes attributs et comportements/facultés :
 - Par exemple, `Chien`, `Chat`, `Personne`, `Client` sont des classes ou ensembles d'objets différents
- Une classe est la caractérisation des points communs de différentes entités.
- En programmation : un patron (“moule”) pour créer des objets (descriptions d'attributs et de méthodes pour les instances)

.

Les Instances (Objets) sont des...

- Individus, avec leur identité propre, (représentée par les valeurs des attributs)



Plusieurs objets uniques et indépendants peuvent être instanciés à partir d'une même classe.

Attributs

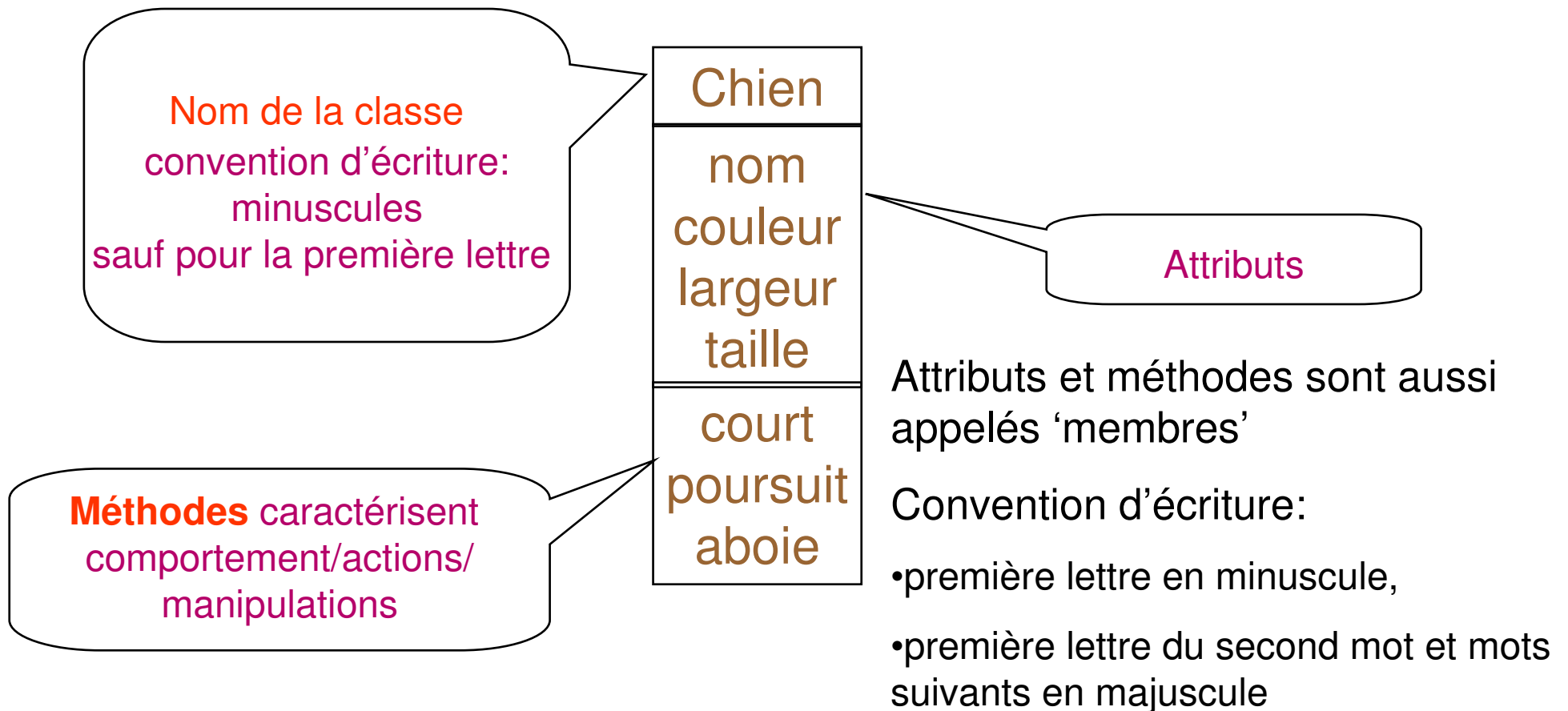
- Caractéristiques à propos d'un objet pouvant être décrites ou mesurées
 - E.g. age, largeur, poids, intelligence...
- Attributs décrivent l'état de l'objet à un instant particulier dans le temps
 - E.g. endormi : oui ou non
- Peuvent avoir un impact sur le comportement
 - E.g. si une personne est endormie, lui dire bonjour peut ne pas avoir d'effet; ceci est vraisemblablement différent si cette même personne est éveillée.
- Certaines caractéristiques de l'objet peuvent ne pas être importantes dans le cadre d'une application informatique, e.g. la taille d'un client peut s'avérer non pertinente dans une application de commandes de produits capillaires.
 - Attention à la sélection des attributs pertinents selon le contexte

Méthodes (opérations)

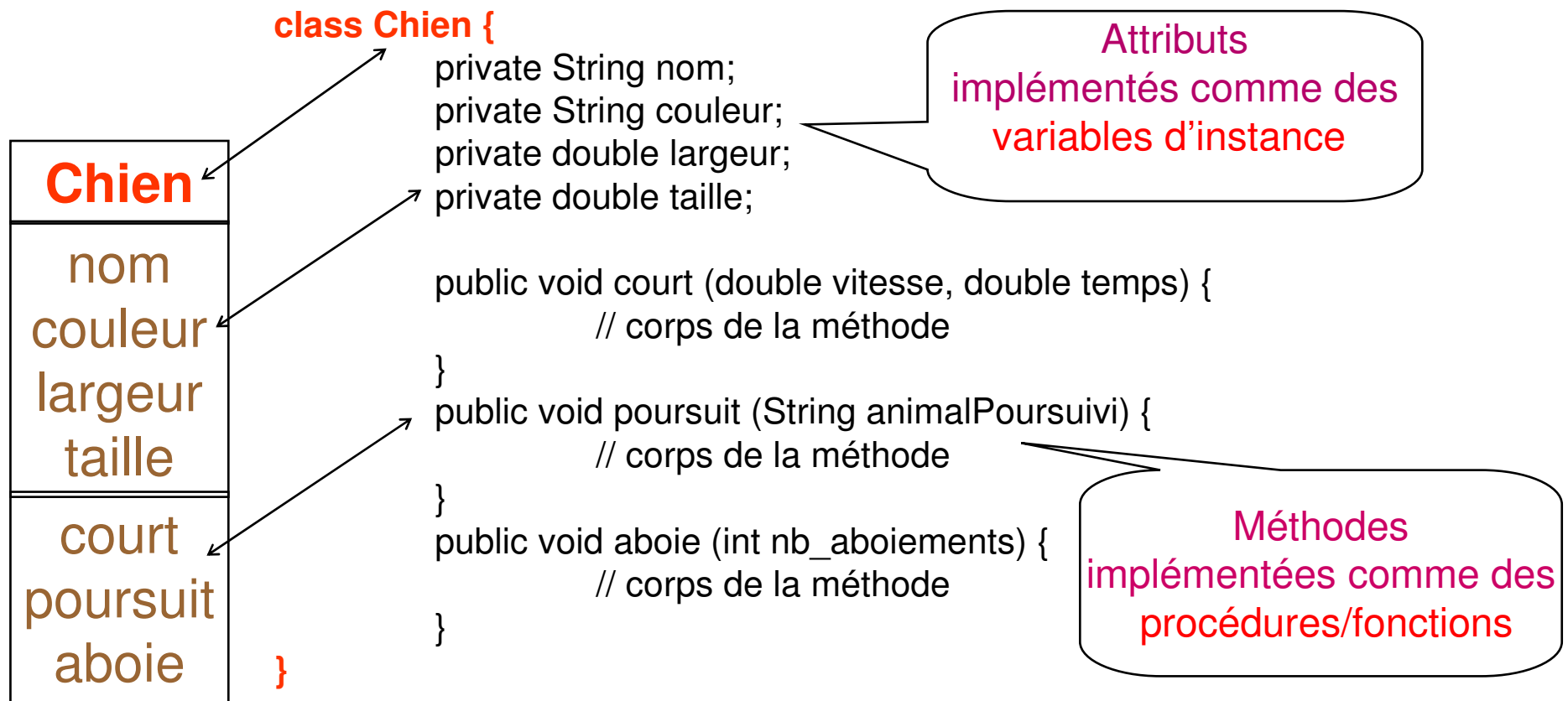
- Décrivent ce qu'un objet peut réaliser (comportement/facultés)
- Moyens de manipuler l'état des objets (mutateurs ou setters)
 - Un objet peut ne pas valider un changement de valeurs proposé
- Moyens de récupérer de l'information à propos des objets (accesseurs ou getters)
 - On dit que les données sont encapsulées au sein des objets

Modéliser une classe par un “diagramme de classe” UML

- Représentation graphique du nom d'une classe, des attributs et des méthodes



Conversion d'un diagramme de classe UML en Java



Constructeurs

- Assurent que l'état initial de l'objet est valide en affectant des valeurs aux variables d'instance

```
class Chien {  
    private String nom;  
    private String couleur;  
    private double largeur;  
    private double taille;  
    // constructeur par défaut pour attribuer des valeurs par défaut  
    Chien() {  
        nom = "anonyme";  
        couleur = "marron";  
        largeur = 50;  
        taille = 50;  
    }  
    // constructeur pour attribuer des valeurs passées en paramètres  
    Chien(String newNom, String newCouleur, double newLargeur,  
        double newTaille) {  
        nom = newNom;  
        couleur = newCouleur;  
        largeur = newLargeur;  
        taille = newTaille;  
    }  
}
```

Instanciación d'objets à partir d'une classe en Java

```
class JoueAvecChiens {  
    public static void main(String argv[ ]) {  
  
        Chien chien321 = new Chien("Rusty", "marron", 45, 40);  
  
        Chien scot9 = new Chien("Scott", "black", 30, 20);  
  
        Chien sprint = new Chien("Sprint", "marronClair", 100, 75);  
  
    }  
}
```

Espace
réservé pour
un objet Chien

Donne à cet espace
un identificateur
unique

Crée un nouvel
objet Chien

Affecte des valeurs aux
variables d'instance

Envoi de messages aux objets instanciés

```
Chien chien321 = new Chien("Rusty", "marron", 45, 40);
```

```
Chien scot9 = new Chien("Scott", "black", 30, 20);
```

```
Chien sprint = new Chien("Sprint", "marronClair", 100, 75);
```

```
chien321.poursuit("Romeo");
```

```
scot9.aboie(2);
```

```
sprint.court(15, 500);
```

message passé
à chien321 - invoque
la méthode poursuit

Identificateur unique

Opérateur pointé

Identificateurs assurent
que la méthode du
'bon' objet est
invoquée. Ici :

- seul *chien321* poursuit
- seul *scot9* aboie
- seul *sprint* court

Encapsulation

- Les méthodes d'un objet opèrent sur celui-ci (et donc sur ses attributs) et non sur un autre.
- Nous dirons que les attributs et les comportements/facultés d'un objet sont **encapsulés**.

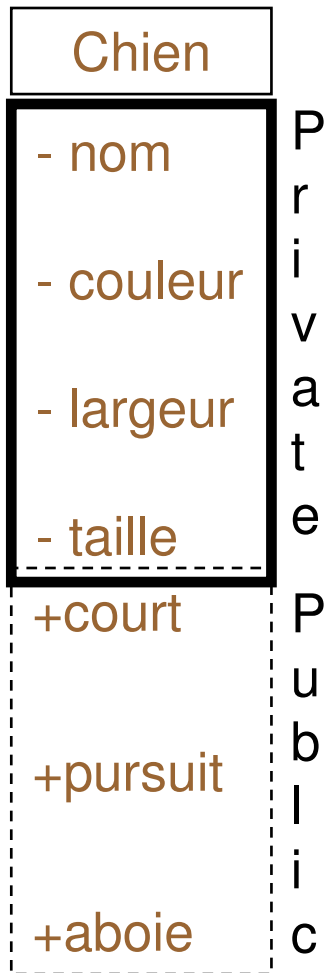
Visibilité (Accessibilité)

- Nous pouvons de manière explicite traduire l'encapsulation en caractérisant la **visibilité** des membres de classe
- On utilise les indicateurs de visibilité tels que **public** (+) et **private** (-)

Chien
- nom
- couleur
- largeur
- taille
+ court
+ poursuit
+ aboie

```
Class Chien {  
    private String nom;  
    private String couleur;  
    private double largeur;  
    private double taille;  
  
    public void court(double vitesse, double temps) {  
        ...  
    }  
    public void poursuit (String animalPoursuivi) {  
        ...  
    }  
    public void aboie (int nb_aboiements) {  
        ...  
    }  
}
```

Visibilité (Accessibilité)



- Membres privés d'un objet sont invisibles / inaccessibles en dehors de l'objet.
 - Ne peuvent donc être manipulés par d'autres objets
- Membres publics sont visibles et accessibles
- Variables d'instance doivent être déclarées privées
- Opérations sont souvent publiques de manière à ce que des messages puissent être passées à l'objet en question.
 - Peuvent être privées lorsque leur usage est strictement interne à l'objet

Accesseurs

- Méthodes publiques permettant l'accès externe aux valeurs des variables d'instance d'un objet
- Ces méthodes peuvent elles accéder directement aux valeurs des variables d'instance

```
String nomChien;  
nomChien = scot9. getNom();
```

```
public String getNom(){  
    return nom;  
}  
public String getCouleur(){  
    return couleur;  
}  
public double getLargeur() {  
    return largeur;  
}  
public double getTaille() {  
    return taille;  
}
```


Mutateurs

- Méthodes publiques permettant la modification externe des valeurs des variables d'instance d'un objet

```
double tailleChien;  
double croissance = 2;  
tailleChien = scot9.getTaille();  
croissance += tailleChien;  
scot9. setTaille(croissance);
```

```
public void setNom(String modNom){  
    nom = modNom;  
}  
public void setCouleur(String modCouleur){  
    couleur = modCouleur;  
}  
public void setLargeur(double modLargeur) {  
    largeur = modLargeur;  
}  
public void setTaille(double modTaille) {  
    taille = modTaille;  
}
```

En résumé: la classe Chien en Java (1/2)

```
class Chien {  
    //début du bloc de classe  
    //variables d'instance  
    private String nom;  
    private String couleur;  
    private double largeur;  
    private double taille;  
    //constructeur par défaut  
    Chien () {  
        nom = "anonyme";  
        couleur = "marron";  
        largeur = 50;  
        taille = 50;  
    }  
    //constructeur avec paramètres  
    Chien (String newNom,  
           String newCouleur,  
           double newLargeur,  
           double newTaille) {  
        nom = newNom;  
        couleur = newCouleur;  
        largeur = newLargeur;  
        taille = newTaille;  
    }  
}
```

```
    //méthodes  
    public void court (double vitesse, double temps) {  
        ...  
    }  
}  
    public void poursuit (String animalPoursuivi) {  
        ...  
    }  
    public void aboie (int nb_aboiements) {  
        ...  
    }  
}
```

En résumé: la classe Chien en Java (2/2)

//mutateurs

```
public void setNom(String modNom){
    nom = modNom;
}
public void setCouleur(String modCouleur){
    couleur = modCouleur;
}
public void setLargeur(double modLargeur) {
    largeur = modLargeur;
}
public void setTaille(double modTaille) {
    taille = modTaille;
}
```

//accesseurs

```
public String getNom(){
    return nom;
}
public String getCouleur(){
    return couleur;
}
public double getLargeur() {
    return largeur;
}
public double getTaille() {
    return taille;
}
} //Fin du bloc de classe
```

Utilisation de la classe Chien (à partir d'une autre classe)

//Variables locales

```
double largeurChien;  
double croissance = 2;
```

//Instantiation d'objets Chien

```
Chien chien321 = new Chien("Rusty",  
    "marron", 45, 40);  
Chien scot9 = new Chien("Scott", "noir", 30,  
    20);  
Chien sprint = new Chien("Sprint",  
    "marronClair", 100, 75);  
Chien bitsa = new Chien();
```

//Utilisation des méthodes

```
chien321.poursuit("Romeo");  
scot9.aboie(2);  
sprint.court(15, 500);  
bitsa.aboie(3);
```

//Utilisation des accesseurs

```
System.out.println("La couleur de Sprint est  
    " + sprint.getCouleur());  
largeurChien = scot9.getLargeur();
```

//Utilisation des mutateurs

```
croissance += largeurChien;  
scot9.setLargeur(croissance);  
bitsa.setNom("Bitsa");
```

Objets & Références

- Type objet : référence vers un emplacement mémoire
- Variable de référence ≠ variable de types primitifs
- Chien chien321, scot9, sprint;
 - Assez d'espace pour la réservation d'un emplacement mémoire
 - faire le lien avec la notion d'adresse en C
 - **null**

<u>Variable</u>	<u>Mémoire</u>	<u>Valeur</u>
chien321	1000	null
	1100	
scot9	1200	null
	1300	
sprint	1400	
	1500	null
	1600	

Objets & Références

- Création d'1 référence pour 1 nouvel objet:

new



- Quantité d'espace requise allouée
- Retourne l'adresse où se trouve l'objet

- Considérons le premier constructeur

- Etat de la mémoire avant son exécution →

<u>Variable</u>	<u>Mémoire</u>	<u>Valeur</u>	<u>Type donnée</u>
chien321	1000	2500	ref
...
scot9	1200	null	ref
...
sprint	1500	null	ref
...
	2500	obj vide	Chien

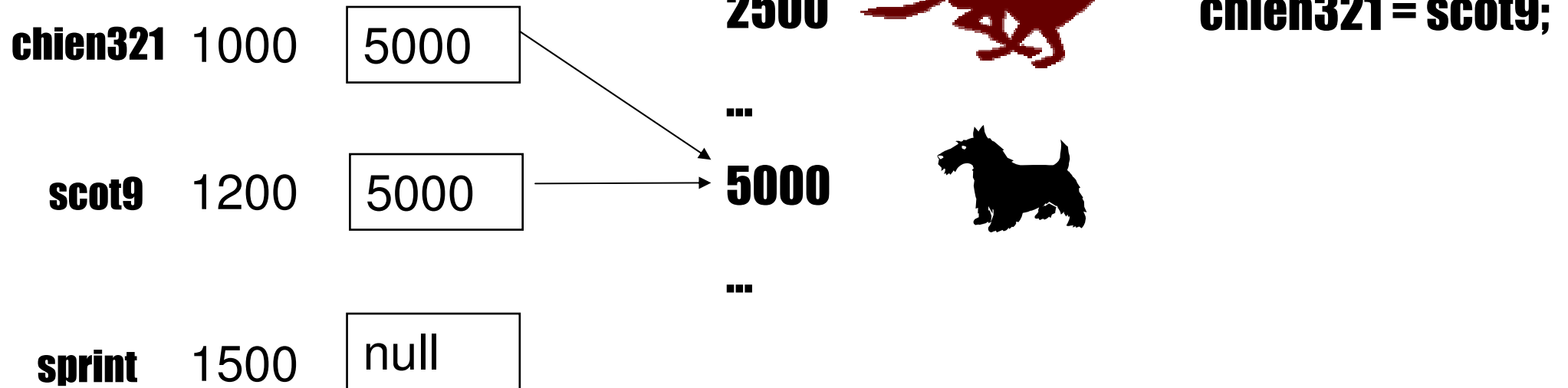
Objets & Références

<u>Variable</u>	<u>Mémoire</u>	<u>Valeur</u>	<u>Type donnée</u>
chien321	1000	2500	ref
...
scot9	1200	5000	ref
...
sprint	1500	null	ref
...
	2500		Chien
	5000		Chien

Objets & Références

Référence
Mémoire Valeur

Mémoire



Objets & Références

Référence
Mémoire Valeur

chien321	1000	5000
scot9	1200	null
sprint	1500	null

Mémoire

2500



...

5000



...

chien321 = scot9;
scot9 = sprint;

Objets & Références

Référence
Mémoire Valeur

chien321	1000	5000
scot9	1200	null
sprint	1500	null

Mémoire

2500



...

5000



...

chien321 = scot9;

scot9 = sprint;

**System.out.println
[scot9 == sprint];**

true

Objets & Références

Référence
Mémoire Valeur

chien321	1000	5000
scot	1200	null
sprint	1500	5000

Mémoire

2500



...

5000



...

chien321 = scot9;

scot9 = sprint;

sprint = chien321;

**System.out.println
[scot9 == sprint];**

false

Opérations sur les références

- Affectation de référence : =
- Comparaison : == et !=
- Cast (type)
 - feu1 = (FeuCirculation)feuStop;
- Opérateur pointé : .
 - feu1.getFeu();

Référence **this**

- Les attributs d'un objet peuvent être différenciés des mêmes attributs d'un autre objet ou variables de même nom

```
class Staff {  
    private String nom; //variable d'instance  
    Staff (String leNom) { //constructeur  
        nom = leNom;  
    }  
}
```

Référence **this**

- Les attributs d'un objet peuvent être différenciés des mêmes attributs d'un autre objet ou variables de même nom

```
class Staff {
```

```
    private String nom; //variable d'instance
```

```
    Staff (String nom) { //constructeur
```

```
        this.nom = nom;
```

}
variable d'instance
de cet objet

paramètre
du constructeur

Références

Affichage d'objets

```
Staff employe = new Staff("Wendy");
```

Variable	Valeur	Type donnée
employe	C40C80	ref
	C40C81	
	

```
System.out.println (employe);
```

Staff@c40c80

Références

Affichage d'objets

```
class Staff {  
    private String nom; //variable d'instance  
  
    ...  
    public String toString () {  
        return ("Nom: " + nom);  
    }  
}
```

```
System.out.println (employee);
```

Nom: Wendy

Application

- On considère un `BienImmobilier` caractérisé par :
 - sa référence unique (entier incrémenté automatiquement)
 - son prix
 - le fait qu'il soit ou non en location
 - le fait qu'il soit ou non en vente
 - le nom de son propriétaire et celui de son locataire (s'il existe)
- Ecrire l'implémentation de la classe avec les getters/setters, la méthode : *`public String toString()`* et les méthodes :
 - *`public boolean estEnLocation()`*
 - *`public boolean estEnVente()`*
 - *`public void location(String nomLocataire)`*
 - *`public void finLocation()`*
 - *`public void vente(double prixPropose, String nomProprietaire)`*
- Mettre en œuvre une classe de test.