Université Abdelmalek Essaâdi
École Normale Supérieure
Tétouan

2023/2024

Chapitre 7

ENRICHIR LES CLASSES

1

Plan

2

- Un nouveau regard sur les références
- Le modificateur static
- Les classes imbriquées
- Les interfaces
- Interfaces de la librairie standard Java
- Les types énumérés

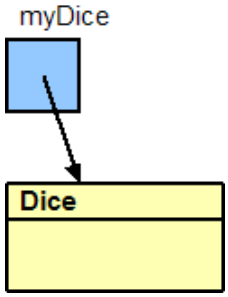
2

Un nouveau regard sur les références (Rappel)

3

- Déclaration d'une variable de référence et création d'un objet Dice

```
Dice myDice = new Dice();
```



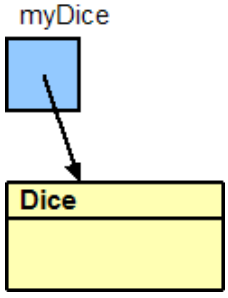
3

Un nouveau regard sur les références (Rappel)

3

- Déclaration d'une variable de référence et création d'un objet Dice

```
Dice myDice = new Dice();
```



- lorsqu'on invoque une méthode de l'objet grâce à l'opérateur point, on utilise l'adresse de l'objet qui est stocké dans la variable de référence pour localiser l'objet en mémoire puis on cherche la bonne méthode et ensuite on l'invoque.

4

La référence null

4

- La variable ne réfère aucun objet, il est dès lors impossible d'appeler une méthode à partir d'une telle référence.

5

La référence null

4

- La variable ne réfère aucun objet, il est dès lors impossible d'appeler une méthode à partir d'une telle référence.

```
String nom;
System.out.println (nom.length());
```

6

La référence null

4

- La variable ne réfère aucun objet, il est dès lors impossible d'appeler une méthode à partir d'une telle référence.

```
String nom;
System.out.println (nom.length());
```

7

La référence null

4

- La variable ne réfère aucun objet, il est dès lors impossible d'appeler une méthode à partir d'une telle référence.

```
String nom;
System.out.println (nom.length());
```

- Il existe en Java le mot réservé ***null*** que l'on peut affecter à une variable ou que l'on peut utiliser lors de comparaison

8

La référence null

4

- La variable ne réfère aucun objet, il est dès lors impossible d'appeler une méthode à partir d'une telle référence.

```
String nom;
System.out.println (nom.length());
```

- Il existe en Java le mot réservé ***null*** que l'on peut affecter à une variable ou que l'on peut utiliser lors de comparaison

```
String nom = null;
 if (nom != null)
{
   System.out.println (nom.length());
}
```

9

Alias

5

- Les alias sont une conséquence des deux types de donnée de Java : les types primitifs et les types objet. En effet, l'utilisation de l'opérateur d'affectation peut être vue de deux manières différentes selon le type de donnée.

10

Affectation pour les types primitifs

6

```
int nb1 = 2;
int nb2 = 9;

nb2 = nb1;
```

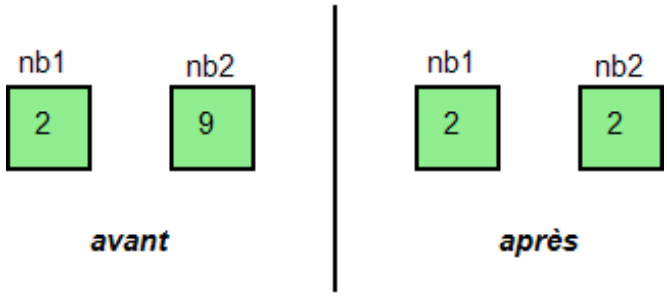
11

Affectation pour les types primitifs

6

```
int nb1 = 2;
int nb2 = 9;

nb2 = nb1;
```



12

Affectation pour les types objet

7

```
Dice dice1 = new Dice();
Dice dice2 = new Dice();

System.out.println (dice1.getFace());
System.out.println (dice2.getFace());

dice2 = dice1;

System.out.println (dice1.getFace());
System.out.println (dice2.getFace());
```

13

Affectation pour les types objet

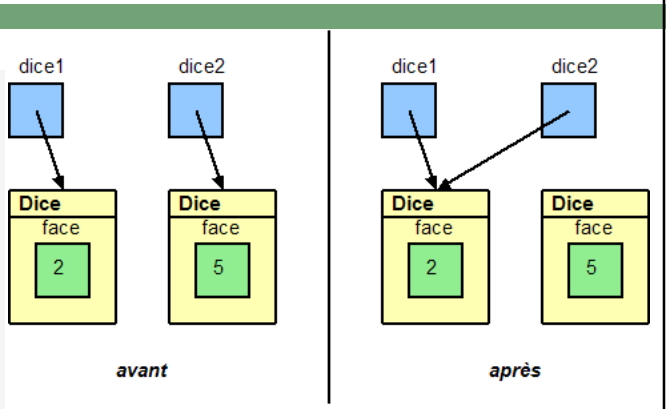
7

```
Dice dice1 = new Dice();
Dice dice2 = new Dice();

System.out.println (dice1.getFace());
System.out.println (dice2.getFace());

dice2 = dice1;

System.out.println (dice1.getFace());
System.out.println (dice2.getFace());
```



14

Affectation pour les types objet

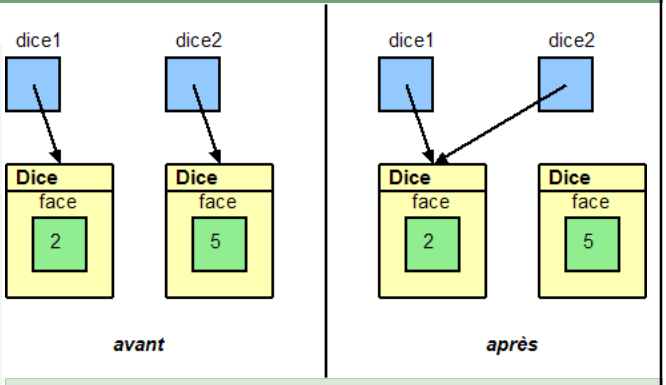
7

```
Dice dice1 = new Dice();
Dice dice2 = new Dice();

System.out.println (dice1.getFace());
System.out.println (dice2.getFace());

dice2 = dice1;

System.out.println (dice1.getFace());
System.out.println (dice2.getFace());
```



on dit que la variable dice2 est un alias de la variable dice1, c'est-à-dire qu'il s'agit d'une seconde manière pour accéder à un même objet.

15

Affectation pour les types objet

8

- On a donc bien un seul objet mais deux variables qui le réfère, ainsi si on change l'état de l'objet via une variable, et qu'ensuite on accède à l'état via une autre variable, l'état aura bien entendu changé.

16

Affectation pour les types objet

8

- On a donc bien un seul objet mais deux variables qui le réfère, ainsi si on change l'état de l'objet via une variable, et qu'ensuite on accède à l'état via une autre variable, l'état aura bien entendu changé.

```
Dice dice1 = new Dice();
Dice dice2 = new Dice();
dice2 = dice1;
dice1.flip(); // On change l'état du premier dé via la première variable
System.out.println (dice2.getFace()) // On consulte l'état du premier dé via la seconde variable
```

17

Comparaison d'objets

9

- Opérateur ==

18

Comparaison d'objets

9

- Opérateur ==

```
Dice dice1 = new Dice();
Dice dice2 = new Dice();
System.out.println (dice1 == dice2); // Affiche false
 dice2 = dice1;
System.out.println (dice1 == dice2); // Affiche true
```

19

Comparaison d'objets

9

- Opérateur ==

```
Dice dice1 = new Dice();
Dice dice2 = new Dice();
System.out.println (dice1 == dice2); // Affiche false
 dice2 = dice1;
System.out.println (dice1 == dice2); // Affiche true
```

- ❿Méthode equals

20

Comparaison d'objets

9

- Opérateur ==

```
Dice dice1 = new Dice();
Dice dice2 = new Dice();
System.out.println (dice1 == dice2); // Affiche false
 dice2 = dice1;
System.out.println (dice1 == dice2); // Affiche true
```

❿Méthode equals

```
Dice dice1 = new Dice();
Dice dice2 = new Dice();
System.out.println (dice1.equals (dice2));// Affiche true si les dés ont la même face
dice2 = dice1;
System.out.println (dice1.equals (dice2));// Affiche true à tous les coups, pourquoi ?
```

21

la méhode equals de la classe Dice

10

```
public boolean equals (Object obj)
{
   Dice d = (Dice) obj;
   return obj.getFace() == face;
}
```

22

Garbage Collector

11

- La machine virtuelle Java contient un mécanisme qui va régulièrement nettoyer la mémoire de tous les déchets, il s'agit du **garbage collector**

23

Garbage Collector

11

- La machine virtuelle Java contient un mécanisme qui va régulièrement nettoyer la mémoire de tous les déchets, il s'agit du **garbage collector**

Exemple: Gestion de la mémoire avec la classe Runtime

24

Garbage Collector

11

- La machine virtuelle Java contient un mécanisme qui va régulièrement nettoyer la mémoire de tous les déchets, il s'agit du **garbage collector**

Exemple: Gestion de la mémoire avec la classe Runtime

```
Runtime r = Runtime.getRuntime(); // Créer un objet de type Runtime
String m = "Maman";
String p = "Papa";
System.out.println ("Max : " + r.maxMemory());
System.out.println ("Free : " + r.freeMemory());
m = null;  p = null;
r.gc(); // Appel implicite au garbage collector
System.out.println ("Free : " + r.freeMemory());
```

25

Garbage Collector

11

- La machine virtuelle Java contient un mécanisme qui va régulièrement nettoyer la mémoire de tous les déchets, il s'agit du **garbage collector**

Exemple: Gestion de la mémoire avec la classe Runtime

```
Runtime r = Runtime.getRuntime(); // Créer un objet de type Runtime
String m = "Maman";
String p = "Papa";
System.out.println ("Max : " + r.maxMemory());
System.out.println ("Free : " + r.freeMemory());
m = null;  p = null;
r.gc(); // Appel implicite au garbage collector
System.out.println ("Free : " + r.freeMemory());
```

```
Max : 66650112
Free : 1871648
Free : 1914656
```

26

Passage d'objets en paramètre

12

- Lorsqu'on fait appel à une méthode et que l'on passe des valeurs en paramètre, Java passe une copie des valeurs des variables à la méthode.

27

Passage d'objets en paramètre

12

- Lorsqu'on fait appel à une méthode et que l'on passe des valeurs en paramètre, Java passe une copie des valeurs des variables à la méthode.

➢C'est-à-dire que la valeur du paramètre réel utilisé lors de l'appel est copié dans la valeur du paramètre formel de la méthode.

28

Passage d'objets en paramètre

12

- Lorsqu'on fait appel à une méthode et que l'on passe des valeurs en paramètre, Java passe une copie des valeurs des variables à la méthode.

➢C'est-à-dire que la valeur du paramètre réel utilisé lors de l'appel est copié dans la valeur du paramètre formel de la méthode.

➢Si la méthode modifie la valeur de la variable, cela n'affectera en rien le paramètre réel

29

Passage de paramètre par valeur

13

- **Exemple**: Passage de paramètre par valeur

```
int i = 56;
System.out.println ("Avant : " + i);
change (i);
System.out.println ("Après : " + i);
Point p = new Point (13, 87);
System.out.println ("Avant : " + p);
change (p);
System.out.println ("Après : " + p);
```

30

Passage de paramètre par valeur

13

- **Exemple**: Passage de paramètre par valeur

```
int i = 56;
System.out.println ("Avant : " + i);
change (i);
System.out.println ("Après : " + i);
Point p = new Point (13, 87);
System.out.println ("Avant : " + p);
change (p);
System.out.println ("Après : " + p);
```

```
public void change (int i) {
   i = 987;
}
 public void change (Point p) {
   p = new Point (4, 8);
}
```

31

Passage d'objets en paramètre

14

- **Exemple**: La méthode translate de la classe Point effectue une translation de la coordonnée de paramètre par valeur

```
public void change (Point p)
{
   p.translate (5, 5);
}
```

```
Point p = new Point (13, 87);
System.out.println ("Avant : " + p);
change (p);
System.out.println ("Après : " + p);
```

32

La référence this

15

- On utilise cette référence dans une classe pour représenter l'instance courante

```
public class MyNumber
{
   private int nb;

   public MyNumber (int nb)
   {
      this.nb = nb;
   }
}
```

33

Le modificateur static

16

- Ce modificateur permet d'associer une variable ou une méthode à la classe plutôt qu'à une instance de la classe

34

Les variables statiques

17

```
public class Student
{
   // Variable d'instance représentant le nom de l'étudiant
   private String name;

   // Construteur
   public Student (String name)   {
      this.name = name;
   }
   // Méthode pour récupérer le nom de l'étudiant
   public String getName()  {
      return name;
   }
}
```

35

Les variables statiques

17

```
Student stud1 = new Student ("John");
Student stud2 = new Student ("Jenny");
```

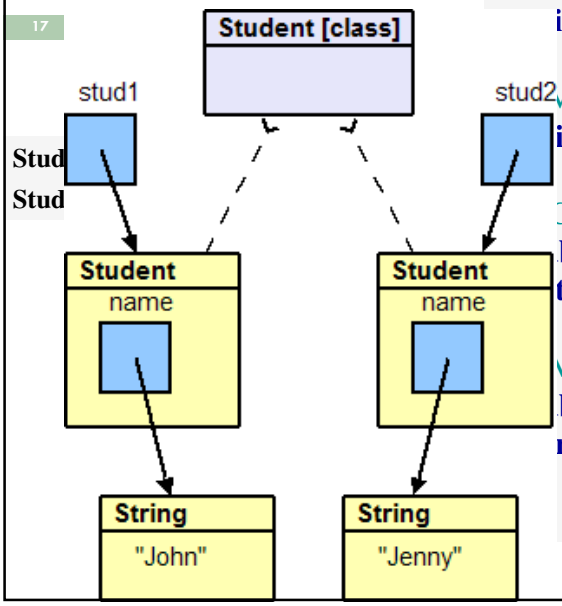
```
public class Student
{
   // Variable d'instance représentant le nom de l'étudiant
   private String name;

   // Construteur
   public Student (String name)   {
      this.name = name;
   }
   // Méthode pour récupérer le nom de l'étudiant
   public String getName()  {
      return name;
   }
}
```

36

Les variables statiques

17




```
ic class Student

Variable d'instance représentant le nom de l'étudiant
ivate String name;

Constructeur
blic Student (String name)   {
this.name = name;

Méthode pour récupérer le nom de l'étudiant
blic String getName()  {
return name;
```

37

Les variables statiques

18

- Les ***variables statiques***, aussi appelées ***variables de classe***, n'appartiennent pas à une instance particulière, elles apartiennent à la classe.
- On les déclare avec le mot réservé **static**.
- Une seule copie de ces variables existe donc et si une instance modifie la valeur de cette variable, la modification est visible pour tout le monde.

38

Les variables statiques

19

Exemple: compter le nombre d'instance d'une classe

```
public class InstanceCounter {
   public static int nb = 0;
    public InstanceCounter()    {
      nb++;
   }}
```

39

Les variables statiques

19

Exemple: compter le nombre d'instance d'une classe

```
public class InstanceCounter {
   public static int nb = 0;
    public InstanceCounter()    {
      nb++;
   }}
```

```
new InstanceCounter();
new InstanceCounter();
new InstanceCounter();
 System.out.println ("Il y a " + InstanceCounter.nb + " instances de la classe InstanceCounter");
```

40

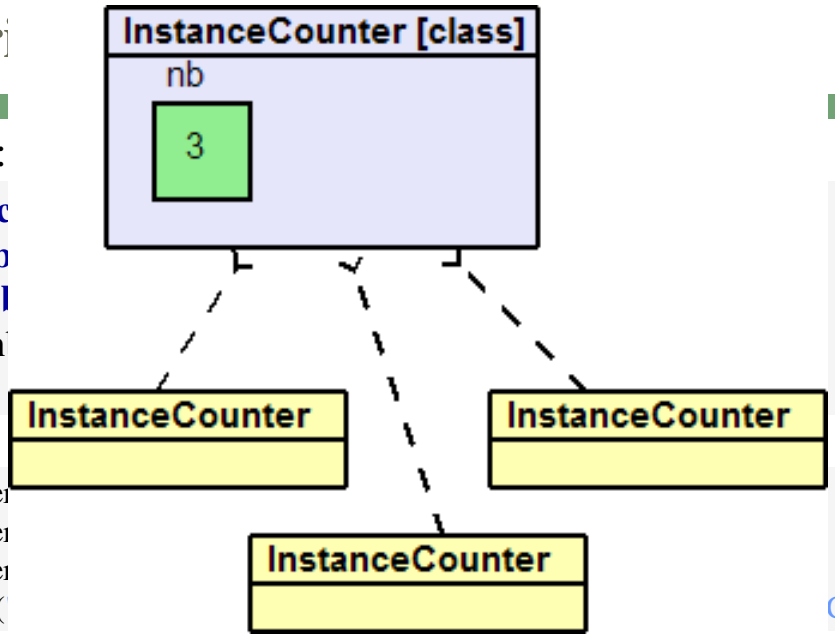
Les vari

19

Exemple:

```
public
  pub
   pul
     n
  }}
```

```
new InstanceCounter
new InstanceCounter
new InstanceCounter
 System.out.println (
```




```
Counter");
```

41

Les méthodes statiques

20

- On peut bien également déclarer une méthode comme étant statique, on a ainsi des **méthodes statiques** ou **méthodes de classe**.
- La méthode abs de la classe *Math* qui permet de récupérer la valeur absolue d'un nombre.

42

Les méthodes statiques

20

- On peut bien également déclarer une méthode comme étant statique, on a ainsi des **méthodes statiques** ou **méthodes de classe**.
- La méthode abs de la classe *Math* qui permet de récupérer la valeur absolue d'un nombre.

```
int nb = -645;
System.out.println ("La valeur absolue de " + nb + " est " + Math.abs (nb));
```

43

Les méthodes statiques

20

- On peut bien également déclarer une méthode comme étant statique, on a ainsi des **méthodes statiques** ou **méthodes de classe**.
- La méthode abs de la classe *Math* qui permet de récupérer la valeur absolue d'un nombre.

```
int nb = -645;
System.out.println ("La valeur absolue de " + nb + " est " + Math.abs (nb));
```

- elles ne peuvent en aucun cas accéder aux variables d'instances qui appartiennent aux instances de la classe

44

Les méthodes statiques

20

- On peut bien également déclarer une méthode comme étant statique, on a ainsi des **méthodes statiques** ou **méthodes de classe**.
- La méthode abs de la classe *Math* qui permet de récupérer la valeur absolue d'un nombre.

```
int nb = -645;
System.out.println ("La valeur absolue de " + nb + " est " + Math.abs (nb));
```

- elles ne peuvent en aucun cas accéder aux variables d'instances qui appartiennent aux instances de la classe

Cannot make a static reference to the non-static field ...

45

Les classes imbriquées

21

- On peut également déclarer des classes.
- Ces classes sont également considérées comme des membres de la classe dans laquelle elles sont déclarées, on les appelle *classes imbriquées*.

46

Les classes imbriquées

21

- On peut égalemer
- Ces classes sont é lasse dans laquelle elle

```
public class Car{
  private Wheel wheel1, wheel2, wheel3, whee4;

  public Car()    {
     wheel1 = new Wheel();
     wheel2 = new Wheel();
     wheel3 = new Wheel();
     wheel4 = new Wheel();
  }
   private static class Wheel    {
     private int pressure;
      public Wheel()      {
        pressure = 2;
     }}}
```

47

Les interfaces

22

- l'ensemble des méthodes publiques à travers lesquelles on peut interagir avec un objet.

48

Les interfaces

22

- l'ensemble des méthodes publiques à travers lesquelles on peut interagir avec un objet.

Exemple: une interface télécommande de télévision.

- Cette interface défini diverses méthodes publiques qui sont par exemple augmenter ou diminuer le son, monter ou descendre de chaine.

49

Les interfaces

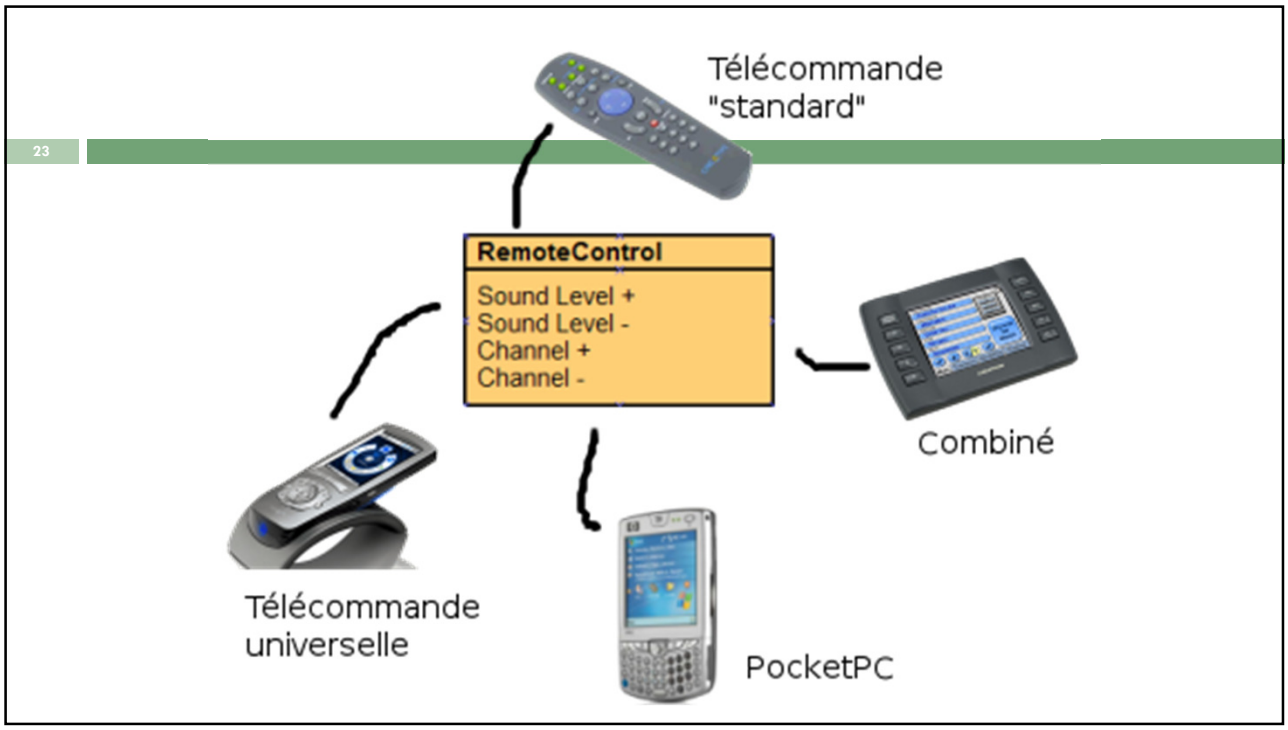
22

- l'ensemble des méthodes publiques à travers lesquelles on peut interagir avec un objet.

Exemple: une interface télécommande de télévision.

- Cette interface défini diverses méthodes publiques qui sont par exemple augmenter ou diminuer le son, monter ou descendre de chaine.
- On dit qu'une classe implémente une interface lorsqu'elle offre toutes les méthodes publiques définies dans l'interface, c'est-à-dire qu'elle implémente toutes les méthodes de l'interface.

50



51

Les interfaces: **En Java**

24

- On définit une interface dans un fichier séparé et on la déclare avec le mot réservé *interface*
- Une interface contient des déclarations de constantes et de méthodes abstraites.
- Une méthode abstraite est une méthode sans corps (sans implémentation), il y a donc juste l'entête de la méthode se terminant par un point-virgule.

52

Les interfaces: **En Java**

24

- On définit une interface dans un fichier séparé et on la déclare avec le mot réservé *interface*
- Une interface contient des déclarations de constantes et de méthodes abstraites.
- Une méthode abstraite est une méthode sans corps (sans implémentation), il y a donc juste l'entête de la méthode se terminant par un point-virgule.

```
public interface AnimalIF {
   public void getNoise();
   public void getFamily();
}
```

53

Les interfaces: **En Java**

25

- Une classe implémente une interface, celà veut dire que la classe fournit une implémentation pour toutes les méthodes abstraites de l'interface.
- Pour signaler qu'une classe implémente une certaine interface, on utilise le mot réservé *implements*.

54

Les interfaces: **En Java**

25

- Une classe implémente une interface, fournit une implémentation pour toutes l'interface.
- Pour signaler qu'une classe implément utilise le mot réservé *implements*.

```
public class Dog implements AnimalIF{
   private String breed;
    public Dog (String breed)    {
      this.breed = breed;
   }
    public void getNoise()    {
      System.out.println ("Wouf");
   }
    public void getFamily()    {
      System.out.println ("I'm a mammal");
   }
    public void getBreed()    {
      System.out.println ("I'm a " + race);
   }
}
```

55

Les interfaces: **En Java**

25

- Une classe implémente une interface, fournit une implémentation pour toutes l'interface.
- Pour signaler qu'une classe implément

```
public class Fly implements AnimalIF{
   public void getNoise()    {
      System.out.println ("Bzzz");
   }
    public void getFamily()    {
      System.out.println ("I'm an insect");
   }
}
```

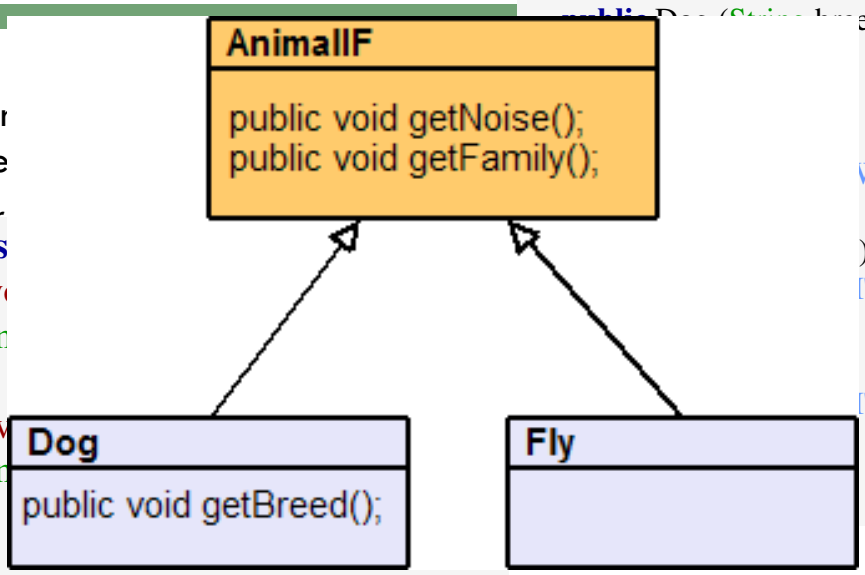
```
public class Dog implements AnimalIF{
   private String breed;
    public Dog (String breed)    {
      this.breed = breed;
   }
    public void getNoise()    {
      System.out.println ("Wouf");
   }
    public void getFamily()    {
      System.out.println ("I'm a mammal");
   }
    public void getBreed()    {
      System.out.println ("I'm a " + race);
   }
}
```

56

Les interfaces: **En Java**

25

- Une
  fourr
  l'inte




```
public clas
   public v
      Systen
   }
   public v
      Systen
   }
}
```

```
public class Dog implements AnimalIF{
   private String breed;
                                ed)   {


                                   {
                                Wouf");

                                )   {
                                'm a mammal");

                                   {
                                'm a " + race);
```

57

Implémentation multiple

26

- Une classe peut implémenter plusieurs interfaces, dans ce cas, on les énumère toutes, après le mot réservé *implements*, séparées par des virgules.
- La classe devra donc implémenter toutes les méthodes définies dans toutes les interfaces qu'elle implémente.

58

Implémentation multiple

26

- Une classe peut implémenter plusieurs interfaces, dans ce cas, on les énumère toutes, après le mot réservé *implements*, séparées par des virgules.
- La classe devra donc implémenter toutes les méthodes définies dans toutes les interfaces qu'elle implémente.
- Si les interfaces ont des méthodes en commun, c'est-à-dire avec la même signature, mais avec une sémantique différente,

59

Implémentation multiple

26

- Une classe peut implémenter plusieurs interfaces, dans ce cas, on les énumère toutes, après le mot réservé *implements*, séparées par des virgules.
- La classe devra donc implémenter toutes les méthodes définies dans toutes les interfaces qu'elle implémente.
- Si les interfaces ont des méthodes en commun, c'est-à-dire avec la même signature, mais avec une sémantique différente,
  - la classe qui implémente toutes les interfaces ne définira qu'une seule fois la méthode.
  - Si ces méthodes ont une sémantique différente, une des deux sera masquée par l'autre.

60

Référence directe ou indirecte

27

- Référence directe :
  - La variable contient une *référence directe.*
  - on peut directement savoir quel genre d'objets est référencé par la variable en regardant sa déclaration.

```
Dog myDog;
myDog = new Dog ("Husky");
```

- Référence indirecte :
  - On peut donner comme type de variable une interface.
  - La variable pourra contenir une référence vers une instance de n'importe quelle classe qui implémente l'interface.
  - On a ainsi une *référence indirecte*.

```
AnimalIF myDog;
myDog = new Dog ("Husky");
```

61

Polymorphisme

28

- Grâce aux interfaces et aux références indirectes, on peut utiliser le *polymorphisme*
- Le *polymorphisme* traite de variables qui pourront prendre plusieurs formes.

62

Polymorphisme

28

- Grâce aux interfaces et aux références indirectes, on peut utiliser le *polymorphisme*
- Le *polymorphisme* traite de variables qui pourront prendre plusieurs formes.

```
AnimalIF myAnimal;
myAnimal = new Dog ("Husky");
myAnimal.getNoise();
((Dog) myAnimal).getBreed();
```

```
Wouf
I'm a Husky
```

63

Polymorphisme

28

- Grâce aux interfaces et aux références indirectes, on peut utiliser le *polymorphisme*
- Le *polymorphisme* traite de variables qui pourront prendre plusieurs formes.

```
AnimalIF myAnimal;
myAnimal = new Dog ("Husky");
myAnimal.getNoise();
((Dog) myAnimal).getBreed();
```

```
Wouf
I'm a Husky
```

- Une référence polymorphique est une référence qui peut se référer à plusieurs types d'objets à des moments différents.
- La variable myAnimal est soit un animal, soit un chien.

64