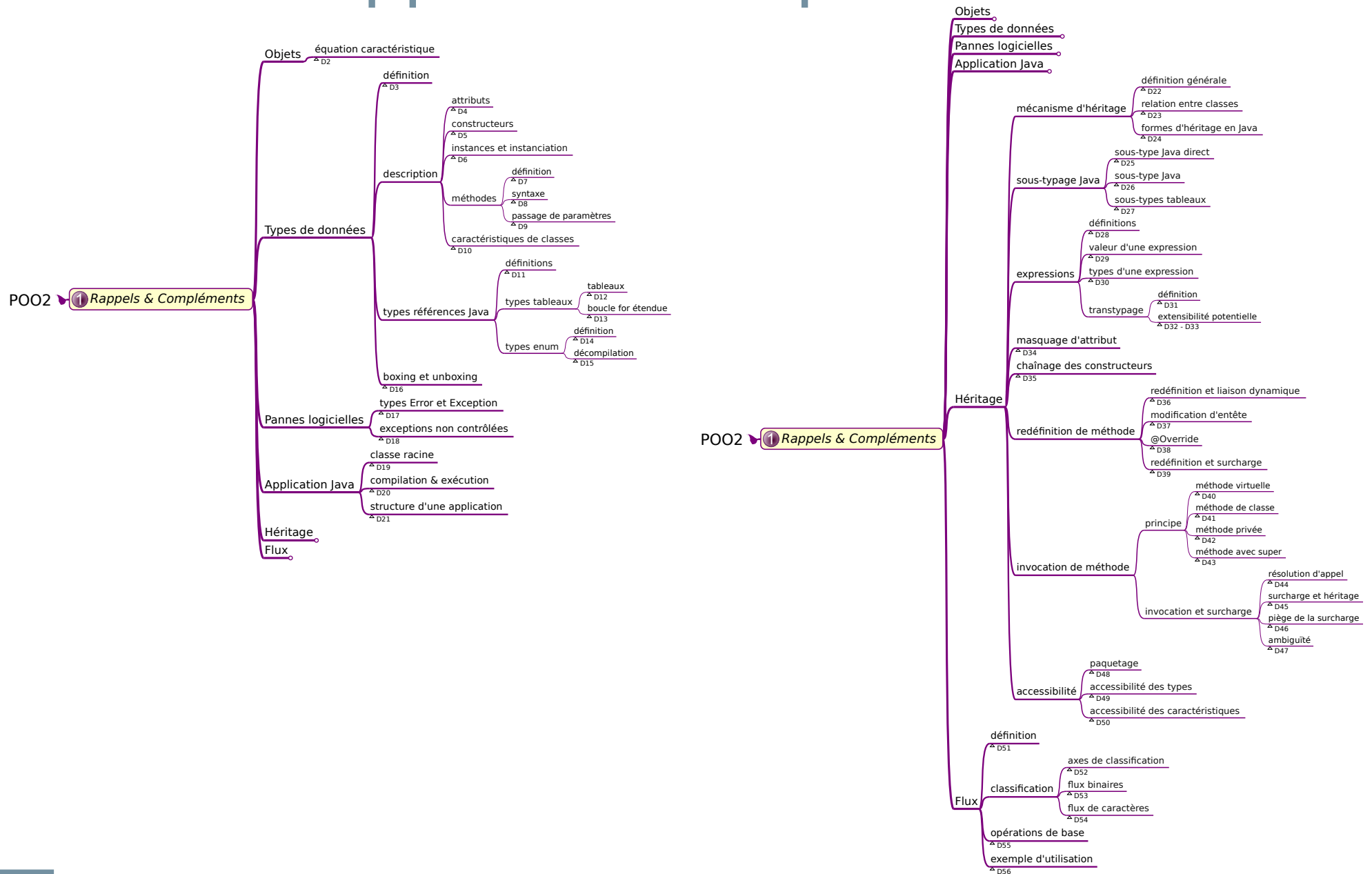


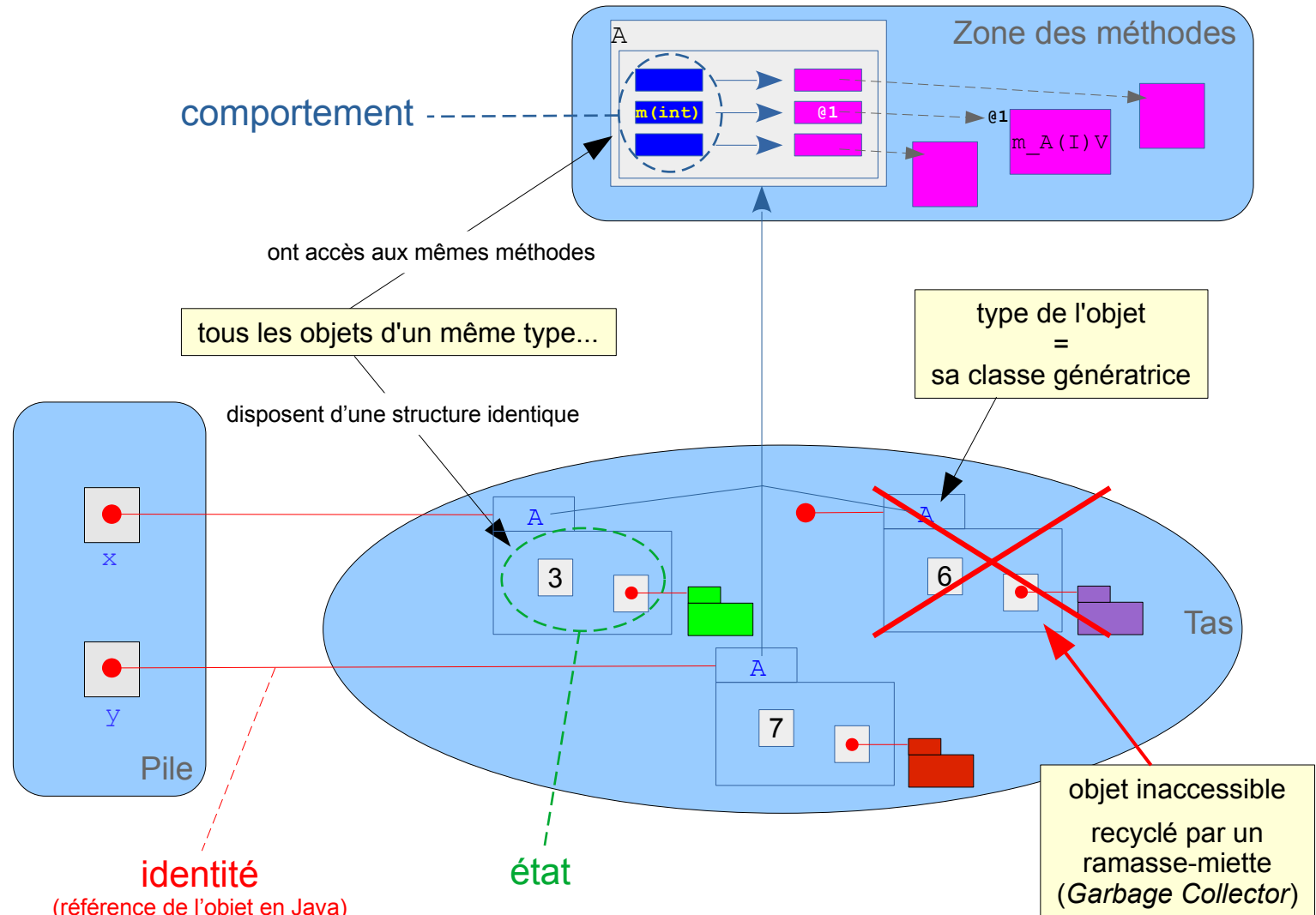
# Rappels et Compléments



## Rappels et compléments

- I. Objets
  1. Équation caractéristique
- II. Types de données
  1. Définition
  2. Description
    - i. Attributs
    - ii. Constructeurs
    - iii. Instance et instanciation
    - iv. Méthodes
      - a. Définition
      - b. Syntaxe
      - c. Passage de paramètres
    - v. Caractéristiques de classes
3. Types références Java
  - i. Définitions
  - ii. Types tableaux
    - a. Tableaux
    - b. Boucle for étendue
  - iii. Types enum
    - a. Définition
    - b. Décompilation
4. Boxing et unboxing
- III. Pannes logicielles
  1. Types Error et Exception
  2. Exceptions non contrôlées
- IV. Application Java
  1. Classe racine
  2. Compilation & Exécution
  3. Structure d'une application

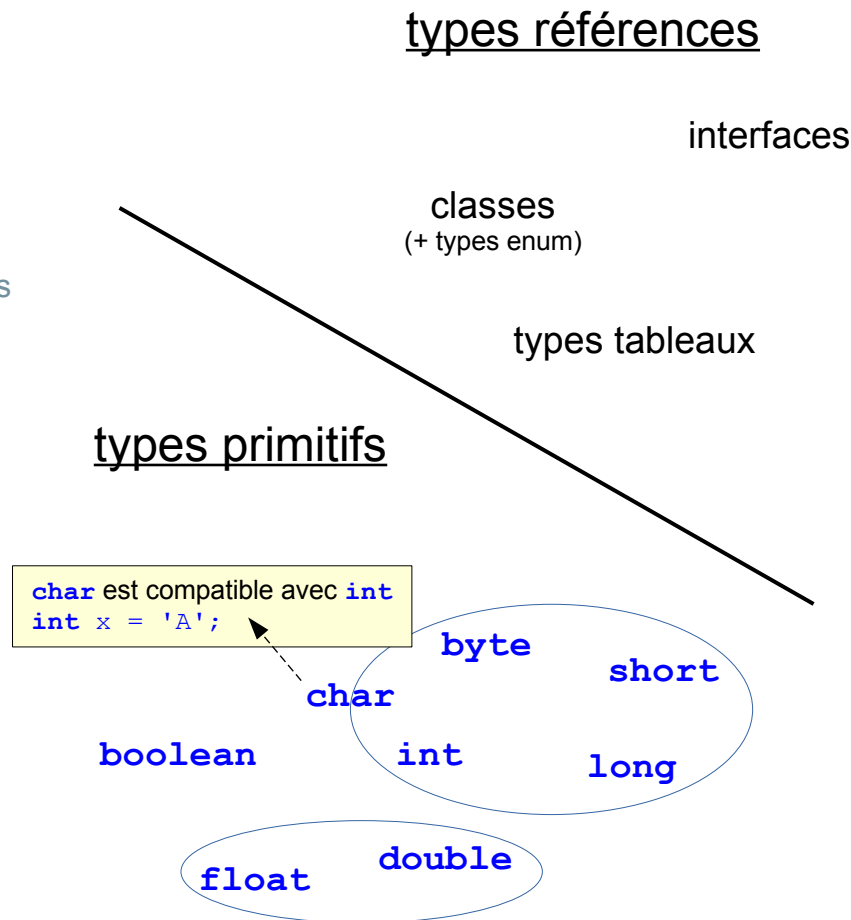
**Équation caractéristique des objets :**  
**objet = identité + état + comportement**



## Rappels et compléments

- I. Objets
1. Équation caractéristique
- II. Types de données
  1. Définition
  2. Description
    - i. Attributs
    - ii. Constructeurs
    - iii. Instance et instantiation
    - iv. Méthodes
      - a. Définition
      - b. Syntaxe
      - c. Passage de paramètres
    - v. Caractéristiques de classes
  3. Types références Java
    - i. Définitions
    - ii. Types tableaux
      - a. Tableaux
      - b. Boucle for étendue
    - iii. Types enum
      - a. Définition
      - b. Décompilation
  4. Boxing et unboxing
- III. Pannes logicielles
  1. Types Error et Exception
  2. Exceptions non contrôlées
- IV. Application Java
  1. Classe racine
  2. Compilation & Exécution
  3. Structure d'une application

**Type de données** : ensemble fini de valeurs muni d'opérations applicables sur ces valeurs.



true

x : boolean

**boolean**

valeurs : true, false  
opérations : &&, ||, !, ...

**Dice**

y : Dice

valeurs :  
opérations :

```
/**
 * @inv
 *   nbOfSides() >= MIN_SIDES_NB
 *   nbOfSides() >= value() >= 1
 */
public class Dice {
    // ATTRIBUTS
    ...
    // CONSTRUCTEURS
    ...
    // REQUETES
    public int nbOfSides() { ... }
    public boolean isTaken() { ... }
    /** @pre !isTaken() */
    public int value() { ... }
    // COMMANDES
    /**
     * @pre isTaken()
     * @post
     *   !isTaken()
     *   nbOfSides() == old nbOfSides()
     *   value() a changé de valeur aléatoirement
     */
    public void roll() { ... }
    /**
     * @pre !isTaken()
     * @post
     *   nbOfSides() == old nbOfSides()
     *   isTaken()
     */
    public void take() { ... }
}
```

## Rappels et compléments

- I. Objets
- 1. Équation caractéristique
- II. Types de données
  - 1. Définition
  - 2. Description
    - i. Attributs
    - ii. Constructeurs
    - iii. Instance et instanciation
    - iv. Méthodes
      - a. Définition
      - b. Syntaxe
      - c. Passage de paramètres
    - v. Caractéristiques de classes
- 3. Types références Java
  - i. Définitions
  - ii. Types tableaux
    - a. Tableaux
    - b. Boucle for étendue
  - iii. Types enum
    - a. Définition
    - b. Décompilation
- 4. Boxing et unboxing
- III. Pannes logicielles
  - 1. Types Error et Exception
  - 2. Exceptions non contrôlées
- IV. Application Java
  - 1. Classe racine
  - 2. Compilation & Exécution
  - 3. Structure d'une application

**Attribut (d'instance)** : variable contenant une partie de l'état d'un objet.

```
class A {  
    private final int i;  
    A() {  
        i = 1;  
    }  
    void m() {  
        i = 2;  
    }  
}
```

### Déclaration d'attribut d'instance

[Access] [Modif] Type IdAttr;

Access →

public  
protected  
private

Modif →

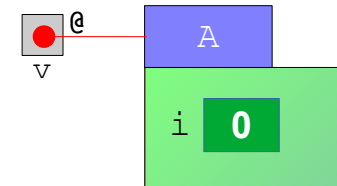
final

private int i;

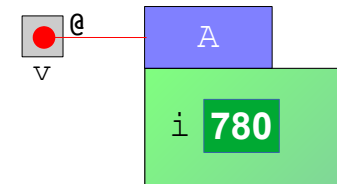
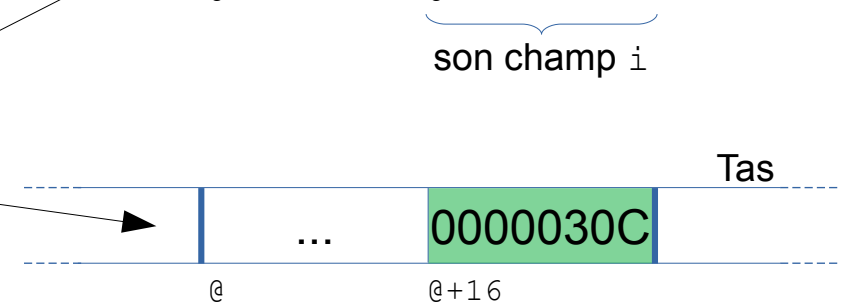
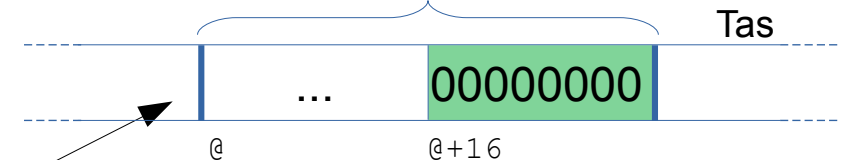
l'attribut i

```
class A {  
    private int i;  
    void set(int k) {  
        this.i = k;  
    }  
}
```

```
void test() {  
    A v = new A();  
    v.set(780);  
}
```



l'« objet v »



## Rappels et compléments

- I. Objets
  1. Équation caractéristique
- II. Types de données
  1. Définition
  2. Description
    - i. Attributs
    - ii. Constructeurs
    - iii. Instance et instanciation
    - iv. Méthodes
      - a. Définition
      - b. Syntaxe
      - c. Passage de paramètres
    - v. Caractéristiques de classes
3. Types références Java
  - i. Définitions
  - ii. Types tableaux
    - a. Tableaux
    - b. Boucle for étendue
  - iii. Types enum
    - a. Définition
    - b. Décompilation
4. Boxing et unboxing
- III. Pannes logicielles
  1. Types Error et Exception
  2. Exceptions non contrôlées
- IV. Application Java
  1. Classe racine
  2. Compilation & Exécution
  3. Structure d'une application

**Constructeur** : procédure d'initialisation des objets.

### Déclaration de constructeur

**[Access]** **IdClasse**(**[Args]**) **[ClauseExc]** **Corps**

**Access** →

**public**  
**protected**  
**private**

**Args** →

**Type**<sub>1</sub> **arg**<sub>1</sub>, ..., **Type**<sub>n</sub> **arg**<sub>n</sub>

**ClauseExc** →

**throws** **Exc**<sub>1</sub>, ..., **Exc**<sub>n</sub>

**public** **A** () { ... }

~~**public class A {**  
~~**private int i = 10;**~~  
~~**}**~~~~

~~**public class A {**  
~~**private int i;**~~  
~~**{ i = 10; }**~~  
~~**}**~~~~

bloc d'initialisation

**public class A {**  
    **private int i;**  
    **public A() {**  
        **i = 10;**  
    **}**  
**}**

**public class A {**  
    **private int i;**  
**}**

**public class A {**  
    **private int i;**  
    **public A() {**  
        **super();**  
    **}**  
**}**

pièges...

~~**class A extends B {**  
~~**A() { this(1); }**~~  
~~**A(int k) { this(); }**~~  
~~**A(String s) {**~~  
~~**this(s.substring(1));**~~  
~~**}**~~  
~~**A(int a, int b) {**~~  
~~**int k = a + b;**~~  
~~**this(k);**~~  
~~**}**~~  
  
    **A() {**  
        // pas d'appel explicite  
        // à this() ni à super()  
    **}**  
**}**~~

pas de cycle !

pas de récursivité !

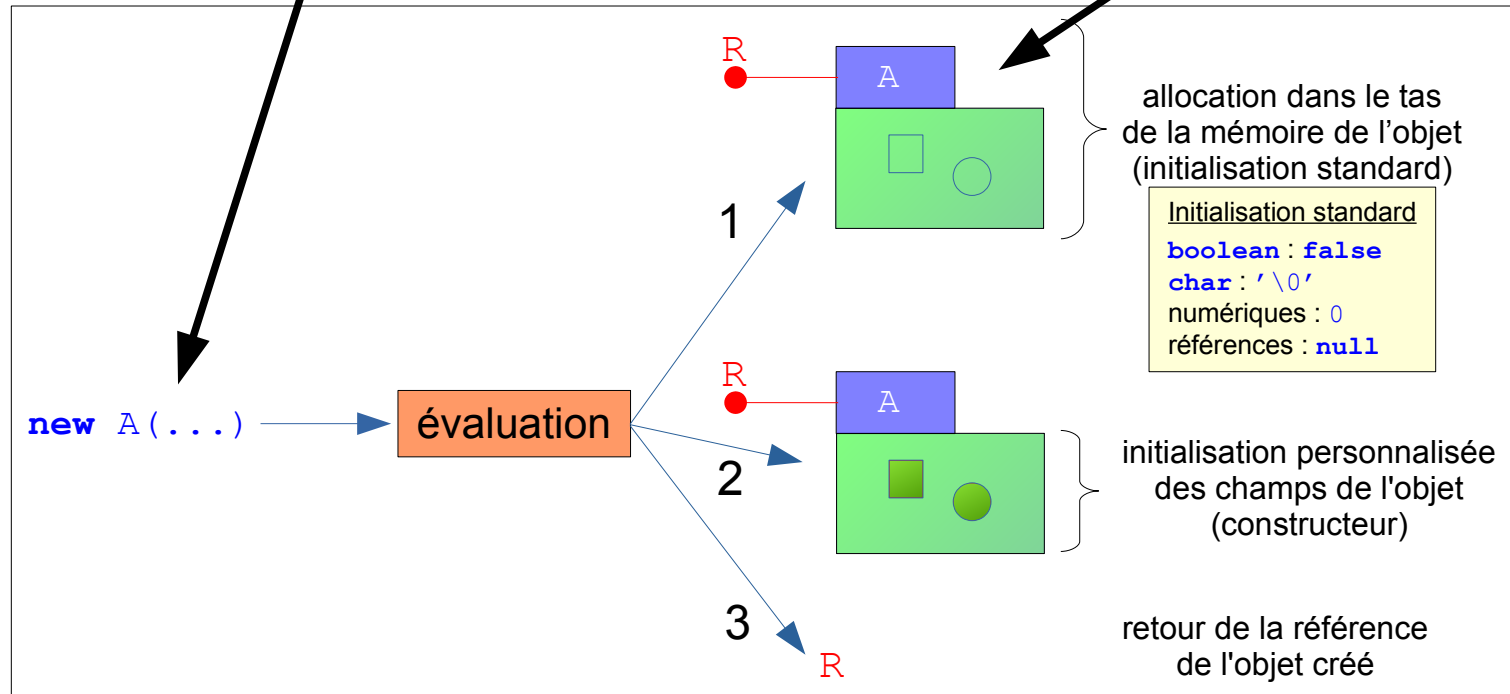
appel de **this()** (ou **super()**)  
en première ligne de code !

chaînage cohérent des appels !

## Rappels et compléments

- I. Objets
  1. Équation caractéristique
- II. Types de données
  1. Définition
  2. Description
    - i. Attributs
    - ii. Constructeurs
    - iii. Instance et instanciation
    - iv. Méthodes
      - a. Définition
      - b. Syntaxe
      - c. Passage de paramètres
    - v. Caractéristiques de classes
  3. Types références Java
    - i. Définitions
    - ii. Types tableaux
      - a. Tableaux
      - b. Boucle for étendue
    - iii. Types enum
      - a. Définition
      - b. Décompilation
  4. Boxing et unboxing
- III. Pannes logicielles
  1. Types Error et Exception
  2. Exceptions non contrôlées
- IV. Application Java
  1. Classe racine
  2. Compilation & Exécution
  3. Structure d'une application

L'*instanciation* d'une classe (concrète) produit une *instance*.



*Instance directe d'un type* :

objet créé conformément au texte de ce type.

*Instance indirecte d'un type* :

instance d'un type héritier de ce type.

*Type générateur d'un objet* :

type dont l'objet est une instance directe.

## Rappels et compléments

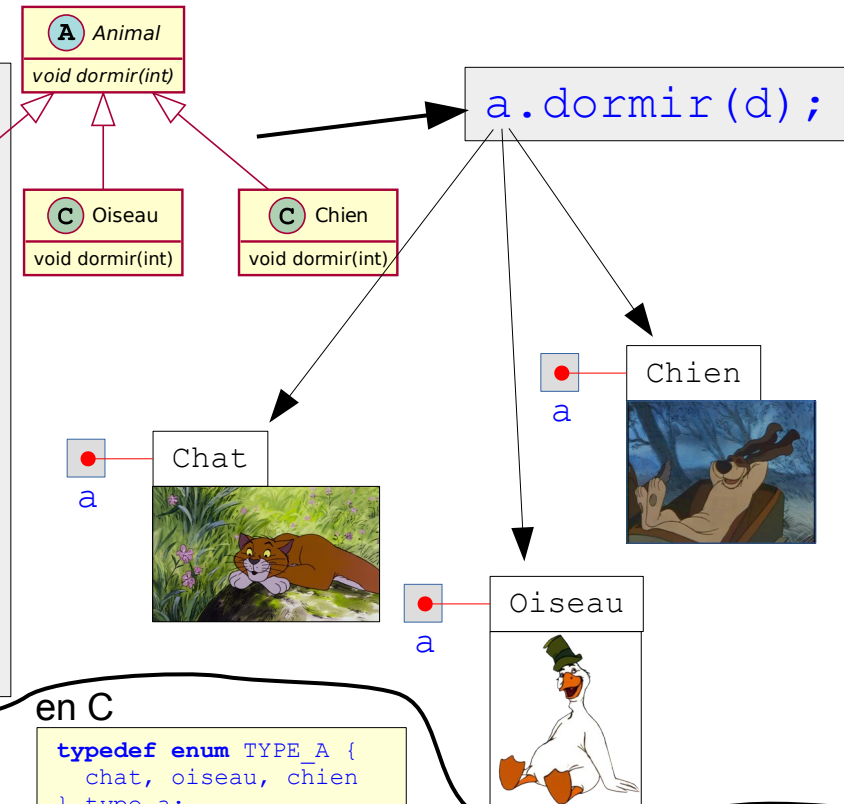
- I. Objets
  1. Équation caractéristique
- II. Types de données
  1. Définition
  2. Description
    - i. Attributs
    - ii. Constructeurs
    - iii. Instance et instanciation
    - iv. Méthodes
      - a. Définition
      - b. Syntaxe
      - c. Passage de paramètres
    - v. Caractéristiques de classes
3. Types références Java
  - i. Définitions
  - ii. Types tableaux
    - a. Tableaux
    - b. Boucle for étendue
  - iii. Types enum
    - a. Définition
    - b. Décompilation
4. Boxing et unboxing
- III. Pannes logicielles
  1. Types Error et Exception
  2. Exceptions non contrôlées
- IV. Application Java
  1. Classe racine
  2. Compilation & Exécution
  3. Structure d'une application

**Méthode** : sous-programme dont le contexte d'exécution est « lié » à un objet.

En réponse à un message reçu contenant le nom de la méthode, l'objet détermine le sous-programme correspondant et l'exécute.

### en Java

```
class Chat extends Animal {  
    void dormir(int d) {  
        // dormir comme un chat  
    }  
}  
  
class Oiseau extends Animal {  
    void dormir(int d) {  
        // dormir comme un oiseau  
    }  
}  
  
class Chien extends Animal {  
    void dormir(int d) {  
        // dormir comme un chien  
    }  
}
```



### en C

```
dormir(a, d);
```

```
typedef enum TYPE_A {  
    chat, oiseau, chien  
} type_a;  
  
typedef struct ANIMAL {  
    type_a t;  
    union {  
        struct {  
            ...  
        } CHAT;  
        struct {  
            ...  
        } OISEAU;  
        struct {  
            ...  
        } CHIEN;  
    } COC;  
} animal;
```

```
void dormir(animal *a, int d) {  
    if (a->t == chat) {  
        // dormir comme un chat  
    } else if (a->t == oiseau) {  
        // dormir comme un oiseau  
    } else if (a->t == chien) {  
        // dormir comme un chien  
    }  
}
```