Base de la Programmation Orientée Objet

IUT de Clermont-Ferrand
Université d'Auvergne

8 février 2016

Quiz

- Rappelez les différents niveaux de visibilité et ce qu'ils signifient ?
- À qui/quoi s'appliquent-ils ?
- Comment les représente-t-on en UML ?
- Comment les représente-t-on en Java ?

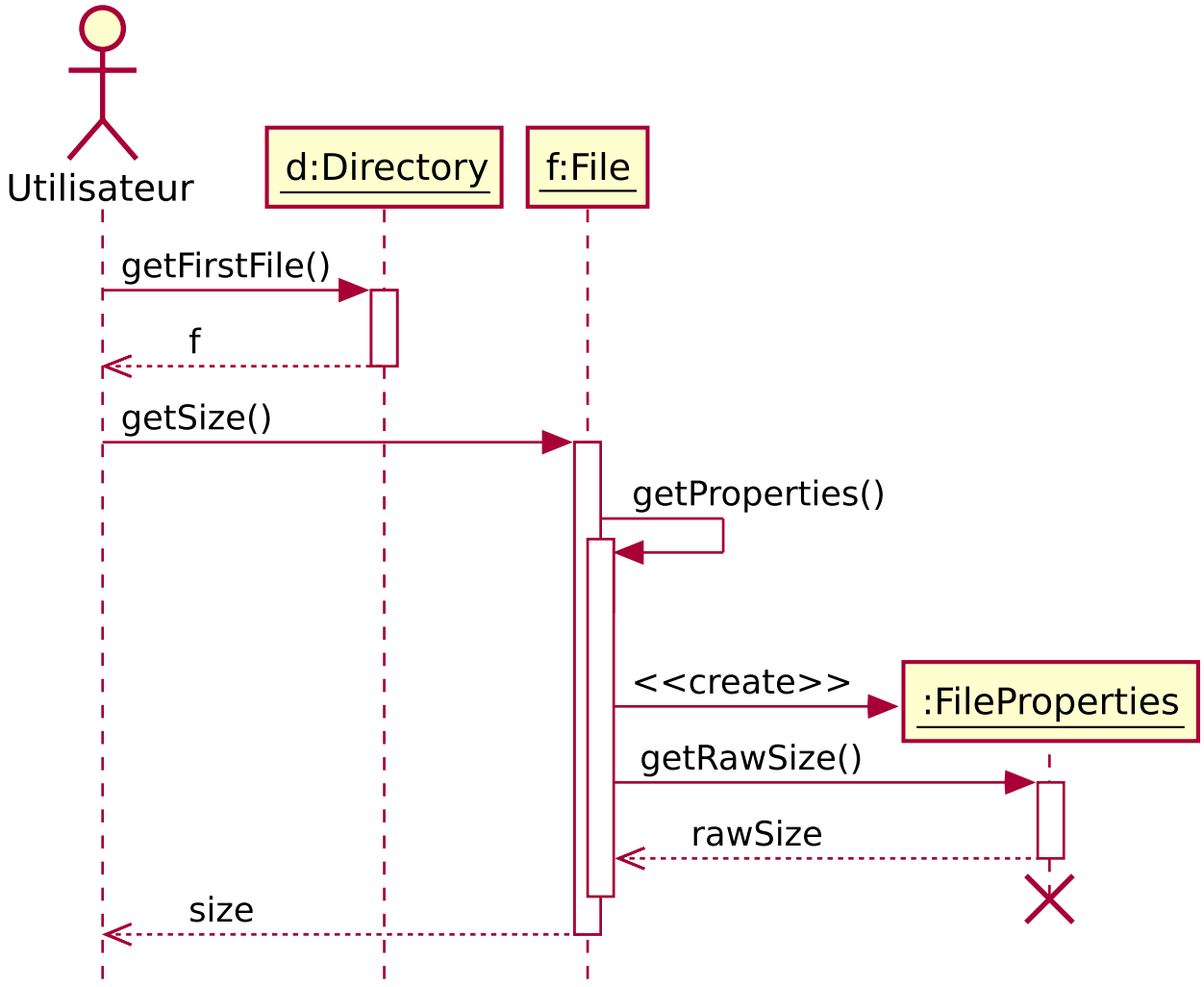
Quiz

- Qu'est-ce que l'encapsulation ?
- À quoi ça sert ?

Quiz

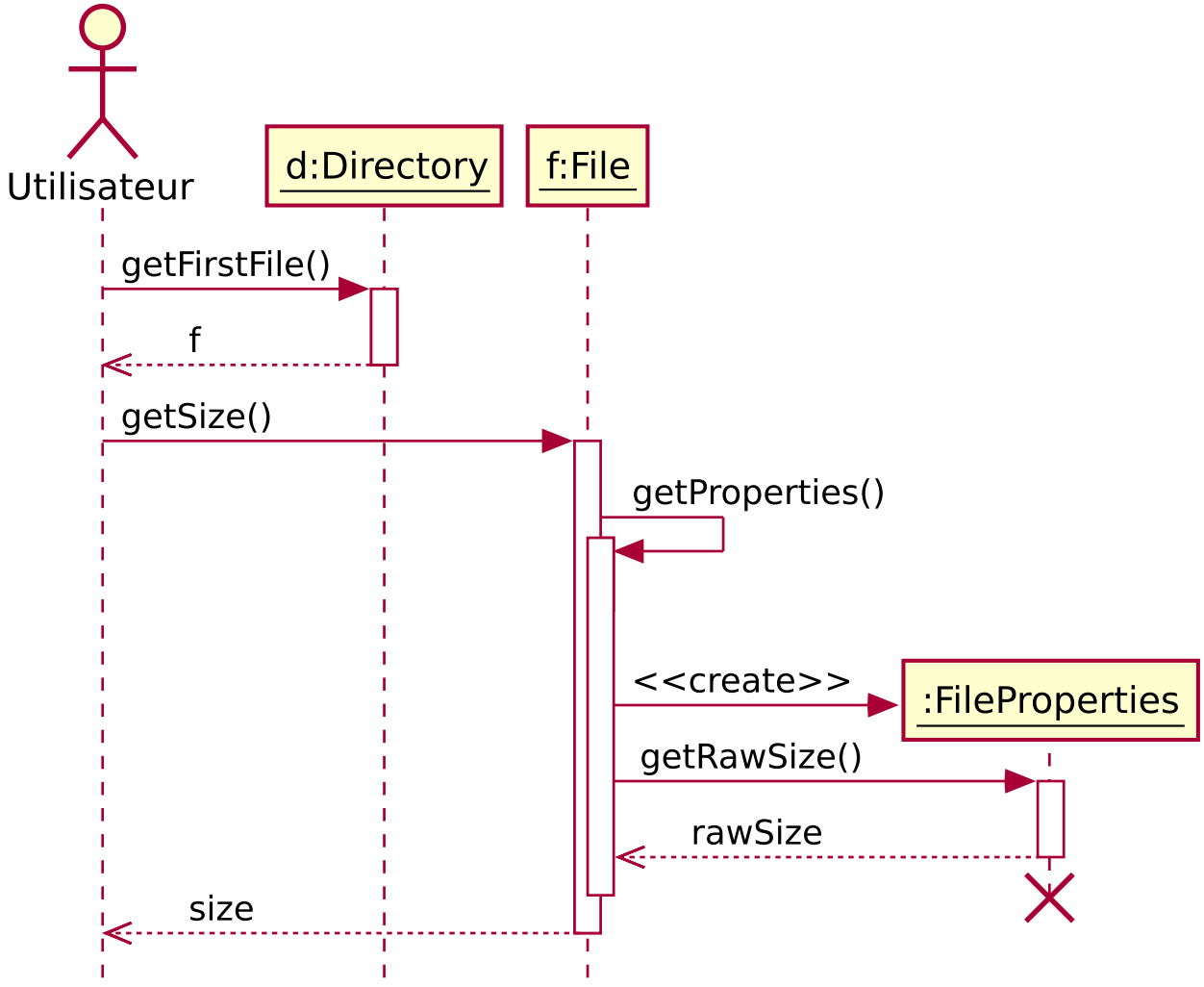
- Quelles sont les nuances entre déclarer, instancier et définir un tableau à 2 dimensions en Java ?
- Quelles sont les différentes syntaxes à employer ?
- Quelles sont les effets en mémoire ?

Quiz



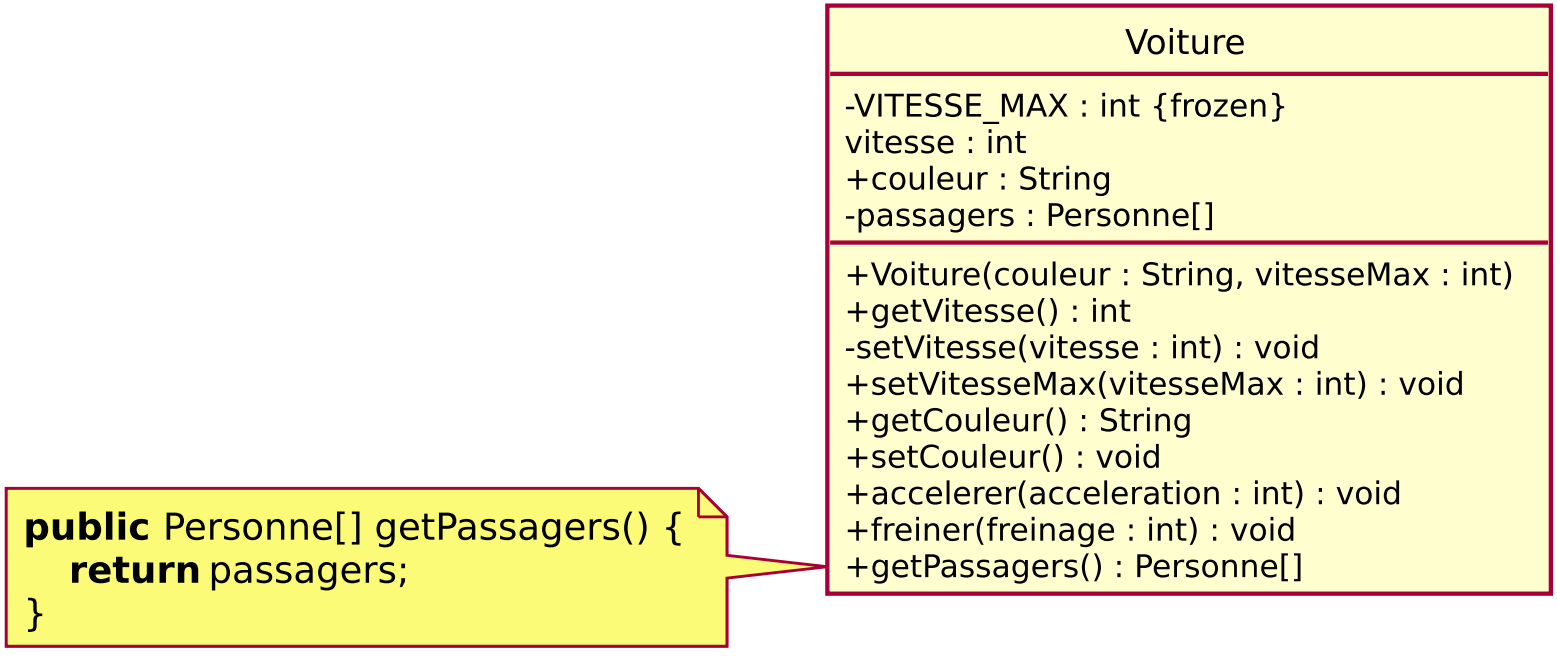
- Qu'est-ce ?

Quiz



- Que nous raconte ce diagramme ?

Quiz



- Qu'est-ce qui ne va pas ou est discutable sur ce diagramme ?

Traitement d'erreurs

```
#include ...
#include <string.h> /* strerror() */
#include <errno.h>  /* errno */

static void teste(const char *nombre)
{
    unsigned long res;

    /* On reinitialise errno */
    errno = 0;

    /* Appel de strtoul : conversion d'une chaine en un entier */
    res = strtoul(nombre, NULL, 10);

    /* Erreur strtoul si retour = ULONG_MAX et errno non nul. */
    if (res == ULONG_MAX && errno != 0)
    {
        /* Il y a eu une erreur ! */
        fprintf(stderr, "Erreur conversion (%s)", strerror(errno));
    }
    else
    {
        printf("Conversion OK. Valeur = %lu", res);
    }
}
```

Exceptions

Question : Comment gérer les erreurs dans un programme Java ?

Statiquement — À la compilation

- On essaye de détecter un maximum d'erreurs à la compilation du code
  ↪ merci au typage
  ↪ malheureusement, pas toujours possible

Dynamiquement — À l'exécution

Exceptions

Question : Comment gérer les erreurs dans un programme Java ?

Statiquement — À la compilation

Dynamiquement — À l'exécution

1. On met au point des codes d'erreurs
   - Les fonctions renvoient les codes d'erreurs pour signifier leur succès ou leur échec
     ↪ ⊖ lourd à gérer
     ↪ ⊖ difficile à traiter
     ↪ ⊖ mélange entre code « utile » et code de « contrôle »
     ↪ ⊖ pas naturel
2. On met en place une gestion « d'exceptions »

Exceptions

Question : Comment gérer les erreurs dans un programme Java ?

Statiquement — À la compilation

Dynamiquement — À l'exécution

1. On met au point des codes d'erreurs
2. On met en place une gestion « d'exceptions »
   - Lors d'une erreur un objet est générer pour décrire l'erreur et le programmeur peut utiliser cet objet pour traiter l'erreur
     $\hookrightarrow \oplus$ ciblage de l'erreur dans le code
     $\hookrightarrow \oplus$ séparation « cas normaux » / « cas exceptionnels »
     $\hookrightarrow \oplus$ une entité spéciale traite l'exception : le *gestionnaire d'exception* (exception handler)

Exceptions

- Qu'est-ce qu'un « cas exceptionnel »
  ↪ une situation qui ne correspond pas au « fonctionnement normal » du programme tel qu'on l'a envisagé

Diviser un nombre par 0 :	`ArithmeticException`
Envoyer un message en passant par une référence `null`	`NullPointerException`
Accéder à des cases d'un tableau en dehors des indices valables	`ArrayIndexOutOfBoundsException`
Downcaster un objet vers une classe dont il n'est pas une instance	`ClassCastException`
Tenter lire un fichier qui n'existe pas	`FileNotFoundException`

Exceptions

Exemple

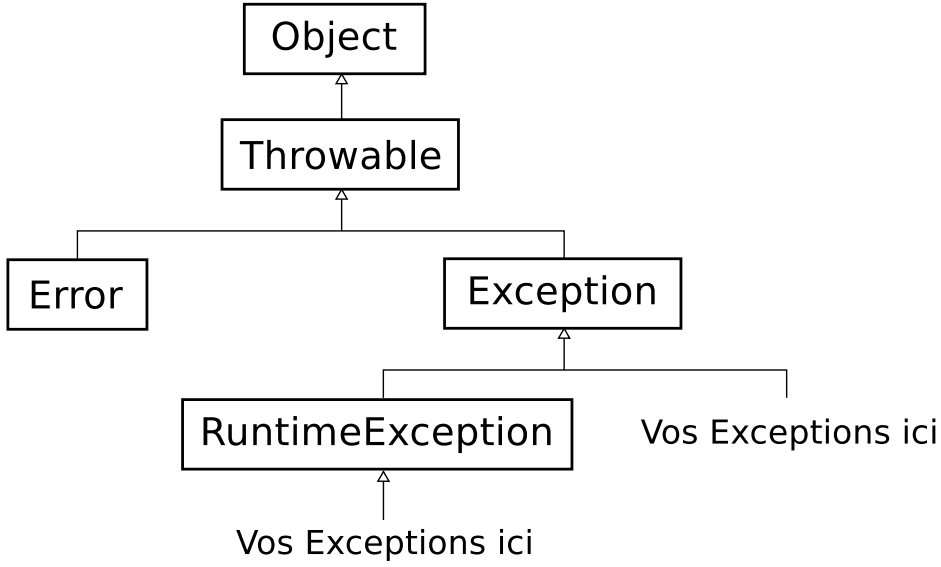
```
public class ExampleCasExeptionnel {
  public static void main (String[] args) {
      int monTab[] = {1, 2, 3, 4};
      for (int i = 0; i <= monTab.length; ++i)
        System.out.println(monTab[i]);
    }
  }
```

Résultat

```
1
2
3
4
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: 4 at
ExampleCasExeptionnel.main(ExampleCasExeptionnel.java:5)
```

Exceptions

- En Java les exceptions sont des objets



- Elles sont toutes du type `Exception` et leur type « précis » est un « sous-type » de `Exception`
- Par convention ces classes se nomment `QuelqueChoseException`

Exceptions

- Comment créer une nouvelle classe d'exceptions ?

Trivial : on utilise la solution par défaut

```
class SimpleException extends Exception {}
```

- Des portions de code peuvent *générer* ou *lever* (raise) des exceptions, signe d'un problème
- Le programmeur dispose d'un moyen de *gérer* ou *capturer* (catch) des exceptions, et ainsi proposer une solution ou une alternative à l'erreur rencontrée

Exceptions

Lever une exception

Instruction **throw**

```
String attention = "Je vais lancer un exception simple !";
throw new SimpleException();
String pasLa = "On n'arrive jamais ici";
```

- Les exceptions sont des objets !
- Nouvelle instance crée à la levée de l'exception
- Utilisation des constructeurs de `SimpleException`

Exceptions

Capturer une exception

Instructions **try**... **catch**

```
try {
  // Code susceptible de lancer une exception
}
catch (TypeExceptionACapturer e) {
  // Traitement de l'exception (infos à travers e)
}
```

- Lorsqu'une exception est levée cela n'arrête pas le progamme
- On quitte le bloc `try` dès qu'une exception est levée dans ce bloc
- Si l'exception est capturée, le traitement associé à cette capture est exécuté

Exceptions

Capturer une exception

- Un même bloc peut être susceptible de lever plusieurs exceptions
- Il est possible des les traiter séparément ou globalement

Exemple

```
try {
  double x = 1 / obj.getValue();
}
catch (NullPointerException e) {
  System.out.println("obj ne référence aucun objet valide !");
}
catch (ArithmeticException e) {
  System.out.println("La valeur de obj est zéro");
}
```

```
try {
  double x = 1 / obj.getValue();
}
catch (Exception e) {
  System.out.println("Ooops... voici l'erreur : " + e);
}
```

Exceptions

Capturer une exception

Clause **finally**

```
try {
  // Code susceptible de lancer une exception
}
catch (TypeException1 e1) {
  // Traitement de l'exception de type 1
}
...
catch (TypeExceptionN eN) {
  // Traitement de l'exception de type N
}
finally {
  // Bloc toujours exécuté
}
```

- Seul le premier bloc `catch` compatible avec l'exception levée est exécuté
- Les instructions du bloc `finally` sont toujours exécutées

Exceptions

Transférer une exception (exception specification)

- Si on ne désire pas traiter l'exception dans une fonction qu'on écrit, on le spécifie (on est poli) aux clients

Instruction **throws**

```
public void uneMethode(...) throws SimpleException {
    ...
}
```

- On peut transférer plusieurs exceptions
- Il faut spécifier toutes les exceptions susceptibles d'être levées par la fonction
- C'est vérifié par le compilateur Java (sauf pour les `RuntimeException`)

Membres de classe

Membre de classes

Un membre de classe est un attribut ou une méthode qui appartient directement à une classe et ne nécessite pas d'instance de cette classe pour être référencé.

- À opposer à *membre d'instance*
- En Java : mot-clé static
- Accès : `MaClasse.maMethodeStatique(…)`
- En UML : attribut ou méthode souligné

- Une méthode de classe ne peut pas accéder à un attribut d'instance !

Usage

```
public class Lapin {
  private static int nbLapins = 0;

  private String nom;
  private int nbCarottesMangees;

  public Lapin(String nom) {
    this.nom = nom;
    nbCarottesMangees = 0;
    ++nbLapins;
  }

  public static int getNbLapins() {
    return nbLapins;
  }

  public void mange() {
    ++nbCarottesMangees;
  }

  public String toString() {
    return nom + " a mangé " + nbCarottesMangees + " carotte(s).";
  }
}
```

Usage

```
public class Main {
  public static void main(String[] args) {
    System.out.println("Lapins créés : " + Lapin.getNbLapins());
    // Affiche Lapins créés : 0

    Lapin bugsBunny = new Lapin("Bugs Bunny");
    bugsBunny.mange();
    System.out.println(bugsBunny);
    // Affiche Bugs Bunny a mangé 1 carotte(s).

    Lapin rogerRabbit = new Lapin("Roger Rabbit");
    System.out.println("Lapins créés : " + Lapin.getNbLapins());
    // Affiche Lapins créés : 2
  }
```

Affectation constante

Affectation constante

Variable dont la valeur ne pourra plus être modifiée une fois l'affectation faite.

- En Java : mot-clé final
- En UML : contrainte `{frozen}`

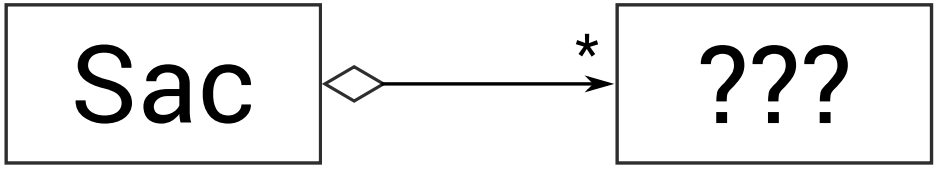
- Dans une classe : que pour les attributs (signification différente pour les méthodes)
- L'affectation doit se faire à la déclaration
- Dans une classe : au plus tard dans le constructeur

- Attribut `static` + `final` = CONSTANTE

Les collections d'objets

Activité

Quels sont les problèmes rencontrés pour implémenter la modélisation suivante :



Je veux :

- Pouvoir stocker n'importe quel type d'éléments
- Avoir un sac de taille indéterminée mais qui «grandit» à volonté (je veux donc pouvoir ajouter/supprimer des éléments à mon sac)
- Pouvoir parcourir les éléments du sac
- Pouvoir retrouver un élément donné dans mon sac