Héritage

Exemple complet

- Modélisation d'un questionnaire -

Modélisation d'un questionnaire

- **Objectif**
  - Concevoir une classe ou un ensemble de classes permettant de modéliser un questionnaire.
  - Un questionnaire est une séquence de plusieurs questions.
  - Chaque question est formulée à l'aide d'un texte. Une réponse est attendue. Sur base de la réponse, il est décidé si la question est réussie (acceptée) ou pas.
  - Les questions peuvent être de types différents
    - question avec réponse unique possible
    - question avec réponse à choisir dans une liste
    - question libre (tte réponse est acceptée)
    - question à choix multiples
    - etc.

Modélisation d'un questionnaire

- **Exemple**

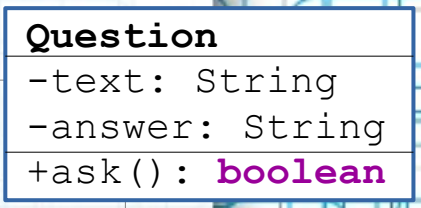
```
Qui a peint "L'empire des lumières" ?
René Magritte

Dans quelle province R. Magritte est-il né ?
   (0) Flandre orientale
   (1) Hainaut
   (2) Luxembourg
   (3) Liège
   (4) Namur
1
```

Modélisation d'un questionnaire

- **Première étape**
  - Modélisation du type question simple par une classe
  - Modélisation d'un questionnaire comme un tableau de questions
  - Application de test (classe séparée)
    - définir un questionnaire composé de 2-3 questions
    - poser les questions à l'utilisateur à travers la console
    - afficher le résultat pour chaque question (réussi/accepté ou pas)
  - Eventuellement 2 modes de fonctionnement
    - toutes les questions sont posées dans l'ordre de la séquence
    - les questions sont choisies aléatoirement

Modélisation d'un questionnaire

- **Question simple**




```
import java.util.Scanner;

public class Question {

  public final String text;
  public final String answer;

  public Question(String text, String answer) {
    this.text= text;
    this.answer= answer;
  }

  public boolean ask() {
    System.out.println(text);
    Scanner scan= new Scanner(System.in);
    String response= scan.nextLine();
    return answer.equals(response);
  }

}
```

Modélisation d'un questionnaire

- **Application de test**

```
public class QuestionTest {

  public static final Question [] questions= new Question[] {
    new Question("Qui a peint L'Empire des Lumières ?",
                 "René Magritte"),
    new Question("Quel mot-clé Java déclare une variable de classe ?",
                 "static"),
  };


  public static void main(String [] args) {
    for (int i= 0; i < questions.length; i++)
      if (questions[i].ask())
        System.out.println("Réponse correcte");
      else
        System.out.println("Réponse incorrecte");
  }

}
```

Modélisation d’un questionnaire

- **Seconde étape**
  - Identifier les tâches d'une question; découper l’implémentation en plusieurs méthodes.
  - Cette étape prépare au support de différents types de questions.
  - Poser une question c'est
    1) Afficher la question
    2) Recueillir la réponse (vérifier éventuellement la validité de l'entrée)
    3) Vérifier si la réponse correspond à la réponse attendue

Modélisation d'un questionnaire

- **Question simple (v2)**

```
import java.util.Scanner;

public class Question {

  public final String text;
  public final String answer;

  public Question(String text, String answer) {
    this.text= text;
    this.answer= answer;
  }

  public void display() {
    System.out.println(text);
  }

  public String ask() {
    Scanner scan= new Scanner(System.in);
    return scan.nextLine();
  }

  public boolean check(String response) {
    return answer.equals(response);
  }

}
```

Question
-text: String -answer: String
+display(): void +ask(): String +check(): boolean

Modélisation d'un questionnaire

- **Application de test (v2)**

```
public class QuestionTest {

  public static final Question [] questions= new Question[] {
    new Question("Qui a peint L'Empire des Lumières ?",
                 "René Magritte"),
    new Question("Quel mot-clé Java déclare une variable de classe ?",
                 "static"),
  };

  public static void main(String [] args) {
    for (int i= 0; i < questions.length; i++)
      questions[i].display();
      String response= questions[i].ask();
      if (questions[i].check(response))
        System.out.println("Réponse correcte");
      else
        System.out.println("Réponse incorrecte");
  }

}
```

Modélisation d'un questionnaire

- **Troisième étape**
  - Supporter différents types de questions
  - Première approche : attribut supplémentaire « `type` » dans la classe `Question`
  - Le comportement dépend de « `type` »

Question simple

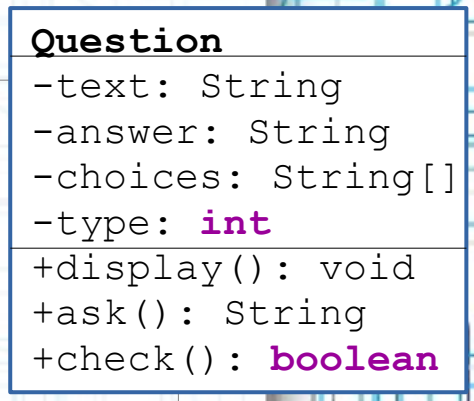
Question à choix multiples

```
Qui a peint "L'empire des lumières" ?
René Magritte

Dans quelle province R. Magritte est-il né ?
   (0) Flandre orientale
   (1) Hainaut
   (2) Luxembourg
   (3) Liège
   (4) Namur
1
```

Modélisation d'un questionnaire

- **Question (v3)**

```
public class Question {

  public final String text;
  public final String answer;
  public final String [] choices;
  public final int type;

  public static final int TYPE_SIMPLE = 0;
  public static final int TYPE_CHOICES= 1;
  public static final int TYPE_FREE   = 2;

  public Question(String text, String answer) {
    type= TYPE_SIMPLE;
    this.text= text;
    this.answer= answer;
  }

  public Question(String text, String answer,
                  String... choices) {
    type= TYPE_CHOICES;
    this.text= text;
    this.answer= answer;
    this.choices= choices;
  }

                                  /* (à suivre) */
```



Modélisation d'un questionnaire

- **Question (v3) – suite**

```
/* (suite Question) */

  public Question(String text) {
    type= TYPE_FREE;
    this.text= text;
  }

  public void display() {
    System.out.println(text);
    if (type == TYPE_CHOICES) {
      for (int i= 0; i < choices.length; i++)
        System.out.println("(" + i + ") " + choices[i]);
    }
  }

 public String check(String response) {
    if ((type == TYPE_SIMPLE) ||
        (type == TYPE_CHOICES))
      return answer.equals(response);
    else if (type == TYPE_FREE)
      return true;
  }

                                        /* (à suivre) */
```

Question
-text: String -answer: String -choices: String[] -type: int
+display(): void +ask(): String +check(): boolean

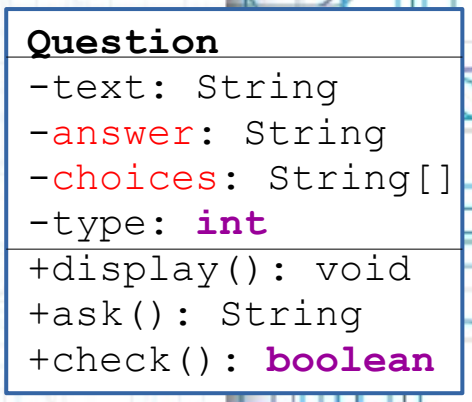
Modélisation d'un questionnaire

- **Question (v3) – suite**

```
/* (suite Question) */

  public String ask() {
    Scanner scan= new Scanner(System.in);
    if ((type == TYPE_SIMPLE) ||
        (type == TYPE_FREE)) {
      return scan.nextLine();
    } else if (type == TYPE_CHOICES) {
      while (true) {
        if (!scan.hasNextInt()) {
          System.out.println("La réponse doit être un entier");
          scan.next();
          continue;
        }
        int response= scan.nextInt();
        if ((response < 0) || (response >= choices.length)) {
          System.out.println("La réponse doit être entre 0 et " +
                             (choices.length - 1));
          continue;
        }
        return choices[response];
      }
    }
  }

}
```

Question
-text: String -answer: String -choices: String[] -type: int
+display(): void +ask(): String +check(): boolean

Modélisation d'un questionnaire



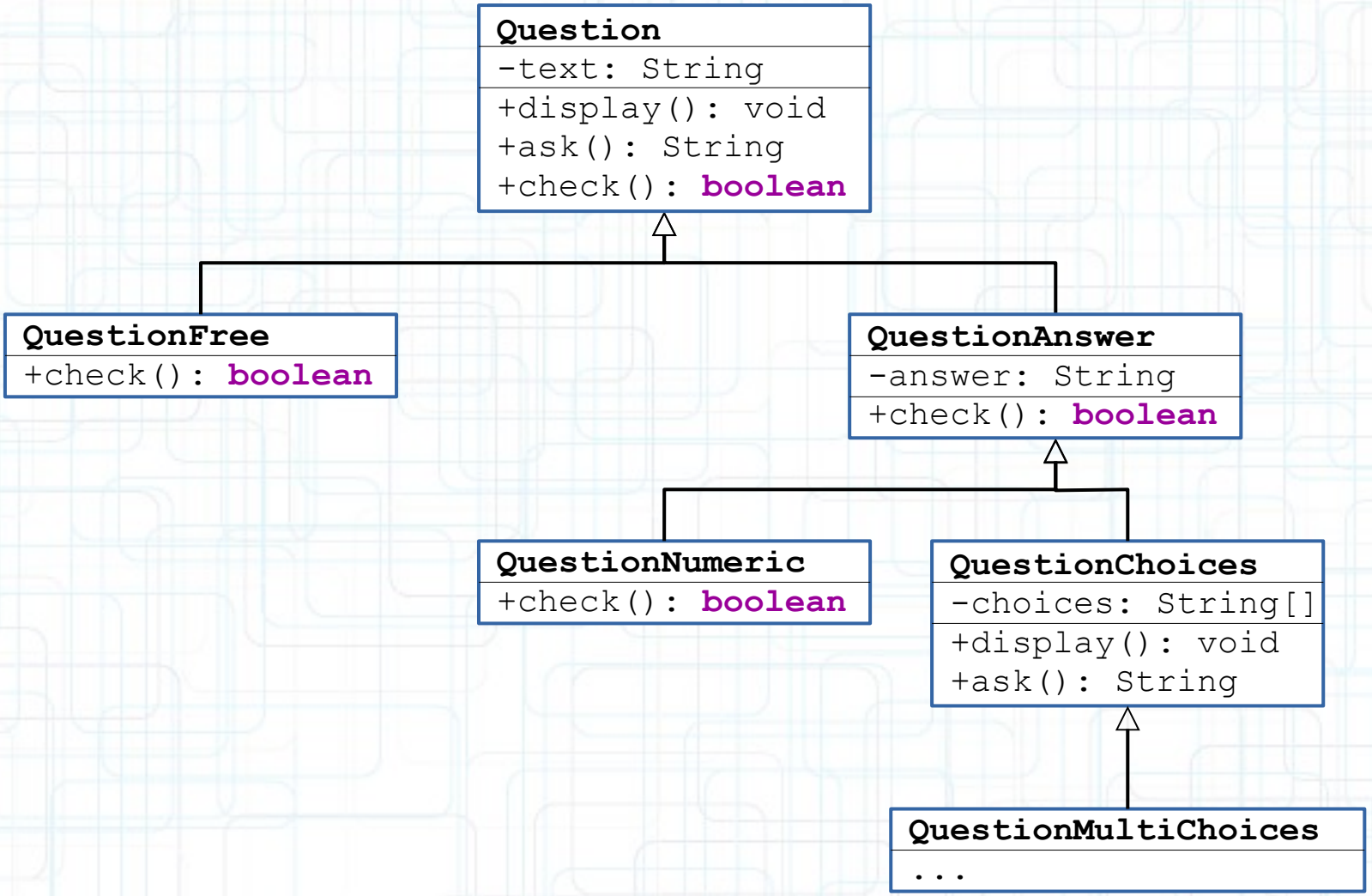
- **Problèmes de cette approche**
  - Certaines variables ne sont utiles qu'à certains types de questions
    - p.ex. `choices` n'est utile que pour `TYPE_CHOICES`, `answer` n'est pas utile pour `TYPE_FREE`.
    - à chaque nouveau type de question ajouté, de nouvelles variables peuvent être requises.
    - risque : accéder à une variable qui n'a pas de sens ou qui n'est pas initialisée pour un type donné.
  - Nombre important de tests (**`if`** / **`else`**) sur la variable type répartis dans plusieurs méthodes afin de spécialiser le comportement de la classe.
    - p.ex. pour `TYPE_CHOICES`, nécessité d'afficher les choix possibles et choix exprimé sous forme d'un entier.
    - risque : oublier l'ajout de tests lors de l'ajout d'un nouveau type de question ou l'ajout de nouvelles méthodes.

Modélisation d'un questionnaire

- **Approche orientée-objet**
  - Supporter différents types de questions
  - Une classe par type de question
  - Héritage afin de factoriser les comportements communs
  - Re-définition de méthodes afin de spécialiser les comportements individuels

Modélisation d'un questionnaire

- **Hiérarchie de classes**



Modélisation d'un questionnaire

- **Question libre**
  - Une « question libre » est implémentée en héritant d'une question/réponse.
  - Seule la méthode vérifiant la réponse est re-définie (spécialisée) : toute réponse est correcte.
  - Les autres méthodes sont obtenues inchangées par héritage.

```
public class QuestionFree extends Question {

  public QuestionFree(String text) {
    super(text);
  }

  public boolean check(String response) {
      return true;
  }

}
```

Appel du constructeur du parent (ré-utilisation)

Modélisation d'un questionnaire

- **Question avec choix**
  - Une question avec choix est implémentée par héritage.
  - Spécialisation de la structure : ajout d'un tableau de choix.
  - Re-définition de `ask()` et `display()` de façon à afficher les choix possibles et obtenir une réponse entière.

```
public class QuestionChoices extends QuestionAnswer {

  public final String [] choices;

  public QuestionChoices(String text, int answerIndex,
                         String... choices) {
    super(text, choices[answerIndex]);
    this.choices= choices;
  }

  public void display() {
    super.display();
    for (int i= 0; i < choices.length; i++)
      System.out.println("(" + i + ") " + choices[i]);
  }

                                          /* (à suivre) */
```

Appel du constructeur du parent (ré-utilisation)

Appel de la méthode définie dans le parent (ré-utilisation)

Modélisation d'un questionnaire

- **Question avec choix – suite**

```
/* (suite QuestionChoices) */

  public String ask() {
    Scanner scan= new Scanner(System.in);
    while (true) {
      if (!scan.hasNextInt()) {
        System.out.println("La réponse doit être un entier");
        scan.next();
        continue;
      }
      int response= scan.nextInt();
      if ((response < 0) || (response >= choices.length)) {
        System.out.println("La réponse doit être entre 0 et " +
                           (choices.length - 1));
        continue;
      }
      return choices[reponse];
    }
  }

}
```

Modélisation d'un questionnaire

- **Application de test**

```
public class QuestionTest {

  public static final Question [] questions= new Question[] {
    new QuestionAnswer("Qui a peint L'Empire des Lumières ?",
                       "René Magritte"),
    new QuestionChoices("Dans quelle province R. Magritte est-il né ?",
                        1, "Flandre orientale", "Hainaut", "Namur",
                        "Luxembourg", "Liège"),
    new QuestionAnswer("Quel mot-clé Java déclare une variable de classe ?",
                       "static"),
    new QuestionFree("Quel est votre prénom ?"),
  };

  public static void main(String [] args) {
    for (int i= 0; i < questions.length; i++)
      questions[i].display();
      String response= questions[i].ask();
      if (questions[i].check(response))
        System.out.println("Réponse correcte");
      else
        System.out.println("Réponse incorrecte");
  }

}
```

L'ajout de nouveaux types de question n'a pas eu d'impact sur le code client.
Ce dernier interagit avec les questions au travers de l'interface formée des méthodes `display()`, `ask()` et `check()`.

Modélisation d'un questionnaire

- **Conclusion**
  - L'approche orientée-objet permet de ré-utiliser du code par héritage.
  - L'ajout de variables permet de spécialiser la structure (p.ex. tableau `choices` dans `QuestionChoices`)
  - La re-définition de méthode permet de spécialiser le comportement. Disparition des tests (**`if`** / **`else`**).
  - Une méthode re-définie peut utiliser la méthode du parent avec **`super`**`.` Pour un constructeur, utilisation de **`super`**`()`.

Modélisation d'un questionnaire

- **Observations**
  - Le découpage préalable de la méthode unique `ask()` en 3 méthodes `display` / `ask` / `check` a permis une spécialisation plus fine.
    - `QuestionFree` n'a redéfini que `check()`
    - `QuestionChoices` a redéfini `ask()` et `display()`
  - Pour bien tirer parti de la programmation orienté-objet, il sera souvent nécessaire de ré-organiser son programme avant d'y introduire héritage et redéfinition de méthodes.
  - La méthode `ask()` pourrait déléguer la vérification de l'entrée (ex. entier) à une autre méthode. Les sous-classes pourraient alors uniquement redéfinir cette dernière.