ÉTS
Le génie pour l'industrie

LOG121
Conception orientée objet

Patron Méthode Template
Frameworks

Chargée de cours: Souad Hadjres

Plan

- Patron Méthode Template
- Les frameworks
- Exemples simples de framework
  - Librairie Java des composants graphiques
  - Les collections

Exemple de problème de conception

- La compagnie Starbuzz Café offre des cafés et des thés.
- Les recettes pour préparer le café et le thé:

1) Bouillir de l'eau
2) Infuser le café dans l'eau bouillante
3) Verser le café dans la tasse
4) Ajouter du sucre et du lait

1) Bouillir de l'eau
2) Faire tremper le thé dans l'eau bouillante
3) Verser le thé dans la tasse
4) Ajouter du citron

Une première conception


```
public class Cafe {
    void preparer() {
        bouillir_eau();
        infuser_cafe();
        verser_tasse();
        ajouter_lait_sucre();
    }
    public void bouillir_eau(){
        System.out.println("L'eau est entrain
                de bouillir!");
    }
    public void infuser_cafe() {
        System.out.println("Le café est entrain
            d'infuser!");
    }
    public void verser_tasse() {
        System.out.println("Je vous sers dans
            une tasse");
    }
    public void ajouter_lait_sucre() {
        System.out.println("J'ajoute du lait et
            du sucre!");
    }
}
```

```
public class The {
    void preparer() {
        bouillir_eau();
        tremper_the();
        verser_tasse();
        ajouter_citron();
    }
    public void bouillir_eau() {
        System.out.println("L'eau est
            entrain de bouillir!");
    }
    public void tremper_the() {
        System.out.println("Le thé est
            entrain de tromper dans l'eau!");
    }
    public void verser_tasse() {
        System.out.println("Je vous sers
            dans une tasse");
    }
    public void ajouter_citron() {
        System.out.println("J'ajoute du
            citron!");
    }
}
```

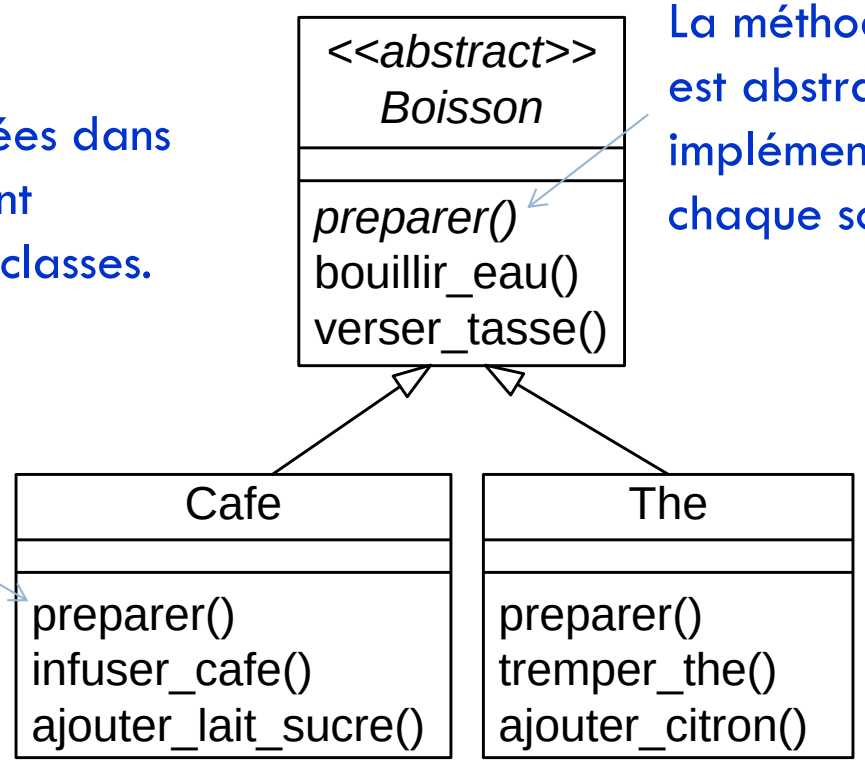
Problèmes avec notre première conception

- Le même code se trouve dans les deux classes
- L'ajout d'un autre type de boisson qui se prépare de la même façon va introduire plus de code dupliqué
- La mise à jour d'un bout de code doit se faire dans plusieurs places

Premier essai

- On abstrait les parties communes des classes Thé et Café dans une super-classe

Les méthodes bouillir_eau() et verser_tasse() sont implémentées dans la super-classe puisqu'elles sont similaires pour toutes les sous-classes.



La méthode preparer() est abstraite. Elle est implémentée par chaque sous-classe.

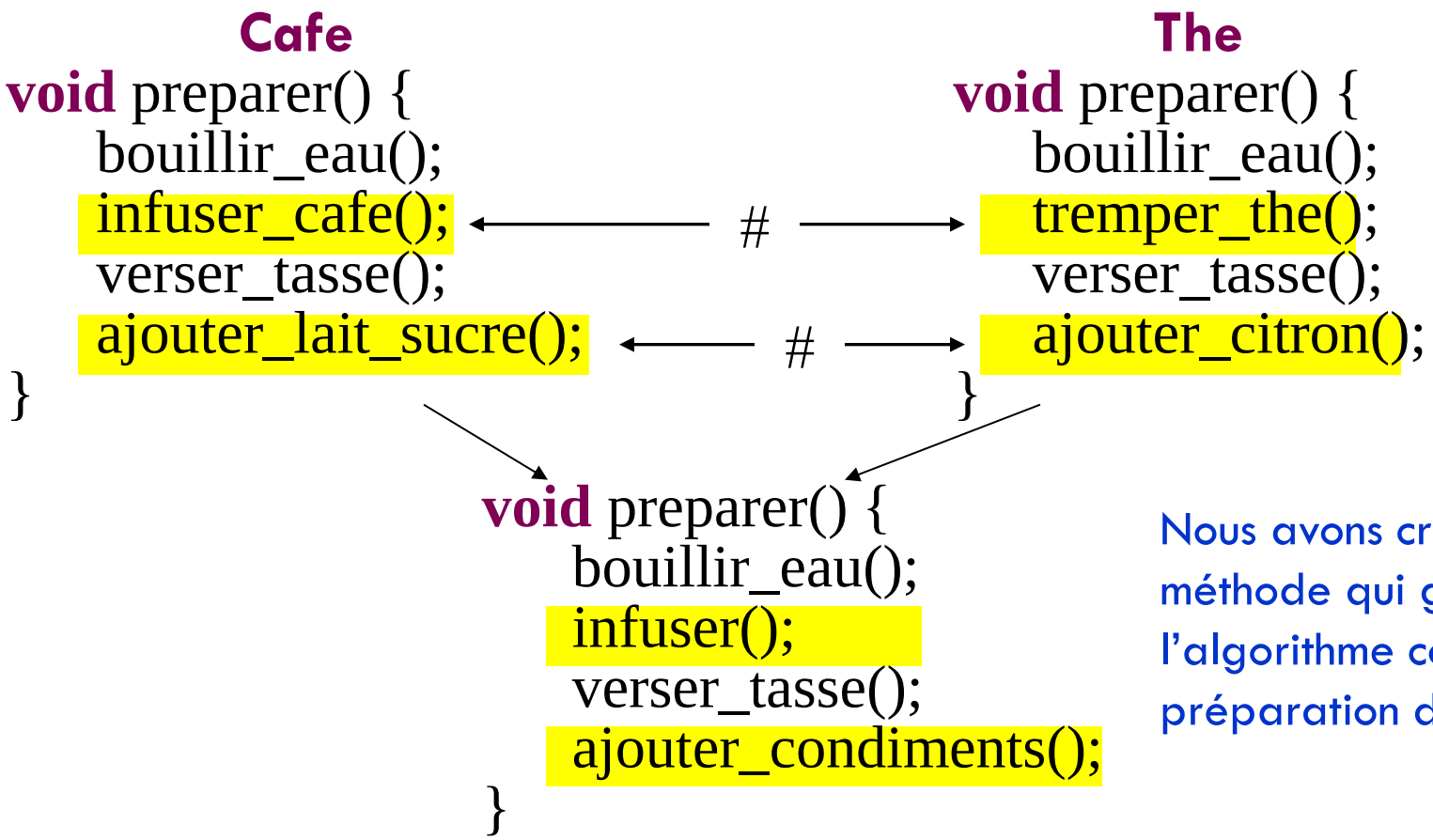
preparer() appelle les méthodes bouillir_eau() et verser_tasse() de la superclasse et les méthodes infuser_cafe() et ajouter_lait_sucre()

- A-t-on factorisé toutes les parties communes?

Aller plus loin dans la conception

- La préparation des deux boissons suit le même algorithme:
  1) Bouillir de l'eau
  2) Infuser le café ou le thé dans l'eau bouillante
  3) Verser la boisson dans la tasse
  4) Ajouter des condiments
- A-t-on un moyen de factoriser l'algorithme de préparation des boissons?

Aller plus loin dans la conception



Nous avons créé une méthode qui généralise l'algorithme commun de préparation de boisson.

- Il faut modifier la conception pour intégrer cette méthode dans la superclasse Boisson

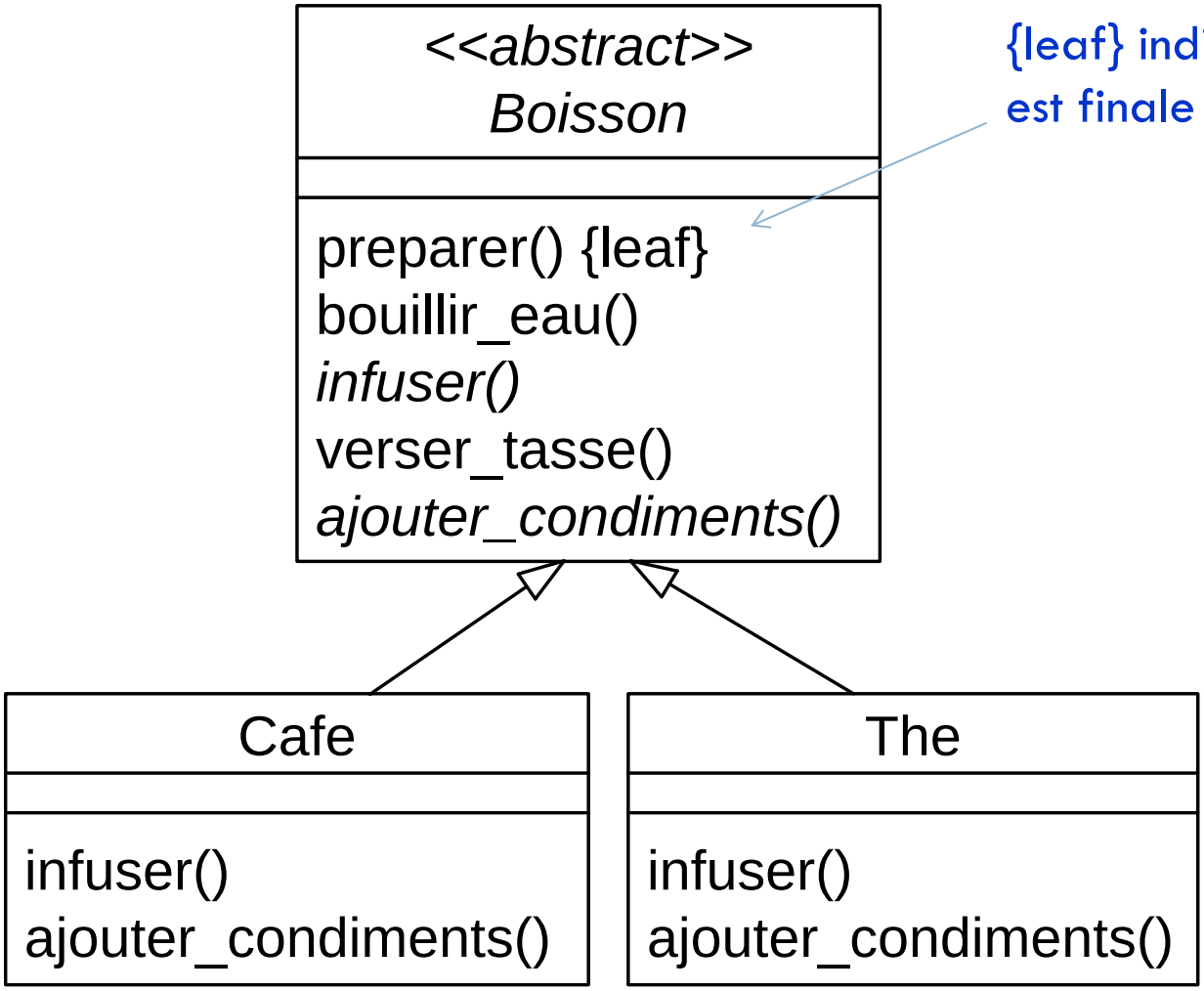
Solution au problème


```
Public abstract class Boisson {
    final void preparer() {
        bouillir_eau();
        infuser();
        verser_tasse();
        ajouter_condiments();
    }

    abstract void infuser();

    abstract void ajouter_condiments();

    public void bouillir_eau() {
        System.out.println("L'eau est
            entrain de bouillir!");
    }

    public void verser_tasse() {
        System.out.println("Je vous sers
            dans une tasse");
    }
}
```

```
public class The extends Boisson{
    public void infuser() {
        System.out.println("Le thé est
            entrain de tromper dans l'eau!");
    }
    public void ajouter_condiments() {
        System.out.println("J'ajoute du
            citron!");
    }
}
```

```
public class Cafe extends Boisson{
    public void infuser() {
        System.out.println("Le café est entrain
            d'infuser!");
    }
    public void ajouter_condiments() {
        System.out.println("J'ajoute du lait et
            du sucre!");
    }
}
```

Solution au problème



Solution au problème

- La méthode préparer() est appelée une « ***Méthode Template*** ».
- Cette méthode contrôle l'algorithme de préparation en définissant précisément les étapes de cet algorithme.
- Chaque étape de l'algorithme correspond à une méthode.
- Certaines méthodes sont implémentées dans la superclasse.
- D'autres méthodes sont déclarées abstraites permettant aux sous-classes de les implémenter pour varier certaines étapes de l'algorithme.

Patron « Méthode Template »

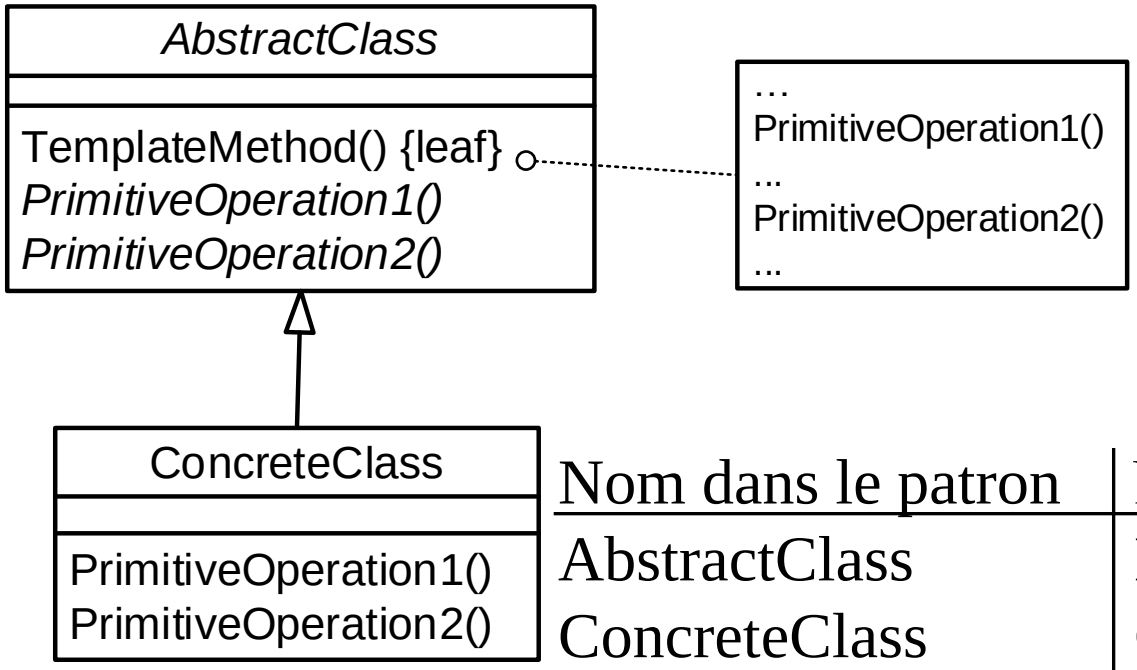
- Contexte
  - On a un algorithme qui s'applique à plusieurs types.
  - L'algorithme peut être divisé en *opérations primitives* que les types peuvent implémenter différemment.
  - L'ordre des opérations primitives ne dépend pas d'un type.

Patron « Méthode Template »

- Solution
  - Définir une superclasse abstraite AbstractClass qui implémente une méthode « template » définissant le squelette de l'algorithme.
  - AbstractClass définit des méthodes primitives abstraites que les sous-classes vont implémenter.
  - La méthode « template » implémente l'algorithme en appelant les opérations primitives dans l'ordre approprié.
    - La méthode « template » est finale

Patron « Méthode Template »

- La structure du patron dans GoF



Nom dans le patron	Nom dans l'exemple
AbstractClass	Boisson
ConcreteClass	Café, Thé
TemplateMethod	preparer
PrimitiveOperation1	infuser
PrimitiveOperation2	ajouter_condiments

Patron « Méthode Template »

- Quelles sont les conséquences d'utilisation de ce patron?
  - On a supprimé la duplication du code et maximisé sa réutilisation.
  - La mise à jour d'un bout de code se fait à une seule place.
  - On contrôle les extensions permises aux sous-classes.

Patron « Méthode Template »

- Le patron « Méthode template » est utilisé dans les frameworks.
  - Il supporte l'implémentation de l'inversion de contrôle.
  - C'est la superclasse qui contrôle le flux d'exécution.
    - C'est la superclasse qui appelle des méthodes fournies par les sous-classes.

Les patrons « Méthode Template » et Stratégie

- Les patrons Stratégie et « Méthode template » s'intéressent à des algorithmes. Quelle est la différence entre les deux?
  - Le patron Stratégie encapsule une famille d'algorithmes dans des objets, permettant à l'utilisateur d'utiliser et d'ajouter des algorithmes facilement.
  - Le patron « Méthode template » définit le squelette d'un algorithme dont certaines parties sont implémentées différemment par les sous-classes. La méthode « Template » contrôle la structure de l'algorithme.

Plan

- Patron Méthode Template
- Les frameworks
- Exemples simples de framework
  - Librairie Java des composants graphiques
  - Les collections

Les frameworks

- Un cadre de développement « *framework* » est une librairie de classes réutilisables
  - Les classes du framework implémentent les mécanismes essentiels à un aspect particulier
  - Elles peuvent être étendues pour implémenter de nouvelles fonctionnalités
  - Exemples: JavaFx, Swing, Junit, Java Collections, etc.
- Un framework fournit une interface bien définie (API: *Application programming interface*)

Les frameworks

- Un cadre d'application « *application framework* » est une librairie de classes qui implémente des services communs à un type d'application
  - Il fournit un système logiciel générique pour un domaine d'application
  - Le programmeur étend les classes du framework pour implémenter les fonctionnalités spécifiques à sa propre application
  - Exemples
    - Mozilla a été utilisé dans le développement de plusieurs applications telles que Netscape, Firefox, Seamonkey, Google Adwords Editor, etc.
    - Plusieurs frameworks permettent de faciliter le développement des application Web selon l'architecture MVC
      - Angular, Laravel, Apache Struts, Ruby on Rail, etc.

Les frameworks

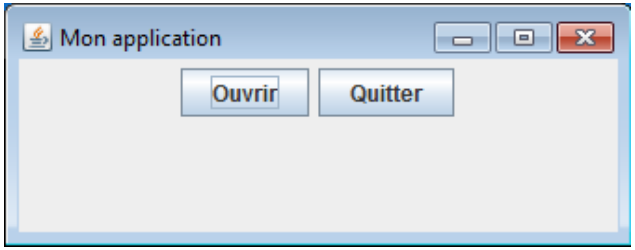
- **Inversion de contrôle:** Lorsqu'une application est développée en utilisant un framework, c'est le framework qui contrôle le flux des tâches réalisées par l'application
  - L'activité est réalisée par les classes du framework qui font appel ensuite au besoin à celles fournies par le programmeur
  - Le principe d'Hollywood
    - « ***Don't call us, we'll call you.*** »

Plan

- Patron Méthode Template
- Les frameworks
- Exemples simples de framework
  - Librairie Java des composants graphiques
  - Les collections

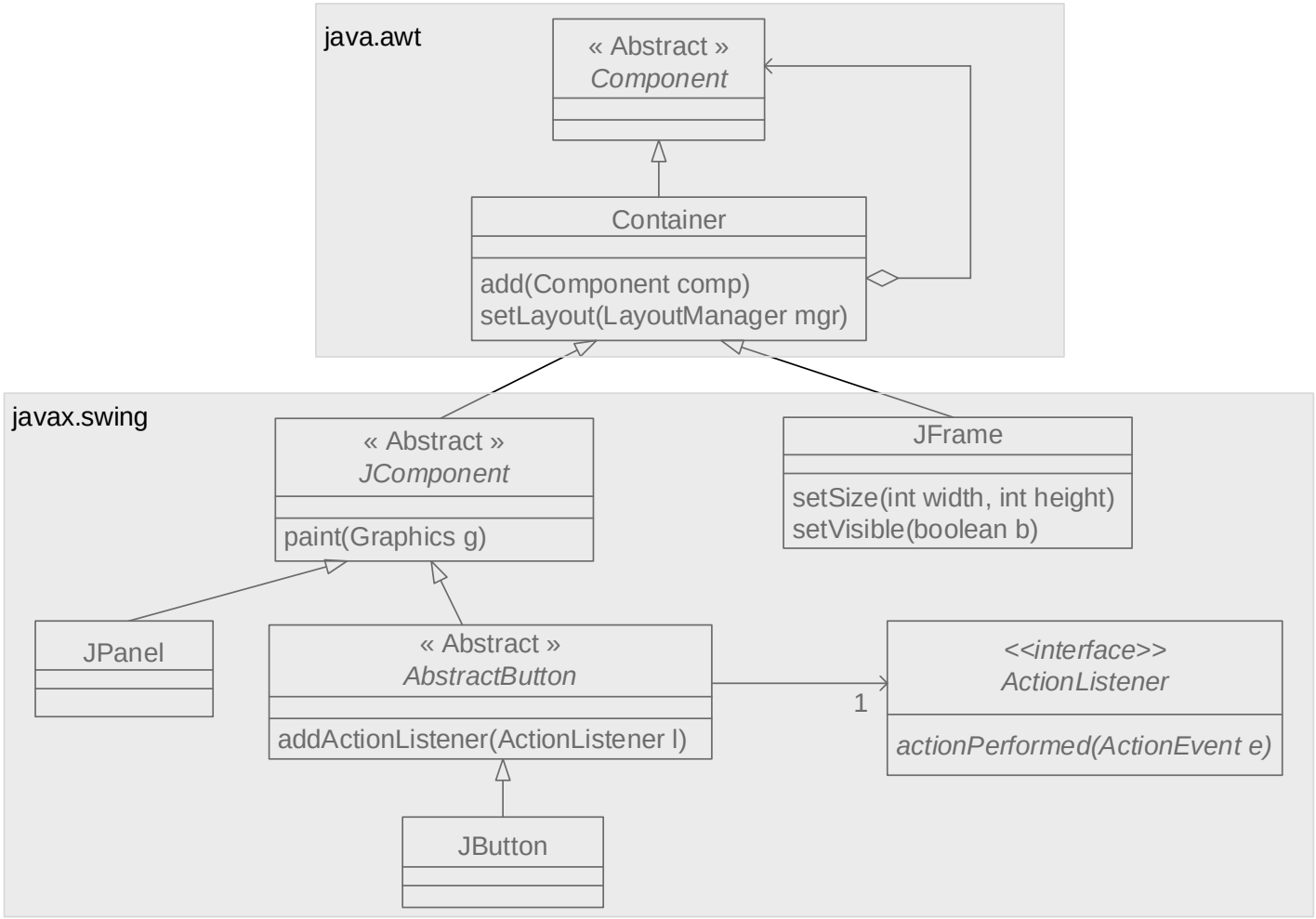
Librairie Java des composants graphiques

- Le paquetage javax.swing constitue un exemple simple de framework
  - Il fournit une libraire de classes facilitant la création des interfaces graphiques de vos application
  - Le programmeur crée son application en
    - Étendant la classe JFrame de ce package,
    - Instanciant d'autres classes du package, et
    - Implémentant ou redéfinissant certaines méthodes définies par les classes et interfaces du package
    - Le flux d'appel à ces méthodes est implémenté dans les classes de la librairie Java
      - Le programmeur ne s'en préoccupe pas



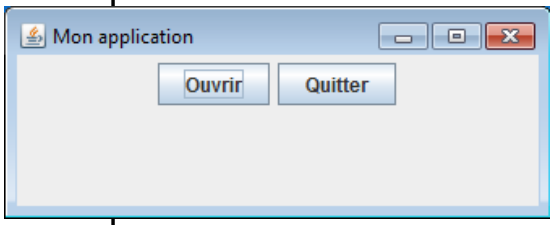
Librairie Java des composants graphiques

- Extrait de la librairie Java des composants graphiques

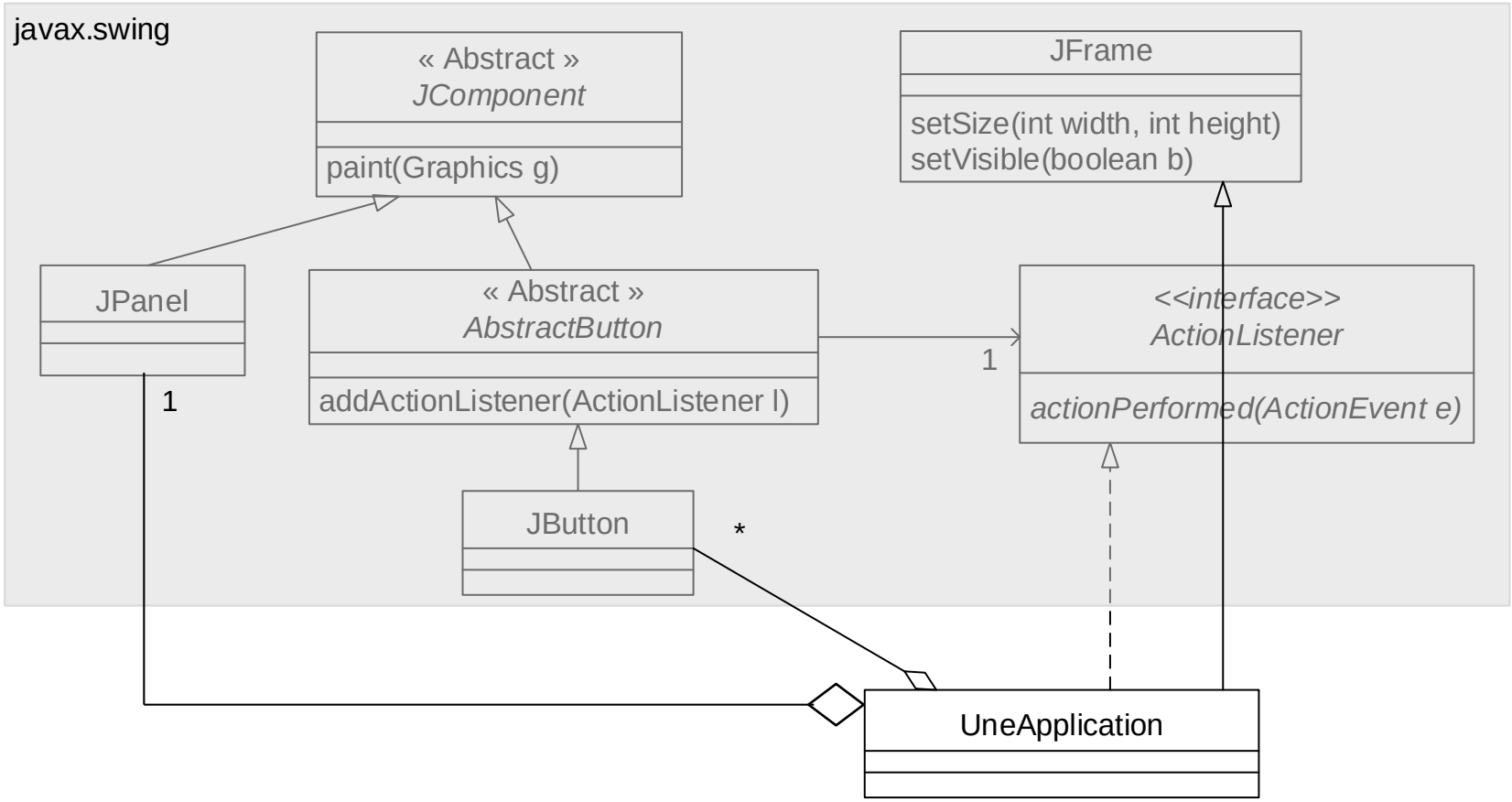


Librairie Java des composants graphiques


```
public class UneApplication extends JFrame implements ActionListener {
  private JButton openButton = new JButton("Ouvrir");
  private JButton exitButton = new JButton("Quitter");
  public UneApplication() {
     super("Mon application");
     setSize(400, 400);
     JPanel myPanel = new JPanel();
     myPanel.add(openButton);
     myPanel.add(exitButton);
     add(myPanel);
     openButton.addActionListener(this);
     exitButton.addActionListener(this);
  }
  public void actionPerformed(ActionEvent event) {
     if (event.getSource()== openButton)
        JOptionPane.showMessageDialog((Component)event.getSource(), "Ouvrir");
     else if (event.getSource()== exitButton)
        System.exit(0);
  }
  public static void main(String[] argv) {
     UneApplication application = new UneApplication();
     application.setVisible(true);
  }
}
```



Librairie Java des composants graphiques

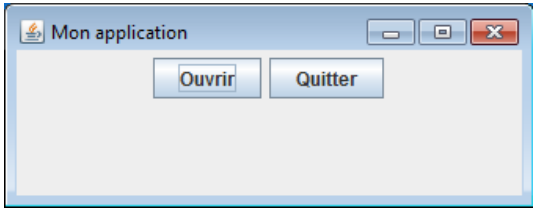


Librairie Java des composants graphiques


```
public class UneApplication extends JFrame implements ActionListener {
  private JButton openButton = new JButton("Ouvrir");
  private JButton exitButton = new JButton("Quitter");
  public UneApplication() {
     super("Mon application");
     setSize(400, 400);
     JPanel myPanel = new JPanel();
     myPanel.add(openButton);
     myPanel.add(exitButton);
     add(myPanel);
     openButton.addActionListener(this);
     exitButton.addActionListener(this);
  }
  public void actionPerformed(ActionEvent event) {
     if (event.getSource()== openButton)
        JOptionPane.showMessageDialog((Component)event.getSource(), "Ouvrir");
     else if (event.getSource()== exitButton)
        System.exit(0);
  }
  public static void main(String[] argv) {
     UneApplication application = new UneApplication();
     application.setVisible(true);
  }
}
```

L'inversion de contrôle:

- Le développeur implémente un ActionListener pour permettre à l'application de réagir aux actions sur les bouttons;
- Pour cela, il implémente la méthode actionPerformed déclarée dans ActionListener;
- L'appel à la méthode actionPerformed se fait par la librairie Swing; le développeur n'a pas le contrôle sur l'appel.



Librairie Java des composants graphiques

- Le paquetage `javax.swing` est un exemple typique de framework
  - Le programmeur utilise l'héritage pour créer sa propre application
    - Ici, on étend la classe JFrame représentant une fenêtre (ou un cadre)
    - La classe JFrame implémente le comportement générique et commun à toutes les fenêtres graphiques
  - Le programmeur implémente aussi des interfaces définies par le package
    - Ces interfaces déclarent des méthodes que le programmeur doit implémenter
  - Le programmeur ne se préoccupe pas de contrôler l'ordre d'exécution des méthodes qu'il hérite ou implémente

    ⇨ L'inversion de contrôle

Plan

- Patron Méthode Template
- Les frameworks
- Exemples simples de framework
  - Librairie Java des composants graphiques
  - Les collections

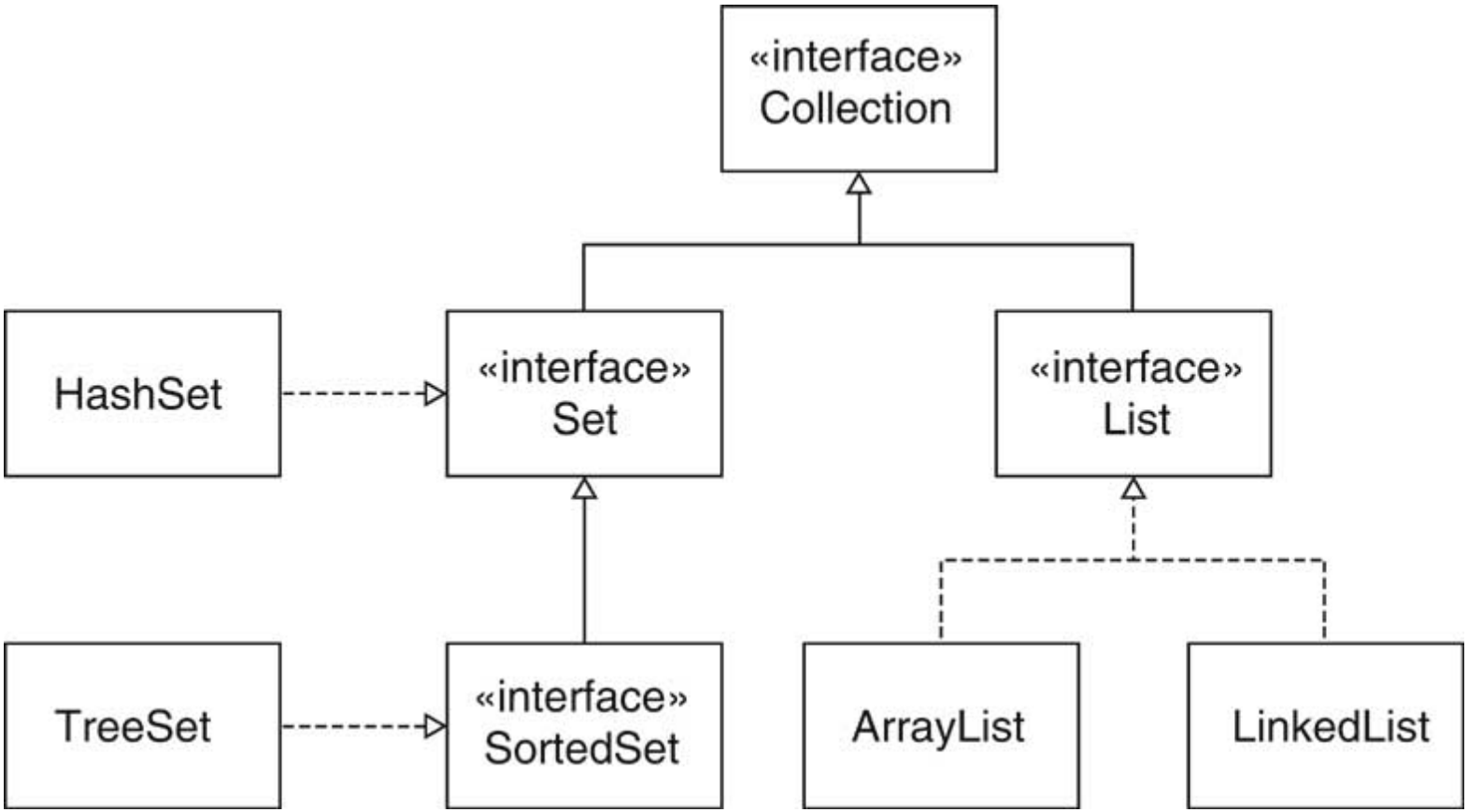
Les collections: un autre framework

- La librairie java des collections fournit des structures de données utiles
- Cette librairie a aussi été conçue comme un framework
  - On peut l'utiliser pour définir de nouvelles classes de collection et de nouveaux services
  - Les nouvelles classes intègrent les nouveaux services et interagissent avec les classes existantes

Les collections: un autre framework

- Quelques interfaces définies par le framework
  - `Collection`: le type le plus général
  - `Set`: une collection qui ne permet pas la duplication des éléments
  - `SortedSet`: est un Set dont les éléments sont visités selon un ordre de tri
  - `List`: une collection ordonnée (indexée)
- Quelques classes fournies par le framework
  - `HashSet`: une implémentation de Set qui utilise le hachage pour localiser les éléments du Set
  - `TreeSet`: une implémentation de `SortedSet` qui stocke ses éléments dans un arbre binaire équilibré
  - `LinkedList` et `ArrayList`: deux implémentations de l'interface `List`

Les collections: un autre framework



Les collections: un autre framework

- Deux interfaces fondamentales du framework
  - `Collection`: une structure de données qui stocke des objets d'une certaine façon
  - `Iterator`: définit le mécanisme pour visiter les éléments d'une collection

Interface Iterator
boolean hasNext()
E next()
void remove()

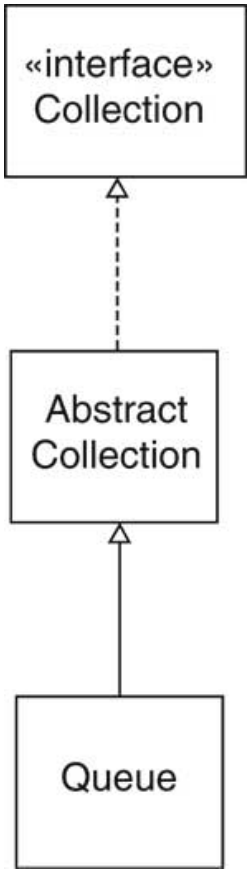
Interface Collection
boolean add(E obj)
boolean addAll(Collection c)
void clear()
boolean contains(E obj)
boolean containsAll(Collection c)
boolean equals(E obj)
int hashCode()
boolean isEmpty()
Iterator iterator()
boolean remove(E obj)
boolean removeAll(Collection c)
boolean retainAll(Collection c)
int size()
E[] toArray()
E[] toArray(E[] a)

Les collections: un autre framework

- Pour minimiser l'effort nécessaire pour implémenter l'interface `Collection`
  - Le framework fournit une classe `AbstractCollection` qui implémente toutes les méthodes de `Collection` sauf les méthodes `size()` et `iterator()`
  - Pour définir une nouvelle collection concrète, il suffit d'étendre `AbstractCollection` et de fournir une implémentation des méthodes `size()` et `iterator()`
  - La plupart des collections concrètes redéfinissent aussi les méthodes `add()` et `remove()`

Les collections: un autre framework

- À consulter: l'exemple dans le chapitre 8 du livre de référence
  - Ajout d'une file dans le framework des collections
    - La file était utilisée pour stocker les messages
  - On définit une classe qui étend AbstractCollection
  - On redéfinit la méthode add
  - On fournit un itérateur avec une méthode remove qui ne fait rien
- Quel est l'avantage d'ajouter une classe au Framework des collections?
  - Réutiliser les méthodes qui s'appliquent à toutes les collections (ex: addAll, containsAll, …)
  - Réutiliser les algorithmes définies et implémentés dans la classe Collections (tri, min, max, etc).



```java
public class BoundedQueue<E> extends AbstractCollection<E> {

   private Object[] elements;
   private int head;
   private int tail;
   private int count;

   /**
       Constructs an empty queue.
       @param capacity the maximum capacity of the queue
       @precondition capacity > 0
   */
   public BoundedQueue(int capacity){
      elements = new Object[capacity];
      count = 0;
      head = 0;
      tail = 0;
   }

   public Iterator<E> iterator(){
      return new
         Iterator<E>() {
            public boolean hasNext(){
               return visited < count;
            }

            public E next(){
               int index = (head + visited) % elements.length;
               E r = (E) elements[index];
               visited++;
               return r;
            }

            public void remove() {
               throw new UnsupportedOperationException();
            }

            private int visited = 0;
         };
   }

   /**
       Remove object at head.
       @return the object that has been removed from the queue
       @precondition size() > 0 */
   public E remove(){
      E r = (E) elements[head];
      head = (head + 1) % elements.length;
      count--;
      return r;
   }

   /**
       Append an object at tail.
       @param anObject the object to be appended
       @return true since this operation modifies the queue.
       (This is a requirement of the collections framework.)
       @precondition !isFull() */
   public boolean add(E anObject){
      elements[tail] = anObject;
      tail = (tail + 1) % elements.length;
      count++;
      return true;
   }
   public int size() {
      return count;
   }

   /**
       Checks whether this queue is full.
       @return true if the queue is full
   */
   public boolean isFull(){
      return count == elements.length;
   }

   /**
       Gets object at head.
       @return the object that is at the head of the queue
       @precondition size() > 0 */
   public E peek() {
      return (E) elements[head];
   }
}
```

Les collections: un autre framework

- Le type interface `Set`
  - `Set` étend `Collection` cependant il n'ajoute aucune méthode à Collection!!
  - La différence est conceptuelle
    - Un `Set` est une collection qui élimine les doublons
    - Ces éléments ne sont pas indexés
    - Deux Sets sont égaux s'ils contiennent les mêmes éléments mais pas nécessairement dans le même ordre
  - Les méthodes `add` et `equals` de `Set` ont donc des restrictions conceptuelles différentes de celles définies dans `Collection`
  - En fournissant une interface `Set`, une méthode peut spécifier un `Set` comme paramètre et rejeter les collections qui ne sont pas des Sets

Les collections: un autre framework

- Le type interface `List`
  - Étend `Collection` pour représenter une collection ordonnée où on peut accéder à chaque élément avec un index entier
  - `List` fournit un itérateur spécifique pour les listes `ListIterator`
  - `ListIterator` permet de parcourir la liste dans deux directions, de modifier la liste durant l'itération et d'obtenir la position actuelle de l'itérateur dans la liste

Interface ListIterator<E>

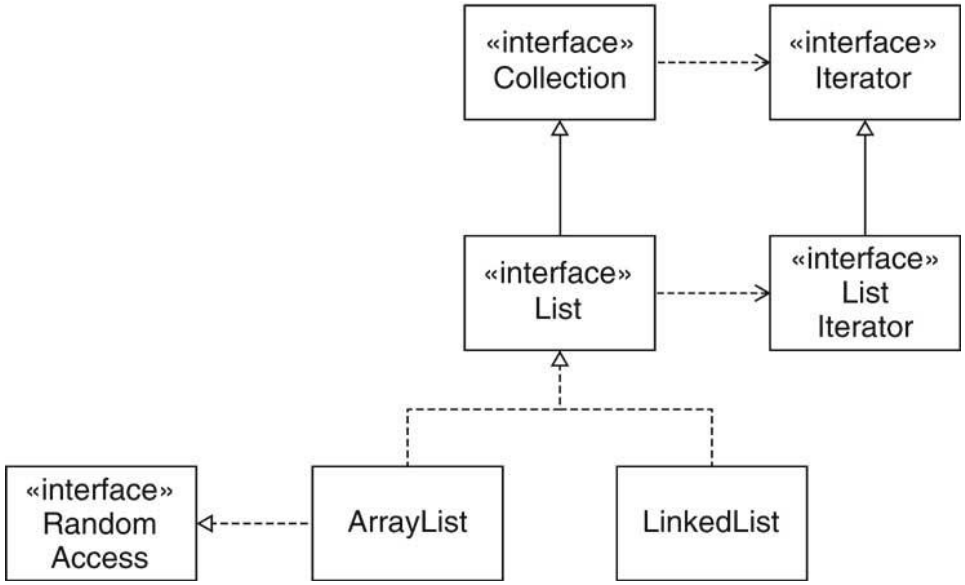
```
void add(E obj)
boolean hasNext()
boolean hasPrevious()
E next()
int nextIndex()
E previous()
int previousIndex()
void remove()
void set(E obj)
```

Méthodes ajoutées par List

```
boolean add(int index, E obj)
boolean addAll(int index, Collection c)
E get(int index)
int indexOf(E obj)
int lastIndexOf(E obj)
ListIterator listIterator()
ListIterator listIterator(int index)
E remove(int index)
E set(int index, E obj)
List subList(int fromIndex, int toIndex)
```

Les collections: un autre framework

- `ArrayList` et `LinkedList`(!!!!) implémentent `List`
- L'accès à une liste chaînée par index est lent: pour accéder à un élément à un index donné on doit d'abord parcourir tous ces prédécesseurs
- Ce problème a été résolu dans Java 1.4 en introduisant le type interface `RandomAccess`. Cette interface est implémentée par des classes pour indiquer qu'elles supportent l'accès aléatoire rapide (l'accès se fait en un temps constant).



Les frameworks

- Un framework est généralement construit en appliquant plusieurs patrons de conception
  - On verra plusieurs patrons que l'on peut utiliser dans l'implémentation des frameworks
  - Méthode Template
  - Observateur
  - Stratégie
  - Commande
  - Mémento
  - Prototype
  - …