L'interface en Java

Introduction

- Autre utilisation du terme interface
  - L'interface d'un objet représente sa partie visible (ses méthodes publiques)
- Dans un fichier JAVA, il est possible de définir une « interface » qui soit simplement une liste de signatures de méthodes
- Il sera possible de déclarer dans une classe que l'on implémente (réalise) cette interface, c.a.d. que l'on implémente (concrétise) toutes les méthodes qui y sont déclarées

Introduction (2)

- **L'interface permet de définir un comportement abstrait, qui sera implémenté par plusieurs classes**
- Pour pouvoir compiler la classe qui implémente une interface, il faut implémenter toutes les méthodes de l'interface en les redéfinissant

Exemple d'interface prédéfinie

- java.lang.Comparable<T>
  - Méthode compareTo
  - Pour les tri
    - Voir D:\abc_26_JAVA\16_ELEMENTS DU LANGAGE_ALGORITHMIE\46_TABLEAUX_de_type_primitif_ACONTINUER\00_aSLIDE
- java.lang.cloneable
- java.lang.serializable

Interface prédéfinie

java.lang

Interface Comparable<T>

Type Parameters:

T - the type of objects that this object may be compared to

All Known Subinterfaces:

Delayed, Name, RunnableScheduledFuture<V>, ScheduledFuture<V>

All Known Implementing Classes:

Authenticator.RequestorType, BigDecimal, BigInteger, Boolean, Byte, ByteBuffer, Calendar, Character, CharBuffer, Charset, ClientInfoStatus, CollationKey, Component.BaselineResizeBehavior, CompositeName, CompoundName, Date, Date, Desktop.Action, Diagnostic.Kind, Dialog.ModalExclusionType, Dialog.ModalityType, Double, DoubleBuffer, DropMode, ElementKind, ElementType, Enum, File, Float, FloatBuffer, Formatter.BigDecimalLayoutForm, FormSubmitEvent.MethodType, GregorianCalendar, GroupLayout.Alignment, IntBuffer, Integer, JavaFileObject.Kind, JTable.PrintMode, KeyRep.Type, LayoutStyle.ComponentPlacement, LdapName, Long, LongBuffer, MappedByteBuffer, MemoryType, MessageContext.Scope, Modifier, MultipleGradientPaint.ColorSpaceType, MultipleGradientPaint.CycleMethod, NestingKind, Normalizer.Form, ObjectName, ObjectStreamField, Proxy.Type, Rdn, Resource.AuthenticationType, RetentionPolicy, RoundingMode, RowFilter.ComparisonType, RowIdLifetime, RowSorterEvent.Type, Service.Mode, Short, ShortBuffer, SOAPBinding.ParameterStyle, SOAPBinding.Style, SOAPBinding.Use, SortOrder, SourceVersion, SSLEngineResult.HandshakeStatus, SSLEngineResult.Status, StandardLocation, String, SwingWorker.StateValue, Thread.State, Time, Timestamp, TimeUnit, TrayIcon.MessageType, TypeKind, URI, UUID, WebParam.Mode, XmlAccessOrder, XmlAccessType, XmlNsForm

Exemple d'écriture d'une d'interface biblio.metier.Comparable

```
public interface Comparable {
   boolean estEgal( Object o);
   boolean estSuperieur( Object o);
   boolean estInferieur( Object o);
}
```

Exemple de définition d'interface (suite)

- Une interface se crée dans un fichier portant son nom (ex:Comparable.java) et se déclare comme une classe, à l'exception du mot clé "interface" qui prend la place du mot « class »
- Toutes les méthodes d'une interface utilisent implicitement les modificateurs "public abstract" et sont suivis d'un « ; »
- C'est la signature des méthodes

Exemple d'implémentation de l'interface

```
public class Carre extends Rectangle implements Comparable{
  public Carre(  double cote) {
    super( cote, cote);
  }
  public boolean estEgal( Object o) {
    Carre c= (Carre )o;
    if( couleur.equals( c.couleur) && longueur = = c.longueur)
      return true;
    else
      return false;
  }
  public boolean estSuperieur( Object o) {
    Carre c= (Carre )o;
    return longueur > c.longueur;
  }
  public boolean estInferieur( Object o) {
    Carre c= (Carre )o;
    return longueur < c.longueur;
  }}
```

L'interface est un contrat

- Si une classe implémente Comparable, elle s'engage à implémenter toutes ses méthodes
- Le nommage des interfaces peut se faire avec des noms se terminant par "able" : Comparable, Triable, Affichable, Imprimable, Serializable, stockable, cloneable, ...

Héritage entre les interfaces

- Une interface peut hériter d'une autre interface, et même de plusieurs autres interfaces en Java
  - L'héritage multiple est ici autorisé, car il n'y a pas de problème de conflits d'implémentation
- Lorsque dans une classe, on implémente une interface qui hérite d'autres interfaces, nous devons évidemment implémenter toutes les méthodes de toutes ces interfaces
  - Il faut donc toutes les examiner, c'est la raison pour laquelle, il vaut mieux ne pas trop utiliser l'héritage dans les interfaces

Les références d'interface

- Un objet qui implémente une interface est du type de cette interface (opérateur instanceof)
- Ainsi, bien que l'on ne puisse pas créer d'objet de type d'une interface, rien ne nous empêche de déclarer une variable du type interface et de demander à un composant de vous fournir un objet qui implémente cette interface
  - Principe de Liskov

Exemple d'utilisation

```
Comparable m = new Carre(25);
```

- Attention, l'objet m étant de type Comparable, seules les méthodes de l'interface Comparable peuvent être invoquées, bien que cet objet soit un Carre !

Des propriétés dans l'interface

- On peut déclarer des propriétés dans une interface (« public final static » implicite)
- L'héritage multiple des interfaces peut générer des ambiguïtés sur le nom au niveau du compilateur.Pour éviter cet effet de bord, les propriétés des interfaces sont systématiquement "public static final", même si on ne les déclare pas comme telles.
- Leur accès ne peut se faire que par le nom de l'interface (~ propriété de classe).
- Utilisation comme constantes globales pour plusieurs classes de l'application.

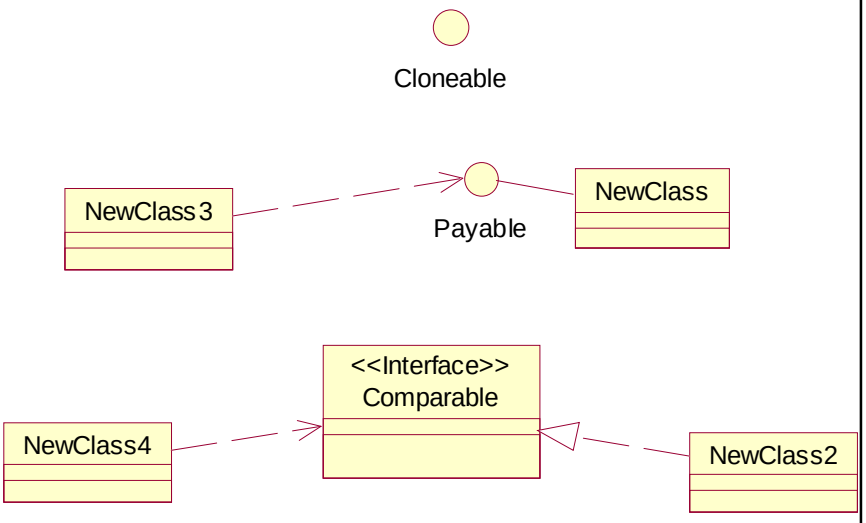
Exemple

```
Public interface MaConstante {
  int ANNEE  = 1970;
  …
}
Public class UnTest {
  public static void main( String [] args )
{
    System.out.println( MaConstante.ANNEE);
   …
  }
}
```

Autre exemple (proche d'enum)

```
Public interface JoursDeLaSemaine {
   int LUNDI = 1, MARDI =2, MERCREDI=3, JEUDI=4, VENDREDI=5, SAMEDI=6,
DIMANCHE=7;
}
```


Représentation d'une interface en UML



Les interfaces fonctionnelles (Java8)

- Depuis Java 8 et pour la programmation fonctionnelle
- @FunctionalInterface
- interface qui n'a qu'une seule méthode
- Permet l'utilisation des lambda expressions

Conclusion

- Les intefaces permettent :
  - L'héritage multiple de concepts
  - L'engagement d'implémenter toutes les méthodes définies dans les interfaces (notion de contrat)
  - Le transtypage d'objets incompatibles vers des types interfaces
    - On a la notion d'ensembles d'objets (sort())