# Software Development

## *Part 3: Interface*

Koen Pelsmaekers

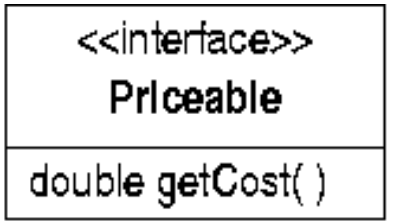
Unit Informatie (GT 03.14.05)

email: koen.pelsmaekers@kuleuven.be

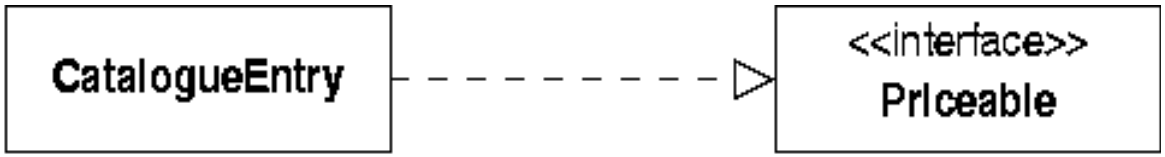
- broad sense (sensu lato)
  - the public interface (= all public methods) of a class
- narrow sense (sensu stricto): Java language feature
  = list of methods: "acts as a", "has an implementation for"
  - ...able: Runnable, Cloneable, Iterable, Observable, ...
  - ...Listener: ItemListener, ActionListener, ... (callback)
  - ultimate separation between declaration (= interface) and implementation (= in a class)
  - java subtype definition
    - can inherit from another interface (f.i. List -> Collection)
  - no instances, no constructors, no instance fields
  - can have static fields and static methods
  - all methods are abstract (= no implementation)
    - except for default methods implementations (since Java 8)
  - multiple implementations possible
    - a class can implement more than one interface
    - an interface can be implemented by more than one class

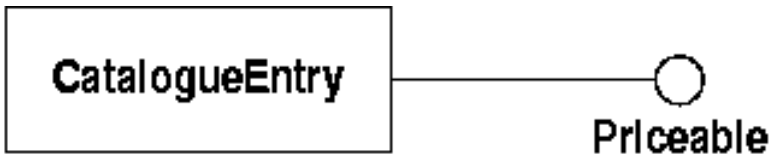
# UML class diagram & interface

- Interface = "named set of operations"
  - two notations:
    - «interface» stereotype notation
    - Little circle



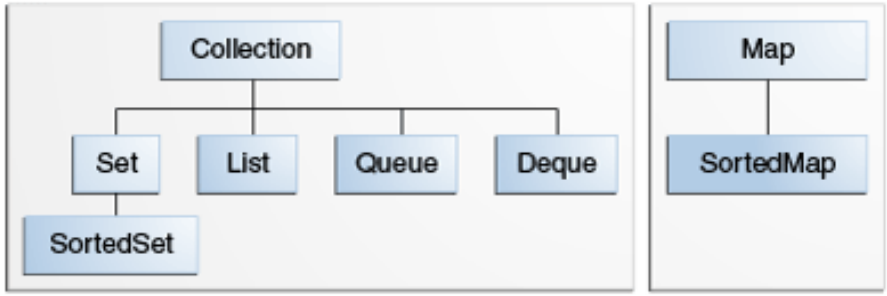

- Realization
  - A class that implements ("realizes") an interface
  - two notations:



# Examples of interfaces

- StudentBehavior and BePolite interface
  - Class demo
- In the Java API
  - Icon interface
    - JOptionPane.showMessageDialog() method
  - Sorting elements (String class + class demo: compare countries)
    - Comparable interface
    - Comparator interface
  - Collection hierarchie (see part 4)
- Design patterns (see part 7)



# Class demo

# Icon interface

```
public interface Icon
{
    /**
     * Draw the icon at the specified location.  Icon implementations
     * may use the Component argument to get properties useful for
     * painting, e.g. the foreground or background color.
     */
    void paintIcon(Component c, Graphics g, int x, int y);

    /**
     * Returns the icon's width.
     *
     * @return an int specifying the fixed width of the icon.
     */
    int getIconWidth();

    /**
     * Returns the icon's height.
     *
     * @return an int specifying the fixed height of the icon.
     */
    int getIconHeight();
}

public static void showMessageDialog(…, Icon icon) …
```

# Comparable & Comparator interface

Natural ordering (1 argument, compare with “this”)

```
public interface Comparable<T> {

    public int compareTo(T o);

}
```

External ordering (2 arguments, compare both arguments of same type T)

```
public interface Comparator<T> {

    public int compare(T o1, T o2);

}
```

# Interface implementation

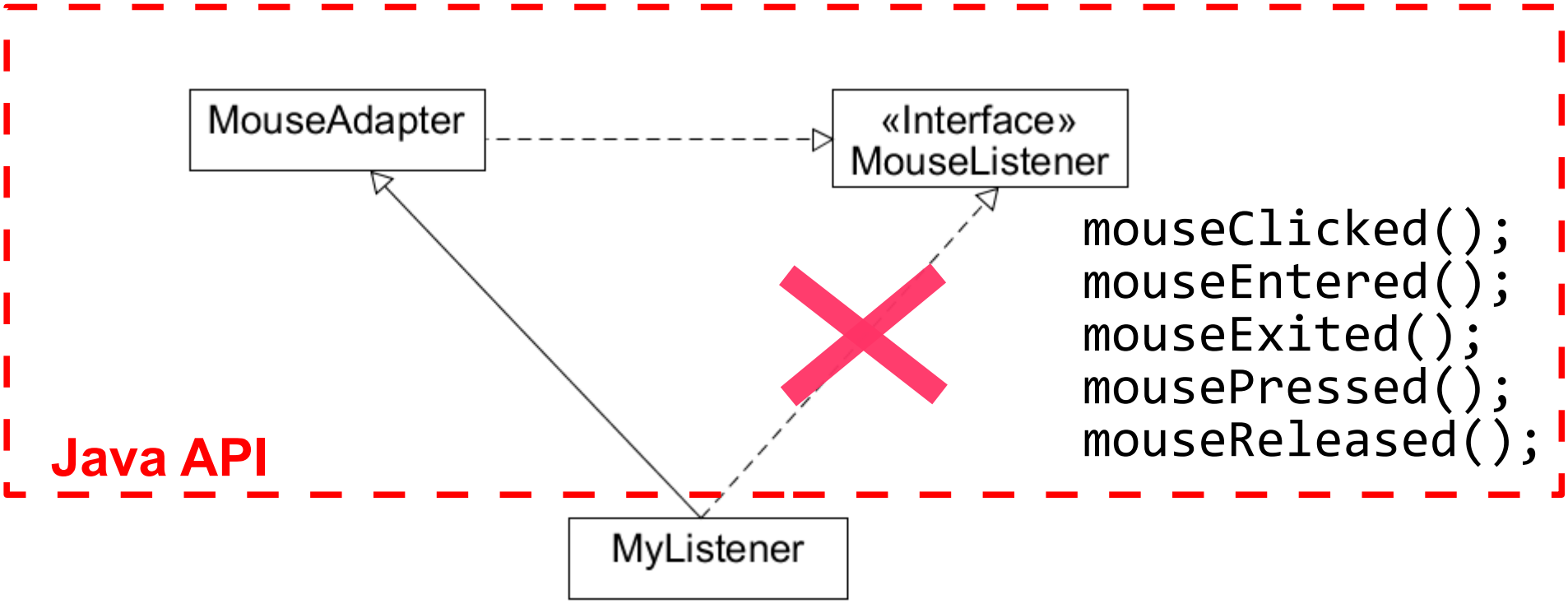
- Interface can be implemented by
  - a class
  - an anonymous inner class
  - a lambda expression (@FunctionalInterface)
- Java alternative for multiple inheritance
  - Object "is-a" (only one!) thing and "acts like" other things
    - "is-a": inheritance (extends)
    - "acts like": interface (implements)
- Subtype definition
  - Static type vs. Dynamic type (see: inheritance)

# Type casting

- Sometimes necessary?
- Be aware of a ClassCastException
- Use concrete class instead of interface?

```
List<String> list = new LinkedList<>();
String s1 = list.getFirst();
String s2 = ((LinkedList)list).getFirst();
list = new ArrayList<>();
String s3 = ((LinkedList)list).getFirst();
```

# Listener & Adapter (1)



# Listener & Adapter (2)

```
public abstract interface MouseListener extends EventListener {
    public abstract void mouseClicked (MouseEvent e);
    public abstract void mouseEntered (MouseEvent e);
    public abstract void mouseExited (MouseEvent e);
    public abstract void mousePressed (MouseEvent e);
    public abstract void mouseReleased (MouseEvent e);
}

class MyListener implements MouseListener {
    public void mouseEntered (MouseEvent e) {
        e.getComponent().setCursor(
                new Cursor(Cursor.HAND_CURSOR)
        );
    }
    public void mouseExited (MouseEvent e) {
        e.getComponent().setCursor(
                new Cursor(Cursor.DEFAULT_CURSOR)
        );
    }
}
```

# Listener & Adapter (3)

```
public class MouseAdapter implements MouseListener  {
    public void mouseClicked (MouseEvent e) {}   // empty
    public void mouseEntered (MouseEvent e) {}   // empty
    public void mouseExited (MouseEvent e) {}    // empty
    public void mousePressed (MouseEvent e) {}   // empty
    public void mouseReleased (MouseEvent e) {}  // empty
}

class MyListener extends MouseAdapter {

    @Override
    public void mouseEntered (MouseEvent e) {
        e.getComponent().setCursor(new Cursor.HAND_CURSOR));
    }

    @Override
    public void mouseExited (MouseEvent e) {
        e.getComponent().setCursor(new Cursor.DEFAULT_CURSOR));
    }
}
```

# Abstract class vs. Interface

- Abstract class
  - fields
  - concrete and abstract methods
  - constructors
- Interface
  - no fields
  - abstract methods
  - default methods (avoid them?)
  - No constructors
- Prefer interface above abstract class
  - more losely coupled: specification vs. implementation
    - but: default methods?
  - more lightweight type
  - the implementing class can still inherit from other class

# Good design principle:

## *Program to an interface*

- Decouple declaration from implementation
  - "What" vs. "How"
- Information hiding or Encapsulation
  - Do not expose the internals of your implementation
- Defer choice of actual class

- Criteria for designing a good interface (see later)
  - **Cohesion**: implements a single abstraction
  - **Completeness**: provides all operations necessary
  - **Convenience**: makes common tasks simple
  - **Clarity**: do not confuse your programmers
  - **Consistency:** keep the level of abstaction