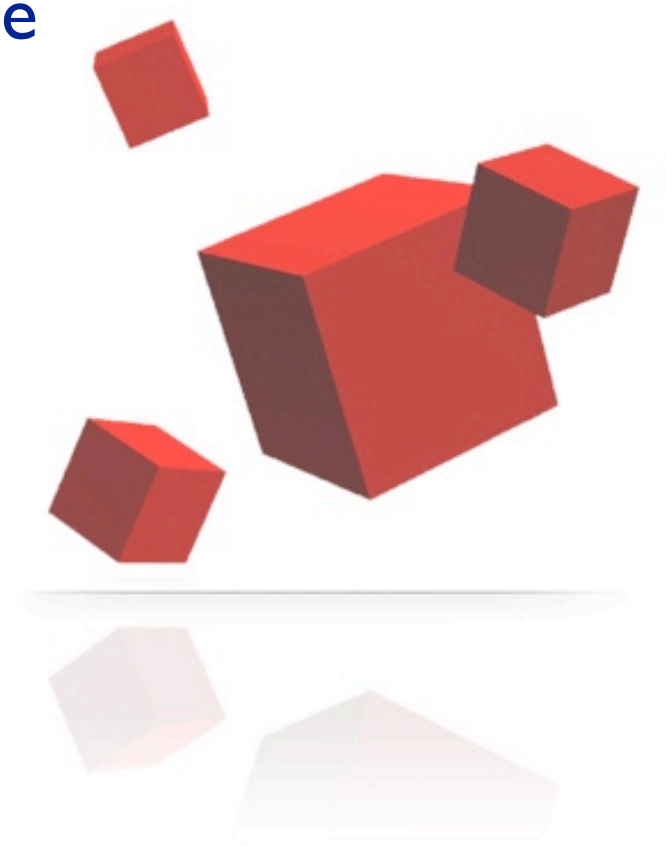
Classbox/J: Controlling the Scope of Change in Java

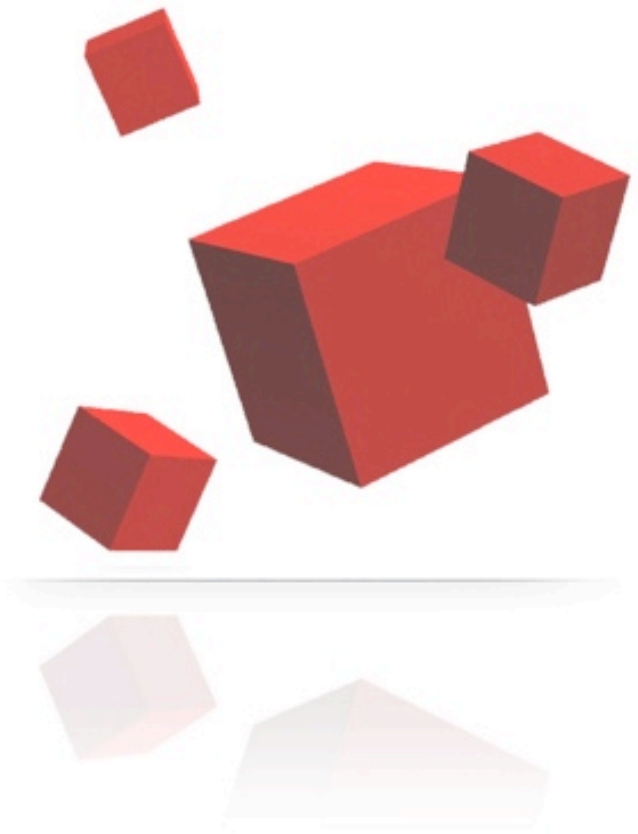
Alexandre Bergel,
Stéphane Ducasse and
Oscar Nierstrasz

bergel@iam.unibe.ch

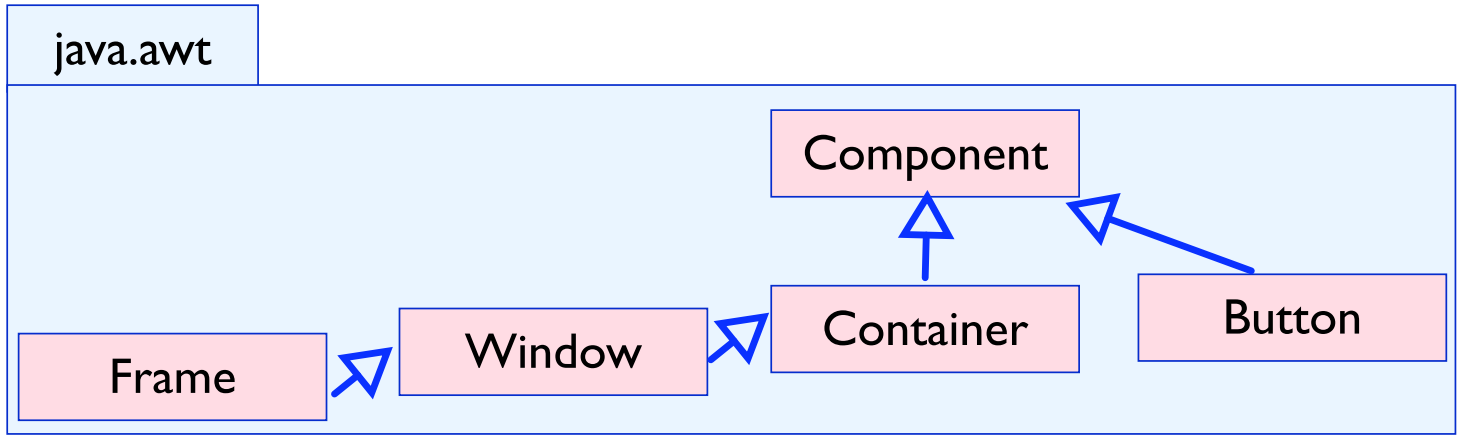


Outline

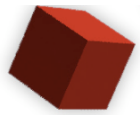
1. AWT and Swing Anomalies
2. Classbox/J
3. Properties of Classboxes
4. Swing as a Classbox
5. Implementation



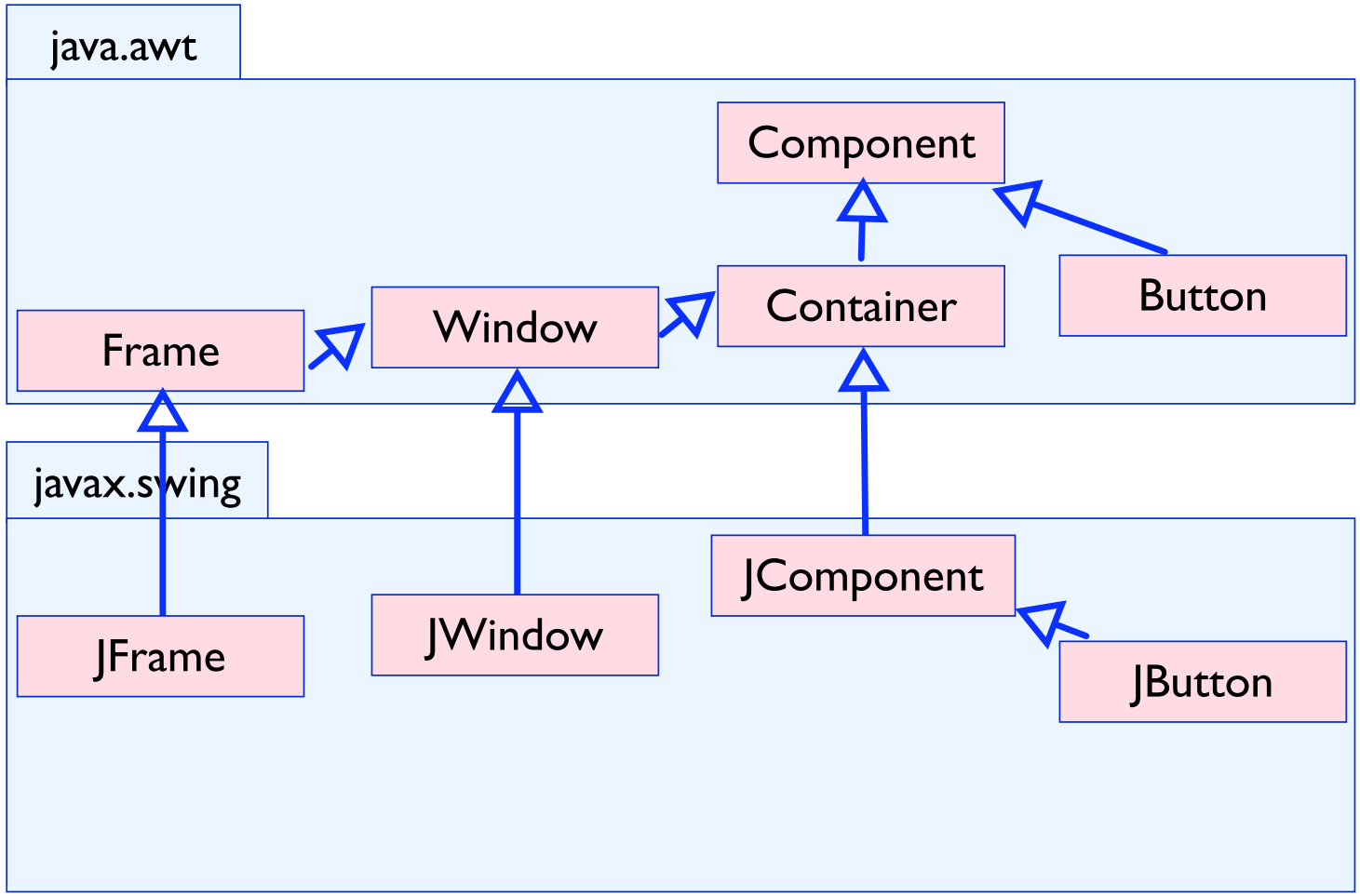
Presentation of AWT

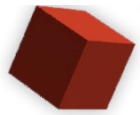


- In the AWT framework:
  - Widgets are components (i.e., inherit from Component)
  - A frame is a window (Frame is a subclass of Window)

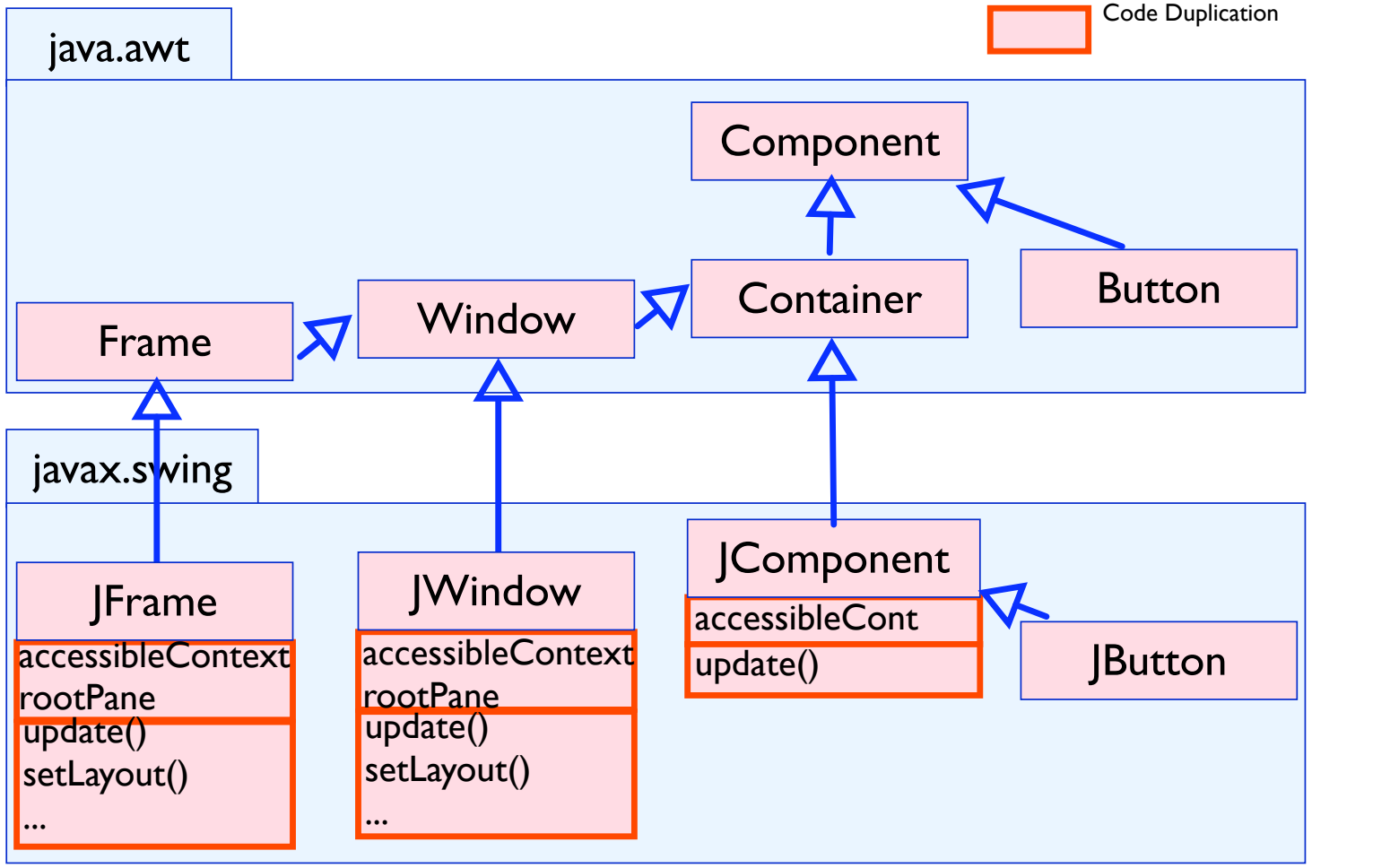


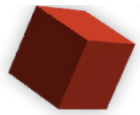
Problem: Broken Inheritance in Swing





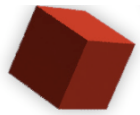
Problem: Code Duplication





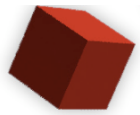
Problem: Explicit Type Checks and Casts

```
public class Container extends Component {
  Component components[] = new Component [0];
  public Component add (Component comp) {...}
}

public class JComponent extends Container {
  public void paintChildren (Graphics g) {
    for (; i>=0 ; i--) {
      Component comp = getComponent (i);
      isJComponent = (comp instanceof JComponent);
      ...
      ((JComponent) comp).getBounds();
    }
  }}
```

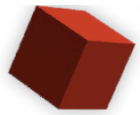
We need to Support Unanticipated Changes

- AWT couldn't be enhanced without risk of breaking existing code.
- Swing is, therefore, built on the top of AWT using subclassing.
- As a result, Swing is a big mess internally!
- We need a mechanism to support unanticipated changes.



Classbox/J

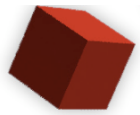
- Module system for Java allowing classes to be refined without breaking former clients.
- A classbox is like a package where:
  - a class defined or imported within a classbox p can be imported by another classbox (**transitive import**).
  - class members can be added or redefined on an imported class with the keyword **refine**.
  - a refined method can access its original behavior using the **original** keyword



Refining Classes (1 / 2)

A classbox widgetsCB

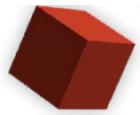
```
package widgetsCB;

public class Component {
   public void update() {this.paint();}
   public void paint () {/*Old code*/}
}

public class Button extends Component {
   ...
}
```

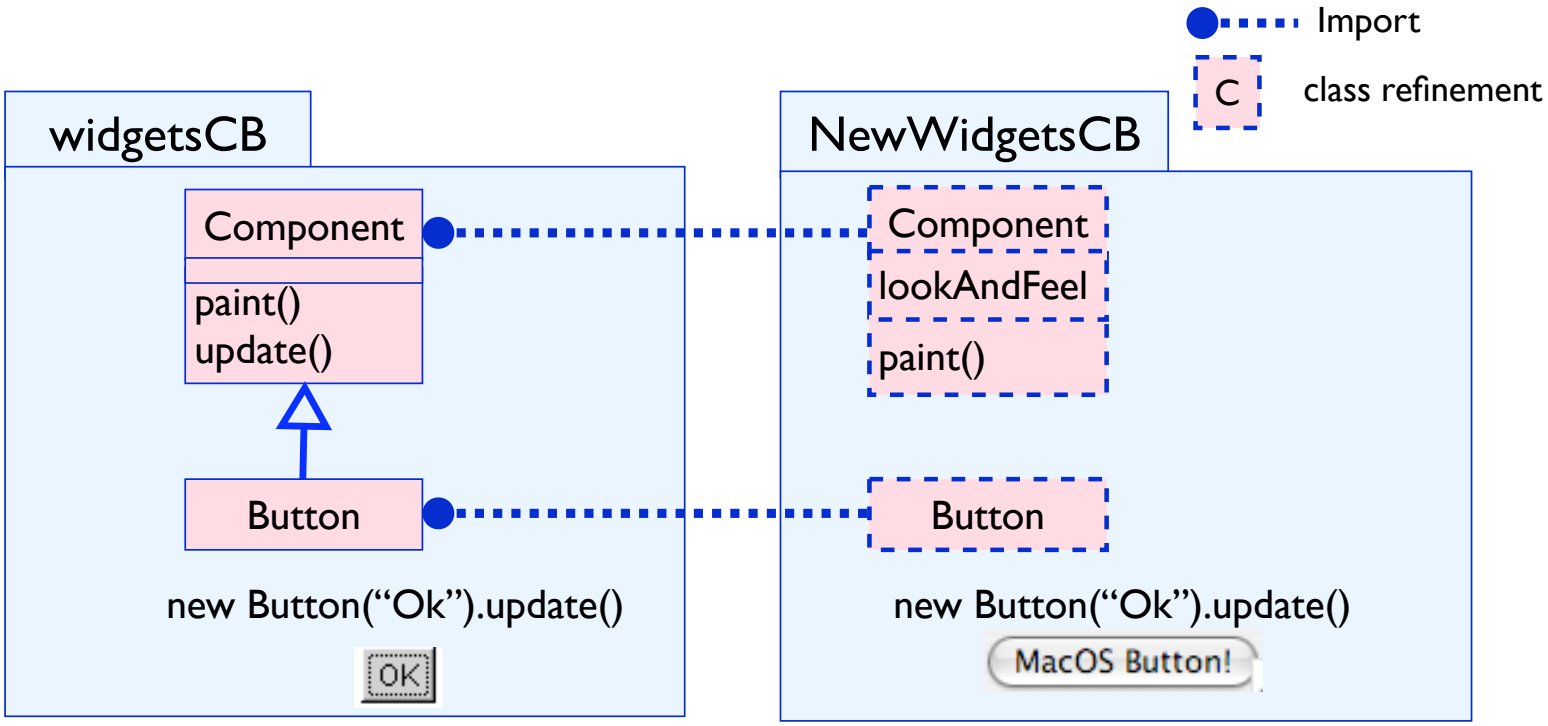
Refining Classes (2 / 2)

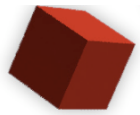
Widget enhancements defined in NewWidgetsCB:

```
package NewWidgetsCB;
import widgetsCB.Component;
import widgetsCB.Button;

refine Component {
   /* Variable addition */
   private ComponentUI lookAndFeel;

   /* Redefinition of paint() */
   public void paint() {
      /* Code that uses lookAndFeel*/ }
}
```

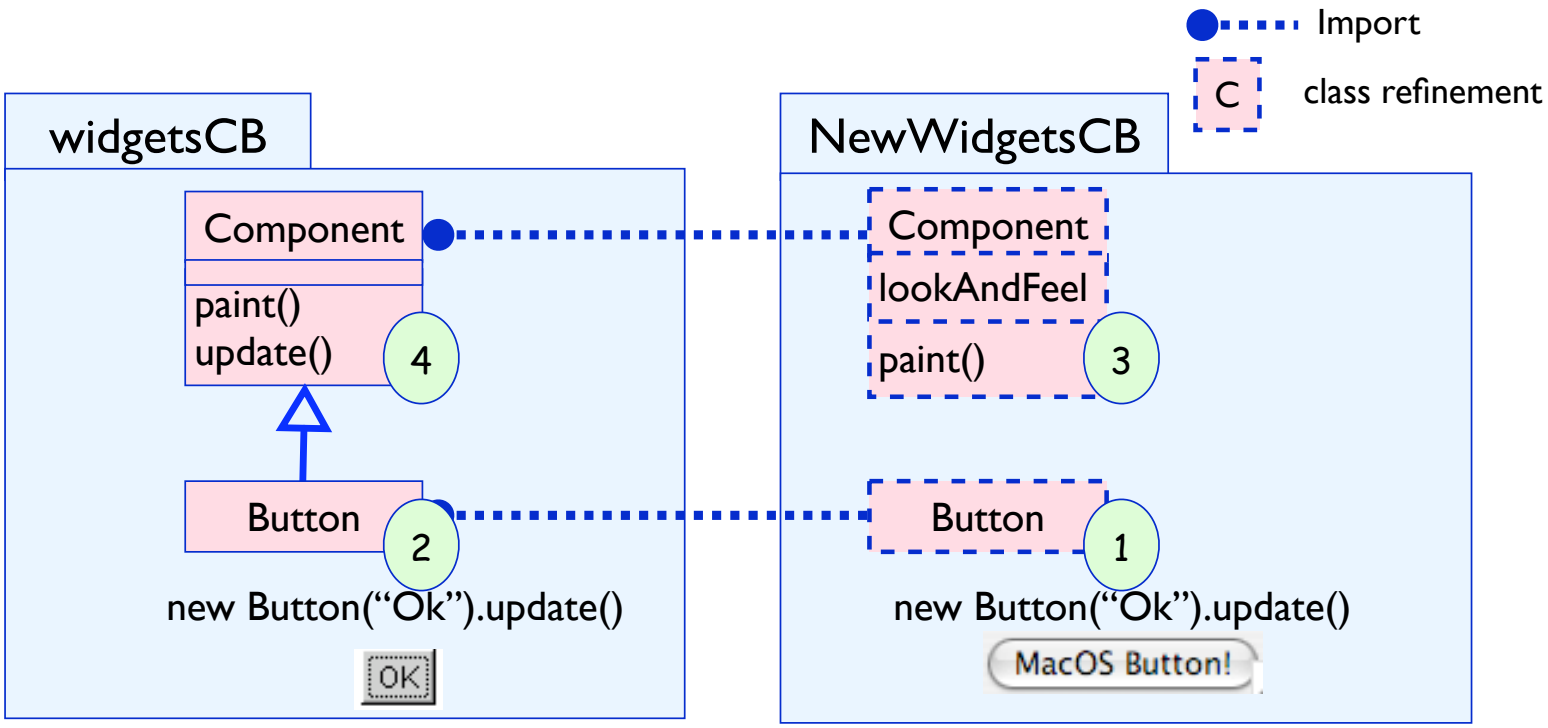



Multiple Versions of Classes

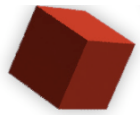




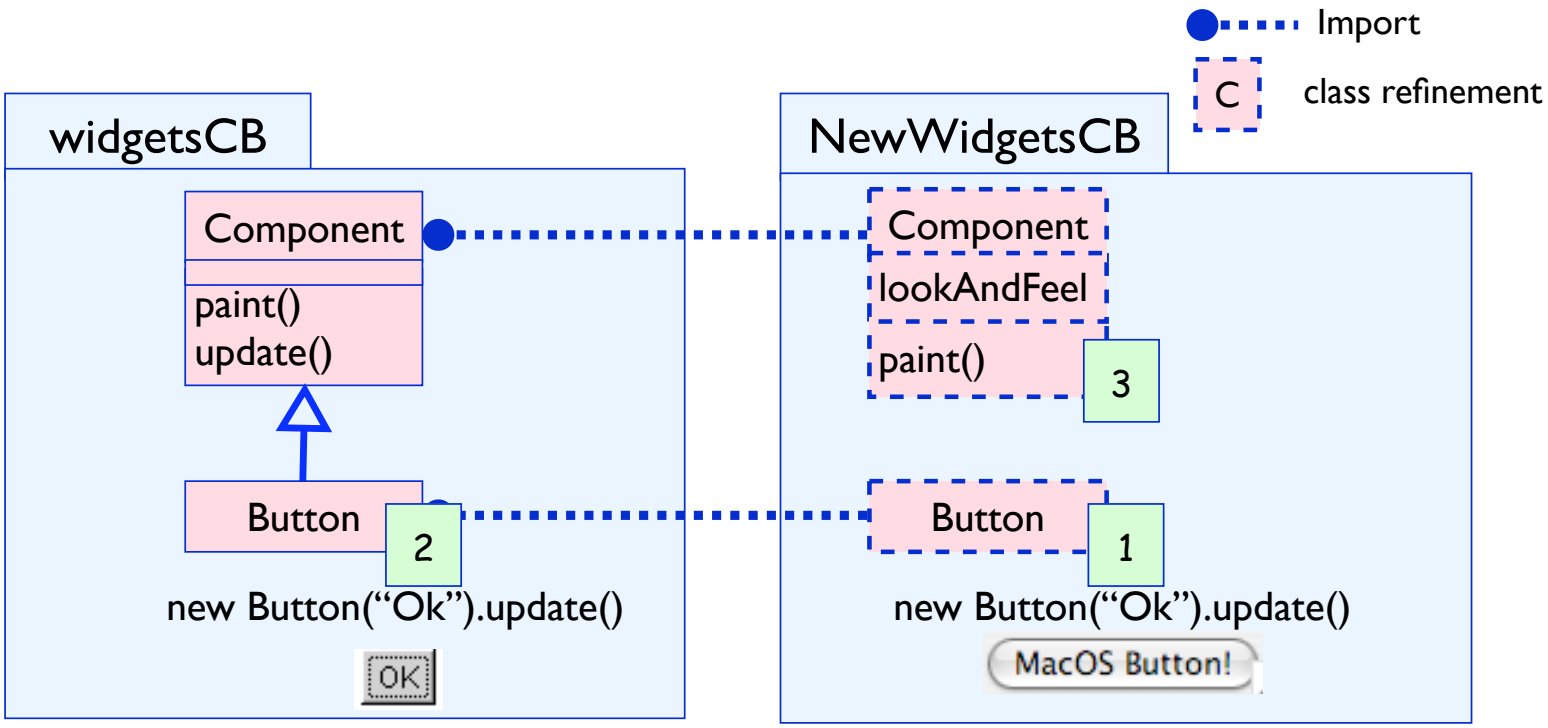
Import over Inheritance



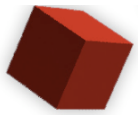
1 Lookup of the update() method triggered within enhWidgetsCB.



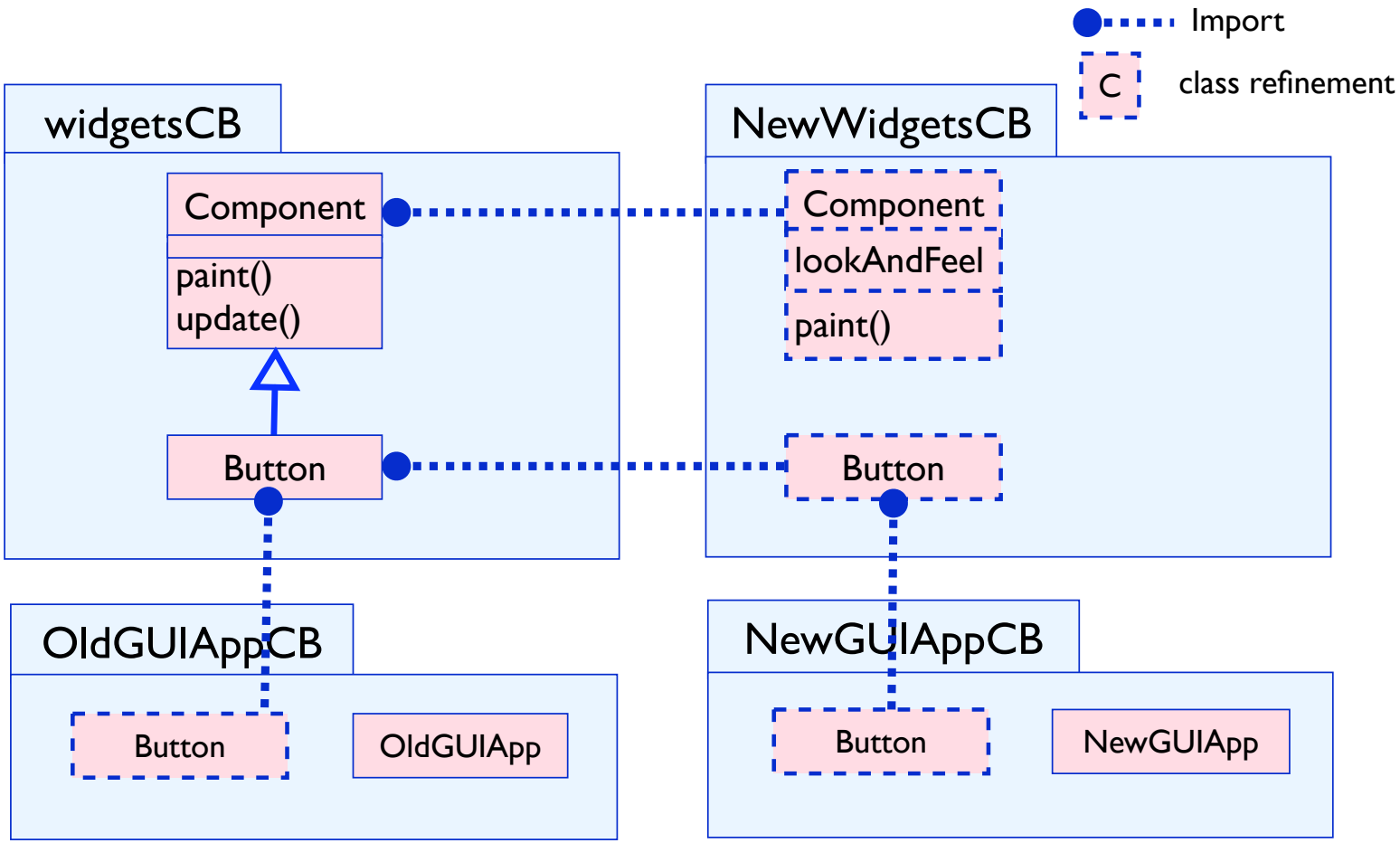
But update() calls paint()

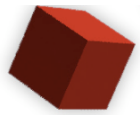


1 Lookup of the paint() method triggered within enhWidgetsCB



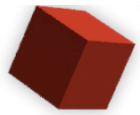
Old and New Clients at the Same Time



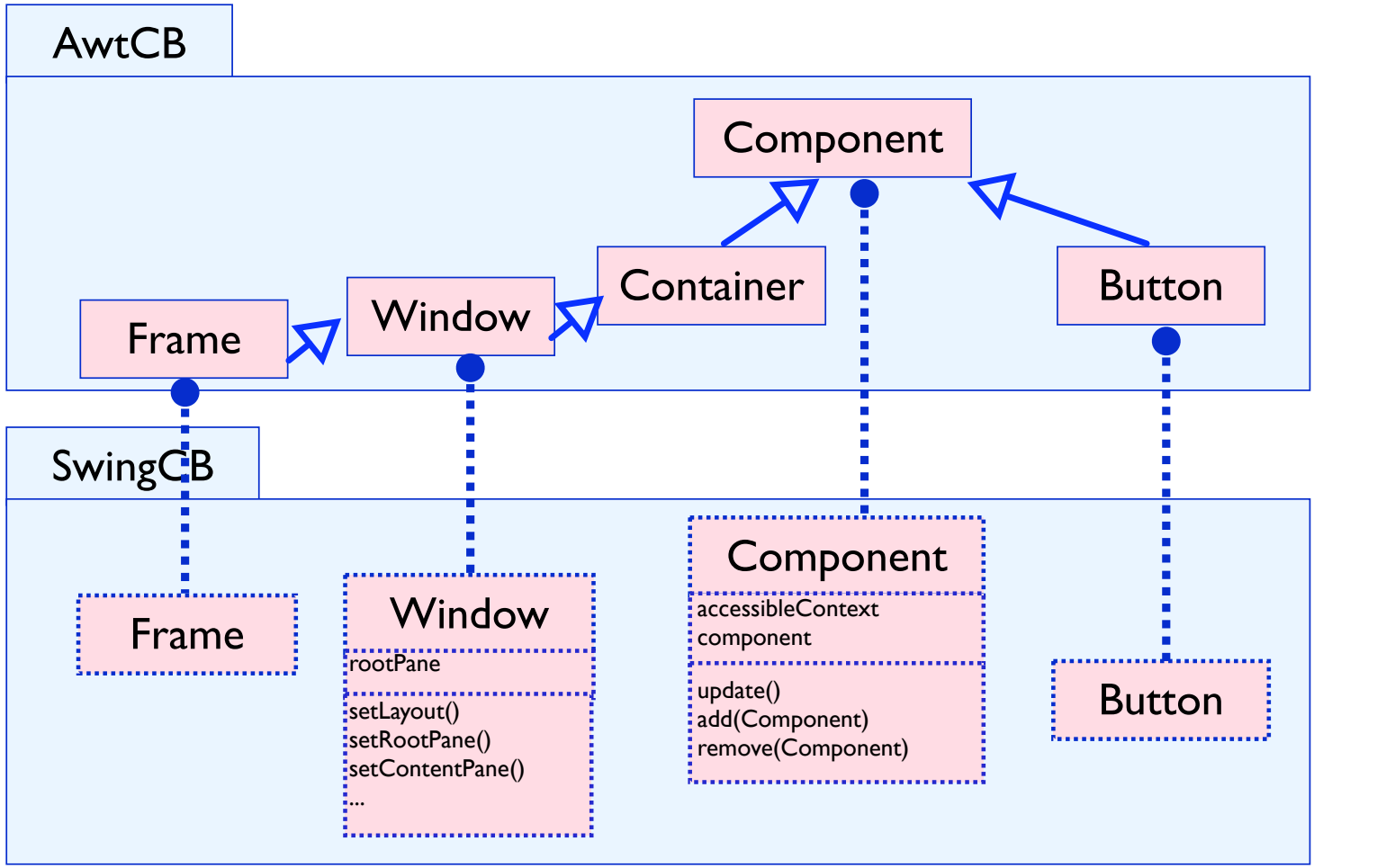


Properties of Classboxes

- Minimal extension of the Java syntax (transitive import, **refine** and **original** keywords).
- Refinements are confined to the classbox that define them and to classboxes that import refined classes.
- Method redefinitions have precedence over previous definitions.
- Classes can be refined without risk of breaking former clients.

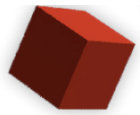


Swing Refactored as a Classbox



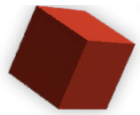
Swing Refactoring

- 6500 lines of code refactored over 4 classes.
- Inheritance defined in AwtCB is fully preserved in SwingCB:
  - In SwingCB, every widget is a component (i.e., inherits from the extended AWT Component).
  - The property "a frame is a window" is true in SwingCB.
- Removed duplicated code: the refined Frame is 29 % smaller than the original JFrame.
- Explicit type checks like *obj instanceof JComponent* and *(JComponent)obj* are avoided.



Naive Implementation

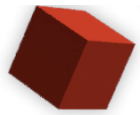
- Based on source code manipulation.
- The method call stack is introspected to determine the right version of a method to be triggered.
- No cost for method additions, however slowdown of 1000 times when calling a redefined method.
- However, much better results were obtained in Smalltalk. 5 byte-codes are added to redefined methods (see our previous work).



Method Call Stack Introspected

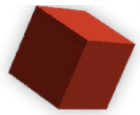
NewWidgetsCB and WidgetsCB define the paint method:

```
package WidgetsCB;
public class Component {
 public void paint() {

   if (ClassboxInfo.methodVisible (
     "NewWidgetsCB", "Component", "paint")){
        //Enhanced paint
   }

   if (ClassboxInfo.methodVisible (
     "WidgetsCB", "Component", "paint")){
        //Original paint
   }}}
```

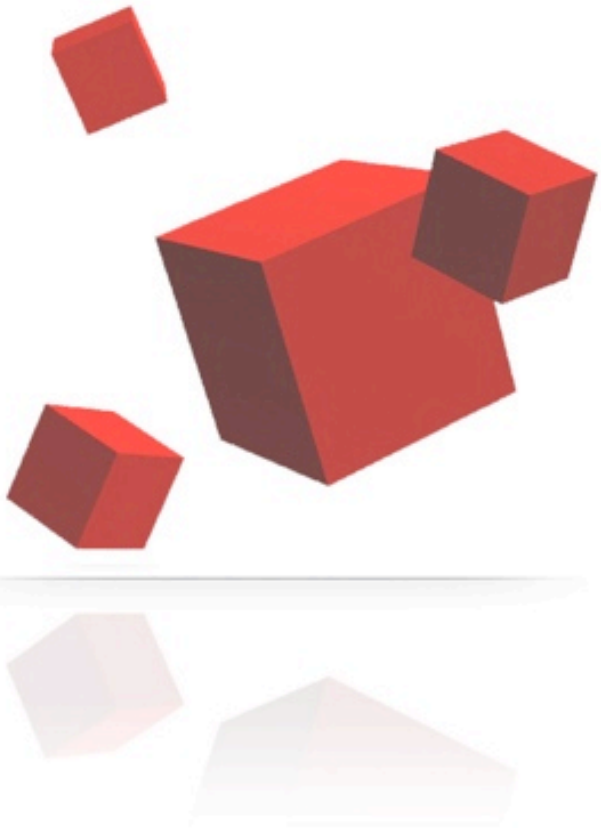



Conclusion

- Classboxes delimit visibility of a change and avoid impacting clients that should not be affected.
- Java is extended with two new keywords and transitive import.
- Large case study showing how classboxes can be more powerful than inheritance to support unanticipated changes.
- Performance could be improved by modifying the VM.



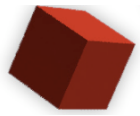
We need an alternative to inheritance to support unanticipated changes!



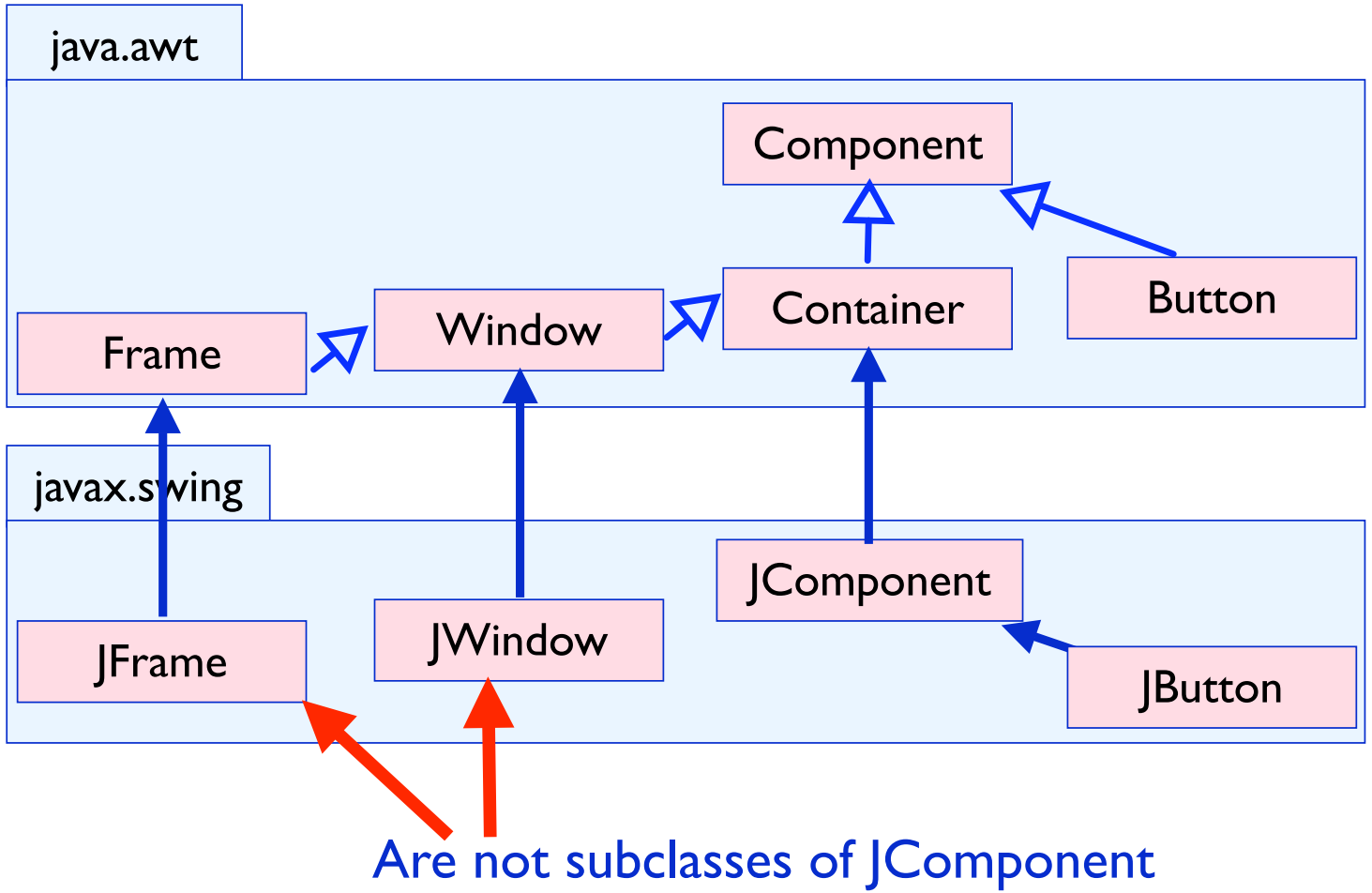
Alexandre Bergel:
bergel@iam.unibe.ch

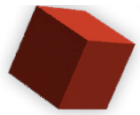
google “classboxes”

END

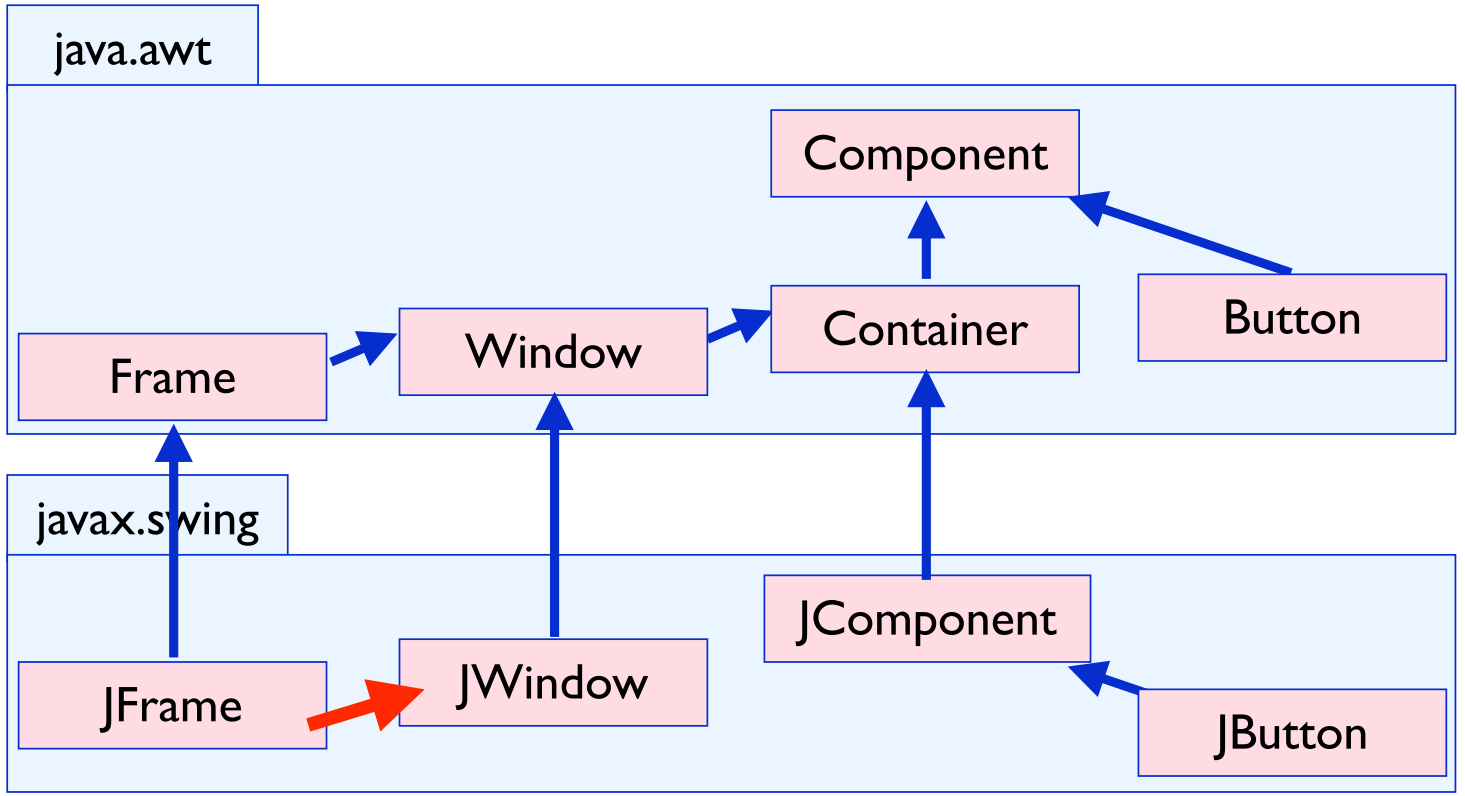


A JWidget is not necessary a JComponent

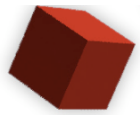




A JFrame is not a JWindow



Missing inheritance link between JFrame and JWindow



AWT and Swing Anomalies

- Features defined in JWindow are duplicated in JFrame (half of JWindow code is in JFrame).
- The Swing design breaks the AWT inheritance relation:
  - AWT: a Window is a Component
  - Swing: a JWindow is **not** a JComponent
- Need of explicit type checks and casts in Swing:
  - For instance a JWindow needs to check if its elements are issued from Swing or not before rendering them
  - 82 type checks (instanceof) and 151 cast to (JComponent)