Examples from Swing

Examples from the Java API

Swing

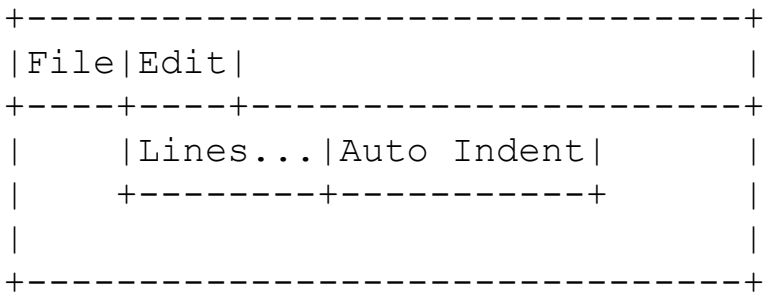
A big part of the Java API (the class library which comes with your installation of Java) is called Swing and contains classes and stuff for creating graphical user interfaces (windows, buttons, labels, menus etc).

We'll take a short look at some Swing classes and how they use inheritance to describe their hierarchical relations.

A window with a Menu

Let’s say we have a small window application with one menu bar and two menus File and Edit.

File only has one menu item, “Quit” and Edit has one menu item “Lines” which has a submenu with one item “Auto indent”.



Let’s look at the Swing classes for this application

JFrame - a class rJepresenting a window - the root container for all components in our window application

JMenuBar - a class representing the menu bar where menus show up

JMenuItem - a class representing an item in a menu

JMenu - a class representing a whole menu

We can add a menu bar to the JFrame

The class hierarchy for JFrame:

- java.lang.Object
  - java.awt.Component
    - java.awt.Container
      - java.awt.Window
        - java.awt.Frame
          - javax.swing.JFrame

So, all JFrames are by inheritance also of type Container.

Container has a method add(Component, Object) so we can use that also with JFrame objects. We can add, for instance a JMenuBar (if it extends Component!)

Why we can add a JMenuBar to a JFrame

- java.lang.Object
  - java.awt.Component
    - java.awt.Container
      - javax.swing.JComponent
        - javax.swing.JMenuBar

So, JMenuBar is a Component! Fine, then we can do this:

```
JFrame frame = new JFrame("Frame with menu");
JMenuBar menuBar=new JMenuBar();
frame.add(menuBar, BorderLayout.NORTH); // add it to the top of the window
```

We can add a menu to a menu bar

- java.lang.Object
  - java.awt.Component
    - java.awt.Container
      - javax.swing.JComponent
        - javax.swing.AbstractButton
          - javax.swing.JMenuItem
            - javax.swing.JMenu

JMenu is a Component, so we can add it to the JMenuBar which has a method add(JMenu c)

```
JFrame frame = new JFrame("Frame with menu");
JMenuBar menuBar=new JMenuBar();
JMenu menu = new JMenu("File");
JMenuItem quit = new JMenuItem("Quit");
menuBar.add(menu);
```

Did you notice that JMenu was a JMenuItem?

- javax.swing.JMenuItem
  - javax.swing.JMenu

JMenu has a method add(JMenuItem menuItem) which means that we can add a menu to a menu, as if it were a menu item.

JMenu extends JMenuItem, so a JMenu is a JMenuItem!

Therefore we can add a JMenu to a JMenu and get a submenu:

```
JMenu edit = new JMenu("Edit");
JMenu lines = new JMenu("Lines..."); // submenu!
JMenuItem autoIndent = new JMenuItem("Auto indent");

lines.add(autoIndent);  // add to the submenu
edit.add(lines);        // add the submenu to the edit menu!
```

The idea behind the Swing API

If everything is a component and (most things) also a container, then we can add components, to components.

A JFrame can have menu bars and panels, for instance. And a panel can have a button and a label. A panel can also have a panel which can have a....

They have created a hierarchy which is very flexible and describes the hierarchical nature of GUIs.

Inheritance allows us to be very general

Did you think about the method add(Component, Object) in the Container class?

It allows us to add any object which is of a subclass to Component to the Container.

This allows us to add basically anything (which is a Component) to a Container. We can add text components, buttons, menus, panels, labels and a large variety of widgets to any container.

More code for the menu application

```
JFrame frame = new JFrame("Frame with menu");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setPreferredSize(new Dimension(480,200));
JMenuBar menuBar=new JMenuBar();
frame.add(menuBar, BorderLayout.NORTH);
JMenu menu = new JMenu("File");
JMenuItem quit = new JMenuItem("Quit");
JMenu edit = new JMenu("Edit");
JMenu lines = new JMenu("Lines...");
JMenuItem autoIndent = new JMenuItem("Auto indent");
lines.add(autoIndent);
edit.add(lines);
menu.add(quit);
menuBar.add(menu);
menuBar.add(edit);
//Display the window.
frame.pack();
frame.setVisible(true);
```

https://github.com/progund/inheritance/blob/master/swing-examples/FrameWithSubmenu.java

Inheritance example using PrintStream

Consider the PrintStream class (System.out is of type PrintStream):

- java.lang.Object
  - java.io.OutputStream
    - java.io.FilterOutputStream
      - java.io.PrintStream

This also shows how inheritance was used in the Java API. It shows that there is a type called OutputStream (which is an abstract class). OutputStream has a subtype, FilterOutputStream which is a concrete class for writing to some OutputStream (passed to the constructor).

PrintStream is a specialization of a FilterOutputStream.

PrintStream

PrintStream is a subtype of FilterOutputStream. It adds convenience methods to the quite few methods in FilterOutputStream and OutputStream.

With a PrintStream we get access to many overloaded versions of print and println, for instance.

PrintStream also adds many convenient constructors to the more simplistic FilterOutputStream.

The various stream classes show us something

The various stream classes (and reader/writer classes) show us another use case for inheritance.

The hierarchy starts with an abstract class with only a few methods and constructors.

Further down the hierarchy, we get specialized streams for different purposes. They add specialized methods and convenience methods.

The further down we go, the more concrete and specialized a class (usually) becomes.