The Abstract Window Toolkit (AWT)

The AWT is in the package `java.awt.*`.

Main superclass `Component` represents widgets, controls, and windows:

- `Button`, `Label`, `Checkbox`, `Choice`, `List`, `Scrollbar`
- `TextComponent`
  - `TextArea`, `TextField`
- `Container`: parent window, parent component
  - `Panel`, `ScrollPane`
  - `Window`
    - `Frame`, `Dialog`

Component

Basic methods of `Component`:

- `Font getFont()`, `void setFont(Font f)`
- `Cursor getCursor()`, `void setCursor(Cursor c)`
- `Color getForeground()`,
  `void setForeground(Color c)`
- `Color getBackground()`,
  `void setBackground(Color c)`
- `Point getLocation()`
- `Dimension getSize()`
- `Rectangle getBounds()`
- `boolean contains(int x, int y)`,
  `boolean contains(Point p)`

Component (cont.)

- `boolean isEnabled()`, `void setEnabled(boolean b)`
- `boolean isVisible()`, `void setVisible(boolean b)`
- `boolean isShowing()`
- `String getName()`, `void setName(String s)`
- `Container getParent()`

Container

A `Container` object can contain `Component` objects.

Basic methods:

- `Component add(Component c)`
  `Component add(Component c, int i)`
  `Component add(String s, Component c)`
  `void add(Component c, Object o)`
  `void add(Component c, Object o, int i)`
- `Component getComponent(int i)`
  `Component[] getComponents()`
  `int getComponentCount()`
- `boolean isAncestorOf(Component c)`
- `void remove(Component c)`
  `void remove(int i)`
  `void removeAll()`
- methods inherited from `Component`.

Example

```
import java.awt.*;

public class CompDemo
{
  public static void main(String[] args) {
    Frame frame = new Frame("Components Demo");
    frame.setBackground(Color.white);
    Panel panel = new Panel();
    frame.add(panel);
    panel.add(new Label("Enter here:"));
    panel.add(new TextField(5));
    panel.add(new Label("Colour:"));
    Choice choice = new Choice();
    choice.add("Red");
    choice.add("Green");
    choice.add("Blue");
    panel.add(choice);
    panel.add(new Button("OK"));

    frame.pack();  // auto-size
    frame.show();
  }
}
```

Component and Container

Layout

How are the sizes and positions decided?

- Hardcoding: quirky but sensitive to changes
- Algorithmic layout: more overhead but easier to maintenance

Java prefers the latter and has the infrastructure for it.
How it works:

1. each component knows its correct size
2. each container is assigned an algorithm
3. algorithm queries children of a container for their correct sizes
4. algorithm assigns positions and sizes to children, using the correct sizes as *guidelines*
5. correct size of container based on layout of children

Layout Mechanism

1. `LayoutManager`, `LayoutManager2`: superclasses for layout algorithms and styles
2. `Component` object methods:
   - `Dimension getPreferredSize()`
     `Dimension getMinimumSize()`, `Dimension getMaximumSize()`
     the correct size
   - `void setBounds(...)`
     `void setLocation(...)`, `void setSize(...)`
     set sizes and positions
   - `void invalidate()`
     marks this component and all ancestors as "invalid": need re-layout
3. `Container` object methods:
   - `void setLayout(LayoutManager m)`
     `LayoutManager getLayout()`
     the layout manager object used for this container
   - `void validate()`
     re-layout children (if invalid)