Design Patterns

Dr. James A. Bednar

jbednar@inf.ed.ac.uk

http://homepages.inf.ed.ac.uk/jbednar

Design Patterns

A design pattern is a standardized solution to a problem commonly encountered during object-oriented software development (Gamma et al. 1995).

A pattern is not a piece of reusable code, but an overall approach that has proven to be useful in several different systems already.

Contents of a Design Pattern

Design patterns usually include:

- A pattern name
- A statement of the problem solved by the pattern
- A description of the solution
- A list of advantages and liabilities

  (good and bad consequences)

Design Patterns and Large-Scale Development

For a large team, design patterns are useful in creating a shared vocabulary.

First, everyone agrees on a standard reference text
(or set of them).

Informal discussions, class naming, etc. can then use the pattern names.

Large groups can develop and name their own patterns.

Design Pattern Examples

Creational Patterns:

- E.g. Abstract Factory, Factory Method

Structural Patterns:

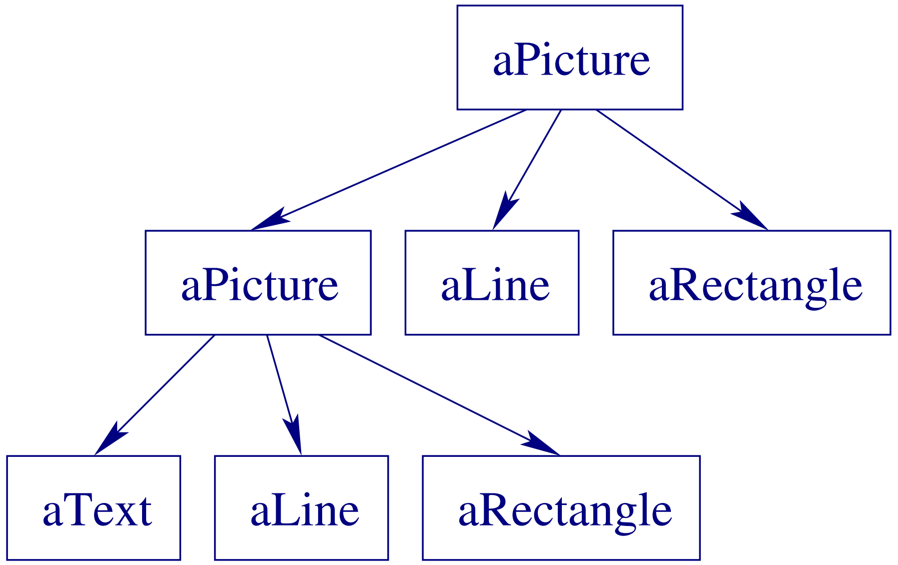
- Composite
- Proxy

Behavioral Patterns:

- E.g. Command, Visitor

These are from Gamma et al. (1995), but there are many other pattern collections.

Composite: Pattern



Composes objects into tree structures to represent part-whole hierarchies.

Lets clients treat individual objects and compositions of objects uniformly.

Composite: Problem

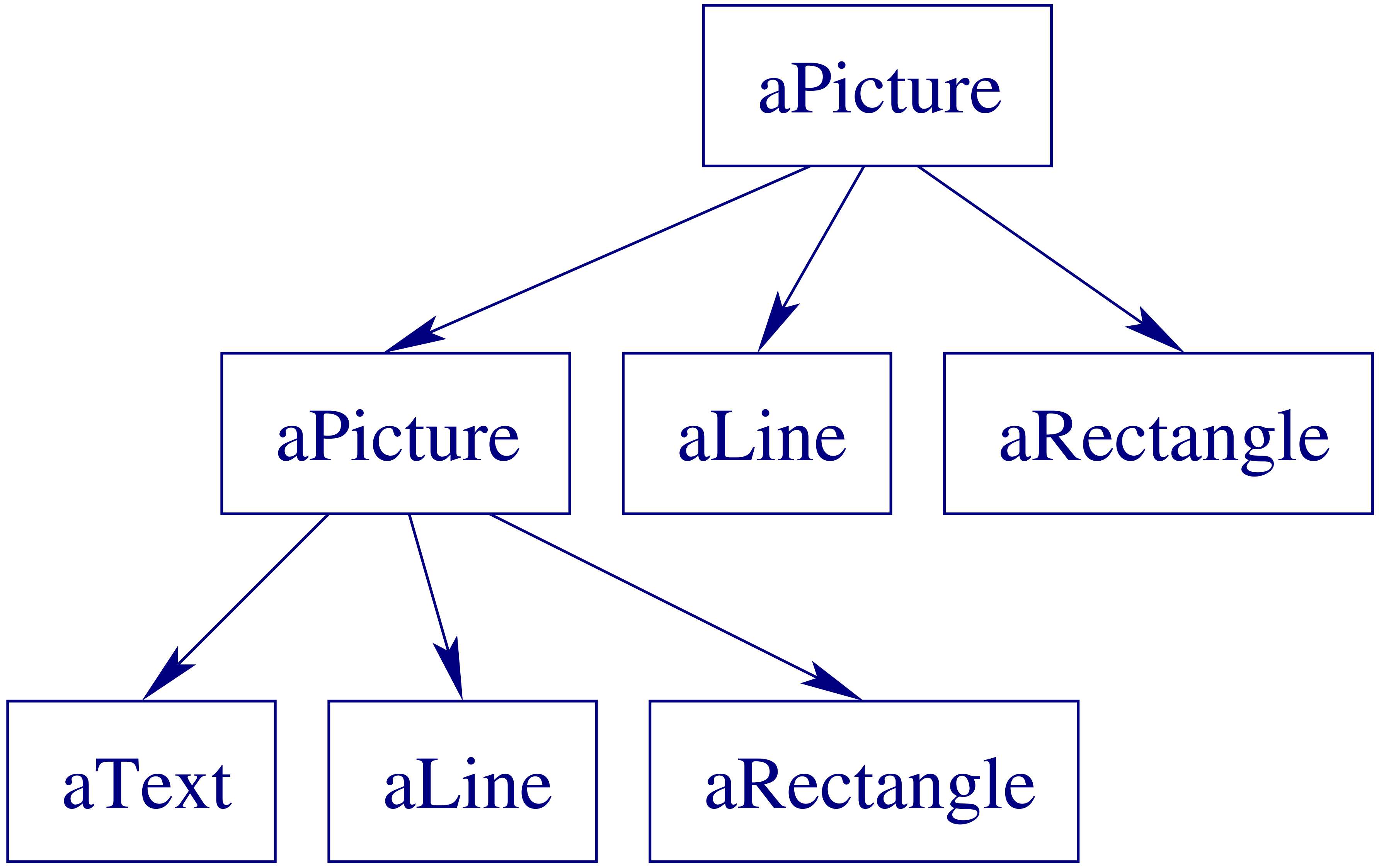
- User wants to be able to treat groups of things as a unit
- Surrounding code would get complex if it were always conditional on whether an object was a group or a primitive
- Want to support hierarchical containers of containers

Composite: Solution

Three classes:

- Component: Shared interface between all, some shared implementation
- Leaf: A primitive, implemented directly
- Composite: forall children Components, do operation

Composite: Example



Composite: Advantages

- Simple support for arbitrarily complex hierarchies
- Clients can be simple — don’t need to know about composition
- New Composite and Leaf classes can be introduced without changing Component

Composite: Liabilities

- Hard for client to predict/restrict what components might be encountered
- Hard to test that client works for all components
- Often need to define operations on Components that make sense only for some Component types, e.g. Composites

Summary

- Many other patterns available
- Design patterns help provide a library of solutions to common OO problems
- Usually low level, but act as a vocabulary for a large team
- Important to agree on definitions, apply consistently

Note: Due Dates

The web site has been updated with the due dates for assessed coursework:

1. Handed out 21 Jan 2005, due 07 Feb 2005

2. Handed out 04 Feb 2005, due 21 Feb 2005

3. Handed out 01 Mar 2005, due 15 Mar 2005

All three are weighted equally. In place of Assignment 3, MSc students will do a literature survey due on the same day as Assignment 3. There is a break after Assignment 2 to give undergraduate students time for their projects.

References

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley.