Welcome

Java Design Patterns

Prakash Badhe

prakash.badhe@vishwasoft.in

Copyright Notice

Making a Film ?

- Suppose you have got to make the film as Box Office Hit and earn profits…
- Which Director you will select ?
  - Subhash Ghai
  - David Dhavan
  - M. Bhandarkar
  - Shyam Benegal
  - Mani Ratnam

What is the story line ?…

- Bichade Hue Bhai,One becomes DON and other ? Obviously Police Inspector..
- A love story…?
- A story of a Boxer (Remake of Rocky Balboa?)
- Story of a successful Business man ?
- Dakoos vs Hero ?

Casting Couch ?

- Which Music Director ?
- Which Hero, Heroine and other casts to select ?
- How to market the film and how to make it hit ?

Standard Formula

- Obviously we will select the successful Director like ‘Priyadarshan’
- Heroes like SRK,Big B,….
- Will you select a new comer as a hero?
- Successful story lines…
- These things are already proven and hit so you just follow them and relax…

Learning Methods

- How do people learn the chess game and become Grand Masters ?
- How to develop good software? How to become a good programmer ?
- Successful solutions to many areas of human endeavor are deeply rooted in following successful people for their successful and proven tasks. These are some kinds of Patterns…
  – important goal of education is transmitting *patterns of learning* from generation to generation

Becoming a Chess Master

- *Learn the rules and physical requirements*
  - e.g. pieces, legal movements, chess board geometry and orientation.
- *Learn the principles*
  - e.g. relative value of certain pieces, strategic values of center squares, power of a threat etc.
- To become a master, one must *study the games of other masters*
- These games contain patterns that must be understood, memorized and applied repeatedly
- There are hundreds of such patterns

Becoming a Software Design Master

- *First learn the rules*
  - e.g. the algorithms, data structures and languages of software
- *Then learn the principles*
  - e.g. principles that govern different programming paradigms i.e.*structured, modular, object-oriented, etc*
- To truly master software design, one must *study the design of other masters*
  - these designs contain *patterns* that must be understood, memorized and applied repeatedly
  - There are hundreds of these patterns

What is a Pattern ?

Each pattern describes a problem, which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice

--Christopher Alexander,

“The Timeless Way of Building”, 1979

Alexander's View of a Pattern

- Three part rule that expresses a relation between a certain context, a problem and a solution.
- *Element of the world* – a relationship between
  - – a context
  - – a system of forces that occur repeatedly in the context
  - – a spatial configuration which allow forces to resolve themselves

Alexander's View of a Pattern - II

- The "*thing – process*" dualism
  - – a thing that happens in the world
  - – a process (rule) which will generate that thing
- *Element of language* – an instruction
  - – describes how the spatial configuration can be repeatedly used
  - – to resolve the given system of forces
  - – wherever the context makes it relevant

Why Use Patterns ?

Patterns help you learn from other's successes, instead of your own failures

Mark Johnson (cited by B. Eckel)

- Patterns provide an additional layer of abstraction
  - *separate things that change from things that stay the same*
  - distilling out common factors between a family of similar problems
  - similar to design
  - most general and flexible solution
- Insightful and clever way to solve a particular *class of problems*

Design Patterns

- Design patterns represent solutions to problems that arise when developing software within a particular context

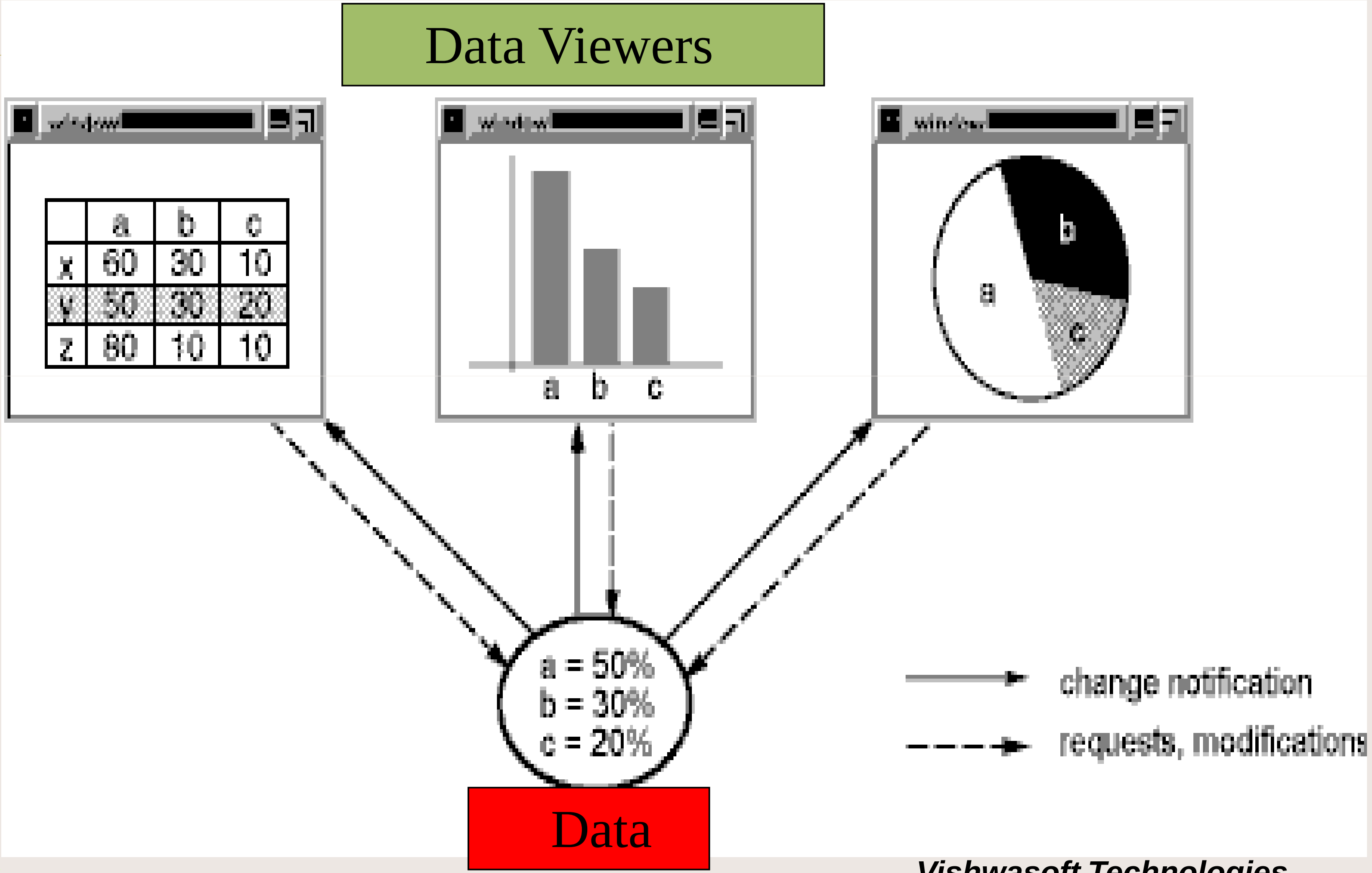
  Pattern = **Problem/Solution** pair in **Context**

- Capture static and dynamic structure and collaboration among key participants in software designs
  - key participant – major abstraction that occur in a design problem
  - useful for articulating the ***how*** and ***why*** to solve *non-functional forces*.

Design patterns - II

- Facilitate reuse of successful software architectures and design
  - i.e. the "*design of masters*"… ;)

Data-Views Consistency..

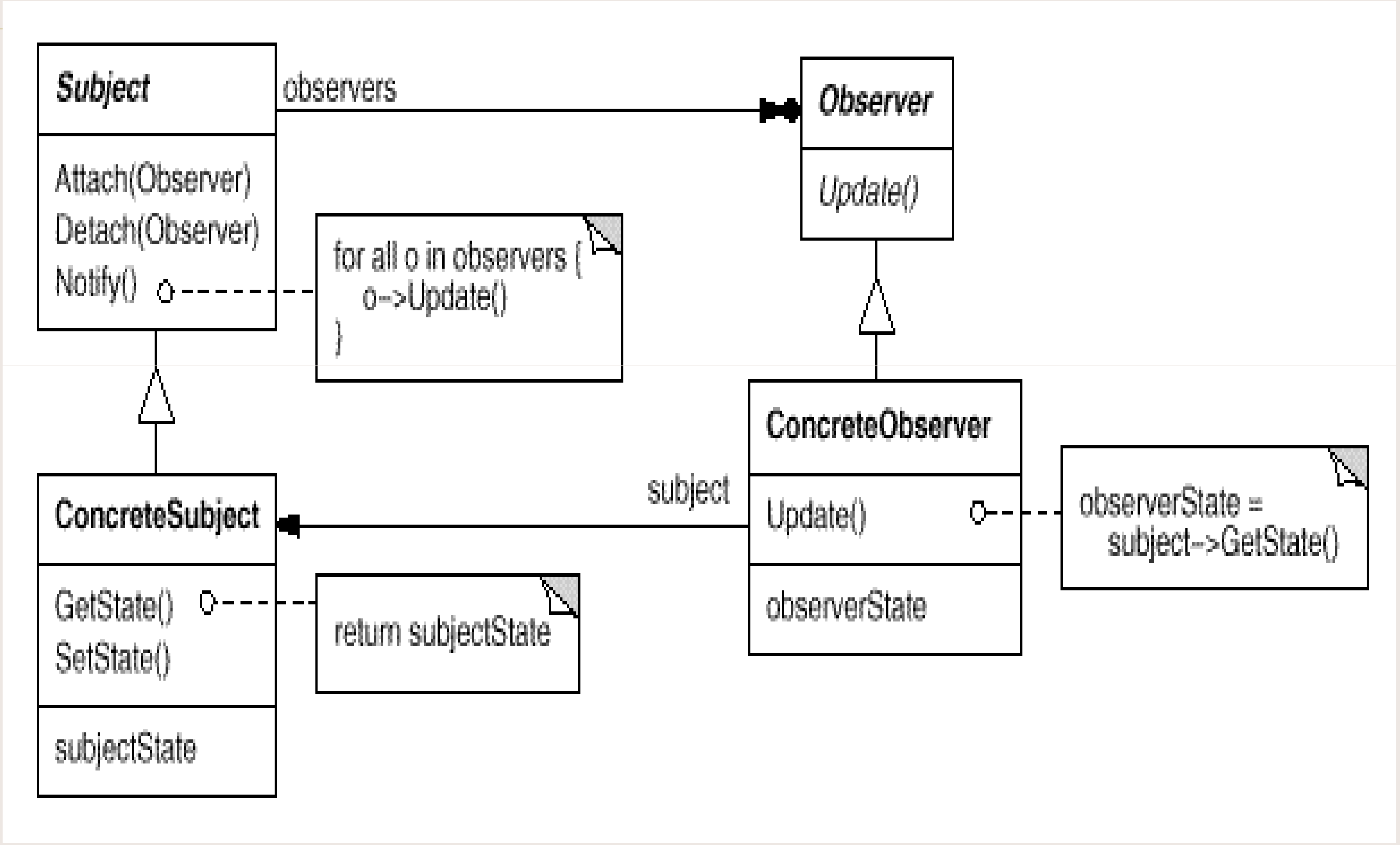


	a	b	c
x	60	30	10
y	50	30	20
z	80	10	10

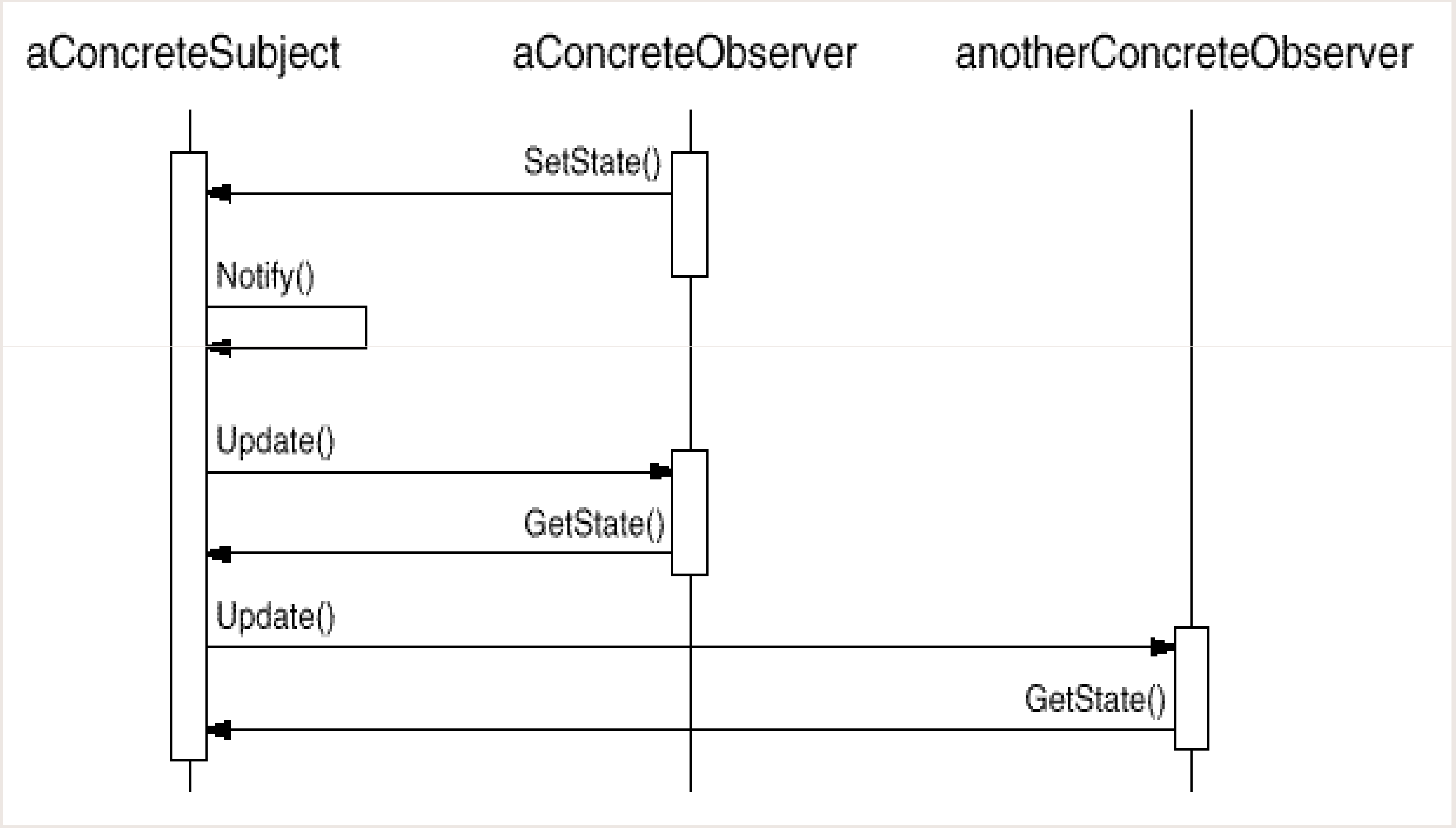
The Pattern and Solution

- Intent
  - Define a one-to-many dependency between objects so that when one object changes state, all its dependencies are notified and updated automatically
- Forces
  - There may be many viewers
  - Each viewer may react differently to the same notification
  - The data-source (subject) should be as decoupled as possible from the viewer
    - to allow viewers to change independently of the data.

Design Solution



Collaborators in the Solution Pattern



Vishwasoft Technologies

What Makes a Pattern ?

A pattern must...

- ...solve a problem
  - i.e. it must be useful
- ...have a context
  - it must describe where the solution can be used
- ...recur
  - must be relevant in other situations

... teach

- provide sufficient understanding to tailor the solution

- ... have a name
  - referred consistently

GANG of FOUR

- **Erich Gamma**
- **Richard helm**
- **Ralph Johnson**
- **John Vlissides**

GoF Form of a Design Pattern

Pattern name and classification

Intent
- what does pattern do

Also known as
- other known names of pattern (if any)

Motivation
- the design problem

Applicability
- situations where pattern can be applied

Structure
- a graphical representation of classes in the pattern

Participants
- the classes/objects participating and their responsibilities

Collaborations
- of the participants to carry out responsibilities

GoF Form of a Design Pattern (contd.)

Consequences
  trade-offs, concerns

Implementation
  hints, techniques

Sample code
  code fragment showing possible implementation

Known uses
  patterns found in real systems

Related patterns
  closely related patterns

Algorithmic Form of Patterns

```
IF    you find yourself in CONTEXT
      for example EXAMPLES,
      with PROBLEM,
      entailing FORCES
THEN  for some REASONS,
      apply DESIGN FORM AND/OR RULE
      to construct SOLUTION
      leading to NEW CONTEXT and
                 OTHER PATTERNS
```

Key Mechanisms in Design Patterns

Design Pattern Conventions

1. Name & Classification
2. Intent
3. Other Names
4. Motivation
5. Applicability
6. Structure
7. Participants
8. Collaborations
9. Consequences
10. Implementation
11. Known Uses
12. Related Patterns

Class vs. Interface Inheritance

- Class – defines an implementation
- Type – defines only the interface
  - the set of requests that an object can respond to
- Relation between Class and Type
  - the class implies the type
- On class, many types. Many classes, same type
- Class Inheritance
  - one implementation in terms of another
- Type Inheritance
  - when an object can be used in place of another

GoF Design Principle no. 1

Program to an interface, not an implementation

- Use interfaces to define common interfaces
  - and/or abstract classes in C++
- Declare variables to be instances of the abstract class
  - not instances of particular classes
- Use Creational patterns
  - to associate interfaces with implementations

Benefits

- Greatly *reduces the implementation dependencies*
  - Client objects remain unaware of the classes that **implement** the objects they use.
  - Clients know only about the abstract classes (or interfaces) that define the interface.

Class Inheritance vs. Composition

- Mechanisms of reuse
  - White-box vs. Black-box
- Class Inheritance
  - easy to use; easy to modify
    - implementation being reused;
  - language-supported
  - static bound ⇒ can't change at run-time;
  - mixture of physical data representation ⇒ breaks encapsulation
    - change in parent ⇒ change in subclass
- Object Composition
  - objects are accessed solely through their interfaces
    - no break of encapsulation
  - any object can be replaced by another at runtime
    - as long as they are the same type

Design Principle no. 2

Favor composition over class inheritance

- **Keeps classes focused on one task**
- **Inheritance and Composition Work Together!**
  - ideally no need to create new components to achieve reuse
  - this is rarely the case!
  - reuse by inheritance makes it easier to make new components
    - modifying old components
- Tendency to overuse inheritance as code-reuse technique
- Designs – more reusable by depending more on object composition

Classification of Design Patterns

- **Creational Patterns**
  - deal with initializing and configuring classes and objects
  - how am I going to create my objects?
- **Structural Patterns**
  - deal with decoupling the interface and implementation of classes and objects
  - how classes and objects are composed to build larger structures
- **Behavioral Patterns**
  - deal with dynamic interactions among societies of classes and objects
  - how to manage complex control flows (communications)

GOF- Design Pattern Catalog

		Purpose		
		Creational	Structural	Behavioral
Scope	Class	• Factory Method	• Adapter	• Interperter
	Object	• Abstract Factory • Builder • Prototype • Singleton	• Adapter • Bridge • Composite • Decorator • Facade • Flyweight • Proxy	• Chain of Responsibility • Command • Iterator • Mediator • Momento • Observer • State • Strategy • Vistor

Class and Object patterns

- class patterns describe how inheritance can be used to provide more useful program interfaces.
- Object patterns, on the other hand, describe how objects can be composed into larger structures using object composition, or the inclusion of objects within other objects

Benefits of Design Patterns

- Inspiration
  - *patterns don't provide solutions, they **inspire** solutions*
  - Patterns **explicitly** capture expert knowledge and design tradeoffs and make this expertise widely available
  - ease the transition to object-oriented technology
- Patterns improve developer communication
  - pattern names form a **vocabulary**
- Help document the architecture of a system
  - enhance understanding
- Enable large-scale reuse of software architectures

Design Pattern Consequences

- Patterns are abstract and hence do not lead to direct code reuse
- Patterns help to apply a solution in a particular context.
- Patterns are deceptively simple
- Patterns standardize the behaviors
- Development Teams may suffer from patterns overload
- Integrating patterns into a software development process is a human-intensive activity