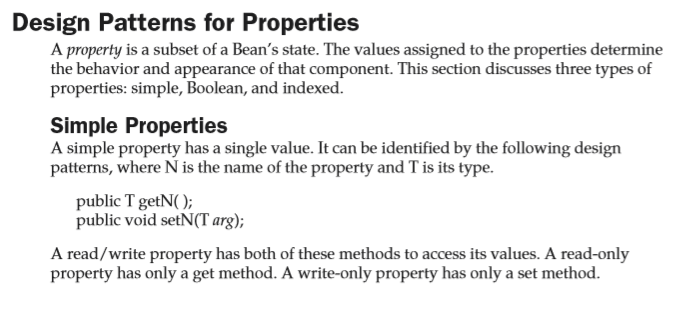
**Introspection**

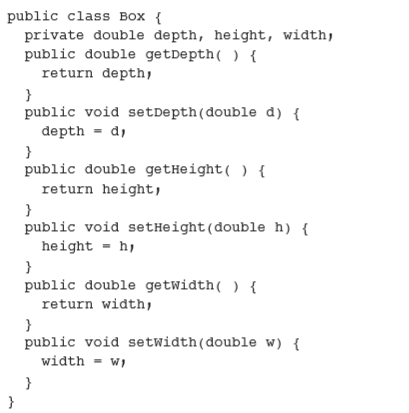
Introspection is the ability to obtain information about the properties, events and methods of Bean.

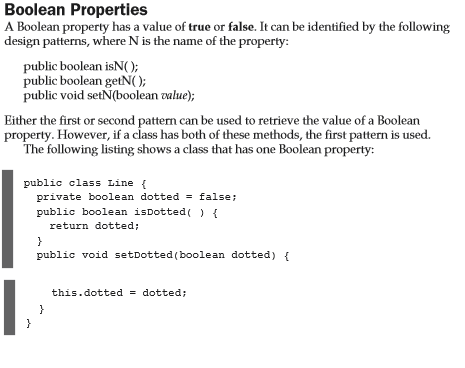
**Two ways**

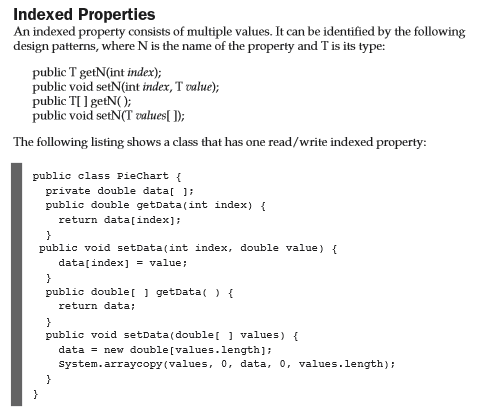
1. Naming patterns for Properties, Events and Methods for Bean class.
2. Extra classes and interfaces are used to provide this information explicitly.

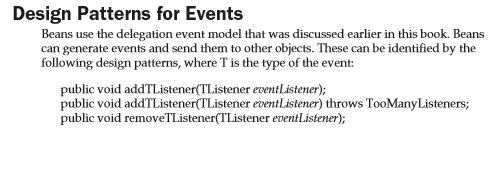
**Naming patterns for Properties, Events and Methods for Bean class.**

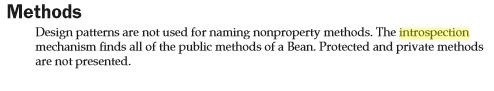
****

****











public class Greeting\_bean\_Simple extends Canvas {

private String message;

private boolean border;

public Greeting\_bean\_Simple()

{

border=true;

message="Hello World";

}

public boolean isBorder() {

return border;

}

public boolean getBorder()

{

return border;

}

public void setBorder(boolean border) {

this.border = border;

repaint();

}

public String getMessage() {

return message;

}

public void setMessage(String message) {

this.message = message;

}

public void paint(Graphics g)

{

message=selectGreeting();

Dimension d=getSize();

int w=d.width;

int h=d.height;

FontMetrics fm=g.getFontMetrics();

int x=(d.width-fm.stringWidth(message))/2;

int y=(d.height+fm.getMaxAscent()-fm.getMaxDescent())/2;

g.drawString(message, x, y);

if(border)

{

g.drawRect(0, 0, w-1, h-1);

}

}

public String selectGreeting()

{

Calendar cal=Calendar.getInstance();

int hour=cal.get(Calendar.HOUR\_OF\_DAY);

if(hour<12)

return "Morning";

else if(hour<19)

return "Afternoon";

else

return "Eveniing";

}

}

**Clock\_Bean**

public class Clock extends Panel implements Runnable {

private TextField tf;

private transient Thread thread;

public Clock()

{

tf=new TextField("", 10);

add(tf);

startThread();

}

public void startThread()

{

thread=new Thread(this);

thread.start();

}

public void run()

{

try {

SimpleDateFormat sdf;

sdf=new SimpleDateFormat("HH:mm:ss");

while(true)

{

Thread.sleep(1000);

tf.setText(sdf.format(new Date()));

}

} catch (Exception e) {

}

}

private void readObject(ObjectInputStream ois)

{

try {

ois.defaultReadObject();

startThread();

} catch (Exception e) {

}

}

}

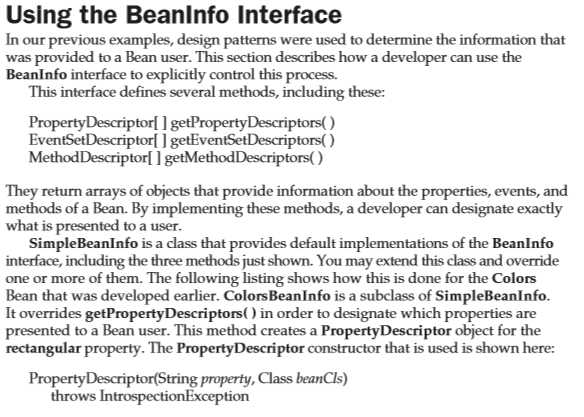
**Extra classes and interfaces are used to provide this information explicitly.**

**Introspector class**

This class is in the java.beans package provide static methods that allow you to obtain information about the properties, events and methods of Bean.

static BeanInfo getbeanInfo(Class beanClass)

static BeanInfo getbeanInfo(Class beanClass, Class ignoreClass)



**PropertyDescriptor**

public PropertyDescriptor[] getPropertyDescriptors()

{

try {

Class cls=Greeting\_bean.class;

PropertyDescriptor pds1,pds2;

pds1=new PropertyDescriptor("border", cls);

pds2=new PropertyDescriptor("message", cls);

PropertyDescriptor pd[]={pds1,pds2};

return pd;

} catch (Exception e) {

}

return null;

}

PropertyDescriptor pd[]=gbi.getPropertyDescriptors();

textField1.setText(pd[0].getName());

textField2.setText(pd[1].getName());

**EventSetDescriptor**

public EventSetDescriptor[] getEventSetDescriptors() {

try {

EventSetDescriptor eds;

String ename[]={"mouseClicked","mouseEntered","mouseExited","mousePressed","mouseReleased"};

eds=new EventSetDescriptor(Greeting\_bean.class,"mouse", MouseListener.class,ename,"addMouseListener","removeMouseListener");

EventSetDescriptor ed[]={eds};

return ed;

} catch (Exception e) {

}

return null;

}

EventSetDescriptor ed[]=gbi.getEventSetDescriptors();

Method mts[]=ed[0].getListenerMethods();

String events="";

for(int i=0;i<mts.length;i++)

{

events=events+mts[i].toString()+"\n";

}

textArea1.setText(events);

**MethodDescriptor**

public MethodDescriptor[] getMethodDescriptors() {

try {

Class c=Greeting\_bean.class;

Class ptype[]=new Class[1];

ptype[0]=Graphics.class;

Method mt1=c.getMethod("paint", ptype);

ParameterDescriptor pds[]=new ParameterDescriptor[1];

pds[0]=new ParameterDescriptor();

MethodDescriptor mds1=new MethodDescriptor(mt1, pds);

MethodDescriptor md[]={mds1};

return md;

} catch (Exception e) {

}

return null;

}

MethodDescriptor md[]=gbi.getMethodDescriptors();

Method imt=md[0].getMethod();

Class cls[]=imt.getParameterTypes();

textArea2.setText(imt.toString()+"\n"+cls[0].toString());

Custom Events ( Page 96)

1:- Create class that define event.

import java.util.EventObject;

public class **ColorEvent** extends **EventObject**{

private Color color;

public **ColorEvent**(Object source,Color color)

{

super(source);

this.color=color;

}

public void setColor(Color color) {

this.color = color;

}

public Color getColor()

{

return color;

}

}

2:-Create Interface that receive this type of Events.

import **java.util.EventListener**;

public interface **ColorListner** extends **EventListener**{

public void **changeColor**(ColorEvent ce);

}

3:-Source must provide methods that allows listner to register and unregister for Events notification.

public void **addColorListner**(**ColorListner** cl)

{

listners.addElement(cl);

}

public void **removeColorListner**(**ColorListner** cl)

{

listners.removeElement(cl);

}

4:- Source must provide code to generate events and send it to all register listeners.

public void **fireColorEvent**(**ColorEvent** ce)

{

for(int i=0;i<listners.size();i++)

{

ColorListner cl=(ColorListner)listners.elementAt(i);

cl.**changeColor**(ce);

}

}

5:- Listener must implement the interface to receive event.

public class **Painter** extends **Canvas** implements **ColorListner** {

6:- Listener must register to receive notification

**selecto1**.addColorListner(**painter1**);