Defining a Class
(The Implementor View)

- An example
  - A GUI Button

- Attributes:
  - State (up or down)
  - Caption
  - Position and Size
  - Visibility
  - etc.

The `Button` Class

- Class diagram

The **Button** Class (cont.)

- The Java code:

```
public class Button
  { public final static int UP = 0;
    public final static int DOWN = 1;

    public int state;
    public String caption;
    public int x, y,   // Position
               w, h;   // Size
    public boolean visible;
  } // class Button
```

Using the **Button** Class (The Client View)

- Need to create **Button** objects:

```
Button b1 = new Button(),
       b2 = new Button();
```

- Using the **Button** objects:

```
b1.caption = ”Press Me”;
...
if (b2.visible)
  { ...
  }
```

Types Are More Than Just Data!

- We want to be able to *operate* on the data
- Could write functions and procedures
  - Separates data and operations
- Rather want to deal with the data and the operations together (*encapsulation*)
  - Objects, with *methods*

Adding Methods to a Class

- Methods for a GUI button:
  - Press and Release
  - Display a *bitmap* (image)

Press Me

Button

```
state, caption
x, y, w, h
visible,
upMap, downmap
```

```
displayBitmap
press, release
```

Java Code

```
public class Button
  { public final static int UP = 0;
    public final static int DOWN = 1;
    public int state;
    public String caption;
    public int x, y, w, h;
    public boolean visible;
    public Bitmap upMap, downMap;
    public void displayBitmap ()
      {...}
    public void press ()
      {...}
    public void release ()
      {...}
  } // class Button
```

Bitmaps

The methods

Using the Methods

```
b1.press();

...

b1.release();
```

Abstract Data Types via Access Control

- Some of the details of the button objects are only of *internal* interest
  - The client should not be concerned with them

- E.g.
  - The bitmaps
  - The **`displayBitmap`** method

ADTs via Access Control

- Separating the interface (external) and the implementation (internal) is very powerful
  - Allows separate development
  - Breaks up complex problems into simple subproblems
- Called an *Abstract Data Type*

Access Control in Java

- Members of a class can be either **`public`** or **`private`**
- Public members can be accessed by client code
- Private members can *only* be accessed from within the class
  - i.e. by the methods of the class

Access Control: Class Diagrams

- Private members in grey

Access Control: Java Code

```
public class Button
  { public final static int UP = 0;
    public final static int DOWN = 1;
    private int state;
    public String caption;
    public int x, y, w, h;
    public boolean visible;
    private Bitmap upMap, downMap;
    private void displayBitmap ()
      {...}
    public void press ()
      {...}
    public void release ()
      {...}
  } // class Button
```

Private Members

Access Control: Client View

- The client *cannot* use the private members
  - They might as well not even exist
- The implementor is free to change the private parts
  - The client is unaffected

Constructors

- Methods called when the object is created
- Useful for initialisation
- Syntax
  - Name is the same as the class
  - No return type

Constructors (cont.)

```
public class Button
  { ...
    public Button ()
      { release();
      } // constructor

    private void displayBitmap ()
      {...}
    public void press ()
      {...}
    public void release ()
      {...}
  } // class Button
```