Copyright Warning

ViSLAB
High-Performance Visualization, Communication, Computing

Object Oriented Design

Week 9-1

- Generation :
  - Singleton
- Structural:
  - Adapter
- Behaviour:
  - Memento

Singleton

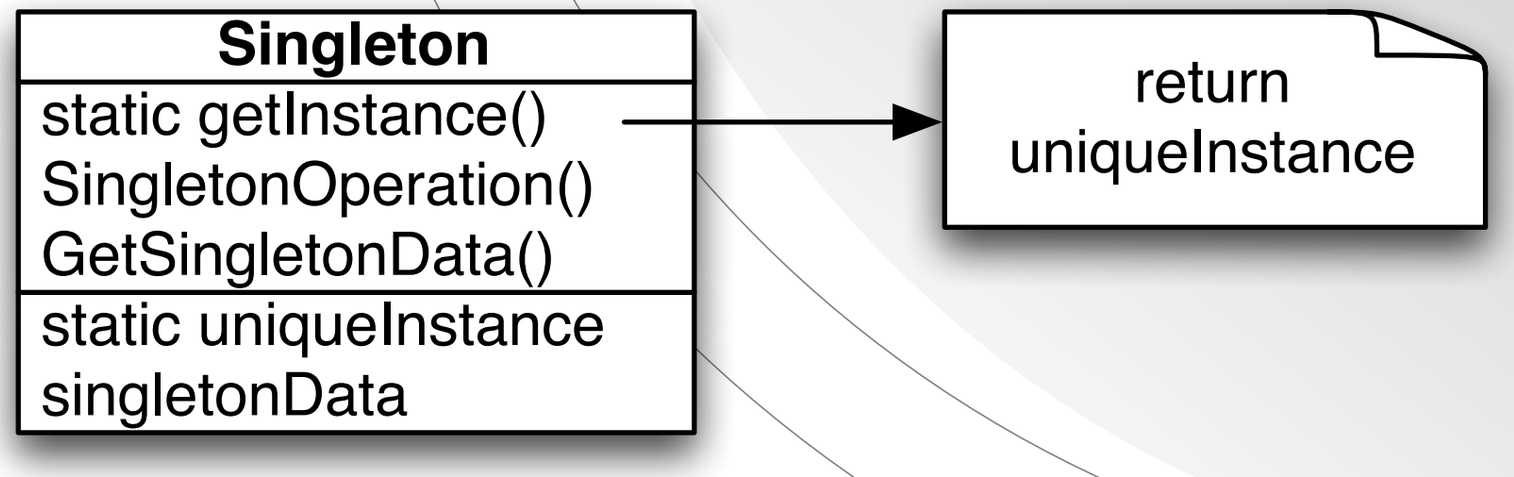
- Guarantees
  - only one instance exists

- Example:
  - Global Timer object in a Computer Game.
  - Single instance of the global timer maintains the consistency

Singleton

- Create just one global variable?
- Singleton Design Pattern
- enforces and makes sure that only one!
- try to make two or more ...error!

Singleton




```
class GameTimer {
private:
   static GameTimer* sInstance;
protected:
   GameTimer();
public:
   static GameTimer* getInstance();
   int getTime();
};
```

Singleton

```
class GameTimer {
private:
   static GameTimer* sInstance;
protected:
   GameTimer();
public:
   static GameTimer* getInstance();
   int getTime();
};
```

```
protected:
   GameTimer();
```

- in main() or any non-GameTimer function
  - `GameTimer gt;` <- compile error

Singleton

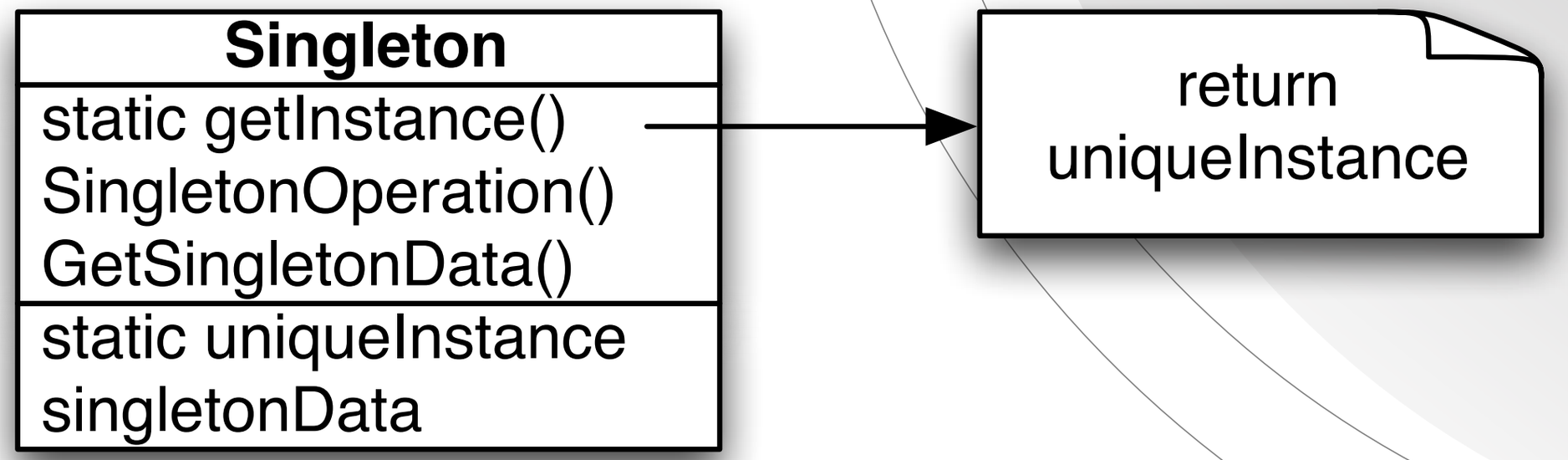
- How do you create an object?
  - getInstance()....static & public
  - sInstance .....static & private

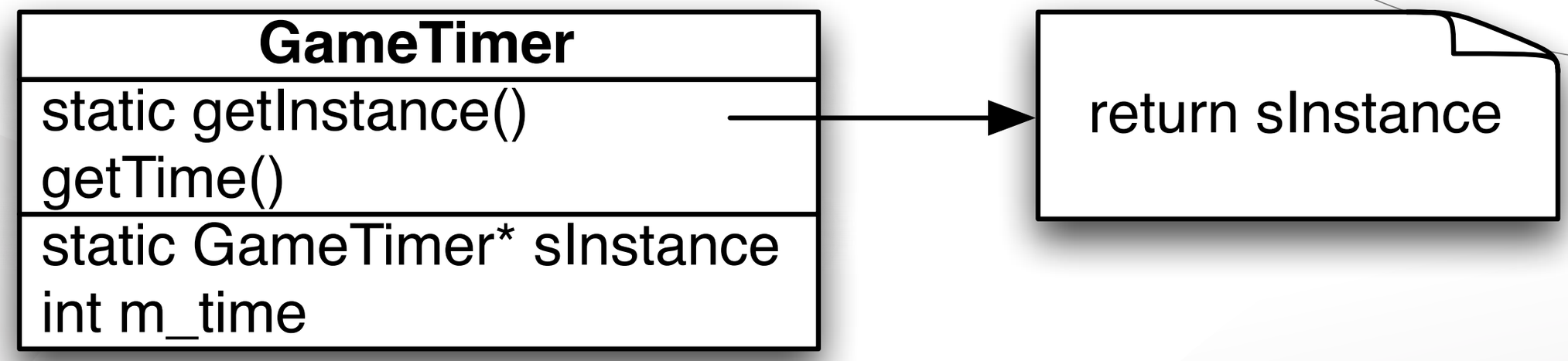
- static:
  - sInstance is allocated on the memory at launch
  - getInstance() function pointer is also allocated on the memory at launch

Singleton

- access the only one GameTimer object through getInstance()
- getInstance()....static & public
- sInstance .....static & private

```
class GameTimer {
private:
   static GameTimer* sInstance;
protected:
   GameTimer();
public:
   static GameTimer* getInstance();
   int getTime();
};

GameTimer* GameTimer::sInstance = 0;

GameTimer* GameTimer::getInstance() {
    if (sInstance == 0)
     sInstance = new GameTimer();
    return sInstance;
}

int main() {
   //...
   GameTimer* gt = GameTimer::getInstance();
}
```





Adapter (Wrapper)

- Wraps another class with the adapter class
  - force another class's interface to match

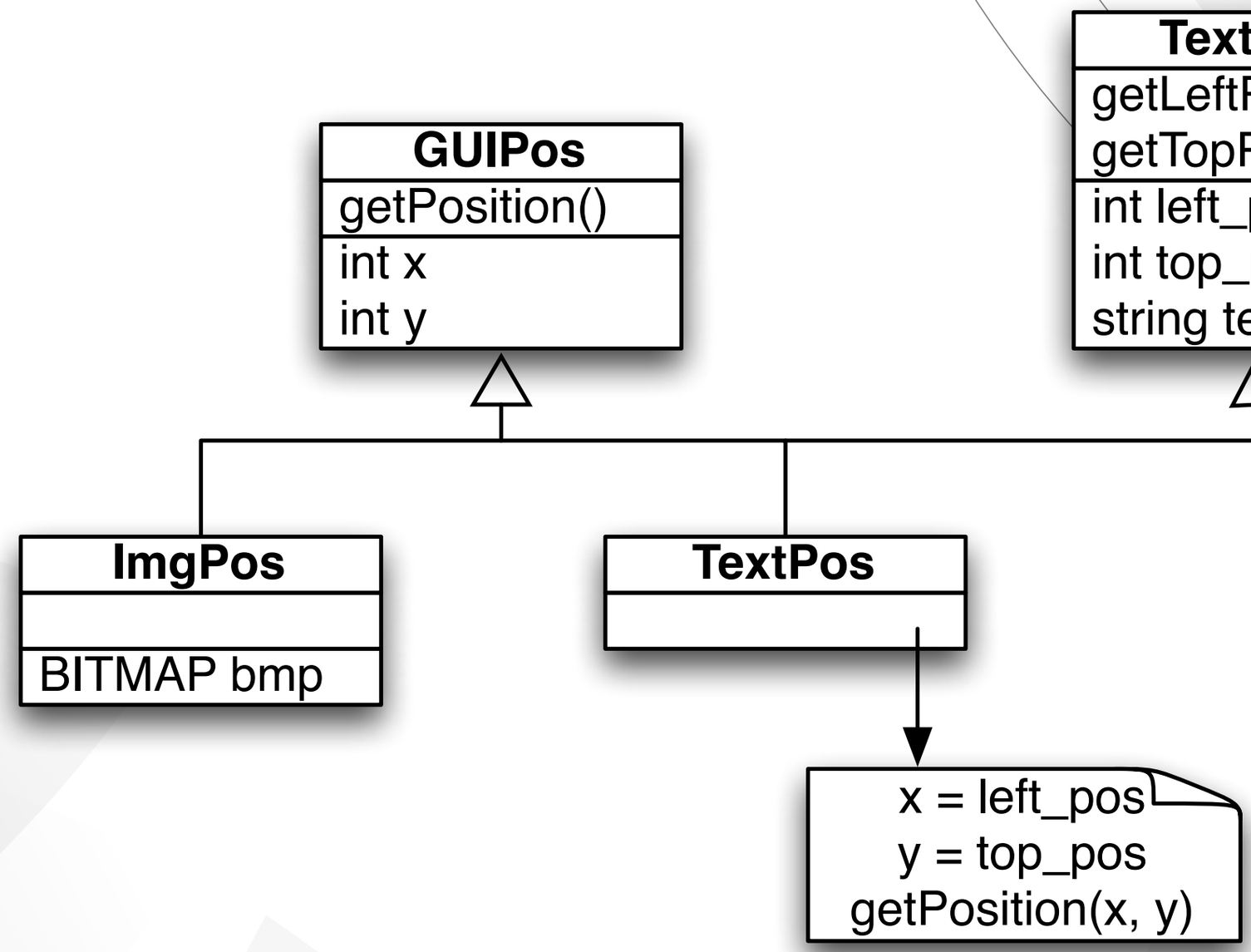
- Example:
  - place a graphical object
  - superclass only maintain 2D position

```
class GUIPos {
protected:
    int x;
    int y;
public:
    void getPosition(int *x, int *y) {
        *x = this->x;
        *y = this->y;
    }
};

class ImgPos : public GUIPos {
protected:
    BITMAP bmp;
};

class TextBox {
protected:
    int left_pos;
    int top_pos;
    string text;
public:
    int getLeftPos() {return left_pos;}
    int getTopPos() {return top_pos;}
    void setText(char *text);
};
```

Adapter (Wrapper)

- ImgPos is already subclass of GUIPos
- we want to use TextBox like GUIPos
- but we did not create TextBox
- use Adapter pattern to force TextBox to match GUIPos

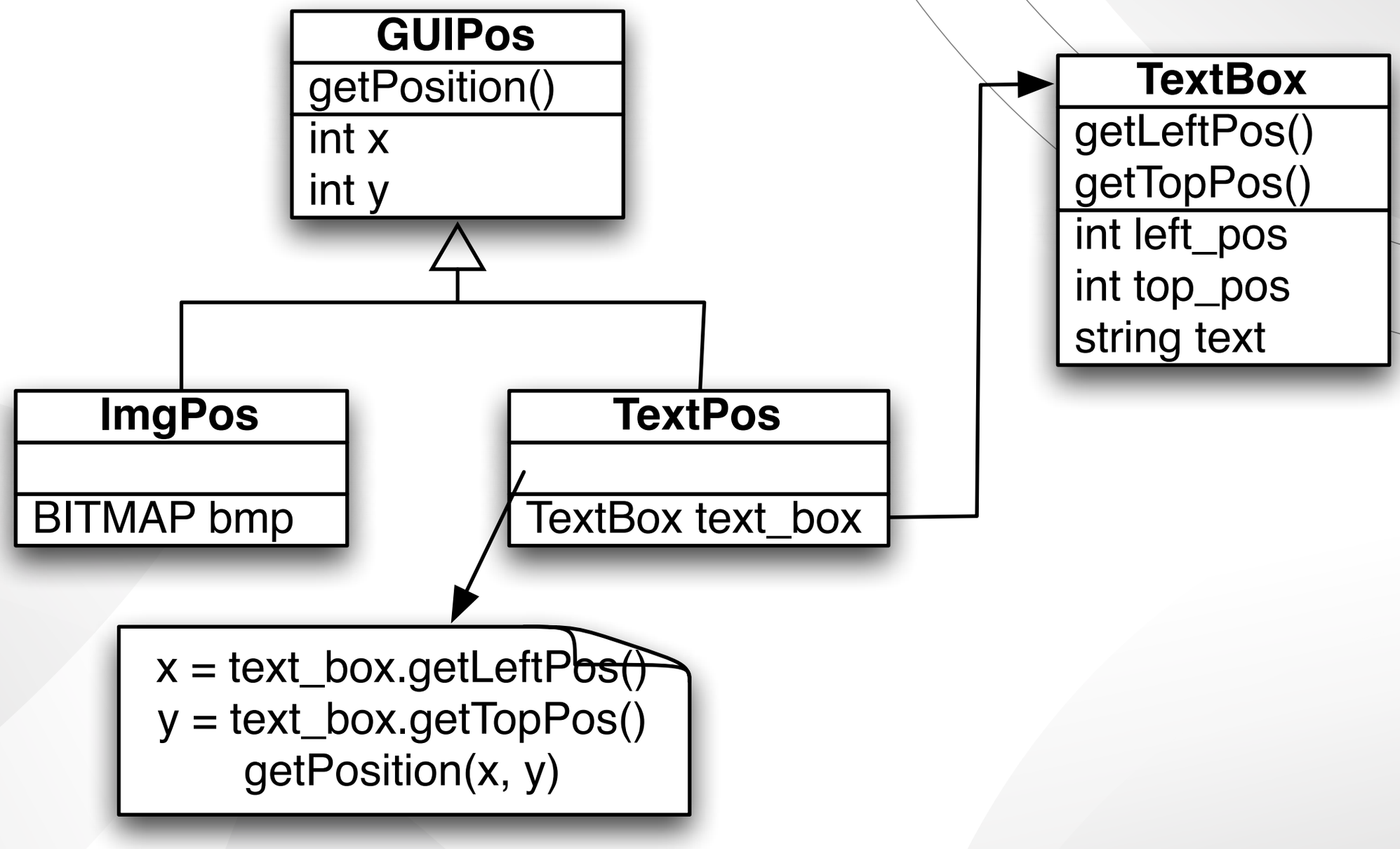
Adapter (multiple inheritance)



Adapter (multiple inheritance)

- type is determined at the compile time
- TextBox's members might be exposed

Adapter (Composition)



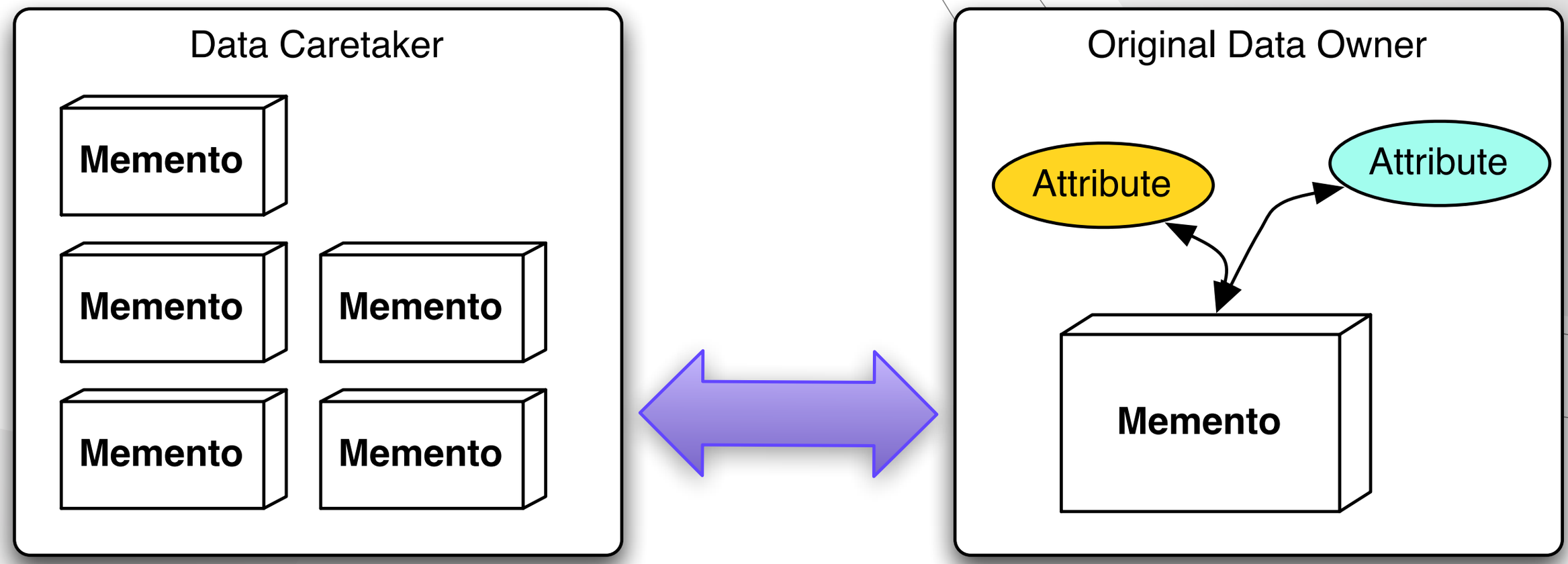
Adapter (Composition)

- hold a pointer to TextBox (might be better)
  - subclass of TextBox can be hold.
  - does not break encapsulation of TextBox
  - but you have to instantiate TextBox.

Memento

- Data storage method

- Example:
  - In game, you want to cancel an event but want to be able to resume it later
  - you need to save the state (data).
  - but you don't want to make the data widely accessible (maintain encapsulation)



What's in the box?

- Only the original owner can see the content of the box (memento)
- Some OOP Lang does not have sufficient mechanism at the lang level
- C++ has "friend"
  - allowed other class to access "private" and "protected"

“friend”

```
class MyFrined;

class MyClass {
private:
   Item myData;
protected:
   Item showMyData() {return myData;}

   friend class MyFriend;
};
```

Memento pattern

- Only OriginalOwner can open Memento class (box)
  - OriginalOwner is a friend of Memeonto
- Other class (caretaker) can only hold Memento objects.

```
   class Memento {
private:
   friend class OriginalOwner;
   State* myFriendState; //OrigianlOwner's state

   Memento(); // does not allow anyone to instantiate
   void setState(State*);
   State* getState();
public:
   ~Memento();
};

class OriginalOwner {
private:
   State* myState;
public:
   Memento* createMemento();//create a memento box with state in it
   void setMemento(Memento*); // receive a box and set myState from it
};
```

example

- pacman
- GameManager
  - keeps current score (int)
  - current location of the pacman (x, y)
  - current map int[20x20]

```
class GameManager {
private:
    int score;
    int x, y; // current location
    int map[400];
public:
    void setMemento(Memento *mem);
    Memento* createMemento();

    void moveTo(int x, int y);
    void reset();
    void calcScore();
};

class Memento {
private:
    int score;
    int x, y;
    int map[400];

    Memento(){};

    void setScore(int scr);
    int getScore();
    void setLocation(int x, int y);
    void getLocation(int *x, int *y);
    void setMap(int size, int *);
    void getMap(int size, int *);

    friend class GameManager;

public:
    ~Memento(){};
};
```

```
int main() {
    ...
    GameManager gMgr;
    Memento* myMemento;

    myMemento = gMgr.getMemento();

    gMgr.setMemento(myMemento);
    ...
}
```

Memento