C++ Interlude 1
C++ Classes

PlainBox Class - Version 1

Basic Class

Pre-processor directives

Initializer list

ce indicator

Default Constructor

Parameterized Constructor

Parameters are passed by constant reference

Accessor methods are declared `const`

Method Implementations

```
#ifndef _PLAIN_BOX
#define _PLAIN_BOX


class PlainBox
{
private:
    double     item;

public:
    PlainBox();
    PlainBox(const double& theItem);
    void setItem(const double& theItem);
    double getItem() const;
};


#endif
```

PlainBox.h

```
#include "PlainBox.h"

inBox::PlainBox()
  : item(0.0)
{   } // end default constructor

PlainBox::PlainBox()
{
   item = 0.0;
} // end default constructor

m)


void PlainBox::setItem(const double& theItem)
{
  item = theItem;
} // end setItem


double PlainBox::getItem() const
{
  return item;
} // end getItem
```

PlainBox.cpp

Pearson

PlainBox Class - Version 1

Basic Class

```
#ifndef _PLAIN_BOX
#define _PLAIN_BOX


class PlainBox
{
private:
    double     item;

public:
    PlainBox();
    PlainBox(const double& theItem);
    void setItem(const double& theItem);
    double getItem() const;
};


#endif
```

PlainBox.h

```
#include "Plai

PlainBox::Plai
    : item(0.0)
{   } // end defa

PlainBox::Plai
    : item(thelte
{   } // end con

void Pl                    (const double& theItem)
{
  item =
} // end s

dou                        m() const
{
  return item;
} // end getIte
```

PlainBox.cpp

Client Code

```
double dish = 8.5;
PlainBox firstBox(dish);
std::cout << firstBox.getItem() << std::endl;

double bowl = 4.0;
PlainBox anotherBox = PlainBox(bowl);
anotherBox.setItem(dish);
std::cout << anotherBox.getItem() << std::endl;
```

8.5

4.0

firstBox

anotherBox

Pearson

PlainBox Class - Version 2

Templates

```
#ifndef _PLAIN_BOX
#define _PLAIN_BOX

template<class ItemType>
class PlainBox
{
private:
    ItemType item;

public:
    PlainBox();
    PlainBox(const ItemType& theItem);
    void setItem(const ItemType& theItem);
    ItemType getItem() const;
};
#include "PlainBox.cpp"

#endif
```

PlainBox.h

`template` allows client to decide type

```
#include "PlainBox.h"

template<class ItemType>
PlainBox<ItemType>::PlainBox()
  :
{ } // end default constructor

template<class ItemType>
PlainBox<ItemType>::PlainBox(const ItemType& theItem)
  : item(theItem);
{ } // end constructor

template<class ItemType>
void PlainBox<ItemType>::setItem(const ItemType& theItem)
{
  item = theItem;
} // end setItem

template<class ItemType>
ItemType PlainBox<ItemType>::getItem() const
{
  return item;
} // end getItem
```

PlainBox.cpp

PlainBox Class - Version 2

Templates

```
#ifndef _PLAIN_BOX
#define _PLAIN_BOX

 template<class ItemType>
class PlainBox
{
private:
    ItemType item;

public:
    PlainBox();
    PlainBox(const ItemType& theItem);
    void setItem(const ItemType& theItem);
    ItemType getItem() const;
};
#include "PlainBox.cpp"

#endif
```

PlainBox.h

Client Code

```
double dish = 8.5;
PlainBox<double> firstBox(dish);
std::cout << firstBox.getItem() << std::endl;

string animal = "Dog";
PlainBox<string> anotherBox = PlainBox<string>(animal);
anotherBox.setItem("Cat");
std::cout << anotherBox.getItem() << std::endl;
```

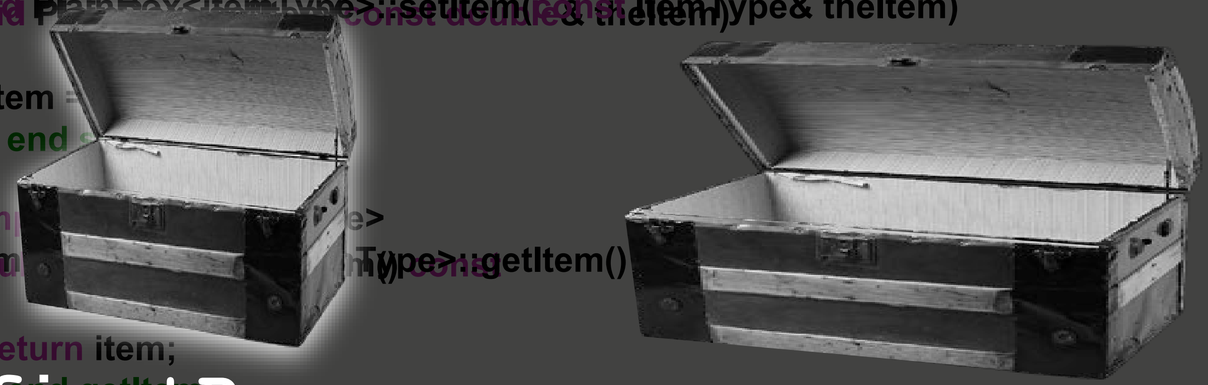
8.5

Dog

Cat

```
template<class ItemType>
void PlainBox<ItemType>::setItem(const ItemType& theItem)
{
   item = theItem;
} // end setItem

template<class ItemType>
ItemType PlainBox<ItemType>::getItem() const
{
   return item;
} // end getItem
```



firstBox

PlainBox.cpp

C++ Interlude 1
C++ Classes

ToyBox Class

Derived Class

```
#ifndef _TOYBOX
#define _TOYBOX
#include "PlainBox.h"

enum Color {BLACK, RED, BLUE, GREEN, YELLOW};

template<class ItemType>
class ToyBox : public PlainBox<ItemType>
{
private:
    Color boxColor;

public:
    ToyBox();
    ToyBox(const Color& theColor);
    ToyBox(const ItemType& theItem,
                        const Color& theColor);

    Color getColor() const;
}; // end ToyBox

#include "ToyBox.cpp"
#endif
```

ToyBox.h

Base class default constructor is called implicitly

```
#include "ToyBox.h"  // optional

template<class ItemType>
ToyBox<ItemType>::ToyBox()
    : boxColor(BLACK)
{   }  // end default constructor

template<class ItemType>
ToyBox<ItemType>::ToyBox(const Color& theColor)
     : boxColor(theColor)
{   }  // end constructor

template<class ItemType>
ToyBox<ItemType>::ToyBox(const ItemType& theItem,
                                        const Color& theColor)
     : PlainBox<ItemType>(theItem), boxColor(theColor)
{   }  // end constructor

template<class ItemType>
Color ToyBox<ItemType>
{
    return boxColor;
}  // end getColor
```

Base class constructor must be first in the initializer list

ToyBox.cpp

Client Code

BoxInterface Class

Abstract Base Class

```
#ifndef _BOXINTERFACE
#define _BOXINTERFACE

template <class ItemType>
class BoxInterface
{
public:
   virtual void setItem(const ItemType& theItem) = 0;
   virtual ItemType getItem() const = 0;
  virtual ~BoxInterface() { }
}; // end BoxInterface

#endif
```

BoxInterface.h

```
#ifndef _PLAIN_BOX
#define _PLAIN_BOX

#include "BoxInterface.h"

template<class ItemType>
class PlainBox : public BoxInterface<ItemType>
{
private:
    ItemType item;

public:
    PlainBox();
    PlainBox(const ItemType& theItem);
    virtual void setItem(const ItemType& theItem);
    virtual ItemType getItem() const;
};

#include "PlainBox.cpp"
#endif
```

PlainBox.h

Pearson