EXAMPLE 58: DON'T EVER USE VIRTUAL FUNCTIONS IN A CONSTRUCTOR OR DESTRUCTOR (UNLIKE IN JAVA!)

EXAMPLE 58: DON'T EVER USE VIRTUAL FUNCTIONS IN A CONSTRUCTOR OR DESTRUCTOR (UNLIKE IN JAVA!)

WHY?

BECAUSE VIRTUAL FUNCTIONS "DON'T EXIST" IN A CONSTRUCTOR OR A DESTRUCTOR. ITS ALWAYS THE CURRENT-CLASS VERSION THAT IS CALLED.

EXAMPLE 58: DON'T EVER USE VIRTUAL FUNCTIONS IN A CONSTRUCTOR OR DESTRUCTOR (UNLIKE IN JAVA!)

WHY?

BECAUSE VIRTUAL FUNCTIONS "DON'T EXIST" IN A CONSTRUCTOR OR A DESTRUCTOR. ITS ALWAYS THE CURRENT-CLASS VERSION THAT IS CALLED.

SO?

ERR..THAT VIOLATES THE DEFINITION OF A VIRTUAL FUNCTION..SO ITS WRONG.

EXAMPLE 58: DON'T EVER USE VIRTUAL FUNCTIONS IN A CONSTRUCTOR OR DESTRUCTOR (UNLIKE IN JAVA!)

WHY? BECAUSE VIRTUAL FUNCTIONS "DON'T EXIST" IN A CONSTRUCTOR OR A DESTRUCTOR. ITS ALWAYS THE CURRENT-CLASS VERSION THAT IS CALLED.

SO? ERR..THAT VIOLATES THE DEFINITION OF A VIRTUAL FUNCTION..SO ITS WRONG.

WHAT'S WORSE -

ANY DERIVED CLASS CONSTRUCTOR THAT CALLS YOUR BASE CLASS CONSTRUCTOR WILL END UP WITH THE WRONG VERSION OF THE FUNCTION BEING CALLED!

EXAMPLE 58: DON'T EVER USE VIRTUAL FUNCTIONS IN A CONSTRUCTOR OR DESTRUCTOR (UNLIKE IN JAVA!)

WHY? BECAUSE VIRTUAL FUNCTIONS "DON'T EXIST" IN A CONSTRUCTOR OR A DESTRUCTOR. ITS ALWAYS THE CURRENT-CLASS VERSION THAT IS CALLED.

SO? ERR, THAT VIOLATES THE DEFINITION OF A VIRTUAL FUNCTION, SO ITS WRONG.

OH, AND BTW, IF THE FUNCTION IS A PURE VIRTUAL FUNCTION, A LINKER ERROR WILL RESULT DURING COMPILATION

WHAT'S WORSE - ANY DERIVED CLASS CONSTRUCTOR THAT CALLS YOUR BASE CLASS CONSTRUCTOR WILL END UP WITH THE WRONG VERSION OF THE FUNCTION BEING CALLED!

EXAMPLE 58: DON'T EVER USE VIRTUAL FUNCTIONS IN A CONSTRUCTOR OR DESTRUCTOR (UNLIKE IN JAVA!)

WHY? BECAUSE VIRTUAL FUNCTIONS "DON'T EXIST" IN A CONSTRUCTOR OR A DESTRUCTOR. ITS ALWAYS THE CURRENT-CLASS VERSION THAT IS CALLED.

SO? FOR THAT MATTER, THE DEFINITION OF A VIRTUAL FUNCTION..SO ITS WRONG.

WHAT'S WORSE - ANY DERIVED CLASS CONSTRUCTOR THAT CALLS YOUR BASE CLASS CONSTRUCTOR WILL END UP WITH THE WRONG VERSION OF THE FUNCTION BEING CALLED!

THIS SOUNDS EXTREMELY WEIRD - AND IT IS. JAVA IS N'T LIKE THIS.

OH, AND BTW, IF THE FUNCTION IS A PURE VIRTUAL FUNCTION, A LINKER ERROR WILL RESULT DURING COMPILATION

THIS SOUNDS EXTREMELY WEIRD - AND IT IS. JAVA IS N'T LIKE THIS.

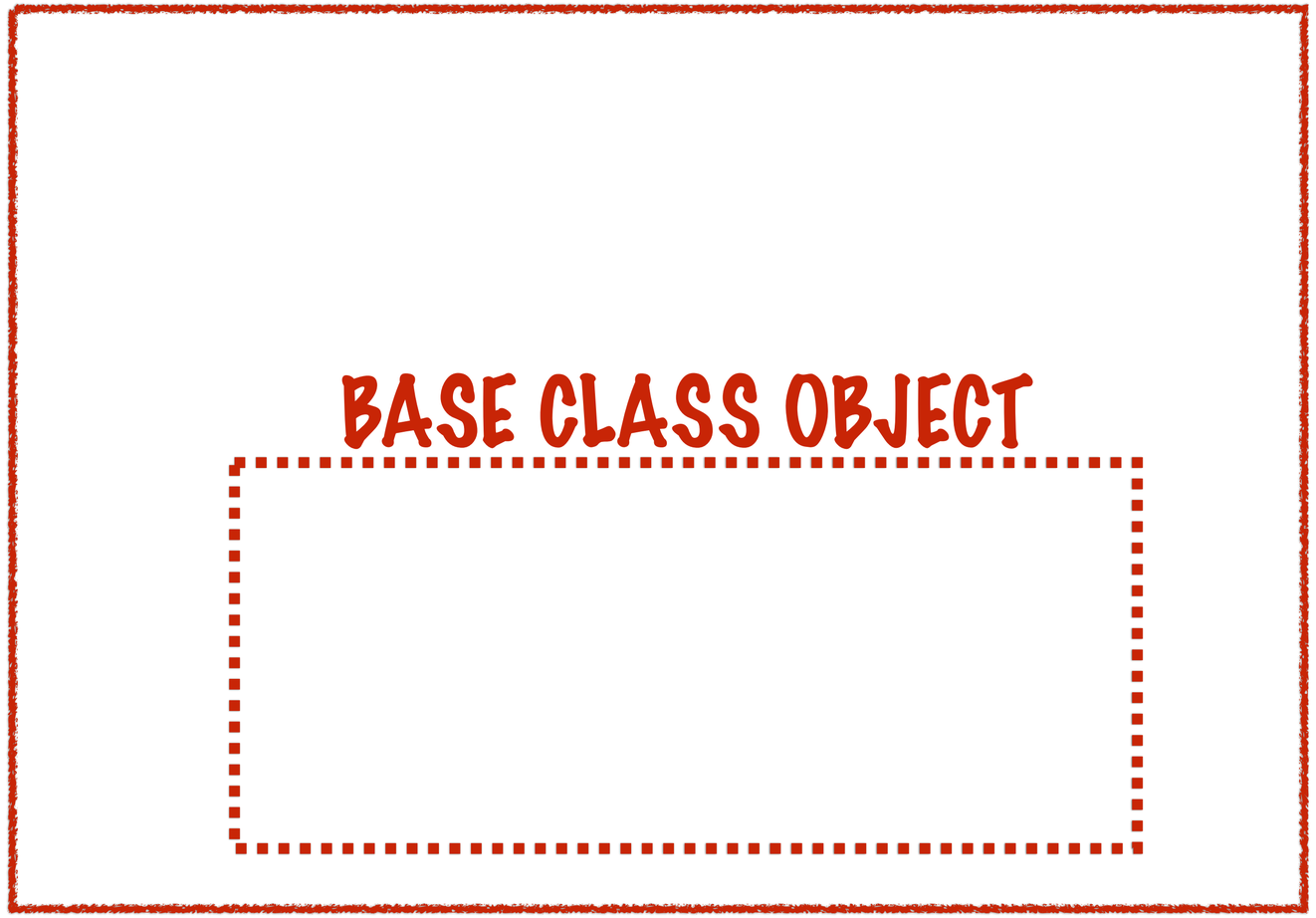
DURING CONSTRUCTION- WHEN A BASE CLASS OBJECT IS BEING CONSTRUCTED, THE DERIVED CLASS OBJECT HAS YET TO BE COME INTO EXISTENCE

DURING DESTRUCTION - WHEN THE BASE CLASS OBJECT IS BEING DESTRUCTED, THE DERIVED CLASS HAS ALREADY CEASED TO EXIST

REMEMBER THE ORDER IN WHICH BASE AND DERIVED CLASS CONSTRUCTORS AND DESTRUCTORS ARE CALLED!

REMEMBER THE ORDER IN WHICH BASE AND DERIVED CLASS CONSTRUCTORS AND DESTRUCTORS ARE CALLED!

ORDER OF CONSTRUCTOR CALLS



ORDER OF DESTRUCTOR CALLS

HERE IS A BAD BASE CLASS

```
class Shape
{
private:
public:
  string shapeType;
  virtual void print()
  {
    cout << "I am a shape" << endl;
  }
  Shape()
  {
    shapeType = "Unknown";
    cout << "Inside the Shape constructor" << endl;
    print(); //BAD! Virtual function call inside constructor
  }

  ~Shape()
  {
    cout << "Inside the Shape destructor" << endl;
    print(); // BAD! Virtual function call inside destructor
  }
};
```

HERE IS A BAD BASE CLASS

```
class Shape
{
private:
public:
  string shapeType;
  virtual void print()
  {
    cout << "I am a shape" << endl;
  }
  Shape()
  {
    shapeType = "Unknown";
    cout << "Inside the Shape constructor" << endl;
    print(); // BAD! Virtual function call inside constructor
  }

  ~Shape()
  {
    cout << "Inside the Shape destructor" << endl;
    print(); // BAD! Virtual function call inside destructor
  }
};
```

IT HAS A VIRTUAL FUNCTION..

HERE IS A BAD BASE CLASS

```
class Shape
{
private:
public:
  string shapeType;
  virtual void print()
  {
    cout << "I am a shape" << endl;
  }
  Shape()
  {
    shapeType = "Unknown";
    cout << "Inside the Shape constructor" << endl;
    print(); //BAD! Virtual function call inside constructor
  }

  ~Shape()
  {
    cout << "Inside the Shape destructor" << endl;
    print(); // BAD! Virtual function call inside destructor
  }
};
```

IT HAS A VIRTUAL FUNCTION..

THAT IT CALLS BOTH IN THE CONSTRUCTOR AND IN THE DESTRUCTOR

HERE IS A THE CORRESPONDING DERIVED CLASS

```
class Rectangle : public Shape
{
public:
  int rectangle_length;
  int rectangle_breadth;
  virtual void print()
  {
    cout << "I am a rectangle" << endl;
  }
  Rectangle()
  {
    cout << "Inside the Rectangle constructor" << endl;
  }
  ~Rectangle()
  {
    cout << "Inside the Rectangle destructor" << endl;
  }
};
```

NOTHING NOTEWORTHY HERE..

HERE IS A THE CORRESPONDING DERIVED CLASS

```
class Rectangle : public Shape
{
public:
  int rectangle_length;
  int rectangle_breadth;
  virtual void print()
  {
    cout << "I am a rectangle" << endl;
  }
  Rectangle()
  {
    cout << "Inside the Rectangle constructor" << endl;
  }
  ~Rectangle()
  {
    cout << "Inside the Rectangle destructor" << endl;
  }
};
```

IMPLEMENTS THE VIRTUAL FUNCTION, THAT'S ABOUT IT.

NOW WE INSTANTIATE THIS POOR, HAPLESS DERIVED CLASS -

```
Rectangle s1;
```

AND WHAT GETS CALLED?

```
[Vitthals-MacBook-Pro:~ vitthalsrinivasan$ ./a.out
Inside the Shape constructor
I am a shape
Inside the Rectangle constructor
Inside the Rectangle destructor
Inside the Shape destructor
I am a shape
```

THE WRONG VERSIONS OF THE FUNCTION!