FRIENDS

EXAMPLE 6: UNDERSTAND THE ACCESS MODIFIERS: PRIVATE, PUBLIC AND PROTECTED

RECAP

EXAMPLE 6: UNDERSTAND THE ACCESS MODIFIERS: PRIVATE, PUBLIC AND PROTECTED

MEMBER FUNCTIONS AND MEMBER VARIABLES CAN BE MARKED AS PUBLIC (IF USABLE OUTSIDE THE CLASS) OR PRIVATE (INTERNAL TO THE CLASS)

PROTECTED IS SIMILAR, BUT RELATED TO INHERITANCE - MORE LATER

RECAP

EXAMPLE 6: UNDERSTAND THE ACCESS MODIFIERS: PRIVATE, PUBLIC AND PROTECTED

MEMBER FUNCTIONS AND MEMBER VARIABLES CAN BE MARKED AS PUBLIC (IF USABLE OUTSIDE THE CLASS) OR PRIVATE (INTERNAL TO THE CLASS)

```
class ComplexNumber
{
private:
  float realPart;
  float complexPart;
public:
  ComplexNumber() : realPart(0.0),complexPart(0.0)
  {
    cout << "No arg-constructor called" << endl;
  }
  ComplexNumber(double c, double r) : realPart(r) , complexPart(c)
  {
    cout << "Inside the 2-argument constructor" << endl;
  }
  void setRealPart(double r)
  {
    realPart = r;
  }

  void setComplexPart(double c)
  {
    complexPart = c;
  }

  float getRealPart()
```

A SECTION MARKED `public` THAT IS FOR MEMBER FUNCTIONS AND DATA ACCESSIBLE TO CODE OUTSIDE THE CLASS AS WELL

RECAP

EXAMPLE 6: UNDERSTAND THE ACCESS MODIFIERS: PRIVATE, PUBLIC AND PROTECTED

MEMBER FUNCTIONS AND MEMBER VARIABLES CAN BE MARKED AS PUBLIC (IF USABLE OUTSIDE THE CLASS) OR PRIVATE (INTERNAL TO THE CLASS)

```
class ComplexNumber
{
private:
  float realPart;
  float complexPart;
public:
  ComplexNumber() : realPart(0.0),complexPart(0.0)
  {
    cout << "No arg-constructor called" << endl;
  }
  ComplexNumber(double c, double r) : realPart(r) , complexPart(c)
  {
    cout << "Inside the 2-argument constructor" << endl;
  }
  void setRealPart(double r)
  {
    realPart = r;
  }

  void setComplexPart(double c)
  {
    complexPart = c;
  }

  float getRealPart()
```

CODE ANYWHERE CAN ACCESS PUBLIC PORTIONS

A SECTION MARKED `public` THAT IS FOR MEMBER FUNCTIONS AND DATA ACCESSIBLE TO CODE OUTSIDE THE CLASS AS WELL

RECAP

EXAMPLE 6: UNDERSTAND THE ACCESS MODIFIERS: PRIVATE, PUBLIC AND PROTECTED

MEMBER FUNCTIONS AND MEMBER VARIABLES CAN BE MARKED AS PUBLIC (IF USABLE OUTSIDE THE CLASS) OR PRIVATE (INTERNAL TO THE CLASS)

A SECTION MARKED `public` THAT IS FOR MEMBER FUNCTIONS AND DATA ACCESSIBLE TO CODE OUTSIDE THE CLASS AS WELL

CODE ANYWHERE CAN ACCESS PUBLIC PORTIONS

RECAP

EXAMPLE 6: UNDERSTAND THE ACCESS MODIFIERS: PRIVATE, PUBLIC AND PROTECTED

MEMBER FUNCTIONS AND MEMBER VARIABLES CAN BE MARKED AS PUBLIC (IF USABLE OUTSIDE THE CLASS) OR PRIVATE (INTERNAL TO THE CLASS)

```
class ComplexNumber
{
private:
  float realPart;
  float complexPart;
public:
  ComplexNumber() : realPart(0.0),complexPart(0.0)
  {
    cout << "No arg-constructor called" << endl;
  }
  ComplexNumber(double c, double r) : realPart(r) , complexPart(c)
  {
    cout << "Inside the 2-argument constructor" << endl;
  }
  void setRealPart(double r)
  {
    realPart = r;
  }

  void setComplexPart(double c)
  {
    complexPart = c;
  }
```

A SECTION MARKED `private` THAT IS FOR MEMBER FUNCTIONS AND DATA ACCESSIBLE ONLY INSIDE THE CLASS

RECAP

EXAMPLE 6: UNDERSTAND THE ACCESS MODIFIERS: PRIVATE, PUBLIC AND PROTECTED

MEMBER FUNCTIONS AND MEMBER VARIABLES CAN BE MARKED AS PUBLIC (IF USABLE OUTSIDE THE CLASS) OR PRIVATE (INTERNAL TO THE CLASS)

```
class ComplexNumber
{
private:
  float realPart;
  float complexPart;
public:
  ComplexNumber() : realPart(0.0),complexPart(0.0)
  {
    cout << "No arg-constructor called" << endl;
  }
  ComplexNumber(double c, double r) : realPart(r) , complexPart(c)
  {
    cout << "Inside the 2-argument constructor" << endl;
  }
  void setRealPart(double r)
  {
    realPart = r;
  }

  void setComplexPart(double c)
  {
    complexPart = c;
  }

  float getRealPart()
```

IF CODE OUTSIDE A CLASS TRIES TO ACCESS THIS, A COMPILE ERROR WILL RESULT

A SECTION MARKED `private` THAT IS FOR MEMBER FUNCTIONS AND DATA ACCESSIBLE ONLY INSIDE THE CLASS

RECAP

EXAMPLE 6: UNDERSTAND THE ACCESS MODIFIERS: PRIVATE, PUBLIC AND PROTECTED

MEMBER FUNCTIONS AND MEMBER VARIABLES CAN BE MARKED AS **PUBLIC** (IF USABLE OUTSIDE THE CLASS) OR **PRIVATE** (INTERNAL TO THE CLASS)

A SECTION MARKED `private` THAT IS FOR MEMBER FUNCTIONS AND DATA ACCESSIBLE ONLY INSIDE THE CLASS

IF CODE OUTSIDE A CLASS TRIES TO ACCESS THIS, A COMPILE ERROR WILL RESULT

BUT ALL OBJECTS OF A CLASS CAN ACCESS EACH OTHERS PRIVATE PARTS

RECAP

EXAMPLE 6: UNDERSTAND THE ACCESS MODIFIERS: PRIVATE, PUBLIC AND PROTECTED

MEMBER FUNCTIONS AND MEMBER VARIABLES CAN BE MARKED AS **PUBLIC** (IF USABLE OUTSIDE THE CLASS) OR **PRIVATE** (INTERNAL TO THE CLASS)

A SECTION MARKED `private` THAT IS FOR MEMBER FUNCTIONS AND DATA ACCESSIBLE ONLY INSIDE THE CLASS

IF CODE OUTSIDE A CLASS TRIES TO ACCESS THIS, A COMPILE ERROR WILL RESULT

BUT ALL OBJECTS OF A CLASS CAN ACCESS EACH OTHERS PRIVATE PARTS

THE ACCESS MODIFIERS EXIST TO FORCE GOOD DESIGN, NOT FOR "SECURITY"!

RECAP

EXAMPLE 6: UNDERSTAND THE ACCESS MODIFIERS: PRIVATE, PUBLIC AND PROTECTED

MEMBER FUNCTIONS AND MEMBER VARIABLES CAN BE MARKED AS PUBLIC (IF USABLE OUTSIDE THE CLASS) OR PRIVATE (INTERNAL TO THE CLASS)

A SECTION MARKED `private` THAT IS FOR MEMBER FUNCTIONS AND DATA ACCESSIBLE ONLY INSIDE THE CLASS

IF CODE OUTSIDE THE CLASS TRIES TO ACCESS THIS, A COMPILE ERROR WILL RESULT

BUT ALL OBJECTS OF A CLASS CAN ACCESS EACH OTHERS PRIVATE PARTS

THE ACCESS MODIFIERS EXIST TO FORCE GOOD DESIGN, NOT FOR "SECURITY"!

THERE'S A LOT GOING ON HERE - LET'S TAKE IT SLOW

RECAP

EXAMPLE 6: UNDERSTAND THE ACCESS MODIFIERS: PRIVATE, PUBLIC AND PROTECTED

MEMBER FUNCTIONS AND MEMBER VARIABLES CAN BE MARKED AS PUBLIC (IF USABLE OUTSIDE THE CLASS) OR PRIVATE (INTERNAL TO THE CLASS)

A SECTION MARKED `private` THAT IS FOR MEMBER FUNCTIONS AND DATA ACCESSIBLE ONLY INSIDE THE CLASS

IF CODE OUTSIDE A CLASS TRIES TO ACCESS THIS, A COMPILE ERROR WILL RESULT

BUT ALL OBJECTS OF A CLASS CAN ACCESS EACH OTHERS PRIVATE PARTS

THE ACCESS MODIFIERS EXIST TO FORCE GOOD DESIGN, NOT FOR "SECURITY"!

RECAP

A SECTION MARKED `private` THAT IS FOR MEMBER FUNCTIONS AND DATA ACCESSIBLE ONLY INSIDE THE CLASS

IF CODE OUTSIDE A CLASS TRIES TO ACCESS THIS, A COMPILE ERROR WILL RESULT

```
int main()
{

   ComplexNumber c1(1.414,1.414);
   cout << "Trying to access a private member: " << c1.realPart <<  endl;


class ComplexNumber
{
private:
   float realPart;
   float complexPart;
```

RECAP

A SECTION MARKED `private` THAT IS FOR MEMBER FUNCTIONS AND DATA ACCESSIBLE ONLY INSIDE THE CLASS

```
int main()
{

   ComplexNumber c1(1.414,1.414);
   cout << "Trying to access a private member: " << c1.realPart <<  endl;
```

IF CODE OUTSIDE A CLASS TRIES TO ACCESS THIS, A COMPILE ERROR WILL RESULT

```
Vitthals-MacBook-Pro:~ vitthalsrinivasan$ g++ -Wall Example6.cpp
Example6.cpp:57:55: error: 'realPart' is a private member of 'ComplexNumber'
  cout << "Trying to access a private member: " << c1.realPart <<  endl;
                                                      ^

Example6.cpp:8:9: note: declared private here
  float realPart;
        ^

1 error generated.
```

RECAP

EXAMPLE 6: UNDERSTAND THE ACCESS MODIFIERS: PRIVATE, PUBLIC AND PROTECTED

MEMBER FUNCTIONS AND MEMBER VARIABLES CAN BE MARKED AS PUBLIC (IF USABLE OUTSIDE THE CLASS) OR PRIVATE (INTERNAL TO THE CLASS)

A SECTION MARKED `private` THAT IS FOR MEMBER FUNCTIONS AND DATA ACCESSIBLE ONLY INSIDE THE CLASS

IF CODE OUTSIDE A CLASS TRIES TO ACCESS THIS, A COMPILE ERROR WILL RESULT

BUT ALL OBJECTS OF A CLASS CAN ACCESS EACH OTHERS PRIVATE PARTS

THE ACCESS MODIFIERS EXIST TO FORCE GOOD DESIGN, NOT FOR "SECURITY"!

RECAP

ERRM..ACTUALLY, THERE IS A WAY AROUND THIS

A SECTION MARKED `private` THAT IS FOR MEMBER FUNCTIONS AND DATA ACCESSIBLE ONLY INSIDE THE CLASS

IF CODE OUTSIDE A CLASS TRIES TO ACCESS THIS, A COMPILE ERROR WILL RESULT

IT INVOLVES THE `friend` KEYWORD

THE `friend` KEYWORD

THE IDEA IS PRETTY SIMPLE: ANY CLASS CAN MARK ANOTHER CLASS, OR EVEN A STANDALONE FUNCTION AS A `friend`

THE `friend` KEYWORD

THE IDEA IS PRETTY SIMPLE: ANY CLASS CAN MARK ANOTHER CLASS, OR EVEN A STANDALONE FUNCTION AS A `friend`

THE `friend` KEYWORD

THE IDEA IS PRETTY SIMPLE: ANY CLASS **CAN MARK ANOTHER CLASS**, OR EVEN A STANDALONE FUNCTION AS A `friend`

THE `friend` KEYWORD

THE IDEA IS PRETTY SIMPLE: ANY CLASS CAN MARK ANOTHER CLASS, OR EVEN A STANDALONE FUNCTION AS A `friend`

THE `friend` KEYWORD

THE IDEA IS PRETTY SIMPLE: ANY CLASS CAN MARK ANOTHER CLASS, OR EVEN A STANDALONE FUNCTION AS A

`friend`

THE `friend` KEYWORD

THE IDEA IS PRETTY SIMPLE: ANY CLASS CAN MARK ANOTHER CLASS, OR EVEN A STANDALONE FUNCTION AS A `friend`

THE `friend` KEYWORD

THE IDEA IS PRETTY SIMPLE: ANY CLASS CAN MARK ANOTHER CLASS, OR EVEN A STANDALONE FUNCTION AS A `friend`

THE `friend` CAN THEN ACCESS THE PRIVATE MEMBER DATA OF THAT CLASS

THE `friend` KEYWORD

THE IDEA IS PRETTY SIMPLE: ANY CLASS CAN MARK ANOTHER CLASS, OR EVEN A STANDALONE FUNCTION AS A `friend`

THE `friend` CAN THEN ACCESS THE PRIVATE MEMBER DATA OF THAT CLASS

THERE IS NO COMPULSION THAT `friend`-SHIP BE MUTUAL, AND IT RARELY IS

JOKES ABOUT THE `friend` KEYWORD ARE UNIVERSAL

"THERE IS NO COMPULSION THAT `friend`-SHIP BE MUTUAL, AND IT RARELY IS"

"A `friend` CAN TOUCH YOUR PRIVATE PARTS"

"LIKE IN REAL LIFE, `friends` ARE USUALLY MORE TROUBLE THAN THEY ARE WORTH"

JOKES ABOUT THE `friend` KEYWORD ARE UNIVERSAL

"THERE IS NO COMPULSION THAT `friend`-SHIP BE MUTUAL AND IT RARELY IS"

"A FRIEND CAN TOUCH YOUR KIDNEY"

USING `friend` IS QUITE CONTROVERSIAL - IT BREAKS ENCAPSULATION, AND OTHER LANGUAGES DON'T INCLUDE SUCH AN IDEA, BUT..

"LIKE IN REAL LIFE, `friends` ARE USUALLY MORE TROUBLE THAN THEY ARE WORTH"

JOKES ABOUT THE `friend`
KEYWORD ARE UNIVERSAL

"THERE IS NO GUARANTEE THAT FRIENDSHIP BE MUTUAL, AND IT RARELY IS"

"A FRIEND CAN TOUCH YOUR PRIVATE PARTS"

THERE ARE SPECIFIC SITUATIONS WHERE `friend` IS A LIFE-SAVER (OPERATOR OVERLOADING OF CERTAIN OPERATORS, FOR INSTANCE)

"LIKE IN REAL LIFE, `friends` ARE USUALLY MORE TROUBLE THAN THEY ARE WORTH"

THERE ARE SPECIFIC SITUATIONS WHERE `friend` IS A LIFE-SAVER (OPERATOR OVERLOADING OF CERTAIN OPERATORS, FOR INSTANCE)

MORE ON THIS WHEN WE GET TO OPERATOR OVERLOADING

EXAMPLE 42

DECLARE A FRIEND FUNCTION, AND USE IT

EXAMPLE 42 DECLARE A FRIEND FUNCTION, AND USE IT

THIS IS A COMPLICATED EXAMPLE,
WITH 2 CLASSES, AND 5 FILES!

THIS IS A COMPLICATED EXAMPLE, WITH 2 CLASSES, AND 5 FILES!

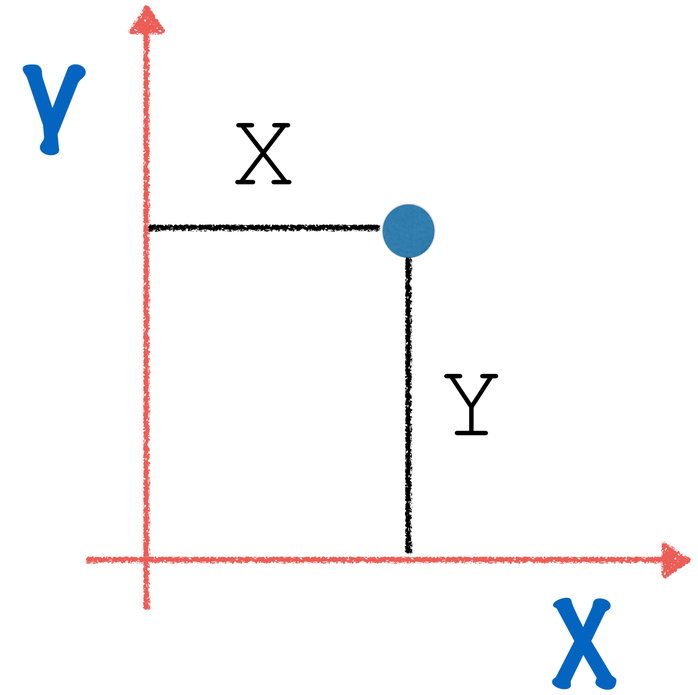
CLASS #1: A COMPLEX NUMBER CLASS (CARTESIAN COORDINATES)

$$z = x + iy$$

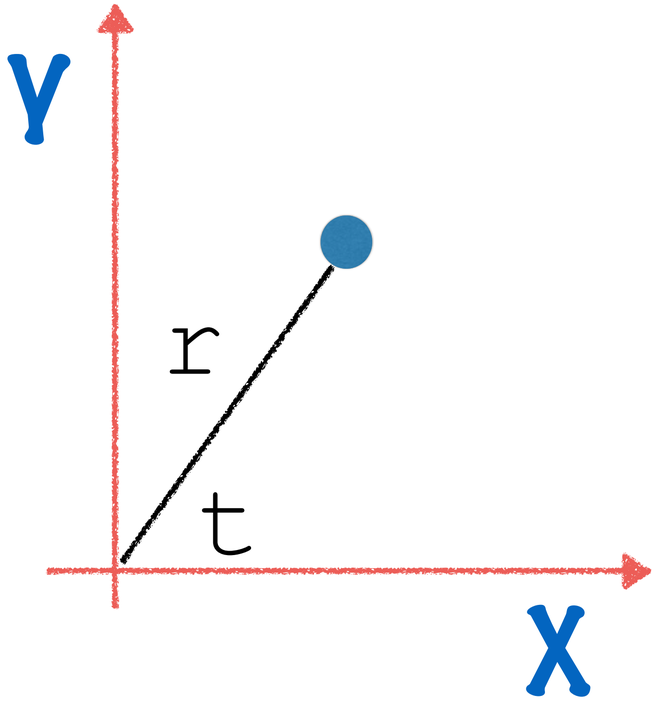
CLASS #2: A COMPLEX NUMBER CLASS (POLAR COORDINATES)

$$z = r(\cos t + i \sin t)$$

THIS IS A COMPLICATED EXAMPLE, WITH 2 CLASSES, AND 5 FILES!



$$z = x + iy$$



$$z = r(\cos t + i \sin t)$$

THIS IS A COMPLICATED EXAMPLE, WITH 2 CLASSES, AND 5 FILES!

CLASS #1: A COMPLEX NUMBER CLASS (CARTESIAN COORDINATES)

```
z = x + iy
```

```
realPart (x),
imaginaryPart (Y)
```

CLASS #2: A COMPLEX NUMBER CLASS (POLAR COORDINATES)

```
z = r(cos t + i sin t)
```

```
modulus (r),
argument (t)
```

TWO MEMBER VARIABLES IN EACH CLASS

THIS IS A COMPLICATED EXAMPLE, WITH 2 CLASSES, AND 5 FILES!

CLASS #1: A COMPLEX NUMBER CLASS (CARTESIAN COORDINATES)

CLASS #2: A COMPLEX NUMBER CLASS (POLAR COORDINATES)

SEPARATE THE DECLARATIONS AND DEFINITIONS INTO .CPP FILE, .H FILE

AND THE MAIN FUNCTION IN A SEPARATE .CPP FILE

THIS IS A COMPLICATED EXAMPLE, WITH 2 CLASSES, AND 5 FILES!

Example42_ComplexNumber.h

Example42_ComplexNumber.cpp

Example42_ComplexNumber_Polar.h

Example42_ComplexNumber_Polar.cpp

Example42_Main.cpp

THIS IS A COMPLICATED EXAMPLE, WITH 2 CLASSES, AND 5 FILES!

Example42_**ComplexNumber.h** DECLARATION OF THE CARTESIAN

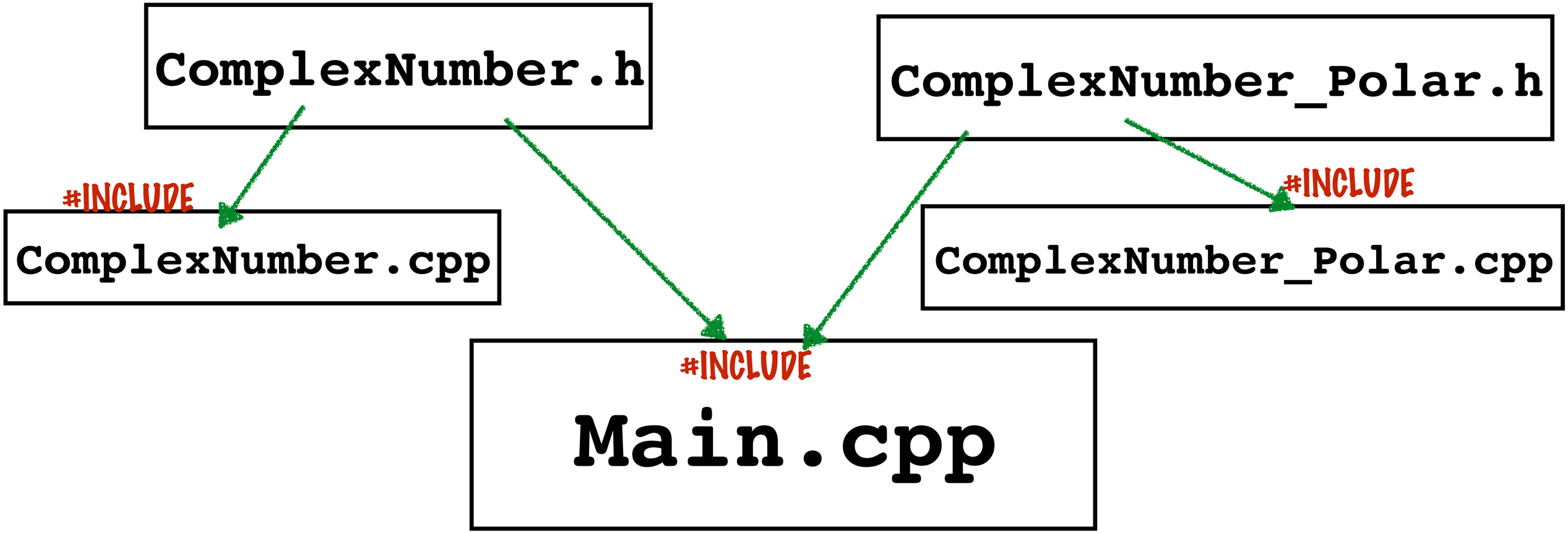
Example42_**ComplexNumber.cpp** DEFINITION OF THE CARTESIAN

Example42_**ComplexNumber_Polar.h** DECLARATION OF THE POLAR

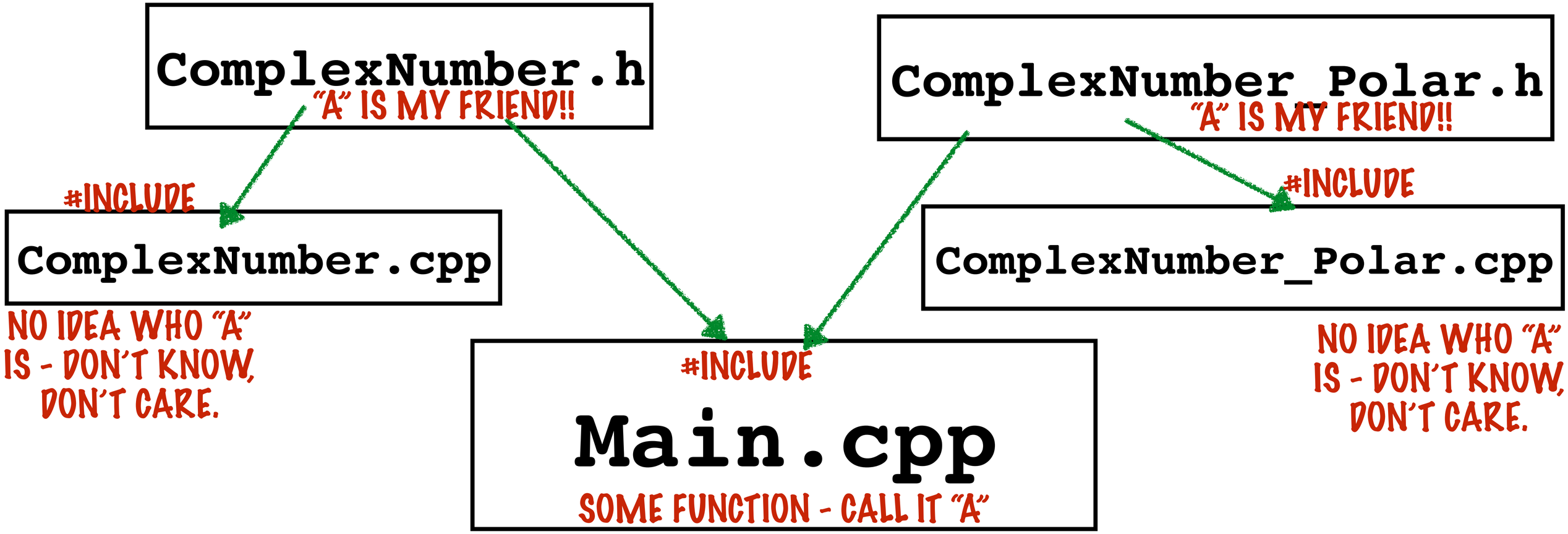
Example42_**ComplexNumber_Polar.cpp** DEFINITION OF THE POLAR

Example42_**Main.cpp**

THIS IS A COMPLICATED EXAMPLE, WITH 2 CLASSES, AND 5 FILES!



THIS IS A COMPLICATED EXAMPLE, WITH 2 CLASSES, AND 5 FILES!



Main.cpp

```cpp
include <iostream>
#include <cmath>

#include "Example42_ComplexNumber.h"
#include "Example42_ComplexNumber_Polar.h"


ComplexNumber addTwoComplexNumbers
  (const ComplexNumber& cart,
   const ComplexNumber_Polar& polar)
{
  float realPart = polar.modulus * cos(polar.argument * ComplexNumber::PI/180);
  float complexPart = polar.modulus * sin(polar.argument * ComplexNumber::PI/180);

  ComplexNumber result(realPart + cart.realPart, complexPart + cart.complexPart);
  return result;

}


int main()
{

  ComplexNumber cart(5,5);
  ComplexNumber_Polar polar(10,45);
  ComplexNumber sumOfTwoComplexNumbers = addTwoComplexNumbers(cart,polar);
  cout << "Real part = " << sumOfTwoComplexNumbers.getRealPart()
       << "Complex part = " << sumOfTwoComplexNumbers.getComplexPart() << endl;
}
```

Main.cpp

#INCLUDE BOTH THE .H FILES

```
include <iostream>
#include <cmath>

#include "Example42_ComplexNumber.h"
#include "Example42_ComplexNumber_Polar.h"

ComplexNumber addTwoComplexNumbers
  (const ComplexNumber& cart,
   const ComplexNumber_Polar& polar)
{
  float realPart = polar.modulus * cos(polar.argument * ComplexNumber::PI/180);
  float complexPart = polar.modulus * sin(polar.argument * ComplexNumber::PI/180);

  ComplexNumber result(realPart + cart.realPart, complexPart + cart.complexPart);
  return result;

}


int main()
{

  ComplexNumber cart(5,5);
  ComplexNumber_Polar polar(10,45);
  ComplexNumber sumOfTwoComplexNumbers = addTwoComplexNumbers(cart,polar);
  cout << "Real part = " << sumOfTwoComplexNumbers.getRealPart()
       << "Complex part = " << sumOfTwoComplexNumbers.getComplexPart() << endl;
}
```

```
include <iostream>
#include <cmath>

#include "Example42_ComplexNumber.h"
#include "Example42_ComplexNumber_Polar.h"
```

Main.cpp

```
ComplexNumber addTwoComplexNumbers
  (const ComplexNumber& cart,
   const ComplexNumber_Polar& polar)
{
  float realPart = polar.modulus * cos(polar.argument * ComplexNumber::PI/180);
  float complexPart = polar.modulus * sin(polar.argument * ComplexNumber::PI/180);

  ComplexNumber result(realPart + cart.realPart, complexPart + cart.complexPart);
  return result;

}
```

THIS IS A FUNCTION THAT ADDS TWO COMPLEX NUMBER OBJECTS - ONE OF EACH TYPE

```
int main()
{

  ComplexNumber cart(5,5);
  ComplexNumber_Polar polar(10,45);
  ComplexNumber sumOfTwoComplexNumbers = addTwoComplexNumbers(cart,polar);
  cout << "Real part = " << sumOfTwoComplexNumbers.getRealPart()
       << "Complex part = " << sumOfTwoComplexNumbers.getComplexPart() << endl;
}
```

Main.cpp

```
include <iostream>
#include <cmath>

#include "Example42_ComplexNumber.h"
#include "Example42_ComplexNumber_Polar.h"


ComplexNumber addTwoComplexNumbers
   (const ComplexNumber& cart,
    const ComplexNumber_Polar& polar)
{
   float realPart = polar.modulus * cos(polar.argument * ComplexNumber::PI/180);
   float complexPart = polar.modulus * sin(polar.argument * ComplexNumber::PI/180);

   ComplexNumber result(realPart + cart.realPart, complexPart + cart.complexPart);
   return result;

}


int main()
{

   ComplexNumber cart(5,5);
   ComplexNumber_Polar polar(10,45);
   ComplexNumber sumOfTwoComplexNumbers = addTwoComplexNumbers(cart,polar);
   cout << "Real part = " << sumOfTwoComplexNumbers.getRealPart()
        << "Complex part = " << sumOfTwoComplexNumbers.getComplexPart() << endl;
}
```

THIS IS A FUNCTION THAT ADDS TWO COMPLEX NUMBER OBJECTS - ONE OF EACH TYPE

Main.cpp

```
include <iostream>
#include <cmath>

#include "Example42_ComplexNumber.h"
#include "Example42_ComplexNumber_Polar.h"


ComplexNumber addTwoComplexNumbers
  (const ComplexNumber& cart,
   const ComplexNumber_Polar& polar)
{
  float realPart = polar.modulus * cos(polar.argument * ComplexNumber::PI/180);
  float complexPart = polar.modulus * sin(polar.argument * ComplexNumber::PI/180);

  ComplexNumber result(realPart + cart.realPart, complexPart + cart.complexPart);
  return result;

}


int main()
{

  ComplexNumber cart(5,5);
  ComplexNumber_Polar polar(10,45);
  ComplexNumber sumOfTwoComplexNumbers = addTwoComplexNumbers(cart,polar);
  cout << "Real part = " << sumOfTwoComplexNumbers.getRealPart()
       << "Complex part = " << sumOfTwoComplexNumbers.getComplexPart() << endl;
}
```

THIS FUNCTION ACCESSES THE PRIVATE MEMBER DATA OF BOTH OF THOSE CLASSES

Main.cpp

```
include <iostream>
#include <cmath>

#include "Example42_ComplexNumber.h"
#include "Example42_ComplexNumber_Polar.h"


ComplexNumber addTwoComplexNumbers
  (const ComplexNumber& cart,
   const ComplexNumber_Polar& polar)
{
  float realPart = polar.modulus * cos(polar.argument * ComplexNumber::PI/180);
  float complexPart = polar.modulus * sin(polar.argument * ComplexNumber::PI/180);

  ComplexNumber result(realPart + cart.realPart, complexPart + cart.complexPart);
  return result;

}


int main()
{

  ComplexNumber cart(5,5);
  ComplexNumber_Polar polar(10,45);
  ComplexNumber sumOfTwoComplexNumbers = addTwoComplexNumbers(cart,polar);
  cout << "Real part = " << sumOfTwoComplexNumbers.getRealPart()
       << "Complex part = " << sumOfTwoComplexNumbers.getComplexPart() << endl;
}
```

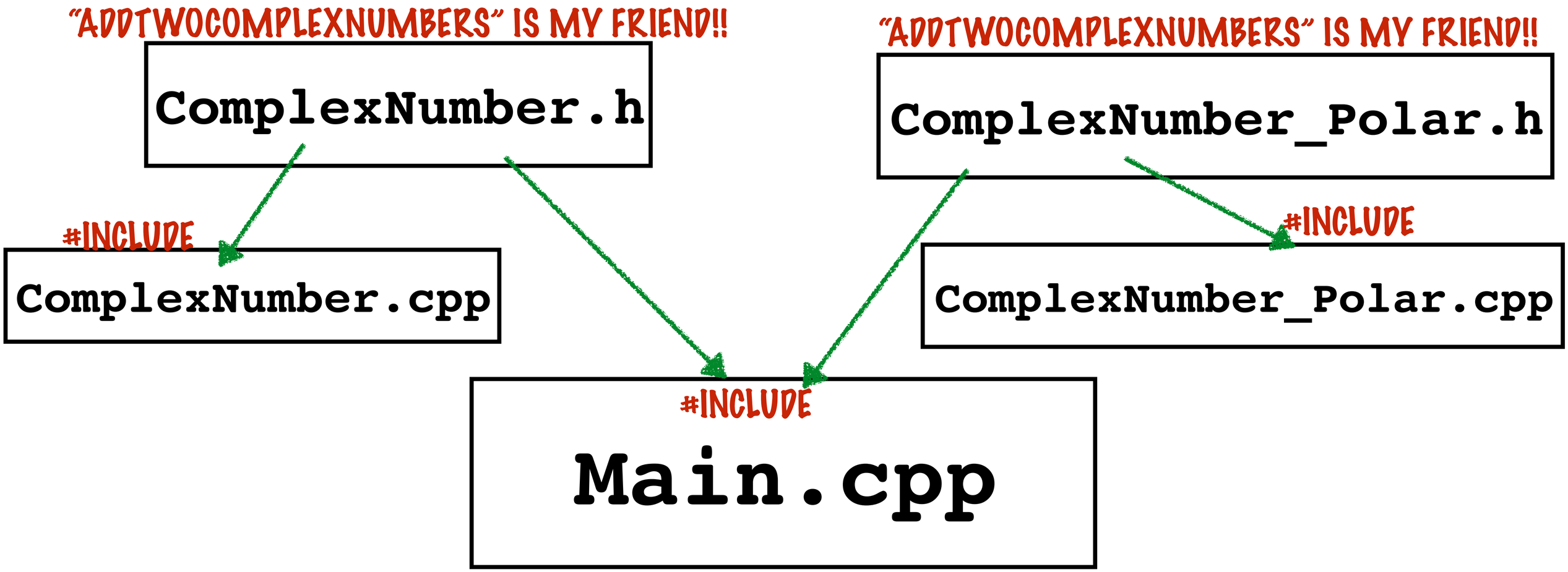
THIS FUNCTION ACCESSES THE PRIVATE MEMBER DATA OF BOTH OF THOSE CLASSES

Main.cpp

```
include <iostream>
#include <cmath>

#include "Example42_ComplexNumber.h"
#include "Example42_ComplexNumber_Polar.h"


ComplexNumber addTwoComplexNumbers
  (const ComplexNumber& cart,
   const ComplexNumber_Polar& polar)
{
  float realPart = polar.modulus * cos(polar.argument * ComplexNumber::PI/180);
  float complexPart = polar.modulus * sin(polar.argument * ComplexNumber::PI/180);

  ComplexNumber result(realPart + cart.realPart, complexPart + cart.complexPart);
  return result;

}


int main()
{

  ComplexNumber cart(5,5);
  ComplexNumber_Polar polar(10,45);
  ComplexNumber sumOfTwoComplexNumbers = addTwoComplexNumbers(cart,polar);
  cout << "Real part = " << sumOfTwoComplexNumbers.getRealPart()
       << "Complex part = " << sumOfTwoComplexNumbers.getComplexPart() << endl;
}
```

AND SO, EACH OF THOSE 2 CLASSES MUST DECLARE THIS FUNCTION TO BE A FRIEND

THIS IS A COMPLICATED EXAMPLE, WITH 2 CLASSES, AND 5 FILES!



Complex Number.h

```cpp
#include <iostream>

using namespace std;

class ComplexNumber_Polar;

class ComplexNumber
{
private:
  float realPart;
  float complexPart;
  static int numObjectsCreated;
public:
  const static double PI;

  ComplexNumber();

  ComplexNumber(double c, double r);

  ComplexNumber(const ComplexNumber& rhs);

  ~ComplexNumber();

  float getRealPart() const;

  void  setRealPart(float r);


  float getComplexPart() const;
  void  setComplexPart(float c);

friend ComplexNumber addTwoComplexNumbers
  (const ComplexNumber& cart,
   const ComplexNumber_Polar& polar);

};
```

THE .H FILE NOW CONTAINS ONLY THE DECLARATIONS, NOT THE DEFINITIONS (THOSE ARE IN THE .CPP FILE)

Complex Number.h

```
#include <iostream>

using namespace std;

class ComplexNumber_Polar;

class ComplexNumber
{
private:
  float realPart;
  float complexPart;
  static int numObjectsCreated;
public:
  const static double PI;

  ComplexNumber();

  ComplexNumber(double c, double r);

  ComplexNumber(const ComplexNumber& rhs);

  ~ComplexNumber();

  float getRealPart() const;

  void  setRealPart(float r);


  float getComplexPart() const;
  void  setComplexPart(float c);

friend ComplexNumber addTwoComplexNumbers
   (const ComplexNumber& cart,
    const ComplexNumber_Polar& polar);

};
```

THIS IS WHERE THE DECLARATION OF **friend**-SHIP MUST RESIDE

Complex Number.h

```
#include <iostream>

using namespace std;

class ComplexNumber_Polar;

class ComplexNumber
{
private:
  float realPart;
  float complexPart;
  static int numObjectsCreated;
public:
  const static double PI;

  ComplexNumber();

  ComplexNumber(double c, double r);

  ComplexNumber(const ComplexNumber& rhs);

  ~ComplexNumber();

  float getRealPart() const;

  void  setRealPart(float r);


  float getComplexPart() const;
  void  setComplexPart(float c);

friend ComplexNumber addTwoComplexNumbers
   (const ComplexNumber& cart,
    const ComplexNumber_Polar& polar);

};
```

THIS IS WHERE THE DECLARATION OF **friend**-SHIP MUST RESIDE

Complex Number.h

```
#include <iostream>

using namespace std;

class ComplexNumber_Polar;

class ComplexNumber
{
private:
  float realPart;
  float complexPart;
  static int numObjectsCreated;
public:
  const static double PI;

  ComplexNumber();

  ComplexNumber(double c, double r);

  ComplexNumber(const ComplexNumber& rhs);

  ~ComplexNumber();

  float getRealPart() const;

  void  setRealPart(float r);


  float getComplexPart() const;
  void  setComplexPart(float c);

friend ComplexNumber addTwoComplexNumbers
   (const ComplexNumber& cart,
    const ComplexNumber_Polar& polar);

};
```

WAIT! HOW WILL OUR CLASS KNOW ANYTHING ABOUT THIS OTHER OBJECT?

ComplexNumber.h

```
#include <iostream>

using namespace std;

class ComplexNumber_Polar;

class ComplexNumber
{
private:
  float realPart;
  float complexPart;
  static int numObjectsCreated;
public:
  const static double PI;

  ComplexNumber();

  ComplexNumber(double c, double r);

  ComplexNumber(const ComplexNumber& rhs);

  ~ComplexNumber();

  float getRealPart() const;

  void  setRealPart(float r);


  float getComplexPart() const;
  void  setComplexPart(float c);

  friend ComplexNumber addTwoComplexNumbers
    (const ComplexNumber& cart,
     const ComplexNumber_Polar& polar);
};
```

ESPECIALLY SINCE WE DON'T #include ITS DECLARATION?

WAIT! HOW WILL OUR CLASS KNOW ANYTHING ABOUT THIS OTHER OBJECT?

Complex Number.h

```
#include <iostream>

using namespace std;

class ComplexNumber_Polar;

class ComplexNumber
{
  float realPart;
  float complexPart;
  static double radiansPerDegree;
public:
  const static double PI;

  ComplexNumber();

  ComplexNumber(double r, double c);

  ComplexNumber(const ComplexNumber& rhs);

  ~ComplexNumber();

  float getRealPart() const;

  void  setRealPart(float r);


  float getComplexPart() const;
  void  setComplexPart(float c);

  friend ComplexNumber addTwoComplexNumbers
    (const ComplexNumber& cart,
     const ComplexNumber_Polar& polar);

};
```

AHA! THAT'S WHY WE HAVE A FORWARD DECLARATION OF THAT CLASS!

WAIT! HOW WILL OUR CLASS KNOW ANYTHING ABOUT THIS OTHER OBJECT?

ESPECIALLY SINCE WE DON'T #include ITS DECLARATION?

FORWARD DECLARATION

THIS IS AN IMPORTANT POINT - REMEMBER TO ADD ANY FORWARD DECLARATIONS FOR TYPES THAT ARE NOT #-INCLUDED IN YOUR FILE

THINK OF THIS AS A PLACEHOLDER CLASS SO THE COMPILER DOES NOT COMPLAIN, YOU KNOW THE REAL CLASS IS GOING TO BE DEFINED AND DECLARED ELSEWHERE

FORWARD DECLARATION

THIS IS AN IMPORTANT POINT - **REMEMBER TO ADD ANY FORWARD DECLARATIONS** FOR TYPES THAT ARE NOT #-INCLUDED IN YOUR FILE

THINK OF THIS AS A **PLACEHOLDER CLASS** SO THE COMPILER DOES NOT COMPLAIN, YOU KNOW THE REAL CLASS IS GOING TO BE DEFINED AND DECLARED ELSEWHERE

FORWARD DECLARATION

THIS IS AN IMPORTANT POINT - REMEMBER TO ADD ANY FORWARD DECLARATIONS **FOR TYPES THAT ARE NOT #-INCLUDED IN YOUR FILE**

THINK OF THIS AS A PLACEHOLDER CLASS SO THE COMPILER DOES NOT COMPLAIN, YOU KNOW THE **REAL CLASS IS GOING TO BE DEFINED AND DECLARED ELSEWHERE**

Complex Number.h

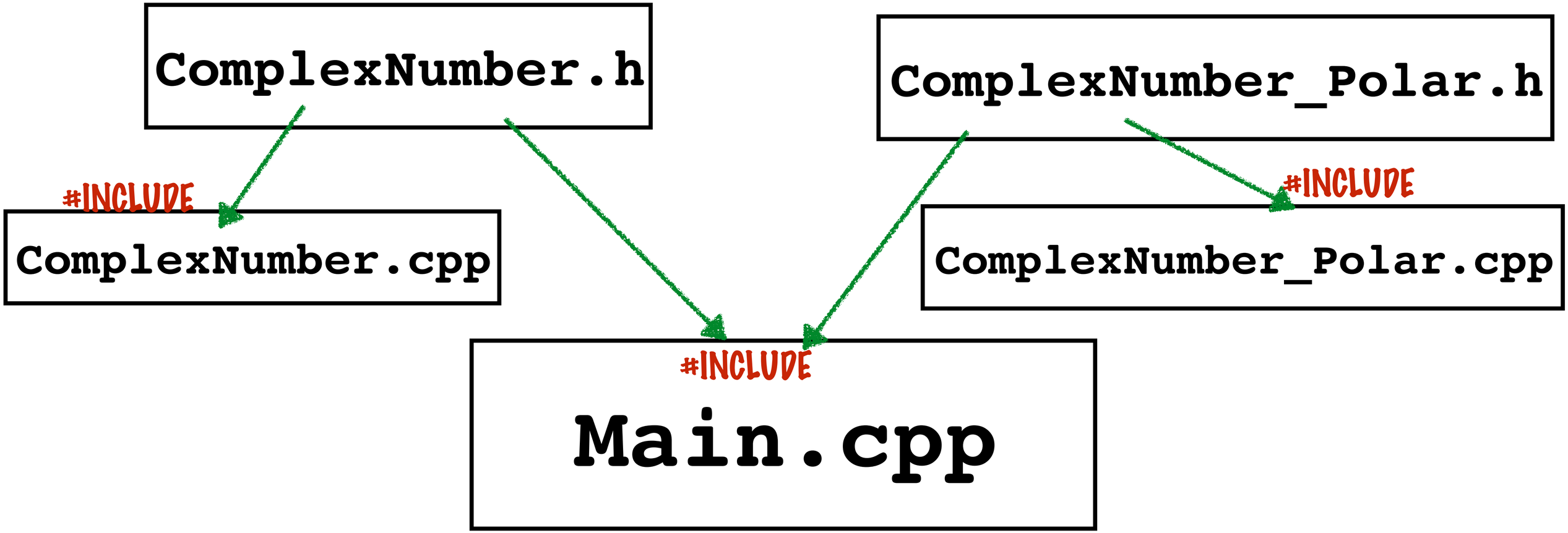
```
#include <iostream>

using namespace std;

class ComplexNumber_Polar;

class ComplexNumber
{
private:
  float realPart;
  float complexPart;
  static double objectsCreated;
public:
  const static double PI;

  ComplexNumber();

  ComplexNumber(double r, double c);

  ComplexNumber(const ComplexNumber& rhs);

  ~ComplexNumber();

  float getRealPart() const;

  void  setRealPart(float r);


  float getComplexPart() const;
  void  setComplexPart(float c);

  friend ComplexNumber addTwoComplexNumbers
    (const ComplexNumber& cart,
     const ComplexNumber_Polar& polar);

};
```

AHA! THAT'S WHY WE HAVE A FORWARD DECLARATION OF THAT CLASS!

WAIT! HOW WILL OUR CLASS KNOW ANYTHING ABOUT THIS OTHER OBJECT?

ESPECIALLY SINCE WE DON'T #include ITS DECLARATION?

THIS IS A COMPLICATED EXAMPLE, WITH 2 CLASSES, AND 5 FILES!



Complex Number. cpp

```cpp
#include <iostream>
#include "Example42_ComplexNumber.h"

using namespace std;

int ComplexNumber::numObjectsCreated = 0; // define the static variable
const double ComplexNumber::PI = 3.1415;


ComplexNumber::ComplexNumber() : realPart(0.0),complexPart(0.0)
{
  // increment the static variable keeping track of objects created                    \

  numObjectsCreated++;
  cout << "No arg-constructor called" << endl;
}

ComplexNumber::ComplexNumber(double c, double r) : realPart(r) , complexPart(c)
{
  // increment the static variable keeping track of objects created                    \

  numObjectsCreated++;
  cout << "Inside the 2-argument constructor" << endl;
}

ComplexNumber::ComplexNumber(const ComplexNumber& rhs) :
  realPart(rhs.realPart), complexPart(rhs.complexPart)
{
  // increment the static variable keeping track of objects created                    \

  numObjectsCreated++;
  cout << "Inside the copy constructor" << endl;
}

ComplexNumber::~ComplexNumber()
{
  cout << "Inside the destructor: realPart = " << realPart <<" complexPart = " << complexPart << endl;
}

float ComplexNumber::getRealPart() const { return realPart; }
void  ComplexNumber::setRealPart(float r) { realPart = r;}

float ComplexNumber::getComplexPart() const { return complexPart; }
void  ComplexNumber::setComplexPart(float c) { complexPart = c;}
```

THE .CPP FILE SIMPLY INCLUDES ALL THE DEFINITIONS

Complex Number.cpp

```
#include "Example42_ComplexNumber.h"
```

THE .CPP FILE MUST INCLUDE THE .H FILE OF COURSE

THE .CPP FILE SIMPLY INCLUDES ALL THE DEFINITIONS

```cpp
int ComplexNumber::numObjectsCreated = 0; // define the static variable
const double ComplexNumber::PI = 3.1415;

ComplexNumber::ComplexNumber()
```

ComplexNumber.cpp

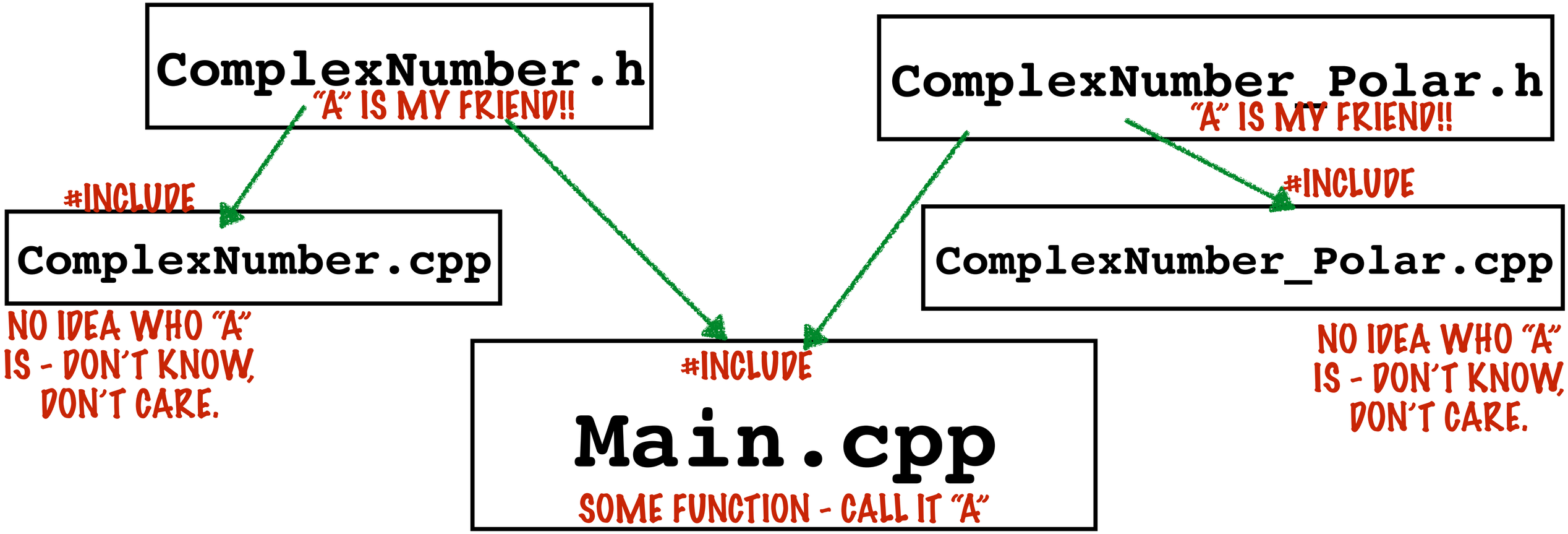
THE .CPP FILE SIMPLY INCLUDES ALL THE DEFINITIONS

```cpp
ComplexNumber::ComplexNumber(double c, double r)
```

ALL DEFINITIONS USE THE SCOPE RESOLUTION OPERATOR OF COURSE

```cpp
ComplexNumber::ComplexNumber(const ComplexNumber& rhs)
```

THIS IS A COMPLICATED EXAMPLE, WITH 2 CLASSES, AND 5 FILES!



THIS IS A COMPLICATED EXAMPLE, WITH 2 CLASSES, AND 5 FILES!

```
Vitthals-MacBook-Pro:~ vitthalsrinivasan$ g++ -Wall Example42*.cpp
```

BTW, TO COMPILE A PROJECT WITH MANY FILES USING G++, ONLY COMPILE THE CPP FILES

THIS IS A COMPLICATED EXAMPLE, WITH 2 CLASSES, AND 5 FILES!

```
Vitthals-MacBook-Pro:~ vitthalsrinivasan$ g++ -Wall Example42*.cpp
```

BTW, TO COMPILE A PROJECT WITH MANY FILES USING G++, ONLY COMPILE THE CPP FILES

EXAMPLE 43

DECLARE A FRIEND CLASS, AND USE IT

THE `friend` KEYWORD

THE IDEA IS PRETTY SIMPLE: ANY CLASS CAN MARK ANOTHER CLASS, OR EVEN A STANDALONE FUNCTION AS A `friend`

RECAP

THE `friend` KEYWORD

THE IDEA IS PRETTY SIMPLE: ANY CLASS CAN MARK ANOTHER CLASS, OR EVEN A STANDALONE FUNCTION AS A `friend`

RECAP

THE `friend` KEYWORD

THE IDEA IS PRETTY SIMPLE: ANY CLASS CAN MARK ANOTHER CLASS, OR EVEN A STANDALONE FUNCTION AS A FRIEND. EVERY MEMBER FUNCTION OF THAT CLASS BECOMES A FRIEND

RECAP

THE `friend` KEYWORD

WHEN YOU DECLARE AN ENTIRE CLASS TO BE A `friend` -

EVERY MEMBER FUNCTION OF THAT CLASS BECOMES A FRIEND

WHEN YOU DECLARE AN ENTIRE CLASS TO BE A `friend` -

EVERY MEMBER FUNCTION OF THAT CLASS BECOMES A FRIEND

LET'S SAY THAT THESE CLASSES ARE DECLARED IN THE SAME FILE

```
class Student
{
private:
  string studentName;
public:
  Student(const char* name) : studentName(name)
  {
    cout << "Initialized string to: " << studentName << endl;
  }


};
```

CLASS STUDENT

```
class School
{
private:
  string schoolName;
public:
  School(const char* name) : schoolName(name)
  {
    cout << "Initialized string to: " << schoolName << endl;
  }

  void admitStudent(const Student& s)
  {
    cout << "Admitting student " << s.studentName << endl;
  }

  void expelStudent(const Student& s)
  {
    cout << "Expelling student " << s.studentName << endl;
  }


};
```

CLASS SCHOOL

WHEN YOU DECLARE AN ENTIRE CLASS TO BE A `friend` -

EVERY MEMBER FUNCTION OF THAT CLASS BECOMES A FRIEND

CLASS STUDENT HAS A PRIVATE MEMBER VARIABLE

```
class Student
{
private:
   string studentName;
public:
   Student(const char* name) : studentName(name)
   {
      cout << "Initialized string to: " << studentName << endl;
   }


};
```

CLASS STUDENT

WHEN YOU DECLARE AN ENTIRE CLASS TO BE A `friend` -

EVERY MEMBER FUNCTION OF THAT CLASS BECOMES A FRIEND

CLASS STUDENT HAS A PRIVATE MEMBER VARIABLE

```
class Student
{
private:
  string studentName;
public:
  Student(const char* name) : studentName(name)
  {
    cout << "Initialized string to: " << studentName << endl;
  }


};
```

CLASS STUDENT

THAT CLASS SCHOOL ATTEMPTS TO ACCESS

```
class School
{
private:
  string schoolName;
public:
  School(const char* name) : schoolName(name)
  {
    cout << "Initialized string to: " << schoolName << endl;
  }

  void admitStudent(const Student& s)
  {
    cout << "Admitting student " << s.studentName << endl;
  }

  void expelStudent(const Student& s)
  {
    cout << "Expelling student " << s.studentName << endl;
  }


};
```

CLASS SCHOOL

WHEN YOU DECLARE AN ENTIRE CLASS TO BE A `friend` -

EVERY MEMBER FUNCTION OF THAT CLASS BECOMES A FRIEND

```
class School
{
private:
  string schoolName;
public:
  School(const char* name) : schoolName(name)
  {
    cout << "Initialized string to: " << schoolName << endl;
  }

  void admitStudent(const Student& s)
  {
    cout << "Admitting student " << s.studentName << endl;
  }

  void expelStudent(const Student& s)
  {
    cout << "Expelling student " << s.studentName << endl;
  }
```

CLASS STUDENT HAS A PRIVATE MEMBER VARIABLE

THAT CLASS SCHOOL ATTEMPTS TO ACCESS

CLASS SCHOOL

WHEN YOU DECLARE AN ENTIRE CLASS TO BE A `friend` -

EVERY MEMBER FUNCTION OF THAT CLASS BECOMES A FRIEND

```
class School
{
private:
  string schoolName;
public:
  School(const char* name) : schoolName(name)
  {
    cout << "Initialized string to: " << schoolName << endl;
  }

  void admitStudent(const Student& s)
  {
    cout << "Admitting student " << s.studentName << endl;
  }

  void expelStudent(const Student& s)
  {
    cout << "Expelling student " << s.studentName << endl;
  }
```

CLASS STUDENT HAS A PRIVATE MEMBER VARIABLE

THAT CLASS SCHOOL ATTEMPTS TO ACCESS

CLASS SCHOOL

THE RESULT IS AN ERROR

```
Vitthals-MacBook-Pro:~ vitthalsrinivasan$ g++ -Wall Example43*.cpp
Example43.cpp:33:39: error: 'studentName' is a private member of 'Student'
    cout << "Admitting student " << s.studentName << endl;
                                      ^

Example43.cpp:11:10: note: declared private here
  string studentName;
         ^

Example43.cpp:38:39: error: 'studentName' is a private member of 'Student'
    cout << "Expelling student " << s.studentName << endl;
                                      ^

Example43.cpp:11:10: note: declared private here
  string studentName;
         ^

2 errors generated.
```

WHICH IS EASILY FIXED BY HAVING STUDENT DECLARE THAT SCHOOL IS A FRIEND!

WHICH IS EASILY FIXED BY HAVING STUDENT DECLARE THAT SCHOOL IS A FRIEND!

```
class Student
{
private:
  string studentName;
public:
  Student(const char* name) : studentName(name)
  {
    cout << "Initialized string to: " << studentName << endl;
  }
  friend class School;
};
```

WHICH IS EASILY FIXED BY HAVING STUDENT DECLARE THAT SCHOOL IS A FRIEND!

```
class Student
{
private:
  string studentName;
public:
  Student(const char* name) : studentName(name)
  {
    cout << "Initialized string to: " << studentName << endl;
  }
  friend class School;
};
```

IN THIS PARTICULAR INSTANCE, NO FORWARD DECLARATION IS NEEDED BECAUSE THE CLASSES ARE IN THE SAME FILE

THE CODE COMPILES AND RUNS FINE NOW!

```
Vitthals-MacBook-Pro:~ vitthalsrinivasan$ g++ -Wall Example43*.cpp
Vitthals-MacBook-Pro:~ vitthalsrinivasan$ ./a.out
 Initialized string to: Vitthal
 Initialized string to: St Johns
Admitting student Vitthal
Expelling student Vitthal
```

NOTE THAT BOTH MEMBER FUNCTIONS OF THE SCHOOL CLASS COULD NOW ACCESS THE PRIVATE PARTS OF STUDENT

THE `friend` KEYWORD

WHEN YOU DECLARE AN ENTIRE CLASS TO BE A `friend` -

EVERY MEMBER FUNCTION OF THAT CLASS BECOMES A FRIEND