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

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

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

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()
```

NOTE THE USE OF THE KEYWORD `public`

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

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()
```

NOTE THE USE OF THE KEYWORD `public`

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

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

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

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()
```

NOTE THE USE OF THE KEYWORD `private`

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

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)

NOTE THE USE OF THE KEYWORD `private`

```
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

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

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"!

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

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

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

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"!

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;
```

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.
```

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"!

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"!

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

LET'S UNDERSTAND THIS VIA SOMETHING CALLED

THE COPY CONSTRUCTOR

```
ComplexNumber c1(1.414,1.414);
ComplexNumber c2 = c1;
```

THE COPY CONSTRUCTOR

```
ComplexNumber c1(1.414,1.414);
ComplexNumber c2 = c1;
```

C++ USES THE COPY CONSTRUCTOR TO CREATE ("COPY") ONE OBJECT FROM ANOTHER

LIKE THE DEFAULT (NO-ARG) CONSTRUCTOR, C++ WILL CREATE ONE EVEN IF YOU DON'T

THE COPY CONSTRUCTOR

C++ USES THE COPY CONSTRUCTOR TO CREATE ("COPY") ONE OBJECT FROM ANOTHER

LIKE THE DEFAULT (NO-ARG) CONSTRUCTOR, C++ WILL CREATE ONE EVEN IF YOU DON'T

BUT WE CAN ALWAYS CREATE OUR OWN COPY CONSTRUCTOR - LET'S SEE HOW!

THE COPY CONSTRUCTOR

C++ USES THE COPY CONSTRUCTOR TO CREATE ("COPY") ONE OBJECT FROM ANOTHER

LIKE THE DEFAULT (NO-ARG) CONSTRUCTOR, C++ WILL CREATE ONE EVEN IF YOU DON'T

BUT WE CAN ALWAYS CREATE OUR OWN COPY CONSTRUCTOR - LET'S SEE HOW!

```
ComplexNumber(const ComplexNumber& rhs) :
    realPart(rhs.realPart), complexPart(rhs.complexPart)
{
  cout << "Inside the copy constructor" << endl;
}
```

BUT WE CAN ALWAYS CREATE OUR OWN COPY CONSTRUCTOR - LET'S SEE HOW!

```
ComplexNumber(const ComplexNumber& rhs) :
    realPart(rhs.realPart), complexPart(rhs.complexPart)
{
  cout << "Inside the copy constructor" << endl;
}
```

A LOT MORE ON THIS LATER, BUT FOR NOW - FOCUS ON THE INITIALISATION LIST

BUT WE CAN ALWAYS CREATE OUR OWN COPY CONSTRUCTOR - LET'S SEE HOW!

```
ComplexNumber(const ComplexNumber& rhs) :
    realPart(rhs.realPart), complexPart(rhs.complexPart)
{
  cout << "Inside the copy constructor" << endl;
}
```

A LOT MORE ON THIS LATER, BUT FOR NOW - FOCUS ON THE INITIALISATION LIST

WE ARE ACCESSING THE PRIVATE MEMBER VARIABLES OF THE OBJECT `rhs`

BUT WE CAN ALWAYS CREATE OUR OWN COPY CONSTRUCTOR - LET'S SEE HOW!

```
ComplexNumber(const ComplexNumber& rhs) :
    realPart(rhs.realPart), complexPart(rhs.complexPart)
{
  cout << "Inside the copy constructor" << endl;
}
```

A LOT MORE ON THIS LATER, BUT FOR NOW - FOCUS ON THE INITIALISATION LIST

WE ARE ACCESSING THE PRIVATE MEMBER VARIABLES OF THE OBJECT `rhs`

BUT WE CAN ALWAYS CREATE OUR OWN COPY CONSTRUCTOR - LET'S SEE HOW!

```
ComplexNumber(const ComplexNumber& rhs) :
    realPart(rhs.realPart), complexPart(rhs.complexPart)
{
  cout << "Inside the copy constructor" << endl;
}
```

A LOT MORE ON THIS LATER, BUT FOR NOW - FOCUS ON THE INITIALISATION LIST

WE ARE ACCESSING THE PRIVATE MEMBER VARIABLES OF THE OBJECT `rhs`

RUN THE CODE TO MAKE SURE IT WORKS :-)

```
ComplexNumber c1(1.414,1.414);
ComplexNumber c2 = c1;
```

```
Vitthals-MacBook-Pro:~ vitthalsrinivasan$ g++ -Wall Example6.cpp
Vitthals-MacBook-Pro:~ vitthalsrinivasan$ ./a.out
Inside the 2-argument constructor
Inside the copy constructor
Printing out c1
real = 1.414 complex = 1.414
Printing out c2
real = 1.414 complex = 1.414
Okey-dokey! All done!
```

ALL GOOD!

BUT WE CAN ALWAYS CREATE OUR OWN COPY CONSTRUCTOR - LET'S SEE HOW!

```
ComplexNumber(const ComplexNumber& rhs) :
    realPart(rhs.realPart), complexPart(rhs.complexPart)
{
  cout << "Inside the copy constructor" << endl;
}
```

A LOT MORE ON THIS LATER, BUT FOR NOW - FOCUS ON THE INITIALISATION LIST

WE ARE ACCESSING THE PRIVATE MEMBER VARIABLES OF THE OBJECT `rhs`

ALL OBJECTS OF A CLASS CAN ACCESS EACH OTHERS PRIVATE PARTS

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"!

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"!

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

THE DISTINCTION BETWEEN PUBLIC AND PRIVATE MEMBERS/METHODS IS IN ORDER TO DRIVE A CLEAN SEPARATION BETWEEN

IMPLEMENTATION AND INTERFACE

IMPLEMENTATION AND INTERFACE

IN GENERAL, ONLY MAKE PUBLIC WHAT YOU MUST - BY DEFAULT KEEP EVERYTHING PRIVATE

KEEP ALL MEMBER VARIABLES PRIVATE - ALWAYS!

IMPLEMENTATION AND INTERFACE

IN GENERAL, ONLY MAKE PUBLIC WHAT YOU MUST - BY DEFAULT KEEP EVERYTHING PRIVATE

KEEP ALL MEMBER VARIABLES PRIVATE - ALWAYS!

PROVIDE PUBLIC GETTER AND SETTER METHODS FOR THOSE VARIABLES

KEEP ALL MEMBER VARIABLES PRIVATE - ALWAYS!

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

THIS WAY, YOU ALWAYS HAVE THE FLEXIBILITY TO CHANGE THE IMPLEMENTATION OF YOUR CLASS

KEEP ALL MEMBER VARIABLES PRIVATE - ALWAYS!

```
class ComplexNumber
{
private:
  float realPart;
  float complexPart;
public:
  void setRealPart(double r)
  {
    realPart = r;
  }

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

  float getRealPart()
  {
    return realPart;
  }
```

THIS WAY, YOU ALWAYS HAVE THE FLEXIBILITY TO CHANGE THE IMPLEMENTATION OF YOUR CLASS

BUT PROVIDE PUBLIC GETTER AND SETTER METHODS FOR THOSE VARIABLES

KEEP ALL MEMBER VARIABLES
PRIVATE - ALWAYS!

THIS WAY, YOU ALWAYS HAVE THE FLEXIBILITY TO CHANGE THE IMPLEMENTATION OF YOUR CLASS

BUT PROVIDE PUBLIC GETTER AND SETTER METHODS FOR THOSE VARIABLES

SO THAT OTHERS CAN RELY ON THE "INTERFACE" OF YOUR CLASS