EXCEPTIONS

EXCEPTIONS

WE ACTUALLY HAVE ENCOUNTERED EXCEPTIONS ONCE BEFORE

EXAMPLE 64: OVERRIDE THE DEFAULT INSTANTIATION FOR SOME SPECIFIC TYPE

FLASHBACK

EXAMPLE 64: OVERRIDE THE DEFAULT INSTANTIATION FOR SOME SPECIFIC TYPE

LET'S SAY WE WOULD LIKE TO WRITE A GENERIC COMPARE FUNCTION

IT WOULD COMPARE NUMBERS EXACTLY AS USUAL

BUT WHILE COMPARING STRINGS, IT WILL TRY AND CONVERT THEM TO NUMBERS FIRST IF POSSIBLE. IF NOT, IT WILL COMPARE AS STRINGS.

FLASHBACK

LET'S SAY WE WOULD LIKE TO WRITE A GENERIC COMPARE FUNCTION

IT WOULD COMPARE NUMBERS EXACTLY AS USUAL

BUT WHILE COMPARING STRINGS, IT WILL TRY AND CONVERT THEM TO NUMBERS FIRST IF POSSIBLE. IF NOT, IT WILL COMPARE AS STRINGS.

IN OTHER WORDS, WE NEED TO OVERRIDE THE DEFAULT TEMPLATE IMPLEMENTATION FOR A SPECIFIC TYPE (STRINGS)

NOT A PROBLEM! ITS REALLY SIMPLE :-)

FLASHBACK

OVERRIDE THE DEFAULT TEMPLATE IMPLEMENTATION FOR A SPECIFIC TYPE (STRINGS)

DEFINE THE FUNCTION TEMPLATE AS USUAL

```
template<class T>
int smartCompare(const T& a, const T& b)
{
  if (a > b)
    return 1;
  if (a < b)
    return -1;
  return 0;
}
```

FLASHBACK

OVERRIDE THE DEFAULT TEMPLATE IMPLEMENTATION FOR A SPECIFIC TYPE (STRINGS)

DEFINE THE FUNCTION TEMPLATE AS USUAL

```
template<class T>
int smartCompare(const T& a, const T& b)
{
  if (a > b)
    return 1;
  if (a < b)
    return -1;
  return 0;
}
```

THEN, EXPLICITLY INSTANTIATE THE TEMPLATE FOR THE SPECIFIC TYPE THAT YOU CARE ABOUT

```
int smartCompare(const string& a, const string& b)
{
  int x,y = 0;
  bool convertStringToInt = true;
  std::string::size_type sz;   // alias of size_t
  try {
   x = std::stoi (a,&sz);
  }
  catch(...) {
    cout << "Conversion failed " << a << endl;
    convertStringToInt = false;
  }
```

FLASHBACK

OVERRIDE THE DEFAULT TEMPLATE IMPLEMENTATION FOR A SPECIFIC TYPE (STRINGS)

DEFINE THE FUNCTION TEMPLATE AS USUAL

```
template<class T>
int smartCompare(const T& a, const T& b)
{
```

THEN, EXPLICITLY INSTANTIATE THE TEMPLATE FOR THE SPECIFIC TYPE THAT YOU CARE ABOUT

```
int smartCompare(const string& a, const string& b)
{
```

NOW, WHEN THE C++ COMPILER COMES ACROSS A COMPARISON OF 2 STRINGS, IT WILL USE YOUR FUNCTION..

FLASHBACK

OVERRIDE THE DEFAULT TEMPLATE IMPLEMENTATION FOR A SPECIFIC TYPE (STRINGS)

DEFINE THE FUNCTION TEMPLATE AS USUAL

```
template<class T>
int smartCompare(const T& a, const T& b)
{
```

THEN, EXPLICITLY INSTANTIATE THE TEMPLATE FOR THE SPECIFIC TYPE THAT YOU CARE ABOUT

```
int smartCompare(const string& a, const string& b)
{
```

JUST BE CAREFUL TO NOT INCLUDE ANY TEMPLATE PARAMETER, OR TEMPLATE INFORMATION IN YOUR SPECIFIC INSTANTIATION. IT WOULD BE WRONG TO DO SO - WILL CONFUSE THE C++ COMPILER

FLASHBACK

OVERRIDE THE DEFAULT TEMPLATE IMPLEMENTATION FOR A SPECIFIC TYPE (STRINGS)

DEFINE THE FUNCTION TEMPLATE AS USUAL

```
template<class T>
int smartCompare(const T& a, const T& b)
{
```

THEN, EXPLICITLY INSTANTIATE THE TEMPLATE FOR THE SPECIFIC TYPE THAT YOU CARE ABOUT

```
int smartCompare(const string& a, const string& b)
{
```

NOW, WHEN THE C++ COMPILER COMES ACROSS A COMPARISON OF 2 STRINGS, IT WILL USE YOUR FUNCTION..

FLASHBACK

NOW, WHEN THE C++ COMPILER COMES ACROSS A COMPARISON OF 2 STRINGS, IT WILL USE YOUR FUNCTION..

```
string firstName("Vitthal");
string lastName("Srinivasan");
i = smartCompare(firstName,lastName);
```

WHILE FOR A COMPARISON OF 2 INTS, THE C++ COMPILER WILL INSTANTIATE THE FUNCTION TEMPLATE

```
int a = 5;
int b = 10;
int i = smartCompare(a,b);
cout << i << endl;
```

FLASHBACK

NOW, WHEN THE C++ COMPILER COMES ACROSS A COMPARISON OF 2 STRINGS, IT WILL USE YOUR FUNCTION..

```
string firstName("Vitthal");
string lastName("Srinivasan");
i = smartCompare(firstName,lastName);
```

EXPLICIT INSTANTIATION FOUND - USE THAT VERSION

WHILE FOR A COMPARISON OF 2 INTS, THE C++ COMPILER WILL INSTANTIATE THE FUNCTION TEMPLATE

```
int a = 5;
int b = 10;
int i = smartCompare(a,b);
cout << i << endl;
```

NO EXPLICIT INSTANTIATION FOUND - C++ COMPILER WILL INSTANTIATE FUNCTION TEMPLATE

FLASHBACK

NOW, WHEN THE C++ COMPILER COMES ACROSS A COMPARISON OF 2 STRINGS, IT WILL USE YOUR FUNCTION..

```
string firstName("Vitthal");
string lastName("Srinivasan");
i = smartCompare(firstName,lastName);
```

EXPLICIT INSTANTIATION FOUND - USE THAT VERSION

```
int smartCompare(const string& a, const string& b)
{
  int x,y = 0;
  bool convertStringToInt = true;
  std::string::size_type sz;   // alias of size_t
  // try and convert both strings to ints.
  // if possible - compare the 2 strings as numbers
  try {
   x = std::stoi (a,&sz);
  }
  catch(...) {
    cout << "Conversion failed " << a << endl;
    convertStringToInt = false;
  }

  try {
   y = std::stoi (b,&sz);
  }
  catch(...) {
    cout << "Conversion failed " << b << endl;
    convertStringToInt = false;
  }
  if (convertStringToInt == true) {
    cout << "Converted both strings to ints.." << x << "," << y << endl;
    return smartCompare(x,y);
  }
  // if not, then compare as strings after all
  if (a > b)
```

THIS IS AN INTERESTING FUNCTION, SO LET'S UNDERSTAND IT LINE-BY-LINE

FLASHBACK

THIS IS AN INTERESTING FUNCTION, SO LET'S UNDERSTAND IT LINE-BY-LINE

```cpp
int smartCompare(const string& a, const string& b)
{
  int x,y = 0;
  bool convertStringToInt = true;
  std::string::size_type sz;   // alias of size_t
  // try and convert both strings to ints.
  // if possible - compare the 2 strings as numbers
  try {
   x = std::stoi (a,&sz);
  }
  catch(...) {
    cout << "Conversion failed " << a << endl;
    convertStringToInt = false;
  }

  try {
   y = std::stoi (b,&sz);
  }
  catch(...) {
    cout << "Conversion failed " << b << endl;
    convertStringToInt = false;
  }
  if (convertStringToInt == true) {
    cout << "Converted both strings to ints.." << x << "," << y << endl;
    return smartCompare(x,y);
  }
  // if not, then compare as strings after all
  if (a > b)
    return 1;
  if (a < b)
    return -1;
  return 0;


}
```

FLASHBACK

THIS IS AN INTERESTING FUNCTION, SO LET'S UNDERSTAND IT LINE-BY-LINE

```
int smartCompare(const string& a, const string& b)
{
  int x,y = 0;
  bool convertStringToInt = true;
  std::string::size_type sz;   // alias of size_t
  // try and convert both strings to ints.
  // if possible - compare the 2 strings as numbers
  try {
   x = std::stoi (a,&sz);
  }
  catch(...) {
    cout << "Conversion failed " << a << endl;
    convertStringToInt = false;
  }

  try {
   y = std::stoi (b,&sz);
  }
  catch(...) {
    cout << "Conversion failed " << b << endl;
    convertStringToInt = false;
  }
  if (convertStringToInt == true) {
    cout << "Converted both strings to ints.." << x << "," << y << endl;
    return smartCompare(x,y);
  }
  // if not, then compare as strings after all
  if (a > b)
    return 1;
  if (a < b)
    return -1;
  return 0;


}
```

THE NAME AND SIGNATURE OF THE FUNCTION TALLIES WITH THE FUNCTION TEMPLATE

```
template<class T>
int smartCompare(const T& a, const T& b)
{
```

FLASHBACK

THIS IS AN INTERESTING FUNCTION, SO LET'S UNDERSTAND IT LINE-BY-LINE

```
int smartCompare(const string& a, const string& b)
{
  int x,y = 0;
  bool convertStringToInt = true;
  std::string::size_type sz;   // alias of size_t
  // try and convert both strings to ints.
  // if possible - compare the 2 strings as numbers
  try {
   x = std::stoi (a,&sz);
  }
  catch(...) {
    cout << "Conversion failed " << a << endl;
    convertStringToInt = false;
  }

  try {
   y = std::stoi (b,&sz);
  }
  catch(...) {
    cout << "Conversion failed " << b << endl;
    convertStringToInt = false;
  }
  if (convertStringToInt == true) {
    cout << "Converted both strings to ints.." << x << "," << y << endl;
    return smartCompare(x,y);
  }
  // if not, then compare as strings after all
  if (a > b)
    return 1;
  if (a < b)
    return -1;
  return 0;



}
```

WHY? BECAUSE IT CAN BE REALLY ANNOYING TO GET NUMBERS SORTED AS STRINGS - LEXICOGRAPHICAL ORDER AND NUMERIC ORDER DON'T ALWAYS MATCH :-)

FLASHBACK

THIS IS AN INTERESTING FUNCTION, SO LET'S UNDERSTAND IT LINE-BY-LINE

```
int smartCompare(const string& a, const string& b)
{
  int x,y = 0;
  bool convertStringToInt = true;
  std::string::size_type sz;   // alias of size_t
  // try and convert both strings to ints.
  // if possible - compare the 2 strings as numbers
  try {
    x = std::stoi (a,&sz);
   }
   catch(...) {
     cout << "Conversion failed " << a << endl;
     convertStringToInt = false;
   }
  try {
   y = std::stoi (b,&sz);
  }
  catch(...) {
    cout << "Conversion failed " << b << endl;
    convertStringToInt = false;
  }
  if (convertStringToInt == true) {
    cout << "Converted both strings to ints" << x << " " << y << endl;
    return smartCompare(x,y);
  }
  // if not, then compare as strings after all
  if (a > b)
    return 1;
  if (a < b)
    return -1;
  return 0;
```

OUR FIRST `try/catch` BLOCK!

BASICALLY - WE `try` TO CONVERT A STRING TO A NUMBER USING THE `stoi` FUNCTION. IF AN ERROR RESULTS, WE `catch` THAT ERROR, AND CONCLUDE THAT THE CONVERSION IS NOT POSSIBLE

FLASHBACK

THIS IS AN INTERESTING FUNCTION, SO LET'S UNDERSTAND IT LINE-BY-LINE

```
int smartCompare(const string& a, const string& b)
{
 int x,y = 0;
 bool convertStringToInt = true;
 std::string::size_type sz;   // alias of size_t
 // try and convert both strings to ints.
 // if possible - compare the 2 strings as numbers
 try {
   x = std::stoi (a,&sz);
  }
  catch(...) {
    cout << "Conversion failed " << a << endl;
    convertStringToInt = false;
  }

 try {
  y = std::stoi (b,&sz);
 }
 catch(...) {
  cout << "Conversion failed " << b << endl;
  convertStringToInt = false;
 }
 if (convertStringToInt == true) {
  cout << "Converted both strings to ints.." << x << "," << y << endl;
  return smartCompare(x,y);
 }
 // if not, then compare as strings after all
 if (a > b)
  return 1;
 if (a < b)
  return -1;
 return 0;
```

WE try TO CONVERT A STRING TO A NUMBER

FLASHBACK

THIS IS AN INTERESTING FUNCTION, SO LET'S UNDERSTAND IT LINE-BY-LINE

```
int smartCompare(const string& a, const string& b)
{
  int x,y = 0;
  bool convertStringToInt = true;
  std::string::size_type sz;   // alias of size_t
  // try and convert both strings to ints.
  // if possible - compare the 2 strings as numbers
  try {
    x = std::stoi (a,&sz);
  }
  catch(...) {
    cout << "Conversion failed " << a << endl;
    convertStringToInt = false;
  }

  try {
   y = std::stoi (b,&sz);
  }
  catch(...) {
    cout << "Conversion failed " << b << endl;
    convertStringToInt = false;
  }
  if (convertStringToInt == true) {
    cout << "Converted both strings to ints: " << x << ", " << y << endl;
    return smartCompare(x,y);
  }
  // if not, then compare as strings after all
  if (a > b)
    return 1;
  if (a < b)
    return -1;
  return 0;
```

WE `try` TO CONVERT A STRING TO A NUMBER USING THE `stoi` FUNCTION.

FLASHBACK

THIS IS AN INTERESTING FUNCTION, SO LET'S UNDERSTAND IT LINE-BY-LINE

```
int smartCompare(const string& a, const string& b)
{
  int x,y = 0;
  bool convertStringToInt = true;
  std::string::size_type sz;   // alias of size_t
  // try and convert both strings to ints.
  // if possible - compare the 2 strings as numbers
  try {
    x = std::stoi (a,&sz);
   }
   catch(...) {
     cout << "Conversion failed " << a << endl;
     convertStringToInt = false;
   }
  try {
   y = std::stoi (b,&sz);
  }
  catch(...) {
    cout << "Conversion failed " << b << endl;
    convertStringToInt = false;
  }
  if (convertStringToInt == true) {
    cout << "Converted both strings to ints.." << x << ", " << y << endl;
    return smartCompare(x,y);
  }
  // if not, then compare as strings after all
  if (a > b)
    return 1;
  if (a < b)
    return -1;
  return 0;
```

IF AN ERROR RESULTS, WE catch THAT ERROR,

FLASHBACK

THIS IS AN INTERESTING FUNCTION, SO LET'S UNDERSTAND IT LINE-BY-LINE

```
int smartCompare(const string& a, const string& b)
{
  int x,y = 0;
  bool convertStringToInt = true;
  std::string::size_type sz;   // alias of size_t
  // try and convert both strings to ints.
  // if possible - compare the 2 strings as numbers
  try {
    x = std::stoi (a,&sz);
   }
   catch(...) {
     cout << "Conversion failed " << a << endl;
     convertStringToInt = false;
   }

  try {
   y = std::stoi (b,&sz);
  }
  catch(...) {
    cout << "Conversion failed " << b << endl;
    convertStringToInt = false;
  }
  if (convertStringToInt == true) {
    cout << "Converted both strings to ints.." << x << ", " << y << endl;
    return smartCompare(x,y);
  }
  // if not, then compare as strings after all
  if (a > b)
    return 1;
  if (a < b)
    return -1;
  return 0;
```

IF AN ERROR RESULTS, WE **catch** THAT ERROR,

FLASHBACK

THIS IS AN INTERESTING FUNCTION, SO LET'S UNDERSTAND IT LINE-BY-LINE

```
int smartCompare(const string& a, const string& b)
{
  int x,y = 0;
  bool convertStringToInt = true;
  std::string::size_type sz;   // alias of size_t
  // try and convert both strings to ints.
  // if possible - compare the 2 strings as numbers
  try {
   x = std::stoi (a,&sz);
  }
  catch(...) {
    cout << "Conversion failed " << a << endl;
    convertStringToInt = false;
  }

  try {
   y = std::stoi (b,&sz);
  }
  catch(...) {
    cout << "Conversion failed " << b << endl;
    convertStringToInt = false;
  }
  if (convertStringToInt == true) {
    cout << "Converted both strings to ints.." << x << "," << y << endl;
    return smartCompare(x,y);
  }
  // if not, then compare as strings after all
  if (a > b)
    return 1;
  if (a < b)
    return -1;
  return 0;
```

IF AN ERROR RESULTS, WE catch THAT ERROR, AND CONCLUDE THAT THE CONVERSION IS NOT POSSIBLE

FLASHBACK

THIS IS AN INTERESTING FUNCTION, SO LET'S UNDERSTAND IT LINE-BY-LINE

```
int smartCompare(const string& a, const string& b)
{
  int x,y = 0;
  bool convertStringToInt = true;
  std::string::size_type sz;   // alias of size_t
  // try and convert both strings to ints.
  // if possible - compare the 2 strings as numbers
  try {
    x = std::stoi (a,&sz);
   }
   catch(...) {
     cout << "Conversion failed " << a << endl;
     convertStringToInt = false;
   }
  try {
   y = std::stoi (b,&sz);
  }
  catch(...) {
    cout << "Conversion failed " << b << endl;
    convertStringToInt = false;
  }
  if (convertStringToInt == true) {
    cout << "Converted both strings to ints " << x << ", " << y << endl;
    return smartCompare(x,y);
  }
  // if not, then compare as strings after all
  if (a > b)
    return 1;
  if (a < b)
    return -1;
  return 0;
```

OUR FIRST try/catch BLOCK!

BASICALLY - WE try TO CONVERT A STRING TO A NUMBER USING THE stoi FUNCTION. IF AN ERROR RESULTS, WE catch THAT ERROR, AND CONCLUDE THAT THE CONVERSION IS NOT POSSIBLE

FLASHBACK

EXCEPTIONS

WE ACTUALLY HAVE ENCOUNTERED EXCEPTIONS ONCE BEFORE

LETS' BUILD ON THIS AND KEEP GOING

EXCEPTIONS

ARE THE MECHANISM PROVIDED IN C++ (AND JAVA AND MOST LANGUAGES) FOR

HANDLING RUNTIME ERRORS

THINK OF AN EXCEPTION AS KINDA LIKE A BURGLAR ALARM

EXCEPTIONS

THINK OF AN EXCEPTION AS KINDA LIKE A BURGLAR ALARM

SOMETHING UNUSUAL OR UNEXPECTED TRIGGERS THE SECURITY SYSTEM

THE SECURITY SYSTEM RAISES THE ALARM TO ALERT ANYONE THAT CARES

YOU MIGHT DEAL WITH THE ALARM YOURSELF IF YOU CAN..

OR YOU MIGHT DECIDE ITS TOO SERIOUS, AND CALL 911

EXCEPTIONS

THINK OF AN EXCEPTION AS KINDA LIKE A
BURGLAR ALARM

SOMETHING UNUSUAL OR UNEXPECTED TRIGGERS
AN ERROR IN SOME FUNCTION

[illegible]

THAT FUNCTION RAISES AN EXCEPTION, "THROWING THE
EXCEPTION" UP TO WHOEVER CALLED THAT FUNCTION

THE SECURITY SYSTEM RAISES THE ALARM TO
ALERT ANYONE THAT CARES

IF YOU ARE THE CODE THAT CALLED THAT FUNCTION,
YOU CATCH THE EXCEPTION AND TRY TO DEAL WITH IT

YOU MIGHT DEAL WITH THE
ALARM YOURSELF IF YOU CAN...

BUT IF ITS TOO SERIOUS AN ERROR, YOU
GIVE UP AND ABORT THE PROGRAM

[illegible]

EXAMPLE 74: LEARN HOW TO `throw` AN EXCEPTION WHEN SOMETHING GOES WRONG

EXAMPLE 74: LEARN HOW TO `throw` AN EXCEPTION WHEN SOMETHING GOES WRONG

SAY WE HAVE A COMPLEX NUMBER CLASS

IN THE CONSTRUCTOR, WE CHECK THAT ITS MODULUS IS NON-NEGATIVE, ELSE WE `throw` AN EXCEPTION

NEVER MIND IF YOU DON'T KNOW WHY ITS AN ERROR FOR THE MODULUS OF A COMPLEX NUMBER TO BE NEGATIVE, JUST FOCUS ON THE ERROR BEING THROWN :-)

IN THE CONSTRUCTOR, WE CHECK THAT ITS MODULUS IS NON-NEGATIVE, ELSE WE throw AN EXCEPTION

```
ComplexNumber_Polar(double amp, double arg) : modulus(amp) , argument(arg)
 {
   if (amp < 0) {
     cout << "Modulus of a complex number can not be negative! Throwing an exception" << endl;
     throw InvalidComplexNumberError("Modulus can't be negative!");
   }
   cout << "Inside the 2-argument constructor" << endl;
 }
```

IN THE CONSTRUCTOR, WE CHECK THAT ITS MODULUS IS NON-NEGATIVE, ELSE WE throw AN EXCEPTION

```
ComplexNumber_Polar(double amp, double arg) : modulus(amp) , argument(arg)
 {
   if (amp < 0) {
     cout << "Modulus of a complex number can not be negative! Throwing an exception" << endl;
    throw InvalidComplexNumberError("Modulus can't be negative!");
   }
   cout << "Inside the 2-argument constructor" << endl;
 }
```

THIS IS SIMPLY AN OBJECT OF A CLASS WE CREATED - NOTHING FANCY!

IN THE CONSTRUCTOR, WE CHECK THAT ITS MODULUS IS NON-NEGATIVE, ELSE WE throw AN EXCEPTION

```
ComplexNumber_Polar(double amp, double arg) : modulus(amp) , argument(arg)
 {
   if (amp < 0) {
     cout << "Modulus of a complex number can not be negative! Throwing an exception" << endl;
    throw InvalidComplexNumberError("Modulus can't be negative!");
   }
   cout << "Inside the 2-argument constructor" << endl;
 }
```

THIS IS SIMPLY AN OBJECT OF A CLASS WE CREATED - NOTHING FANCY!

```
class InvalidComplexNumberError
{
public:
  string errorMessage;
  InvalidComplexNumberError(string error) : errorMessage(error) {}

};
```

THIS IS SIMPLY AN OBJECT OF A CLASS WE CREATED - NOTHING FANCY!

```
class InvalidComplexNumberError
{
public:
  string errorMessage;
  InvalidComplexNumberError(string error) : errorMessage(error) {}

};
```

SINCE THE EXCEPTION IS MERELY AN OBJECT, WE CAN PASS VERY SOPHISTICATED ERROR HANDLING INFORMATION VIA THIS MECHANISM!

HERE WE THREW AN OBJECT OF A CLASS THAT WE HAD CREATED..

BUT THERE ARE ALSO STANDARD TYPES OF ERROR OBJECTS THROWN BY STANDARD C++ FUNCTIONS

FOR INSTANCE THE `stoi` FUNCTION WE USED THROWS `invalid_argument` AND `out_of_range` EXCEPTIONS

TWO WORDS OF CAUTION

NEVER THROW EXCEPTIONS FROM INSIDE A DESTRUCTOR

EVEN IF YOU THROW AN EXCEPTION, NEVER LEAK MEMORY OR RESOURCES

EXAMPLE 75: LEARN HOW TO USE `try/catch` TO HANDLE EXCEPTIONS THAT WERE THROWN BY OTHERS

EXAMPLE 75: LEARN HOW TO USE `try/catch` TO HANDLE EXCEPTIONS THAT WERE THROWN BY OTHERS

IN THE EXAMPLE WE JUST DID, WE HAD A COMPLEX NUMBER CLASS

IN THE CONSTRUCTOR, WE CHECK THAT ITS MODULUS IS NON-NEGATIVE, ELSE WE `throw` AN EXCEPTION

EXAMPLE 75: LEARN HOW TO USE `try/catch` TO HANDLE EXCEPTIONS THAT WERE THROWN BY OTHERS

NOW, LET'S USE A `try/catch` BLOCK TO CHECK FOR ERRORS..

IF THE ERROR IS THE ONE WE KNOW HOW TO HANDLE, WE WILL HANDLE IT, ELSE WE PASS

NOW, LET'S USE A `try/catch` BLOCK TO CHECK FOR ERRORS..

IF THE ERROR IS THE ONE WE KNOW HOW TO HANDLE, WE
WILL HANDLE IT, ELSE WE PASS

```
  try {
    ComplexNumber_Polar c1(-7.0,45);
    //throw string("Some random other exception");
  }
  catch(InvalidComplexNumberError e) {
    cout << "Please ensure that the modulus is positive!" << endl;
  }
  catch (...) {
    cout << "Something else went wrong..throwing the error onwards, I can't 
deal with it" << endl;
    throw;
  }
```

NOW, LET'S USE A `try/catch` BLOCK TO CHECK FOR ERRORS..

IF THE ERROR IS THE ONE WE KNOW HOW TO HANDLE, WE
WILL HANDLE IT, ELSE WE PASS

```
try {
  ComplexNumber_Polar c1(-7.0,45);
  //throw string("Some random other exception");
}
catch(InvalidComplexNumberError e) {
  cout << "Please ensure that the modulus is positive!" << endl;
}
catch (...) {
  cout << "Something else went wrong..throwing the error onwards, I can't
deal with it" << endl;
  throw;
}
```

NOW, LET'S USE A `try/catch` BLOCK TO CHECK FOR ERRORS..

IF THE ERROR IS THE ONE WE KNOW HOW TO HANDLE, WE WILL HANDLE IT, ELSE WE PASS

```
try {
  ComplexNumber_Polar c1(-7.0,45);
  //throw string("Some random other exception");
}
catch(InvalidComplexNumberError e) {
  cout << "Please ensure that the modulus is positive" << endl;
}
catch (...) {
  cout << "Something else went wrong..throwing the error onwards, I can't
deal with it" << endl;
  throw;
}
```

SURROUND THE CODE THAT MIGHT THROW THE ERROR WITH A TRY BLOCK, LIKE THIS.

NOW, LET'S USE A try/catch BLOCK TO CHECK FOR ERRORS..

IF THE ERROR IS THE ONE WE KNOW HOW TO HANDLE, WE WILL HANDLE IT, ELSE WE PASS

```
try {
  ComplexNumber_Polar c1(-7.0,45);
  //throw string("Some random other exception");
}
```

NOTE AGAIN - THE CODE THAT MIGHT THROW THE EXCEPTION MUST BE INSIDE THE TRY BLOCK

NOW, LET'S USE A `try/catch` BLOCK TO CHECK FOR ERRORS..

IF THE ERROR IS THE ONE WE KNOW HOW TO HANDLE, WE
WILL HANDLE IT, ELSE WE PASS

```
try {
    ComplexNumber_Polar c1(-7.0,45);
    //throw string("Some random other exception");
}
catch(InvalidComplexNumberError e) {
    cout << "Please ensure that the modulus is positive!" << endl;
}
catch (...) {
    cout << "Something else went wrong, throwing the error onwards, I can't
deal with it" << endl;
    throw;
}
```

YOU CAN CATCH EXCEPTIONS OF SPECIFIC
TYPES USING ONE OR MORE CATCH BLOCKS

NOW, LET'S USE A `try/catch` BLOCK TO CHECK FOR ERRORS..

IF THE ERROR IS THE ONE WE KNOW HOW TO HANDLE, WE WILL HANDLE IT, ELSE WE PASS

```
try {
   ComplexNumber_Polar c1(-7.0,45);
   //throw string("Some random other exception");
}
catch(InvalidComplexNumberError e) {
   cout << "Please ensure that the modulus is positive!" << endl;
}
catch (...) {
   cout << "Something else went wrong..throwing the error onwards, I can't 
deal with it" << endl;
   throw;
}
```

HERE WE KNOW WHAT CAUSES THIS TYPE OF EXCEPTION BECAUSE WE THREW IT OURSELVES :-)

NOW, LET'S USE A `try/catch` BLOCK TO CHECK FOR ERRORS..

IF THE ERROR IS THE ONE WE KNOW HOW TO HANDLE, WE
WILL HANDLE IT, ELSE WE PASS

```
try {
    ComplexNumber_Polar c1(-7.0,45);
    //throw string("Some random other exception");
}
catch(InvalidComplexNumberError e) {
    cout << "Please ensure that the modulus is positive!" << endl;
}
catch (...) {
    cout << "Something else went wrong. throwing the error onwards, I can't
deal with it" << endl;
    throw;
}
```

SO - JUST LET OFF THE USER WITH A
PRINTED WARNING, NOT THAT BIG A DEAL,
PROGRAM EXECUTION CAN GO ON

SO - JUST LET OFF THE USER WITH A PRINTED WARNING, NOT THAT BIG A DEAL, PROGRAM EXECUTION CAN GO ON

IMPORTANT! AFTER AN EXCEPTION IS THROWN AND CAUGHT, PROGRAM EXECUTION CONTINUES FROM AFTER THE ENTIRE TRY BLOCK.

NOW, LET'S USE A try/catch BLOCK TO CHECK FOR ERRORS..

IF THE ERROR IS THE ONE WE KNOW HOW TO HANDLE, WE
WILL HANDLE IT, ELSE WE PASS

```
try {
   ComplexNumber Polar c1(-7.0,45);
   throw string("Some random other exception");
}
catch(InvalidComplexNumberError e) {
   cout << "Please ensure that the modulus is positive!" << endl;
}
catch (...) {
   cout << "Something else went wrong..throwing the error onwards, I can't
deal with it" << endl;
   throw;
}
```

BUT AN EXCEPTION COULD BE ANY KIND
OF OBJECT, NOT JUST THE ONE WE THREW

NOW, LET'S USE A `try/catch` BLOCK TO CHECK FOR ERRORS..

IF THE ERROR IS THE ONE WE KNOW HOW TO HANDLE, WE
WILL HANDLE IT, ELSE WE PASS

```
try {
  ComplexNumber_Polar c1(-7.0,45);
  //throw string("Some random other exception");
}
catch(InvalidComplexNumberError e) {
  cout << "Please ensure that the modulus is positive!" << endl;
}
catch (...) {
  cout << "Something else went wrong, throwing the error onwards, I can't
deal with it" << endl;
  throw;
}
```

SO A CATCH-ALL (LITERALLY) IS POSSIBLE
WITH THE ... SYNTAX USED HERE

NOW, LET'S USE A `try/catch` BLOCK TO CHECK FOR ERRORS..

IF THE ERROR IS THE ONE WE KNOW HOW TO HANDLE, WE
WILL HANDLE IT, ELSE WE PASS

```
   try {
     ComplexNumber_Polar c1(-7.0,45);
     //throw string("Some random other exception");
   }
   catch(InvalidComplexNumberError e) {
     cout << "Please ensure that the modulus is positive!" << endl;
   }
   catch (...) {
     cout << "Something else went wrong, throwing the error onwards, I can't
deal with it" << endl;
     throw;
   }
```

SO A CATCH-ALL (LITERALLY) IS POSSIBLE
WITH THE ... SYNTAX USED HERE

NOW, LET'S USE A `try/catch` BLOCK TO CHECK FOR ERRORS..

IF THE ERROR IS THE ONE WE KNOW HOW TO HANDLE, WE
WILL HANDLE IT, ELSE WE PASS

```
try {
  ComplexNumber_Polar c1(-7.0,45);
  //throw string("Some random other exception");
}
catch(InvalidComplexNumberError e) {
  cout << "Please ensure that the modulus is positive" << endl;
}
catch (...) {
  cout << "Something else went wrong..throwing the error onwards, I can't
deal with it" << endl;
  throw;
}
```

PASS THE EXCEPTION ON - WE CAN'T
HANDLE IT OURSELVES, CALL 911