EXAMPLE 10:

RULE #3: USE `new[]/delete[]` FOR ARRAY VARIABLES OF ALL TYPES.

RULE #3: USE `new[]`/`delete[]` FOR ARRAY VARIABLES OF ALL TYPES.

ANYTIME YOU NEED TO CREATE AN ARRAY OF ANY TYPE, JUST `new[]` TO BOTH ALLOCATE AND CONSTRUCT THE ARRAY

`new[]` WILL CYCLE THROUGH AND CALL THE NO-ARGUMENT CONSTRUCTOR FOR EACH ELEMENT OF THE ARRAY

YOU CAN'T PASS IN ARGUMENTS TO THE CONSTRUCTOR WITH `new[]`

REFRESHER

RULE #3: USE `new[]`/`delete[]` FOR ARRAY VARIABLES OF ALL TYPES.

ANYTIME YOU NEED TO CREATE AN ARRAY OF ANY TYPE, JUST `new[]` TO BOTH ALLOCATE AND CONSTRUCT THE ARRAY

`new[]` WILL CYCLE THROUGH AND CALL THE NO-ARGUMENT CONSTRUCTOR FOR EACH ELEMENT OF THE ARRAY

YOU CAN'T PASS IN ARGUMENTS TO THE CONSTRUCTOR WITH `new[]`

ANYTHING YOU ALLOCATE+CONSTRUCT USING `new[]`, CLEAN UP USING `delete[]`

REFRESHER

RULE #3: USE `new[]/delete[]` FOR ARRAY VARIABLES OF ALL TYPES.

ANYTHING YOU ALLOCATE+CONSTRUCT USING `new[]`, CLEAN UP USING `delete[]`

`new[]` WILL FIRST ALLOCATE MEMORY FOR THE ARRAY, AND THEN CALL THE DEFAULT CONSTRUCTOR FOR EACH ARRAY ELEMENT. IT WILL TRACK ARRAY LENGTH TOO

`delete[]` WILL FIRST CALL THE DESTRUCTOR FOR EACH ARRAY ELEMENT, AND THEN DEALLOCATE MEMORY

REFRESHER

RULE #3: USE new[]/delete[] FOR ARRAY VARIABLES OF ALL TYPES.

```
ComplexNumber * cDynamic = new ComplexNumber[10];
  for(int i = 0;i<10;i++)
  {
    cout << "Printing out dynamically allocated object" << i << endl;
    cDynamic[i].print();
  }
  delete[] cDynamic;
```

USE new[] TO CREATE AN ARRAY OF 10 OBJECTS

RULE #3: USE new[]/delete[] FOR ARRAY VARIABLES OF ALL TYPES.

```
ComplexNumber * cDynamic = new ComplexNumber[10];
 for(int i = 0;i<10;i++)
 {
   cout << "Printing out dynamically allocated object" << i << endl;
   cDynamic[i].print();
 }
 delete[] cDynamic;
```

USE new[] TO CREATE AN ARRAY OF 10 OBJECTS

MAKE SURE WE CAN ACCESS EACH ONE, AND THAT IT PRINTS WHAT WE EXPECT (ALL ZEROS, DEFAULT CONSTRUCTOR!)

RULE #3: USE new[]/delete[] FOR ARRAY VARIABLES OF ALL TYPES.

```
ComplexNumber * cDynamic = new ComplexNumber[10];
 for(int i = 0;i<10;i++)
 {
   cout << "Printing out dynamically allocated object" << i << endl;
   cDynamic[i].print();
 }
 delete[] cDynamic;
```

NOTICE HOW WE USE THE DOT OPERATOR, NOT ->, TO REFER TO AN INDIVIDUAL ELEMENT OF THIS ARRAY

USE new[] TO CREATE AN ARRAY OF 10 OBJECTS

RULE #3: USE new[]/delete[] FOR ARRAY VARIABLES OF ALL TYPES.

```
ComplexNumber * cDynamic = new ComplexNumber[10];
 for(int i = 0;i<10;i++)
 {
   cout << "Printing out dynamically allocated object" << i << endl;
   cDynamic[i].print();
 }
 delete[] cDynamic;
```

NOTICE HOW WE USE THE DOT OPERATOR, NOT ->, TO REFER TO AN INDIVIDUAL ELEMENT OF THIS ARRAY

USE new[] TO CREATE AN ARRAY OF 10 OBJECTS

RULE #3: USE new[]/delete[] FOR ARRAY VARIABLES OF ALL TYPES.

```
ComplexNumber * cDynamic = new ComplexNumber[10];
 for(int i = 0;i<10;i++)
 {
   cout << "Printing out dynamically allocated object" << i << endl;
   cDynamic[i].print();
 }
 delete[] cDynamic;
```

LASTLY, USE delete[] TO CLEAN UP

USE new[] TO CREATE AN ARRAY OF 10 OBJECTS

RULE #3: USE new[]/delete[] FOR ARRAY VARIABLES OF ALL TYPES.

```
ComplexNumber * cDynamic = new ComplexNumber[10];
  for(int i = 0;i<10;i++)
  {
    cout << "Printing out dynamically allocated object" << i << endl;
    cDynamic[i].print();
  }
  delete[] cDynamic;
```

AND THIS IS THE OUTPUT FROM RUNNING THIS LINE OF CODE

```
Vitthals-MacBook-Pro:~ vitthalsrinivasan$ ./a.out
No arg-constructor called
No arg-constructor called
No arg-constructor called
No arg-constructor called
No arg-constructor called
No arg-constructor called
No arg-constructor called
No arg-constructor called
No arg-constructor called
No arg-constructor called
No arg-constructor called
```

RULE #3: USE new[]/delete[] FOR ARRAY VARIABLES OF ALL TYPES.

```
ComplexNumber * cDynamic = new ComplexNumber[10];
  for(int i = 0;i<10;i++)
  {
    cout << "Printing out dynamically allocated object" << i << endl;
    cDynamic[i].print();
  }
  delete[] cDynamic;
```

AND THIS IS THE OUTPUT FROM RUNNING THIS LINE OF CODE

```
Vitthals-MacBook-Pro:~ vitthalsrinivasan$ ./a.out
No arg-constructor called
No arg-constructor called
No arg-constructor called
No arg-constructor called
No arg-constructor called
No arg-constructor called
No arg-constructor called
No arg-constructor called
No arg-constructor called
No arg-constructor called
```

WHICH IS WHAT WE'D EXPECT AND HOPE FOR.

```
ComplexNumber() : realPart(0.0),complexPart(0.0)
{
  cout << "No arg-constructor called" << endl;
}
```

RULE #3: USE new[]/delete[] FOR ARRAY VARIABLES OF ALL TYPES.

```
ComplexNumber * cDynamic = new ComplexNumber[10];
 for(int i = 0;i<10;i++)
 {
   cout << "Printing out dynamically allocated object" << i << endl;
   cDynamic[i].print();
 }
 delete[] cDynamic;
```

```
Printing out dynamically allocated object0
real = 0 complex = 0
Printing out dynamically allocated object1
real = 0 complex = 0
Printing out dynamically allocated object2
real = 0 complex = 0
Printing out dynamically allocated object3
real = 0 complex = 0
Printing out dynamically allocated object4
real = 0 complex = 0
Printing out dynamically allocated object5
real = 0 complex = 0
Printing out dynamically allocated object6
real = 0 complex = 0
Printing out dynamically allocated object7
real = 0 complex = 0
Printing out dynamically allocated object8
real = 0 complex = 0
Printing out dynamically allocated object9
real = 0 complex = 0
```

MAKE SURE WE CAN ACCESS EACH ONE, AND THAT IT PRINTS WHAT WE EXPECT (ALL ZEROS, DEFAULT CONSTRUCTOR!)

AND CHECK THAT THE OUTPUT IS OK (IT IS)

RULE #3: USE new[]/delete[] FOR ARRAY VARIABLES OF ALL TYPES.

```
ComplexNumber * cDynamic = new ComplexNumber[10];
 for(int i = 0;i<10;i++)
 {
   cout << "Printing out dynamically allocated object" << i << endl;
   cDynamic[i].print();
 }
 delete[] cDynamic;
```

```
Inside the destructor: realPart = 0 complexPart = 0
Inside the destructor: realPart = 0 complexPart = 0
Inside the destructor: realPart = 0 complexPart = 0
Inside the destructor: realPart = 0 complexPart = 0
Inside the destructor: realPart = 0 complexPart = 0
Inside the destructor: realPart = 0 complexPart = 0
Inside the destructor: realPart = 0 complexPart = 0
Inside the destructor: realPart = 0 complexPart = 0
Inside the destructor: realPart = 0 complexPart = 0
Inside the destructor: realPart = 0 complexPart = 0
Okey-dokey! All done!
```

AND THE delete[] CALL TO CLEAN UP SHOULD CALL THE DESTRUCTOR FOR EACH OF THE 10 OBJECTS

CHECK THAT THE OUTPUT IS OK (IT IS)