Security Assessment Report

Project: Shapes Calculator

Version 1

1 May 2023

Summary

The main goal of the security assessment was to implement more secure coding techniques into the existing code. A major finding was that I was using unsecured pointer methods within the source code.

Assessment Scope

I used JetBrains Clion IDE to implement and test my code with the new changes I implemented. I also used Google Chrome to find documentation on code/methods I was trying to implement into the project.

Summary of Findings

Refer to Figure 1 - Detailed Findings and Figure 3 – Swot Analysis.

There were a few major issues that were discovered. One of those issues was the incorrect use of GitHub. I was only using it to store my source code online instead of utilizing its version control functionalities. I did not have any logging of changes being made to the code. Another major issue that I encountered were buffer overflows related to the pointers I was using in the source code. When testing the code these buffer overflows would lead to bugs in the code.

Summary of Recommendations

The most important changes I needed to make were the integration of version control using GitHub as well as the fix of buffer overflows. Alongside those issues I also needed to integrate proper OS access controls to only allow authorized changes to be made to the code. Also, I needed to utilize GitHub as a tool to help me hide certain files or information that were not supposed to be made available publicly.

Goals, Findings, and Recommendations

Assessment Goals

The purpose of the assessment was to find security issues within the project. If major issues were found, then the next goal of the assessment was to learn how to fix those issues and to implement solutions for them.

Detailed Findings

All security vulnerabilities that I found when doing the assessment on the project are listed and explained in Figure 1 - Detailed Findings.

Recommendations

All security recommendations that I came up with for the project are listed and explained in Figure 2 – Recommendations

Methodology for the Security Control Assessment

Refer to Figure 4 – Risk Assessment.

The findings that were most accurate were the use of unsecure coding techniques, improper code testing, and improper documentation. These were the most accurate because there were findings of coding techniques that were later fixed such as buffer overflows. There was not any proper code testing since the buffer overflow bugs were not found when the project was deployed. Improper documentation was another finding since accounting integration with GitHub had not been integrated until recently.

Testing and Tools

During the testing of the Shape Calculator project, I utilized a combination of Grey Box and White Box testing. Grey box testing was utilized to test random functionalities of the project without specifically testing certain code portions. It isn't considered Black Box testing since the code was available to me and written by me, but I did view the code during this type of testing. I later used White Box testing to look for bugs within certain functionalities and to test for buffer overflows.

Tools:

GitHub – Used as a version control system to help with accounting and documentation.

Clion – Utilized as the IDE to build and test source code.

Figures and Code

Figure 1 – Detailed Findings

Description	Type	Explanation
Unsecure coding techniques	Weakness	The use of unsecured coding techniques such as not using smart pointers could lead to bugs or security breaches was determined to be a weakness, since it was already implemented it could be easily exploited.

Improper testing	Threat	Did not properly test code throughout development was deemed a threat because there is a high possibility that there are existing bugs in the code that weren't discovered throughout development that could lead to a security breach.
Poor code maintainability	Threat	I did not maintain the code after original goals were met and was categorized as a threat since I didn't keep making changes to try to improve the performance or security of the code.
Poor documentation	Vulnerability	I did not use proper documentation or accounting throughout the project. This was determined to be a vulnerability since it would make it difficult to view previous changes to code and to log future changes that could be made.
Unsecured networks	Threat	The use of unsecured networks during the development of the project could have been a security threat since it could have led to security breaches throughout development.
Network security	Threat	The use of an unsecured computer due to not using a VPN nor a firewall during development was determined to be a threat. This left my computer open to cyber-attacks.
Unencrypted Files	Vulnerability	I did not encrypt any of the files I used for the development of the project, which is a vulnerability. This could lead to unauthorized users viewing and changing my source code files.

Application updates	Vulnerability	I did not keep updating the application after submission which can be considered a vulnerability, since I did not fix any existing bugs.

Figure 2 – Recommendations

Recommendation	Explanation
OS access controls	To prevent unauthorized changes being made to my source code files, I utilized my OS access controls. I went through security properties for each source code file and edited the permissions. This allowed me to modify permissions for each user or group to where only I am allowed to make changes to the files.
Accounting: Process of logging	To track changes being made to the project, as well as tracking to see who made the changes, I fully integrated Git and GitHub into my development process. Rather than just uploading the source code files from my file directory into GitHub, I will pull and push changes to GitHub through my IDE. This will allow me to track changes being made to the project.
Integration of cloud-based platform for hiding of information	The use of GitHub as the VCS allows me to only push files/changes to the repo that can be shown to the public. It allows me to protect sensitive information and code such as files like .gitignore, cache files, or any file that may have a security risk if shown on a public repository.
Code issue	Implemented the use of STL array style over the c-style to the string menu array. This allows for a more updated method. Also, changed variable type to const array<string, 5> menu to prevent any changes to the size of the array.
Buffer Overflows	To prevent memory leaks and buffer overflow issues, I implemented smart_pointers so that dynamically allocated memory would be deleted automatically. I also implemented for loops that use an iterator of type vector<smart_pointer>, which iterate better through dynamically allocated memory such as pointers. This will also help prevent buffer overflows since the .size method of a vector is not in use anymore.

Figure 3 – Swot Analysis

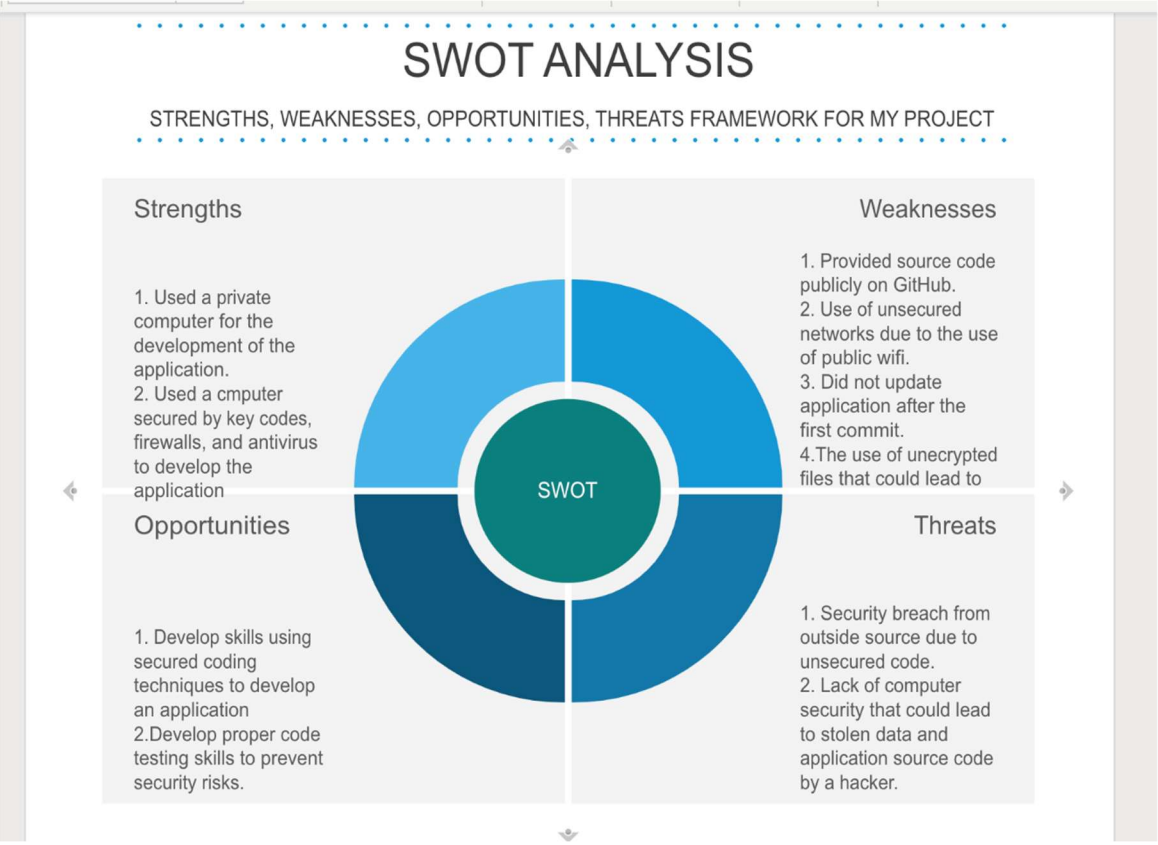
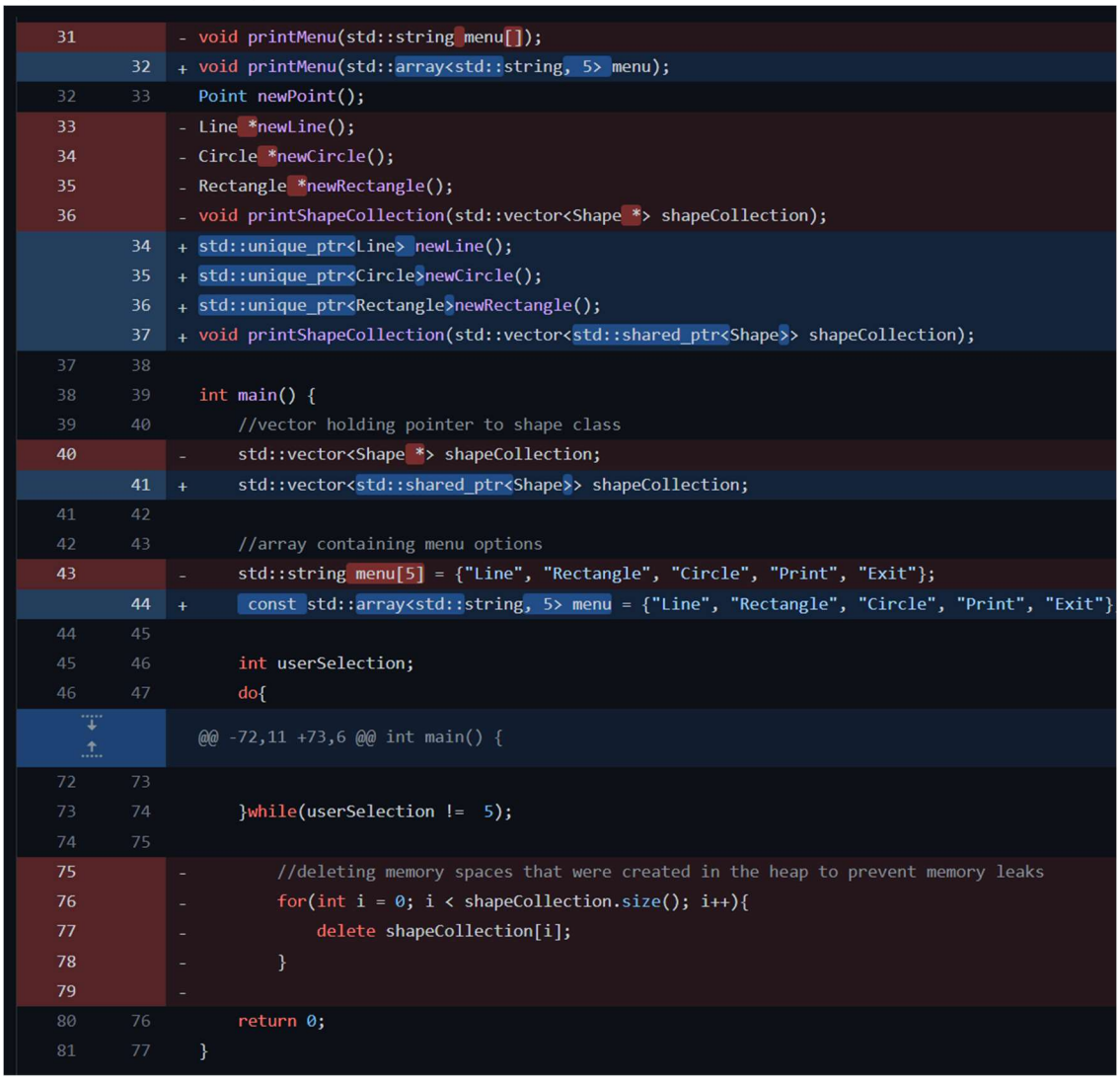


Figure 4 – Risk Assessment

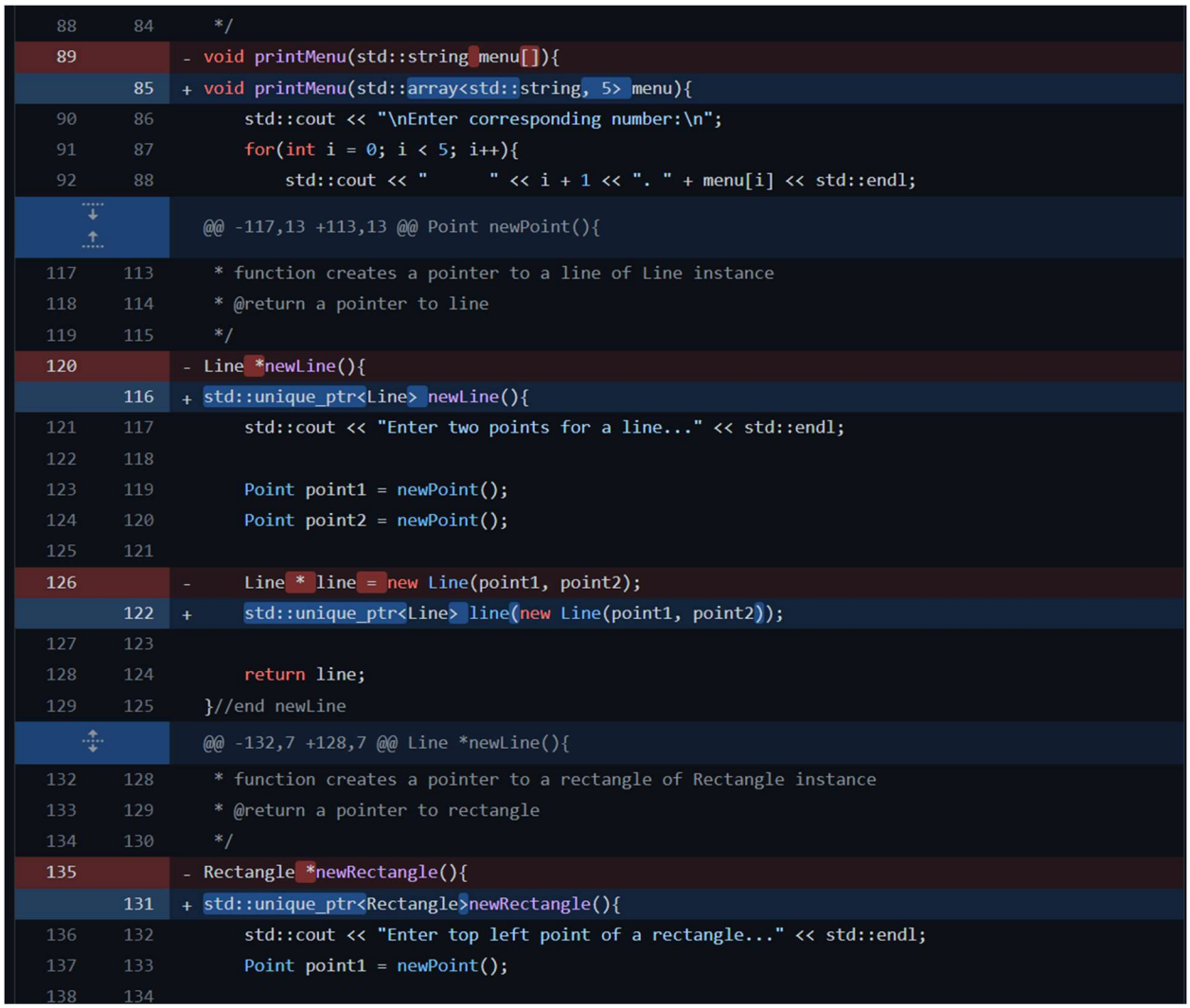
1							
2			Probability ---------->				
3	Severity		Frequent	Probable	Likely	Possible	Rare
4	|	Emergency	Used unsecure coding techniques which could lead to security breach	Did not maintain code after original goals were met	hard coded variables	Unsecured network due to public wifi	Unencrypted files
5	|	Major	Did not properly test code through development	source code available publicly on GitHub	Did not update application after submission	Unsecured cmputer due to lack of security such as a VPN	
6	|	Moderate		improper documentation			
7	|	Minor					
8	|	Negatable					
9	V						
10							
11							
12							

Figure 5 – Code Changes



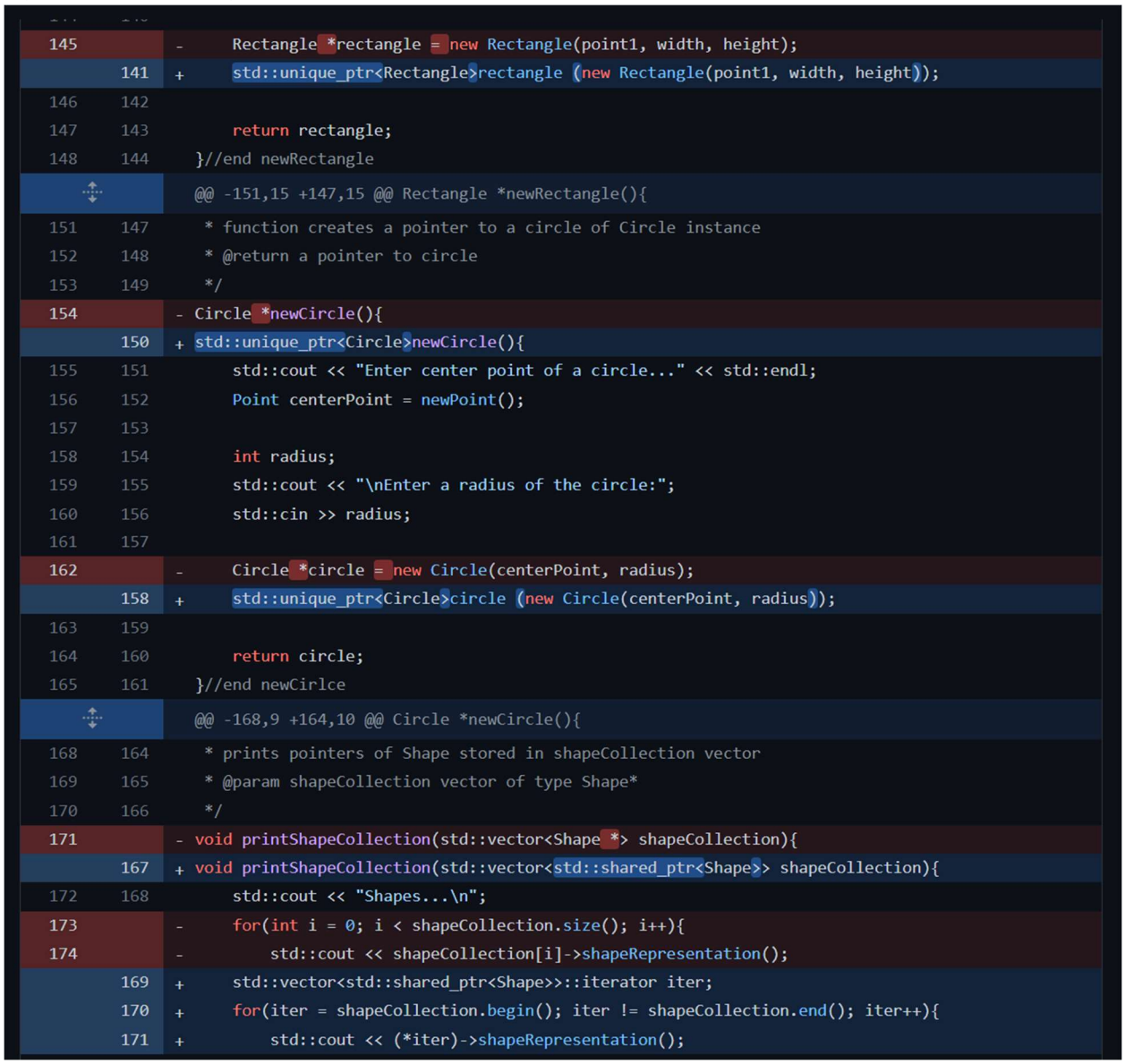
```
31      - void printMenu(std::string menu[]);
    32  + void printMenu(std::array<std::string, 5> menu);
32  33    Point newPoint();
33      - Line *newLine();
34      - Circle *newCircle();
35      - Rectangle *newRectangle();
36      - void printShapeCollection(std::vector<Shape *> shapeCollection);
    34  + std::unique_ptr<Line> newLine();
    35  + std::unique_ptr<Circle>newCircle();
    36  + std::unique_ptr<Rectangle>newRectangle();
    37  + void printShapeCollection(std::vector<std::shared_ptr<Shape>> shapeCollection);
37  38
38  39    int main() {
39  40        //vector holding pointer to shape class
40      -     std::vector<Shape *> shapeCollection;
    41  +     std::vector<std::shared_ptr<Shape>> shapeCollection;
41  42
42  43        //array containing menu options
43      -     std::string menu[5] = {"Line", "Rectangle", "Circle", "Print", "Exit"};
    44  +     const std::array<std::string, 5> menu = {"Line", "Rectangle", "Circle", "Print", "Exit"};
44  45
45  46        int userSelection;
46  47        do{
          @@ -72,11 +73,6 @@ int main() {
72  73
73  74        }while(userSelection !=  5);
74  75
75      -         //deleting memory spaces that were created in the heap to prevent memory leaks
76      -         for(int i = 0; i < shapeCollection.size(); i++){
77      -             delete shapeCollection[i];
78      -         }
79      -
80  76        return 0;
81  77    }
```

Figure 6 - Code



```
88  84    */
89      - void printMenu(std::string menu[]){
        85 + void printMenu(std::array<std::string, 5> menu){
90  86        std::cout << "\nEnter corresponding number:\n";
91  87        for(int i = 0; i < 5; i++){
92  88            std::cout << "      " << i + 1 << ". " + menu[i] << std::endl;
        @@ -117,13 +113,13 @@ Point newPoint(){
117 113    * function creates a pointer to a line of Line instance
118 114    * @return a pointer to line
119 115    */
120     - Line *newLine(){
        116 + std::unique_ptr<Line> newLine(){
121 117        std::cout << "Enter two points for a line..." << std::endl;
122 118
123 119        Point point1 = newPoint();
124 120        Point point2 = newPoint();
125 121
126     -      Line * line = new Line(point1, point2);
        122 +      std::unique_ptr<Line> line(new Line(point1, point2));
127 123
128 124        return line;
129 125    }//end newLine
        @@ -132,7 +128,7 @@ Line *newLine(){
132 128    * function creates a pointer to a rectangle of Rectangle instance
133 129    * @return a pointer to rectangle
134 130    */
135     - Rectangle *newRectangle(){
        131 + std::unique_ptr<Rectangle>newRectangle(){
136 132        std::cout << "Enter top left point of a rectangle..." << std::endl;
137 133        Point point1 = newPoint();
138 134
```

Figure 7 – Code Changes



```
145     -     Rectangle *rectangle = new Rectangle(point1, width, height);
    141 +     std::unique_ptr<Rectangle>rectangle (new Rectangle(point1, width, height));
146 142
147 143         return rectangle;
148 144     }//end newRectangle
          @@ -151,15 +147,15 @@ Rectangle *newRectangle(){
151 147      * function creates a pointer to a circle of Circle instance
152 148      * @return a pointer to circle
153 149      */
154     - Circle *newCircle(){
    150 + std::unique_ptr<Circle>newCircle(){
155 151         std::cout << "Enter center point of a circle..." << std::endl;
156 152         Point centerPoint = newPoint();
157 153
158 154         int radius;
159 155         std::cout << "\nEnter a radius of the circle:";
160 156         std::cin >> radius;
161 157
162     -     Circle *circle = new Circle(centerPoint, radius);
    158 +     std::unique_ptr<Circle>circle (new Circle(centerPoint, radius));
163 159
164 160         return circle;
165 161     }//end newCirlce
          @@ -168,9 +164,10 @@ Circle *newCircle(){
168 164      * prints pointers of Shape stored in shapeCollection vector
169 165      * @param shapeCollection vector of type Shape*
170 166      */
171     - void printShapeCollection(std::vector<Shape *> shapeCollection){
    167 + void printShapeCollection(std::vector<std::shared_ptr<Shape>> shapeCollection){
172 168         std::cout << "Shapes...\n";
173     -     for(int i = 0; i < shapeCollection.size(); i++){
174     -         std::cout << shapeCollection[i]->shapeRepresentation();
    169 +     std::vector<std::shared_ptr<Shape>>::iterator iter;
    170 +     for(iter = shapeCollection.begin(); iter != shapeCollection.end(); iter++){
    171 +         std::cout << (*iter)->shapeRepresentation();
```

Works Cited

Tutorials. cplusplus.com. (n.d.). Retrieved May 1, 2023, from https://cplusplus.com/

Whitebox and Blackbox Testing.pptx from lectures and in canvas

Risk to Threat.pptx from lectures and in canvas

AccessControls1.pptx from lectures and in canvas

Smart pointers. cppreference.com. (n.d.). Retrieved May 1, 2023, from https://en.cppreference.com/book/intro/smart_pointers

Logging cheat sheet¶. Logging - OWASP Cheat Sheet Series. (n.d.). Retrieved May 1, 2023, from https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html