**Практична робота №1**

**Тема роботи: Застосування методів рефакторингу при розробці програмного забезпечення.**

**Мета роботи**: Навчитися застосовувати на практиці прийоми рефакторингу.

**Виконавець**: Любченко Олег.

**Хід роботи**

Фрагмент коду, в якому зображена функція для перевірки можливості розміщення корабля у обраних координатах для гри «Морський бій»:

|  |
| --- |
| // Function to check if a ship can be placed at a given position  int canPlaceShip(char grid[GRID\_SIZE][GRID\_SIZE], int x, int y, int size, char orientation) {      if (orientation != 'h' && orientation != 'v') {          system("cls");          printf(">>> Invalid input. Try again.\n\n");          printGrid(grid);          return 0; // Invalid orientation      }      if (orientation == 'h') {          if (y + size > GRID\_SIZE) {              system("cls");              printf(">>> Invalid input. Try again.\n\n");              printGrid(grid);              return 0; // Ship exceeds grid boundaries          }          for (int i = y; i < y + size; i++) {              if (grid[x][i] != '-') {                  system("cls");                  printf(">>> Invalid input. Try again.\n\n");                  printGrid(grid);                  return 0; // Collision with another ship              }              if (x > 0 && grid[x - 1][i] != '-') {                  system("cls");                  printf(">>> Invalid input. Try again.\n\n");                  printGrid(grid);                  return 0; // Collision with another ship horizontally              }              if (x < GRID\_SIZE - 1 && grid[x + 1][i] != '-') {                  system("cls");                  printf(">>> Invalid input. Try again.\n\n");                  printGrid(grid);                  return 0; // Collision with another ship horizontally              }              if (i > 0 && grid[x][i - 1] != '-') {                  system("cls");                  printf(">>> Invalid input. Try again.\n\n");                  printGrid(grid);                  return 0; // Collision with another ship vertically              }              if (i < GRID\_SIZE - 1 && grid[x][i + 1] != '-') {                  system("cls");                  printf(">>> Invalid input. Try again.\n\n");                  printGrid(grid);                  return 0; // Collision with another ship vertically              }          }      } else if (orientation == 'v') {          if (x + size > GRID\_SIZE) {              system("cls");              printf(">>> Invalid input. Try again.\n\n");              printGrid(grid);              return 0; // Ship exceeds grid boundaries          }          for (int i = x; i < x + size; i++) {              if (grid[i][y] != '-') {                  system("cls");                  printf(">>> Invalid input. Try again.\n\n");                  printGrid(grid);                  return 0; // Collision with another ship              }              if (y > 0 && grid[i][y - 1] != '-') {                  system("cls");                  printf(">>> Invalid input. Try again.\n\n");                  printGrid(grid);                  return 0; // Collision with another ship horizontally              }              if (y < GRID\_SIZE - 1 && grid[i][y + 1] != '-') {                  system("cls");                  printf(">>> Invalid input. Try again.\n\n");                  printGrid(grid);                  return 0; // Collision with another ship horizontally              }              if (i > 0 && grid[i - 1][y] != '-') {                  system("cls");                  printf(">>> Invalid input. Try again.\n\n");                  printGrid(grid);                  return 0; // Collision with another ship vertically              }              if (i < GRID\_SIZE - 1 && grid[i + 1][y] != '-') {                  system("cls");                  printf(">>> Invalid input. Try again.\n\n");                  printGrid(grid);                  return 0; // Collision with another ship vertically              }          }      }      return 1; // Valid placement  } |

**Причина рефакторингу:** Код має велику кількість рядків що повторюються, це погіршує його читабельність та збільшує громадність, що впливає на оптимізацію.  
 **План рефакторингу:** Винести функцію виведення повідомлення про невдалу спробу розташування корабля, а також спростити функції перевірки колізій.

Фрагмент коду після рефакторингу:

|  |
| --- |
| // Function to print an error message and the grid  void printErrorAndGrid(char grid[GRID\_SIZE][GRID\_SIZE]) {      system("cls");      printf(">>> Invalid input. Try again.\n\n");      printGrid(grid);  }  // Function to check if a ship can be placed at a given position  int canPlaceShip(char grid[GRID\_SIZE][GRID\_SIZE], int x, int y, int size, char orientation) {      if (orientation != 'h' && orientation != 'v') {          printErrorAndGrid(grid);          return 0; // Invalid orientation      }      if (orientation == 'h') {          if (y + size > GRID\_SIZE) {              printErrorAndGrid(grid);              return 0; // Ship exceeds grid boundaries          }          for (int i = y; i < y + size; i++) {              if (grid[x][i] != '-' || (x > 0 && grid[x - 1][i] != '-') || (x < GRID\_SIZE - 1 && grid[x + 1][i] != '-')                  || (i > 0 && grid[x][i - 1] != '-') || (i < GRID\_SIZE - 1 && grid[x][i + 1] != '-')) {                  printErrorAndGrid(grid);                  return 0; // Collision with another ship              }          }      } else if (orientation == 'v') {          if (x + size > GRID\_SIZE) {              printErrorAndGrid(grid);              return 0; // Ship exceeds grid boundaries          }          for (int i = x; i < x + size; i++) {              if (grid[i][y] != '-' || (y > 0 && grid[i][y - 1] != '-') || (y < GRID\_SIZE - 1 && grid[i][y + 1] != '-')                  || (i > 0 && grid[i - 1][y] != '-') || (i < GRID\_SIZE - 1 && grid[i + 1][y] != '-')) {                  printErrorAndGrid(grid);                  return 0; // Collision with another ship              }          }      }      return 1; // Valid placement  } |

**Висновок:** За допомогою проведення рефакторингу було значно скорочено кількість рядків коду, а також покращено читабельність та оптимізацію.

**Контрольні запитання:**1. Рефакторинг - це процес зміни вихідного коду програми без зміни його зовнішньої поведінки.  
2.1 Необхідно додати нову функцію, яка недостатньо вкладається в прийняте архітектурне рішення.  
2.2 Необхідно виправити помилку, причини виникнення якої відразу не ясні.  
2.3 Подолання труднощів у командній розробці, які обумовлені складною логікою програми.  
3. Уникайте зайвого коду, використовуйте ефективні алгоритми, оптимізуйте запити, та використовуйте кешування для збереження результатів обчислень.  
4. Зміна сигнатури методу, інкапсуляція поля, виділення класу, виділення інтерфейсу, виділення локальної змінної, виділення методу, генералізація типу, вбудовування, введення фабрики, введення параметра, підйом методу, спуск методу, перейменування методу.