Programowanie obiektowe

Zajęcia 3. Automatyczne zarządzanie pamięcią

Zadanie 1. Projekt recykling w Java

1. Utwórz w wybranym miejscu na dysku folder `RecyclingJava`.
2. W VSC upewnij się, że masz zainstalowany plugin *Project Manager for Java*.
3. Otwórz folder `RecyclingJava`.
4. Dodaj w projekcie folder `recycling` a następnie dodaj do niego plik o nazwie `Wezel.java` z implementacją:

```
package recycling;

import java.util.ArrayList;
import java.util.Scanner;

public class Wezel {
    @Deprecated
    @Override
    protected void finalize() throws Throwable {
        System.out.println("Finalizowanie obiektu");
        super.finalize();
    }

    public static void main(String[] args) {
```

```
        Scanner s = new Scanner(System.in);
        for (int i = 0; i <= 10000; i++) {
            new Wezel();
        }
        s.nextLine();
        s.close();
    }
}
```

5. W okienku Explorera powinna pojawić się zakładka JAVA PROJECTS. Po najechaniu myszą na wspomnianą zakładkę powinna pojawić się opcja exportu projektu do pliku jar stanowiącego skompresowaną wersję skompilowanych do byte code'u plików wykonywalnych Java'y (ikona dla tej opcji wygląda jak strzałka skierowana w dół). Z tej opcji mamy zamiar właśnie skorzystać. Po wybraniu opcji "Export Jar..." VSC zapyta którą spośród klas zawartych w projekcie chcemy uznać za "Entry Point" aplikacji czyli klasę której statyczna metoda main ma być uruchomiona jako aplikacja w momencie uruchomienia pliku jar. Informacja ta zawarta będzie w tzw. pliku manifestu pliku jar zawierającego informacje konfiguracyjne tego pliku. Wybieramy zatem klasę Wezel. W tym momencie w folderze projektu powinien pojawić się plik `RecyclingJava.jar`. Żeby uruchomić jara – przejdź do okienka terminala (np. poprzez naciśnięcie skrótu klawiszowego CTRL+`) i wpisz:

   ```
   $java -jar RecyclingJava.jar
   ```

   Czy na konsoli został wypisany jakikolwiek tekst "Finalizowanie obiektu"?

6. Zmień implementację metody main i zamiast 10000 iteracji pętli wymuś uruchomienie 1000000 iteracji.

7. Utwórz jeszcze raz plik jar, uruchom go, zweryfikuj efekty i spróbuj je zinterpretować.

8. Do implementacji klasy węzeł dodaj pole `polaczenia` o typie `ArrayList<Wezel>`. Zainicjalizuj pole podstawiając pod nie wartość `new ArrayList<>()`.

9. Dodaj do klasy `Wezel` metodę `dodajPolaczenie`, która będzie przyjmowała w parametrze referencję na węzeł z którym ma być połączony węzeł `this`.

10. Pojedynczą iterację pętli w metodzie `main` zastąp kodem:

```
Wezel w1 = new Wezel();
Wezel w2 = new Wezel();
w1.dodajPolaczenie(w2);
w2.dodajPolaczenie(w1);
```

11. Ponownie utwórz plik jar, uruchom go i zweryfikuj efekty. Jak wyjaśnisz fakt, że pomimo tego że do poszczególnych węzłów nadal są odwołania (do węzłów utworzonych jako `w1` z `w2` a do węzłów `w2` z `w1`) odśmiecacz je usunął?

12. Do „zapamiętania" połączeń między obiektami typu `Wezel` skorzystano z klasy generycznej `ArrayList`. Jak sądzisz dlaczego tradycyjna tablica obiektów typu `Wezel` nie byłaby najlepszym wyborem?

13. Jeśli zależałoby nam na szybszej niż linowa (w zależności od ilości następników) złożoności dostępu do informacji czy dany węzeł jest następnikiem danego węzła zamiast struktury listy/tablicy moglibyśmy zastosować jedną z kolekcji o interfejsie `Map<Wezel, Boolean>`. Do dyspozycji mamy tutaj cały wachlarz możliwych implementacji, z których najbardziej rozpoznawalne będą `HashMap<Wezel, Boolean>` oraz `TreeMap<Wezel, Boolean>`. Zaproponuj implementację wykorzystującą oba z tych rozwiązań.

Zadanie 2. Projekt recykling w C++

1. Utwórz w wybranym miejscu na dysku folder `RecyclingCpp` i otwórz go w VSC.

2. Dodaj do projektu implementację klasy `Wezel` (pamiętaj o dodawaniu zarówno pliku nagłówkowego jak i źródłowego) z destruktorem wypisującym na standardowe wyjście informację o destrukcji obiektu. Np.:

```
Wezel::~Wezel() {
    std::cout << "Destrukcja obiektu" << std::endl;
}
```

3. Dodaj do projektu plik źródłowy `main.cpp`, zawierające implementację:

```
#include <memory>
#include "Wezel.h"

int main() {
    for (int i = 0; i < 10; i++) {
        std::unique_ptr<Wezel>(new Wezel());
    }
```

```
    return 0;
}
```

4. Skompiluj i uruchom projekt (nie zapomnij o zadbaniu by wszystkie pliki źródłowe były dodane do procesu kompilacji), zweryfikuj standardowe wyjście programu i spróbuj je zinterpetować w kontekście wyników projektu `RecyclingJava`.

5. Kod funkcji main zastąp tak by zamiast `std::unique_ptr` wykorzystać `std::shared_ptr`:

```
int main() {
    for (int i = 0; i < 10; i++) {
        std::shared_ptr<Wezel> w1 = std::make_shared<Wezel>();
    }
}
```

6. Czy uruchomienie takiego kodu coś zmieniło?

7. Do klasy `Wezel` dodaj wskaźnik `std::shared_ptr<Wezel>`.

8. Kod funkcji main zastąp jeszcze raz, tym razem tak by były cykliczne referencje pomiędzy węzłami:

```
int main() {
    for (int i = 0; i < 10; i++) {
        std::shared_ptr<Wezel> w1 = std::make_shared<Wezel>();
        std::shared_ptr<Wezel> w2 = std::make_shared<Wezel>();
        w1->next = w2;
        w2->next = w1;
    }
}
```

9. Czy teraz program zwalnia pamięć jak należy?

10. Zastanów się w jaki sposób można wykorzystać wskaźnik typu `std::weak_ptr` do rozwiązania powyższego problemu i spróbuj ubrać to w kod.