Notatki z Programowania Obiektowego w C++

1. Pojęcie klasy i obiektu na przykładzie języka C++

Klasa

- **Klasa** to podstawowy element programowania obiektowego. Określa strukturę (pola, czyli dane) oraz zachowanie (metody, czyli funkcje członkowskie) obiektów, które z tej klasy mogą być tworzone.
- Jest swoistym „szablonem” (wzorcem), na podstawie którego tworzone są obiekty.

Obiekt

- **Obiekt** to konkretny egzemplarz (instancja) klasy. Obiekt posiada swoje własne pola (zawartość), a także może wywoływać metody zdefiniowane w klasie.
- Przykładowo, jeśli mamy klasę `Samochod`, to obiekt tej klasy to będzie konkretny samochód z określonymi parametrami (np. marka, kolor, rok produkcji).

Przykład prostej klasy i tworzenia obiektu

```
#include <iostream>
#include <string>

class Samochod {
public:
    // pola (atrybuty)
    std::string marka;
    std::string kolor;
    int rokProdukcji;

    // metody
    void wyswietlInformacje() {
        std::cout << "Marka: " << marka
                  << ", Kolor: " << kolor
                  << ", Rok: " << rokProdukcji << std::endl;
    }
};

int main() {
    // Tworzymy obiekt (instancję klasy Samochod)
    Samochod mojSamochod;

    // Ustawiamy wartości pól
    mojSamochod.marka = "Toyota";
    mojSamochod.kolor = "Czerwony";
    mojSamochod.rokProdukcji = 2020;

    // Wywołujemy metodę
    mojSamochod.wyswietlInformacje();
```

```
    return 0;
}
```

Przykładowe pytania – Punkt 1

1. **Co to jest klasa w C++?**

   Odpowiedź: Klasa to swoisty szablon (wzorzec) opisujący wspólne cechy (pola, metody) grupy obiektów. Na jej podstawie tworzy się obiekty (instancje).

2. **Co to jest obiekt?**

   Odpowiedź: Obiekt to konkretny egzemplarz klasy, posiadający własną kopię pól oraz dostęp do metod zdefiniowanych w klasie.

3. **W jaki sposób tworzy się obiekt w języku C++?**

   Odpowiedź: Obiekt tworzy się poprzez deklarację zmiennej typu klasy. Np. `Samochod mojSamochod;`.

---

2. Mechanizm dziedziczenia klas i polimorfizm w C++. Koncepcja hermetyzacji (enkapsulacji) w programowaniu obiektowym

Dziedziczenie

- **Dziedziczenie** pozwala tworzyć nowe klasy (klasy pochodne) na podstawie już istniejących klas (klas bazowych).
- Klasa pochodna przejmuje cechy (pola, metody) klasy bazowej, może je rozszerzać oraz modyfikować.
- W C++ wyróżnia się kilka trybów dziedziczenia: `public`, `protected` i `private`. Najczęściej stosowane jest dziedziczenie `public`.

Schemat dziedziczenia

```
KlasaBazowa
    |
    |
KlasaPochodna
```

Polimorfizm

- **Polimorfizm** polega na tym, że w zależności od typu obiektu wywoływana jest odpowiednia wersja metody (mimo takiej samej nazwy).
- W C++ polimorfizm osiągamy głównie poprzez:
    1. **Funkcje wirtualne** (dynamiczny polimorfizm).
    2. **Przeciążanie funkcji** (statyczny polimorfizm).

Hermetyzacja (enkapsulacja)

- **Hermetyzacja** (enkapsulacja) to ukrywanie wewnętrznej implementacji klasy i udostępnianie jej tylko za pośrednictwem interfejsu (metod publicznych).
- Dzięki temu zapobiegamy niepożądanym zmianom w zewnętrznym kodzie programu.

Przykład dziedziczenia i polimorfizmu

```
#include <iostream>
#include <string>

// Klasa bazowa
class Ksztalt {
public:
    virtual void rysuj() {
        std::cout << "Rysuję kształt." << std::endl;
    }
};

// Klasa pochodna
class Kolo : public Ksztalt {
public:
    void rysuj() override {
        std::cout << "Rysuję koło." << std::endl;
    }
};

int main() {
    Ksztalt* ksztalt1 = new Ksztalt();
    Ksztalt* ksztalt2 = new Kolo();    // polimorfizm

    ksztalt1->rysuj();  // wypisze: "Rysuję kształt."
    ksztalt2->rysuj();  // wypisze: "Rysuję koło."

    delete ksztalt1;
    delete ksztalt2;

    return 0;
}
```

Przykładowe pytania – Punkt 2

1. **Na czym polega dziedziczenie w C++?**

   Odpowiedź: Dziedziczenie to mechanizm tworzenia nowej klasy (pochodnej) na bazie innej klasy (bazowej), dzięki czemu klasa pochodna przejmuje właściwości i metody klasy bazowej i może je rozszerzać lub modyfikować.

2. **Co to jest polimorfizm i jak się go osiąga w C++?**

Odpowiedź: Polimorfizm polega na wywoływaniu różnych wersji metody w zależności od rzeczywistego typu obiektu. W C++ osiąga się go głównie za pomocą funkcji wirtualnych (dynamiczny polimorfizm) oraz przeciążania funkcji (statyczny polimorfizm).

3. **Na czym polega hermetyzacja (enkapsulacja)?**

   Odpowiedź: Hermetyzacja polega na ukrywaniu implementacji (szczegółów wewnętrznych) klasy i udostępnianiu z zewnątrz tylko niezbędnych metod (interfejsu). Chroni to klasę przed niepożądanym dostępem i modyfikacjami.

---

3. Klasy abstrakcyjne w języku C++

Definicja

- **Klasa abstrakcyjna** to klasa, która nie może być instancjonowana (nie można utworzyć obiektu tej klasy).
- Zawiera **co najmniej jedną metodę wirtualną czysto wirtualną** (w C++ zapisywaną jako `virtual void nazwaFunkcji() = 0;`).

Przykład

```
#include <iostream>

// Klasa abstrakcyjna
class Figura {
public:
    virtual void rysuj() = 0; // metoda czysto wirtualna
};

// Klasa pochodna z implementacją metody rysuj()
class Kwadrat : public Figura {
public:
    void rysuj() override {
        std::cout << "Rysuję kwadrat." << std::endl;
    }
};

int main() {
    // Figura fig; // BŁĄD - nie można tworzyć obiektu klasy abstrakcyjnej
    Kwadrat kw;
    kw.rysuj(); // "Rysuję kwadrat."
    return 0;
}
```

Przykładowe pytania – Punkt 3

1. **Co wyróżnia klasę abstrakcyjną w C++?**

Odpowiedź: Klasa abstrakcyjna zawiera co najmniej jedną metodę czysto wirtualną i nie można tworzyć jej obiektów.

2. **Do czego służą klasy abstrakcyjne?**

   Odpowiedź: Służą do definiowania interfejsów i ogólnych struktur, które muszą być doprecyzowane w klasach dziedziczących.

---

4. Wskaźniki i referencje oraz dynamiczne zarządzanie pamięcią w języku C++

Wskaźniki

- **Wskaźnik** (pointer) to zmienna przechowująca adres innej zmiennej lub obiektu w pamięci.
- W C++ używa się operatora `*` (dereferencja) oraz `&` (pobranie adresu).

```
int x = 10;
int* ptr = &x;  // ptr przechowuje adres x
```

Referencje

- **Referencja** to „przyjazny wskaźnik”, ale nie może być zmieniony po inicjalizacji i zawsze musi wskazywać na jakiś obiekt.

```
int y = 20;
int& ref = y; // ref to alias na y
```

Dynamiczne zarządzanie pamięcią

- Umożliwia tworzenie obiektów w czasie działania programu (na stercie – ang. **heap**) za pomocą operatora `new` i zwalnianie pamięci za pomocą operatora `delete`.
- Od C++11 do dyspozycji mamy też inteligentne wskaźniki (`std::unique_ptr`, `std::shared_ptr`, `std::weak_ptr`), które automatycznie zarządzają zwalnianiem pamięci.

Przykład dynamicznego tworzenia obiektu

```
#include <iostream>

int main() {
    int* dynamiczna = new int(5); // alokacja pamięci na wartość 5
    std::cout << *dynamiczna << std::endl; // wypisze 5

    delete dynamiczna; // zwolnienie pamięci
```

```
    return 0;
}
```

Przykładowe pytania – Punkt 4

1. **Czym jest wskaźnik w C++?**

   Odpowiedź: To zmienna przechowująca adres innej zmiennej (w pamięci). Pozwala na manipulację obiektem za pośrednictwem jego adresu.

2. **Jaka jest różnica między wskaźnikiem a referencją?**

   Odpowiedź: Referencja zawsze musi być zainicjalizowana przy deklaracji i nie może zmieniać tego, na co wskazuje. Wskaźnik może być zmieniany i przechowywać różne adresy w trakcie działania programu.

3. **Co to jest dynamiczna alokacja pamięci i jak się ją realizuje w C++?**

   Odpowiedź: To przydzielanie pamięci w trakcie działania programu (na stercie) przy użyciu operatora `new` i zwalnianie jej za pomocą `delete`.

---

5. Przeciążenie operatorów, przeciążenie funkcji oraz koncepcja funkcji zaprzyjaźnionych w języku C++

Przeciążenie operatorów

- W C++ można **przeciążać** (nadpisywać działanie) operatorów, by móc używać ich w kontekście własnych typów (klas).
- Składnia przeciążania operatora: `typZwracany operator+(const Typ& inny) { ... }`

Przykład przeciążenia operatora `+`

```
#include <iostream>

class Wektor {
private:
    int x, y;
public:
    Wektor(int x, int y) : x(x), y(y) {}

    // Przeciążenie operatora +
    Wektor operator+(const Wektor& other) const {
        return Wektor(this->x + other.x, this->y + other.y);
    }

    void wyswietl() const {
        std::cout << "(" << x << ", " << y << ")" << std::endl;
    }
};
```

```
int main() {
    Wektor w1(1, 2);
    Wektor w2(3, 4);

    Wektor w3 = w1 + w2; // wywołuje przeciążony operator +
    w3.wyswietl();       // (4, 6)

    return 0;
}
```

Przeciążenie funkcji

- **Przeciążenie funkcji** to definiowanie kilku funkcji o tej samej nazwie, ale różniących się listą parametrów (typy, liczba parametrów).

```
#include <iostream>

// przykład przeciążenia funkcji
int dodaj(int a, int b) {
    return a + b;
}

double dodaj(double a, double b) {
    return a + b;
}
```

Funkcje zaprzyjaźnione

- **Funkcja zaprzyjaźniona** (friend) to funkcja, która może mieć dostęp do prywatnych składników klasy, ale sama do niej nie należy.
- Definiujemy ją w klasie z użyciem słowa kluczowego `friend`.

```
#include <iostream>

class Prostokat {
private:
    double szerokosc, wysokosc;
public:
    Prostokat(double s, double w) : szerokosc(s), wysokosc(w) {}

    // Deklaracja funkcji zaprzyjaźnionej
    friend double obliczPole(const Prostokat& p);
};

double obliczPole(const Prostokat& p) {
    // Ma dostęp do prywatnych pól
    return p.szerokosc * p.wysokosc;
}
```

```
int main() {
    Prostokat prost(5.0, 3.0);
    std::cout << "Pole: " << obliczPole(prost) << std::endl; // 15

    return 0;
}
```

Przykładowe pytania – Punkt 5

1. **Na czym polega przeciążenie operatorów w C++?**

   Odpowiedź: Polega na zdefiniowaniu lub zmodyfikowaniu działania operatorów (np. `+`, `-`, `<<`, itp.) tak, aby działały z obiektami klas własnych.

2. **Czym różni się przeciążenie funkcji od funkcji wirtualnych?**

   Odpowiedź: Przeciążenie funkcji polega na definiowaniu kilku funkcji o tej samej nazwie, ale innych parametrach. Funkcje wirtualne służą do polimorficznego zachowania metod w klasach dziedziczących (decyzja o tym, którą metodę wywołać, zapada w czasie wykonania).

3. **Co to są funkcje zaprzyjaźnione (friend) w C++?**

   Odpowiedź: To funkcje niezależne od klasy (lub inne klasy), które mają dostęp do jej prywatnych składników. Deklarujemy je słowem `friend` wewnątrz klasy.

---

6. Poziomy dostępu do składników klasy oraz funkcje i zmienne statyczne w języku C++

Poziomy dostępu

- **public** – składniki publiczne są dostępne dla każdego (zarówno wewnątrz klasy, jak i poza nią).
- **protected** – składniki chronione są dostępne w tej klasie oraz w klasach dziedziczących.
- **private** – składniki prywatne dostępne są tylko wewnątrz tej klasy (nie są dostępne dla klas pochodnych, z wyjątkiem funkcji zaprzyjaźnionych).

Składniki statyczne

- **Zmienna statyczna w klasie** – jest współdzielona przez wszystkie obiekty tej klasy (jedna kopia na całą klasę).
- **Funkcja statyczna** – może być wywoływana bez tworzenia obiektu (np. `NazwaKlasy::funkcjaStatyczna()`), nie ma dostępu do `this` (bo nie jest związana z konkretnym obiektem).

Przykład

```
#include <iostream>
```

```cpp
class Licznik {
private:
    static int ileObiektow;
public:
    Licznik() {
        ileObiektow++;
    }
    ~Licznik() {
        ileObiektow--;
    }

    static int getIleObiektow() {
        return ileObiektow;
    }
};

// Definicja zmiennej statycznej
int Licznik::ileObiektow = 0;

int main() {
    std::cout << "Poczatkowo: " << Licznik::getIleObiektow() << std::endl; // 0
    Licznik l1;
    std::cout << "Po utworzeniu l1: " << Licznik::getIleObiektow() << std::endl;
// 1
    {
        Licznik l2;
        std::cout << "W bloku: " << Licznik::getIleObiektow() << std::endl; // 2
    }
    std::cout << "Po wyjsciu z bloku: " << Licznik::getIleObiektow() << std::endl;
// 1
    return 0;
}
```

Przykładowe pytania – Punkt 6

1. **Wymień poziomy dostępu do członków klasy w C++.**

   Odpowiedź: public, protected, private.

2. **Co to jest zmienna statyczna w klasie i jak ją zdefiniować?**

   Odpowiedź: Jest to zmienna współdzielona przez wszystkie obiekty klasy. Deklarujemy ją wewnątrz klasy jako `static`, a definiujemy (zazwyczaj w pliku .cpp) poza klasą, np. `int Klasa::nazwaZmiennej = 0;`.

3. **Czym się różni metoda statyczna od niestatycznej?**

   Odpowiedź: Metoda statyczna nie wymaga istnienia obiektu, może być wywoływana przez `NazwaKlasy::metodaStatyczna()`. Nie ma dostępu do wskaźnika `this`, więc nie manipuluje danymi obiektu, a jedynie danymi statycznymi klasy.

---

7. Konstruktory i destruktory w C++

Konstruktor

- **Konstruktor** to specjalna metoda wywoływana automatycznie w momencie tworzenia obiektu.
- Ma taką samą nazwę jak klasa i nie ma typu zwracanego.
- Służy do inicjalizacji pól obiektu.

Destruktor

- **Destruktor** to metoda wywoływana w momencie usuwania/niszczenia obiektu (np. gdy obiekt wyjdzie z zasięgu).
- Nazwa destruktora to `~NazwaKlasy()`.
- Zwalniamy w nim zasoby, jeśli obiekt z nich korzystał (np. pamięć dynamicznie alokowaną).

Przykład

```
#include <iostream>

class Test {
public:
    Test() { // konstruktor
        std::cout << "Konstruktor wywołany." << std::endl;
    }

    ~Test() { // destruktor
        std::cout << "Destruktor wywołany." << std::endl;
    }
};

int main() {
    std::cout << "Przed utworzeniem obiektu." << std::endl;
    Test t; // wywołanie konstruktora
    std::cout << "Po utworzeniu obiektu." << std::endl;
    return 0;
    // wywołanie destruktora w momencie wyjścia z funkcji main
}
```

Przykładowe pytania – Punkt 7

1. **Czym jest konstruktor?**

   Odpowiedź: To specjalna metoda klasy, wywoływana przy tworzeniu obiektu, która służy do inicjalizacji jego danych (pól).

2. **Czym jest destruktor i kiedy jest wywoływany?**

   Odpowiedź: To metoda wywoływana automatycznie podczas niszczenia obiektu (np. wyjścia poza zasięg zmiennej lokalnej). Służy do zwalniania zasobów zajętych przez obiekt.

3. **Czy można przeciążać konstruktory?**

   Odpowiedź: Tak, w C++ można definiować wiele konstruktorów w tej samej klasie, o ile różnią się listą parametrów.

---

8. Obsługa wyjątków i biblioteka STL w języku C++

Obsługa wyjątków

- **Wyjątki** pozwalają obsługiwać błędy (albo sytuacje wyjątkowe) w bardziej elegancki sposób niż tradycyjne kody błędów.
- Główne słowa kluczowe:
  - `try` – blok kodu, w którym może wystąpić wyjątek.
  - `throw` – służy do „rzucenia" wyjątku (np. obiektu typu `std::exception`).
  - `catch` – blok przechwytujący wyjątek.

Przykład

```
#include <iostream>
#include <stdexcept>

int dziel(int a, int b) {
    if(b == 0) {
        throw std::runtime_error("Dzielenie przez zero!");
    }
    return a / b;
}

int main() {
    try {
        std::cout << dziel(10, 2) << std::endl; // 5
        std::cout << dziel(10, 0) << std::endl; // rzucony wyjątek
    }
    catch(const std::runtime_error& e) {
        std::cerr << "Wyjatek: " << e.what() << std::endl;
    }
    return 0;
}
```

Biblioteka STL (Standard Template Library)

- Zawiera wiele przydatnych narzędzi, m.in.:
  - **Kontenery** (np. `std::vector`, `std::list`, `std::map`),
  - **Algorytmy** (np. `std::sort`, `std::find`, `std::accumulate`),
  - **Iteratory** (pozwalają poruszać się po kontenerach w ujednolicony sposób),
  - **Funkcje funkcyjne** (np. `std::function`).
- STL jest podstawą nowoczesnego C++ i ułatwia pisanie wydajnego i czytelnego kodu.

Przykład użycia `std::vector` i `std::sort`

```
#include <iostream>
#include <vector>
#include <algorithm> // std::sort

int main() {
    std::vector<int> liczby = {5, 1, 4, 2, 3};

    // Sortowanie
    std::sort(liczby.begin(), liczby.end());

    for(int i : liczby) {
        std::cout << i << " ";
    }
    // Wypisze: 1 2 3 4 5

    return 0;
}
```

Przykładowe pytania – Punkt 8

1. **Jakie są słowa kluczowe do obsługi wyjątków w C++?**

   Odpowiedź: `try`, `throw`, `catch`.

2. **Co to jest STL i dlaczego jest ważny?**

   Odpowiedź: STL (Standard Template Library) to biblioteka standardowa C++, zawiera kontenery, algorytmy, iteratory i inne narzędzia, które ułatwiają pracę z danymi. Jest podstawą programowania w nowoczesnym C++.

3. **Podaj przykład kontenera z STL i do czego służy.**

   Odpowiedź: `std::vector` – dynamiczna tablica, pozwalająca na efektywne dodawanie elementów na końcu i dostęp przez indeks.
