Programowanie obiektowe

Programowanie obiektowe inaczej nazywane programowaniem zorientowanym obiektowo. Jest to wzorzec programowania przy pomocy obiektów posiadających swoje właściwości, czy pola a także metody, które określają zachowania obiektu. Takie programowanie polega na definiowaniu obiektów i wywoływaniu ich metod w celu współpracy ich między soba.

Do lepszego zrozumienia programowania obiektowego należy wytłumaczyć kilka pojęć z nim związanych:

1. Klasa.
2. Obiekt
3. Właściwości i metody

Klasa jest to częściowa lub całkowita definicja obiektu. Opisuje właściwości i funkcje obiektu. Może być typem. Nie jest samodzienym bytem.
Obiekt jest instancją klasy. Na podstawie klas tworzymy konkretne obiekty.
Właściwości i metody to zawarte w klasie, na podstawie której tworzony jest obiekt. Zainicjonowany obiekt posiada właściwości i otrzymuje metody zapisane w klasie.

W definicji klasy napisane jest, że jest to częściowa lub całkowita definicja obiektu. Dlaczego częściowa? Tutaj pojawia się pojęcie dziedziczenia.

Dziedziczenie pozwala klasie pochodnej dziedziczyć metody, właściwości i pola z bazy bazowej. Dzięki temu zabiegowi kod jest czystszy, czytelniejszy i dużo krótszy, ponieważ się nie powtarza, a także staje się łatwo rozszerzalny. W programowaniu obiektowym istnieją trzy typy dziedziczenia:

1. Private
2. Protected
3. Public

Dziedziczenie prywatne oznacza, tyle że klasa pochodna składowe (właściwości, pola i metody) publiczne i chronione, klasy bazowej, dziedziczy jako składowe prywatne.
Dziedziczenie chronione (protected) oznacza, tyle że klasa pochodna składowe publiczne i chronione dziedziczy jako chronione.
Dziedziczenie publiczne oznacza, tyle że klasa pochodna składowe publiczne dziedziczy jako publiczne, a składowe chronione jako prywatne.

Składowych prywatnych w większości języków nie da się dziedziczyć. W niektórych językach możliwa jest opcja zaprzyjaźnienia klas, dzięki czemu umożliwione jest dziedziczenie składowych prywatnych, ale nie ma do nich dostępu klasa pochodna.

Z dziedziczeniem i hierarchią klas ważne jest pojęcie polimorfizmu.
Polimorfizm - filar programowania obiektowego, polega na różnym zachowaniu tych samych metod o tych samych deklaracjach w klasach będących w relacji dziedziczenia. Metodami polimorficznymi są metody wirtualne i metody abstrakcyjne.
W programowaniu może występować polimorfizm dynamiczny - to właśnie funkcje wirtualne i abstrakcyjne, oraz polimorfizm statyczny - przeciążanie funkcji i operatorów.

Metody wirtualne - tworzy się ją w klasie bazowej, a w klasach dziedziczących można polimorficznie przesłonić jej nazwę (słówko override) i zapewnić inną implementację. Metody te muszą być public.
Przykład metody wirtualnej w języku c#.

```
public class Pracownik
{
```

```
        virtual public void Pracuj() { Console.WriteLine("Pracuje pracownik");}
}
public class Sekretarka : Pracownik
{
        override public void Pracuj() { Console.WriteLine("Pracuje pracownik, dokładniej sekretarka");}
}
```

Po stworzeniu obiektu klasy Sekretarka, jego zainicjonowaniu i wywołaniu metody Pracuj() program wyświetli "Pracuje pracownik, dokładniej sekretarka".
Bez słówka kluczowego override program wykonałby metodę z klasy Pracownik.

Metody abstrakcyjne działają podobnie jak wirtualne z tą różnicą, że mogą występować w klasie tylko abstrakcyjnej. W języku c# klasy abstrakcyjne nie mogą mieć swojej instancji, a więc nie możemy stworzyć dla niej obiektu, dopiero dla klasy dziedziczącej po klasie abstrakcyjnej możemy stworzyć obiekt i go zainicjalizować.

W wielu językach istnieją również interfejsy, które są opisem reguł i zasad jak klasy mają być opisane. Klasa dziedzicząca interfejs musi posiadać implementację każdej z funkcji zapisanych w interfejsie. Sam interfejs nie posiada ciała funkcji, a także nie da się utworzyć obiektu interfejsu. Przykład w języku c#.

```
public interface IKalkulator
{
        public int dodawanie(int a, int b);
        public int odejmowanie(int a, int c);
}
public class Kalkulator : IKalkulator
{
        int a, b;
        int dodawanie(int a, int b)
        {
                return a + b;
        }
        int odejmowanie(int a int c)
        {
                return a - c;
        }
}
```