Programowanie II
**Lista 8**

## Dziedziczenie :

Dziedziczenie to nic innego jak definiowanie nowych klas w oparciu o już istniejące.

Jest to najważniejsza cecha świadcząca o sile programowania obiektowo zorientowanego.

Dzięki tej technice możliwe jest najbardziej naturalne odzwierciedlenie związków jakie zachodzą pomiędzy obiektami modelowanej w programie rzeczywistości.

## Przykład najprostszy:

```
class Tpunkt
{
 int x,y;
 void odbij_wzgledem_poczatku()
  {
   //...
  };
 void obroc(int kat)
  {
   //...
  };
};
```

Zawartość metod składowych została już dawno „wymyślona" podobnie jak i sama definicja punktu. Okazuje się jednak, już w przypadku ekranu komputerowego taka definicja już nie wystarcza. Klasę Tpunkt należało by bowiem rozszerzyć o wartości RGB, tak aby mógł on zostać poprawnie wyświetlony.

```
class TPunkt_monitor
{
 int x,y;
 int R,G,B;
  ……
  ……
};
```

Oczywiście w przypadku programowania strukturalnego musielibyśmy ponownie sprecyzować metody ***odbij_wzgledem_poczatku()*** i ***obroc(int kat)***, czyli musielibyśmy wykonać tę samą pracę ponownie. Dzięki jednak technice dziedziczenia możemy zaproponować nową klasę, wykorzystując możliwości starej:

```
class Tpunkt_monitor:public Tpunkt
{
 public:
 int R,G,B;
};
```

A następnie w programie głównym użyć konstrukcji:

```
Tpunkt_monitor punkt;
 punkt.x=10;
Punkt.obroc(90);
```

Fakty bardzo istotne:

- dziedziczenie nie polega na przekopiowaniu zawartości jednej klasy do drugiej. Składniki dziedziczone nadal mają zakres klasy podstawowej, na który to zostaje nałożony zakres klasy pochodnej
- w przypadku ponownego definiowania składników w klasie pochodnej o takich samych nazwach jak w klasie podstawowej tracimy bezpośredni dostęp do tych ostatnich
- aby odwołać się do pierwotnego znaczenia przesłoniętego składnika należy użyć kwalifikatora zakresu, np.:

**Tpunkt_monitor obiekt;**
**obiekt.wypisz();** *//metoda składowa klasy Tpunkt_monitor*
**Obiekt.Tpunkt::wypisz();** *//metoda składowa klasy Tpunkt*

- należy pamiętać, iż dziedziczenie dotyczy klas (czyli typów danych), a nie obiektów będących ich reprezentantami.

## Dziedziczenie kilkupokoleniowe :

```
class A {
//--- ciało klasy A
};
class B: public A {
//--- ciało klasy B
};
class C: public B  {
//--- ciało klasy C
};
```

## Wywoływanie konstruktorów :

```
class Tpunkt
{
 public:
 int x,y;
 Tpunkt()
 {
  cout<<"Konstruktor punktu"<<"\n";
 };
};

class Tpunkt_monitor:public Tpunkt
{
 public:
 int R,G,B;
 Tpunkt_monitor()
 {
  cout<<"Konstruktor punktu ekranowego"<<"\n";
 };
};

class TPunkt_3D:public Tpunkt_monitor
{
 public:
 TPunkt_3D()
 {
```

```
  cout<<"Konstruktor punktu 3D"<<"\n";
 };
};
```

## Dziedziczenie, a dostęp do składników

Mechanizm dziedziczenia pozwala określać w jaki sposób mają być dziedziczone składniki klasy podstawowej.

Prywatne składniki klasy podstawowej są dziedziczone, ale nie ma do nich dostępu.

Nie prywatne składniki klasy podstawowej są dostępne w zakresie klasy pochodnej bezpośrednio.

## Dziedziczenie, a składniki chronione

Składniki chronione (PROTECTED) traktowaliśmy do tej pory jak prywatne.

W praktyce klauzula PROTECTED oznacza, że składnik jest zastrzeżony tylko dla klas pochodnych. W praktyce oznacza to, że składnik typu PROTECTED:

- dla całego świata powinien być traktowany jak prywatny czyli niedostępny
- dla wszystkich klas pochodnych od danej klasy powinien być dostępny, tak jakby był składnikiem publicznym.

Można zatem powiedzieć, iż to klasa podstawowa poprzez odpowiednią specyfikację dostępu decyduje, które składniki zamierza udostępnić otoczeniu, a które tylko klasom pochodnym.

## Sposoby dziedziczenia

Również klasa pochodna ma możliwość określania sposobu w jaki chce dziedziczyć składniki klasy podstawowej.
Wybór taki dokonywany jest w momencie definicji klasy pochodnej:
**class Tpunkt_monitor:public Tpunkt**
Słowo **public** może zostać zastąpione **protected** lub **private**.
Dzięki temu mamy możliwość określenia sposobu dziedziczenia składników nie prywatnych (!!!).
Zasada definiowania sposobu dziedziczenia jest prosta:

- **PUBLIC** powoduje, że wszystkie składniki publiczne dziedziczone są w sposób publiczny, chronione jako chronione, a prywatne jako prywatne
- **PROTECTED** powoduje, że wszystkie składniki publiczne i chronione dziedziczone są jako chronione
- **PRIVATE** powoduje, że wszystkie składniki dziedziczone są jako prywatne.

**Zadanie 1 (10 pkt)**
Zrealizuj zadanie 12.2 z kursu Logika programowania na platformie edukacyjnej – oczywiście w C++

**Zadanie 2 (5 pkt)**
Rozbuduj przykład z zadania 1 o wykorzystanie konstruktorów jak w przykładach z kursu

**Zadanie 3 (10 pkt)**
Proszę zoptymalizować strukturę klas swojej wersji gry Monopolly. Oczywiście modyfikacja ma dotyczyć wykorzystania mechanizmów dziedziczenia.