Podstawy Informatyki

OOP – dziedziczenie i polimorfizm

06.12.2019

dr inż. Michał Rad

Plan wykładu:

1. Powtórka
2. Dziedziczenie
3. Polimorfizm
4. Klasy wirtualne

Programowanie obiektowe

- Opis problemu za pomocą obiektów
- Obiekty (klasy) mogą zawierać własne funkcje (metody)
- Istnieje możliwość ograniczenia dostępu do metod i składowych
- Metody mają pełen dostęp do składowych własnej klasy

Programowanie obiektowe *cd..*

- Metody specjalne: konstuktory i destruktor
- Uwaga na “wyciek pamięci”
- Przeciążanie operatorów
- Opis klas w UML

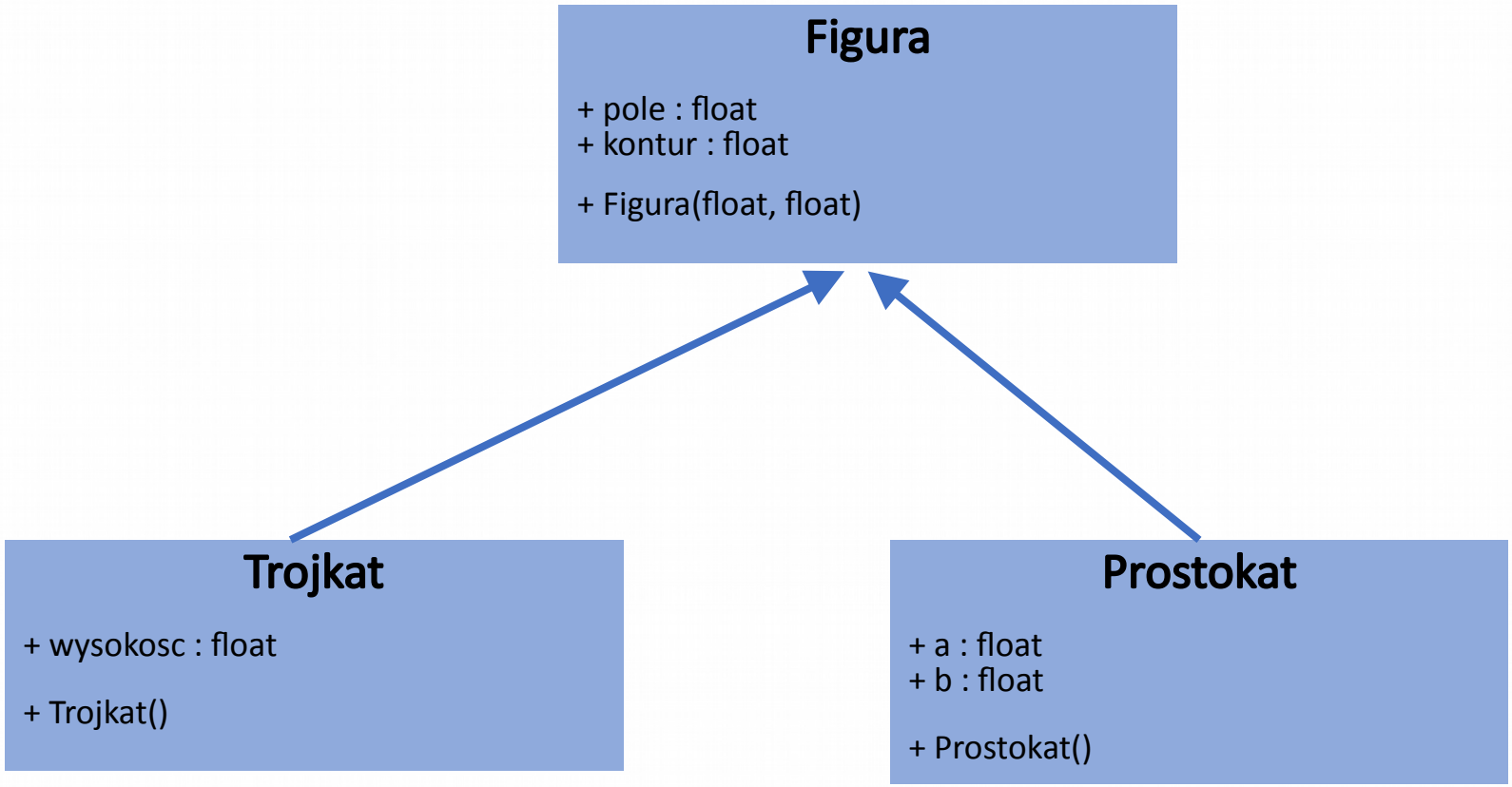
Dziedziczenie

- Kolejną bardzo ważną cechą programowania obiektowego jest możliwość dziedziczenia
- Dziedziczenie to nic innego jak zawieranie obiektu “rodzica” w obiekcie “potomnym”

Przykład dziedziczenia

```
class Figura
{
  public:
   float pole;
   float kontur;
};
class Trojkat : public Figura
{
  public:
   float wysokosc;
}
```

Jak zapisać dziedziczenie? *(UML)*



Co daje dziedziczenie?

- Oszczędność kodu – brak potrzeby powielania kodu co jest wygodniejsze i bezpieczniejsze
- Zapewnienie zestawu tych samych cech obiektom wyprowadzonym z tej samej klasy bazowej

Dostęp do dziedziczonych składowych i funkcji

Specyfikator dostępu klasy bazowej	Typ dziedziczenia		
	public	**protected**	**private**
public	public	protected	private
protected	protected	protected	private
private	n/a Ukryte	n/a Ukryte	n/a Ukryte

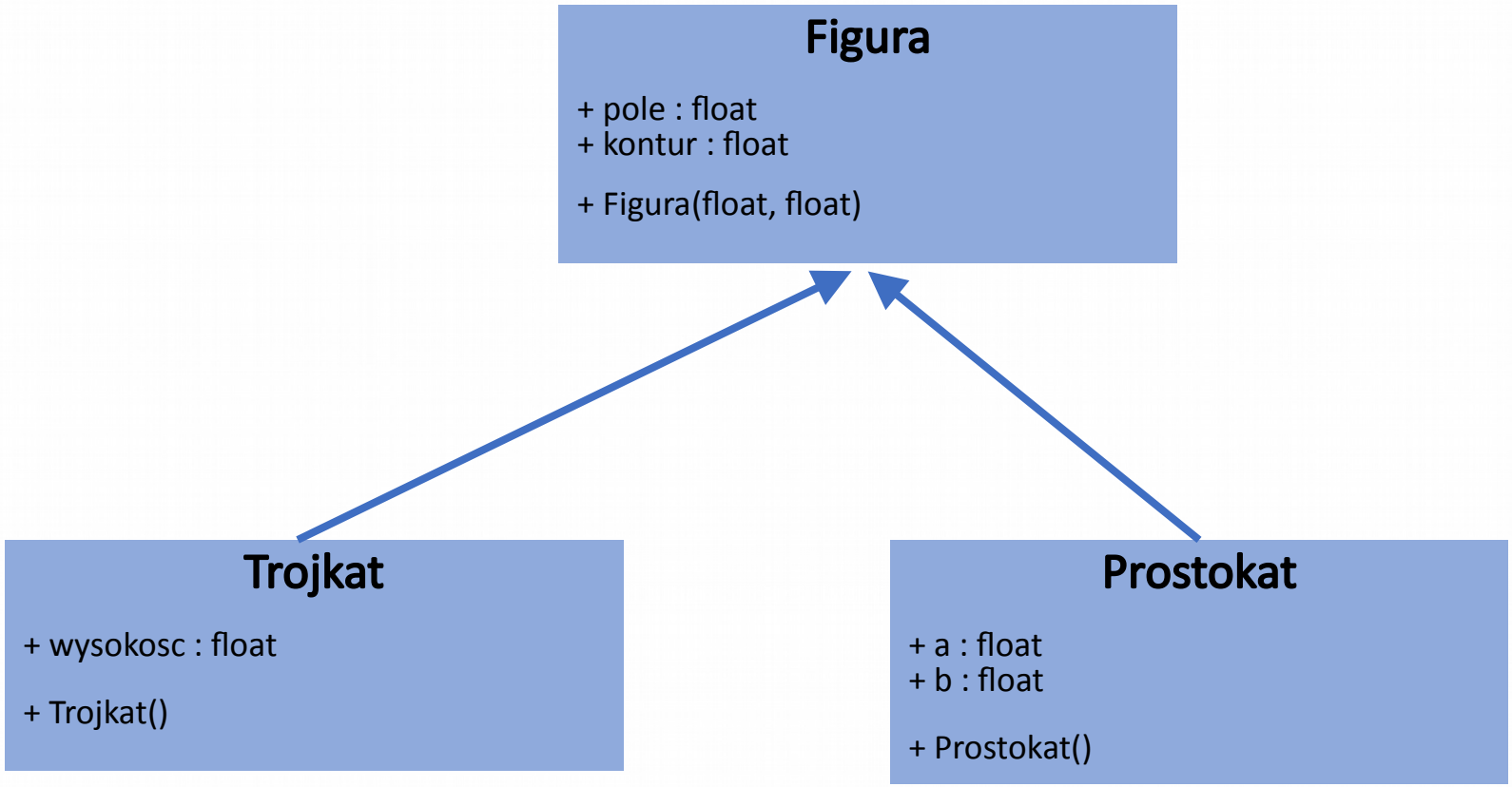


Figura
+ pole : float
+ kontur : float
+ Figura(float, float)
Trojkat
+ wysokosc : float
+ Trojkat()
Prostokat
+ a : float
+ b : float
+ Prostokat()

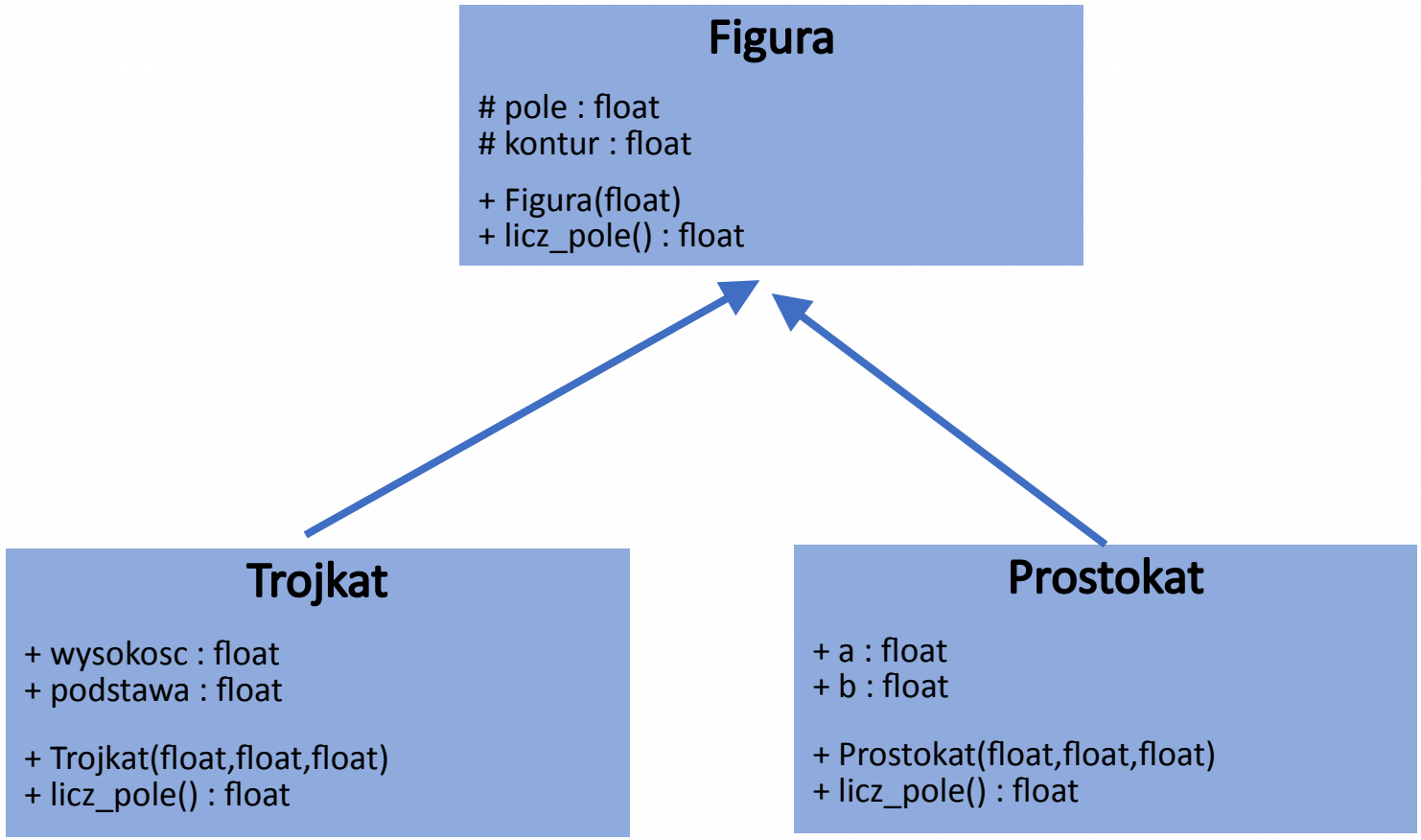


Figura
pole : float
kontur : float
+ Figura(float)
+ licz_pole() : float
Trojkat
+ wysokosc : float
+ podstawa : float
+ Trojkat(float,float,float)
+ licz_pole() : float
Prostokat
+ a : float
+ b : float
+ Prostokat(float,float,float)
+ licz_pole() : float

```
class Figura
{
  protected:
   float pole;
   float kontur;
  public:
   float licz_pole();
   Figura(float _kontur);
};
class Trojkat : public Figura
{
  public:
   float podstawa;
   float wysokosc;
   Trojkat(float p,float w);
}
```

```
Figura::Figura(float _kontur)
{
kontur=_kontur;
};

Trojkat::Trojkat(float p, float w, float kont):
Figura(kont),podstawa(p),wysokosc(w)
{
//nothing to do
}
```

Tworzenie obiektów potomnych

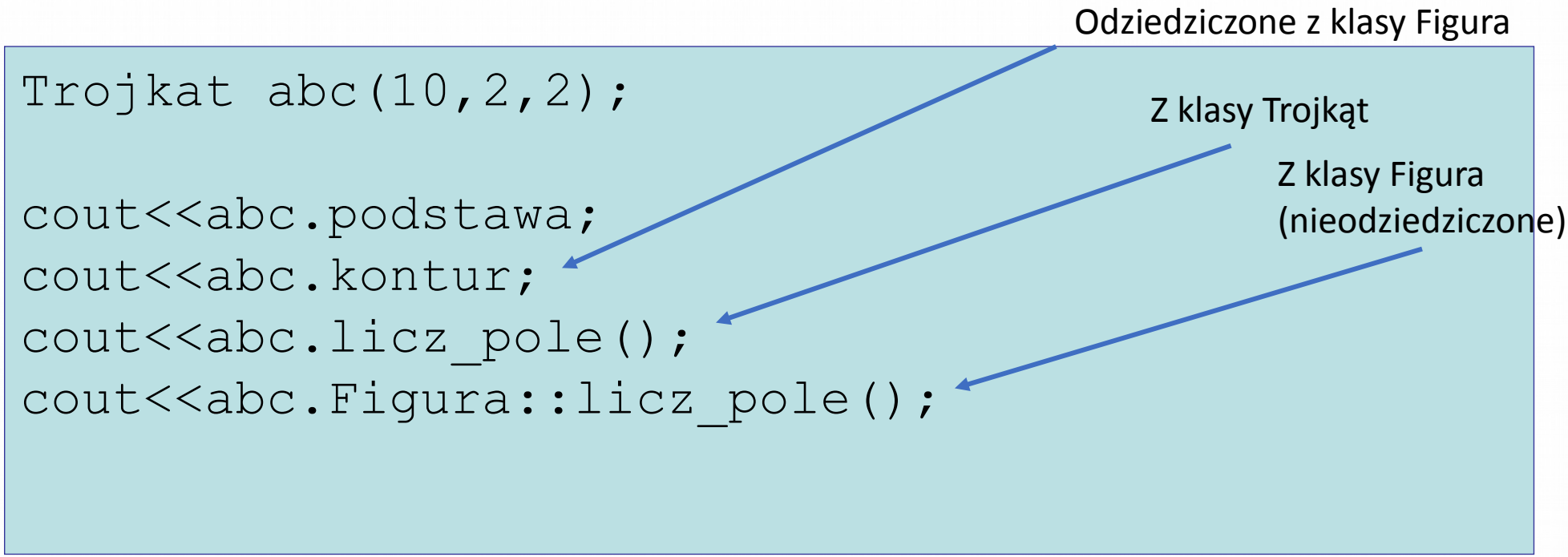
- Aby powstał obiekt potomny muszą powstać (najpierw) obiekty bazowe
- Warto pamiętać o tym aby było to możliwe

- aby zdefiniowane były odpowiednie konstruktory

Usuwanie obiektów potomnych

» W czasie usuwania obiektów potomnych usuwane są również obiekty bazowe

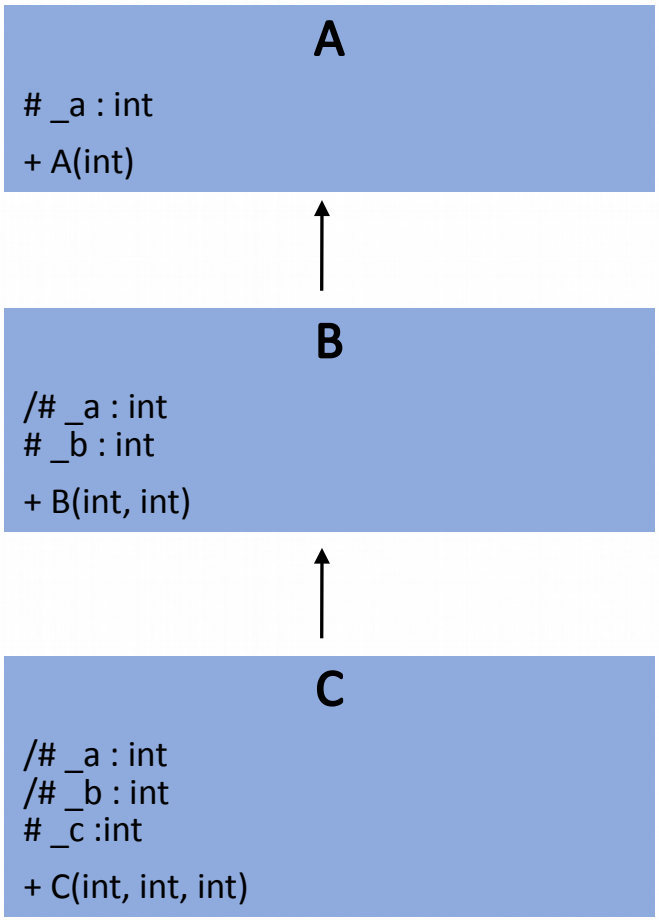
Dostęp do dziedziczonych funkcji i składowych



Co jest dziedziczone?

» Wszystko prócz:

- Konstruktorów i destruktorów
- definicji operatora przypisania
- Klas zaprzyjaźnionych w stosunku do bazowych (friend)




```
A::A(int a) :
  _a(a)
{ cout << "A!" << endl; }
```

```
B::B(int a, int b) :
  A(a), _b(b)
{ cout << "B!" << endl; }
```

```
C::C(int a, int b, int c) :
  B(a, b), _c(c)
{ cout << "C!" << endl;}
```

```
C objectC(1, 2, 3);
```

STOS

(int _a) 1 | 1500
(int _b) 2 | 1504
(int _c) 3 | 1508

Możliwe kombinacje

```
Trojkat abc(10,2,2);
Figura *wsk = &abc;
Figura &ref = abc;

void fun(Figura * fig);

fun(abc);
```

Możliwe kombinacje - slicing

```
Trojkat abc(10,2,2);
Figura *wsk = &abc;
Figura &ref = abc;

wsk->podstawa;  // niedostępne!!! Błąd
wsk->licz_pole(); // jedynie z Figury
```

» Slicing – “obcinanie” dostępu do składowych i funkcji

Polimorfizm

- Aby jednak wywołać funkcję obiektu potomnego z poziomu wskaźnika do obiektu bazowego należy “włączyć” polimorfizm
- W tym celu wystarczy dodać słówko “virtual” przed deklaracją funkcji w obiekcie bazowym

```
class Figura
{
  protected:
   float pole;
   float kontur;
  public:
   virtual float licz_pole();
   Figura(float _kontur);
};
class Trojkat : public Figura
{
  public:
   float podstawa;
   float wysokosc;
   Trojkat(float p,float w);
}
```

Włączamy polimorfizm

Polimorfizm

```
Trojkat abc(10,2,2);
Figura *wsk = &abc;
Figura &ref = abc;

wsk->podstawa;   // niedostępne!!! Błąd
wsk->licz_pole();// dzięki polimorfizmowi
        // wykona się funkcja z
//Trójkata faktycznie liczaca pole
```

Co to daje?

- Daje to możliwość tworzenia tablicy różnych obiektów wskazywanych wskaźnikiem tego samego typu – typu bazowego
- Daje również jednolity dostęp do funkcji dla zadeklarowanych jako wirtualne w klasie bazowej a implementowanych w klasach pochodnych

Przykład

```
Trojkat abc(10,2,2);
Kwadrat ab(4,3);
Prostokat abcd(4,3,2);
Figura *tab[3];
tab[0]=&abc;
tab[1]=&ab;
tab[2]=&abcd;
for(int i=0;i<3;i++)
{
tab[i]->licz_pole();
}
```

A co z funkcją licz_pole dla Figury?

Dwie możliwości:

» Albo trzeba ją jakoś zdefiniować

» Albo określić jako czysto wirtualną:

```
class Figura
{
  protected:
   float pole;
   float kontur;
  public:
   virtual float licz_pole() = 0;
   Figura(float _kontur);
};
```

Klasy abstrakcyjne

- Klasy które zawierają choć jedną funkcję czysto wirtualną to klasy **abstrakcyjne**
- Nie można powoływać do życia obiektów klas abstrakcyjnych
- Można natomiast używać wskaźników typu takich klas

Przykład

» W kodzie…

AGH

Quiz!