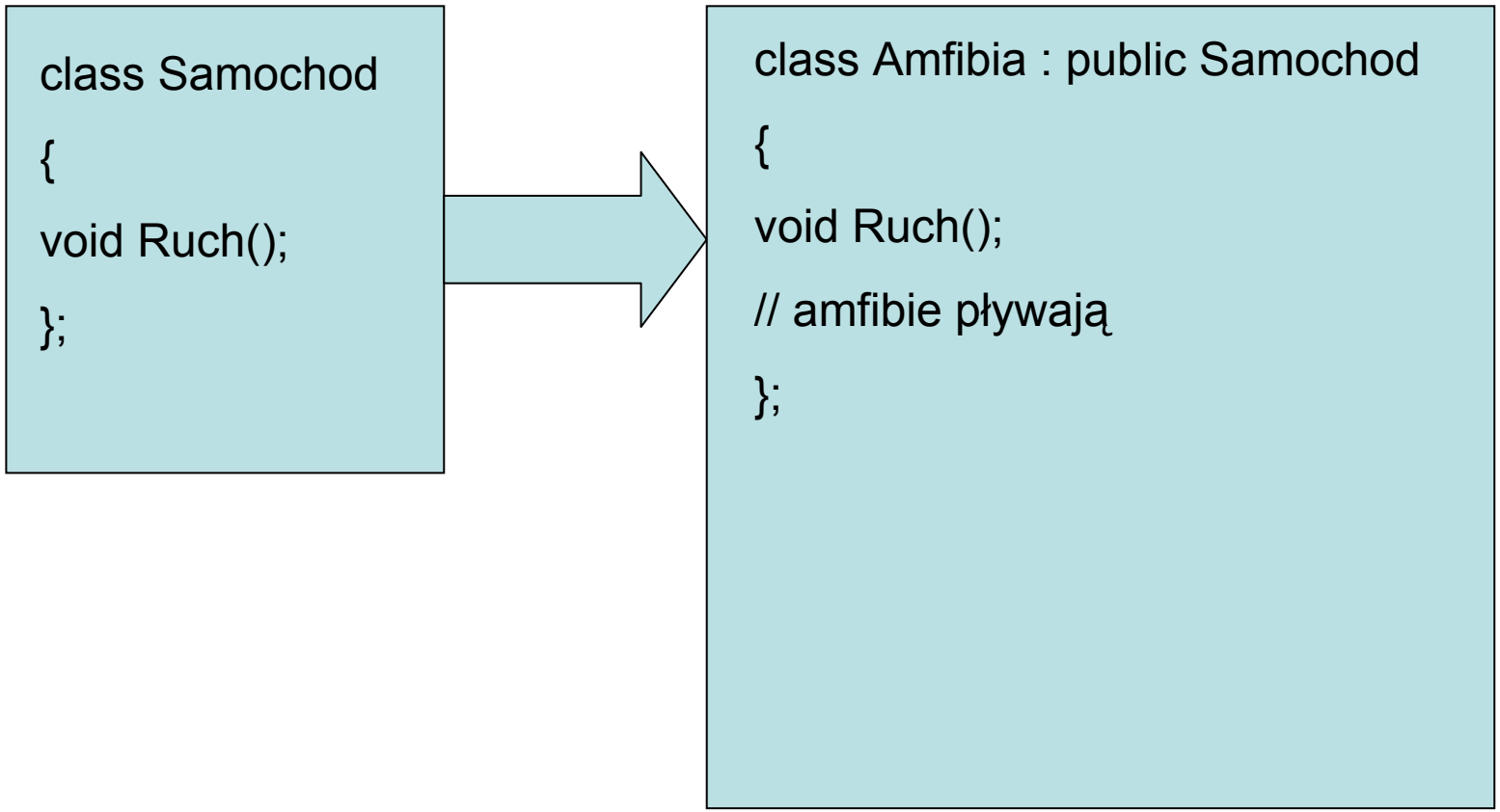

class Samochod
{
void Ruch();
};
class Amfibia : public Samochod
{
void Ruch();
// amfibie pływają
};

Rzutowanie w C++

Mechanizm pozwalający na tymczasową lub stałą zmianę interpretacji obiektu przez kompilator – nie powoduje rzeczywistej zmiany samego obiektu:

1. Nowa funkcja w klasie Samochod (np. Jazda() zamiast Ruch() i przesłonięcie funkcji Ruch(), tak, żeby wykonywała funkcję Plywanie().
2. Dodanie funkcji Plywanie() w klasie Samochod (przenoszenie w górę)
3. Zatrzymanie funkcji Plywanie() w klasie Amfibia i uruchamianie jej tylko wtedy, gdy wskaźnik do obiektu wskazuje amfibię. Konieczna możliwość spytania wskaźnika jaki obiekt wskazuje - RTTI (Run Time Type Identification) – operatory static_cast, dynamic_cast.
4. Stworzenie klasy Pojazd, w której istnieją funkcje z klasy Statek, jak i Samochod.

```
/* Rozwiązanie nr 1 i 2 - tworzymy
dodatkową funkcję w klasie Samochod*/

#include <iostream>
using namespace std;

class Samochod
{
public:
	Samochod ()
	{cout<<"Konstruktor samochodu...\n";}
	void Jazda()
	{cout<<"Catch me if you can !!\n";}
	virtual void Ruch()
	{cout<<"Help me !! I'm drowning !!\n";}
	virtual ~Samochod()
	{cout<<"Zderzenie z TIRem na S8 !!!...\n";}
};
```

```
class Amfibia : public Samochod
{
public:
	Amfibia ()
	{cout<<"Konstruktor amfibii...\n";}
	void Ruch()
	{cout<<"Plywam i nie tone !!\n";}
	virtual ~Amfibia()
	{cout<<"Dziura w dnie !!!...\n";}
};

void main()
{
	Samochod* Garaz[3];
	Samochod* ptr;
	int wybor, i;
	for (i = 0; i<3; i++)
	{
		cout<<"(1) samochod (2) amfibia:";
		cin>>wybor;
```

```
if (wybor == 2)
                        ptr = new Amfibia;
                else
                        ptr = new Samochod;
                Garaz[i]= ptr;
                cout<<endl;
        }
cout<<endl;
        for (i = 0; i < 3; i++)
        {
                Garaz[i]->Jazda();
                Garaz[i]->Ruch();
                delete Garaz[i];
                cout<<endl;
        }
        system("pause");
}
```

```
(1) samochod (2) amfibia:1
Konstruktor samochodu...

(1) samochod (2) amfibia:2
Konstruktor samochodu...
Konstruktor amfibii...

(1) samochod (2) amfibia:1
Konstruktor samochodu...


Catch me if you can !!
Help me !! I'm drowning !!
Zderzenie z TIRem na S8 !!!...

Catch me if you can !!
Plywam i nie tone !!
Dziura w dnie !!!...
Zderzenie z TIRem na S8 !!!...

Catch me if you can !!
Help me !! I'm drowning !!
Zderzenie z TIRem na S8 !!!...

Press any key to continue . . .
```

```
/* Rozwiązanie nr 1 i 2 -
tym razem obiekty*/

#include <iostream>
using namespace std;

class Ssak
{
public:
	void Rozmnazanie()
	{cout<<"Rozmnazanie ssaka\n";}
	virtual void Reproduce()
	{cout<<"Rozmnazanie ssaka(virtual)\n";}
};

class Dziobak : public Ssak
{
public:
	void Rozmnazanie()
	{cout<<"Dziobak znosi jaja !!\n";}
	virtual void Reproduce()
	{cout<<"Dziobak znosi jaja (virtual)!!\n";}
};
```

```
void main()
{
	cout<<"Obiekty : \nslon\n";
	Ssak slon;
	Dziobak dziobak;
	slon.Rozmnazanie();
	slon.Reproduce();
	cout<<"\ndziobak:\n";
	dziobak.Rozmnazanie();
	dziobak.Reproduce();

	cout<<"\nWskazniki : \nslon\n";
	Ssak* slon1=new Ssak;
	slon1->Rozmnazanie();
	slon1->Reproduce();
	Dziobak* dziobak1=new Dziobak;
	cout<<"\ndziobak:\n";
	dziobak1->Rozmnazanie();
	dziobak1->Reproduce();
	system("pause");
}
```

```
Obiekty :
slon
Rozmnazanie ssaka
Rozmnazanie ssaka(virtual)

dziobak:
Dziobak znosi jaja !!
Dziobak znosi jaja (virtual)!!

Wskazniki :
slon
Rozmnazanie ssaka
Rozmnazanie ssaka(virtual)

dziobak:
Dziobak znosi jaja !!
Dziobak znosi jaja (virtual)!!
Press any key to continue . . .
```

```
/* Rozwiązanie nr 3 - sprawdzenie rtti*/

#include <iostream>
using namespace std;

class Samochod
{
public:
	Samochod ()
	{cout<<"Konstruktor samochodu...\n";}
	void Jazda()
	{cout<<"Catch me if you can !!\n";}
	virtual ~Samochod()
	{cout<<"Zderzenie z TIRem na S8 !!!...\n";}
};

class Amfibia : public Samochod
{
public:
	Amfibia ()
	{cout<<"Konstruktor amfibii...\n";}
	void Plyn()
	{cout<<"Plyne i nie tone !!\n";}
```

```
virtual ~Amfibia()
	{cout<<"Dziura w dnie !!!...\n";}
};

void main()
{
	Samochod* Garaz[3];
	Samochod* ptr;
	int wybor, i;
	for (i = 0; i<2; i++)
	{
		cout<<"(1) samochod (2) amfibia:";
		cin>>wybor;
		if (wybor == 2)
			ptr = new Amfibia;
		else
			ptr = new Samochod;
		Garaz[i]= ptr;
		cout<<endl;
	}
```

```
for (i = 0; i < 2; i++)
{
        cout<<"\nPojazd numer "<<i<<
        "\nUzycie dynamic_cast\n\n";
        //sprawdzenie wskaźnika do klasy bazowej
        Amfibia *ptr =dynamic_cast<Amfibia*> (Garaz[i]);

        if (ptr !=NULL)
        {
                ptr->Jazda();
                ptr->Plyn();
        }

        else
        {
                ptr->Jazda();
                cout<<"To zwykly samochod –
                nie plywa !...\n";
        }
```

```
cout<<"\nPojazd numer "<<i<<"\nUzycie static_cast\n\n";
            //sprawdzenie wskaźnika do klasy bazowej
            Amfibia *ptr1 = static_cast<Amfibia*> (Garaz[i]);

            if (ptr1 !=NULL)
            {
                  ptr1->Jazda();
                  ptr1->Plyn();
            }

            else
            {
                  ptr1->Jazda();
                  cout<<"To zwykly samochod –
                  nie plywa !...\n";
            }
            delete Garaz[i];
      }
      system("pause");
}
```

```
(1) samochod (2) amfibia: 1
Konstruktor samochodu...

(1) samochod (2) amfibia: 2
Konstruktor samochodu...
Konstruktor amfibii...


Pojazd numer 0
Uzycie dynamic_cast

Catch me if you can !!
To zwykly samochod - nie plywa !...

Pojazd numer 0
Uzycie static_cast

Catch me if you can !!
Plyne i nie tone !!
Zderzenie z TIRem na S8 !!!...
```

```
Pojazd numer 1
Uzycie dynamic_cast

Catch me if you can !!
Plyne i nie tone !!

Pojazd numer 1
Uzycie static_cast

Catch me if you can !!
Plyne i nie tone !!
Dziura w dnie !!!...
Zderzenie z TIRem na S8 !!!...
Press any key to continue . . .
```

Dziedziczenie wielokrotne

Polega na dziedziczeniu z więcej niż jednej klasy bazowej:

```
class klasa_pochodna: public klasa_bazowa_1,
public klasa_bazowa_2… {};
```

```
/*Dziedziczenie wielokrotne*/

#include <iostream>
using namespace std;

class Ssak
{
public:
	Ssak()
	{cout<<"Konstruktor klasy Ssak ....\n";}
	virtual void Rozmnazanie()
	{cout<<"Rozmnazanie ssaka\n";}
	virtual void Glos()
	{cout<<"Glos ssaka\n";}
	virtual ~Ssak()
	{cout<<"Destruktor klasy Ssak ....\n";}
};
```

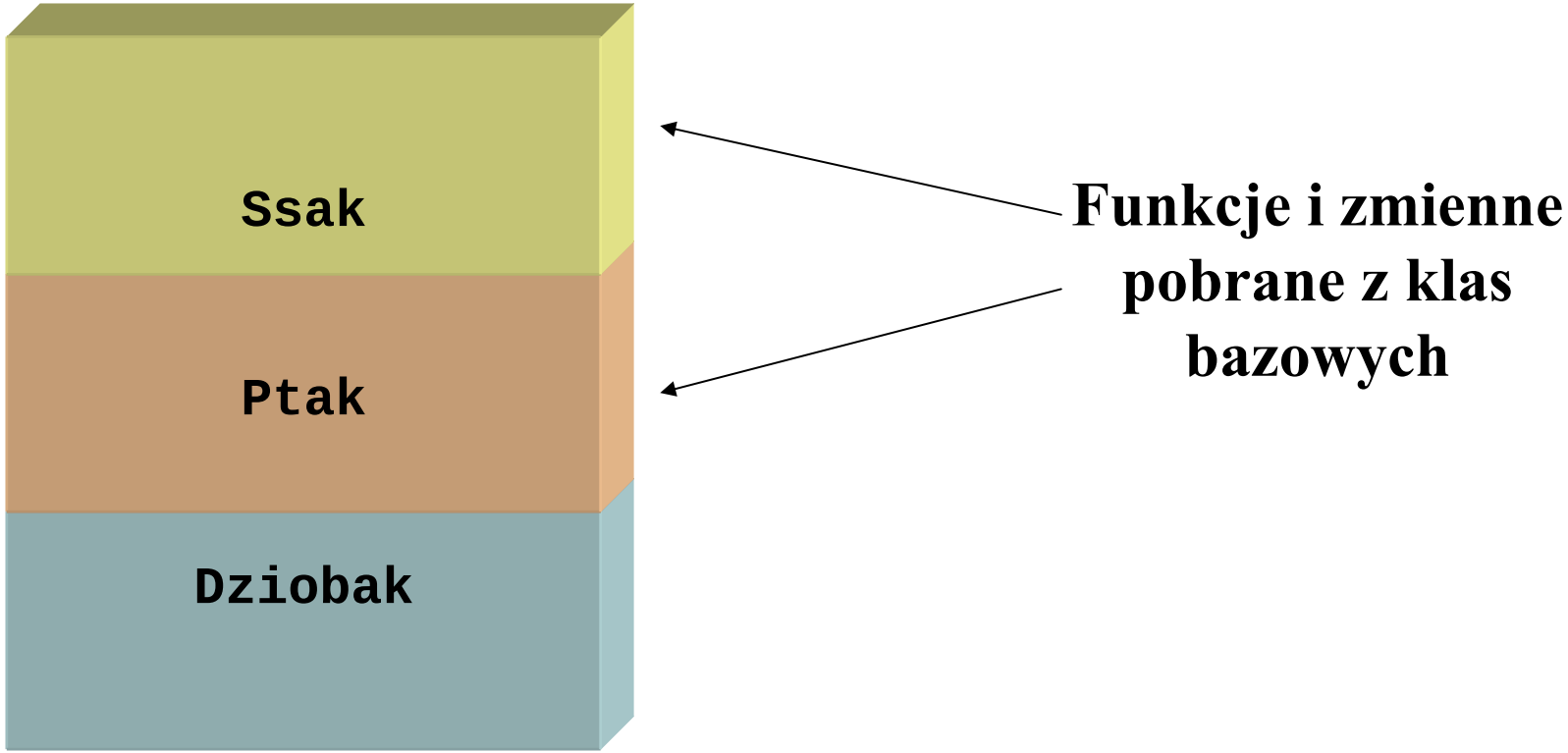
```
class Ptak
{
public:
	Ptak()
	{cout<<"Konstruktor klasy Ptak ....\n";}
	virtual void Rozmnazanie()
	{cout<<"Ptak znosi jaja\n";}
	virtual void Glos()
	{cout<<"Glos ptaka\n";}
	virtual ~Ptak()
	{cout<<"Destruktor klasy Ptak ....\n";}
};

class Dziobak : public Ssak, public Ptak
{
public:
	void Rozmnazanie()
	{cout<<"Dziobak znosi jaja !!\n";}
	void Glos()
	{cout<<"Glos dziobaka !!\n";}
	virtual ~Dziobak()
	{cout<<"Destruktor klasy Dziobak ....\n";}
};
```

```
void main()
{
	Ssak* ZOO[2];
	Ptak* Kurnik[2];
	Ssak* pSsak;
	Ptak* pPtak;
	int wybor, i;
	for (i = 0; i<2; i++)
	{
		cout<<"\n(1) Ssak (2) Dziobak:";
		cin>>wybor;
		if (wybor == 2)
			pSsak = new Dziobak;
		else
			pSsak = new Ssak;
		ZOO[i]= pSsak;
	}
	for (i = 0; i<2; i++)
	{
		cout<<"\n(1) Ptak (2) Dziobak:";
```

```
cin>>wybor;
	if (wybor == 2)
		pPtak = new Dziobak;
	else
		pPtak = new Ptak;
	Kurnik[i]= pPtak;
}
cout<<endl;
for (i = 0; i < 2; i++)
{
	cout<<"\nZOO["<<i<<"]:\n";
	ZOO[i]->Glos();
	ZOO[i]->Rozmnazanie();
	delete ZOO[i];
}
for (i = 0; i < 2; i++)
{
	cout<<"\nKurnik["<<i<<"]:\n";
	Kurnik[i]->Glos();
	Kurnik[i]->Rozmnazanie();
	delete Kurnik[i];
}
system("pause"); }
```

```
(1) Ssak (2) Dziobak:1
Konstruktor klasy Ssak ....

(1) Ssak (2) Dziobak:2
Konstruktor klasy Ssak ....
Konstruktor klasy Ptak ....

(1) Ptak (2) Dziobak:1
Konstruktor klasy Ptak ....

(1) Ptak (2) Dziobak:2
Konstruktor klasy Ssak ....
Konstruktor klasy Ptak ....
```

```
ZOO[0]:
Glos ssaka
Rozmnazanie ssaka
Destruktor klasy Ssak ....

ZOO[1]:
Glos dziobaka !!
Dziobak znosi jaja !!
Destruktor klasy Dziobak ....
Destruktor klasy Ptak ....
Destruktor klasy Ssak ....

Kurnik[0]:
Glos ptaka
Ptak znosi jaja
Destruktor klasy Ptak ....

Kurnik[1]:
Glos dziobaka !!
Dziobak znosi jaja !!
Destruktor klasy Dziobak ....
Destruktor klasy Ptak ....
Destruktor klasy Ssak ....
Press any key to continue . . .
```


Ssak
Ptak
Dziobak
Funkcje i zmienne pobrane z klas bazowych

Problemy przy dziedziczeniu wielokrotnym:

1. Co się dzieje, gdy obie klasy bazowe mają dane lub funkcje o tych samych nazwach ?
2. Jak są inicjalizowane konstruktory klas bazowych ?
3. Co się dzieje, gdy obie klasy bazowe dziedziczą z tej samej klasy ?

```
/*Dziedziczenie wielokrotne
Jawne użycie funkcji klas bazowych */

#include <iostream>
using namespace std;

class Ssak
{
public:
	Ssak()
	{cout<<"Konstruktor klasy Ssak ....\n";}
	virtual void Rozmnazanie()
	{cout<<"Rozmnazanie ssaka\n";}
	void Glos()
	{cout<<"Glos ssaka\n";}
	virtual ~Ssak()
	{cout<<"Destruktor klasy Ssak ....\n";}
};
```

```
class Ptak
{
public:
	Ptak()
	{cout<<"Konstruktor klasy Ptak ....\n";}
	virtual void Rozmnazanie()
	{cout<<"Znosi jaja\n";}
	virtual void Glos()
	{cout<<"Glos ptaka\n";}
	virtual ~Ptak()
	{cout<<"Destruktor klasy Ptak ....\n";}
};

class Dziobak : public Ptak, public Ssak
{
public:
	void Rozmnazanie()
	{return Ptak::Rozmnazanie();}
	void Glos()
	{return Ssak::Glos();}
	virtual ~Dziobak()
	{cout<<"Destruktor klasy Dziobak ....\n";}
};
```

```
void main()
{
	Ssak* ZOO[2];
	Ptak* Kurnik[2];
	Ssak* pSsak;
	Ptak* pPtak;
	int wybor, i;
	for (i = 0; i<2; i++)
	{
		cout<<"\n(1) Ssak (2) Dziobak:";
		cin>>wybor;
		if (wybor == 2)
			pSsak = new Dziobak;
		else
			pSsak = new Ssak;
		ZOO[i]= pSsak;
	}
	for (i = 0; i<2; i++)
	{
		cout<<"\n(1) Ptak (2) Dziobak:";
		cin>>wybor;
```

```
        if (wybor == 2)
                pPtak = new Dziobak;
        else
                pPtak = new Ptak;
        Kurnik[i]= pPtak;
}
cout<<endl;

for (i = 0; i < 2; i++)
{
        cout<<"\nZOO["<<i<<"]:\n";
        ZOO[i]->Glos();
        ZOO[i]->Rozmnazanie();
        delete ZOO[i];
}
for (i = 0; i < 2; i++)
{
        cout<<"\nKurnik["<<i<<"]:\n";
        Kurnik[i]->Glos();
        Kurnik[i]->Rozmnazanie();
        delete Kurnik[i];
}
system("pause");    }
```

```
(1) Ssak (2) Dziobak:1
Konstruktor klasy Ssak ....

(1) Ssak (2) Dziobak:2
Konstruktor klasy Ptak ....
Konstruktor klasy Ssak ....

(1) Ptak (2) Dziobak:1
Konstruktor klasy Ptak ....

(1) Ptak (2) Dziobak:2
Konstruktor klasy Ptak ....
Konstruktor klasy Ssak ....
```

```
ZOO[0]:
Glos ssaka
Rozmnazanie ssaka
Destruktor klasy Ssak ....

ZOO[1]:
Glos ssaka
Znosi jaja
Destruktor klasy Dziobak ....
Destruktor klasy Ssak ....
Destruktor klasy Ptak ....

Kurnik[0]:
Glos ptaka
Znosi jaja
Destruktor klasy Ptak ....

Kurnik[1]:
Glos ssaka
Znosi jaja
Destruktor klasy Dziobak ....
Destruktor klasy Ssak ....
Destruktor klasy Ptak ....
Press any key to continue . . .
```

```
/*Dziedziczenie wielokrotne
Użycie wielu konstruktorów*/

#include <iostream>
using namespace std;

char* naped_ziemia[]={"benzyna", "diesel" ,"hybryda"};
char* naped_woda[]={"diesel","turbina"};

class Samochod
{
public:
char* silnik;
int predkosc;
      Samochod (int,int);
      void Jazda(int v)
      {cout<<"Jade "<<v<<" km/h \n";}
};
```

```
class Statek
{
public:
	char* silnik;
	double szybkosc;
	Statek (int,int);
	void Plyne(int v)
	{cout<<"Plyne z predkoscia "<<v<<" wezlow\n";}
};

class Amfibia : public Samochod, public Statek
{
	int ladownosc;
public:
	Amfibia(int,int,int,int,int);
	Amfibia();

	void Ladownosc()
	{cout<<"Nie zatone przy ciezarze "<<ladownosc<<" ton
!!\n";}
};
```

```
Samochod::Samochod(int i, int pred):
silnik(naped_ziemia[i]),predkosc(pred){}

Statek::Statek(int a, int v):
silnik(naped_woda[a]),szybkosc(v){}

Amfibia::Amfibia(int i1,int v1,int i2, int v2, int masa)
:Samochod(i1,v1),Statek(i2,v2),ladownosc(masa)
{cout<<"Konstruktor z parametrami\n";}

/*Amfibia::Amfibia(int i1,int v1,int i2, int v2, int masa)
{
        Samochod(i1,v1);
        Statek(i2,v2);
        ladownosc(masa);
}*/

Amfibia::Amfibia():
        Samochod(1,130),
        Statek(1,15)
        {ladownosc=2;
        cout<<"Konstruktor domyslny\n";}
```

```
void main()
{
	cout<<"Aligator\n\n";
	Amfibia aligator(0,70,0,6,2);
	aligator.Jazda(30);
	aligator.Ladownosc();
	aligator.Plyne(6);
	//cout<<aligator.silnik<<endl;
	cout<<aligator.Samochod::silnik<<endl;
	cout<<aligator.Statek::silnik<<endl;
	cout<<"Kubelwagen\n\n";
	Amfibia* kubelwagen= new Amfibia;
	kubelwagen->Jazda(25);
	kubelwagen->Plyne(8);
	kubelwagen->Ladownosc();
	cout<<kubelwagen->predkosc<<endl;
	cout<<kubelwagen->szybkosc<<endl;
	system("pause");
}
```

```
Aligator

Konstruktor z parametrami
Jade 30 km/h
Nie zatone przy ciezarze 2 ton !!
Plyne z predkoscia 6 wezlow
benzyna
diesel
Kubelwagen

Konstruktor domyslny
Jade 25 km/h
Plyne z predkoscia 8 wezlow
Nie zatone przy ciezarze 2 ton !!
130
15
Aby kontynuować, naciśnij dowolny klawisz . . .
```

Metody eliminacji niejednoznaczności:

1. Jawne wywołanie funkcji, którą chcemy użyć
2. Użycie funkcji wirtualnej w klasie bazowej