Rozwiązanie:

```
(---1----):
namespace cmp {
(---2----):
complex operator+(complex);
operator int();
(---3----):
cmp::complex cmp::complex::operator+(complex c){
(---4----):
cmp::complex::operator int(){
return static_cast<int>(re);
}
(---5----):
virtual double pole(){return -1;}
(---6----):
kolo::kolo(string o, punkt s, double p):
      figura(o), sr(s), prom(p){}
double kolo::pole(){
     return 4.*atan(1.)*prom*prom;
}
(---7----):
ostream& operator<<(ostream& strum, cmp::complex c){
     strum << "(" << c.re << ", " << c.im << ")";
     return strum;
}
ostream& operator<<(ostream& strum, punkt p){
     strum << "(" << p.x << ", " << p.y << ")";
     return strum;
}
(---8----):
template<typename Typ, int i> class MojaTab{
public:
   Typ Tablica[i];
   Typ& operator[](int index){return Tablica[index];}
};
(---9----):
using namespace cmp;
(---10----):
double complex::*wskdbl;
wskdbl = &complex::re;
```

Pełny kod programu:

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

using namespace std;

namespace cmp {      // (---1---)
	class complex{
	public:
		double re, im;
		complex(double r = 0., double i = 0.): re(r), im(i) {}
		// (---2---)->
		complex operator+(complex);
		operator int();
		// (---2---)<-
	};
}

cmp::complex cmp::complex::operator+(complex c){      // (---3---)
	complex ret;
	ret.re = re + c.re;
	ret.im = im + c.im;
	return ret;
}

// definicja operatora konwersji complex ---> int
// (---4---)->
cmp::complex::operator int(){     // (---4---)
	return static_cast<int>(re);
}
// (---4---)<-

class punkt{
public:
	double x, y;
	punkt(double xx = 0., double yy = 0.): x(xx), y(yy){}
};

class figura{
public:
	string opis;
	figura(string o = ""): opis(o) {}
	virtual double pole(){return -1;};       // (---5---)
};

class kolo : public figura{
public:
	punkt sr;
	double prom;
	kolo(string = "", punkt = punkt(), double = 1.);
	double pole();
};

// (---6---)->
// definicje metod klasy kolo
kolo::kolo(string o, punkt s, double p):
      figura(o), sr(s), prom(p){}

double kolo::pole(){
	return 4.*atan(1.)*prom*prom;
```

```
}
// (---6---)<-

// (---7---)->
// przeciążenia metody operator<<
ostream& operator<<(ostream& strum, cmp::complex c){
	strum << "(" << c.re << ", " << c.im << ")";
	return strum;
}

ostream& operator<<(ostream& strum, punkt p){
	strum << "(" << p.x << ", " << p.y << ")";
	return strum;
}
// (---7---)<-

ostream& operator<<(ostream& strum, kolo k){
	strum << k.opis << "[" << k.sr << ", " << k.prom << "]";
	return strum;
}

double pole(figura & fig){
   return fig.pole();
}

// (---8---) ->
template<typename Typ, int i> class MojaTab{
public:
   Typ Tablica[i];
   Typ& operator[](int index){return Tablica[index];}
};
// (---8---) <-

int main(){
	using namespace cmp;// (---9---)
	complex c1(1., 3.), c2(5., -1.);

	// (---10---)->
	double complex::*wskdbl;
	wskdbl = &complex::re;
	// (---10---)<-
	cout << "Czesc rzeczywista sumy liczb c1 i c2 = " << (c1+c2).*wskdbl << endl;
	int i = c1;
	cout << "Liczba zespolona c1 = " << c1 << endl;
	cout << "Czesc rzeczywista liczby c1 po konwerscji do int = " << i << endl;

	kolo k("Kolo k", punkt(2., 3.), 1.);
	cout << k << endl;
	cout << "Pole kola k = " << pole(k) << endl;

	MojaTab<complex, 3> MojeZespolone; // (---11---)
	for(int i = 0; i < 3; i++) MojeZespolone[i] =
		complex(i, -i);
	for(int i = 0; i < 3; i++){
		cout << "MojeZespolone[" << i << "]= " << MojeZespolone[i] << endl;
	}

	system("PAUSE");
	return 0;
}
```