VIRTUAL CONSTRUCTORS – DESTRUCTORS AND INHERITANCE

```
#ifndef A_H_
#define A_H_

class A
{
protected:
  int deger;

public:
      A();
      virtual void f()=0;
      void g();
      virtual ~A();
};

#endif /*A_H_*/

#include "A.h"
#include <iostream>
using namespace std;

A::A()
{
  cout<<"kurucu a"<<endl;
  deger=2;
  //f();
  g();
}

void A::g()
{
  cout<<"ag"<<deger<<endl;
}
A::~A()
{
}



#ifndef B_H_
#define B_H_

#include "A.h"

class B : public A
{
protected:
  int deger;
public:
      B();
      B(const B &);
      void f();
      void g();
      virtual ~B();
};

#endif /*B_H_*/
```

```
#include "B.h"
#include <iostream>
using namespace std;

B::B()
{
  cout<<"kurucu b"<<endl;
  f();
  deger=1;
}

B::B(const B& cb)
{
  cout<<"kopya kurucu b"<<endl;
  f();
  deger=5;
}

void B::f()
{
  cout<<"bf"<<deger<<endl;
}

void B::g()
{
  cout<<"bg"<<endl;
}

B::~B()
{
}
```

```
#ifndef C_H_
#define C_H_

#include "A.h"

class C : public A
{
protected:
  int deger;
public:
      C();
      void f();
      virtual ~C();
};

#endif /*C_H_*/
```

```
#include "C.h"
#include <iostream>
using namespace std;

C::C()
{
  deger=3;
  cout<<"kurucu c"<<endl;
}
```

```
void C::f()
{
  cout<<"cf"<<deger<<endl;
}

C::~C()
{
}

#ifndef D_H_
#define D_H_
#include "C.h"
#include "A.h"

class D : public C
{
public:
      D(A*);
      virtual ~D();
};

#endif /*D_H_*/


#include "D.h"
#include <iostream>
using namespace std;

D::D(A* o_b)
{
  cout<<"kurucu d"<<A::deger<<endl;
}

D::~D()
{
}

#include<iostream>
using namespace std;
#include "A.h"
#include "B.h"
#include "C.h"
#include "D.h"

int main()
{
  //A  n_a;
  //A* nr_a=new A();

  //B n_b=new B();
  //B* nr_b=new B();

  //C  n_c;
  //C* nr_c=new C();

  //A* n_b=new B();
  //D  n_d(n_b);
  //D* nr_d=new D(n_b);
  //D* nr_d=new D(nr_b);
```

```
    return 0;
}
```

POLYMORPHISM IN USE : STRATEGY DESIGN PATTERN

```
#ifndef SIRALA_H_
#define SIRALA_H_

class Sirala {
  private:
   int m_min, m_max, m_median;
   void sort( int[], int);
public:
   void vektorOku( int[], int);
   int getMin()    { return m_min; }
   int getMax()    { return m_max; }
   int getMedian() { return m_median; }

};

#endif /*SIRALA_H_*/
```

```
#include "Sirala.h"
#include<iostream>
using namespace std;

void Sirala::vektorOku( int v[], int n ) {
   sort( v, n );
   m_min = v[0];
   m_max = v[n-1];
   m_median = v[n/2];
}

void Sirala::sort( int v[], int n )
{
   for (int i=n-1; i > 0; --i)
      for (int j=0; j < i; ++j)
         if (v[j] > v[j+1]) {
            int t = v[j];
            v[j] = v[j+1];
            v[j+1] = t;
         }

   cout << "Bubble: ";

   for (int k=0; k < n; ++k)
      cout << v[k] << ' ';

   cout << '\n';
}
```

```
#ifndef SIRALAALG_H_
#define SIRALAALG_H_

class SiralaAlg
```

```
{
public:
   virtual void sort( int[], int ) = 0;
};

#endif /*SIRALAALG_H_*/

#ifndef BUBBLESIRALA_H_
#define BUBBLESIRALA_H_

#include "SiralaAlg.h"

class BubbleSirala : public SiralaAlg
{
public:
      void sort(int[],int);
};

#endif /*BUBBLESIRALA_H_*/


#include "BubbleSirala.h"
#include<iostream>
using namespace std;

void BubbleSirala::sort(int v[], int n)
{

     for (int i=n-1; i > 0; --i)
        for (int j=0; j < i; ++j)
           if (v[j] > v[j+1]) {
              int t = v[j];
              v[j] = v[j+1];
              v[j+1] = t;
           }
     cout << "Bubble: ";
     for (int k=0; k < n; k++)
        cout << v[k] << ' ';
     cout << '\n';

}


#ifndef SHELLSIRALA_H_
#define SHELLSIRALA_H_

#include "SiralaAlg.h"

class ShellSirala : public SiralaAlg
{
public:
  void sort(int[],int);
};

#endif /*SHELLSIRALA_H_*/


#include "ShellSirala.h"
#include<iostream>
using namespace std;
```

```cpp
void ShellSirala::sort(int v[], int n)
{

  for (int g = n/2; g > 0; g /= 2)
     for (int i = g; i < n; ++i)
        for (int j = i-g; j >= 0; j -= g)
           if (v[j] > v[j+g]) {
              int temp = v[j];
              v[j] = v[j+g];
              v[j+g] = temp;
           }
  cout << "Shell:  ";
  for (int k=0; k < n; k++)
     cout << v[k] << ' ';
  cout << '\n';

}


#ifndef SIRALASTRAT_H_
#define SIRALASTRAT_H_
#include "SiralaAlg.h"

class SiralaStrat {
  private:
   int m_min, m_max, m_median;
   SiralaAlg*  siralayici;
public:
   SiralaStrat();
   void vektorOku( int[], int);
   int getMin()    { return m_min; }
   int getMax()    { return m_max; }
   int getMedian() { return m_median; }

};

#endif /*SIRALA_H_*/

#include "SiralaStrat.h"
#include "BubbleSirala.h"
#include "ShellSirala.h"
#include<iostream>
using namespace std;

SiralaStrat::SiralaStrat()
{
  int karar;

  cout<<"Neyle Siralansin? ";
  cin>>karar;

  if(karar==0)
    siralayici=new BubbleSirala();
  else if(karar==1)
    siralayici=new ShellSirala();

}

void SiralaStrat::vektorOku( int v[], int n ) {
   siralayici->sort( v, n );
   m_min = v[0];
   m_max = v[n-1];
   m_median = v[n/2];
}

#include<iostream>
using namespace std;
#include "Sirala.h"
#include "SiralaStrat.h"

int main()
{
   const int NUM = 9;
   int dizi[NUM];
   srand( time(0) );
   cout << "Dizi: ";
   for (int i=0; i < NUM; ++i) {
     dizi[i] = rand() % 9 + 1;
      cout << dizi[i] << ' ';
   }
   cout << '\n';

   SiralaStrat siraci;
   siraci.vektorOku( dizi, NUM );
   cout << "min is " << siraci.getMin() << ", max is " << siraci.getMax()
        << ", median is " << siraci.getMedian() << '\n';
}
```

MOVING BEYOND POLYMORPHISM : VISITOR DESIGN PATTERN

```
#ifndef RENK_H_
#define RENK_H_

class Renk
{
public:
   virtual void say() = 0;
   virtual void cagir()  = 0;
   static void sayi_al() {
      cout << "Siyahlar " << sayi_siyah
           << ", Beyazlar " << sayi_beyaz << '\n';
   }
protected:
   static int sayi_siyah, sayi_beyaz;
};

int Renk::sayi_siyah = 0;
int Renk::sayi_beyaz = 0;

#endif /*RENK_H_*/


#ifndef SIYAH_H_
#define SIYAH_H_

#include "Renk.h"

class Siyah : public Renk
{
```

```
public:
  void say() { ++sayi_siyah; }
  void cagir() { kapali(); }
  void kapali() { cout << "Siyah\n"; }
};

#endif /*SIYAH_H_*/

#ifndef BEYAZ_H_
#define BEYAZ_H_

#include "Renk.h"

class Beyaz : public Renk
{
public:
  void say() { ++sayi_beyaz; }
  void cagir() { acik(); }
  void acik() { cout << "Beyaz\n"; }
};

#endif /*BEYAZ_H_*/


#ifndef RENKV_H_
#define RENKV_H_

class RenkV
{
public:
  virtual void cagir( class Visitor* ) = 0;
};

#endif /*RENKV_H_*/


#ifndef SIYAH_H_
#define SIYAH_H_

#include "Renk.h"

class Siyah : public Renk
{
public:
  void say() { ++sayi_siyah; }
  void cagir() { kapali(); }
  void kapali() { cout << "Siyah\n"; }
};

#endif /*SIYAH_H_*/


#ifndef BEYAZV_H_
#define BEYAZV_H_

#include "RenkV.h"
#include <iostream>
using namespace std;

class BeyazV : public RenkV
```

```cpp
{
public:
  void cagir( Visitor* );
  void acik() { cout << "Beyaz\n"; }
};

#endif /*BEYAZV_H_*/


#ifndef VISITOR_H_
#define VISITOR_H_

#include "SiyahV.h"
#include "BeyazV.h"

class Visitor
{
public:
  virtual void visit( SiyahV* ) = 0;
  virtual void visit( BeyazV* ) = 0;
};

#endif /*VISITOR_H_*/


#ifndef CAGIRVISITOR_H_
#define CAGIRVISITOR_H_

#include "Visitor.h"

class CagirVisitor : public Visitor
{
public:
  /*virtual*/ void visit( SiyahV* s ) { s->kapali(); }
  /*virtual*/ void visit( BeyazV* b ) { b->acik(); }
};

#endif /*CAGIRVISITOR_H_*/


#ifndef SAYIVISITOR_H_
#define SAYIVISITOR_H_

#include "Visitor.h"

class SayiVisitor : public Visitor
{
private:
  int  sayi_siyah, sayi_beyaz;
public:
  SayiVisitor() { sayi_siyah = sayi_beyaz = 0; }
  /*virtual*/ void visit( SiyahV* ) { ++sayi_siyah; }
  /*virtual*/ void visit( BeyazV* ) { ++sayi_beyaz; }
  void sayi_al() {
     cout << "Siyahlar " << sayi_siyah
          << ", Beyazlar " << sayi_beyaz << '\n';
  }

};

#endif /*SAYIVISITOR_H_*/
```

```
#include<iostream>
using namespace std;
#include "Renk.h"
#include "Siyah.h"
#include "Beyaz.h"
#include "RenkV.h"
#include "SiyahV.h"
#include "BeyazV.h"
#include "SayiVisitor.h"
#include "CagirVisitor.h"

void SiyahV::cagir( Visitor* v ) { v->visit( this ); }
void BeyazV::cagir( Visitor* v ) { v->visit( this ); }

int main()
{

    Renk* carsi[] = { new Siyah, new Beyaz, new Siyah,
                     new Beyaz, new Siyah, 0 };
    for (int i=0; carsi[i]; ++i) {
       carsi[i]->say();
       carsi[i]->cagir();
    }
    Renk::sayi_al();


   RenkV* carsi[] = { new SiyahV, new BeyazV, new SiyahV, new BeyazV, new
SiyahV, 0 };

   SayiVisitor sayan;
   CagirVisitor cagiran;

   for (int i=0; carsi[i]; ++i) {
     carsi[i]->cagir(&sayan);
     carsi[i]->cagir(&cagiran);
   }

   Renk::sayi_al();

}
```