# Interface Segregation Principle

**Clients should not be forced to dependent on methods that they don’t need.**

**An interface should only contain methods that are relevant to the clients that use it and they should not be forced to implement methods that they don’t need.**

Lets take an example to understand the above with a **Printer System Example**.

We have to implement different types of printers like **InkJetPrinter,LaserPrinter,LEDPrinter,3Dprinter.**

We can have a abstract class for Printer which will implement the methods like Print, Scan and Fax.

```
Class Printer
{
  public:
       virtual void print()=0;
       virtual void scan()=0;
        virtual void fax()=0;
};

class InkJetPrinter : public Printer
{
   public:
         void print() {
               cout<<”Inkjet Printer printer”<<endl;
           }
           void scan() {
                cout<<”Inkjet Printer scanner”<<endl;
             }
            void fax() {
                cout<<”Inkjet Printer fax”<<endl;
             }
};
class LaserPrinter : public Printer
{
   public:
        void print(){
                 cout<<” LaserPrinter printer”<<endl;
           }
           void scan() {
                cout<<” LaserPrinter scanner”<<endl;
             }
            void fax() {
                throw runtime_error(“Laser printer cant fax”)<<end; }
};
```

The *LaserPrinter* class does not need the fax() method, but it is forced to implement it because it has to adhere to the Printer interface(abstract class).

This violates the Interface Segregation Principle as *LaserPrinter* class is forced to implement the fax even it does not support the functionality of Faxing.

## Solutions to above

To fix this violation, we can create the separate interfaces for *print*(), *scan*() and *fax*() methods.

```
Class Iprintable {
      public:
           virtual void print() =0;
};
class Iscanble {
 public:
      virtual void scan()=0;
};
class Ifaxable {
 public:
     virtual void fax()=0;
};

class InkjetPrinter : public Iprintable, public Iscanable,Ifaxable
{
     public:
             void print() {
                 cout<<”Inkjet Printer print”<<endl;
            }
             void scan() {
                 cout<<”Inkjet Printer scan”<<endl;
            }
             void fax() {
                cout<<”Inkjet Printer fax”<<endl;
            }
};
class LaserPrinter : public Iprintable, public Iscanable
{
     public:
        void print() {
             cout<<”LaserPrinter print”<<endl;
         }
         void scan(){
           cout<<”LaserPrinter scan”<<endl;
        }
 };
```

Now, the *LaserPrinter* has to only implement the methods that are relevant to its functionality. Now this adheres to Interface Segregation Principle as classes are not forced to implement unnecessary methods.