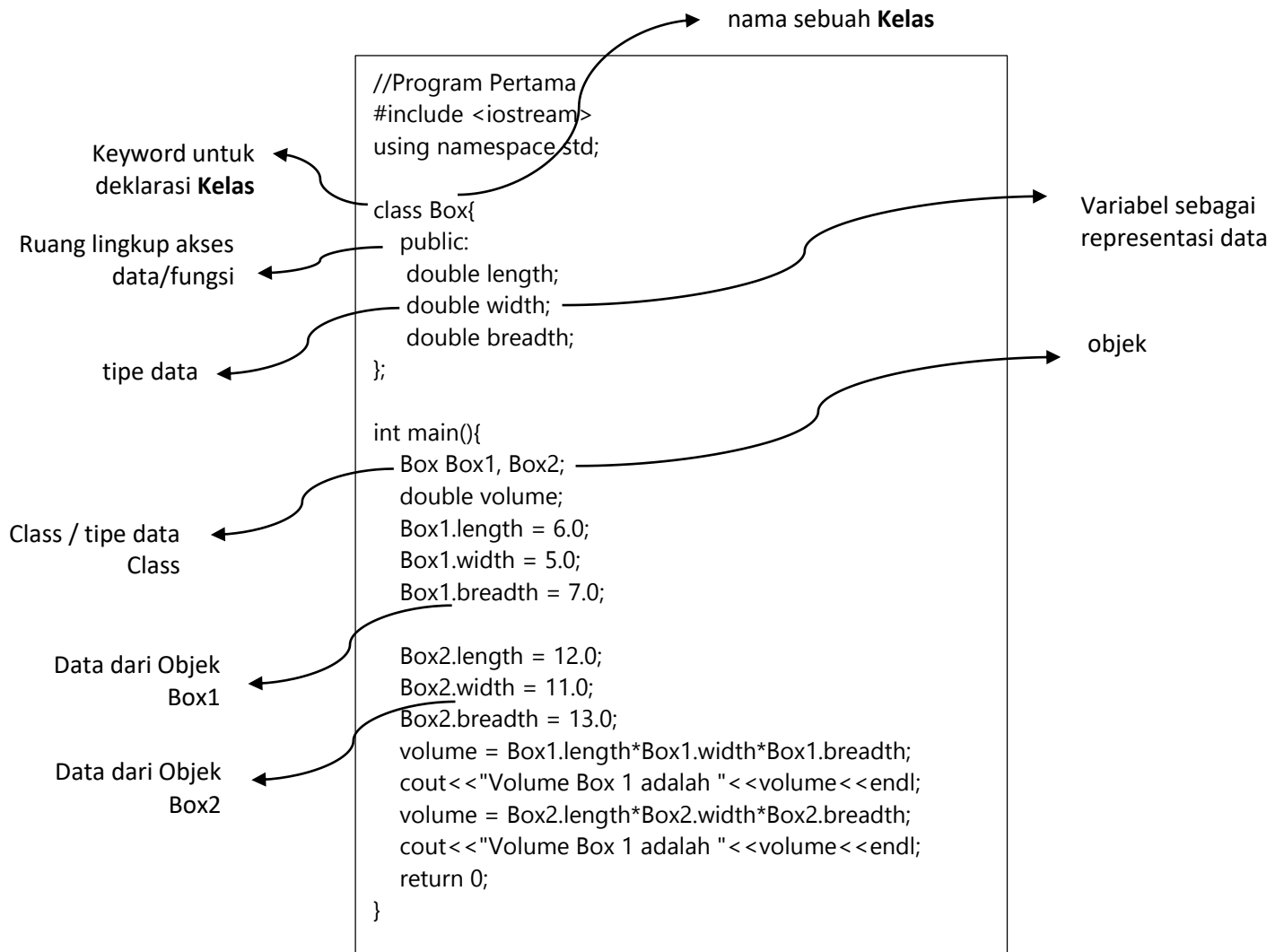


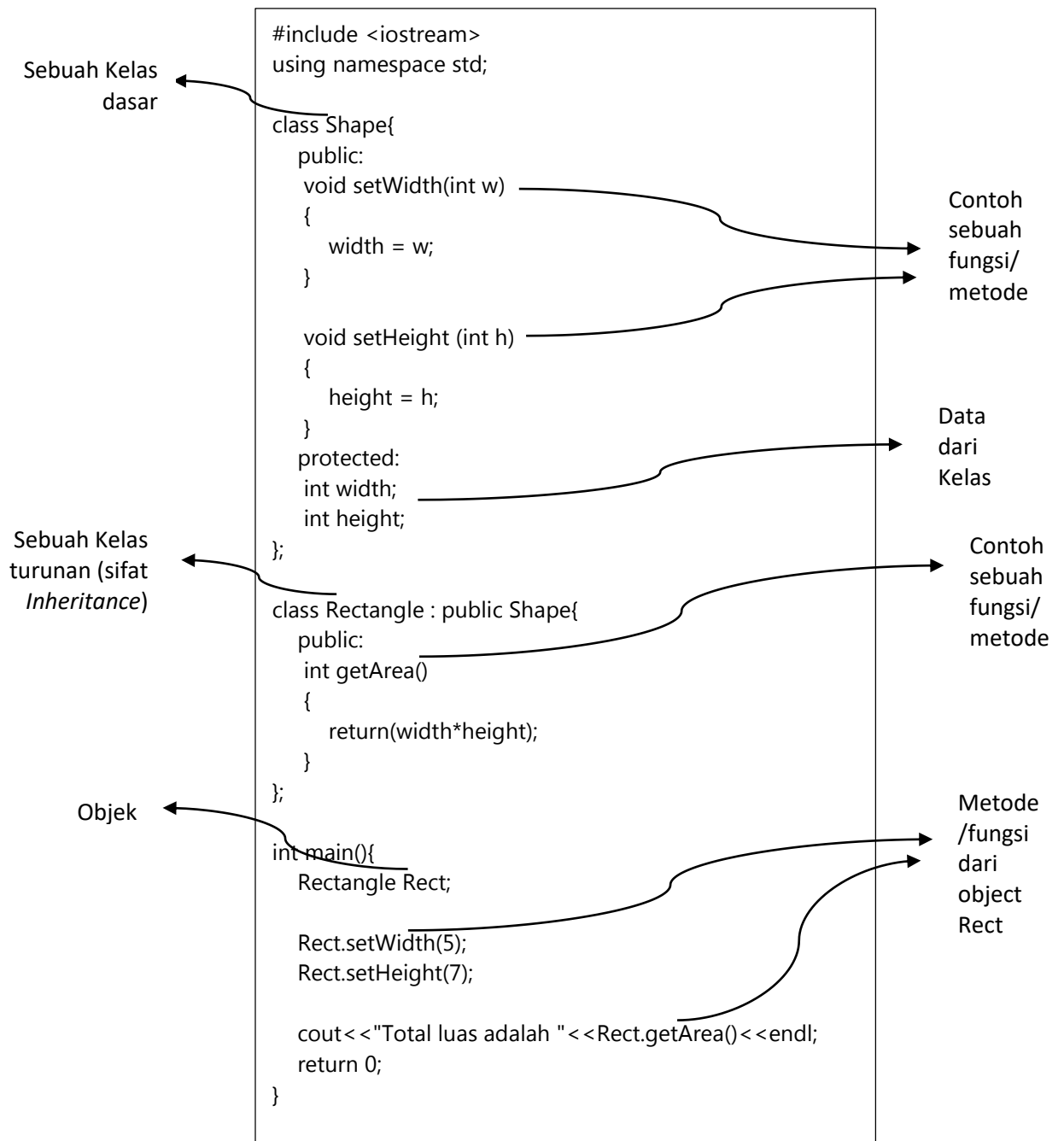
KELAS (CLASS) dan OBJEK

Kelas digunakan untuk menspesifikasikan bentuk suatu objek yang mengkombinasikan representasi data dengan metode (fungsi) untuk memanipulasi data tersebut dalam satu paket. Data dan fungsi dalam suatu kelas disebut anggota kelas. Kelas didefinisikan dengan keyword **class** dan diikuti nama kelas.

Perhatikan program berikut* :



Program kedua* :



INHERITANCE (Pewarisan)

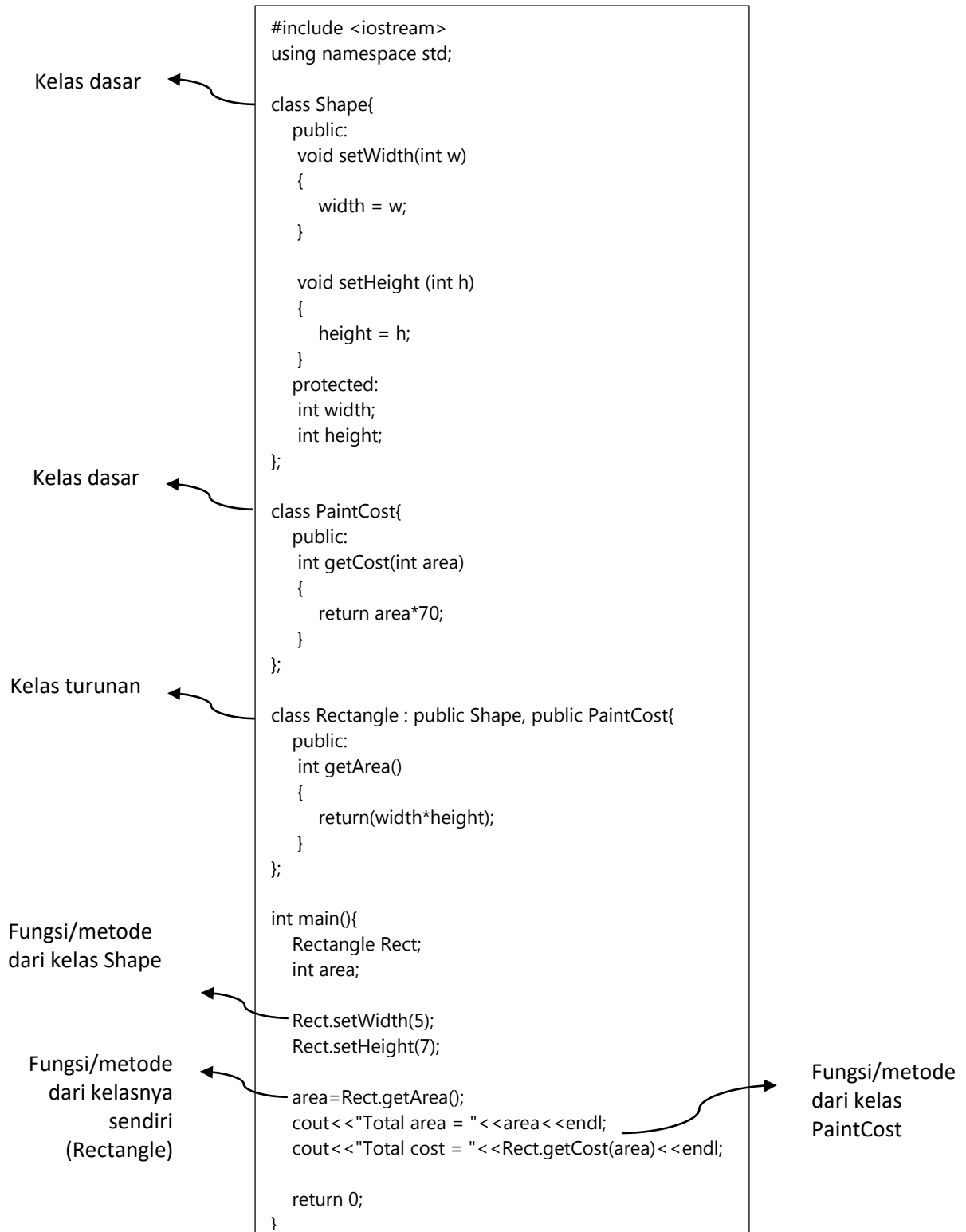
Pewarisan adalah salah satu konsep dalam paradigma Pemrograman Berorientasi Objek yang intinya suatu kelas dasar (dalam contoh program kedua, kelas dasar adalah Shape) dapat diturunkan menjadi suatu kelas yang lain atau kelas turunan (*derived class*) dengan sifat yang sama dengan kelas dasarnya (dalam contoh program kedua, kelas turunannya adalah Rectangle).

Jika suatu objek dideklarasikan dengan tipe data kelas turunan (contoh : Rectangle Rect) maka objek tersebut akan memiliki data dan fungsi yang dimiliki oleh kelas dasar maupun kelas turunannya dengan tetap memperhatikan ruang lingkup akses (public/protected/private) yang menyertai data serta fungsi tersebut.

Adapun arti masing-masing ruang lingkup akses semacam public, private dan protected adalah sebagai berikut :

- a. Private adalah default ruang lingkup data dan fungsi dalam sebuah kelas. Jika data dan fungsi dalam kelas dideklarasikan dengan tidak diawali public ataupun protected maka data dan fungsi tersebut bersifat private. Jika data atau fungsi dideklarasikan dengan private maka data dan fungsi hanya dapat diakses dalam kelas dimana data dan fungsi tersebut dideklarasikan.
- b. Ruang lingkup protected berarti suatu data dan fungsi dapat diakses oleh kelas dasar dan kelas turunannya, tetapi tidak untuk kelas diluar kelas tersebut.
- c. Ruang lingkup public berarti suatu data dan fungsi dapat diakses secara umum oleh kelas apapun.

Perhatikan contoh program ketiga berikut ini* :



OVERLOADING

Overloading adalah kemampuan program mengeksekusi suatu fungsi memiliki nama sama tetapi memiliki parameter dengan tipe data berbeda.

Pada contoh keempat berikut ini tampak bahwa terdapat sebuah kelas printData dengan sebuah objek, **pd**. Kelas printData memiliki 2 metode/fungsi yang bernama sama yaitu print dengan parameter yang berbeda. Pertama adalah int i dan yang kedua adalah double f.

Karena kedua fungsi tersebut bersifat public, maka objek pd dapat mengakses metode/fungsi print. Pada statement pd.print(5) maka objek **pd** akan mengakses metode dengan parameter int i. Sedangkan pada statement pd.print(500.23) maka objek **pd** akan mengakses metode dengan parameter double f.

Kejadian semacam ini disebut overloading.

Perhatikan contoh program keempat berikut ini:

```
//OVERLOADING
#include <iostream>
using namespace std;

class printData{
public:
    void print(int i)
    {
        cout<<"Printing int:"<<i<<endl;
    }

    void print(double f)
    {
        cout<<"Printing float :"<<f<<endl;
    }
};

int main(){
    printData pd;
    pd.print(5);
    pd.print(500.23);
    return 0;
}
```

Program LinkedList dengan Class (LinkedList_Class)

Setelah kita mengenal Class, sekarang saatnya kita gunakan class dalam mempelajari LINKED LIST. Berikut adalah suatu aplikasi linked list yang dilakukan dengan menggunakan class. Silahkan coba menggunakan aplikasi CodeBlock yang Saudara miliki.

```
#include<iostream>
using namespace std;
struct node
{
    int data;
    node *next;
};
class list
{
private:
    node *head, *tail;
public:
    list()
    {
        head=NULL;
        tail=NULL;
    }
    void createnode(int value)
    {
        node *temp=new node;
        temp->data=value;
        temp->next=NULL;
        if(head==NULL)
        {
            head=temp;
            tail=temp;
            temp=NULL;
        }
        else
        {
            tail->next=temp;
            tail=temp;
        }
    }
    void display()
    {
        node *temp=new node;
        temp=head;
        while(temp!=NULL)
        {
            cout<<temp->data<<"\t";
            temp=temp->next;
        }
    }
};
```

```

    }
}
void insert_start(int value)
{
    node *temp=new node;
    temp->data=value;
    temp->next=head;
    head=temp;
}
void insert_position(int pos, int value)
{
    node *pre=new node;
    node *cur=new node;
    node *temp=new node;
    cur=head;
    for(int i=1;i<pos;i++)
    {
        pre=cur;
        cur=cur->next;
    }
    temp->data=value;
    pre->next=temp;
    temp->next=cur;
}
void delete_first()
{
    node *temp=new node;
    temp=head;
    head=head->next;
    delete temp;
}
void delete_last()
{
    node *current=new node;
    node *previous=new node;
    current=head;
    while(current->next!=NULL)
    {
        previous=current;
        current=current->next;
    }
    tail=previous;
    previous->next=NULL;
    delete current;
}
void delete_position(int pos)
{
    node *current=new node;

```

```

        node *previous=new node;
        current=head;
        for(int i=1;i<pos;i++)
        {
            previous=current;
            current=current->next;
        }
        previous->next=current->next;
    }
};

int main()
{
    list obj;
    obj.createnode(25);
    obj.createnode(50);
    obj.createnode(90);
    obj.createnode(40);
    cout<<"\n-----\n";
    cout<<"-----Displaying All nodes-----";
    cout<<"\n-----\n";
    obj.display();
    cout<<"\n-----\n";
    cout<<"-----Inserting At End-----";
    cout<<"\n-----\n";
    obj.createnode(55);
    obj.display();
    cout<<"\n-----\n";
    cout<<"-----Inserting At Start-----";
    cout<<"\n-----\n";
    obj.insert_start(50);
    obj.display();
    cout<<"\n-----\n";
    cout<<"-----Inserting At Particular-----";
    cout<<"\n-----\n";
    obj.insert_position(5,60);
    obj.display();
    cout<<"\n-----\n";
    cout<<"-----Deleting At Start-----";
    cout<<"\n-----\n";
    obj.delete_first();
    obj.display();
    cout<<"\n-----\n";
    cout<<"-----Deleing At End-----";
    cout<<"\n-----\n";
    obj.delete_last();
    obj.display();
    cout<<"\n-----\n";
    cout<<"-----Deleting At Particular-----";

```



```
    cout<<"\n-----\n";  
    obj.delete_position(4);  
    obj.display();  
    cout<<"\n-----\n";  
    system("pause");  
    return 0;  
}
```