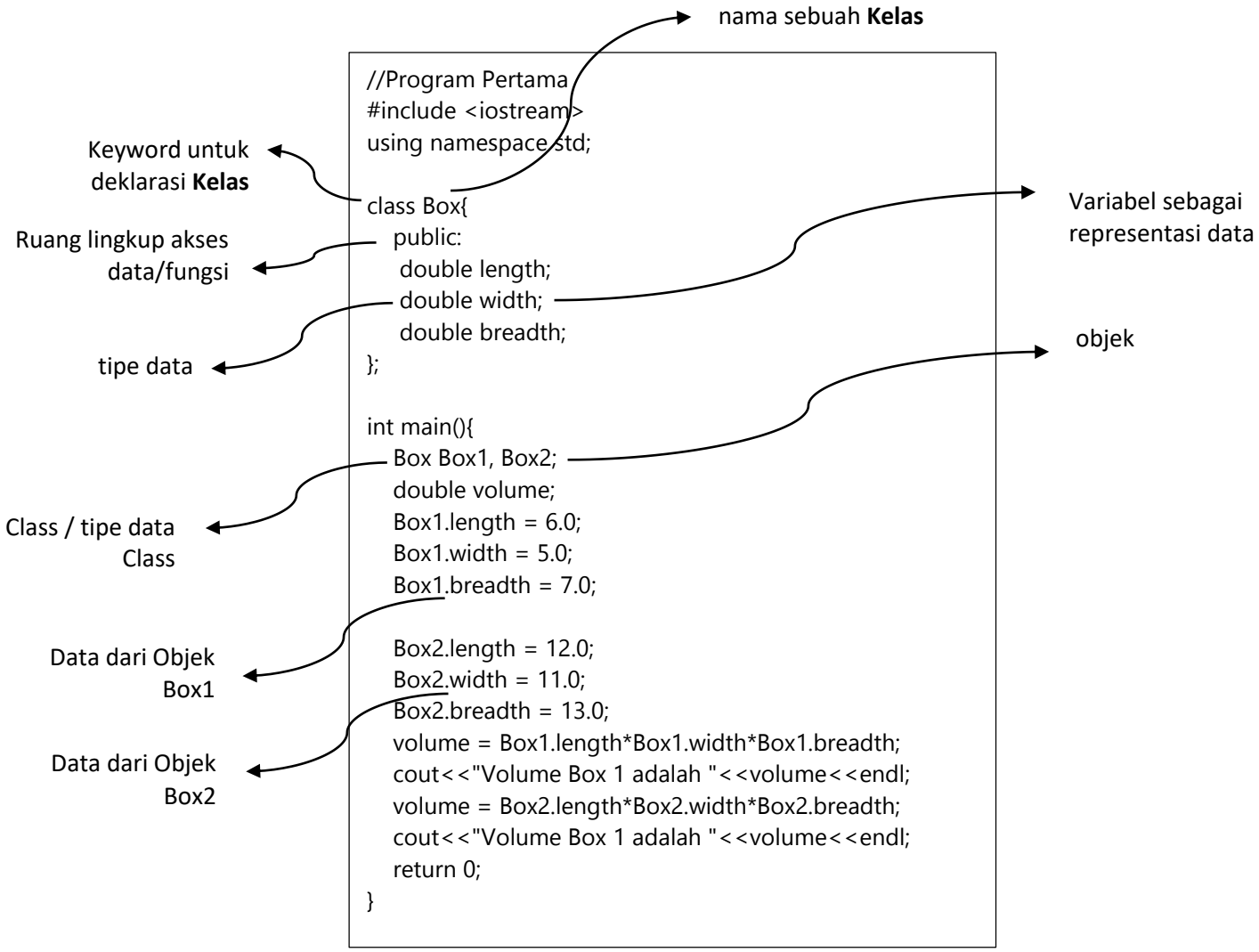
MATERI
JUMAT 10/5/2019

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 :



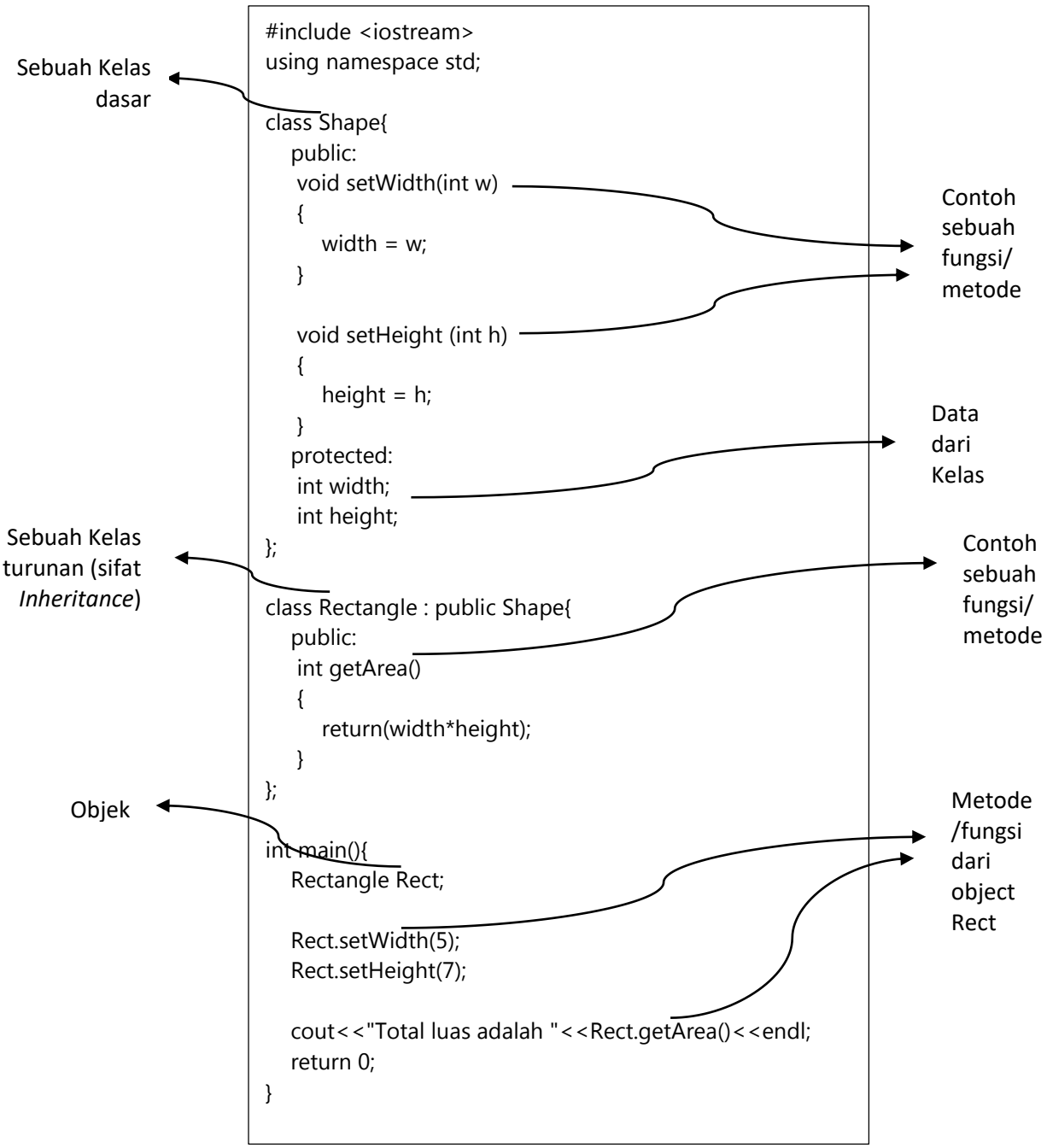

```
//Program Pertama
#include <iostream>
using namespace std;

class Box{
   public:
    double length;
    double width;
    double breadth;
};

int main(){
   Box Box1, Box2;
   double volume;
   Box1.length = 6.0;
   Box1.width = 5.0;
   Box1.breadth = 7.0;

   Box2.length = 12.0;
   Box2.width = 11.0;
   Box2.breadth = 13.0;
   volume = Box1.length*Box1.width*Box1.breadth;
   cout<<"Volume Box 1 adalah "<<volume<<endl;
   volume = Box2.length*Box2.width*Box2.breadth;
   cout<<"Volume Box 1 adalah "<<volume<<endl;
   return 0;
}
```

Program kedua




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

class Shape{
   public:
    void setWidth(int w)
    {
       width = w;
    }

    void setHeight (int h)
    {
       height = h;
    }
   protected:
    int width;
    int height;
};

class Rectangle : public Shape{
   public:
    int getArea()
    {
       return(width*height);
    }
};

int main(){
   Rectangle Rect;

   Rect.setWidth(5);
   Rect.setHeight(7);

   cout<<"Total luas adalah "<<Rect.getArea()<<endl;
   return 0;
}
```

INHERITANCE (Pewarisan)

Pewarisan adalah salah satu konsep dalam paradigm Pemrograman Berorientasi Objek yang intinya suatu kelas dasar (dalam contoh program kedua adalah Shape) dapat diturunkan menjadi suatu kelas yang lain atau kelas turunan dengan sifat yang sama dengan kelas dasarnya (dalam contoh program kedua 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 baik kelas dasar maupun kelas turunan dengan tetap memperhatikan penyematan ruang lingkup akses yang menyertai data serta fungsi yang ada.

Adapun ragam ruang akses adalah public, private dan protected. Private adalah default ruang lingkup data dan fungsi dalam sebuah kelas. Maka jika data dan fungsi dalam kelas dalam deklarasinya tidak diawali dengan public ataupub protected maka data dan fungsi tersebut bersifat private.

Ruang lingkup private berarti suatu data dan fungsi hanya dapat diakses dalam kelas dimana data dan fungsi tersebut dideklarasikan.
Ruang lingkup protected berarti suatu data dan fungsi dapat diakses oleh kelas dasar dan kelas turunannya, tetapi tidak untuk kelas diluar kelas tersebut.
Ruang lingkup public berarti suatu data dan fungsi dapat diakses secara umum oleh kelas apapun.

Perhatikan contoh program ketiga berikut ini:

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

class Shape{
   public:
    void setWidth(int w)
    {
       width = w;
    }

    void setHeight (int h)
    {
       height = h;
    }
   protected:
    int width;
    int height;
};


class PaintCost{
   public:
    int getCost(int area)
    {
       return area*70;
    }
};

class Rectangle : public Shape, public PaintCost{
   public:
    int getArea()
    {
       return(width*height);
    }
};

int main(){
   Rectangle Rect;
   int area;

   Rect.setWidth(5);
   Rect.setHeight(7);

   area=Rect.getArea();
   cout<<"Total area = "<<area<<endl;
   cout<<"Total cost = "<<Rect.getCost(area)<<endl;

   return 0;
}
```

OVERLOADDING

Overloading adalah kemampuan program mengeksekusi suatu fungsi yang sama tetapi dengan 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:

```
//OVERLOADDING
#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;
}
```

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

Program LinkedList dengan Class (LinkList_Class)

```
#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;
}
```