Object Oriented Paradigm

Object and Class

- What is Object?
  - Any real life entity which has properties and behaviours
  - **Instance of class**
  - Variable of UDT(User defined type)
- What is class?
  - **Blueprint for Object**
  - A programming construct which defines objects properties and behaviour. Class define data access policy using public, private and protected keywords.
  - Definition of UDT(User defined type)

Object Oriented Paradigm

- Object oriented Paradigm has Object Oriented Principles
  - **Object Oriented Principles**
    - **Encapsulation**
    - **Inheritance**
    - **Polymorphism**
    - **Cohesion**
    - **Coupling**
  - **Programming practices**
    - Data abstraction
    - Data Handling

Object Oriented Principles

- **Encapsulation:**
  - binding data members with member functions of object.
  - **binding properties with behaviours of object.**

Inheritance:

  - Creation of new object by acquiring common properties and behaviours and extending behaviour of parent object if required.
  - **Generalization to Specialization**

Object Oriented Principles

- **Polymorphism:**
  - many forms of same thing
  - **different behaviour for different caller**
- **Coupling:**
  - **Interaction between different objects(Message passing)**
  - Coupling should be low
  - **Cohesion:**
  - **Interaction within object**
  - Cohesion should be high(Self sufficient object)

Programming practices

- **Data Abstraction:**
  - Knowing required details about Object
  - **Never details about how the object performs behaviours**
- **Data Hiding:**
  - Controlling the accessibility of objects data(Properties and behaviours)
  - **Hiding details from outside world**.

Syntax for Class

```
 class <class_name>
{
private:
<Data members>;
<Members functions>;
protected:
<Data members>;
<Members functions>;
public:
<Data members>;
<Members functions>;
}; // Terminating semicolon is extremely required.
```

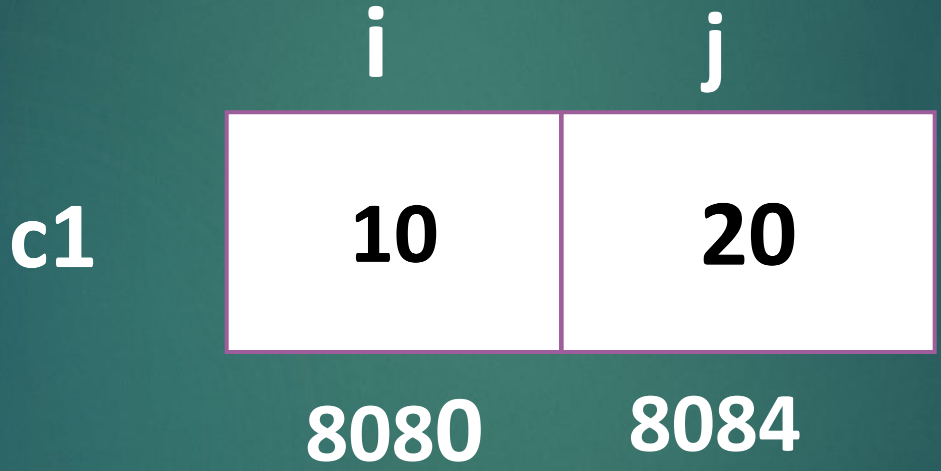
First Example of Class and Object

- Create a class to denote complex numbers and implement behaviors.
- Task List
  - Create class with data members and member functions.
  - Write main function
    - Create Object
    - Call member functions

First Example of Class and Object

```
#include<iostream>
using namespace std;
class Complex{
 private:
 int i;
 int j;
 public:
 void Accept(){
cout<<"Enter real & img part"<<endl;
cin>>i>>j;
 }
void Display(){
cout<<"I="<<i<<"\nJ="<<j;
}
};
int main(){
 Complex c1;
 c1.Accept();
 c1.Display();
 return 0;
}
```

Object graphical representation of Complex class object



i j

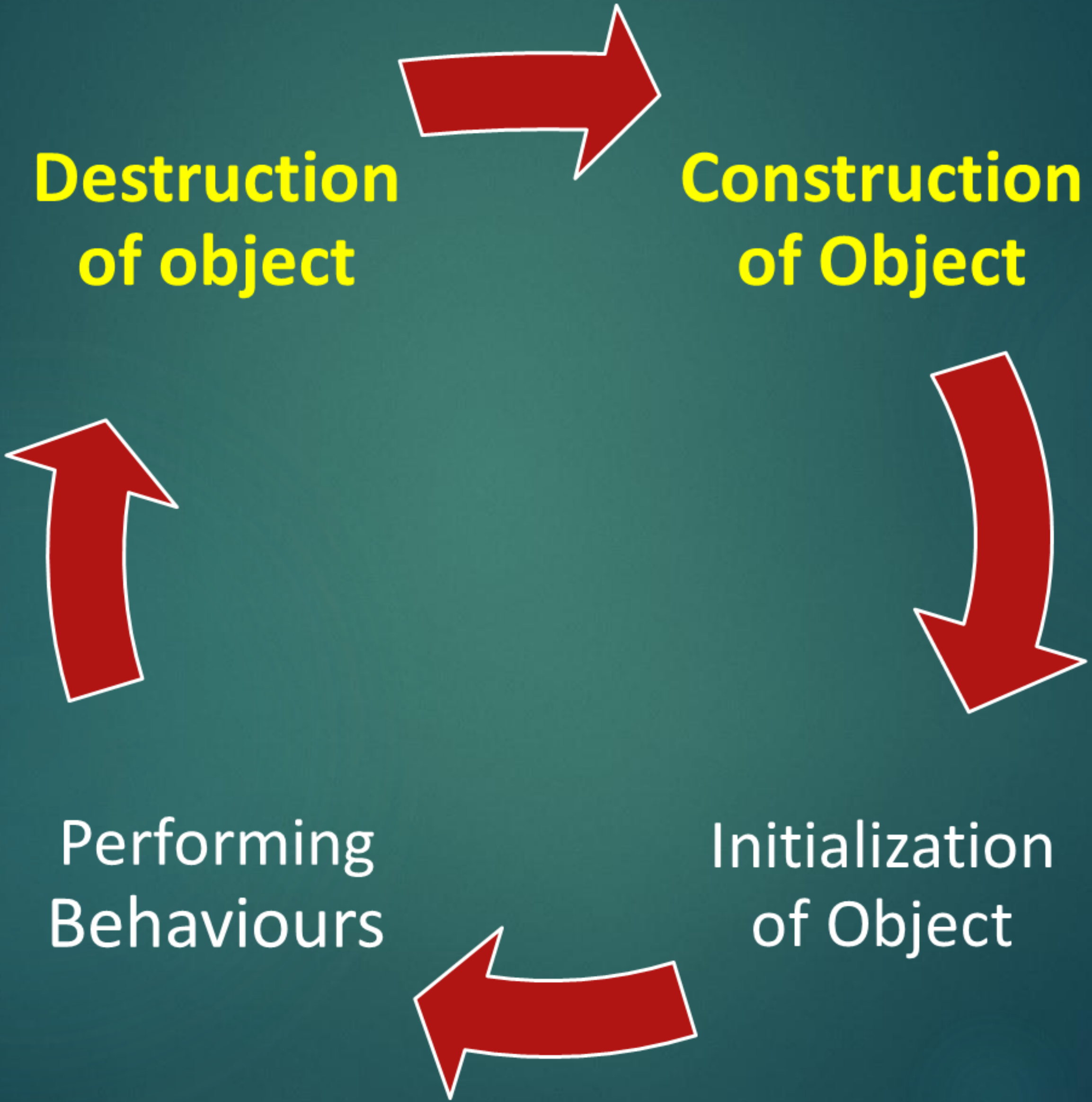
c1 10 20

8080 8084

Properties of class

- Default access specifier inside class is **private**.
- **private** data members can be **accessed within class only**
- **protected** data members can be **accessed within class and inside child classes**
- **public** data members can be **accessed anywhere**
- Size of class /object is sum of sizes of all data members
- **Size of empty class is 1 byte**

Life Cycle of an Object



Destruction of object

Construction of Object

Initialization of Object

Performing Behaviours

Constructor (ctor)

- Constructor is special member function with same name as class and with or without arguments but no return type.
- **Constructor is used to initialize object**
- If programmer do not write ctor for class, **complier automatically inserts default ctor(0 args ctor)**
- Types of Constructors
  - **Default ctor (0 arguments)**
  - **Parameterized ctor / Conversion ctor(with args)**
  - **Copy ctor**

Constructor Examples

```
 #include<iostream>
using namespace std;
class Complex{
 private:
 int i;
 int j;
 public:
 Complex() // Default ctor
{ i=0;
 j=0;     }
Complex(int p, int k)
{ i=p; //Parameterized ctor
 j=k;     }
void Accept(){
cin>>i>>j;   }
void Display(){
cout<<"\n"<<i<<" "<<j; }
};
int main(){
Complex c1; // Default constructor call
Complex c2(10,10); // Parameterized ctor
c1.Display();
c2.Display();
return 0; }
 // If programmer implements any
constructor compiler will not provide default
constructor
```

Constructor with initialization List

- **Constructor with initialization List does pure initialization**
- Constant data members can only be initialized using ctor with initialization list
- **Array data members can not be initialized using ctor with initialization list**

```
Complex() {  // Default ctor
     i=0;
     j=0;    }
Complex() : i(0),j(0){  // Ctor with initialization list
}
```

Constructor with initialization List

```
#include<iostream>
using namespace std;
class Sample{
private:
int arr[3];
public:
// Sample(): arr[0](0)
 // { } // Error at above line
Sample(){   // Array D.M init in ctor
for(int i=0;i<3; i++){
      arr[i] =i;
}}
//Array D.M init is not possible with ctor with init list
```

```
void Display(){
      for(int i=0;i<3; i++){
      arr[i] =i;    }
      }
};
int main()
{    Sample s;
s.Display();
return 0;
}
```

Different ways to create and init object

Assuming Complex class is implemented with Default and parameterized constructor (Refer Slide no.12)

```
int main(){
Complex c1;
Complex c2(12,12);
Complex c3 = c1; // Copy ctor
Complex c4(c2);  //Copy ctor
Complex c5 = Complex();
//Copy ctor
```

```
Complex *cp = new Complex;
Complex *cp2 = new
Complex(5,5);
c2.Display();  //Obj need . (dot)
cp2->Display(); //Ptr  need ->
return 0;}
```

Array of Objects

```
int main(){
Complex carr[3]; // Static array
for(int i=0; i<3; i++){
carr[i].Accept();
}
for(int i=0; i<3; i++){
carr[i].Display();
}
return 0; }
```

```
int main(){
Complex *cp = new Complex[3];
for(int i=0; i<3; i++){
cp[i].Accept();
} //cp[i] is object hence . opr
for(int i=0; i<3; i++){
cp[i].Display();
}
return 0; }
```

Thank You

Think about Objects around you!!!!
……….OOP