OOP

Prabhat Kumar Padhy

Contents

- C++ Namespaces
- Objects and Classes
  - Local Variable and Instance variable
- Class Diagram
  - Class Diagram for Book
  - Class Diagram for Author
  - Class diagram for Pen
- C++
  - Structures
  - Classes
- Java Classes
- Constructors
- Destructors
- Getters
- Setters
- this

C++ – Namespaces

- Namespaces in C++ are used to organize too many classes, for easy handing in big applications
- For accessing the class of a namespace,

we need to use namespacename::classname.

We can use **using** keyword

- In C++, global namespace is the root namespace.
- The global::std will always refer to the namespace

"std" of C++ Framework.

Example

```
#include <iostream>
using namespace std;
namespace First {
    void sayHello() {
        cout<<"Hello First Namespace"<<endl;
    }
}
namespace Second  {
       void sayHello() {
           cout<<"Hello Second Namespace"<<endl;
       }
}
int main()
{
 First::sayHello();
 Second::sayHello();
return 0;
}
```

Example

```
#include <iostream>
using namespace std;
namespace First{
   void sayHello(){
      cout << "Hello First Namespace" <<
endl;
   }
}
namespace Second{
   void sayHello(){
      cout << "Hello Second Namespace"
<< endl;
   }
}
using namespace First;
int main () {
   sayHello();
   return 0;
}
```

Object and Class

- Objects
  - Objects are nothing but real world entities such as pen, chair, table, car, point , pencil, rectangle etc..
  - An entity that has state and behavior is known as object. In Java each object also has an identity which is used by JVM to identify the obhect.
  - It can be physical or logical
  - Object is a runtime entity and also is an instance of a class
- Class
  - It is a classification of real world objects
  - Also classification is bound to have collection of objects
  - It can have fields, methods, constructor, nested class etc.
- Instance Variable
  - A variable which is created inside the class but outside the method known as instance variable. These does not get memory at compile time, rather when object is created.

Class Diagram

- Class diagram

Class Definitions

Various classes

Classname

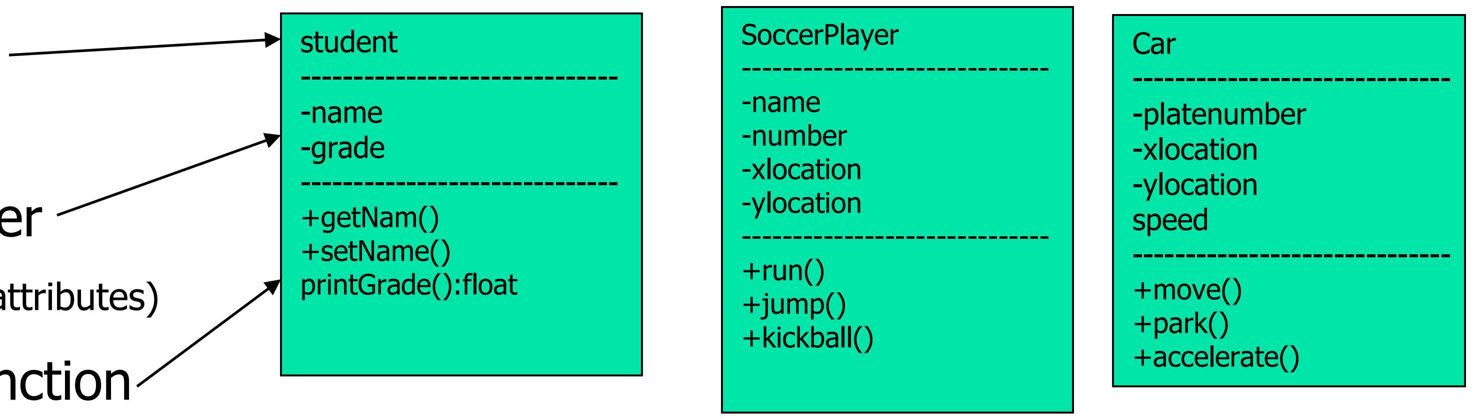
(identifier)

Data Member

(static / private attributes)

Member Function

(static / private attributes)

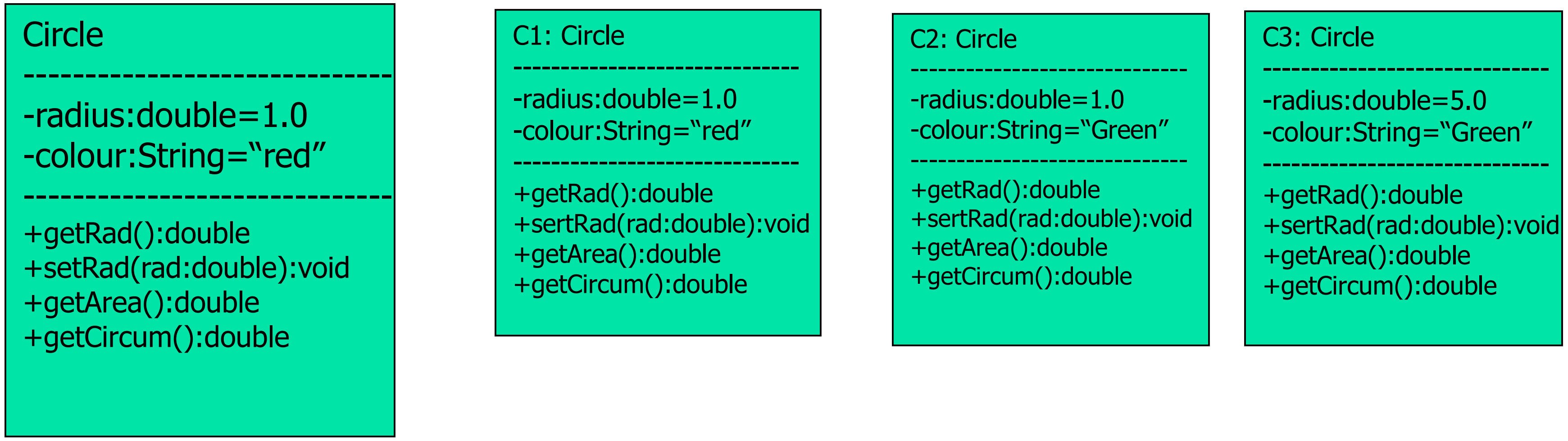


Class Diagram

- Class diagram

Class Definitions

Instances



```
Circle
------------------------------
-radius:double=1.0
-colour:String="red"
------------------------------
+getRad():double
+setRad(rad:double):void
+getArea():double
+getCircum():double
```

```
C1: Circle
------------------------------
-radius:double=1.0
-colour:String="red"
------------------------------
+getRad():double
+sertRad(rad:double):void
+getArea():double
+getCircum():double
```

```
C2: Circle
------------------------------
-radius:double=1.0
-colour:String="Green"
------------------------------
+getRad():double
+sertRad(rad:double):void
+getArea():double
+getCircum():double
```

```
C3: Circle
------------------------------
-radius:double=5.0
-colour:String="Green"
------------------------------
+getRad():double
+sertRad(rad:double):void
+getArea():double
+getCircum():double
```

- A class called Circle is to be defined as illustrated in the class diagram. It contains two data members: radius (of type double) and color (of type String); and three member functions: getRad(),setRad(), and getArea() and getCircum().

- Three instances of Circles called c1, c2, and c3 shall then be constructed with their respective data members, as shown in the instance diagrams.

Object and Class

- Objects
  - Objects are nothing but real world entities such as pen, chair, table, car, point , pencil, rectangle etc..
  - An entity that has state and behavior is known as object. In Java each object also has an identity which is used by JVM to identify the obhect.
  - It can be physical or logical
  - Object is a runtime entity
  - Object is an instance of a class

- Class
  - It is a classification of real world objects
  - Also classification is bound to have collection of objects

Java – classes, objects

- Main method within the class

```
//Java Program to illustrate how to define a
class and fields
//Defining a Student class.
class Student {
 //defining fields
 int id;  //field or data member or instance
variable
 String name;
 //creating main method inside the Student
class
 public static void main(String args[]) {
  //Creating an object or instance
  Student s1 = new Student(); //creating an object
of Student
  //Printing values of the object

System.out.println(s1.id); //accessing member
through reference variable
  System.out.println(s1.name);

 }
}
```

- Main method outside the class

```
//Java Program to demonstrate having the
main method in another class
//Creating Student class.

class Student {
 int id;
 String name;
}

//Creating another class TestStudent1 which
contains the main method

class TestStudent1{
 public static void main(String args[]){
  Student s1 = new Student();
  System.out.println(s1.id);
  System.out.println(s1.name);
 }
}
```

Java – classes, objects(2)

- Initialization of object can be done in 3 ways

  - Main method within the class

```
//Java Program to illustrate how to define a
class and fields
//Defining a Student class.
class Student{
 //defining fields
 int id;//field or data member or instance
variable
 String name;
 //creating main method inside the Student class
 public static void main(String args[]){
  //Creating an object or instance
  Student s1=new Student();//creating an object
of Student
  //Printing values of the object
  System.out.println(s1.id);//accessing member
through reference variable
  System.out.println(s1.name);
 }
}
```

  - Main method outside the class

```
//Java Program to demonstrate having the main
method in
//another class
//Creating Student class.
class Student{
 int id;
 String name;
}
//Creating another class TestStudent1 which contains
the main method
class TestStudent1{
 public static void main(String args[]){
  Student s1=new Student();
  System.out.println(s1.id);
  System.out.println(s1.name);
 }
}
```

Java – classes, objects(2)

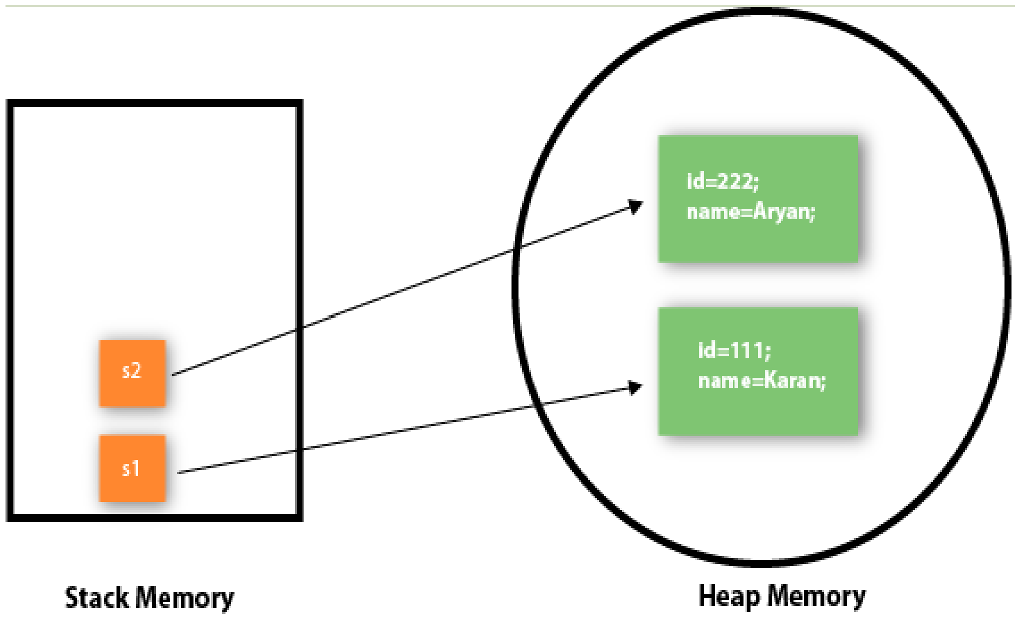
- Objects can be initialized in 3 ways
  - By reference variable
  - By method
  - By Constructor

```
//Initialize through reference

class Student{
 int id;
 String name;
}
class TestStudent2{
 public static void main(String args[]){
  Student s1=new Student();
  s1.id=101;
  s1.name="Sonoo";
  System.out.println(s1.id+" "+s1.name);//printing
members with a white space
 }
}
```

```
//Java Program to show object initialization through
method

class Student{
 int rollno;
 String name;
 void insertRecord(int r, String n){
  rollno=r;
  name=n;
 }
 void displayInformation(){System.out.println(rollno+"
"+name);}
}
class TestStudent4{
 public static void main(String args[]){
  Student s1=new Student();
  Student s2=new Student();
  s1.insertRecord(111,"Karan");
  s2.insertRecord(222,"Aryan");
  s1.displayInformation();
  s2.displayInformation();
 }
}
```

Java – classes, objects(3)

- Memory foot print of an object



```
//Java Program to show object initialization through
method

class Student{
 int rollno;
 String name;
 void insertRecord(int r, String n){
  rollno=r;
  name=n;
 }
 void displayInformation(){System.out.println(rollno+"
"+name);}
}
class TestStudent4{
 public static void main(String args[]){
  Student s1=new Student();
  Student s2=new Student();
  s1.insertRecord(111,"Karan");
  s2.insertRecord(222,"Aryan");
  s1.displayInformation();
  s2.displayInformation();
 }
}
```

C++ – classes, objects

```
#include <iostream>
using namespace std;
class Student {
   public:
       int id; //data member (also instance
variable)
       string name; //data member(also instance
variable)
       void insert(int i, string n)
        {
           id = i;
           name = n;
        }
       void display()
        {
           cout<<id<<"  "<<name<<endl;
        }
};
int main(void) {
    Student s1;  //creating an object of Student
    Student s2;  //creating an object of Student
    s1.insert(201, "Hari Shyam");
    s2.insert(202, "Nakul");
    s1.display();
    s2.display();
    return 0;
}
```

```
#include <iostream>
using namespace std;
class Employee {
   public:
       int id; //data member (also instance
variable)
       string name; //data member(also instance
variable)
       float salary;
       void insert(int i, string n, float s)
        {
           id = i;
           name = n;
           salary = s;
        }
       void display()
        {
           cout<<id<<"  "<<name<<"
"<<salary<<endl;
        }
};
int main(void) {
    Employee e1; //creating an object of Employee
    Employee e2; //creating an object of Employee
    e1.insert(201, "Harishyam",990000);
    e2.insert(202, "Nakul", 29000);
    e1.display();
    e2.display();
    return 0;
}
```

C++ - Structure

- Structs in C++
  - Structs are used for lightweight objects such as Rectangle, color, Point, etc.
  - structs in C++ are value type than reference type
  - It is useful, if data that is not intended to be modified after creation of struct.
  - **C++ Structure** is a collection of different data types, similar to the class that holds different types of data.
- Syntax
  - Structure in C++ is an extension of struct in C
  - It can have variable, methods, constructors
  - When structure is created, no memory is allocated
  - Memory allocation is done when variable is created

Structure syntax

```
struct Student
{
    char name[20];
     int id;
     int age;
};

int main(void) {
    struct Stundent stud;
    stud.id=8;
    stud.age=15;
   cout<<"ID and age of student are:
"<<stud.id<<stud.id<<endl;
 return 0;
}
```

C++ - Structure

- Struct in C++
  - By default access specifier is public
  - Instance of this is "Structure variable"

```
struct struct_name
{
//Body includes variable, method, constructor, etc..
}
```

- Class
  - By default access specifier is private
  - Instance of this is "Object"

```
class class_name
{
//Body of the class, include variable, method, constructor, etc..
}
```

Structure example

```
#include <iostream>
using namespace std;
 struct Rectangle    {
   int width, height;
  Rectangle(int w, int h)
    {
        width = w;
        height = h;
    }
  void areaOfRectangle() {
    cout<<"Area of Rectangle is:
"<<(width*height); }
 };
int main(void) {
    struct Rectangle rec=Rectangle(4,6);
    rec.areaOfRectangle();
   return 0;
}
```

C++ – Structure

```
#include <iostream>
using namespace std;
class Student {
   public:
       int id; //data member (also instance
variable)
       string name; //data member(also instance
variable)
       void insert(int i, string n)
        {
            id = i;
            name = n;
        }
       void display()
        {
            cout<<id<<"  "<<name<<endl;
        }
};
int main(void) {
    Student s1;  //creating an object of Student
    Student s2;  //creating an object of Student
    s1.insert(201, "Hari Shyam");
    s2.insert(202, "Nakul");
    s1.display();
    s2.display();
    return 0;
}
```

```
#include <iostream>
using namespace std;
class Employee {
   public:
       int id; //data member (also instance
variable)
       string name; //data member(also instance
variable)
       float salary;
       void insert(int i, string n, float s)
        {
            id = i;
            name = n;
            salary = s;
        }
       void display()
        {
            cout<<id<<"  "<<name<<"
"<<salary<<endl;
        }
};
int main(void) {
    Employee e1; //creating an object of Employee
    Employee e2; //creating an object of Employee
    e1.insert(201, "Harishyam",990000);
    e2.insert(202, "Nakul", 29000);
    e1.display();
    e2.display();
    return 0;
}
```

Constructor

- Constructors
  - Constructors are nothing but special type of member functions or methods, having same name as class
  - Constructors are used to initialize the objects of its class.
  - Constructor is invoked whenever an object of its associated class is created.

- Types of Constructors
  - Default constructors
  - Parameterized constructors
  - Copy constructors

Java – Constructor

- In Java, a constructor is a block of codes similar to the method with class name.
  - It is called when an instance of the object is created, and memory is allocated for the object.
  - Every time an object is created using new() keyword, at least one constructor is called and it is known as default constructor
  - It is known as constructor as it constructs the values at the time of object creation.
  - It is not mandatory to have a constructor because java compiler creates a default constructor class does not have any.
- Rules for creating constructor
  - Constructor name must be same as class name
  - It should not have any explicit return type
  - It can not be abstract, static, final and synchoronized

Java – Constructor(2)

- Types of constructor in Java
  - Default constructor → No arguments
  - Parameterized constructor
- Default constructor
  - Constructor can be called default if no parameter
  - <class_name>){}

```
class IIIT{
//creating a default constructor
IIIT(){System.out.println(“IIIT is created");}
//main method
public static void main(String args[]){
//calling a default constructor
IIIT ob=new IIIT();
}
}
```

```
class IIIT{
//creating a default constructor
int pincode;
String name;

Void
display(){System.out.println(pincode+" "+name);}

//main method
public static void main(String args[]){
//calling a default constructor
IIIT ob1 = new IIIT();
IIIT ob2 = new IIIT();
Ob1.display();
Ob2.display
}
}
```

Java – Constructor(2)

- Parameterized constructor
  - This Constructor is with no of parameters

```
class IIIT{
//creating a default constructor
int pincode;
String name;

IIIT(int i,String n){
    id = i;
    name = n;
    }


Void display(){System.out.println(pincode+" "+name);}

//main method
public static void main(String args[]){
//calling a default constructor
IIIT ob1 = new IIIT(517646,"Sri City");
IIIT ob2 = new IIIT(600127,"Kancheepuram");
Ob1.display();
Ob2.display();
}
}
```

Java – Constructor(2)

- Constructor Overloading

  This Constructors are with respective no of parameters

```
class IIIT{
//creating a default constructor
int pincode;
String name;
String state;

IIIT(int i,String n){
    id = i;
    name = n;
    }

IIIT(int i,String str1, String str2){
    id = i;
    name = str2;
    state = str2
    }

Void
display(){System.out.println(pincode+" "+name);}

//main method
public static void main(String args[]){
//calling a default constructor
IIIT ob1 = new IIIT(517646,"Sri City");
IIIT ob2 = new
IIIT(600127,"Kancheepuram","Tamilnadu");
Ob1.display();
Ob2.display();
}
}
```

C++ – Constructor

- In C++, a constructor is a block of codes similar to the method with class name or structure name
- It is called when an instance of the object is created, and memory is allocated for the object.
- It is used to initialize the data members of new object generally
- It is known as constructor as it constructs the values at the time of object creation.
- It is not mandatory to have a constructor because compiler creates a default constructor class does not have any.
- Rules for creating constructor
  - Constructor name must be same as class name
  - It should not have any explicit return type
  - It can not be static

C++ – Constructor(1)

- Default constructor
  - This Constructor is with no of parameters

```
#include <iostream>
using namespace std;
class Employee
 {
   public:
        Employee()
        {
            cout<<"Default Constructor Invoked"<<endl;
        }
};
int main(void)
{
    Employee e1; //creating an object of Employee
    Employee e2;
    return 0;
}
```

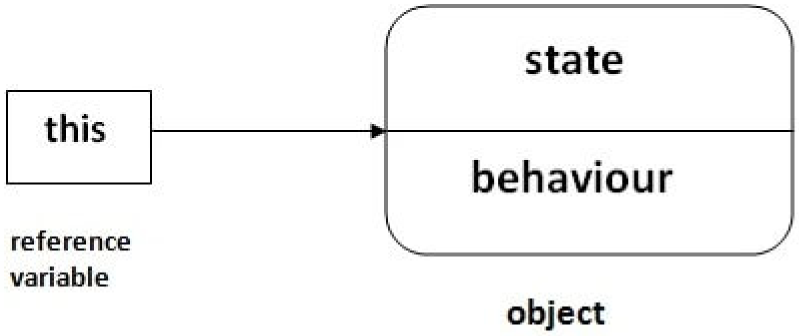
C++ – Constructor(2)

- Parameterized constructor
  - This Constructor is with no of parameters

```
#include <iostream>
using namespace std;
class IIIT {
//creating a default constructor
Public:
int pincode;
string name;

IIIT(int i, string n){
    id = i;
    name = n;
    }

void display(){System.out.println(pincode+" "+name);
};

int main(void) {
    IIIT i1 =IIIT(101001, "Sri City"); //creating an object of IIIT
    IIIT i2 =IIIT(101005, "New Delhi"); //creating an object of IIIT
    i1.display();
    i2.display();
    return 0;
}
```

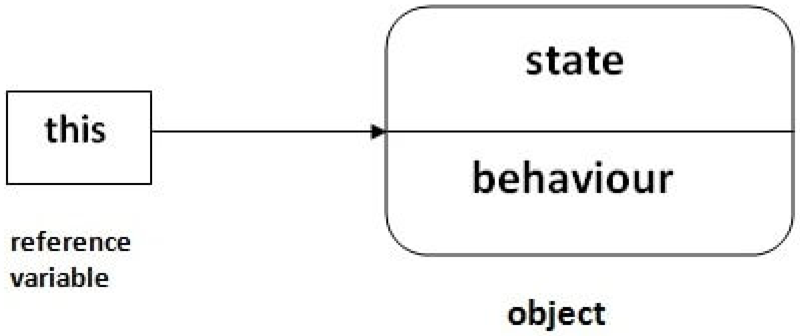
Java – this keyword

- In Java this is a keyword and also a reference variable.
- Usage of “this” keyword
  - It can be used to refer current class instance variable.
  - It can be used to invoke current class method implicitly
  - Can be used to invoke current class constructor
  - Can be passed an argument in the constructor/method call
  - Can also be used to return the current class instance from the method



C++ – this pointer

- In C++ programming, **this** is a keyword that refers to the current instance of the class. There can be 3 main usage of this keyword in C++.
- Usage of “this” keyword
  - used to pass current object as a parameter to another method.
  - used to refer current class instance variable.
  - used to declare indexers.



Example

```
#include <iostream>
using namespace std;
class Employee {
   public:
       int id; //data member
       string name; //data member
       float salary;
       Employee(int id, string name, float salary) {
             this->id = id;
            this->name = name;
            this->salary = salary;
        }
       void display() {
            cout<<id<<"  "<<name<<"
"<<salary<<endl;
        }
};
int main(void) {
    Employee e1 =Employee(101, "Shyam",
890000);
    Employee e2=Employee(102, "Nakul", 59000);
    e1.display();
    e2.display();
    return 0;
}
```