RAMAIAH
Institute of Technology

M.S. Ramaiah Institute of Technology
(Autonomous Institute, Affiliated to VTU)
Department of Computer Science and Engineering

Course Name: Object Oriented Programming
Course Code: CS36
Credits: 3:0:0

Term: September – December 2020

Faculty:
Dr. Geetha J
Hanumantha Raju

RAMAIAH
Institute of Technology

Understanding static

We can define a class member that will be used independently of any object of that class.

When a member (variable or method) is declared static, it can be accessed before any objects of its class are created.

Methods declared as static have several restrictions

- They can only directly call other static methods.
- They can only directly access static data.
- They cannot refer to this or super in any way.

RAMAIAH
Institute of Technology

Understanding static

```
class StaticDemo{
        static int a = 10;          //Class Variable
        int b = 10;                 //Instance Variable

        public void display(){
                a = a + 10;
                b = b + 10;
                System.out.println("a : " + a);
                System.out.println("b : " + b);
        }

        public void display1(){
                a = a + 10;
                //b = b + 10;
                System.out.println("a : " + a);
                //System.out.println("b : " + b);
        }
}

public class Test{
        public static void main(String args[]){
                StaticDemo obj = new StaticDemo();
                StaticDemo obj1 = new StaticDemo();

                obj.display();  //a = 20,   b = 20
                obj1.display1(); //a = 30
        }
}
```

RAMAIAH
Institute of Technology

Understanding static

```
class UseStatic
{
        static int a = 3;
        static int b;
        static void meth(int x)
        {
                System.out.println("x = " + x);
                System.out.println("a = " + a);
                System.out.println("b = " + b);
        }
        static
        {
                System.out.println("Static block initialized.");
                b = a * 4;
        }
        public static void main(String args[])
        {
                meth(42);
        }
}
```

RAMAIAH
Institute of Technology

Understanding static

```
class test {
        static int a = 10;
        public static void display() {
                a = a + 20;
        }
}

public class TestStatic {
        public static void main(String[] args) {
                System.out.println("Hello World");
                test.display();
                System.out.println("a : " + test.a);
                test obj = new test();
                test.display();
                System.out.println("a : " + test.a);
        }
}
```

RAMAIAH
Institute of Technology

Constructors In Java

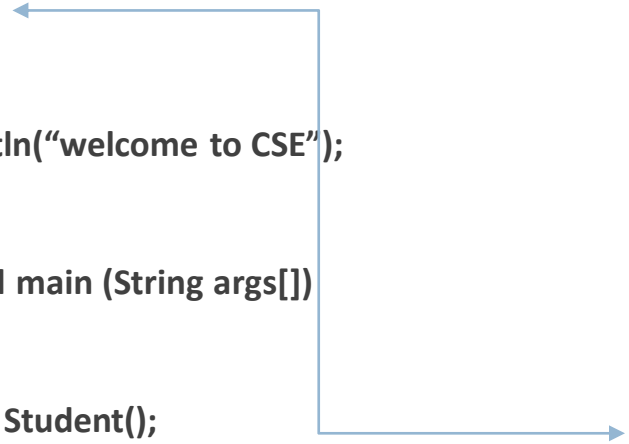
A constructor is a special method that is invoked when a new object is created. If we want to perform any one-time activities on an object at the time of its creation, then the constructor is the right place.

1. Generally, the initialization of **instance variables are done in the constructor.**
2. The purpose of a constructor is not limited to initialization. We can perform any activity in the constructor that we can do in any normal method.
3. The difference is constructor is invoked automatically (implicitly) and normal method is invoked explicitly.
4. Name of the constructor and its class name should be same.
5. A constructor should not have a return type (void is also not allowed)

RAMAIAH Institute of Technology

Example

```
Class Student
{
//Constructor
 Student()
{
System.out .prntln("welcome to CSE");
{
Public static void main (String args[])
{
Student s1=new Student();
…….   New Keyword creates the object of Student class
          & invokes the constructor to initialize the created object
 }
}
```



RAMAIAH Institute of Technology

Constructor in Java

The purpose of constructor is to initialize an object called object initialization. Initialization is a process of assigning user defined values and eliminate default value.

Constructor has the same name as the class name.

Constructor cannot return values.

A class can have more than one constructor as long as t signature (i.e., different input arguments syntax).

- There is NO explicit invocation statement needed: When the object creation statement is executed, the constructor method will be executed automatically.

```
class Sum  {
static int a,b;
Sum()
{
a=10; b=20;
}
public static void main(String s[]) {
Sum s1=new Sum();
int c=a+b;
System.out.println("Sum: "+c); } }
```

RAMAIAH Institute of Technology

Difference between Method and Constructor

	Method	Constructor
1	Method can be any user defined name	Constructor must be class name
2	Method should have return type	It should not have any return type (even void)
3	Method should be called explicitly either with object reference or class reference	It will be called automatically whenever object is created
4	Method is not provided by compiler in any case.	The java compiler provides a default constructor if we do not have any constructor.

RAMAIAH Institute of Technology

Types of constructors

Default Constructor

```
class Student
{
int roll;
float marks;
String name;
void show()
{
System.out.println("Roll: "+roll);
System.out.println("Marks: "+marks);
System.out.println("Name: "+name);
}
}
class TestDemo
{
public static void main(String [] args)
{
Student s1=new Student();
s1.show();
}
}
```

Parameterized Constructor

```
class Test
{
int a, b;
Test(int n1, int n2)
{
System.out.println("I am from Parameterized Constructor...");
a=n1;
b=n2;
System.out.println("Value of a = "+a);
System.out.println("Value of b = "+b);
}
};
class TestDemo1
{
public static void main(String k [])
{
Test t1=new Test(10, 20);
}
};
```

RAMAIAH
Institute of Technology

Constructor Overloading

```
class Student{
  int id, age;
  String name;
  Student(int i,String n)
   {
  id = i;
  name = n;
  }
  Student(int i,String n,int a)
   {
  id = i;
  name = n;
  age=a;
  }
  void display(){
System.out.println(id+" "+name+" "+age);
}
```

Constructor overloading is a technique in Java in which a class can have any number of constructors that differ in parameter lists. The compiler differentiates these constructors by taking the number of parameters, and their type.

Output
111 Raman 0
222 Mohan 25

RAMAIAH Institute of Technology

Types of polymorphism in java

There are two types of polymorphism in java

- **Runtime polymorphism(Dynamic polymorphism)**
- **Compile time polymorphism (Static polymorphism)**.

Method overriding is a perfect example of runtime polymorphism.

Method overloading is a perfect example of compile time polymorphism.

- Operator Overloading (Supported in C++, but not in Java)

RAMAIAH
Institute of Technology

Method Overloading in Java

Method Overloading is a feature that allows a class to have two or more methods having same name, if their argument lists are different. It is also known as **Static Polymorphism.**

Argument lists could differ in –

1. Number of parameters.
2. Data type of parameters.
3. Sequence of Data type of parameters.

```
class Overloading
{
    public void disp(char c, int num)
    {
        System.out.println("first disp method"+c+num);
    }
    public void disp(int num, char c)
    {
        System.out.println("second disp method"+num+c);
    }
}
class Sample3
{
    public static void main(String args[])
    {
        Overloading obj = new Overloading();
        obj.disp('x', 51 );
        obj.disp(52, 'y');
    }
}
```

RAMAIAH
Institute of Technology

Different Number of parameters in argument list

```
class ExOverload
{
     public void disp(char c)
     {
          System.out.println(c);
     }
     public void disp(char c, int num)
     {
          System.out.println(c + " "+num);
     }
}
class Sample
{
    public static void main(String args[])
    {
         ExOverload obj = new ExOverload ();
         obj.disp('a');
         obj.disp('a',10);
    }
}
```

Difference in data type of arguments

```
class ExOverload1
{
     public void disp(char c)
     {
          System.out.println(c);
     }
     public void disp(int c)
     {
        System.out.println(c );
     }
}

class Sample2
{
     public static void main(String args[])
     {
          ExOverload1 obj = new ExOverload1();
          obj.disp('a');
          obj.disp(5);
     }
}
```

RAMAIAH Institute of Technology

The *this* keyword

this keyword can be used to refer to the object itself. It is generally used for accessing class members (from its own methods) when they have the same name as those passed as arguments.

```
public class Circle {
   public double x,y,r;
     // Constructor
   public Circle (double x, double y, double r) {
      this.x = x;
      this.y = y;
      this.r = r;
    }
      //Methods to return circumference and area
  }
```

RAMAIAH
Institute of Technology

Introducing Nested and Inner Classes

Defining a class within another class is known as nested class.

The scope of a nested class is bounded by the scope of its enclosing class.

Thus, if class B is defined within class A, then B does not exist independently of A.

A nested class has access to the members, including private members, of the class in which it is nested.

RAMAIAH
Institute of Technology

Introducing Nested and Inner Classes

However, the enclosing class does not have access to the members of the nested class.

A nested class that is declared directly within its enclosing class scope is a member of its enclosing class.

It is also possible to declare a nested class that is local to a block.

There are two types of nested classes:

- **Static**
- **Non-Static.**

RAMAIAH
Institute of Technology

Introducing Nested and Inner Classes

The most important type of nested class is the **inner class**.

An inner class is a non-static nested class.

It has access to all of the variables and methods of its outer class and may refer to them directly in the same way that other non-static members of the outer class do.

RAMAIAH Institute of Technology

Non-Static Inner Classes

```
// Demonstrate an inner class.
class Outer {
  int outer_x = 100;

  void test() {
    Inner inner = new Inner();
    inner.display();
  }

  // this is an inner class
  class Inner {
    void display() {
      System.out.println("display: outer_x = " + outer_x);
    }
  }
}

class InnerClassDemo {
  public static void main(String args[]) {
    Outer outer = new Outer();
    outer.test();
  }
}
```

RAMAIAH Institute of Technology

Inner Classes within a Scope

```
// Define an inner class within a for loop.
class Outer {
    int outer_x = 100;
    void test() {
        for(int i=0; i<10; i++) {
            class Inner {
                void display() {
                    System.out.println("display: outer_x = " + outer_x);
                }
            }
            Inner inner = new Inner();
            inner.display();
        }
    }
}
public class InnerClassDemo {
    public static void main(String args[]) {
        Outer outer = new Outer();
        outer.test();
    }
}
```

RAMAIAH
Institute of Technology

Static Inner Class

```
public class Main{
        public static void main(String[] args){
                System.out.println("Hello World");
                Test obj = new Test();
                Test.A obj1 = new Test.A();
        }
}
class Test{
        Test(){
                System.out.println("Test Constructor");
        }
        public static class A{
                A(){
                        System.out.println("Class A");
                }
        }
}
```

RAMAIAH Institute of Technology

Pass By Value

Java uses *pass by value* to pass arguments to a method. There are important differences between passing a value of variables of primitive data types and passing arrays.

For a parameter of a primitive type value, the actual value is passed. Changing the value of the local parameter inside the method does not affect the value of the variable outside the method.

For a parameter of an array type, the value of the parameter contains a reference to an array; this reference is passed to the method. Any changes to the array that occur inside the method body will affect the original array that was passed as the argument.

RAMAIAH Institute of Technology

Simple Example

```
public class Test {
   public static void m(int number, int[] numbers) {
     number = 1001; // Assign a new value to number
     numbers[0] = 5555; // Assign a new value to numbers[0]
   }

   public static void main(String[] args) {
     int x = 1; // x represents an int value
     int[] y = new int[10]; // y represents an array of int values

     m(x, y); // Invoke m with arguments x and y

     System.out.println("x is " + x);
     System.out.println("y[0] is " + y[0]);
   }
```

RAMAIAH Institute of Technology

Does Java pass by value or pass by reference - Interview Question

As per Java specification **everything in Java is pass by value** whether its primitive value or objects and it does make sense because Java doesn't support pointers or pointer arithmetic.

Answer to this question is simple whenever a method parameter expect object, reference of that object is passed.

In reality if you pass object as method parameter in Java it passes "value of reference" or in simple term object reference or handle to Object in Java. Here reference term is entirely different than reference term used in C and C+ which directly points to memory address of variable and subject to pointer arithmetic

Example-passing object

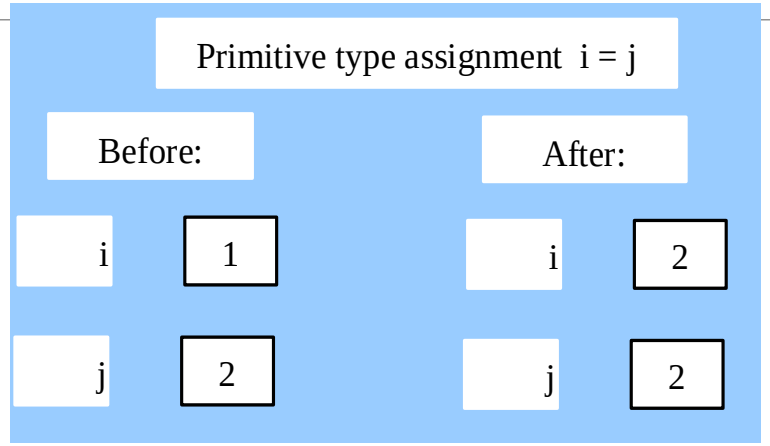
```
public class ObjPass {
private int value;
public static void increment(ObjPass a)
{
System.out.println(a);
a.value++;
}
public static void main(String args[])
{
   ObjPass p = new ObjPass();
   p.value = 5;
   System.out.println("Before:" + p.value);
   increment(p);
   System.out.println("After: " + p.value);
   System.out.println(p);
}}
```

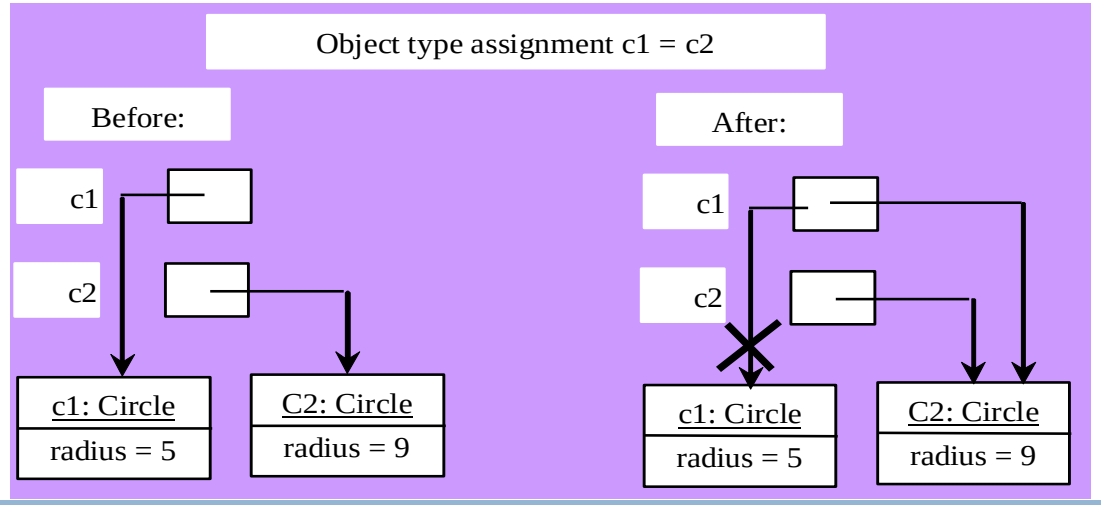
Here we pass exactly is a handle of an object, and in the called method a new handle created and pointed to the same object. From the example above you can see that both **p** and **a** refer to the same object.

```
class Car
 {
  String model;   //instance variable
  Car() {         //constructor to initialize
    model="Maruthi";
System.out.println("Car Model is:"+model);
}
void disp(Car m) {
System.out.println("My Car Model is:"+m.model);
} }
public class ObjPass1 {
public static void main(String args[])
{
  Car mycar = new Car();
   mycar.model="Zen";
   mycar.disp(mycar);
} } }
```

We can access the instance variables of the object passed inside the called method. It is good practice to initialize instance variables of an object before passing object as parameter to method otherwise it will take default initial values.

Copying Variables of Primitive Data Types and Object Types





RAMAIAH
Institute of Technology

Garbage Collection

As shown in the previous figure, after the assignment statement c1 = c2, c1 points to the same object referenced by c2. The object previously referenced by c1 is no longer referenced. This object is known as garbage. Garbage is automatically collected by JVM.

RAMAIAH
Institute of Technology

Stack Implementation

```
class TestStack {
        public static void main(String args[]) {
                Stack obj1 = new Stack();
                Stack obj2= new Stack();
                for(int i=0; i<10; i++)
                        obj1.push(i);
                for(int i=0; i<10; i++)
                        obj2.push(i);
                System.out.println("\nFirst Stack:\n");
                for(int i=0; i<10; i++)
                        System.out.println(obj1.pop()+"\t");
                System.out.println("\nSecond Stack:\n");
                for(int i=0; i<10; i++)
                        System.out.println(obj2.pop()+"\t");
        }
}
```

RAMAIAH
Institute of Technology

Introducing Nested and Inner Classes

Defining a class within another class is known as nested class.

The scope of a nested class is bounded by the scope of its enclosing class.

Thus, if class B is defined within class A, then B does not exist independently of A.

A nested class has access to the members, including private members, of the class in which it is nested.

RAMAIAH
Institute of Technology

Exercises

Write a Java class Student to meet the following specification.

- The class should be able to support a 5 digit student ID, student name, marks for 3 subjects. You should have methods to set and get each of the attributes, and calculate the average for the student. Write a tester program to test your class. You should create 2 or 3 students and write code to test the class. Aim - Understand how to define a class and create objects of the class.

For the above student Class display total number of students you have entered using static variable.

Write a java program to get cube of a given number by static method.

Thank you