CLASSES AND OBJECTS-PROGRAMS AND CONSTRUCTORS

Presented by:

Dr. Shivangi K. Surati

Assistant Professor,
Department of Computer Science and Engineering,
School of Technology,
Pandit Deendayal Energy University

Outline

- Default Arguments
- Variable Arguments (Varargs) and examples
- Class- Simple Program
- Creating object of the same class
- Constructors
- Destructor

Default Arguments

- an argument to a function that a programmer is not required to specify
- Default values can be given
- EX:

```
class defaultArgu{
    void add(int a, int b=5) {
            System.out.println(a+b);
    }
    public static void main(String[] args) {
            add(5,10);
            add(5);
    }
}
```

Variable Arguments (Varargs)

- A method that takes a variable number of arguments
- Syntax of Varargs

```
public static void fun(int ... a) // (data_type … variable_name)
{
   // method body
}
```

- Internally, the Varargs method is implemented by using the one dimensional arrays concept.
- Hence, in the Varargs method, arguments are differentiated by using Index.

Varargs Example

```
class Test1 {
    // Method that takes variable number of integer arguments.
    static void fun(int... a)     {
        System.out.println( "Number of arguments: ” + a.length);
        for (int i : a) // using for each loop to display contents of a
            System.out.print(i + " ");
   }
 public static void main(String args[])     {
        fun(100);             // one parameter
        fun(1, 2, 3, 4);   // four parameters
        fun();                   // no parameter
    }
}
```

Varargs...

- A method can have variable length parameters with other parameters too
- only one varargs parameter that should be written last in the parameter list of the method declaration

EX: int nums(int a, float b, double … c)

<u>Errors:</u>

- Specifying two Varargs in a single method:

```
void method(String... gfg, int... q) //error
```

- Specifying Varargs as the first parameter of the method instead of the last one:

```
void method(int... gfg, String q) //error
```

Class- Simple Program

```
Class student{
        String name;
        String roll no;
        int marks;
        void setData(); //setter
        void printData();
}
```

Creating object of the same class

```
class Temp{
        int a;
        public static void main(String[] args) {
                Temp objTemp = new Temp();
                objTemp.a=10;
                System.out.println(objTemp.a);
                System.out.println(objTemp); //check output
        }
}
```

Constructors

- A special member function
  - to initialize objects with default values unless different values are supplied
  - that takes the same name as the class name
  - that cannot return values (No return type)
  - that is invoked automatically at the time of object creation
  - that can be overloaded
  - The syntax generally is as given below:

<class name> {arguments};

Constructors…

- Several forms:
  - default constructor (without parameter)
  - parameterized
  - copy constructor
- We can define constructors with default arguments
- Unlike methods, constructors are not considered members of the class
- Constructor overloading

Simple Program

```
Class Point{
    int x, y, z;
    Point(); //default constructor
    Point(int, int, int); // parameterized constructor
    setData();
    getData();
    translate();
    calDistanceOrigin();
    }
```

Lect-7_prog

Copy Constructor

- When it is required to create an exact copy of an existing object of the class such that
  - if we have made any changes in the copy it should not be reflected in the original one and vice-versa.
- A special type of constructor that creates an object using another object of the same Java class (Deep copy)
- Parameter- Object of the same class
- Copies all attributes of first object into second object
- Returns a duplicate copy of an existing object of the class
- EX:

Copy constructors.doc

Destructor

- A special member function
  - To release dynamic allocated memory
  - Same name as class name
  - No return type
  - Cannot be overloaded (only one)
  - finalize() method:

```
protected void finalize()
{
    System.out.println("Object is destroyed by the Garbage Collector");
}
```

Questions

- Difference between constructor and method in Java?
- Can we overload main method?
- Difference between VarArgs and method overloading?
- Difference between copy constructor and ob.clone() method?