CS205 Object Oriented Programming in Java

Module 2 - **Core Java Fundamentals (Part 7)**

Prepared by

Renetha J.B.

AP

Dept.of CSE,

Lourdes Matha College of Science and Technology

Topics

- Core Java Fundamentals**:**

  ✓ **Returning Objects**

  ✓ **Recursion**

  ✓ **Access Control**

  ✓ **Static Members**

Returning objects

- A method can **return** **any type of data,**

  – **Primitive data (int ,float, char, double etc.)**

  – class types(objects) that you create.

  – etc.

```java
// Returning an object.
class Test {
    int a;
    Test(int i)
    {
        a = i;
    }
    Test increase()
    {
        Test temp = new Test(a+10);
        return temp;
    }
}
```
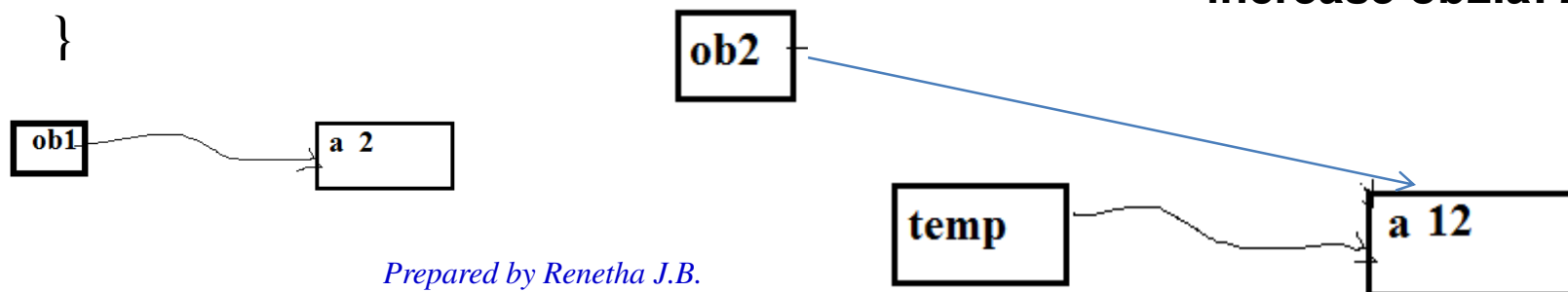
```java
class RetOb {
    public static void main(String args[]) {
        Test ob1 = new Test(2);
        Test ob2;
        ob2 = ob1.increase();
        System.out.println("ob1.a: " + ob1.a);
        System.out.println("ob2.a: " + ob2.a);
        ob2 = ob2.increase ();
        System.out.println("increase ob2.a: " +ob2.a);
    }
}
```

OUTPUT
ob1.a: 2
ob2.a: 12
increase ob2.a: 22

ob2

temp   a 12

ob1   a 2

4

Prepared by Renetha J.B.

Recursion

- Recursion is the process of **defining something in terms of itself.**

- A method that calls itself is called *recursive function.*

```java
// A simple example of recursion.
class Factorial {
    int fact(int n)
     {
         int result;
         if(n==1)
            return 1;
         result = n* fact(n-1) ;
         return result;
}
}
class Recursion {
public static void main(String args[]) {
Factorial f = new Factorial();
int s= f.fact(3)
System.out.println("Factorial of 3 is " + s);
}
}
```

Example

```java
class RecTest {
int values[];
RecTest(int i) {
values = new int[i];
}
void printArray(int i) {
if(i==0)
    return;
else
  printArray(i-1);
System.out.println("[" + (i-1) + "] " + values[i-1]);
}
}
```

```java
class Recursion2 {
public static void main(String args[])
    {
RecTest ob = new RecTest(5);
int i;
for(i=0; i<5; i++)
    ob.values[i] = i;
ob.printArray(5);
}
}
```

- printArray(5) $i = 5$ System.out.println("[" + (i-1) + "] " + values[i-1]); [4] 4
  - printArray(4) $i = 4$ System.out.println("[" + (i-1) + "] " + values[i-1]); [3] 3
    - printArray(3) $i = 3$ System.out.println("[" + (i-1) + "] " + values[i-1]); [2] 2
      - printArray(2) $i = 2$ System.out.println("[" + (i-1) + "] " + values[i-1]); [1] 1
        » printArray(1) $i = 1$ System.out.println("[" + (i-1) + "] " + values[i-1]); [0] 0
        » printArray(0)
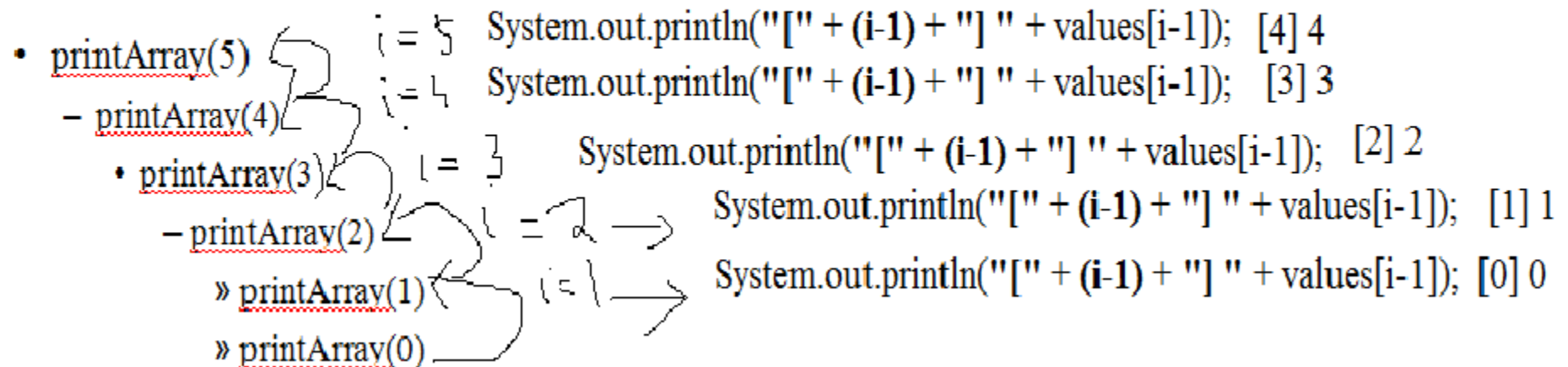
```
OUTPUT
[0] 0
[1] 1
[2] 2
[3] 3
[4] 4
[5] 5
```

Access Control

- Through encapsulation, we can **control** what parts of a program can **access the members** of a class.

  – By controlling access, you can prevent misuse.

- How a member can be accessed is determined by the _access specifier that modifies its declaration_

- Java's access specifiers are

  ✓ **public**

  ✓ **private**

  ✓ **protected**

  ✓ _default_

Access Control(contd.)

- When a **member** of a class is modified by the **public** specifier, then that <u>member can be accessed by any other code.</u> (ACCESSIBLE TO ALL)

    – public int i;

- When a member of a class is specified as **private**, then that member can only be <u>accessed by any members of the same class.</u>

    – private int a;

Access Control(contd.)

- When a member of a class is specified as **protected** , then that member can be accessed within the package and by any of its subclasses.

  protected char c;

- When <u>no access specifier</u> is there, then its access specifier is **default**.

  - It can be accessed within its own package, but **cannot be accessed outside of its package**

  int c;

Access sprcifier-E.g.

```java
class A{
    public int i;

    private double j;

    protected char c;

    float f;                        //default access

    public int myMethod(int a, char b)        //public method

    {   //..

    }

}
```

	PRIVATE	DEFAULT	PROTECTED	PUBLIC
Same class	Yes	Yes	Yes	Yes
Same package Subclass	No	Yes	Yes	Yes
Same package Non-subclass	No	Yes	Yes	Yes
Different package Subclass	No	No	Yes	Yes
Different package Non-subclass	No	No	No	Yes

SAME CLASS   SAME PACKAGE,   SAME PACKAGE , ANY SUBCLASS   ALL

Prepared by Renetha J.B.

```java
class Test
{
int a;              // default access
public int b;       // public access
private int c;      // private access


void setc(int i)            //setter
{
c = i;
}


int getc()          //getter
{
return c;
}   }
```

```java
class AccessTest {
public static void main(String args[]) {
Test ob = new Test();
ob.a = 10;
ob.b = 20;
// ob.c = 100; // Error! // PRIVATE
// You must access private variable c
    //through its methods
ob.setc(100);           // OK
System.out.println("a="+ ob.a);
System.out.println("b="ob.b");
System.out.println("c= " + ob.getc() );
}
}
```

static Members

- Usually we access the member of another class using object. *Syntax is:* objectname**.**member**;**

- If we want to <u>access a member of another class without using object</u>, then we have to make it a **make it a static member.**

  - Static <u>class member</u> is **independent of any object** of that class. We can make a member static by <u>preceding the member declaration with the keyword</u> **static.**

  static datatype member**;**

static Members(contd.)

- When a member is declared **static,** it can be accessed before any objects of its class are created, and without reference to any object.

- Static member can be accessed using

classname.member;

static Members(contd.)

- The most common example of a **static member is** main function.

  - main( ) is declared as static because it must be called before any objects is created.

- Instance variables declared as **static** are <u>global variables</u>.

- When objects of its class are declared, separate copy of a static variable is NOT made.

- All instances(objects) of the class **share the same static variable.**

static Members(contd.)

- Methods declared as **static(static methods)** have several restrictions:

  - **static methods** can only call other **static methods.**

  - **static methods** must only access **static data.**

  - **static methods** cannot refer to **this** or **super.**

static Members(contd.)

- If we need to do <u>computation to initialize your static variables, we</u> can **declare a static block** <u>that gets executed exactly once, when the class is first loaded.</u>

- // Demonstrate static variables, methods, and blocks.

```java
class UseStatic {
static int a = 3;
static int b;
static void show(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[])
{show(42);
}
}
```

```
OUTPUT
Static block initialized.
x = 42
a = 3
b = 12
```

Working of e.g. code

- As soon as the **UseStatic** class is loaded, all of the static statements are run.

  - First, static member **a** is set to 3,

  - then the static block executes, which prints a message and then initializes b to a * 4 or 12.

  - Then main( ) is called, which calls show( ), passing 42 to x.

  - The three println( ) statements in show refer to the two static variables a and b, as well as to the local variable x.

- if we want to call a **static method from outside its class, we** can do so using the following general form:

classname.*method( )*

- *Here classname is the name of the class in which the* **static method is declared.**

Non-static method invocation

```java
class Demo {
int a = 42;
int b = 99;
void callme()
{
System.out.println("a = " + a);
}
}
class Sample {
public static void main(String args[]) {
Demo dm=new Demo ();
dm.callme();
System.out.println("b = " + dm.b);
} }
```

static method invocation

```java
class StaticDemo {

static int a = 42;

static int b = 99;

static void callme()

{

System.out.println("a = " + a);

}

}

class StaticByName {

public static void main(String args[])

{

StaticDemo.callme();

System.out.println("b = " + StaticDemo.b);

} }
```

Nonnstatic members   static members

```java
class Demo {
int a = 42;
int b=5;
void callme()
{
System.out.println("a = " + a);
}

}
class Sample {
public static void main(String args[])
    {
Demo dm=new Demo();
dm.callme();
System.out.println("b = " + dm.b);
} }
```

```java
class StaticDemo {
int a = 42;
static int b = 5;
static void callme()
{
System.out.println("a = " + a);
}
}
class StaticByName {
public static void main(String args[])
{
StaticDemo.callme();
System.out.println("b = " + StaticDemo.b);
} }
```

```java
class Sample
{
static int a = 0;
int b;
Sample()
{
    b=0;

}
void callme()
{

a=a+2;
b=b+2;
System.out.println("static after +2 a = " + a);
System.out.println("b after +2 = " + b);
}
}
```
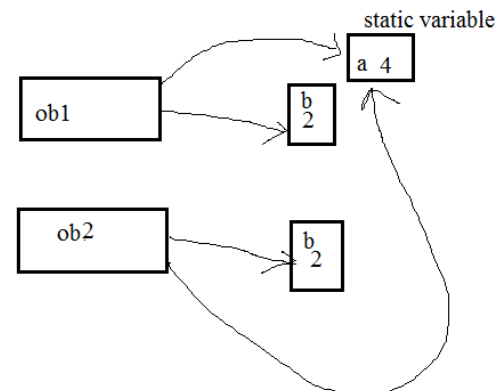
```java
class Samplestat {
public static void main(String args[])
{
Sample ob1=new Sample();
System.out.println("ob1");
ob1.callme();

Sample ob2=new Sample();
System.out.println("ob2");
ob2.callme();

} }
```

OUTPUT
ob1
static after +2 a = 2
b after +2 = 2
ob2
static after +2 a = 4
b after +2 = 2

Reference

- Herbert Schildt, Java: The Complete Reference, 8/e, Tata McGraw Hill, 2011.