Marwadi University

Unit – 3
Objects and Classes

Prepared By
Prof. Ravikumar Natarajan
Assistant Professor, CE Dept.

KNOWLEDGE IS THE CURRENCY
FOR THE 21st CENTURY

Contents

Marwadi University

- Defining classes for objects
- Declaring objects
- New keyword
- Defining and calling methods in class
- Array of objects
- Constructors
- This keyword
- Garbage collection
- Finalize() method
- Passing object as parameters
- Returning object
- Static members

Defining classes for objects

- A class defines the properties and behaviors for objects.
- Object-oriented programming (OOP) involves programming using objects.
- An object represents an entity in the real world that can be distinctly identified. For example, a student, a desk, a circle, a button, and even a loan can all be viewed as objects.
- An object has a unique identity, state, and behavior.

Declaring Objects

Marwadi University

- The object is a basic building block of an OOPs
- language. In Java, we cannot execute any program without creating an object.
- Java provides five ways to create an object.

- Using new Keyword
- Using clone() method
- Using newInstance() method of the Class class
- Using newInstance() method of the Constructor class
- Using Deserialization

Keyword “new”

Marwadi University

- Using the new keyword is the most popular way to create an object or instance of the class.
- When we create an instance of the class by using the new keyword, it allocates memory (heap) for the newly created object and also returns the reference of that object to that memory.
- The new keyword is also used to create an array. The syntax for creating an object is:

- ClassName object = new ClassName();

Keyword “new”

Marwadi University

```
Class Main
{
    void show()
    {
        System.out.println(“Hi");
    }
    public static void main(String[] args)
    {
    //creating an object using new keyword
        Main obj = new Main();
        obj.show();
}   }
```

Method in Java

Marwadi University

A Java method is a collection of statements that are grouped together to perform an operation. When you call the System.out.println() method, for example, the system actually executes several statements in order to display a message on the console.

Now we will learn how to create your own methods with or without return values, invoke a method with or without parameters, and apply method abstraction in the program design.

Creating method

Marwadi University

Considering the following example to explain the syntax of a method –

Syntax:

```
modifier returnType nameOfMethod (Parameter List) {
// method body
}
```

Methods in Java

The syntax shown above includes –

- **modifier** – It defines the access type of the method and it is optional to use.
- **returnType** – Method may return a value.
- **nameOfMethod** – This is the method name. The method signature consists of the method name and the parameter list.
- **Parameter List** – The list of parameters, it is the type, order, and number of parameters of a method. These are optional, method may contain zero parameters.
- **method body** – The method body defines what the method does with the statements.

Example

Marwadi University

```
public static int minFunction(int n1, int n2) {
int min;
  if (n1 > n2)
        min = n2;
  else
        min = n1;
  return min;
}
```

Method Calling

Marwadi University

For using a method, it should be called. There are two ways in which a method is called i.e., method returns a value or returning nothing (no return value).

The process of method calling is simple. When a program invokes a method, the program control gets transferred to the called method. This called method then returns control to the caller in two conditions, when

- ✓ the return statement is executed.
- ✓ it reaches the method ending closing brace.

The methods returning void is considered as call to a statement. Lets consider an example − System.out.println("This is method!");

The method returning value can be understood by the following example − int result = sum(6, 9);

Examples

Marwadi University

Check Programs → FunctionWithReturn.java , FunctionWithoutReturn.java

FunctionWithReturn

Marwadi University

```
//A method with a return value
public class Main {
  static int myMethod(int x) {
    return 5 + x;
  }
  public static void main(String[] args) {
    System.out.println(myMethod(3));
  }
}
```

```
public class ReturnTypeTest1 {
  public int add() { // without arguments
    int x = 30;      int y = 70;      int z = x+y;
    return z;
  }
  public static void main(String args[]) {
    ReturnTypeTest1 test = new ReturnTypeTest1();
    int add = test.add();
    System.out.println("The sum of x and y is: " + add);
  }
}
```

```
public class ReturnTypeTest2 {
  public int add(int x, int y) { // with arguments
    int z = x+y;
    return z;
  }
  public static void main(String args[]) {
    ReturnTypeTest2 test = new ReturnTypeTest2();
    int add = test.add(10, 20);
    System.out.println("The sum of x and y is: " + add);
  }
}
```

FunctionWithoutReturn

Marwadi University

```
//A method without any return values:
public class Main {
  static void myMethod() {
    System.out.println("I just got
executed!");
  }

  public static void main(String[] args) {
    myMethod();
  }
}
```

```
// Java Program to Illustrate a Method
// with 2 Parameters and without Return Type
import java.util.*;
public class Main {
        public static void main(String args[])
        {
                int a = 4;
                int b = 5;

                // Calling the function with 2 parameters
                calc(a, b);
        }
        public static void calc(int x, int y)
        {
                int sum = x + y;
                // Displaying the sum
                System.out.print("Sum of two numbers is :" + sum);
        }

}
```

Parameter Passing in Java

Marwadi University

There are different ways in which parameter data can be passed into and out of methods and functions. Let us assume that a function B() is called from another function A(). In this case A is called the "caller function" and B is called the "called function or callee function". Also, the arguments which A sends to B are called actual arguments and the parameters of B are called formal arguments.

Types of parameters:

Formal Parameter : A variable and its type as they appear in the prototype of the function or method.

Syntax:

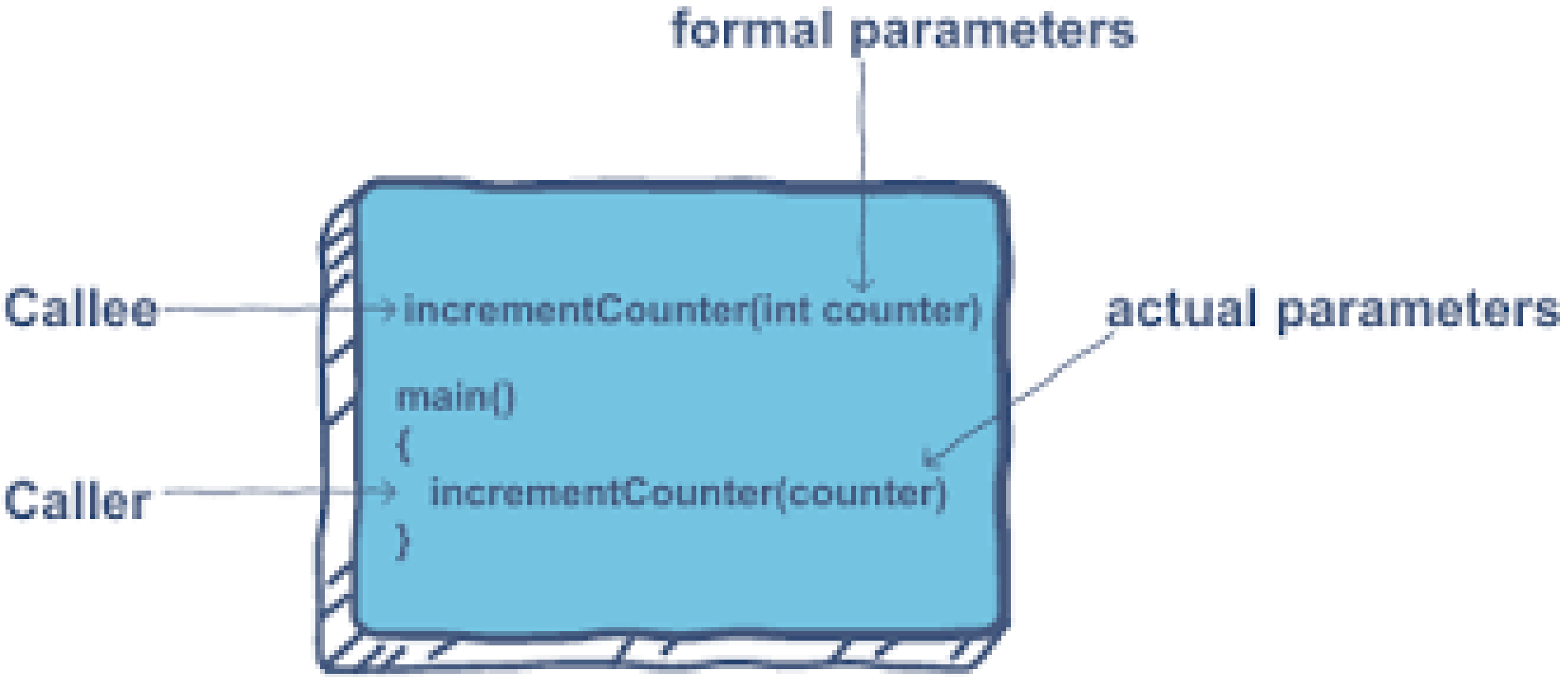
```
function_name(datatype variable_name)
```

Actual Parameter : The variable or expression corresponding to a formal parameter that appears in the function or method call in the calling environment.

Syntax:

```
func_name(variable name(s));
```

Marwadi University

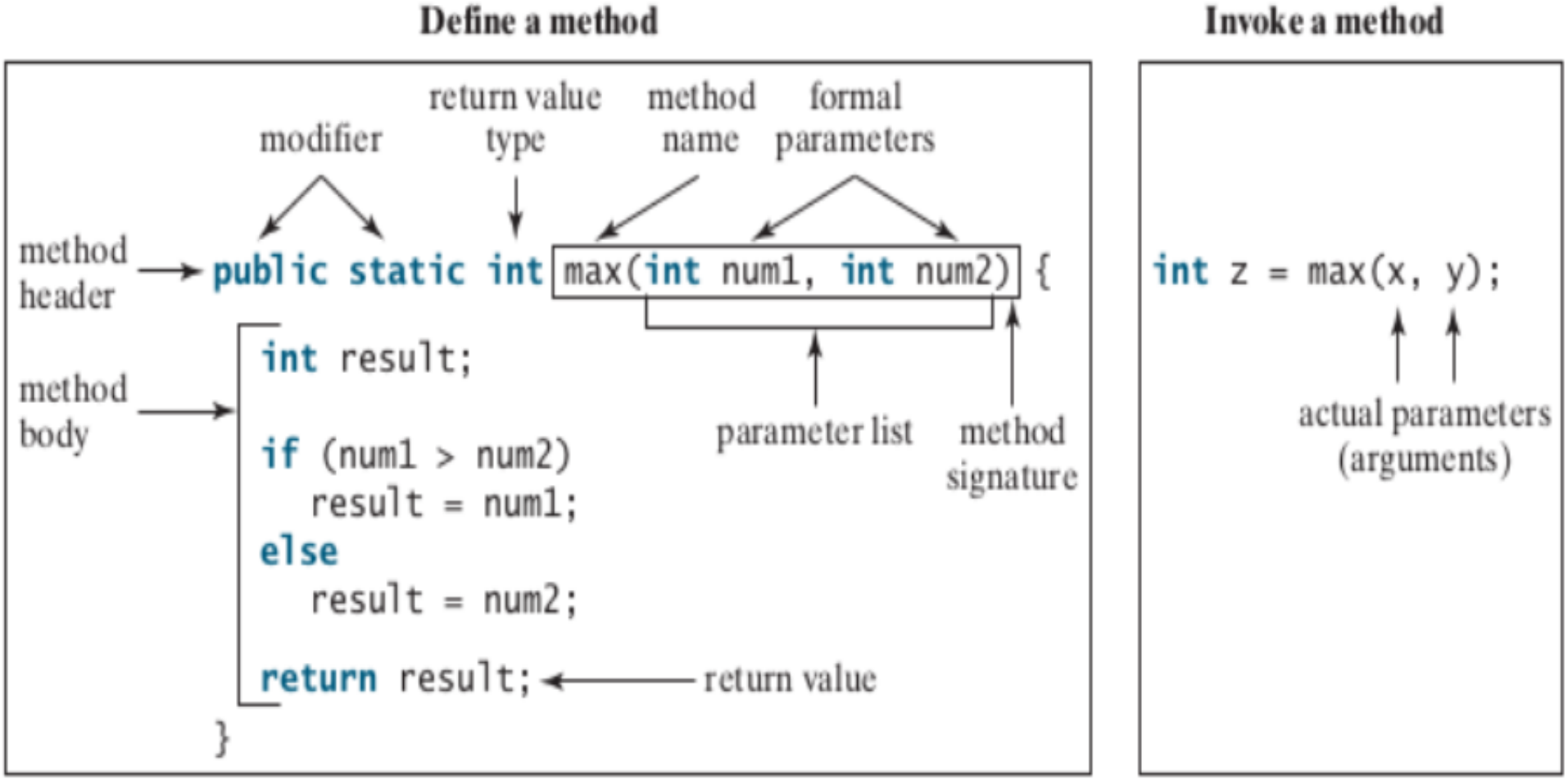


```
class Summation
{
        int x,y,z;
        void sum(int a, int b)
        {
                x=a;
                y=b;
                z=x+y;
                System.out.println (z);
        }
        public static void main(String args[])
        {
                Summation ob=new
Summation();
                Int m,n;
                m=20;
                n=30;
                ob.sum(m,n);
        }
}
```

Formal Parameter

Actual Parameter

Define and Invoke Method in Java



Parameter Passing in Java

Marwadi University

parameter passing in Java is always Pass-by-Value. However, the context changes depending upon whether we're dealing with Primitives or Objects:

- ✓ For Primitive types, parameters are pass-by-value
- ✓ For Object types, the object reference is pass-by-value

In case of primitives, the value is simply copied inside stack memory which is then passed to the callee method.

In case of non-primitives, a reference in stack memory points to the actual data which resides in the heap. When we pass an object, the reference in stack memory is copied and the new reference is passed to the method.

Check Example : Unit 3 → CallByValue.java , CallByReference.java

Pass By Value / Call by Value

Marwadi University

- "Call by value" in Java means that argument's value is copied and is passed to the parameter list of a method.
- That is, when we call a method with passing argument values to the parameter list, these argument values are copied into the small portion of memory and a copy of each value is passed to the parameters of the called method.
- When these values are used inside the method either for "read or write operations", we are actually using the copy of these values, not the original argument values which are unaffected by the operation inside the method.
- That is, the values of the parameters can be modified only inside the scope of the method but such modification inside the method doesn't affect the original passing argument.
- When the method returns, the parameters are gone and any changes to them are lost. This whole mechanism is called call by value or pass by value.

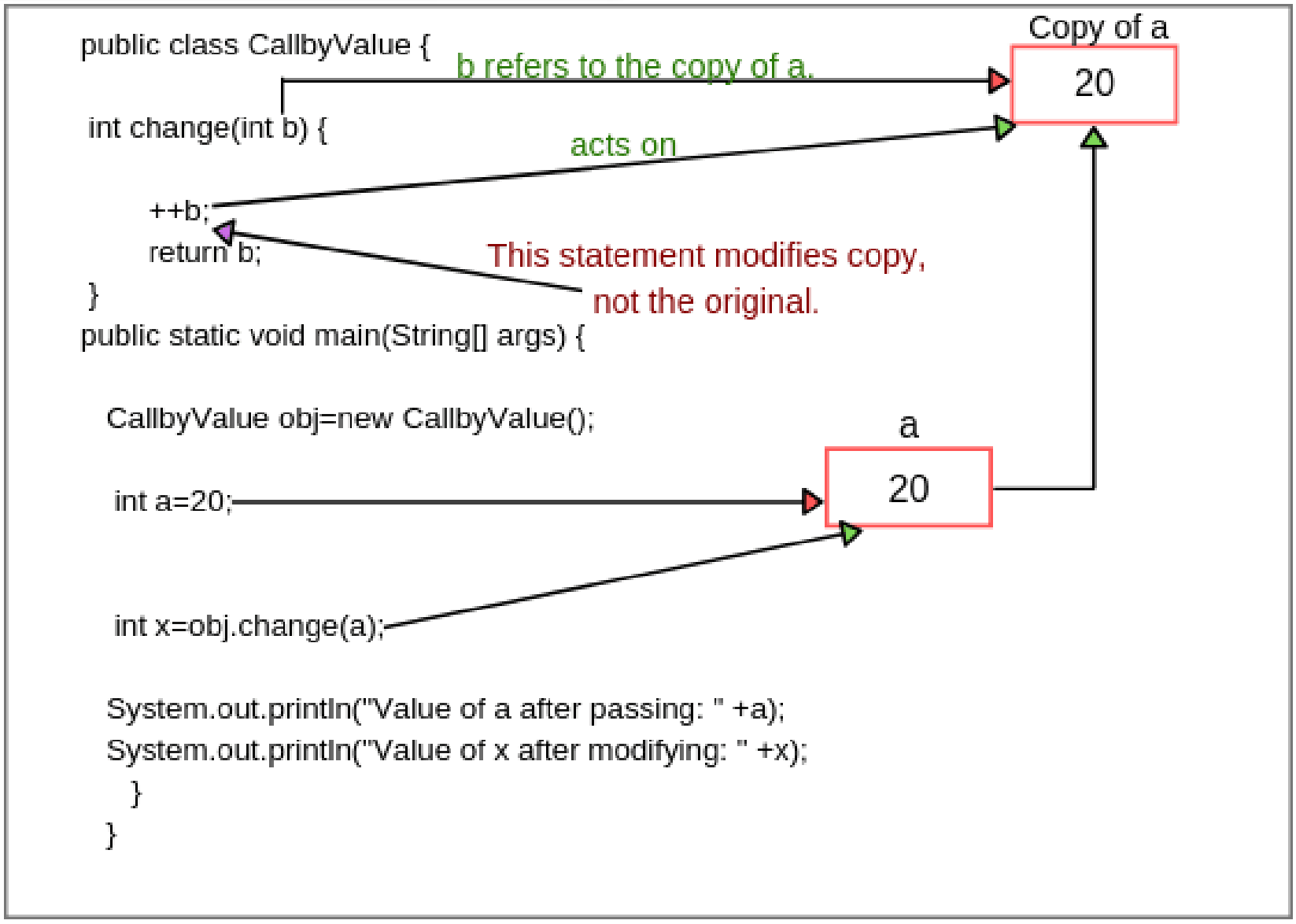
Check Example : Unit 3 → CallByValue.java

Pass By Value / Call by Value

Marwadi University

```
//CallByValue
public class Main
{
  int change(int b)
  {
    ++b; // Changes will be in the local variable only.
    return b;
  }
public static void main(String[] args)
{
// Create an object of class.
    Main obj = new Main();
     int a = 20;
     int x = obj.change(a);
     System.out.println("Value of a after passing: " +a);
     System.out.println("Value of x after modifying: " +x);
 }
}
```

Pass By Value / Call by Value



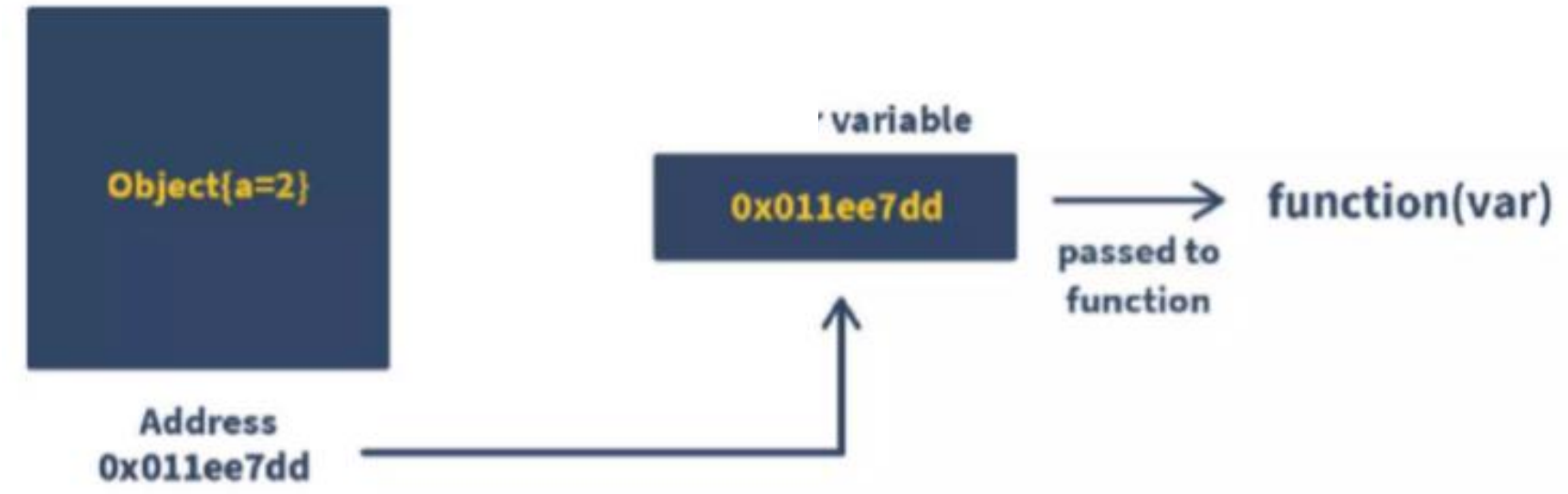
Call by reference(aliasing)

Marwadi University

- Call by Reference means calling a method with a parameter as a reference. Through this, the argument reference is passed to the parameter.
- In call by value, the modification done to the parameter passed does not reflect in the caller's scope while in the call by reference, the modification done to the parameter passed are persistent and changes are reflected in the caller's scope.

Check Example : Unit 3 → CallByReference.java

Call by reference(aliasing)



Call By Value vs Call By Reference

Marwadi University

```
public class Main{
//call-by-value
   int a = 10;
   void call(int a) {

           a = a+10;
   }
    public static void main(String[] args) {
       Main eg = new Main();
      S.o.p("Before call-by-value: " + eg.a);
       eg.call(50510);
      S.o.p("After call-by-value: " + eg.a);
   }
}
```

```
public class Main{
   int a = 10;
   void call(Main e) {
      e.a = e.a+10;
   }
   public static void main(String[] args) {

      Main e = new Main();
     S.o.p("Before call-by-reference: " + e.a);
     e.call(e);
     S.o.p("After call-by-reference: " + e.a);


   }
}
```

Variables in Java

Marwadi University

- There are three kinds of variables in Java –
- Local variables
- Instance variables
- Class/Static variables

- **What about Global Variable?**
- **Global variables are not technically allowed in Java.**
- Since Java is object-oriented, everything is part of a class.
- A **static** variable can be declared, which can be available to all instances of a class.

Local Variables

Marwadi University

- Local variables are declared in methods, constructors, or blocks.

```
public class Main{
  public void Age() {
    int age = 0; // Local variable
    age = age + 7;
    System.out.println("Age is : " + age);
  }
  public static void main(String args[]) {
    Main test = new Main();
    test.Age();
  } }
```

Instance Variables

- Instance variables are declared in a class, but outside a method, constructor or any block.

```
public class Main {
String name;
public Main (String empName) {
    name = empName;
  }
public void printEmp() {
    System.out.println("name  : " + name ); }
 public static void main(String args[]) {
    Main emp = new Main("abc");
   emp.printEmp();
 }}
```

Class/Static Variables

Marwadi University

- Class variables also known as static variables are declared with the static keyword in a class, but outside a method, constructor or a block.

```
class Main{
   //int global=0;
  static  int global=0;
   public static void main (String[] args) {
    //int global=0;
     System.out.println(global);
   }
}
```

Overloading methods

- In Java, two or more methods can have same name if they differ in parameters (different number of parameters, different types of parameters, or both). These methods are called overloaded methods and this feature is called method overloading. For example:

- void func() { ... }
- void func(int a { ... }
- float func(double a) { ... } float func(int a, float b) { ... }

- Here, the func() method is overloaded. These methods have the same name but accept different arguments.
- Notice that, the return type of these methods is not the same. Overloaded methods may or may not have different return types, but they must differ in parameters they accept.

Why method overloading ?

Marwadi University

- Suppose, you have to perform the addition of given numbers but there can be any number of arguments (let's say either 2 or 3 arguments for simplicity).

- In order to accomplish the task, you can create two methods :
  sum2num(int, int) and
- sum3num(int, int, int) for two and three parameters respectively.

- However, other programmers, as well as you in the future may get confused as the behavior of both methods are the same but they differ by name.

- The better way to accomplish this task is by overloading methods. And, depending upon the argument passed, one of the overloaded methods is called. This helps to increase the readability of the program.

Method overloading

Marwadi University

Two or more methods can have same name inside the same class if they accept different arguments. This feature is known as method overloading.

Method overloading occurs during compile time.

Method overloading is achieved by either:

- ✓ changing the number of arguments.
  (**Check Program:** Unit – 3 → MethodOverloading1.java)

- ✓ changing the datatype of arguments.
- ✓ (**Check Program:** Unit – 3 → MethodOverloading2.java)

Method overloading is not possible by changing the return type of methods.

Method overloading

```
//Overloading by changing the number of
parameters
class MethodOverloading {
   private static void display(int a){
      System.out.println("Arguments: " + a);
   }

   private static void display(int a, int b){
      System.out.println("Arguments: " + a + "
and " + b);
   }

   public static void main(String[] args) {
      display(1);
      display(1, 4);
   }
}
```

```
class MethodOverloading2 {

private static void display(int a){
      System.out.println("Got Integer data.");
   }

private static void display(String a){
      System.out.println("Got String object.");
   }

   public static void main(String[] args) {
      display(1);
      display("Hello");
   }
}
```

Can we overload java main() method?

Yes, by method overloading. You can have any number of main methods in a class by method overloading. **But JVM calls main() method which receives string array as arguments only**. Let's see the simple example:

```
class TestOverloading4{
public static void main(String[] args)
        {System.out.println("main with String[]");}
public static void main(String args)
        {System.out.println("main with String");}
public static void main()
        {System.out.println("main without args");}
}
```

Output:

main with String[]

Method Overloading

Marwadi University

Why Method Overloading is not possible by changing the return type of method only?

In java, method overloading is **not possible** by changing the return type of the method only because of ambiguity. Let's see how ambiguity may occur:

```
class Adder{
static int add(int a,int b){return a+b;}
static double add(int a,int b){return a+b;}
}
class TestOverloading3{
public static void main(String[] args){
System.out.println(Adder.add(11,11));//ambiguity
}}
```

OUTPUT :

Compile Time Error: method add(int,int) is already defined in class Adder

Method Overriding

Marwadi University

If subclass (child class) has the same method as declared in the parent class, it is known as method overriding in Java.

Method overriding is one of the way by which java achieve **Run Time Polymorphism**.

Rules for Java Method Overriding

- The method must have the same name as in the parent class
- The method must have the same parameter as in the parent class.
- There must be an IS-A relationship (inheritance).

Method Overriding

Marwadi University

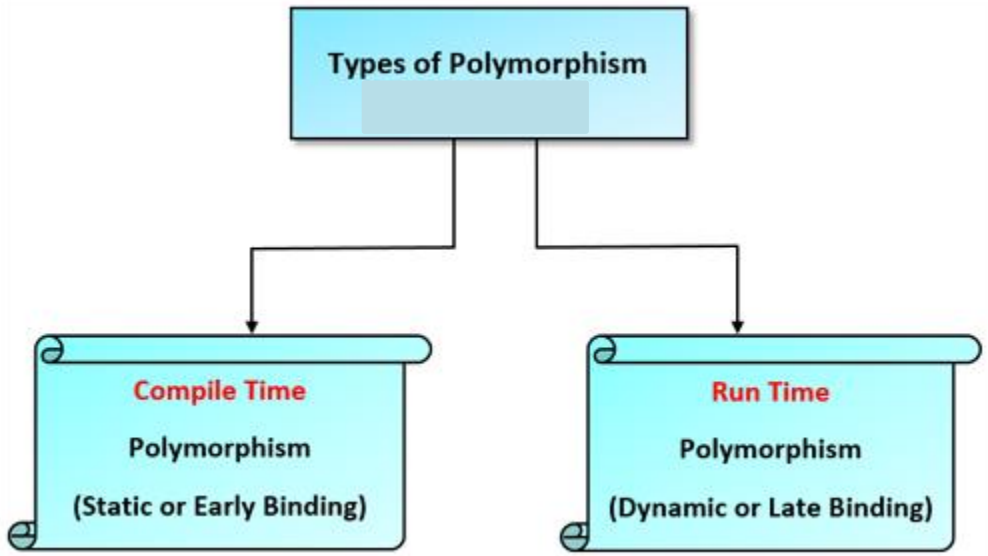
```
class parent{
void property() {
System.out.println("Car+Land+Home");  }

void marriage( ) {
System.out.println("abc"); }
}

class son extends parent {
Void marriage() {
System.out.println("XYZ"); }

public static void main(String args[]) {
son s = new son();
s.property(); s.marriage();  }
}
```

Polymorphism

- Polymorphism means "many forms", and it occurs when we have many classes that are related to each other by inheritance.

- Inheritance lets us inherit attributes and methods from another class.
- Polymorphism uses those methods to perform different tasks. This allows us to perform a single action in different ways.

Method Overloading & Operator Overloading occurs in compile time



Method Overriding occurs in Run time

Scope of Variables

- **The scope of a variable is the part of the program where the variable can be referenced.**

- A variable defined inside a method is referred to as a local variable. The scope of a local variable starts from its declaration and continues to the end of the block that contains the variable.

```
public class Main{
public static void main(String[] args) {
 for(int i=0; i<10; i++){
System.out.println(i);              }
 System.out.println(i); //Out of scope
}}
```

Array of Objects

- **When we require a single object to store in our program we do it with a variable of type Object. But when we deal with numerous objects, then it is preferred to use an Array of Objects.**
- Unlike the traditional array stores values like String, integer, Boolean, etc an Array of Objects stores objects that mean objects are stored as elements of an array.
- Note: that when we say Array of Objects it is not the object itself that is stored in the array but the reference of the object.
- Syntax:
- ClassName obj[ ]=new ClassName[array_length]; //declare and instantiate an array of objects

Array of Objects


```
class Main{

   public static void main(String args[])
   {
      // Creating an array of objects
      // declared with initial values
      Object[ ] aobj
         = { "Maruti", new Integer(2019), "Suzuki", new Integer(2019) };

     // Printing the values
      System.out.println(aobj[0]);
      System.out.println(aobj[1]);
      System.out.println(aobj[2]);
      System.out.println(aobj[3]);
   }}
```

Array of Objects

Marwadi University

```
public class ArrayExample {
   public static void main(String[] args) {
      Car cars[] = new Car[4];

      cars[0] = new Car("Toyota", 56600);
      cars[1] = new Car("Honda", 63500);
      cars[2] = new Car("Tata", 87400);
      cars[3] = new Car("Hyundai", 63000);

      for(Car car: cars)
         car.printDetails();
   }
}
```

```
class Car {
   public String name;
   public int miles;

   public Car(String name, int miles) {
      this.name = name;
      this.miles = miles;
   }

   public void printDetails() {

System.out.println(name+"..."+miles);
   }
}
```

Constructors

- In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created.
- **At the time of calling constructor, memory for the object is allocated in the memory.**
- It is a special type of method which is used to initialize the object.
- Every time an object is created using the new() keyword, at least one constructor is called.
- **It calls a default constructor if there is no constructor available** in the class. In such case, Java compiler provides a default constructor by default.

Constructors

Marwadi University

- Rules for creating Java constructor
- Constructor name must be the same as its class name
- A Constructor must have no explicit return type
- A Java constructor cannot be abstract, static, final, and synchronized

- Two types of constructor
- Default/No-argument constructor
- Parameterized Constructor

Default Constructor

Marwadi University

A constructor is called "Default Constructor" when it doesn't have any parameter.

Syntax of default constructor:

```
<class_name>( ) {
}
```

```
class Main{

//creating a default constructor
Main(){System.out.println("No arg constructor");}

public static void main(String args[]){
//calling a default constructor
Main b=new Main();
} }
```

Default Constructor

Marwadi University

What is the purpose of a default constructor?

The default constructor is used to provide the default values to the object like 0, null, etc., depending on the type.

```
class Main{
int id;
String name;

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

public static void main(String args[]){

Main s1=new Main();
s2.display();
}
}
```

Parameterized Constructor

Marwadi University

A constructor which has a specific number of parameters is called a parameterized constructor.

Why use the parameterized constructor?

The parameterized constructor is used to provide different values to distinct objects. However, you can provide the same values also.

Syntax:

```
classname (int a) // constructor with one parameter
{
x = a;
}
```

Parameterized Constructor

Marwadi University

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

 public static void main(String args[]){
Main m = new Main(1,”abc”);
m.display();
}
}
```

“this” Keyword

Marwadi University

- The this keyword refers to **the current object in a method or constructor.**

- The most common use of the this keyword is to eliminate the confusion between class attributes and parameters with the same name.

- this can also be used to:
- Invoke current class constructor
- Invoke current class method
- Return the current class object
- Pass an argument in the method call
- Pass an argument in the constructor call

“this” Keyword

Marwadi University

Using this for Ambiguity Variable Names

```
class MyClass {
    // instance variable
    int age;

    // parameter
    MyClass(int age){
        age = age;  //not allowed
    }
}
```

“this” Keyword

Marwadi University

```
class Main {

    int age;
    Main(int age){
        this.age = age;
    }

    public static void main(String[] args) {
        Main obj = new Main(8);
        System.out.println("obj.age = " + obj.age);
    }
}
```

Garbage Collection

Marwadi University

- Garbage Collection in Java is a process by which the programs perform memory management automatically.
- The Garbage Collector(GC) finds the unused objects and deletes them to reclaim the memory.
- In Java, dynamic memory allocation of objects is achieved using the new operator that uses some memory and the memory remains allocated until there are references for the use of the object.

- When there are no references to an object, it is assumed to be no longer needed, and the memory, occupied by the object can be reclaimed. There is no explicit need to destroy an object as Java handles the de-allocation automatically.

- The technique that accomplishes this is known as Garbage Collection.

Garbage Collection

Marwadi University

- Programs that do not de-allocate memory can eventually crash when there is no memory left in the system to allocate. These programs are said to have memory leaks.
- Note: All objects are created in **Heap** Section of memory.

- How can an object be unreferenced?
- **By nulling the reference**
  - Employee e=new Employee();
  - e=null;
- **By assigning a reference to another**
  - Employee e1=new Employee();
  - Employee e2=new Employee();
  - e1=e2
- **By anonymous object.**
  - **new** Employee();

Garbage Collection

- **finalize ( ) method**
- The finalize() method is invoked each time before the object is garbage collected. This method can be used to perform cleanup processing. This method is defined in Object class as:
- Syntax:
- protected void finalize( ){ }

- **gc( ) method**
- The gc( ) method is used to invoke the garbage collector to perform cleanup processing. The gc( ) is found in System and Runtime classes.
- Syntax:
- public static void gc( ){ }

Garbage Collection

Marwadi University

```
public class Main{

public void finalize()
{System.out.println("object is garbage collected");}

 public static void main(String args[]){
  Main s1=new Main();
  Main s2=new Main();

     System.out.println(s1.hashCode());
     System.out.println(s2.hashCode());
  s1=null;
  s2=null;
  System.gc();
  } }
```

Passing Object as Parameters and Returning Object

Marwadi University

Can we pass objects as an argument in Java?

Yes

When we pass primitive data types to method it will pass only values to function parameters so any change made in parameter will not affect the value of actual parameters.

Whereas Objects in java are reference variables, so for objects a value which is the reference to the object is passed. Hence the whole object is not passed but its referenced gets passed. All modification to the object in the method would modify the object in the Heap.

One of the most common uses of objects as parameters involves constructors. A constructor creates a new object initially the same as passed object. It is also used to initialize private members.

Passing Object as Parameters

```
class Add
{
    private int a,b;

    Add(Add A) //Passing Object as
                Parameter in Constructor
    {
        a=A.a;
        b=A.b;
    }
    Add(int x,int y)
    {
        a=x;
        b=y;
    }
    void sum()
    {
        int sum=a+b;
        System.out.println("Sum of a
and b :"+sum);
    } }

class Main
{
    public static void main(String arg[])
    {
        Add A=new Add(15,8);
        Add A1=new Add(A);
        A1.sum();    } }
```

Static Members

Marwadi University

- The static keyword in Java is used for **memory management** mainly.
- Static keyword can be used with class, variable, method and block.
- Static members belong to the class instead of a specific instance, this means if you make a member static, you can access it without object.

Static Method

Marwadi University

```
class Main
{

    static void myMethod()
   {
      System.out.println("myMethod");
   }

   public static void main(String[] args)
   {
         myMethod();
   }
}
```

Static Block

- These are a block of codes with a static keyword. In general, these are used to initialize the static members.
- **JVM executes static blocks before the main method at the time of class loading.**

```
public class MyClass {
  static{
    System.out.println("Hello this is a static block");
  }
  public static void main(String args[]){
    System.out.println("This is main method");
  }
}
```

Which block will be executed first?

static

Static Variable

Marwadi University

```
class Student{
   int rollno;
   String name;
   static String college =“MU";
Student(int r, String n){
   rollno = r;
   name = n;
   }
void display ()
{System.out.println(rollno+" "+name+" "+college);
}
}
```

```
public class Main{
 public static void main(String args[]){
 Student s1 = new Student(1, “Aarav");
 Student s2 = new Student(2, “Bhuvi");
 s1.display();
 s2.display();
 }
}
```

Static Class

Marwadi University

A class can be made static only if it is a nested class.

- Nested static class doesn't need reference of Outer class
- A static class cannot access non-static members of the Outer class

```
class Main{
  private static String str = "Hi";

static class MyNestedClass{
        public void disp() {
          System.out.println(str);
        } }
  public static void main(String args[])
  {
        Main.MyNestedClass obj = new Main.MyNestedClass();
        obj.disp();     } }
```

Static Restrictions

Marwadi University

There are two main restrictions for the static method.
They are:

1. The static method can not use non static data member or call non-static method directly.
2. this and super cannot be used in static context.

Why is the Java main method static?

- It is because the object is not required to call a static method.
- If it were a non-static method, JVM creates an object first then call main() method that will lead the problem of extra memory allocation.

Summary

Marwadi University

- Defining classes for objects
- Declaring objects
- New keyword
- Defining and calling methods in class
- Array of objects
- Constructors
- This keyword
- Garbage collection
- Finalize() method
- Passing object as parameters
- Returning object
- Static members

END OF UNIT - 3