

Chapter 5:

JAVA GUI Programming

Revision Tour -III

Informatics Practices

Class XII

By- Rajesh Kumar Mishra

PGT (Comp.Sc.)

KV No.1, AFS, Suratgarh

e-mail : rkmalld@gmail.com

What is method (function)?

□ Definition:

A Method or function is sequence of statement which are written to perform a specific job in the application.

In Object Oriented Programming, Method represents the behavior of the object.

A message can be thought as a call to an object's method.

Why Methods?

The following three advantages/reasons describes that why we use methods.

❑ To cope with complexity:

When programs become more complex and big in size, it is best technique to follow “Divide and conquer” i.e. a complex problem is broken in to smaller and easier task, so that we can make it manageable. Some times it is also called Modularization.

❑ Hiding Details:

Once a method is defined, it works like a Black-box and can be used when required, without concerning that “How it Works?”

❑ Reusability of code:

Once a method is implemented, it can be invoked or called from anywhere in the program when needed i.e. Method can be reused. Even a packaged method may be used in multiple applications. This saves our time and effort.

Most of the method like `Math.sqrt()` is available as ready to use which can be used anywhere in the application.

How to define a Method

A method must be defined before its use. The method always exist in a class. A Java Program must contain a main() method from where program execution starts. The general form of defining method is as-

```
[Access specifier] <return_type> <method_name> (<parameter(s)>)
{ ..... ;
  body of the method i.e. statement (s);
}
```

❑ Access Specifier:

It specified the access type and may be public or protected or private.

❑ Return Type:

Specifies the return data type like int, float etc. Void is used when nothing is to be returned.

❑ Method Name:

Specified the name of method and must be a valid Java identifier.

❑ Parameters List:

It is list of variable(s), also called Formal Parameter or Argument, which are used to catch the values when method is invoked. Also a method may have no parameters.

Calling a User defined methods/functions

- ❑ A method can be called by its name and parameters value (if any) in any other methods or Event method of JButtons.
- ❑ A method can be defined and called as –

// calling method/ function

```
private void jButton1ActionPerformed(.....)
{
    int a,b,c,d;
    a=4;
    b=6;
    c=sum(a,b); // function call
    d=sum(3,5); // again function call
    jTextField1.setText(""+c);
    System.out.println(""+c);
    System.out.println(""+d);
}
```

Actual
parameters

// called Method

```
int Sum (int x, int y)
{
    int z= x+y;
    return(z);
}
```

Formal
parameters

Signature

```
int Sum (int x, int y)
```

Prototype

Note : The number of parameters and their data type must be matched during a method call. Formal Parameters must be variable.

Passing Arguments to Methods

You can pass arguments (Actual parameters) to method (Formal Parameters) using valid data types like int, float, byte, char, double, boolean etc. or Reference data type like Object and Arrays.

A method can called in two ways –

❑ Call by Value:

In this method, the **values** of Actual parameters are copied to Formal parameters, so any changes made with Formal parameters in Method's body, will not reflected back in the calling function.

The original value of Actual parameters is unchanged because the changes are made on copied value.

❑ Call by Reference:

In Reference method, the changes made on the formal parameters are reflected back in the Actual parameters of calling function because instead of values, a **Reference** (Address of Memory location) is passed.

In general, all primitive data types are passed by Value and all Reference types (Object, Array) are passed by Reference.

Example : Call by Value method

// calling method/ function

```
private void jButton1ActionPerformed(.....)
{
    int a,b,c,d;
    a=4;
    b=6;
    System.out.println ("Before calling the method");
    System.out.println(""+a);    // value is 4
    System.out.println(""+b);    // value is 6
    change (a,b);    // function call
    System.out.println ("After calling the method");
    System.out.println(""+a);    // value is still 4
    System.out.println(""+b);    // value is still 6
}
```

// called Method

```
private void change (int x, int y)
{
    int t= x;
    x=y;
    y=t;
}
```

Example : Call by Reference method

```
public class test                // Declare a class
{ int x=4;
  int y=6; }
private void jButton1ActionPerformed(....)
{ test myobj = new test();      // create an object
  System.out.println ("Before calling the method");
  System.out.println (" "+myobj.x); // value is 4
  System.out.println (" "+myobj.y); // value is 6
  change (myobj); // function call
  System.out.println ("After calling the method");
  System.out.println (" "+myobj.x); // value is 6
  System.out.println (" "+myobj.y); // value is 4
}
```

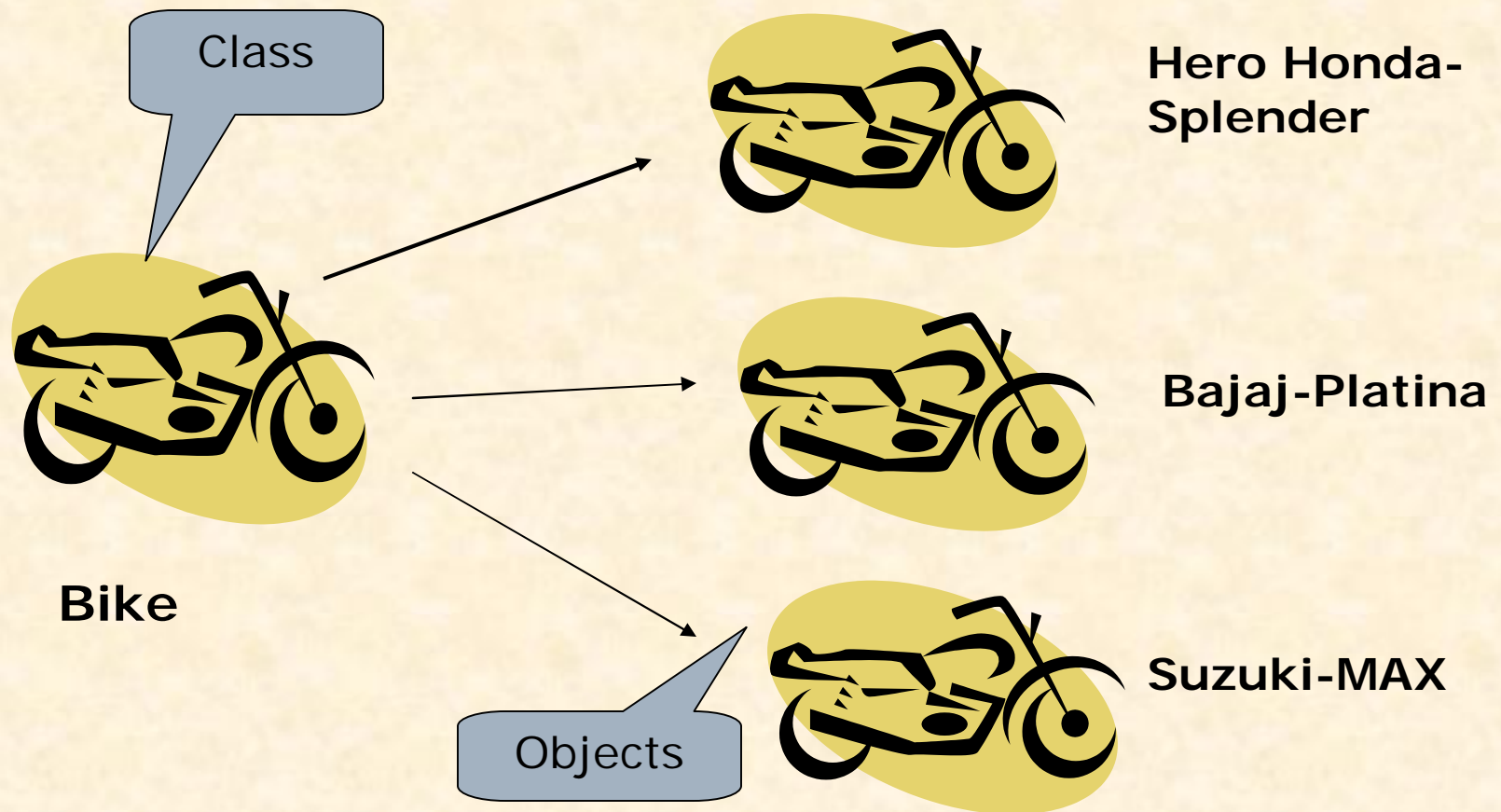
```
// called Method
private void change (test t)
{ int a= t.x;
  t.x=t.y;
  t.y=a;
}
```

Class & Object

JAVA is a pure Object Oriented Programming language, since each program must contain at least one class definition. OOP is more real programming approach than other approaches.

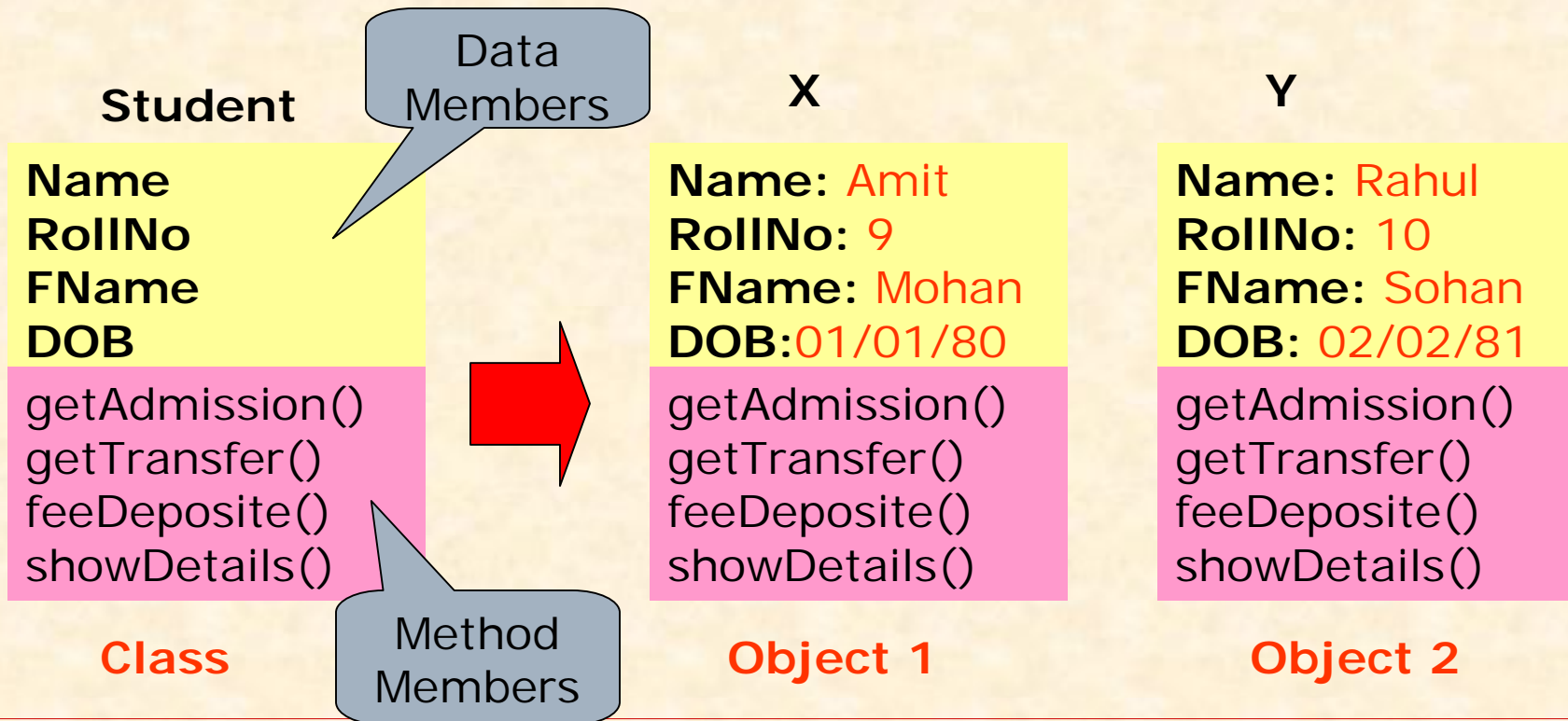
- ❑ The basic unit of OOP is the Class. It can be described as a blue print of Objects. In other words, an Object is an instance of a class.
 - ❑ A JAVA program may have various class definitions.
 - ❑ An Object is an entity having a unique Identity, characteristics (Properties) and Behavior (Methods)
-

Class and Objects



How to define a Class

- ❖ A class is composed by a set of Attributes (Properties) and Behavior.
- ❖ Properties are represented by **Data Members** and Behavior is simulated by **Method Members**.



How to declare a Class in Java

```
public class Student  
{ String Name;  
  int RollNo;  
  String FName;  
  String DOB;
```

Data Members
(Properties)

```
void getAdmission()  
{ .....  
  .....  
}  
void getTransfer()  
{ .....  
  .....  
}  
void feeDeposit()  
{ .....  
  .....  
}  
}
```

Method Members
(Behavior)

In Java a Class is declared/defined by using **class** keyword followed by a class name.

Declaration of Data Members

Data Members in a class may be categorized into two types-

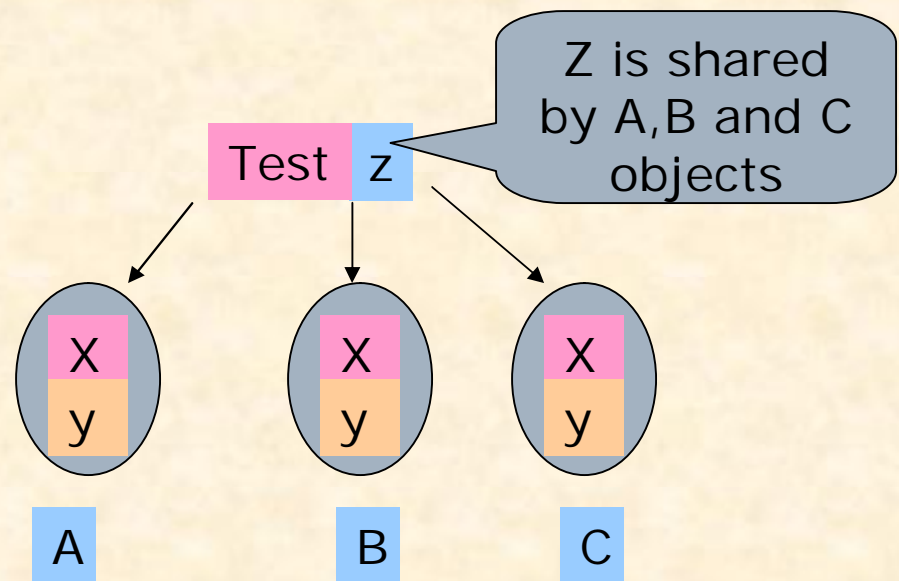
- ❖ **Instance Variable-**

These data members are created for every object of the class i.e. replicated with objects.

- ❖ **Class variable (static)-**

These data members that are declared once for each class and all objects share these members. Only a single copy is maintained in the memory. These are declared with **static** keyword.

```
public class test
{ int x;
  int y;
  static int z;
  .....
}
```



Declaration of Method Members

Like Data Members, Method members are also categorized into two types-

- ❖ **Instance Method-**

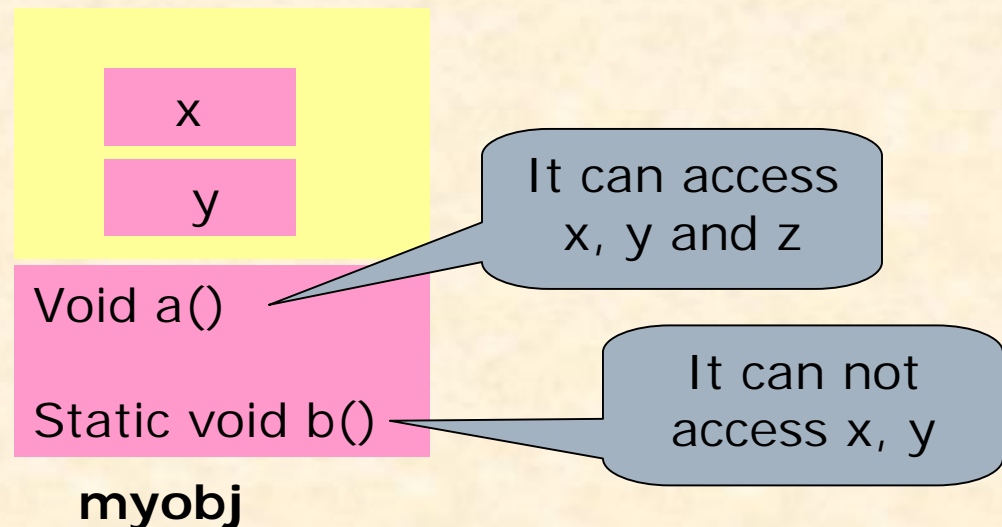
These methods may operate the object's instance variables and can access to the class (static) variables.

- ❖ **Class Methods (static)-**

These methods can not access the instance variables declared within class directly. (Indirect methods to access them is call by reference).

A class method is invoked by its name (within own class) or by <class name>. <method name> outside its class. E.g. student.getAdmission()

```
public class test  
{ int x;  
  int y;  
  static int z;  
  void a()  
  { .....  
  }  
  static void b()  
  { .....  
  }  
}
```



How to creating Objects

A class defined earlier does not occupy any space, it defines a blue-print for the objects. When object is instantiated then it occupies space in the memory. An object is created by the following steps-

❑ **Declaration:**


Object is declared like an ordinary variables using class name as data type.

❑ **Instantiation:**

An object is instantiated by using new keyword.

❑ **Initialization:**

The new operator followed by a call to the constructor method (method having same name as class) is used to initialize and object.



```
student x;           // x object is declared  
x= new student();    // x is instantiated and initialized.
```

```
Student x= new student(); // all in a single step (alternate)
```

Creating Objects- Example

// define a class

```
public class city
{ String name;
  long population;
  void display()
  { System.out.println("City Name: "+name);
    System.out.println("Population: "+population);
  }
}
```

// create an object and access it

```
public class myprog
{ public static void main (String arg[])
  { city x= new city();
    x.name="Jaipur";
    x.population = 100000;
    System.out.println("City Details" + '\n');
    x.display();
  }
}
```

Use of Constructor Method

A Constructor is a member method of a class, used to initialize an Object, when it is created (instantiated).

- ❑ A Constructor must have the same name as the class name and provides initial values to its data members.
- ❑ A constructor have no return type not even void.
- ❑ JAVA automatically creates a constructor method, if it is not defined with default values.

```
public class student  
{ int rollNo;  
  float marks;  
  public student()  // constructor method  
  { rollNo=0;  
    marks= 0.0;  
  }  
  ..... // other method members  
}
```

Public or Private Constructor Method

A Constructor may be Public or Private.

- ❑ A Private constructor is not available to non-member methods i.e. it can be called within member method only.
- ❑ Public constructor is available to member and non-member methods.
- ❑ Generally, a constructor is defined public, so that its object can be created any where in the program.

```
class x
{ int i, j;
  private x()
  { i= 10;
    j= 20;
  }
  void getValue()
  { .....
    .....
  }
  void check()
  { x obj1= new x();
    .....
  }
}
```

Available to
member method

```
class y
{ .....;
  .....;
  void test()
  { x obj2= new x();
    .....
    .....
  }
}
```

Invalid..
(Not Available to
non-member
method)

Difference between Constructors and Methods

Though Constructors are member methods of the class like other methods, but they are different from other method members

- ❑ Constructor creates (initializes) an Object where a method is a group of statements which are packaged to perform a specific job.
 - ❑ Constructor has no return type, even void also. Whereas method may have any return type including void.
 - ❑ The Constructor has the same name as Class, but method may have any name except Class name.
 - ❑ It is called at the time of object creation, but a method can be called any time when required.
-

Types of Constructor Method

Constructor is a method having the same name as the class.

Like other methods it may be -

- ❑ **Parameterized** : constructor with parameters.
- ❑ **Non-Parameterized**: Constructor without parameters.

// Ex. Non-parameterized

```
class A
{ int i;
  float j;
  public A()
  { i= 10;
    j= 20.5;
  }
  .....
}
class B
{ void test()
  { A obj1= new A();
    .....
  }
}
```

//Ex. Parameterized constructor

```
class A
{ int i;
  float j;
  public A(int x, float y)
  { i= x;
    j= y;
  }
  .....
}
class B
{ void test()
  { A obj1= new A(10,20.5);
    .....
  }
}
```

Using Class as composite data type

Since a class may have various data members of primitive data types like int, float, long etc. In general class may be assumed as a bundle of primitive data types to make a user-defined composite data type.

```
// use of class as composite data type
class date
{ byte dd, mm, yy;
  public date( byte d, byte m, byte y)
  { dd= d;
    mm= m;
    yy= y;
  }
  void display()
  {system.out.println(""+dd+"/ "+mm+"/ "+yy);
  }
};
date DOB = new date(13,2,1990);
```

Primitive data type v/s Composite Data type

Primitive	Composite
Built-in data type offered by compiler.	User- defined created as per need.
Fixed sized i.e. fixed memory allocation.	Allocation size depends on its constituent data type.
May be used any where in the program.	Availability depends on its scope.

Referencing Object Members

Once an object has been created, you may require to do the following -

- ❑ Manipulate or inspect its data members (Change its state)
- ❑ Perform an action (by invoking its method members)

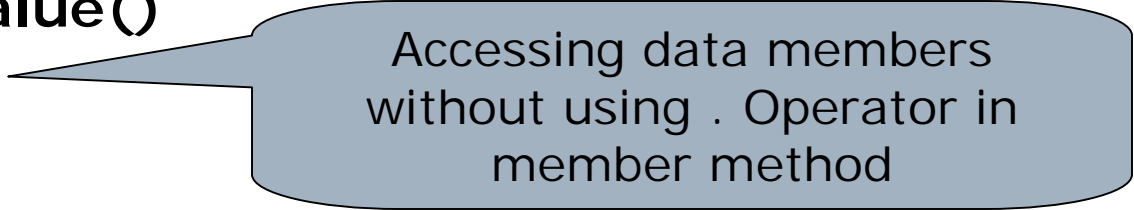
In General, we can refer an object's data member by qualifying (using . Operator) like < Object name>.<variable name>

e.g. **System.out.println("Name:- " + x.name);**

However within member method you can refer it without using (.)

Class demo

```
{ int a, b;  
  public setvalue()  
  { a=5;  
    b=10;  
  }  
}
```



Accessing data members
without using . Operator in
member method

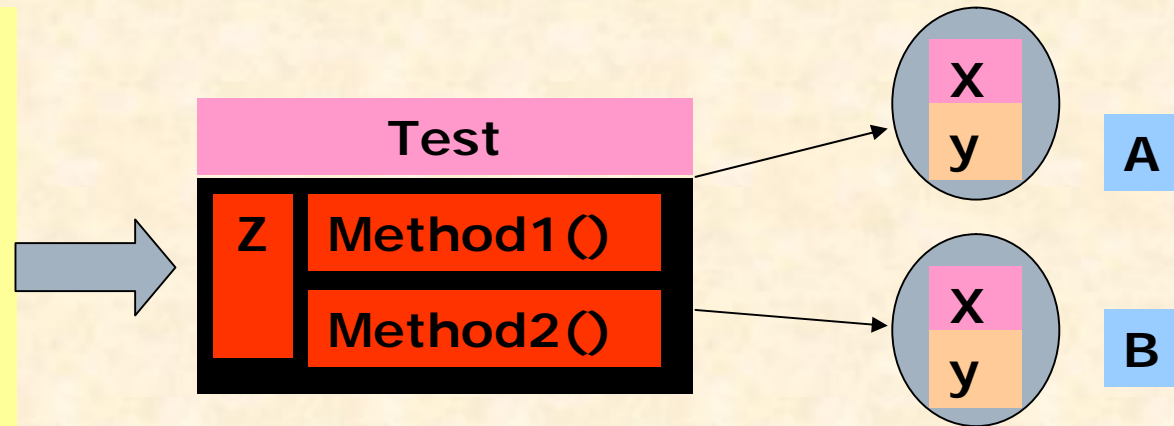
- ❑ You can also refer object's member methods by qualified name like -

<object name>.<method name ([parameters])>;

Using 'this' keyword

As you are aware that static data and method members of a class is kept in the memory in a single copy only. All the object are created by their instance variables but shares the class variables (static) and member methods.

```
public class test
{ int x, y;
  static int z;
  static method1()
  { ..... }
  static method2()
  { ..... }
}
```



Suppose method2() is changes X data member, then big question arises that **which object's x variable will be changed?**

This is resolved by using 'this' keyword.

The keyword 'this' refers to currently calling object. Instead of using object name, you may use 'this' keyword to refer current object.

Ex. `this.method2()`

Scope of a variable

- ❑ In Java, a variable can be declared anywhere in the program but before using them.
- ❑ The area of program within which a variable is accessible, known as its scope.
- ❑ A variable can be accessed within the block where it is declared.

