

# Learning Java - A Foundational Journey

## Session: 4

### Classes, Objects, and Methods





- ◆ Explain process of creation of classes in Java
- ◆ Explain instantiation of objects in Java
- ◆ Explain purpose of instance variables and instance methods
- ◆ Explain constructors and methods
- ◆ Explain process of creation and invocation of methods
- ◆ Explain passing and returning values from methods
- ◆ Explain variable argument methods
- ◆ Describe use of Javadoc to lookup methods
- ◆ Explain memory management in Java
- ◆ Explain object initializers
- ◆ Describe access specifiers and the types of access specifiers
- ◆ Explain concept of method overloading
- ◆ Explain the use of this keyword



## ◆ Class in Java:

- ◆ Is the prime unit of execution for object-oriented programming in Java.
- ◆ Is a logical structure that defines the shape and nature of an object.
- ◆ Is defined as a new data type that is used to create objects of its type.
- ◆ Defines attributes referred to as fields that represents the state of an object.
- ◆ The initialization of objects is done using constructors and the behavior of the objects is defined using methods.

For Aptech Centre Use Only



## Conventions to be followed while naming a class

- Class declaration should begin with the keyword `class` followed by the name of the class.
- Class name should be a noun.
- Class name can be in mixed case, with the first letter of each internal word capitalized.
- Class name should be simple, descriptive, and meaningful.
- Class name cannot be Java keywords.
- Class name cannot begin with a digit. However, they can begin with a dollar (\$) symbol or an underscore character.

# Declaring and Creating an Object



- ◆ An object is created using the `new` operator.
- ◆ On encountering the `new` operator:
  - ◆ JVM allocates memory for the object.
  - ◆ Returns a reference or memory address of the allocated object.
  - ◆ The reference or memory address is then stored in a variable called as reference variable.

## Syntax

```
<class_name> <object_name> = new <class_name> ();
```

where,

`new`: Is an operator that allocates the memory for an object at runtime.

`object_name`: Is the variable that stores the reference of the object.

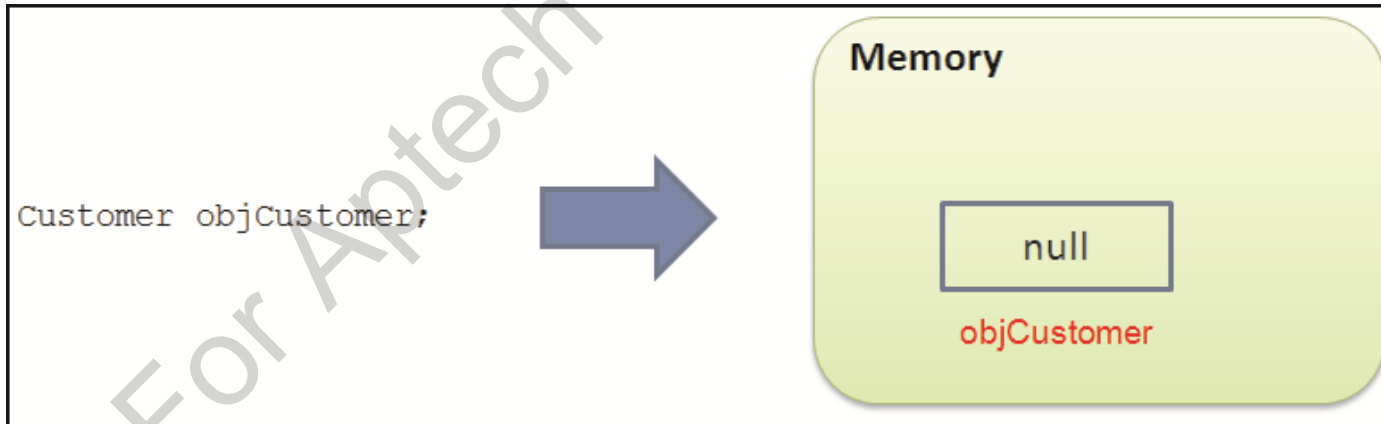
# Creation of an Object: Two Stage Process 1-2



- ◆ Alternatively, an object can be created using two steps that are as follows:
  - ◆ Declaration of an object reference.
  - ◆ Dynamic memory allocation of an object.
- ◆ **Declaration of an object reference:**
  - ◆ The syntax for declaring the object reference is as follows:

## Syntax

```
<class_name> <object_name>;
```

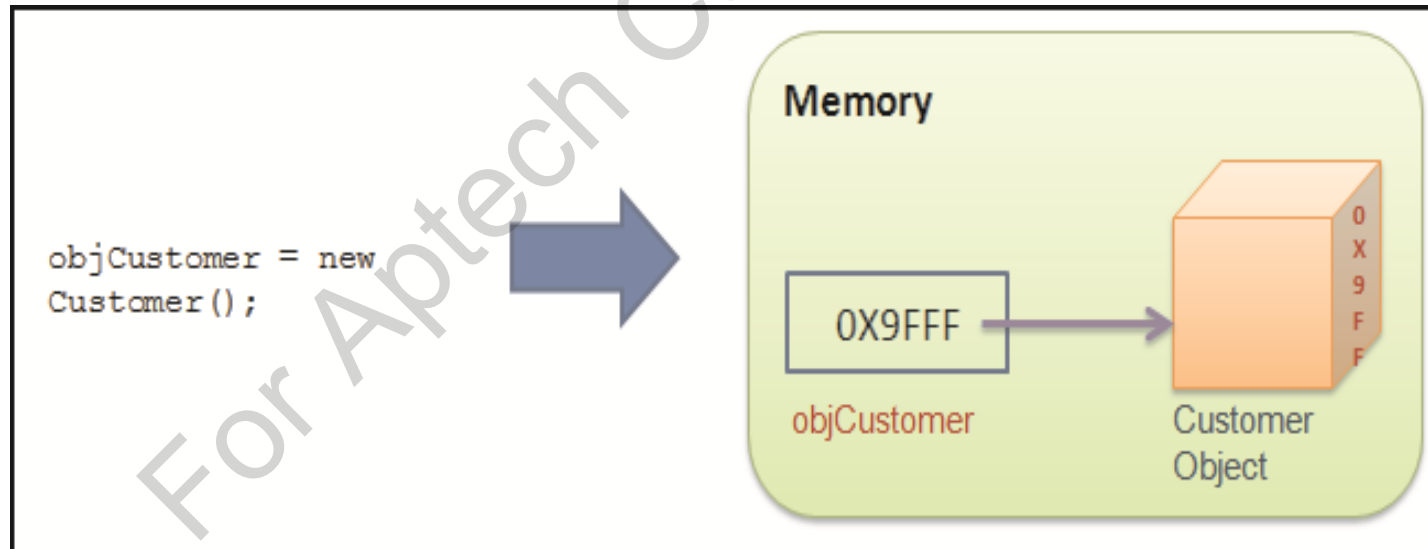




- ◆ **Dynamic memory allocation of an object:**

- ◆ The object should be initialized using the new operator which dynamically allocates memory for an object.
- ◆ For example, the statement, **objCustomer = new Customer () ;** allocates memory for the object and memory address of the allocated object is stored in the variable **objCustomer**.

- ◆ Following figure shows the creation of object in the memory and storing of its reference in the variable, **objCustomer**:





- ◆ The members of a class are fields and methods.

## Fields

- Define the state of an object created from the class.
- Referred to as instance variables.

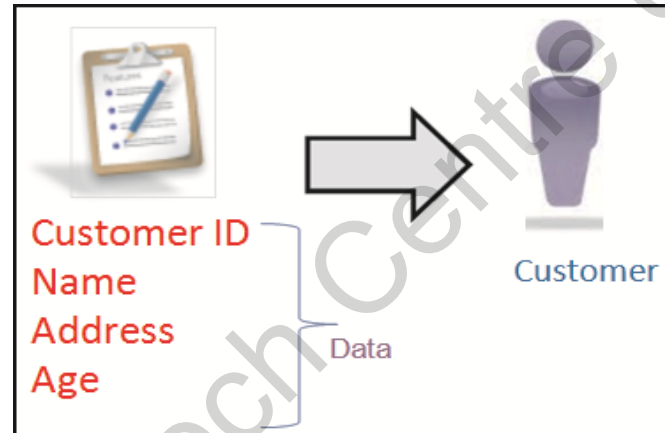
## Methods

- Implement the behavior of the objects.
- Referred to as instance methods.





- ◆ They are called instance variables because each instance of the class, that is, object of that class will have its own copy of the **instance variables**.
- ◆ Following figure shows a **Customer** object with its data requirement:

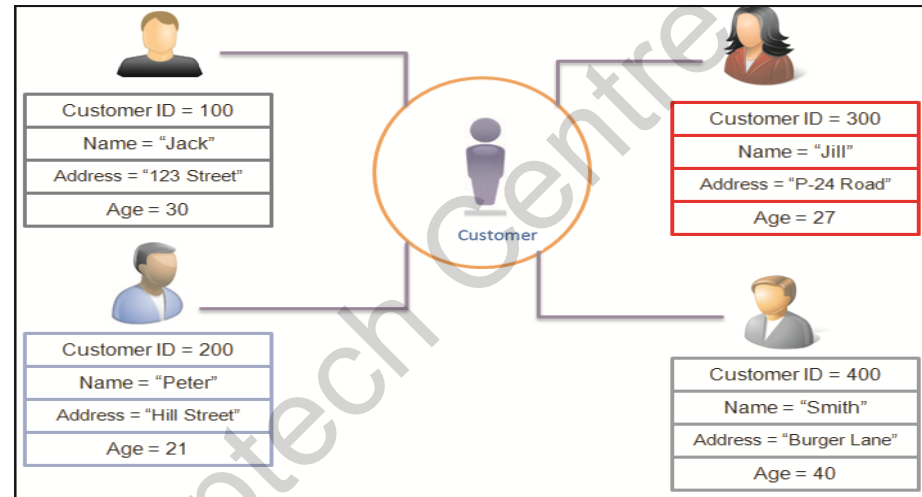


The identified data requirements for a bank customer includes: Customer ID, Name, Address, and Age.

To map these data requirements in a **Customer** class, **instance variables** are declared.



- ◆ Each instance created from the **Customer** class will have its own copy of the instance variables.
- ◆ Following figure shows various instances of the class with their own copy of instance variables:

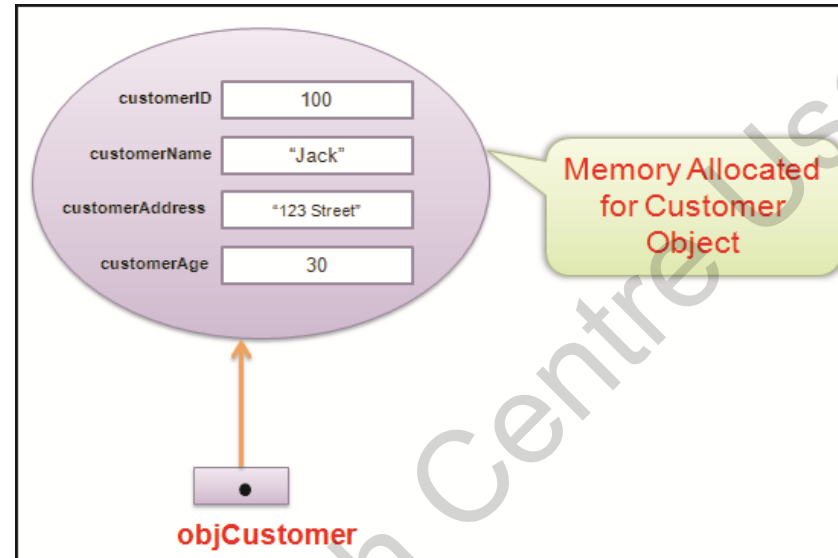


## Syntax

```
[access_modifier] data_type instanceVariableName;
```



- ◆ Following figure shows the allocation of **Customer** object in the memory:



- ◆ Following figure shows the output of the code:

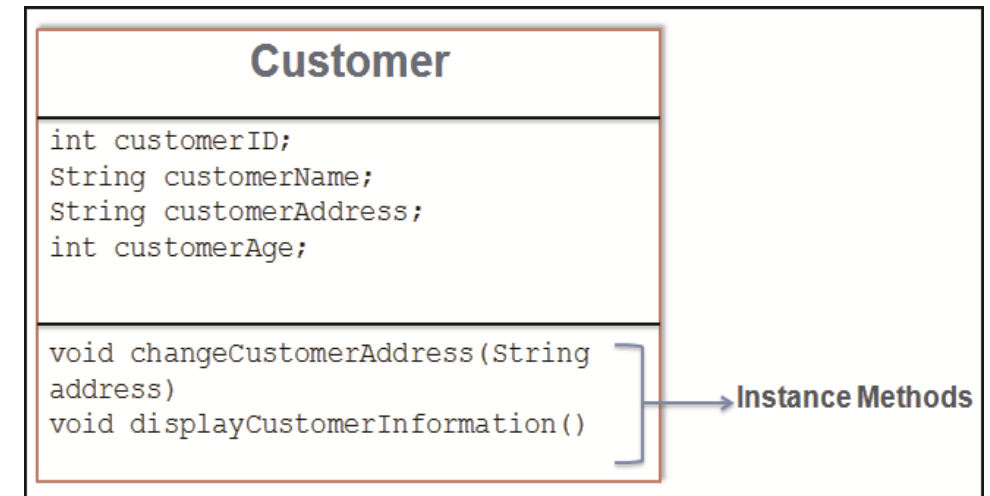
```
run:
Customer Identification Number: 100
Customer Name: John
Customer Address: 123 Street
Customer Age: 30
BUILD SUCCESSFUL (total time: 1 second)
```



- ◆ They are functions declared in a class.
- ◆ They implement the behavior of an object.
- ◆ They are used to perform operations on the instance variables.
- ◆ For example: the class **Car** can have a method **Brake ()** that represents the 'Apply Brake' action.
  - ◆ To perform the action, the method `Brake ()` will have to be invoked by an object of class **Car**.

## Conventions to be followed while naming a method are as follows:

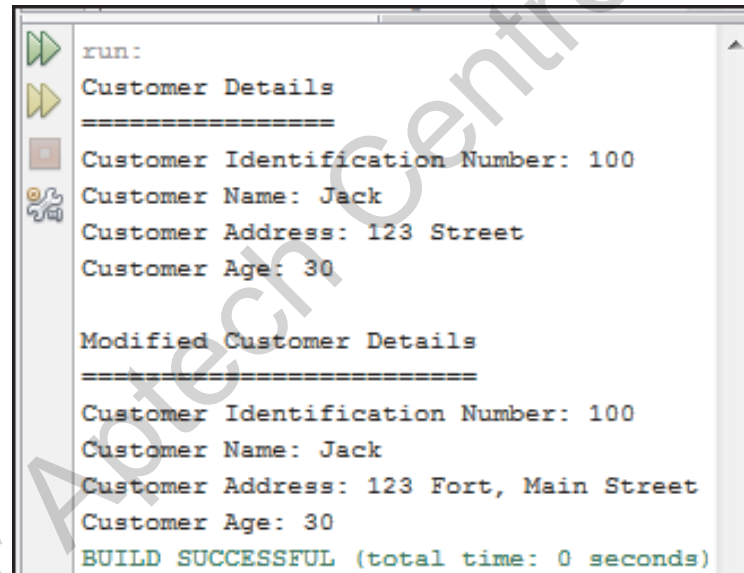
- Cannot be a Java keyword.
- Cannot contain spaces.
- Cannot begin with a digit.
- Can begin with a letter, underscore, or a '\$' symbol.
- Should be a verb in lowercase.
- Should be descriptive and meaningful.
- Should be a multi-word name that begins with a verb in lowercase, followed by adjectives, nouns, and so forth.



# Invoking Methods



- ◆ To invoke a method, the object name is followed by the dot operator (.) and the method name.
- ◆ A method is always invoked from another method.
- ◆ The method which invokes a method is referred to as the **calling method**.
- ◆ The invoked method is referred to as the **called method**.

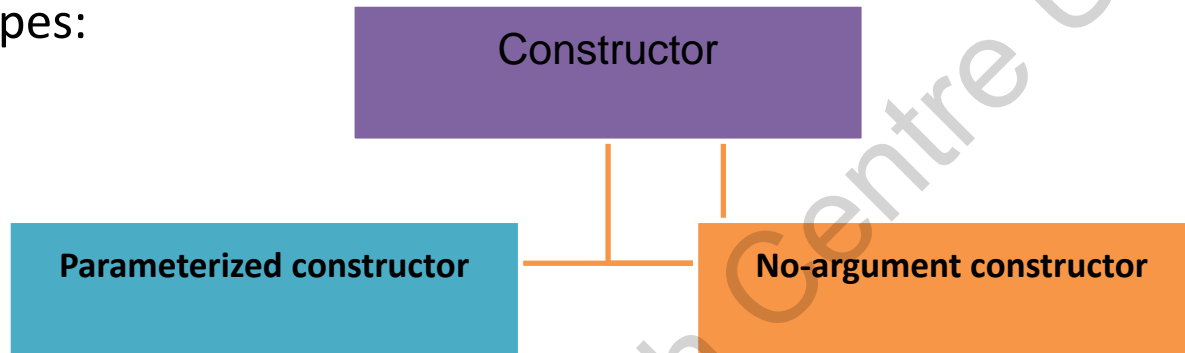
A screenshot of a Python IDE's output window. The window has a vertical toolbar on the left with icons for running (green play button), stepping through (yellow play button), and other debugging actions. The main area displays the output of a program. It starts with 'run:' followed by 'Customer Details' and a separator line. Then it shows customer information: 'Customer Identification Number: 100', 'Customer Name: Jack', 'Customer Address: 123 Street', and 'Customer Age: 30'. This is followed by 'Modified Customer Details' and another separator line. The modified information is: 'Customer Identification Number: 100', 'Customer Name: Jack', 'Customer Address: 123 Fort, Main Street', and 'Customer Age: 30'. The output ends with 'BUILD SUCCESSFUL (total time: 0 seconds)' in green text.

```
run:
Customer Details
=====
Customer Identification Number: 100
Customer Name: Jack
Customer Address: 123 Street
Customer Age: 30

Modified Customer Details
=====
Customer Identification Number: 100
Customer Name: Jack
Customer Address: 123 Fort, Main Street
Customer Age: 30
BUILD SUCCESSFUL (total time: 0 seconds)
```



- ◆ It is a method having the same name as that of the class.
- ◆ It initializes the variables of a class or perform startup operations only once when the object of the class is instantiated.
- ◆ It can accept parameters and do not have return types.
- ◆ It is of two types:



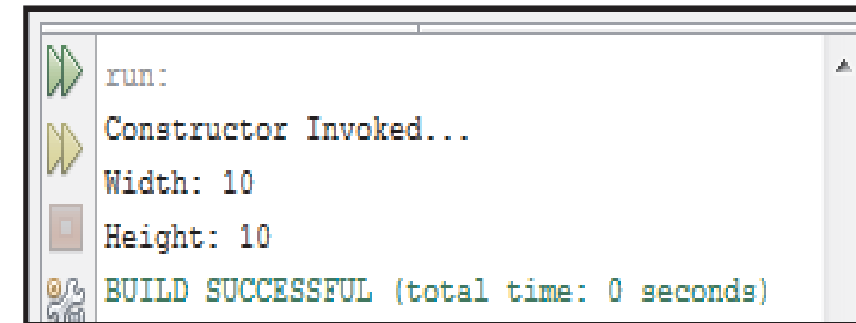
```
class <ClassName>
{
    <ClassName> () ← Constructor
    {
        // Initialization code
    }
}
```



- ◆ The constructor is invoked immediately during the object creation which means:
  - ◆ Once the `new` operator is encountered, memory is allocated for the object.
  - ◆ Constructor method is invoked by the JVM to initialize the object.
- ◆ Following figure shows the use of `new` operator to understand the constructor invocation:

```
<class_name> <object_name> = new <class_name>();
```

The parenthesis after the class name indicates the invocation of the constructor.





## Default Constructor:

- ◆ Created for the classes where explicit constructors are not defined.
- ◆ Initializes instance variables of newly created object to their default values.

Data Type	Default Value
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0
char	'\u0000'
boolean	False
String (any object)	Null

A screenshot of a Java IDE's console window. The window has a title bar and a toolbar with icons for running, stepping through, and debugging. The text in the console is as follows:

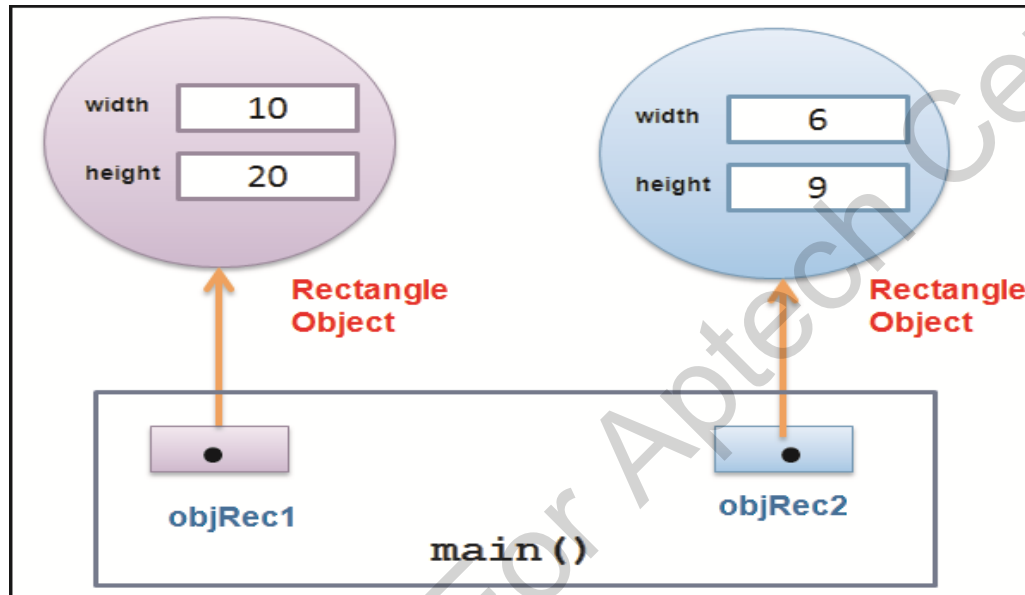
```
run:
Employee Details
=====
Employee Name: null
Employee Age: 0
Employee Salary: 0.0
Employee MaritalStatus:false
BUILD SUCCESSFUL (total time: 2 seconds)
```



# Parameterized Constructor



- ◆ The parameterized constructor contains a list of parameters that initializes instance variables of an object.
- ◆ The value for the parameters is passed during the object creation.
- ◆ Each object contains its own copy of instance variables that are initialized through constructor:



```
run:
Parameterized Constructor Invoked...
Parameterized Constructor Invoked...

Rectangle1 Details
=====
Width: 10
Width: 20

Rectangle2 Details
=====
Width: 6
Width: 9
BUILD SUCCESSFUL (total time: 1 second)
```



- ◆ The memory comprises two components namely, stack and heap.

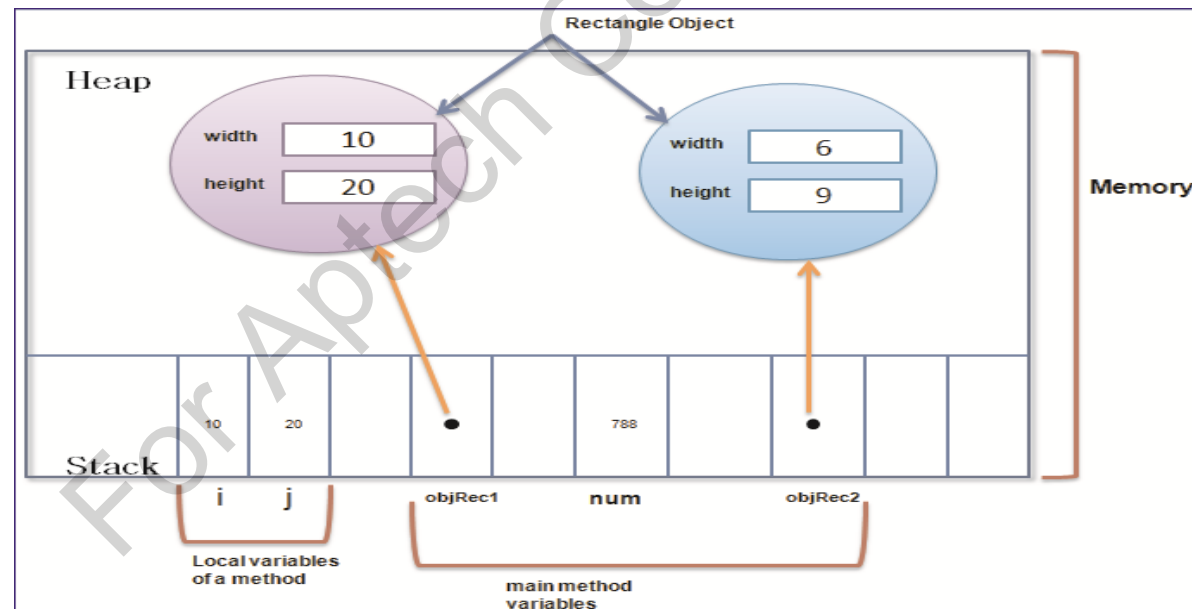
## Stack

- It is an area in the memory which stores object references and method information.
- It stores parameters of a method and local variables.

## Heap

- It is area of memory deals with dynamic memory allocations.
- In Java, objects are allocated physical memory space on the heap at runtime, that is, whenever JVM executes the new operator.

Following figure shows the memory allocation for objects in stack and heap for **Rectangle** object:

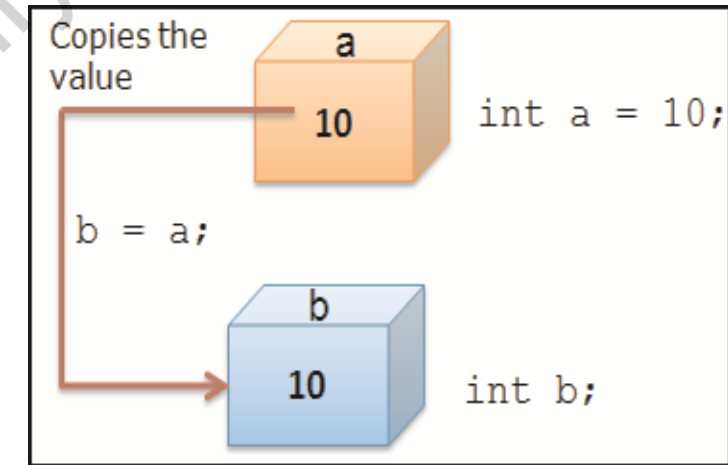


# Assigning Object References



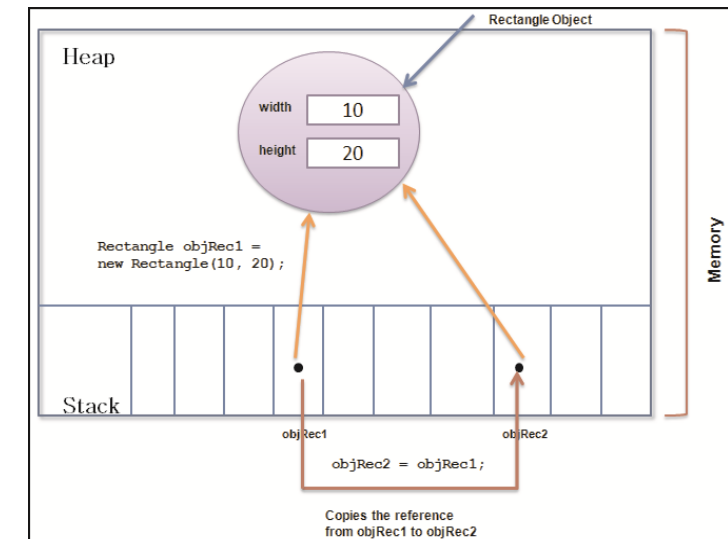
## ◆ Working with primitive data types:

- ◆ The value of one variable can be assigned to another variable using the assignment operator.
- ◆ For example, `int a = 10; int b = a;` copies the value from variable **a** and stores it in the variable **b**.
- ◆ Figure shows assigning of a value from one variable to another.



## ◆ Working with object references:

- ◆ Similar to primitive data types, the value stored in an object reference variable can be copied into another reference variable.
- ◆ Both the reference variables must be of same type, that is, both the references must belong to the same class.





- ◆ In OOP languages, the concept of hiding implementation details of an object is achieved by applying the concept of encapsulation.

## Encapsulation

- It is a mechanism which binds code and data together in a class.
- Its main purpose is to achieve data hiding within a class which means:
  - Implementation details of what a class contains need not be visible to other classes and objects that use it.
  - Instead, only specific information can be made visible to the other components of the application and the rest can be hidden.
- By hiding the implementation details about what is required to implement the specific operation in the class, the usage of operation becomes simple.



In Java, data hiding is achieved by using **access modifiers**.

## Access Modifiers:

- ◆ Determine how members of a class, such as instance variable and methods are accessible from outside the class.
- ◆ Decide the scope or visibility of the members.

<b>public</b>	<ul style="list-style-type: none"><li>• Members declared as public can be accessed from anywhere in the class as well as from other classes.</li></ul>
<b>private</b>	<ul style="list-style-type: none"><li>• Members are accessible only from within the class in which they are declared.</li></ul>
<b>protected</b>	<ul style="list-style-type: none"><li>• Members to be accessible from within the class as well as from within the derived classes.</li></ul>
<b>package (default)</b>	<ul style="list-style-type: none"><li>• Allows only public members of a class to be accessible to all the classes present within the same package.</li><li>• This is the default access level for all the members of the class.</li></ul>

# Rules for Access Control



While declaring members, a private access modifier cannot be used with abstract, but it can be used with final or static.

No access modifier can be repeated twice in a single declaration.

Private cannot be used with fields and methods of an interface.

The most restrictive access level must be used that is appropriate for a particular member.

Mostly, a private access modifier is used at all times unless there is a valid reason for not using it.



- ◆ Object Initializers provide a way to create an object and initialize its fields and complement the use of constructors to initialize objects.
- ◆ Two approaches to initialize fields or instance variables of newly created objects:

Using instance  
variable  
initializers

- In this approach, you specify the names of the fields and/or properties to be initialized, and give an initial value to each of them.

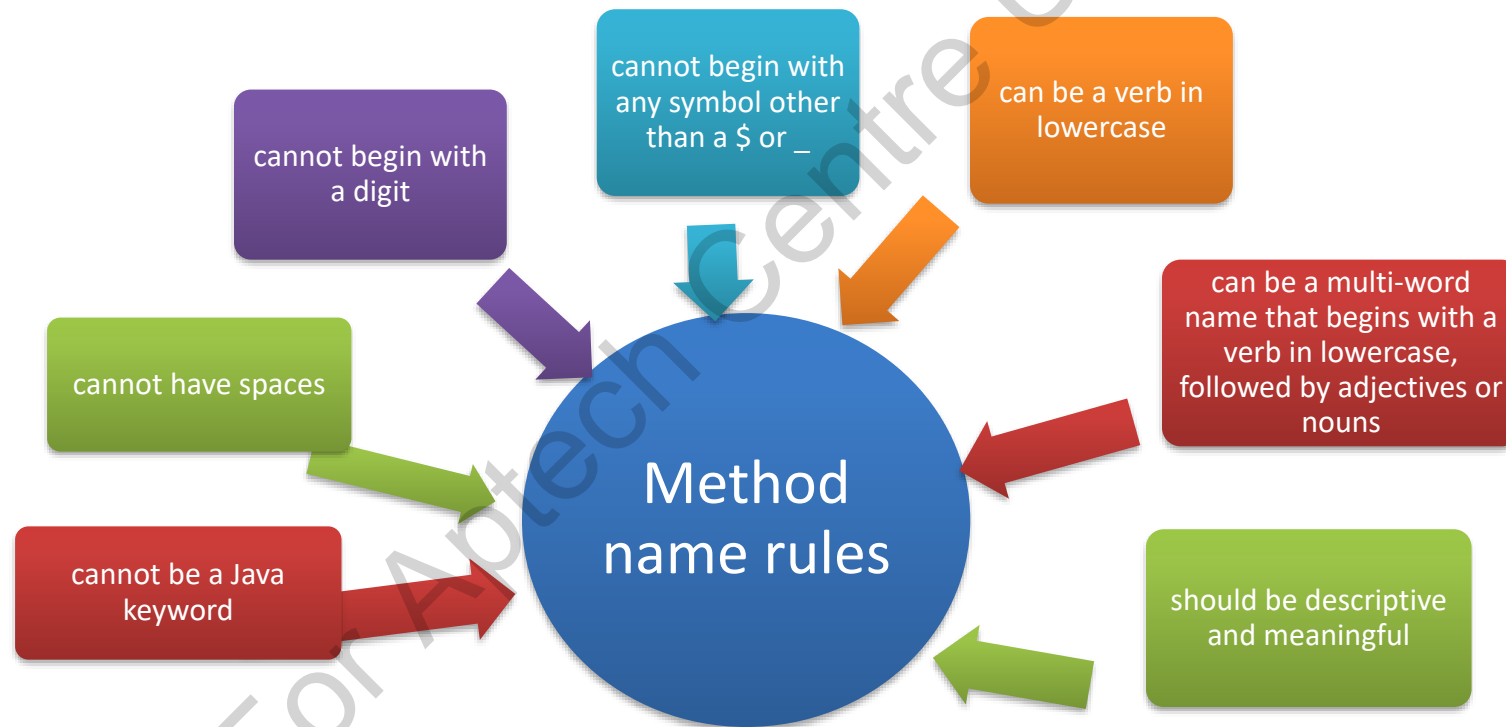
Using  
initialization  
block

- In this approach, an initialization block is specified within the class.

For Aptech Centre Use Only



- ◆ A Java method can be defined as a set of statements grouped together for performing a specific task.
- ◆ For example, a call to the `main()` method which is the point of entry of any Java program, will execute all the statements written within the scope of the `main()` method.

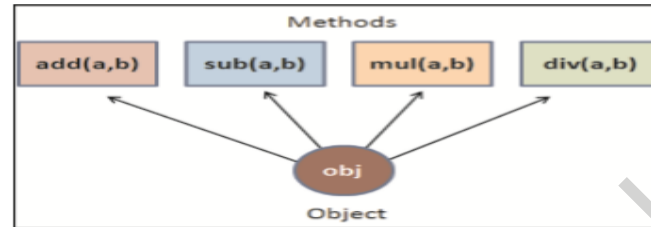




# Creating and Invoking Methods

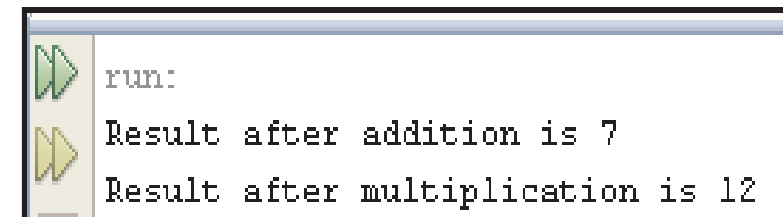
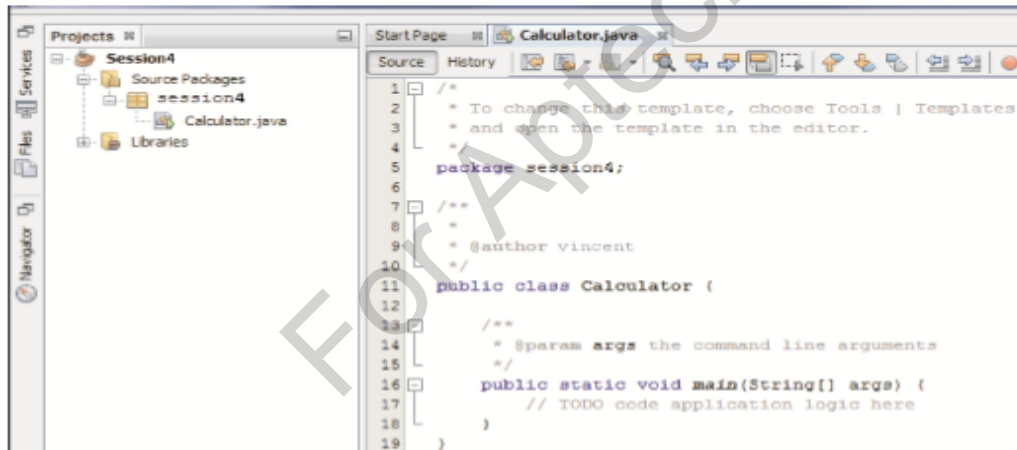


- ◆ A program is modular when different tasks in a program are grouped together into modules or sections.



A method can be invoked in one of following ways:

- ◆ If the method returns a value, then, a call to the method results in return of some value from the method to the caller.
- ◆ For example, `int result = obj.add(20, 30);`



# Passing and Returning Values from Methods 1-2



- ◆ Parameters are the list of variables specified in a method declaration, whereas arguments are the actual values that are passed to the method when it is invoked.
- ◆ A method can accept value of any data type as a parameter.

## Passing Arguments by Value

- ◆ A copy of the argument is passed from the calling method to the called method.
- ◆ Changes made to the argument passed in the called method will not modify the value in the calling method.
- ◆ Variables of primitive data types such as int and float are passed by value.

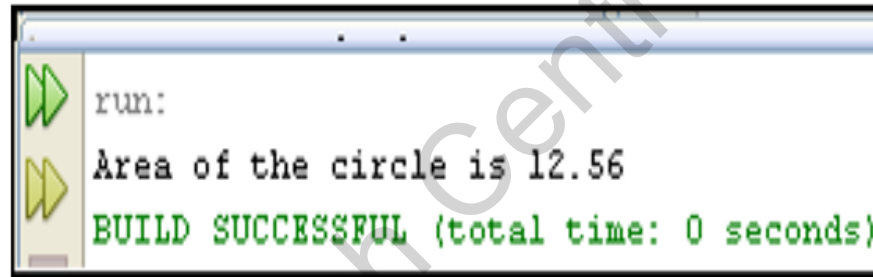
A screenshot of a code editor window. On the left, there are two green arrow icons pointing right. The main area of the editor contains the following text:

```
run:  
Value of num1 after invoking setVal is 10  
BUILD SUCCESSFUL (total time: 0 seconds)
```



## Passing Arguments by Reference

- ◆ The actual memory location of the argument is passed to the called method and the object or a copy of the object is not passed.
- ◆ The called method can change the value of the argument passed to it.
- ◆ Variables of reference types such as objects are passed to the methods by reference.



```
run:  
Area of the circle is 12.56  
BUILD SUCCESSFUL (total time: 0 seconds)
```

## Returning Values from Methods

- ◆ A method will return a value to the invoking method only when all the statements in the invoking method are complete or when it encounters a return statement or when an exception is thrown.

# Declaring Variable Argument Methods

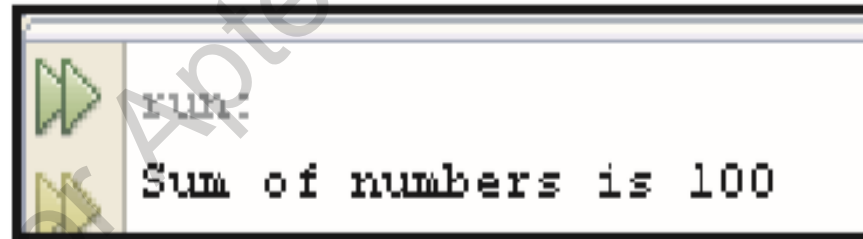


- ◆ Java provides a feature called varargs to pass variable number of arguments to a method.
- ◆ varargs is used when the number of a particular type of argument that will be passed to a method is not known until runtime.

## Syntax:

```
<method_name>(type ... variableName) {  
    // method body  
}
```

where, '...': Indicates the variable number of arguments.



# Using Javadoc to Lookup Methods of a Class

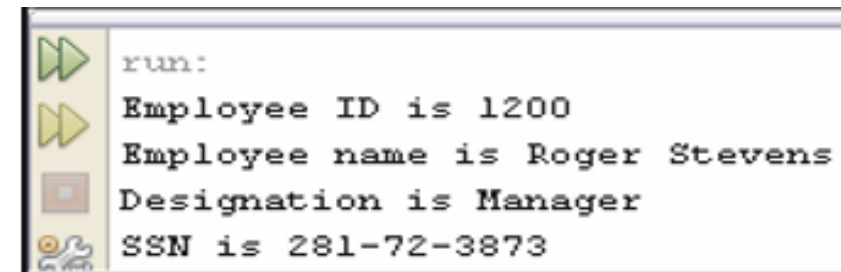
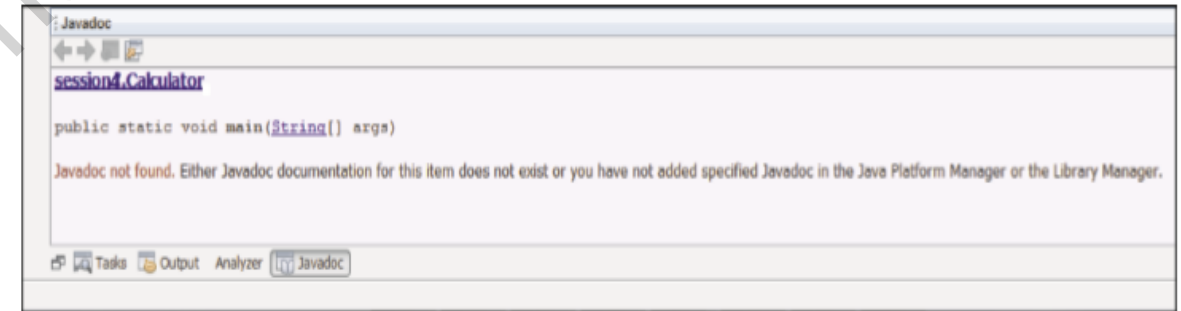
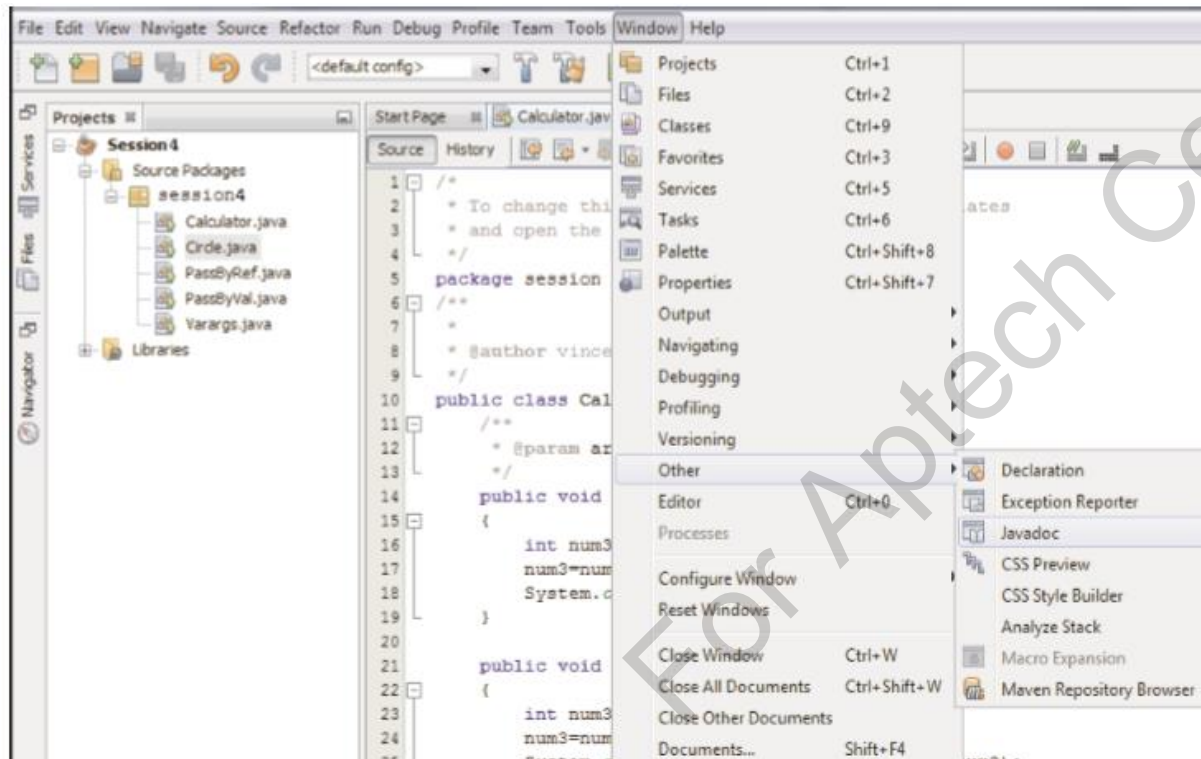


## API documentation or API docs

- These are the online or hard copy descriptions of the API that are primarily intended for the programmers.
- The API specification consists of all assertions for a proper implementation of the Java Platform to ensure that the 'write once, run anywhere' feature of Java is retained.

## Documentation comments or doc comments

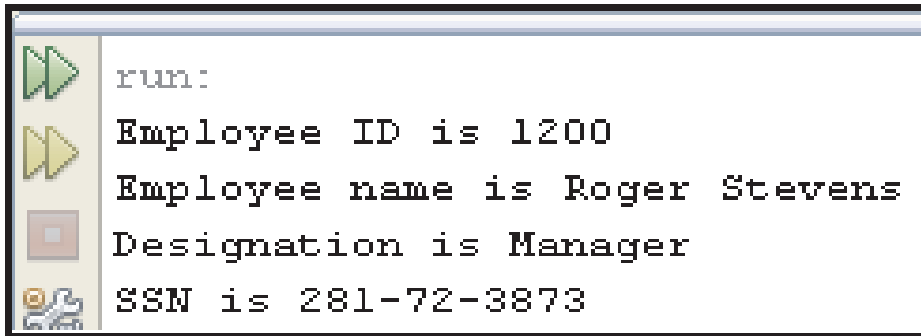
- These are special comments in the Java source code. They are written within the `/** ... */` delimiters.
- These comments are processed by the Javadoc tool for generating the API docs.



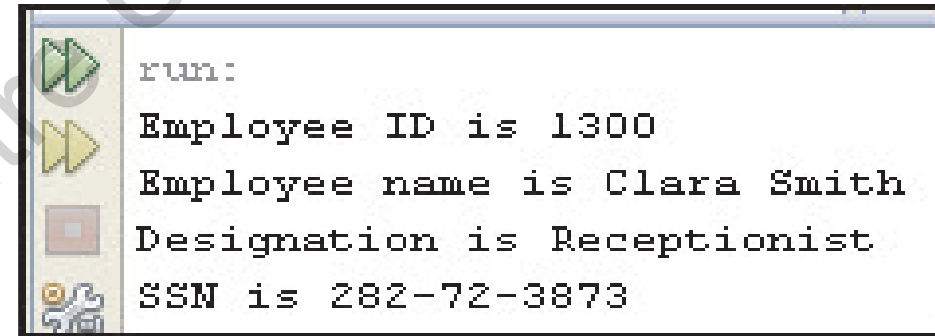
# Using Access Modifiers with Variables and Methods



- ◆ The access modifiers discussed earlier can be used with variables and methods of a class to restrict access from other classes.



```
run:  
Employee ID is 1200  
Employee name is Roger Stevens  
Designation is Manager  
SSN is 281-72-3873
```

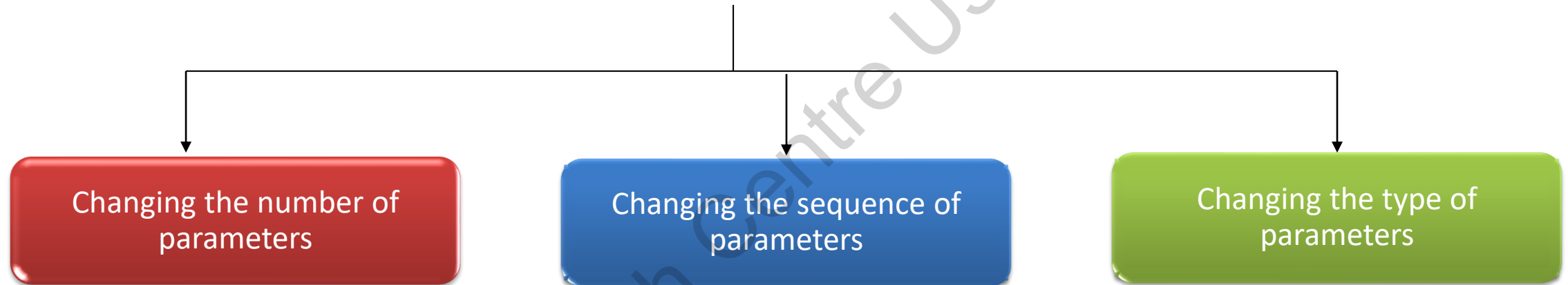


```
run:  
Employee ID is 1300  
Employee name is Clara Smith  
Designation is Receptionist  
SSN is 282-72-3873
```

For Aptech Centre Use Only



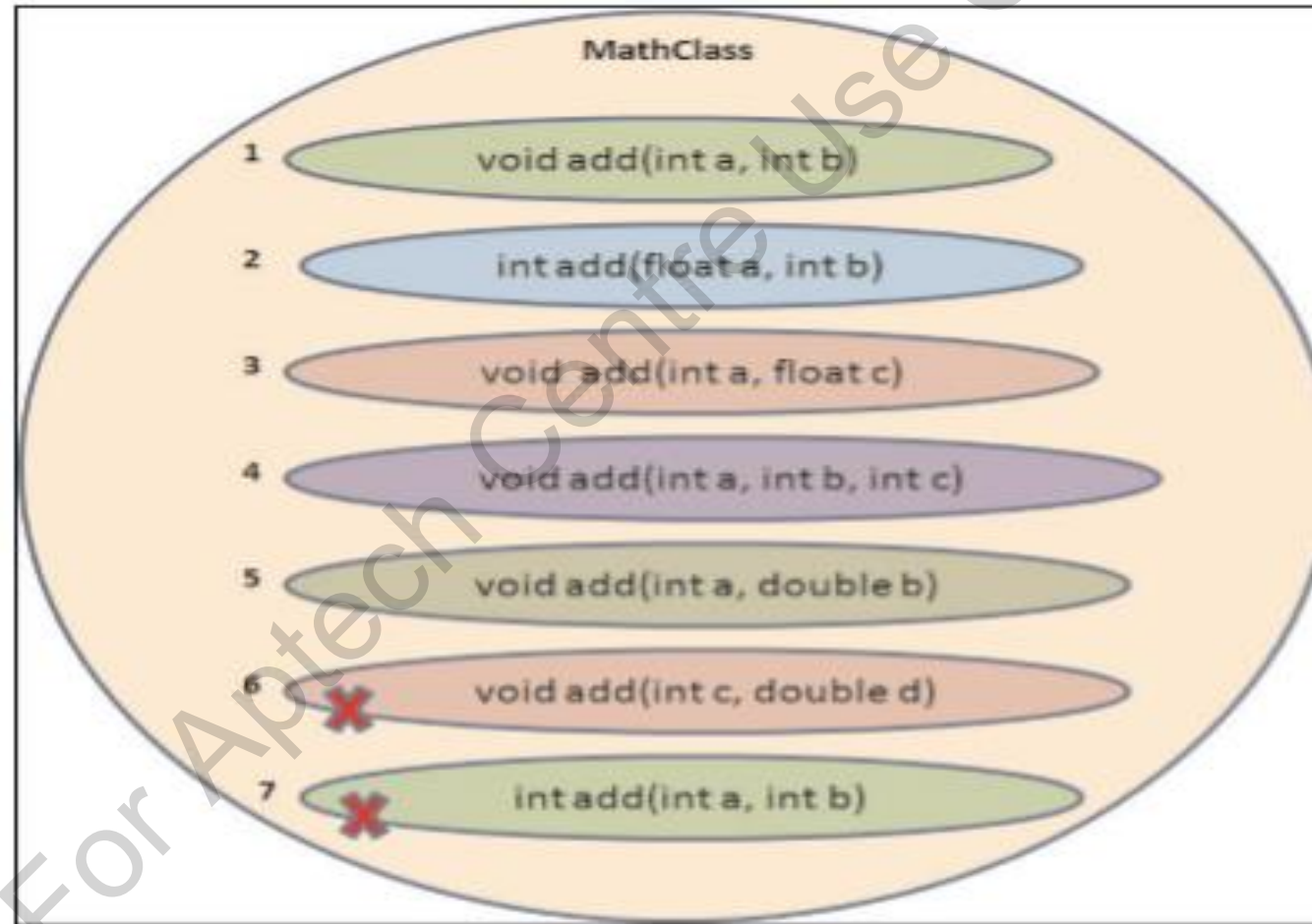
- ◆ Using method overloading, multiple methods of a class can have the same name, but with different parameter lists.
- ◆ Method overloading can be implemented in following three ways:



# Overloading with Different Parameter List



- ◆ Methods can be overloaded by changing the number or sequence of parameters of a method.





# Overloading with Different Data Types



- ◆ Methods can be overloaded by changing the data type of parameters of a method.

```
package session4;
public class MathClass {
    /**
     * Method to add two integers
     *
     * @param num1 an integer variable storing the value of first number
     * @param num2 an integer variable storing the value of second number
     * @return void
     */
    public void add(int num1, int num2) {
        System.out.println("Result after addition is "+ (num1+num2));
    }
    /**
     * Overloaded method to add three integers
     *
     * @param num1 an integer variable storing the value of first number
     * @param num2 an integer variable storing the value of second number
     * @param num3 an integer variable storing the value of third number
     * @return void
     */
    public void add(int num1, int num2, int num3) {
        System.out.println("Result after addition is "+ (num1+num2+num3));
    }
    /**
     * Overloaded method to add a float and an integer
     *
     * @param num1 a float variable storing the value of first number
     * @param num2 an integer variable storing the value of second number
     * @return void
     */
    public void add(float num1, int num2) {
```

```
        System.out.println("Result after addition is "+ (num1+num2));
    }
    /**
     * Overloaded method to add a float and an integer accepting the values
     * in a different sequence
     *
     * @param num1 an integer variable storing the value of first number
     * @param num2 a float variable storing the value of second number
     * @return void
     */
    public void add(int num1, float num2) {
        System.out.println("Result after addition is "+ (num1+num2));
    }
    /**
     * Overloaded method to add two floating-point numbers
     *
     * @param num1 a float variable storing the value of first number
     * @param num2 a float variable storing the value of second number
     * @return void
     */
    public void add(float num1, float num2) {
        System.out.println("Result after addition is "+ (num1+num2));
    }
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // Instantiate the MathClass class
        MathClass objMath = new MathClass();
        // Invoke the overloaded methods with relevant arguments
        objMath.add(3.4F, 2);
        objMath.add(4,5);
        objMath.add(6,7,8);
    }
}
```

```
run:
Result after addition is 5.4
Result after addition is 9
Result after addition is 21
```

# Constructor Overloading



- ◆ Constructor is a special method of a class that has the same name as the class name.
- ◆ A constructor is used to initialize the variables of a class.

```
run:
Rollno :0
Student name:David
Address 302, Washington Street
Score 0.0
-----
Rollno :103
Student name:null
Address null
Score 46.0
-----
Rollno :104
Student name:Roger
Address null
Score 50.0
-----
```



- ◆ Java provides the keyword `this` which can be used in an instance method or a constructor to refer to the current object, that is, the object whose method or constructor is being called.
- ◆ Any member of the current object can be referred from within an instance method or a constructor by using the 'this' keyword.
- ◆ The keyword `this` is not explicitly used in instance methods while referring to variables and methods of a class.

For Aptech Centre Use Only



- ◆ The class is a logical construct that defines the shape and nature of an object.
- ◆ Objects are the actual instances of the class and are created using the new operator. The new operator instructs JVM to allocate the memory for the object.
- ◆ The members of a class are fields and methods. Fields define the state and are referred to as instance variables, whereas methods are used to implement the behavior of the objects and are referred to as instance methods.
- ◆ Each instance created from the class will have its own copy of the instance variables, whereas methods are common for all instances of the class.
- ◆ Constructors are methods that are used to initialize the fields or perform startup operations only once when the object of the class is instantiated.
- ◆ Data encapsulation hides the instance variables that represents the state of an object through access modifiers. The only interaction or modification on objects is performed through the methods.
- ◆ A Java method is a set of statements grouped together for performing a specific operation.
- ◆ Parameters are the list of variables specified in a method declaration, whereas arguments are the actual values that are passed to the method when it is invoked.
- ◆ The variable argument feature is used in Java when the number of a particular type of arguments that will be passed to a method is not known until runtime.
- ◆ Java comes with four access modifiers namely, public, private, protected, and default.
- ◆ Using method overloading, multiple methods of a class can have the same name, but with different parameter lists.
- ◆ Java provides the this keyword which can be used in an instance method or a constructor to refer to the current object, that is, the object whose method or constructor is being invoked.