User-Defined Methods

Classification of Methods

Considering Defined By Whom

1. Methods are two kinds **considering defined by whom**
   1) Pre-defined Methods : Defined by Java developers. E.g.: Math.pow(), nextInt() etc.
   2) User-defined Methods: Defined by users. E.g.: area(), average() etc.

Four Forms (Models) of Methods

2. Methods can be defined in four forms
   1) Input, process and output in method. (Non-parameterized)
   2) Input and process in method. (Non-parameterized)
   3) Process in method. (Parameterized)
   4) Process and output in method. (Parameterized)

The main() Method

3. Which method is the **integral** part of Java?
   The **main()** method is the integral part of Java. It is the **initial** and **final** method. So it is known as **terminal** method or **driver** method.

Need (Use) of Methods

4. What are the uses of a method? Explain briefly.
   1) **Reuse**: We can reuse a method many times wherever we need it.
   2) **Reduce complexity (simplification)**: By dividing complex problems into small parts in the form of methods we can solve them easily.
   3) **Hiding details**: Statements and variables are **hidden in a particular method** from other methods. We **need not bother about the variable names** and **statements** in different methods.

Reuse, Reduce Complexity and Hiding Details: A Demo Program

Find area of **two different size rooms** using a method **area() t**hat finds area of a rectangular shape.

```
import java.util.*;
public class Rectangle
{
    public static void area()
    {
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter length and breadth");
        double l=sc.nextDouble();
        double b=sc.nextDouble();
        double a=l*b;
        System.out.println("Area = "+ a);
    }
    public static void main()
    {
        area();
        area();
    }
}
```

Reduce Complexity (Simplification): Only single set of statements are defined to find two area

Hiding Details

Re-use: Method is called two times

Input/Output:

```
Enter length and breadth
10
8.5
Area = 85.0
Enter length and breadth
12
10
Area = 120.0
```

Methods: Definition, Syntax and Example

5. What is a method?
   A method is a **named block of statements** that **perform some tasks**.
6. Write syntax of method definition with an example.
   Syntax: [access_specifier] [modifier] returntype methodname([parameter list])

```
{
    method body;
}
```

E.g.:

```
public static double area(double x, double y)
{
    double a = x*y;
    return a;
}
```

Note: The [ ] indicates that it is optional.

Method Prototype / Header

7. Define method prototype with an example.

A **method prototype** is the **first line** of the method definition that includes **access specifier, modifier, return type, method name** and **parameter list**.

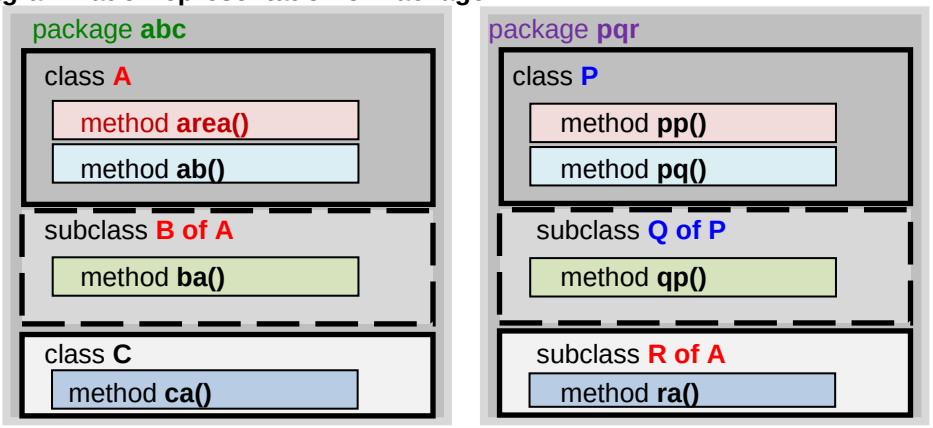
Syntax: [access_specifier] [modifier] returntype methodname([parameter list])

E.g.: public static double area(double x, double y)

The Access Specifiers / Visibility Modifiers

The **specifiers** of the access of a method or a variable or a class from different program regions are referred to **access specifiers**. The three keywords **public**, **protected** and **private** are access specifiers.

Diagrammatic Representation of Package



8. Which are the **four accessibilities**? Describe briefly each of them.

1) **public**: Accessible by **any method** of **any class** in **any package**. E.g.: **public** static void area()

2) **protected**: Accessible by methods of **same package classes** and **sub classes of the same class that are in other packages**. E.g.: **protected** static void area()

3) **Non-specified (*Default*. *This is also known as* *Friendly* *or* *Package*).** Accessible by methods of classes in the **same package** only. E.g.: **static** void area()

4) **private**: Accessible by methods of same class only. E.g.: **private** static void area()

9. Which are **the places** that an access specifier can be used?

Access specifier can be used for **class definition**, **method definition, instance** and **class variable definitions**.

Example for ***class definition****:* **public** class Rectangle

Example for ***method definition****:* **public** static void area()

Example for ***variable definition in class level****:*

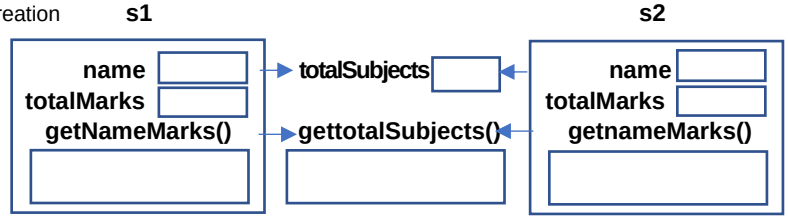
```
public class Rectangle
{    private String name; //instance variable
     private static int totalSubjets; //class variable
}
```

The static Modifier

The **static** is a keyword to specify that **a variable or method is common for class not idividual for objects**.

An Example Program to Demonstrate Use of static Keyword

```
import java.util.*;
public class Student
{
    String name;  double totalMarks;    //individual for each object
    static int totalSubjects;   //common for all objects
    static Scanner sc=new Scanner(System.in); //the sc can be used in static and non-static methods
    public void getNameMarks() //non-static individual for each object
    {    System.out.println("Enter name and total marks");
         name=sc.nextLine();
         totalMarks=sc.nextDouble();
    }
    public static void gettotalSubjects() //common for all objects
    {    System.out.println("Enter number of subjects");
         totalSubjets=sc.nextInt();
    }
    public static void main()
    { //Following two statements are object creation
         Student s1 = new Student();
         Student s2 = new Student();
         s1.getNameMarks();
         s2.getNameMarks();
         gettotalSubjects();
    }
}
```



Two Types of Methods

10. Which are the **two kinds of methods considering memory allocation**?

Two types of methods are **member methods (instance methods)** and **class methods (static methods)**.

Example of member method public void area() (Note: There is no **static** keyword.)

Example of class method: public **static** void area() *(Note: There is* **static** *keyword.)*

Member Method / Instance Method (Non-static)

11. Describe briefly about **member method**.

The method **without *static*** keyword is known as *member method (i.e., method for objects)* or *instance method.* **When objects are created** each object gets **individual copy** of it.

E.g.: public void area()
{ }

Class Method (static Method)

12. Describe briefly about **class method**.

Method with **static** keyword is known as **class method** or **static method**. Class method is common to all objects. Each object shares the single method.

E.g.: public **static** void area()
{ }

Four Kinds of Variables

13. Which are the **four kinds of variables**?

Four kinds of variables are **instance variables, class variables, local variables** and **argument variables**.

Instance Variables / Member Variables / Data Members / Attributes / Characteristics

14. Describe briefly about **instance variables**.

The variables declared in class level **without *static*** keyword are instance variables. **When objects are created** each object gets **individual copy** of these variables.

E.g.: public class Rectangle
{
double l,b;
}

Class Variable (Static variable)

15. Describe briefly **class variable**.

The variable declared in class level with **static** keyword is **class variable**. A class variable is **common** to all objects. Each object shares the single variable.

E.g.: public class Rectangle
{
static double h;
}

Note: Global Variable: The variable declared in class level with **static** keyword and **public** access specifier is known as global variable. E.g.: **public static double h;**

Local Variable

16. Describe briefly about **local variable**.

Variable declared **inside method body** is **local variable**. A local variable has access in that method only.

E.g.: public static double area()
{
double a; return a;
}

Argument Variable

17. Describe briefly about **argument variables**.

Variables declared as **arguments** of a method are **argument variables**. Each argument variable should be declared individually with comma separator (not **;)**. Argument variables have access in that method only.

E.g.: public static void area(**double x, double y**)

Static Scanner Object

18. What is the use of **static Scanner object**?

Instead of creating **Scanner object** in various **static methods** we can create the object **in class level** with **static** keyword. So we can avoid repetition.

E.g.: **static** Scanner **sc** = new Scanner(System.in));

Return Type

19. What is meant by **return type**?

    Data type of the value that is **returned from the method is the return type** of the method. If the type of the value which is returned is **double** then the return type of the method is **double**. If the type of the value which is returned is **int** then the return type is **int**. If a method does not return a value then the return type is **void**.

20. What is the significance (use) of a **void** method?

    If there is **no value to be returned** from a method then the return type of the method should be **void**.

    E.g.: public static **void** main()

Method Name

21. What is the use of method name?

    Method name is used to **identify** a method definition.

    E.g.: **area** in public static double **area()**

22. What are the rules and conventions of writing method name?

    Rules

    1) Identifiers can have only alphabets, digits, underscore and dollar sign.
       (**h.c.f()** is invalid; **phone-bill()** is in invalid. **phone_bill()** is valid).
    2) They must not begin with a digit. (**10B()** is invalid. **B10()** is valid).
    3) They must be a single word; **space is not allowed**. **find Area()** is invalid.
       Multiple words can be written as single word. **findArea()** is valid.
    4) They must not be a keyword or reserved word. (**case()** is invalid. **C**ase() is valid.)
    5) Capital letters and small letters are treated different. (**area()** and **Area()** are different).

    Conventions

    1) The method name should be **meaningful**.
    2) Method name is in **lowercase letters**. For multiple word names, the first letter of subsequent words can be uppercase letter. E.g.: find**A**rea(), calculate**I**ncome**T**ax().
    3) The method name generally begins with a verb followed by a noun. E.g.: findArea()

Parameter List / Argument List

23. What is meant by parameter list?

    The parameter list is a **comma separated list of variables** or **values** in a method prototype or method call inside parenthesis. It is also known as arguments.

    E.g.: **(double x, double y)** *in method prototype.* **(l,b)** *in method call*

Actual and Formal Parameters

24. Describe actual parameter and formal parameter.

    The parameter in the **method call** is **actual parameter**. Actual values reside here and copies of them are passed to formal parameter. E.g.: **l,b** in area(**l,b**);

    The parameter in the **method prototype** is the **formal parameter**. Copies of the actual values are received by formal parameter.

    E.g.: **double x, double y** *in* area(**double x, double y**)

25. Some **points to be noted** when using **variable names in both parameters**.

    - **Different variable names can be used in formal parameter and actual parameter** because values in variables are passed, not the variable name.
    - **Same variable names also can be used** because both methods are different memory locations. Different locations can hold variables with same name.

Method Signature

26. What is a method signature? Write syntax with an example.

    The term **method signature** is referred to **the method name with formal parameter**.

    Syntax: methodname(type variable1, type variable n) E.g.: area(double x, double y)

The Method Body

27. Define method body.

    Method body is the statements enclosed in a pair of braces of method that carry out some specific tasks.

    E.g.:
    ```
    {
        double a=l*b;
        return a;
    }
    ```

The return Statement

28. What is a return statement?

    The statement that returns a value or return the flow of execution from a method definition to the method call is the return statement. E.g.: return a;

29. Write examples of various return statements.

    1) **return a;** → Return of a value in a variable
    2) **return 1;**
    3) **return true;**
    4) **return "Even";**
    5) **return 'A';**

    → Return of constants

    6) **return l*b**; → Return of value of an expression
    7) **return;** → Return of program control. It is optional that can be used in void methods.

30. **How many values** can be returned from a method?

    Only one value can be returned from a method.

31. **How many return statements** can be written in a method?

    As many return statements can be written but it should be with a **selection statement** (if else and switch case) so that only one return statement can be executed.

    E.g.:
    ```
    if(a>b)
        return a;
    else
        return b;
    ```

32. Write the **similarity and difference** between **return** and **System.exit()**.

    Both the **return** statement and **System.exit()** method are used to jump out of a method. The return statement helps to return program control to the calling method but the **System.exit()** helps to exit from the program.

    Note: There should be an integer argument in the method. E.g.: System.exit(**0**);

33. What is the significance of **return** statement in **main()** method?

    The **return** statement in **main()** terminates the program.

Method Call

34. What is a method call? Write syntax and example.

    A method call is a statement that is used to invoke (execute/activate) a method.

    Syntax: methodname([actual parameters]); *E.g.:* area(l,b);

Method Call in an Expression

35. What is the **rule of using a method call in an expression**?

    - **If a method returns a value**
      - then the method call should be initialized/assigned to a variable of **same data type** of the returned value. E.g.: **double a = area(l,b)**;
      - or the method call should be in **print** statement. E.g.: **System.out.println(**"Area = " + **area(l,b))**;
      - or it should be in **if** statement. E.g.: **if(area(l,b)** > 100**)**
    - **If the method is void type**
      - then the method call **should not be used** with an **assignment statement** or in **print statement** or in **if statement**. E.g.: **area();** and **area(l,b)**;

A static Method Call from Another Class [For additional knowledge. Not for exam.]

36. How to call a **static** method of a class from an another class. Illustrate with an example.

    To call a **static** method of a class from another class it requires **class reference**.

    Syntax: classname.methodname(); E.g.: **Math.sqrt(36);**

```
class Square
{
   public static double area(double x)
   {
      return x*x;
   }
}
public class Shape
{
   public static void main()
   {
      double a = Square.area(6);
   }
}
```

Note: To call a **static method** from **same class** it **does not require** any reference, only method call is needed.

A Non-static Method Call from Another Class [For additional knowledge. Not for exam.]

37. How to call a **non-static method** of a class from another class. Illustrate with an example.

To call a non-static method (instance method) of a class from another class it requires object reference. Syntax: objectname.methodname(); E.g.: **ob.area();**

```
class Square
{
   public double area(double x)  //Note: there is non-static keyword
   {
      return x*x
   }
}
public class Shape
{
   public static void main()
   {
       Square ob = new Square();
       double a = ob.area(6.0);
   }
}
```

Note: To call a **non-static method** from **same class** it requires **object** (of same class) **reference**. See page 2

Method Overloading

38. What is method overloading?

Several different method definitions using **same method name** that are **differentiable** by the **type of arguments** or **number of arguments** is known as **method overloading**.

39. Illustrate **method overloading** with an example.

```
public class Shapes
{
    public static void area(float x)
    {
       System.out.println("Area of square: "+x*x);
    }
    public static void area(double x, double y)
    {
       System.out.println("Area of rectangle: "+ x*y);
    }
    public static void area(double x)
    {
       System.out.println("Area of circle: "+ 3.14*x*x);
    }
    public static void main()
    {
       area(10.0);
       area(9.0,5.0);
       area(7F);
    }
}
```

Output:

```
Area of circle: 314.0
Area of rectangle: 45.0
Area of square: 49.0
```

Note: Though the method names are same a method call distinguishes its own method definition by **matching** the **actual argument** with **formal argument** considering the **data type** and **number of arguments.**

40. How does a method call **distinguish** its method definition when method names are same?

By **matching** the **actual argument** with **formal argument** with **data type** and **number of arguments**, a method call can distinguish its own method definition even though the method names are same.

41. What is the **use of method overloading**?

When using method overloading technique the programmer **need not to bother about the method name** and he has to **choose only the right type of values**, rest will be considered by the compiler.

42. What is the **role of return type** of method in method overloading?

There is **no special role** for return type in method overloading; arguments have the important role.

43. How the method overloading become an **important part of the OOP**?

Polymorphism is one of the principles of OOP. Method overloading implements this principle. So method overloading becomes an important part of the OOP.

Call By Value and Call By Reference

44. Illustrate **Call By Value / Pass by Value** with an example.

```
public class Swap
{
   static int a=8,b=6;
   public static void interchange(int a, int b)
   {
      int c=a; //8
      a=b; //6
      b=c;  //8
      System.out.println("Result in interchange : a  = "+a+", b = "+b);
   }
   public static void main()
   {
      System.out.println("Before method call: a = "+a+", b = "+b);
      interchange(a,b);
      System.out.println("After method call    : a = "+a+",  b = "+b);
  }
}
```

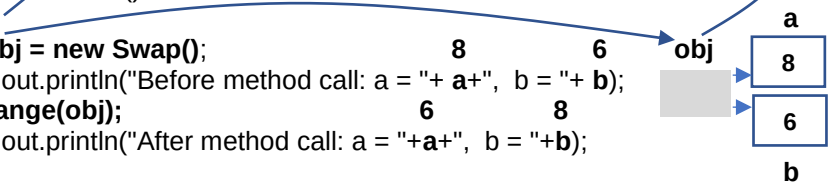
Output:

```
Before method call      : a = 8,  b = 6
Result in interchange   : a = 6,  b = 8
After method call       : a = 8,  b = 6
```

When a method is called by **passing primitive values** then it is **call by value**. In call by value **copies of values are passed** to method definition. Process is done with these copies and changes happen only on these copies and the **original value remain unchanged**.

45. Illustrate **Call By Reference** with an example.

```
public class Swap
{
   static int a=8,b=6;
   public static void interchange(Swap ob)
   {
       int c=ob.a;  //8
       ob.a=ob.b;  //6
       ob.b=c;  //8                                    6          8
       System.out.println("Result in interchange: a = "+ob.a+", b = "+ob.b);
   }
   public static void main()
   {
       Swap obj = new Swap();                          8          6      obj
       System.out.println("Before method call: a = "+ a+",  b = "+ b);
       interchange(obj);                               6          8
       System.out.println("After method call: a = "+a+",  b = "+b);
   }
}
```



Output:

```
Before method call   : a = 8,  b = 6
Result in interchange: a = 6,  b = 8
After method call    : a = 6,  b = 8
```

When a method is called by **passing reference values** in **objects** then it is known as **call by reference**. **Process is done with these reference values** and the **change** that happens **is reflected in the original data**.

46. Describe **Call By Value** and **Call By Reference**.

When a method is called by a method call by passing primitive values then it is **call by value**. When a method is called by a method call by passing reference values like objects then it is known as **call by reference**.

In call by value copies of values are passed to method definition and these copies are received by formal parameters. Process is done with these work copies and changes happen only on these work copies and the **original data remain unchanged**.

In call by reference, reference of object or array is passed to method definition and this is received by formal parameter. Process is done with this reference value and the **change** that happens **is reflected in the original data**.

47. What is the **difference between call by value and call by reference**?

Write the above description.

48. **Arguments** to the methods can be of **two kinds**. Which are they?

    There are two kinds of arguments to methods. 1) Primitive values. 2) Reference values.

49. **How primitive** type values and **objects and arrays** are **passed to other methods**?

    Primitive type values are passed by value (call by value) and objects and arrays (reference type values) are passed by reference (call by reference).

50. What is the **advantage** of **call by value**?

    Passing arguments by value is useful when the original values are not to be modified. Call by value offers security that the function cannot harm original values.

51. What is the **advantage** of **call by reference**?

    Passing arguments by reference is useful when the original values are to be modified.

52. What is meant by **reference variable**?

    Objects and **arrays** are reference variables. E.g.: **obj** in Swap **obj**=new Swap();

Pure and Impure Methods

53. What is a pure method and an impure method?

    A **pure method** is the one that takes primitives as arguments but **does not modify** the passed **primitives**.

    E.g.:
    ```
    public static void count(int c)
    {
        c=c+1;
    }
    ```

    An **impure method** is one that **modifies the value of the original object**.

    E.g.:
    ```
    public static void count(Number x)
    {
        x.c=x.c+1;
    }
    ```

54. What is a **modifier method**?

    An **impure method** is also known as **modifier method** because it modifies the state of the received object.

Three Categories of Methods Considering Return Value

55. Which are the three categories of methods based on **returning of values**?

    1) **Computational methods:** Methods that calculate some values and return the computed value. For example, method to find average of some marks.
    2) **Manipulative methods**: Methods that manipulate information and return a success or failure code. For example boolean return type methods such as to check whether a number is even or odd.
    3) **Procedural methods**: Methods that perform an action and has no explicit return value. For example void return type methods such as to print something.

Various Categorizations of Methods

56. Which are the various **categorizations** of methods?

    1) **Considering defined by whom**
        a) Pre-defined methods
        b) User-defined methods
    2) **Considering memory allocation**
        a) Member method
        b) Class methods
    3) **Considering modification of original data**
        a) Pure methods
        b) Impure methods
    4) **Considering return value**
        a) Computational methods
        b) Manipulative methods
        c) Procedural methods