3.3.5 Methods

- Method provides the functionality or operation or behavior of an object.

  Syntax:

```
[modifiers] <retrunType> <methodName> (<params>)
{
   // Method implementation
}
```

Types of Methods

1) Instance Methods
2) Static Methods

Instance Methods

- Methods defined inside the class without using static modifier are called as Instance methods.

Ex:

```
class Hello
{
  void m1(){
   ...
  }

}
```

```
Hello h = new Hello();
h.m1();   //Valid

Hello.m1(); //Invalid

Hello h=null;
h.m1(); // Invalid
```

- Instance methods must be invoked with the reference variable which contains object.

Static Methods

- Methods defined inside the class with using static modifier are called as Static methods.

Ex:

```
class Hello
{

static void m2(){
   ...
  }
}
```

```
Hello h = new Hello();
h.m2();  //Valid

Hello.m2(); //Valid

Hello h=null;
h.m2();// Valid
```

- **Static methods can be invoked in three ways:**
    1. **With class name**
    2. **With the reference variable which contains null**
    3. **With the reference variable which contains object**

Lab312.java

```
class Hello{
int a=10;
static int b=20;

void m1(){
System.out.println("I am m1()");
}
static void m2(){
System.out.println("I am m2()");
}

void show(){
System.out.println(a);
System.out.println(b);
m1();
m2();
}
}
class Lab312{
public static void main(String args[]){
Hello h=new Hello();
h.show();
}
}
```

Lab313.java

```
class Hello{
int a=10;
static int b=20;

void m1(){
System.out.println("I am m1()");
}
static void m2(){
System.out.println("I am m2()");
}

static void show(){
System.out.println(a);
System.out.println(b);
m1();
m2();
}
}
class Lab313{
public static void main(String args[]){
Hello h=new Hello();
h.show();
}
}
```

Lab314.java

```
class Hello{
void m1(){
System.out.println("I am m1()");
}
}

class Lab314{
public static void main(String args[]){
Hello h=new Hello();
h.m1();
}
}
```

Lab315.java

```
class Hello{
void m1(){
System.out.println("I am m1()");
}
}

class Lab315{
public static void main(String args[]){
Hello h=null;
h.m1();
}
}
```

Lab316.java

```
class Hello{
void m1(){
System.out.println("I am m1()");
}
}

class Lab316{
public static void main(String args[]){
Hello.m1();
}
}
```

Lab317.java

```
class Hello{
static void m2(){
System.out.println("I am m2()");
}
}

class Lab317{
public static void main(String args[]){
Hello h= new Hello();
h.m2();
}
}
```

Lab318.java

```
class Hello{
static void m2(){
System.out.println("I am m2()");
}
}

class Lab318{
public static void main(String args[]){
Hello h= null;
h.m2();
}
}
```

Lab319.java

```
class Hello{
static void m2(){
System.out.println("I am m2()");
}
}

class Lab319{
public static void main(String args[]){
Hello.m2();
}
}
```

3.3.5.1 Methods Return Type

- You can define a method for performing an operation. After finishing operation, Method may produce some result or may not.
- Result produced from method can be used in two ways:
  1. Using the result inside the method
  2. Returning the result to the caller method

- When you want to return the result to the caller of the method then you must specify return type depending on the result.
- When you don't want to return the result to the caller of the method then you must specify return type as void.

Lab320.java

```
class Hello{
 show(){
System.out.println("I am show()");
}
}

class Lab320{
public static void main(String args[]){
Hello h= new Hello();
h.show();
}
}
```

Lab321.java

```
class Hello{
void show(){
System.out.println("I am show()");
}
}

class Lab321{
public static void main(String args[]){
Hello h= new Hello();
h.show();
}
}
```

Lab322.java

```
class Hello{
void show(){
System.out.println("I am show()");
return;
}
}

class Lab322{
public static void main(String args[]){
Hello h= new Hello();
h.show();
}
}
```

Lab323.java

```
class Hello{
void show(){
System.out.println("I am show()");
return 99;
}
}

class Lab323{
public static void main(String args[]){
Hello h= new Hello();
h.show();
}
}
```

Lab324.java

```
class Hello{
int show(){
System.out.println("I am show()");
return 99;
}
}
class Lab324{
public static void main(String args[]){
Hello h1= new Hello();
int x=h1.show();
System.out.println(x);

Hello h2= new Hello();
h2.show();
}
}
```

Lab325.java

```
class Hello{
int show(){
System.out.println("I am show()");
}
}

class Lab325{
public static void main(String args[]){
Hello h= new Hello();
h.show();
}
}
```

Lab326.java

```
class Hello{
int show(){
System.out.println("I am show()");
return;
}
}

class Lab326{
public static void main(String args[]){
Hello h= new Hello();
h.show();
}
}
```

Lab327.java

```
class Hello{
int show(){
System.out.println("I am show()");
return 99L;
}
}

class Lab327{
public static void main(String args[]){
Hello h= new Hello();
h.show();
}
}
```

Lab328.java

```
class Hello{
long show(){
System.out.println("I am show()");
return 99;
}
}

class Lab328{
public static void main(String args[]){
Hello h= new Hello();
h.show();
}
}
```

Lab329.java

```
class Hello{
long show(){
System.out.println("I am show()");
return 'A';
}
}

class Lab329{
public static void main(String args[]){
Hello h= new Hello();
h.show();
}
}
```

Lab330.java

```
class Hello{
boolean show(){
System.out.println("I am show()");
return 'A';
}
}
class Lab330{
public static void main(String args[]){
Hello h= new Hello();
h.show();
}
}
```

Lab331.java

```
class Hello{
boolean show(){
System.out.println("I am show()");
return true;
}
}
class Lab331{
public static void main(String args[]){
Hello h= new Hello();
h.show();
}
}
```

Lab332.java

```
class Hello{
String show(){
System.out.println("I am show()");
String str="Hello Guys !!!";
return str;
}
}
class Lab332{
public static void main(String args[]){

Hello h= new Hello();
String x  = h.show();
System.out.println(x);
}
}
```

Lab333.java

```
class Hello{
String show(){
System.out.println("I am show()");
return null;
}
}
class Lab333{
public static void main(String args[]){

Hello h= new Hello();
String x  = h.show();
System.out.println(x);
}
}
```

Lab334.java

```
class Hello{
int show(){
System.out.println("I am show()");
return null;
}
}

class Lab334{
public static void main(String args[]){
Hello h= new Hello();
 h.show();
}
}
```

Lab335.java

```
class Hello{
int show(){
System.out.println("I am show()");
return 88;
System.out.println("Hello Guys");
}
}

class Lab335{
public static void main(String args[]){
Hello h= new Hello();
 h.show();
}
}
```

Lab336.java

```
class Hello{
int show(){
System.out.println("I am show()");
return 88;
return 99;
}
}

class Lab336{
public static void main(String args[]){
Hello h= new Hello();
 h.show();
}
}
```

Lab337.java

```
class Hello{
boolean isEven(int n){
if(n%2==0)
  return true;
else
return false;
}
}

class Lab337{
public static void main(String args[]){
Hello h= new Hello();
boolean x= h.isEven(9);
System.out.println(x);
}
}
```

3.3.5.2 Methods Parameters

There are two types of arguments:

1) Formal Arguments

2) Actual Arguments

Formal Arguments

- ✓ Arguments specified with the method definition or declaration are called as Formal Arguments.
- ✓ Formal Arguments are also called as Parameters.

Actual Arguments

- ✓ Arguments specified with method call are called as Actual Arguments.

Ex:

```
class Hello{
void show(int a, int b){
...
}
}
```

Formal Arguments

```
Hello h=new Hello();
h.show(10,20);
```

Actual Arguments

Lab338.java

```
class Hello{
void sum(int a,int b){
int sum=a+b;
System.out.println(sum);
}
}

class Lab338{
public static void main(String args[]){
Hello h= new Hello();
h.sum(10,20);
}
}
```

Lab339.java

```
class Hello{
void sum(int a,int b){
int sum=a+b;
System.out.println(sum);
}
}

class Lab339{
public static void main(String args[]){
Hello h= new Hello();
h.sum();
h.sum(10);
}
}
```

Lab340.java

```
class Hello{
void sum(int a,int b){
int sum=a+b;
System.out.println(sum);
}
}

class Lab340{
public static void main(String args[]){
Hello h= new Hello();
h.sum(10.5,9.5);
}
}
```

Lab341.java

```
class Hello{
void sum(int a,int b){
int sum=a+b;
System.out.println(sum);
}
}

class Lab341{
public static void main(String args[]){
Hello h= new Hello();
byte b1=10;
byte b2=20;
h.sum(b1,b2);
}
}
```

3.3.5.3 Method Overloading

- **You can write multiple methods inside the class with the same name by changing parameters. This process is called as METHOD OVERLOADING.**
- **When you are overloading methods then you must follow the following rules:**
  1) **Method name must be same**
  2) **Method parameter must be changed in terms of**
     a. **Number of parameters**
     b. **Type of parameters**
     c. **Order of parameters**
  3) **Method return type can be anything**

Lab342.java

```
class Hello{
void show(int a,int b){
System.out.println("show(int,int)");
}

int show(int x,int y){
System.out.println("show(int,int)");
return 99;
}
}
class Lab342{
public static void main(String args[]){
Hello h= new Hello();
h.show(10,20);
}
}
```

Lab343.java

```
class Hello{
void show(int a,int b){
System.out.println("show(int,int)");
}

int show(int x,int y,int z){
System.out.println("show(int,int,int)");
return 99;
}
}
class Lab343{
public static void main(String args[]){
Hello h= new Hello();
h.show(10,20);
h.show(10,20,30);
}
}
```

Lab344.java

```
class Hello{
void show(int a,int b){
System.out.println("show(int,int)");
}
void show(String x,int y){
System.out.println("show(String,int)");
}
}
```

```
class Lab344{
public static void main(String args[]){
Hello h= new Hello();
h.show(10,20);
h.show("Hello",20);
}
}
```

Lab345.java

```
class Hello{

void show(int a,String b){
System.out.println("show(int,String)");
}

void show(String x,int y){
System.out.println("show(String,int)");
}

}

class Lab345{
public static void main(String args[]){
Hello h= new Hello();
h.show(10,"Hello");
h.show("Hello",20);
}
}
```

Lab346.java

```
class Hello{

void show(int a,byte b){
System.out.println("show(int,byte)");
}

}

class Lab346{
public static void main(String args[]){
Hello h= new Hello();
h.show(10,20);

}
}
```

Lab347.java

```
class Hello{

void show(int a,byte b){
System.out.println("show(int,byte)");
}

}

class Lab347{
public static void main(String args[]){
Hello h= new Hello();
byte b1=10;
byte b2=20;

h.show(b1,b2);
}
}
```

Lab348.java

```
class Hello{

void show(byte a,int b){
System.out.println("show(byte,int)");
}

}

class Lab348{
public static void main(String args[]){
Hello h= new Hello();
h.show(10,20);
}
}
```

Lab349.java

```
class Hello{
void show(byte a,int b){
System.out.println("show(byte,int)");
}
}
class Lab349{
public static void main(String args[]){
Hello h= new Hello();
byte b1=10;
byte b2=20;
h.show(b1,b2);
}
}
```

Lab350.java

```
class Hello{
void show(int a,byte b){ //1
System.out.println("show(int,byte)");
}
void show(byte a,int b){ //2
System.out.println("show(byte,int)");
}
}
class Lab350{
public static void main(String args[]){
Hello h= new Hello();
byte b1=10;
byte b2=20;
h.show(10,b2); //1
h.show(b1,20); //2
}
}
```

Lab351.java

```
class Hello{

void show(int a,byte b){ //1
System.out.println("show(int,byte)");
}

void show(byte a,int b){ //2
System.out.println("show(byte,int)");
}

}

class Lab351{
public static void main(String args[]){
Hello h= new Hello();
byte b1=10;
byte b2=20;
h.show(b1,b2);
}
}
```

Lab352.java

```
class Hello{

void show(String str){
System.out.println("show(String)");
}

void show(Object obj){
System.out.println("show(Object)");
}

}

class Lab352{
public static void main(String args[]){
Hello h= new Hello();
h.show("JLC");
h.show(null);
}
}
```

Lab353.java

```
class Hello{

void show(Object obj){
System.out.println("show(Object)");
}

}

class Lab353{
public static void main(String args[]){
Hello h= new Hello();
h.show("JLC");
h.show(null);
}
}
```

Lab354.java

```
class Hello{

void show(String str){
System.out.println("show(String)");
}

void show(Hello hello){
System.out.println("show(Hello)");
}

}

class Lab354{
public static void main(String args[]){
Hello h= new Hello();
h.show(null);
}
}
```

Lab355.java

```
class Hello{

void show(String str){
System.out.println("show(String)");
}

void show(Hello hello){
System.out.println("show(Hello)");
}

}

class Lab355{
public static void main(String args[]){
Hello h= new Hello();
h.show(null);
}
}
```

Lab356.java

```
class Hello{
void show(Object obj){
System.out.println("show(Object)");
}
void show(String str){
System.out.println("show(String)");
}
void show(Hello hello){
System.out.println("show(Hello)");
}
}

class Lab356{
public static void main(String args[]){
Hello h= new Hello();
h.show(null);
}
}
```

Lab357.java

```
class Hello{
void show(int a){
System.out.println("show(int)");
}
void show(long a){
System.out.println("show(long)");
}
}

class Lab357{
public static void main(String args[]){
Hello h= new Hello();
byte b=10;
h.show(b); //

short s=10;
h.show(s);

int a=10;
h.show(a);

long x=10;
h.show(x);

}
}
```

Lab358.java

```
class Hello{
void show(long a){
System.out.println("show(long)");
}
}

class Lab358{
public static void main(String args[]){
Hello h= new Hello();
byte b=10;
h.show(b); //

short s=10;
h.show(s);

int a=10;
h.show(a);

long x=10;
h.show(x);

}
}
```

Lab359.java

```
class Hello{
/*
void show(byte a){
System.out.println("show(byte)");
}
void show(short a){
System.out.println("show(short)");
}

void show(int a){
System.out.println("show(int)");
}

void show(long a){
System.out.println("show(long)");
}
```

Lab360.java

```
class A{ }
class B extends A{ }
class C extends B{ }

class Hello{
/*
void show(C co){
System.out.println("show(C)");
}

void show(B bo){
System.out.println("show(B)");
}

void show(A ao){
System.out.println("show(A)");
}
void show(float a){
System.out.println("show(float)");
}
*/
void show(double a){
System.out.println("show(double)");
}

}

class Lab359{
public static void main(String args[]){
Hello h= new Hello();
byte b=10;
h.show(b); //

short s=10;
h.show(s);

int a=10;
h.show(a);

long x=10;
h.show(x);

float f=9.9F;
h.show(f);

double d=9.9;
h.show(d);

}
}
```

```
*/
void show(Object obj){
System.out.println("show(Object)");
}

}

class Lab360{
public static void main(String args[]){
Hello h= new Hello();

C co=new C();
h.show(co);

B bo=new B();
h.show(bo);

A ao=new A();
h.show(ao);

Object obj = new Object();
h.show(obj);
}
}
```

3.3.5.4. Method Calls and Stack

What happens when Method is called?

- ✓ Whenever the method is called then that method call will be pushed into Stack in One Stack Frame.
- ✓ Memory will be allocated for All the Local Variables of that method in the Stack Frame.

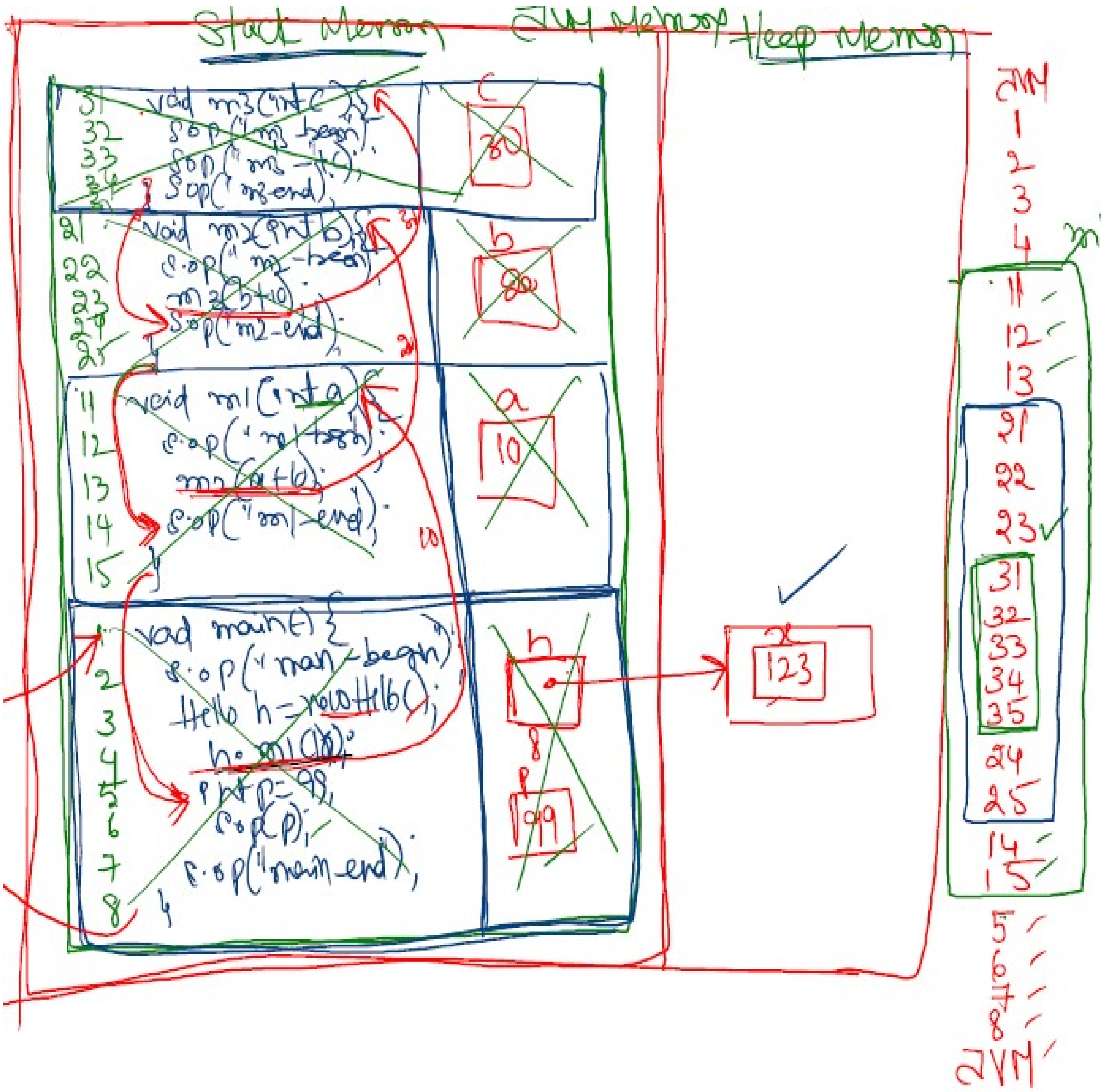
What happens when the Method execution is completed?

- ✓ Whenever the Method execution is completed then that Stack Frame will be deleted.
- ✓ Memory of Local Variables will be cleaned.

Lab361.java

```
class Hello{

int x= 123;

void m1(int a){
System.out.println("m1 - begin");
m2(a+10);
System.out.println("m1 - end");
}

void m2(int b){
System.out.println("m2 - begin");
m3(b+10);
System.out.println("m2 - end");
}

void m3(int c){
System.out.println("m3 - begin");
System.out.println("m3 - "+c);
System.out.println("m3 - end");
}

}
```

```
class Lab361{
public static void main(String args[]){

System.out.println("main - begin");

Hello h= new Hello();
h.m1(10);

int p=99;
System.out.println("p : "+p);

System.out.println("main - end");

}
}
```



```
31 void m3(int c){
32     S.o.p("m3-begin");
33     S.o.p("m3-"+c);
34     S.o.p("m3-end");

21 void m2(int b){
22     S.o.p("m2-begin");
23     m3(b+10);
24     S.o.p("m2-end");
25

11 void m1(int a){
12     S.o.p("m1-begin");
13     m2(a+10);
14     S.o.p("m1-end");
15 }

1  void main(){
2      S.o.p("main-begin");
3      Hello h=new Hello();
4      h.m1(10);
5      int p=99;
6      S.o.p(p);
7      S.o.p("main-end");
8  }
```

3.3.5.5 Call by Value / Call by Reference

Call By Value

- When you invoke a method by passing primitive data types then it is called as CALL BY VALUE mechanism.
- In the case of call by value, modifications happened inside the called method will not be reflected to caller method because both called method and caller method will have their own local copies.

Call By Reference

- When you invoke a method by passing reference data types then it is called as CALL BY REFERENCE mechanism.
- In the case of call by reference, modifications happened inside the called method will be reflected to caller method because both called method and caller method will share same object address.

Lab362.java

```
class Hello{

void show(int a){
System.out.println("show : a = "+a);
 a= a+10;
System.out.println("show : a = "+a);
}

}

class Lab362{
public static void main(String args[]){
int a=10;
System.out.println("main : a = "+a);
Hello h= new Hello();
h.show(a);
System.out.println("main : a = "+a);
}
}
```

Lab363.java

```
class Hai{
int a=10;
}

class Hello{
void show(Hai hai){
System.out.println("show : a = "+hai.a);
 hai.a= hai.a+10;
System.out.println("show : a = "+hai.a);
}
}

class Lab363{
public static void main(String args[]){
Hai hai = new Hai();
System.out.println("main : a = "+hai.a);
Hello h= new Hello();
h.show(hai);
System.out.println("main : a = "+hai.a);
}
}
```

3.3.5.6. Recursion

- Calling the Method from itself is called as Recursion.

- When You call the Method Recursively, you must specify the Condition to Terminate the Recursive Calls. That Condition is Called as Base Consition or Base Case.
- When Base case is not specified then **StackOverflowError** will be thrown at Runtime.

Recursion is the process of calling one method from itself until the given base case is met.

- You can solve every problem is two approaches
  - Iterative Approach
  - Recursive Approach
- Iterative Approach is best suited in some use-cases and Recursive Approach will be good in some use-cases.

Q) When We have to Recursion?
Ans:

- When You are able to divide the Problem into Sub-Problems and You are able to find the solution for Sub-Problem then Go for Recursion

Structure of Recursive Methods.

void show(){
//.1 Base Case
//2. SubTask Logic
//3. Recursive Call
}
	void show(){
	//1. Base Case
	//2. Recursive Call
	//3. SubTask Logic
	}

Lab364.java

```
class Hello{
void show(){
System.out.println("show - begin");
show();
System.out.println("show - end");
}
}

class Lab364{
public static void main(String args[]){
Hello h= new Hello();
h.show();
}
}
```

Lab365.java

```
class Hello{
void printNums(int n){
//1. Base Case
if(n==0)
return;
//2. SubTask Logic
System.out.println(n);

//3. Recursive Call
printNums(n-1);
}
}
class Lab365{
public static void main(String args[]){
Hello h= new Hello();
h.printNums(5);
System.out.println("Done !!!");
}
}
```

Lab366.java

```
class Hello{
void printNums(int n){
//1. Base Case
if(n==0)
return;
//2. Recursive Call
printNums(n-1);

//3. Sub-Task Logic
System.out.println(n);
}
}
class Lab366{
public static void main(String args[]){
Hello h= new Hello();
h.printNums(5);
System.out.println("Done !!!");
}
}
```

Lab367.java

```
class Hello{

int sum(int n){

//1. Base Case
if(n==1)
return 1;

//2. Recursive Call
return n + sum(n-1);

}
}

class Lab367{
public static void main(String args[]){

Hello h= new Hello();
int sum = h.sum(3);
System.out.println("Sum : "+sum);

}
}
```

Lab368.java

```
class Hello{

int fact(int n){

//1. Base Case
if(n==0 || n==1)
return 1;

//2. Recursive Call
return n * fact(n-1);

}
}

class Lab368{
public static void main(String args[]){

Hello h= new Hello();
int fact = h.fact(5);
System.out.println("Fact : "+fact);

}
}
```

3.3.5.7. Var-Args

- It is a newly added feature of Java 5.
- When you want to define a method to take multiple values of one specific type then you can use the following options:
  - Overload the methods
  - Use array as parameters

  Ex:

  ```
  class Hello{
     void sum(int a,int b) {...}
     void sum(int a,int b,int c) {...}
     void sum(int a,int b,int c,int d) {...}
  ...
  }

  class Hello{
    void sum(int[] arr) {...}
  }
  ```

- The above said approaches having some complexity. To avoid the complexity VAR-ARGS concept has introduced.

- With variable arguments concept, you can define a method which can take any number of arguments of the same type.

  Syntax:

  ```
  <returnType> <methodName>(<dataType> <FixedArg>,
  <dataType>...<VarArgs>)
  {
  ...
  }
  ```

  Ex:

  ```
  void sum(int... values){ ... }
  void sum(String str, int... values){ ... }
  ```

Lab369.java

```
class Hello{
void show(int a, int b){ //Fixed Args
System.out.println("show(int,int)");
}
void show(int a, int b,int c){  //Fixed Args
System.out.println("show(int,int,int)");
}
}
class Lab369{
public static void main(String args[]){
Hello h= new Hello();
h.show(10,20);
h.show(10,20,30);
}
}
```

Lab370.java

```
class Hello{
void show(int []arr){ //Fixed Args
System.out.println("show(int []arr)");
for(int x:arr)
System.out.println(x);
}
}
class Lab370{
public static void main(String args[]){
Hello h= new Hello();
h.show(new int[]{10,20});

int arr[] = new int[5];
arr[0]=10;   arr[1]=20;   arr[2]=30;
arr[3]=40;  arr[4]=50;
h.show(arr);
}
}
```

Lab371.java

```
class Hello{
void show(int... arr){ //Var Args
System.out.println("show(int... arr)");
for(int x:arr)
System.out.println(x);
}
}
class Lab371{
public static void main(String args[]){
Hello h= new Hello();
h.show();
h.show(10);
h.show(10,20);
h.show(10,20,30);
h.show(10,20,30,40);
h.show(10,20,30,40,50);
}
}
```

Lab372.java

```
class Hello{
void show(int x,int... arr){ //Fixed and Var Args
System.out.println("show(int... arr)");
}
}

class Lab372{
public static void main(String args[]){
Hello h= new Hello();
h.show();
}
}
```

Lab373.java

```
class Hello{
void show(int x,int... arr){
System.out.println("show(int... arr)");
System.out.println("x : "+x);
System.out.println("Length : "+arr.length);
}
}

class Lab373{
public static void main(String args[]){
Hello h= new Hello();
h.show(10);
h.show(55,66,77,88,99);
}
}
```

Lab374.java

```
class Hello{
void show(int... arr){
System.out.println("show(int... arr)");
}
void show(String... arr){
System.out.println("show(String... arr)");
}
}
class Lab374{
public static void main(String args[]){
Hello h= new Hello();
h.show(10);
h.show("Srinivas");
h.show();
}
}
```

Lab375.java

```
class Hello{
void show(int... arr,String... str){
System.out.println("show(int... arr,String...
str)");
}
}

class Lab375{
public static void main(String args[]){
Hello h= new Hello();
h.show();
}
}
```

Lab376.java

```
class Hello{
void show(int... arr,String str){
System.out.println("show(int... arr,String str)");
}
}

class Lab376{
public static void main(String args[]){
Hello h= new Hello();
h.show("JLC");
}
}
```

Lab377.java

```
class Hello{
void show(String str,int... arr){
System.out.println("show(String str,int...
arr)");
}
}

class Lab377{
public static void main(String args[]){
Hello h= new Hello();
h.show("JLC");
}
}
```

Lab378.java

```
class Hello{

void sum(int... arr){
int sum = 0;
for(int x:arr){
sum = sum +x;
}
System.out.println("Sum : "+sum);
}


}

class Lab378{
public static void main(String args[]){
Hello h= new Hello();
h.sum();
h.sum(10);
h.sum(10,20);
h.sum(10,20,30);
}
}
```

3.3.6. Introduction to Modifiers

	Top Level Class	Class Variables	Local Variables	Class Blocks	Local Blocks	Constructors	Methods
private	NA	A	NA	NA	NA	A	A
protected	NA	A	NA	NA	NA	A	A
public	A	A	NA	NA	NA	A	A
final	A	A	A	NA	NA	NA	A
static	NA	A	NA	A	NA	NA	A
abstract	A	NA	NA	NA	NA	NA	A
native	NA	NA	NA	NA	NA	NA	A
strictfp	A	NA	NA	NA	NA	NA	A
synchronized	NA	NA	NA	NA	A	NA	A
transient	NA	A	NA	NA	NA	NA	NA
volatile	NA	A	NA	NA	NA	NA	NA

A -> Allowed **NA->Not Allowed**

Note : Order of Modifier can be anything.

3.3.7. Exploring Main Method

- **main() method is a special method which will be called by the JVM.**
- **main() method has the standard signature as follows:**
  - **public static void main(String[] args){}**
  - **static public void main(String[] args){}**
- **You can overload main() method but JVM always invokes main() method with standard signature.**
- **main() method works as application launcher i.e it is starting point of your application.**

Explanation of main() method:

public	**It can be accessed from anywhere.**
static	**It can be accessed using Class Name directly.**
void	**It will not return any value to caller(JVM).**
main	**Method Name (Taken from C/C++)**
String[]	**To pass command line values.**

Lab379.java

```
class Lab379{
static {
System.out.println("I am S.B");
}
public static void main(String args){
System.out.println("I am main");
}
}
```

Lab380.java

```
class Lab380{
static {
System.out.println("I am S.B");
}
static public void main(String args[]){
System.out.println("I am main");
}
}
```

Lab381.java

```
class Lab381{
final synchronized static public void
main(String args[]){
System.out.println("I am main");
}
}
```

Lab382.java

```
class Lab382{
static public void main(String... args){
System.out.println("I am main");
}
}
```

Lab385.java

```
class Lab385{
static public void main(String args[]){
System.out.println("Hello Guys!!!");
main(args);
}
}
```

Lab386.java

```
class Lab386{
static public void main(String args[]){
System.out.println("Hello Guys!!!");
}

public static void main(String... args){
System.out.println("Hai Guys!!!");
}
}
```

Lab383.java	Lab384.java

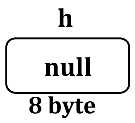
class **Lab383**{
static public void main(String args[]){
System.out.println("I am main(String[])");
}
public static void main(String arg){
System.out.println("I am main(String)");
}
public static void main(int [] args){
System.out.println("I am main(int[])");
}
static public int main(int arg){
System.out.println("I am main(int)");
return 99;
}
 	
}
	class **Lab384**{
	static public void main(String args[]){
	System.out.println("I am main(String[])");
	main("MyJLC");
	main(99);
	main(new int[]{10,20,30});
	}
	
	public static void main(String arg){
	System.out.println("I am main(String)");
	}
	public static void main(int [] args){
	System.out.println("I am main(int[])");
	}
	static public int main(int arg){
	System.out.println("I am main(int)");
	return arg;
	}
	}

3.3.8. Class Loading and Object Creation

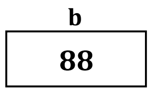
```
class Hello{
int a=99;
static int b=88;
Hello(){
        System.out.println("Hello D.C.");
}
{
        System.out.println("Hello I.B");
}
static{
        System.out.println("Hello S.B.");
}
}
```

Hello h=new Hello ();

1) **Allocates 8 bytes of memory for the reference variable and initializes with null value.**

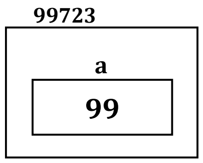


2) **Verifies whether class is loaded or not.**
3) **If not loaded then it loads the class by doing the following task:**
    a. **Reads the byte code from .class file and loads into main memory.**
    b. **Allocates memory for static variables and initializes with default or specified values.**

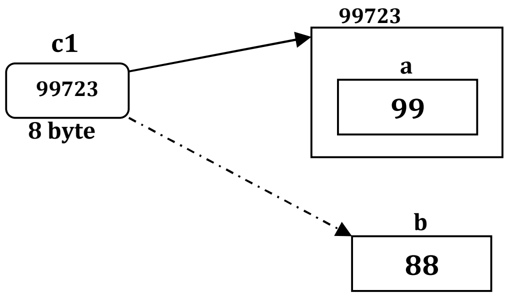


    c. **Executes static blocks.**

4) **Constructor will be invoked. Statement of the constructor will not be executed.**
5) **Allocates the memory for instance variables of the class and initializes with default or specified values.**



6) **Instance block will be executed.**
7) **Statements from the constructor will be executed.**
8) **Address of newly created object will be assigned to the reference variable.**



SUMMARY

1) You can access both instance and static members from instance methods.
2) You can access only static members from static methods.
3) When return type is void then:
   a. You should not specify return statement.
   b. You can specify EMPTY RETURN statement.
4) When return type is other than void then:
   a. You must specify return statement
   b. You should not specify EMPTY RETURN statement.
5) Return value must be same or compatible to the return type.
6) Actual arguments and Formal arguments must be
   a. Same in terms of number of arguments.
   b. Same or compatible in terms of type of arguments.
   c. Same in terms of order of arguments.

7) Consider the following example:

```
    class Hello{
        void m1(byte b){...}
        void m1(short s){...}
        void m1(int i){...}
        void m1(long a){...}
        void m1(float f){...}
        void m1(double d){...}
        }
Hello h=new Hello();
```

Case 1: Hello class contains above said six methods.

```
byte b=12;
h.m1(b);                -> invokes m1(byte b)
```

Case 2: Hello class contains five methods (remove m1 (byte)).

```
byte b=12;
h.m1(b);                -> invokes m1(short s)
```

Case 3: Hello class contains four methods (remove m1 (byte) & m1 (short)).

```
byte b=12;
h.m1(b);            -> invokes m1(int a)
```

Case 4: Hello class contains three methods (remove m1 (byte), m1 (short) & m1 (int)).

```
byte b=12;
h.m1(b);            -> invokes m1(long a)
```

Case 5: Hello class contains two methods (remove m1 (byte), m1 (short), m1 (int) & m1(long)).

```
byte b=12;
h.m1(b);            -> invokes m1(float f)
```

Case 6: Hello class contains one methods (remove m1 (byte), m1 (short), m1 (int), m1 (long) & m1 (float)).

```
byte b=12;
h.m1(b);            -> invokes m1(double f)
```

Case 7: Hello class contains no methods (remove all).

```
byte b=12;
h.m1(b);            -> Compilation Error
```

VAR-ARGS

8) To define Var-Args parameter, you need to use ellipsis [three dot (...)] after data type followed by variable name.

   Ex:

   ```
   void sum(int... values)
   ```

9) Var-Args parameter will be converted to an array by the java compiler while generating class file.

   Ex:

   ```
   void sum(int... values)        -> void sum(int[] values)
   ```

10) When you are calling a method which contains Var-Args parameter by passing zero or more values then Java compiler converts it to an array.

   Ex:

   ```
   sum();              -> sum(new int[0]);
   sum(99,88);         -> sum(new int[]{99,88});
   sum(99,88,77);      -> sum(new int[]{99,88,77});
   ```

11) Var-Args can be used only to define parameters of method or constructor and can't be used for instance variables, static variables or local variables.

12) Var-Args parameter is equivalent to an array. So data available in Var-Args parameter can be accessed like array.

Ex:

```
void sum(int... values){
for(int i=0;i<values.length;i++)
System.out.println(values[i]);
}
```

13) You can pass actual values or array object to the method which is having Var-Args.

Ex:

```
sum(10,20);
int arr[]=new int[]{10,20};
sum(arr);
```

14) Only one argument of the method can be defined as Var-Args and it must be the last.

15) Fixed argument method will have higher priority over Var-Args method.

main

16) Running java Application

Ex:

```
java Hello
```

Before Java7

- Loads the class.
- It checks whether main() method with standard signature available or not.
- If available then invokes the main() method.
- If not available then JVM will throw "java.lang.NoSuchMethodError: main" error.

Java7 onwards

- It checks whether main() method with standard signature available or not.
- If available then loads the class first and invokes main() method.
- If not available then JVM will throw Error:

  Main method not found in class Hello, please define the main method as:

  public static void main(String[] args)

17) You can use Var-Args as main() method parameter.

18) You can use following modifiers for standard main() method:

- public
- static
- final
- strictfp
- synchronized

19) You cannot use following modifiers for standard main() method:

- abstract
- native
- private
- protected

20) You can invoke main() method explicitly.

Assignment #9

Q1) What is method and what is its usage?

Q2) What are the types of methods available in Java?

Q3) What is empty return statement?

Q4) Can I use empty return statement in a method whose return type is other then void?

Q5) Can I write multiple return statements inside a method?

Q6) What is use of return statement inside the method?

Q7) What is use of void in method declaration?

Q8) What is Instance Method?

Q9) What is Static Methods?

Q10)What are the ways to invoke Instance method?

Q11) What are the ways to invoke Static method?

Q12) Can I access Instance members from Instance Methods?

Q13) Can I access Static members from Instance Methods?

Q14) Can I access Instance members from Static Methods?

Q15) Can I access Static members from Static Methods?

Q16) What is caller method?

Q17) What is called method?

Q18) What is Formal Argument?

Q19) What is Actual Argument?

Q20) What is the difference between Call By Value and Call By Reference?

Q21) What is Method Overloading?

Q22) What are the rules to be followed for method overloading?

Q23) What is Recursion?

Q24) What is the reason of StackOverflowError?

Q25) What are Var-Args?

Q26) Which version of Java is required if I want to use Var-Args in method parameter?

Q27) What is the use of method with Var-Args?

Q28) How many arguments of a method can be used as Var-Args?

Q29) Can I use combination of Var-Args and Fixed Args arguments in same method?

Q30) Can I pass array as an argument for Var-Args method calls?

Q31) Can I call a method of Var-Args type without passing any value?

Q32) Can I call a method of Var-Args type by passing null value?

Q33) Can I use ellipsis [three dot (...)] before data type in Var-Args parameter?

Q34) What are modifiers allowed for Top Level Classes?

Q35) What are modifiers allowed for Class Level Variables?

Q36) What are modifiers allowed for Local Variables?

Q37) What are modifiers allowed for Class Level Blocks?

Q38) What are modifiers allowed for Local Blocks?

Q39) What are modifiers allowed for Constructors?

Q40) What are modifiers allowed for Methods?

Q41) What is main () method?

Q42) What is the standard signature of main () method?

Q43) Who is responsible to invoke main () method?

Q44) Can I invoke main () method explicitly?

Q45) Can I overload main () method?

Q46) Which main () method will be invoked by JVM, when I overload main() method?

Q47) What is the starting point of standalone application?

Q48) Why main () method is declared as public?

Q49) Why main () method is declared as static?

Q50) Why main () method return type is void?

Q51) Why main () method name given as main?

Q52) Why main () method takes String array as a parameter?

Q53) Can I execute java program without main () method?

Q54) What are other modifiers allowed for standard main () method?

Q55) Can I write return statement in main ()?

Q56) Explain all the tasks done by JVM at Class Loading time and Object creation time?

Q57) What are the differences between Instance Variables and static Variables?

Q58) What are the differences between Class Level Variables and Local Variables?

Q59) What are the differences between Instance Blocks and Static Blocks?

Q60) What are the differences between Class Level Blocks and Local Blocks?

Q61) What are the differences between Instance Methods and Static Methods?

Q62) What are the differences between Methods and Constructors?

Practice Test #9

Q.No	Question	Options	Answer

1	public class Test1 {
	public static void main(String[] args) {
	 System.out.println("main");
	 show();
	}
	void show(){
	 System.out.println("show");
	} }
		A) Compilation Error
		B) Runtime Error
		C) show
		main
		D) main
		show	

2	class Test2 {
	public static void main(String[] args) {
	 System.out.println("main");
	 new Test2().show();
	}
	void show(){
	 System.out.println("show");
	} }
		A) Compilation Error
		B) Runtime Error
		C) show
		main
		D) main
		show	

3	class Test3 {
	public static void main(String[] args) {
	 System.out.println("main");
	 show();
	}
	static void show(){
	 System.out.println("show");
	}
	}
		A) Compilation Error
		B) Runtime Error
		C) show
		main
		D) main
		show	

4	class Test4 {
	 public static void main(String[] args) {
	 System.out.println("main");
	 Hello h=new Hello();
	 System.out.println(h.show());
	}
	}
	class Hello {
	 String show() {
	 return "show";
	 }
	}
		A) Compilation Error
		B) Runtime Error
		C) main
		D) show
		E) show
		main
		F) main
		show	

5	class Test5 {
	 public static void main(String[] args) {
	 System.out.println("main");
	 Hello h=new Hello();
	 System.out.println(h.show());
	}
	}
	class Hello {
	 String show() {
	 return "";
	 }
	}
		A) Compilation Error
		B) Runtime Error
		C) main
		D) No Output	

6	class Test6 {
	 public static void main(String[] args) {
	 System.out.println("main");
	 Hello h=new Hello();
	 System.out.println(h.show());
	}
	}
	
	class Hello {
	 void show() { }
	}
		A) Compilation Error
		B) Runtime Error
		C) main
		D) No Output	

7	class Test7 {
	public static void main(String[] args) {
	 if (Hello.process(0))
	  System.out.println("IF");
	 else
	  System.out.println("ELSE");
	}
	}
	
	class Hello {
	static boolean process(int x) {
	if (x >= 0)
	 return true;
	else
	 return false;
	}
	}
		A) Compilation Error
		B) Runtime Error
		C) IF
		D) ELSE
		E) IF
		ELSE	

8	class Test8 {
	public static void main(String[] args) {
	 for (int i = 0; Hello.process(i); i++)
	  System.out.println(i);
	} }
	class Hello {
	static boolean process(int x) {
	 if (x < 2)
	  return true;
	else
	  return false;
	} }
		A) Compilation Error
		B) Runtime Error
		C) 0
		D) 1
		E) 2
		F) 0
		1
		2
		G) 0
		1	

9	class Test9 {
	public static void main(String[] args) {
	 if (m1() || m2())
	     System.out.println("IF");
	 else System.out.println("ELSE");
	}
	static boolean m1() {
	 System.out.println("m1");
	 return true;
	}
	static boolean m2() {
	System.out.println("m2");
	 return false;
	}}
		A) Compilation Error
		B) m1
		m2
		IF
		C) m1
		m2
		ELSE
		D) m1
		IF
		E) None of the above	

10	class Test10{
	public static void main(String[] args) {
	 if (m1() && m2())
	  System.out.println("IF");
	 else
	  System.out.println("ELSE");
	}
	static boolean m1() {
	 System.out.println("m1");
	 return true;
	}
	static boolean m2() {
	System.out.println("m2");
	 return false;
	}}
		A) Compilation Error
		B) m1
		m2
		IF
		C) m1
		m2
		ELSE
		D) m1
		IF
		E) m2
		ELSE	

11	class Test11 {
	public static void main(String[] args) {
	 if (m2() && m1())
	  System.out.println("IF");
	 else
	  System.out.println("ELSE");
	}
	static boolean m1() {
	 System.out.println("m1");
	 return true;
	}
	static boolean m2() {
	System.out.println("m2");
	 return false;
	} }
		A) Compilation Error
		B) m2
		m1
		IF
		C) m2
		m1
		ELSE
		D) m2
		IF
		E) m2
		ELSE
			`

12	class Test12 {
	public static void main(String[] args) {
	 if (m2() & m1())
	  System.out.println("IF");
	 else
	  System.out.println("ELSE");
	}
	static boolean m1() {
	 System.out.println("m1");
	 return true;
	}
	static boolean m2() {
	System.out.println("m2");
	 return false;
	}
	}
		A) Compilation Error
		B) m2
		m1
		IF
		C) m2
		m1
		ELSE
		D) m2
		IF
		E) m2
		ELSE	

13	class Test13 {
	 public static void main(String[] args) {
	 System.out.println(new Test());
	}
	static int Test() {
	System.out.println("TEST");
	 return 10;
	}
	}
		A) Compilation Error
		B) TEST
		10
		C) 10
		D) TEST
		E) No Output
		F) None of above	

14	class Test14{
	public static void main(String[] args) {
	 System.out.println(new Test());
	}
	Test() {
	 System.out.println("TEST");
	 return;
	}
	}
		A) Compilation Error
		B) TEST
		10
		C) 10
		D) TEST
		E) No Output
		F) None of above	

15	class Test15 {
	public static void main(String[] args) {
	 show(123.45);
	}
	static void show(int ab) {
	 System.out.println("show");
	 System.out.println(ab);
	}
	}
		A) Compilation Error
		B) show
		123
		C) 123
		D) show
		E) None of above	

16	class Test16 {
	public static void main(String[] args) {
	 show((char)-1);
	}
	static void show(int ab) {
	 System.out.println("show");
	 System.out.println(ab);
	}
	}
		A) Compilation Error
		B) show
		-1
		C) show
		0
		D) show
		E) None of above	

17	class Test17 {
	public static void main(String[] args) {
	 show(0);
	}
	static void show(short sh) {
	 System.out.println("show");
	 System.out.println(sh);
	}
	}
		A) Compilation Error
		B) show
		0
		C) show
		D) None of above	

18	class Test18 {
	public static void main(String[] args) {
	 show((short)0);
	}
	static void show(short sh) {
	 System.out.println("show");
	 System.out.println(sh);
	}
	}
		A) Compilation Error
		B) show
		0
		C) show
		D) None of above	

19	class Test19{
	public static void main(String[] args) {
	 byte b='A';
	 System.out.println(b);
	}
	}
		A) Compilation Error
		B) A
		C) 65
		D) None of above	

20	class Test20{
	public static void main(String[] args) {
	 show('A');
	}
	static void show(byte sh) {
	 System.out.println(sh);
	}
	}
		A) Compilation Error
		B) A
		C) 65
		D) None of above	

21	class Test21 {
	public static void main(String[] args) {
	 show('A');
	}
	static void show(int a) {
	 System.out.println("int");
	}
	static void show(char a) {
	 System.out.println("char");
	}
	}
		A) Compilation Error
		B) int
		C) char
		D) int
		char
		E) char
		int	

22	class Test22 {
	 public static void main(String[] args) {
	 show('A','A');
	}
	static void show(char c,long a) {
	 System.out.println("char-long");
	}
	static void show(char c,int a) {
	 System.out.println("char-int");
	}
	}
		A) Compilation Error
		B) char-long
		C) char-int
		D) None of above	

23	class Test23 {
	public static void main(String[] args) {
	 show('A','A');
	}
	static void show(long a,char c) {
	 System.out.println("long-char");
	}
	static void show(char c,int a) {
	 System.out.println("char-int");
	}
	}
		A) Compilation Error
		B) long-char
		C) char-int
		D) None of above	

24	class Test24 {
	public static void main(String[] args) {
	 show(null);
	}
	static void show(Object obj) {
	 System.out.println("Object");
	}
	static void show(String str) {
	 System.out.println("String");
	}
	}
		A) Compilation Error
		B) String
		C) Object
		D) None of above	

25	class Test25 {
	public static void main(String[] args) {
	 show(null);
	}
	static void show(String str) {
	 System.out.println("String");
	}
	static void show(Test25 t) {
	 System.out.println("Test");
	}
	}
		A) Compilation Error
		B) String
		C) Test
		D) Test
		String
		E) String
		Test	

26	class Test26 {
	public static void main(String[] args) {
	 show((String)null);
	}
	static void show(String str) {
	 System.out.println("String");
	}
	static void show(Test26 t) {
	 System.out.println("Test");
	}
	}
		A) Compilation Error
		B) String
		C) Test
		D) Test
		String
		E) String
		Test	

27	class Test27 {
	public static void main(String[] args) {
	 show((Test)null);
	}
	static void show(String str) {
	 System.out.println("String");
	}
	static void show(Test27 t) {
	 System.out.println("Test");
	}
	}
		A) Compilation Error
		B) String
		C) Test
		D) Test
		String
		E) String
		Test	

28	class Test28 {
	public static void main(String[] args) {
	 show(10.0);
	}
	static void show(int ab) {
	 System.out.println("int");
	}
	static void show(float f) {
	 System.out.println("float");
	}
	}
		A) Compilation Error
		B) int
		C) float
		D) int
		float
		E) float
		int	

29	class Test29 {
	public static void main(String[] args) {
	 show((Object)null);
	}
	static void show(String str) {
	 System.out.println("String");
	}
	static void show(Test29 t) {
	 System.out.println("Test");
	}
	}
		A) Compilation Error
		B) String
		C) Test
		D) Test
		String
		E) String
		Test	

30	public class Test30 {
	static int a;
	public static void main(String[] args) {
	 int a = 10;
	 show(a);
	 System.out.println(Test.a + "\t" + a);
	}
	static void show(int a) {
	 a = a + 20;
	 Test.a = a;
	}
	}
		A) Compilation Error
		B) 0 10
		C) 20 10
		D) 30 10
		E) 0 30
		F) 0 20	

31	class Test31 {
	public static void main(String... args) {
	 System.out.println("Main");
	}
	}
		A) Compilation Error
		B) Runtime Error
		C) Main
		D) No Result	

32	class Test32 {
	static int count = 0;
	public static void main(String[] args) {
	System.out.println("Main");
	if (count > 0)
	  main();
	}
	}
		A) Compilation Error
		B) Runtime Error
		C) Main
		    Main
		D) Main	

33	class Test33 {
	static int count = 0;
	public static void main(String... args) {
	System.out.println("Main");
	if (count > 0)
	  main();
	}
	}
		A) Compilation Error
		B) Runtime Error
		C) Main
		    Main
		D) Main	

34	class Test34 {
	static int count = 0;
	public static void main(String... args) {
	System.out.println("Main");
	count++;
	if (count ==1)
	  main();
	}
	}
		A) Compilation Error
		B) Runtime Error
		C) Main
		    Main
		D) Main	

35	class Test35 {
	static int count = 0;
	public static void main(String... args) {
	System.out.println("Main "+args.length);
	count++;
	if (count == 1)
	  main("Sri");
	}
	}
		A) Compilation Error
		B) Runtime Error
		C) Main 0
		    Main 1
		D) Main 1
		    Main 1
		E) Main 0
		    Main
		F) Main 1
		    Main	

36	class Test36 {
	static int count = 0;
	public static void main(String... args) {
	System.out.println("Main "+args.length);
	count++;
	if (count == 1)
	  main("Sri");
	}
	static void main(String str){
	 System.out.println("Main");
	}
	}
		A) Compilation Error
		B) Runtime Error
		C) Main 0
		    Main 1
		D) Main 1
		    Main 1
		E) Main 0
		    Main
		F) Main 1
		    Main	

37	class Test37 {
	public static void main(String... args) {
	int count = 0;
	count++;
	if (count == 1)
	  main("Sri");
	System.out.println("Main " + args.length);
	}
	}
		A) Compilation Error
		B) Runtime Error
		C) Main 0
		    Main 1
		D) Main 1
		    Main 1	

38	class Test38 {
	public static void main(String... args) {
	System.out.println("Main 1");
	 if (args.length == 0)
	   return;
	System.out.println("Main 2");
	}
	}
		A) Compilation Error
		B) Runtime Error
		C) Main 1
		    Main 2
		D) Main 1	