Types of Variables

Division 1: Based on the type of value represented by a variable all variables are divided into 2 types. They are:

- Primitive variables
- Reference variables

Primitive variables:

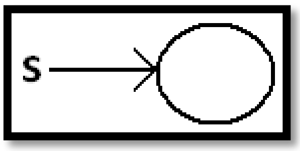
Primitive variables can be used to represent primitive values.

Example: int x=10;

Reference variables:

Reference variables can be used to refer objects.

Example: Student s=new Student();



Division 2: Based on the behavior and position of declaration all variables are divided into the following 3 types.

- Instance variables
- Static variables
- Local variables

Instance variables:

- If the value of a variable is varied from object to object such type of variables are called instance variables.
- For every object a separate copy of instance variables will be created.
- Instance variables will be created at the time of object creation and destroyed at the time of object destruction hence the scope of instance variables is exactly same as scope of objects.
- Instance variables will be stored on the heap as the part of object.
- Instance variables should be declared with in the class directly but outside of any method or block or constructor.
- Instance variables can be accessed directly from Instance area. But cannot be accessed directly from static area.
- But by using object reference we can access instance variables from static area.

Example:

```
class Test
{
   int i=10;
   public static void main(String[] args)
   {
      //System.out.println(i);
//C.E:non-static variable i cannot be referenced from a static context(invalid)
      Test t=new Test();
      System.out.println(t.i);//10(valid)
      t.methodOne();
   }
   public void methodOne()
   {
      System.out.println(i);//10(valid)
   }
}
```

- For the instance variables it is not required to perform initialization JVM will always provide default values.

Example:

```
class Test
{
   boolean b;
   public static void main(String[] args)
   {
      Test t=new Test();
      System.out.println(t.b);//false
   }
}
```

- Instance variables also known as object level variables or attributes.

Static variables:

- If the value of a variable is not varied from object to object such type of variables is not recommended to declare as instance variables. We have to declare such type of variables at class level by using static modifier.
- In the case of instance variables for every object a separate copy will be created but in the case of static variables for entire class only one copy will be created and shared by every object of that class.
- Static variables will be crated at the time of class loading and destroyed at the time of class unloading hence the scope of the static variable is exactly same as the scope of the .class file.

- Static variables will be stored in method area. Static variables should be declared with in the class directly but outside of any method or block or constructor.
- Static variables can be accessed from both instance and static areas directly.
- We can access static variables either by class name or by object reference but usage of class name is recommended.
- But within the same class it is not required to use class name we can access directly.

  - java Test
  - Start JVM.
  - Create and start Main Thread by JVM
  - Locate(find) Test.class by main Thread.
  - Load Test.class by main Thread // static variable creation
  - Execution of main() method.
  - Unload Test.class // static variable destruction
  - Terminate main Thread.
  - Shutdown JVM.

Example:

```
class Test
{
   static int i=10;
   public static void main(String[] args)
   {
      Test t=new Test();
      System.out.println(t.i);//10
      System.out.println(Test.i);//10
      System.out.println(i);//10
   }
}
```

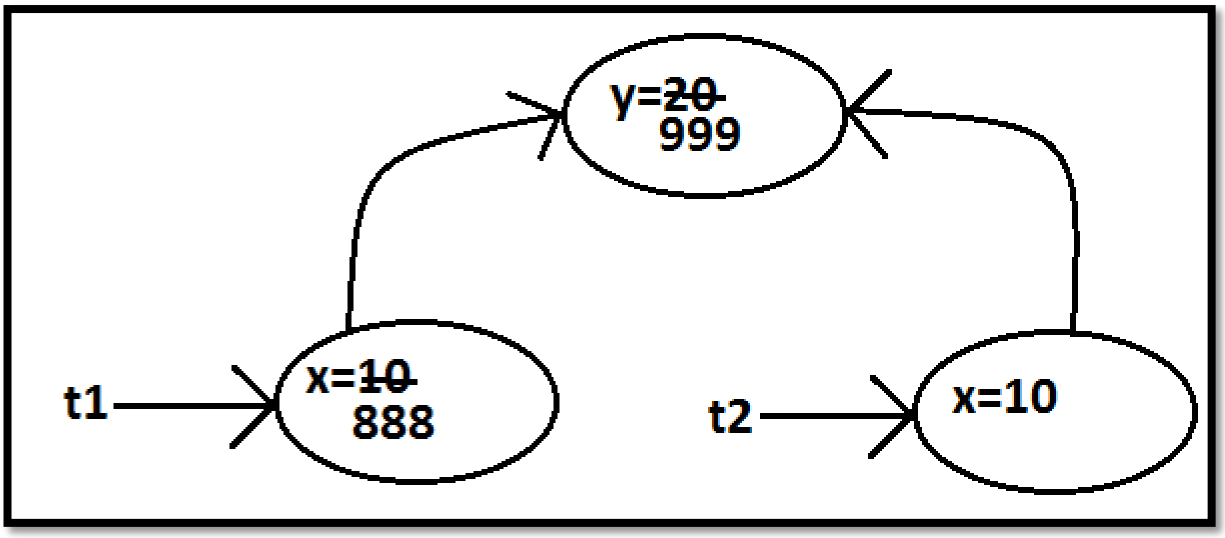
For the static variables it is not required to perform initialization explicitly, JVM will always provide default values.

Example:

```
class Test
{
   static String s;
   public static void main(String[] args)
   {
      System.out.println(s);//null
   }
}
```

Example:

```
class Test
{
   int x=10;
   static int y=20;
   public static void main(String[] args)
   {
      Test t1=new Test();
      t1.x=888;
      t1.y=999;
      Test t2=new Test();
      System.out.println(t2.x+"----"+t2.y);//10----999
   }
}
```



Static variables also known as class level variables or fields.

Local variables:

- Some times to meet temporary requirements of the programmer we can declare variables inside a method or block or constructors such type of variables are called local variables or automatic variables or temporary variables or stack variables.
- Local variables will be stored inside stack.
- The local variables will be created as part of the block execution in which it is declared and destroyed once that block execution completes. Hence the scope of the local variables is exactly same as scope of the block in which we declared.

Example 1:

```java
class Test
{
   public static void main(String[] args)
   {
      int i=0;
      for(int j=0;j<3;j++)
      {
         i=i+j;
      }
   }
}
```

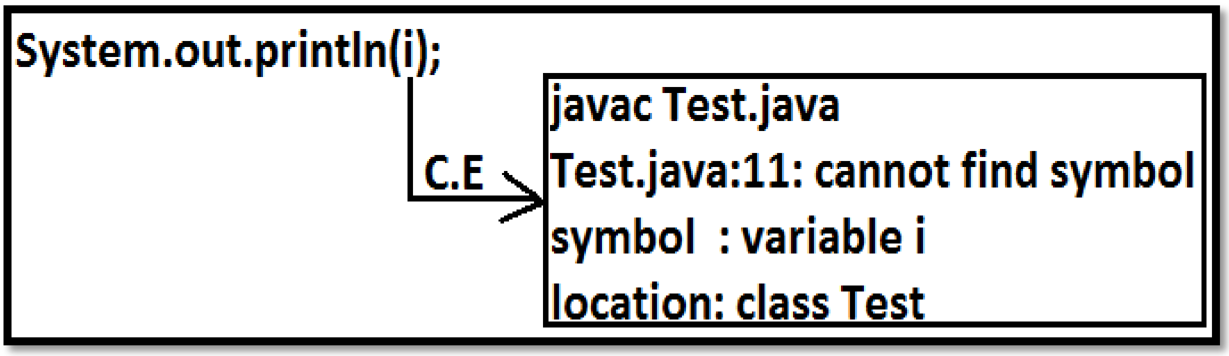
```
System.out.println(i+"----"+j);
```

C.E →

```
javac Test.java
Test.java:10: cannot find symbol
symbol  : variable j
location: class Test
```

Example 2:

```java
class Test
{
   public static void main(String[] args)
   {
      try
      {
         int i=Integer.parseInt("ten");
      }
      catch(NullPointerException e)
      {
      }
   }
}
```



```
System.out.println(i);
        |
        C.E
        └──> javac Test.java
             Test.java:11: cannot find symbol
             symbol  : variable i
             location: class Test
```

- The local variables will be stored on the stack.
- For the local variables JVM won't provide any default values compulsory we should perform initialization explicitly before using that variable.

Example:

```
class Test
{
    public static void main(String[] args)
    {
        int x;
        System.out.println("hello");//hello
    }
}
```

```
class Test
{
    public static void main(String[] args)
    {
        int x;
      System.out.println(x);//C.E:variable x might not have been initialized
    }
}
```

Example:

```
class Test
{
   public static void main(String[] args)
   {
      int x;
      if(args.length>0)
      {
         x=10;
      }
      System.out.println(x);
             //C.E:variable x might not have been initialized
   }
}
```

Example:

```
class Test
{
   public static void main(String[] args)
   {
      int x;
      if(args.length>0)
      {
         x=10;
      }
      else
      {
         x=20;
      }
      System.out.println(x);
   }
}
```

Output:

```
java Test x
10

java Test x y
10

java Test
20
```

- It is never recommended to perform initialization for the local variables inside logical blocks because there is no guarantee of executing that block always at runtime.
- It is highly recommended to perform initialization for the local variables at the time of declaration at least with default values.

Q. Consider the code

```
public class  Triangle
{
   static double area;
   int b=30,h=40;
   public static void main(String[] args)
   {
      double p,b,h;// Line-1
      if(area ==0)
      {
         b=3;
         h=4;
```

```
            p=0.5;
        }
        area=p*b*h;// Line-2
        System.out.println(area);
    }
}
```

What is the result?
A) 6.0
B) 3.0
C) Compilation fails at Line-1
D) Compilation fails at Line-2

Answer: D

<u>Note:</u> The only applicable modifier for local variables is final. If we are using any other modifier we will get compile time error.

<u>Example:</u>

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

        public int x=10; //(invalid)
        private int x=10; //(invalid)
        protected int x=10; //(invalid)    C.E: illegal start of expression
        static int x=10; //(invalid)
        volatile int x=10; //(invalid)
        transient int x=10; //(invalid)
        final int x=10;//(valid)
    }
}
```

Conclusions:

- For the static and instance variables it is not required to perform initialization explicitly JVM will provide default values. But for the local variables JVM won't provide any default values compulsory we should perform initialization explicitly before using that variable.
- For every object a separate copy of instance variable will be created whereas for entire class a single copy of static variable will be created. For every Thread a separate copy of local variable will be created.
- Instance and static variables can be accessed by multiple Threads simultaneously and hence these are not Thread safe but local variables can be accessed by only one Thread at a time and hence local variables are Thread safe.

- If we are not declaring any modifier explicitly then it means default modifier but this rule is applicable only for static and instance variables but not local variable.

Example:

```
class Test
{
   int[] a;
   public static void main(String[] args)
   {
      Test t1=new Test();
      System.out.println(t1.a);//null
      System.out.println(t1.a[0]);//R.E:NullPointerException
   }
}
```

Instance level:

Example 1:

```
int[] a;
System.out.println(obj.a);//null
System.out.println(obj.a[0]);//R.E:NullPointerException
```

Example 2:

```
int[] a=new int[3];
System.out.println(obj.a);//[I@3e25a5
System.out.println(obj.a[0]);//0
```

Static level:

Example 1:

```
static int[] a;
System.out.println(a);//null
System.out.println(a[0]);//R.E:NullPointerException
```

Example 2:

```
static int[] a=new int[3];
System.out.println(a);//[I@3e25a5
System.out.println(a[0]);//0
```

Local level:

Example 1:

```java
int[] a;
System.out.println(a); //C.E: variable a might not have been initialized
System.out.println(a[0]);
```

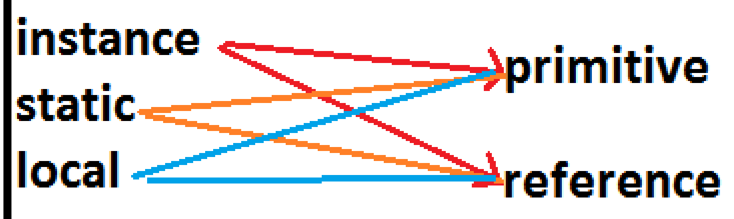
Example 2:

```java
int[] a=new int[3];
System.out.println(a);//[I@3e25a5
System.out.println(a[0]);//0
```

Note:
Once we create an array every element is always initialized with default values irrespective of whether it is static or instance or local array.

Note:

- Every variable in Java should be either instance or static or local.
- Every variable in Java should be either primitive or reference.
- Hence the following are the various possible combinations for variables

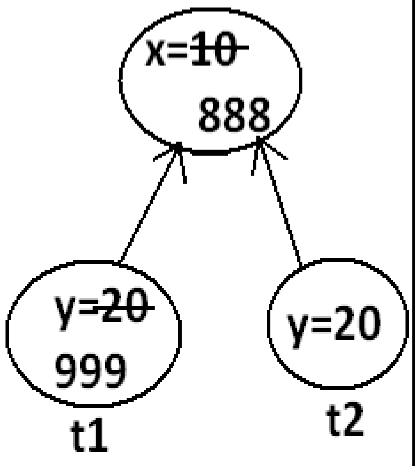


```java
class Test {
 int[]  a=new int[3];     // instance-reference
 static  int  x=20;         //static-primitive
 public static void main(String[] args) {
 String s="xyz";          //local-reference
  }
}
```

* Static is the modifier applicable for methods, variables and blocks.
* We can't declare a class with static but inner classes can be declaring as the static.
* In the case of instance variables for every object a separate copy will be created but in the case of static variables a single copy will be created at class level and shared by all objects of that class.

Example:

```
class Test{
static int x=10;
int y=20;
public static void main(String args[]){
Test t1=new Test();
t1.x=888;
t1.y=999;
Test t2=new Test();
System.out.println(t2.x+"....."+t2.y);
}
}
```



Output:

```
D:\Java>javac Test.java
D:\Java>java Test
888.....20
```

* Instance variables can be accessed only from instance area directly and we can't access from static area directly.
* But static variables can be accessed from both instance and static areas directly.

Q. Consider the following 4 declarations

1) int x=10;

2) static int x=10;

3)
```
public void m1(){
    System.out.println(x);
  }
```

```
4)        public static void m1(){
              System.out.println(x);
          }
```

Which of the following declarations are allowed within the same class simultaneously ?

a) 1 and 3
b) 1 and 4
c) 2 and 3
d) 2 and 4
f) 1 and 2
f) 3 and 4

Answer: a,c,d

Example:

```
class Test
{
int x=10;
public void methodOne(){
System.out.println(x);
}}
```

Output:

Compile successfully.

Example:

```
class Test
{
int x=10;
public static void methodOne(){
System.out.println(x);
}}
```

Output:

Compile time error.

```
D:\Java>javac Test.java
Test.java:5: non-static variable x cannot be referenced from a static context
System.out.println(x);
```

Example:

```
class Test
{
static int x=10;
public void methodOne(){
System.out.println(x);
}}
```

Output:
Compile successfully.

Example:
```
class Test
{
static int x=10;
public static void methodOne(){
System.out.println(x);
}}
```

Output:
Compile successfully.

e) 1 and 2

Example:
```
class Test
{
int x=10;
static int x=10;
}
```

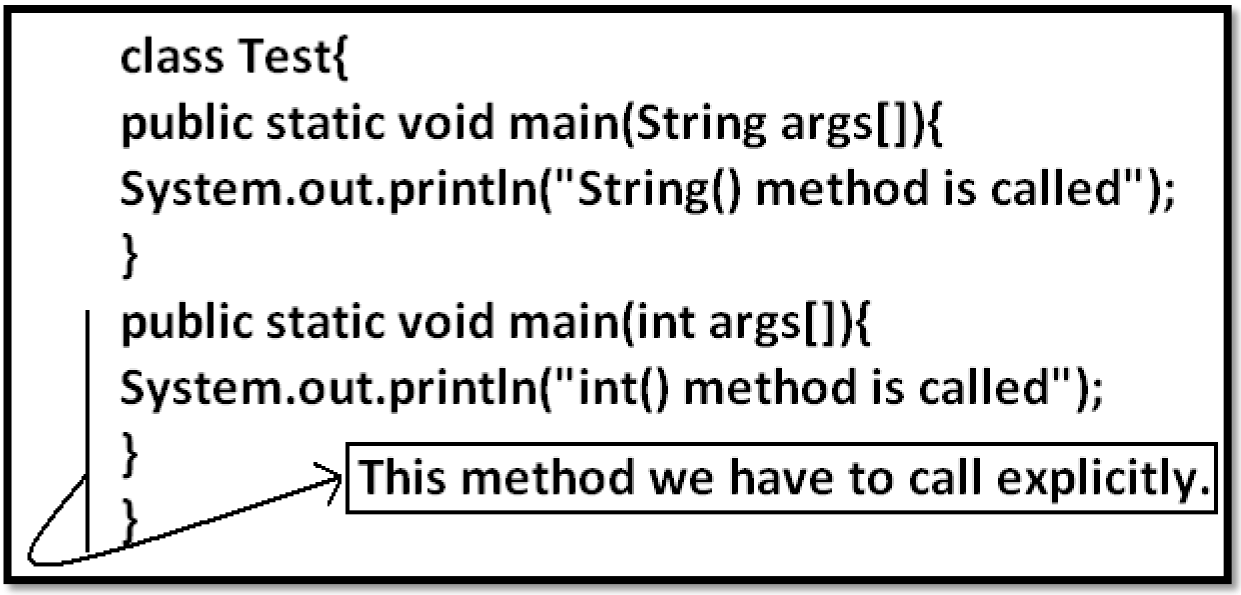
Output:
Compile time error.
```
D:\Java>javac Test.java
Test.java:4: x is already defined in Test
static int x=10;
```

Example:
```
class Test{
public void methodOne(){
System.out.println(x);
}
public static void methodOne(){
System.out.println(x);
}}
```

Output:
Compile time error.
```
D:\Java>javac Test.java
Test.java:5: methodOne() is already defined in Test
public static void methodOne(){
```
For static methods implementation should be available but for abstract methods implementation is not available hence static abstract combination is illegal for methods.

Overloading concept is applicable for static method including main method also.But JVM will always call String[] args main method .
The other overloaded method we have to call explicitly then it will be executed just like a normal method call .

<u>Example:</u>



```
class Test{
public static void main(String args[]){
System.out.println("String() method is called");
}
public static void main(int args[]){
System.out.println("int() method is called");
}
}
```

<u>Output :</u>

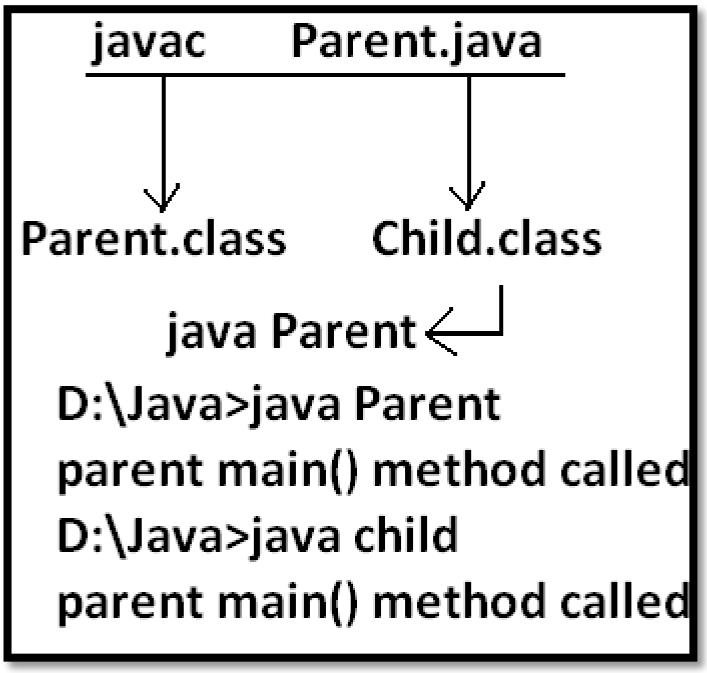
```
String() method is called
```

Inheritance concept is applicable for static methods including main() method hence while executing child class, if the child doesn't contain main() method then the parent class main method will be executed.

<u>Example:</u>

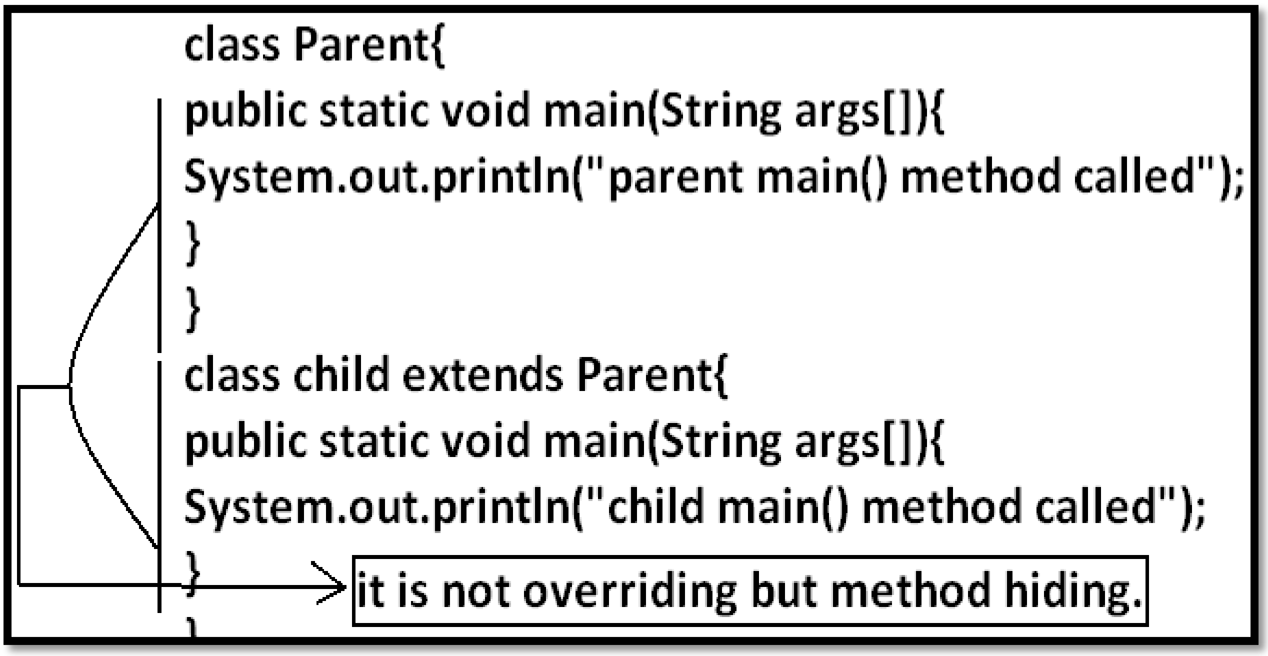
```
class Parent{
public static void main(String args[]){
System.out.println("parent main() method called");
}
}
class child extends Parent{
}
```

Output:



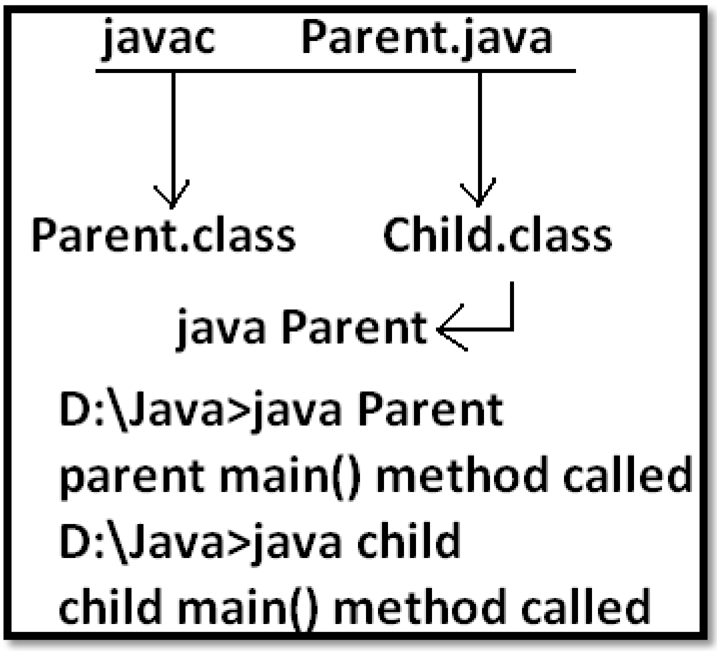
```
   javac     Parent.java
      |            |
      v            v
Parent.class    Child.class
        java Parent
 D:\Java>java Parent
 parent main() method called
 D:\Java>java child
 parent main() method called
```

Example:



```
class Parent{
public static void main(String args[]){
System.out.println("parent main() method called");
}
}
class child extends Parent{
public static void main(String args[]){
System.out.println("child main() method called");
}
}
```

Output:



```
D:\Java>java Parent
parent main() method called
D:\Java>java child
child main() method called
```

It seems to be overriding concept is applicable for static methods but it is not overriding it is method hiding.

Q. Consider the following code

```java
public class  Test
{
   String myStr="7007";
   public void doStuff(String s)
   {
      int myNum=0;
      try
      {
         String myStr=s;
         myNum=Integer.parseInt(myStr);
      }
      catch(NumberFormatException e)
      {
         System.err.println("Error");
      }
      System.out.println("myStr: "+myStr+" ,myNum: "+myNum);
   }
   public static void main(String[] args)
   {
      Test t = new Test();
      t.doStuff("9009");
   }
}
```

A) myStr: 9009 ,myNum: 9009
B) myStr: 7007 ,myNum: 7007
C) myStr: 7007 ,myNum: 9009
D) Compilation Fails

Answer: C

Explanation: In the above example, the variable myStr,which is declared inside try block is not available outside of try block. Hence while printing, instance variable myStr will be considered.

Q. Consider the following code

```
public class  Test
{
   public void doStuff(String s)
   {
      int myNum=0;
      try
      {
         String myStr=s;
         myNum=Integer.parseInt(myStr);
      }
      catch(NumberFormatException e)
      {
         System.err.println("Error");
      }
      System.out.println("myStr: "+myStr+" ,myNum: "+myNum);
   }
   public static void main(String[] args)
   {
      Test t = new Test();
      t.doStuff("9009");
   }
}
```

A) myStr: 9009 ,myNum: 9009
B) myStr: 7007 ,myNum: 7007
C) myStr: 7007 ,myNum: 9009
D) Compilation Fails

Answer: D

Explanation: myStr is local variable of try block and we cannot access outside of try block.

Q. Consider the code

```
class Test
{
   int x =10;
   static int y = 20;
   public static void main(String[] args)
   {
      Test t1 =new Test();
      Test t2 =new Test();
      t1.x=100;
      t1.y=200;
      t2.x=300;
      t2.y=400;
      System.out.println(t1.x+".."+t1.y+".."+t2.x+".."+t2.y);
   }
}
```

A) 100..400..300..400
B) 100..400..100..400
C) 200..400..300..400
D) 100..200..300..400

Answer: A

Q. Consider the following code

```
public class Test
{
   static int x;
   int y;
   public static void main(String[] args)
   {
      Test t1 = new Test();
      Test t2 = new Test();
      t1.x=3;
      t1.y=4;
      t2.x=5;
      t2.y=6;
      System.out.println(t1.x+":"+t1.y+":"+t2.x+":"+t2.y);
   }
}
```

What is the result?
A) 3:4:5:6
B) 3:4:3:6
C) 5:4:5:6
D) 3:6:4:6

Answer: C

Q. Consider the code

```
public class Test
{
   static int count=0;
   int i =0;
   public void modify()
   {
      while(i<5)
      {
         i++;
         count++;
      }
   }
   public static void main(String[] args)
   {
      Test t1 = new Test();
      Test t2 = new Test();
      t1.modify();
      t2.modify();
      System.out.println(t1.count+".."+t2.count);
   }

}
```

What is the result?
A) 10..10
B) 5..5
C) 5..10
D) Compilation Fails

Answer: A

Q. Consider the code

```
class Test
{
   int count;
   public static void display()
   {
      count++;//Line-1
      System.out.println("Welcome Visit Count:"+count);//Line-2
   }
   public static void main(String[] args)
   {
        Test.display();//Line-3
        Test.display();//Line-4
    }
}
```

What is the result?

A) Welcome Visit Count: 1
Welcome Visit Count: 2
A) Welcome Visit Count: 1
Welcome Visit Count: 1
C) Compilation Fails at Line-1 and Line-2
D) Compilation Fails at Line-3 and Line-4

Answer: C

Q. Consider the code

```
public class Test
{
    public static int x=100;
    public int y = 200;
    public String toString()
    {
        return y+":"+x;
    }
    public static void main(String[] args)
    {
        Test t1 = new Test();
        t1.y=300;
        System.out.println(t1);
        Test t2 = new Test();
        t2.x=300;
        System.out.println(t2);
    }

}
```

What is the result?
A) 300:100
200:300
B) 200:300
200:300
C) 300:0
0:300
D) 300:300
200:300

Answer: A

Q. Consider the code

```
class Demo
{
   int ns;
   static int s;
   Demo(int ns)
   {
      if(s<ns)
      {
         s=ns;
         this.ns=ns;
      }
   }
   void display()
   {
      System.out.println("ns = "+ns+"  s = "+s);
   }
}
public class Test
{
   static int x;
   int y;
   public static void main(String[] args)
   {
      Demo d1= new Demo(50);
      Demo d2= new Demo(125);
      Demo d3= new Demo(100);
      d1.display();
      d2.display();
      d3.display();
   }
}
```

A. ns = 50 s = 125
   ns = 125 s = 125
   ns = 0 s = 125

B. ns = 50 s = 125
   ns = 125 s = 125
   ns = 100 s = 125

C. ns = 50 s = 50
   ns = 125 s = 125
   ns = 0 s = 0

D. ns = 50 s = 125
ns = 125 s = 125
ns = 100 s = 100

Answer: A

Q. Consider the code

```
public class Test
{
    public static void main(String[] args)
    {
        int x =200;
        System.out.print(m1(x));
        System.out.print(" "+x);
    }
    public static int m1(int x)
    {
        x=x*2;
        return x;
    }
}
```

What is the result?

A) 400 200
B) 200 200
C) 400 400
D) Compilation Fails

Answer: A

Q. Consider the code

```
public class Test
{
    public static void main(String[] args)
    {
        try
        {
            int n=10;
            int d=0;
            int ans=n/d;
        }
        catch (ArithmeticException e)
        {
            ans=0;//Line-1
```

```
    }
    catch(Exception e)
    {
      System.out.println("Invalid Calculation");
    }
    System.out.println("Answer="+ans);//Line-2
  }
}
```

What is the result?

A) Answer=0
B) Invalid Calculation
C) Compiation Fails at Line-1
D) Compiation Fails at Line-2
E) Compiation Fails at Line-1 and Line-2

Answer: E

Q. Consider the code

```
public class Test
{
  char c;
  boolean b;
  float f;
  public void print()
  {
    System.out.println("c = "+c);
    System.out.println("b = "+b);
    System.out.println("f = "+f);
  }
  public static void main(String[] args)
  {
    Test t = new Test();
    t.print();
  }
}
```

What is the result?

A. c =
   b = false
   f = 0.0

B. c = null
   b = false
   f = 0.0

C. c = 0
   b = false
   f = 0.0

D. c =
   b = true
   f = 0.0

Answer: A

Passing Arguments:

If we pass primitive type to a method and within that method if we perform any changes then those changes won't be reflected to the caller. In this case a separate local copy will be created for the primitive variable in that method.

```java
public class Test
{
   public void m1(int i, int j)
   {
      i=i+10;
      j=j+10;
      System.out.println("Inside Method:"+i+".."+j);
   }
   public static void main(String[] args)
   {
      int x=100;
      int y =200;
      Test t = new Test();
      t.m1(x,y);
      System.out.println("After Completing Method:"+x+".."+y);
   }
}
```

Output
```
Inside Method:110..210
After Completing Method:100..200
```

Q. Consider the code

```java
public class Test
{
   public static void main(String[] args)
   {
      int x=200;
      System.out.print(m1(x));
```

```
        System.out.print(" "+x);
    }
    public static int m1(int x)
    {
        x= x*2;
        return x;
    }
}
```

What is the result?

A) 400 200
B) 200 200
C) 400 400
D) Compilation Fails

Answer: A

Q. Consider the following code

```
public class Test
{
    int i,j;
    public Test(int i,int j)
    {
        initialize(i,j);
    }
    public void initialize(int i,int j)
    {
        this.i = i*i;
        this.j=j*j;
    }
    public static void main(String[] args)
    {
        int i =3, j= 5;
        Test t = new Test(i,j);
        System.out.println(i+"..."+j);
    }
}
```

What is the result?

A) 9...25
B) 0...0
C) 3...5
D) Compilation Fails

Answer: C

If we pass object reference to a method and within the method if we perform any changes to the object state, then those changes will be reflected to the caller also. In this case just duplicate reference variable will be created and new object won't be created.

Eg 2:

```java
class Demo
{
   int x;
   int y;
};
public class Test
{
   public void m1(Demo d)
   {
      d.x=888;
      d.y=999;
   }
   public static void main(String[] args)
   {
      Demo d1 = new Demo();
      d1.x=10;
      d1.y=20;
      Test t = new Test();
      t.m1(d1);
      System.out.println(d1.x+"..."+d1.y);
   }
}
```

Output: 888...999

In the above example we are passing Demo object reference as argument to method m1(). Inside method m1(),we are changing the state of the object. These changes will be reflected to the caller.

Practice Question:

```java
class Product
{
   double price;
}
public class Test
{
   public void updatePrice(Product p,double price)
   {
      price=price*2;
      p.price=p.price+price;
   }
   public static void main(String[] args)
```

```
    {
        Product prt = new Product();
        prt.price=200;
        double newPrice=100;

        Test t = new Test();
        t.updatePrice(prt,newPrice);
        System.out.println(prt.price+"...."+newPrice);
    }
}
```

What is the result?
A) 200.0....100.0
B) 400.0....400.0
C) 400.0....100.0
D) Compilation Fails

Answer: C

Explanation:
In the above example, we are passing Product object reference as argument to updatePrice() method and within the method we are changing the state of object. These changes will be reflected to the caller.

Q. Consider the code

```
public class Test
{
    int x;
    int y;
    public void doStuff(int x,int y)
    {
        this.x=x;
        y=this.y;
    }
    public void print()
    {
        System.out.print(x+":"+y+":");
    }
    public static void main(String[] args)
    {
        Test t1=new Test();
        t1.x=100;
        t1.y=200;

        Test t2 = new Test();
        t2.doStuff(t1.x,t1.y);
        t1.print();
```

```
        t2.print();
    }
}
```

What is the result?

A) 100:200:100:0:
B) 100:0:100:0:
C) 100:200:100:200:
D) 100:0:200:0:

Answer: A

Q. Consider the code

```
public class Vowel
{
    private char ch;
    public static void main(String[] args)
    {
        char ch1='a';
        char ch2=ch1;
        ch2='e';

        Vowel obj1= new Vowel();
        Vowel obj2=obj1;
        obj1.ch='i';
        obj2.ch='o';

        System.out.println(ch1+":"+ch2);
        System.out.println(obj1.ch+":"+obj2.ch);
    }
}
```

What is the result?

A. a:e
   o:o

B. e:e
   i:o

C. a:e
   i:o

D. e:e
   o:o

Answer: A