```
Polymorphism
++++++++++++
Same name with different form is the concept of polymorphism
      eg: abs(int),abs(float),abs(long),.....

We can use the parent reference to hold chid class object
   List l  =new ArrayList();
   List l  =new LinkedList();
   List l = new Vector();
   List l = new Stack();

 Polymorphism
 ============
 a. CompileTime/static/early binding
      a. Overloading
      b. methodhiding

 b. Runtime/dynamic/latebinding
      a. Overriding

Pillars of oops
 a. inheritance(code-reusability)
 b. polymorphism(flexiblity)
 c. encapsulation(code security[Abstraction + datahiding])

eg#1.
public class Sample
{
                        //1
      public void methodOne(int i){
            System.out.println("general method");
      }

                        //0..n
      public void methodOne(int... i){
            System.out.println("var arg method");
    }
      public static void main(String[] args)
      {
            Sample t =new Sample();
            t.methodOne();//var-arg
            t.methodOne(10,20);//var-arg
            t.methodOne(10);//general method
      }
}
Note: In general var-args method will get less priority that is if no other methods
matches only then var-arg method will get a chance
 for execution, it is almost same as default casse of switch statement.

eg#2.
class Animal{}
class Monkey extends Animal{}
public class Sample
{
      //Overloaded -> Compiler
      public void methodOne(Animal a){
            System.out.println("Animal version");
      }
      //Overloaded -> Compiler
      public void methodOne(Monkey m){
            System.out.println("Monkey version");
      }
      public static void main(String[] args)
      {
            Sample s =new Sample();


            Animal a =new Animal();
            s.methodOne(a);//Animal version

            Monkey m = new Monkey();
            s.methodOne(m);//Monkey version

            Animal a1 =new Monkey();
            s.methodOne(a1);//Animal version
      }
}
```

Q>
Consider below code of main.java file:

```
public class main {
    static String main = "ONE";  //static variable

    public main() { //constructor
        System.out.println("TWO");
    }
    public static void main(String [] args) {//JVM expected main method
        main();
    }
    public static void main () {  //user-defined static method with a name main.
        System.out.println(main);
    }
}
```

Also consider below statements:
1. Code doesn't compile
2. Code compiles successfully
3. Only ONE will be printed to the console//Answer
4. Only TWO will be printed to the console
5. Both ONE and TWO will be printed to the console

Q>
Given code of Test.java file:

```
public class Test {
    public static void main(String[] args){
        args[1] = "Day!";//RE
        System.out.println(args[0] + " " + args[1]);
    }
}
```

And the commands:

```
javac Test.java
java Test Good
```

What is the result?
A. Good
B. Good Day!
C. Compilation Error

D. An Exception is thrown at runtime.//Answer

```
Test.main(new String[]{"Good"});
       |
       |
args[0] = "Good";
```

Q>

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

    public static void main(Integer[] args) {
        System.out.println("Integer");
    }

    public static void main(byte[] args) {
        System.out.println("byte");
    }
}
```

And the commands:

```
javac Test.java
java test 10
```

```
Test.main(new String[]{"10"})
       |
    main(String[] args)
```

What is the result?

A. Integer
B. String//Answer
C. byte
D. Compilation error
E. An Exception is thrown at RunTime

Q>

Given the code of Test.java file:

```
class Point {
    int x;
    int y;
    void assign(int x, int y) {
        x = this.x;
        this.y = y;
    }

    public String toString(){
        return "Point(" + x + ", " + y + ")";
    }
}

public class Test {
    public static void main(String[] args){
        Point p1 = new Point();
        p1 . x = 10 ;
        p1 . y = 20 ;
        Point p2 = new Point();
        p2.assign(p1.x, p1.y);
        System.out.println(p1.toString() + ";" + p2.toString());
    }
}
```

What will be the result of compiling and executing Test class?
 A. Point(10,20); Point(10,20)
 B. Point(10,20); Point(0,20); //Answer
 C. Point(0,20); Point(0,20);
 D. Point(0,20); Point(10,20);
 E. None of the other options

```
p1
x = 0,10
y = 0,20

p2
x= 0 ,
y= 0 , 20

local variable
 x= 10, 0
 y= 20

Point(10,20);Point(0,20)
```

Q>
Consider below code:

```
public class Counter {
    int count;

    private static void increment(Counter counter) {
        counter.count++;
    }

    public static void main(String[] args){
        Counter c1 = new Counter();
        Counter c2 = c1;
        Counter c3 = null;
        c2.count = 1000;
        increment(c2);
    }
}
```

On executing Counter class, how many Counter objects are created in the memory?
A. 1
B. 2
C. 3
D. 4

Answer: A

Q>

```
public class MainApp {
      private static void add(double d1, double d2) {
            System.out.println("double version: " + (d1 + d2));
      }

      private static void add(Double d1, Double d2) {
            System.out.println("Double version: " + (d1 + d2));
      }
      public static void main(String[] args) {
                  add(10.0, null);
      }
}
```
A. CompilationError
B. Double version: 10.0
C. double version: 10.0
D. An exception is thrown at runtime

Answer: D (NullPointerException because of 10.0 + null)

Q>
What will be the result of compiling and executing Test class?
```
public class Test {
    public static void main(String[] args) {
        Double [] arr = new Double[2];
        System.out.println(arr[0] + arr[1]);
    }
}
```
A. NullPointerException is thrown at Runtime
B. 0.0
C. Compilationerror
D. ClassCastException is thrown at runtime

Answer: A

Q>
What will be the result of compiling and executing Test class?
```
public class Test {
    static Boolean[] arr = new Boolean[1]; // arr[0] = null
    public static void main(String[] args) {
        if(arr[0]) {
            System.out.println(true);
        } else {
            System.out.println(false);
        }
    }
}
```
A. true
b. false
C. Compilation error
D. NullPointerException is thrown at runtime
E. ArrayIndexOutOfBoundsException is thrown at runtime

Answer: D(arr[0] = null, if(null) so NullPointerException)

Q>
What will be the result of compiling and executing Test class?
```
public class Test {
    public static void main(String[] args) {
        Boolean b = new Boolean("tRUe");
        switch(b) {
            case true:
                System.out.println("ONE");
            case false:
                System.out.println("TWO");
            default:
                System.out.println("THREE");
        }
    }
}
```
A. ONE
   TWO
   THREE

B. TWO
   THREE

C. THREE

D. None of the above options

Answer: D

Q>
What will be the result of compiling and executing Test class?
```
public class Test {
    public static void main(String[] args) {
        Boolean b1 = new Boolean("tRuE");
        Boolean b2 = new Boolean("fAlSe");
        Boolean b3 = new Boolean("abc");
        Boolean b4 = null;
        System.out.println(b1 + ":" + b2 + ":" + b3 + ":" + b4);
    }
}
```
A. falsefalsefalsenull
B. truefalsefalsenull
C. falsefalsetruenull
D. Compilation error

Answer: B

Note: Any data other than case insenitive value of true is regarding as false for boolean wrapper class type.

Q>
```
public class Test {
    public static void main(String[] args) {
        m(1);
    }

    private static void m(Object obj) {
        System.out.println("Object version");
    }

    private static void m(Number obj) {
        System.out.println("Number version");
    }

    private static void m(Double obj) {
        System.out.println("Double version");
    }
}
```

A. Compilation error
B. Object version
C. Number version

D. Double version

Answer: C

Q>
For the given code what is the output?
```
   int x=100;
   int a=x++;
   int b= ++x;
   int c= x++;
   int d= (a<b) ? (a<c) ? a: (b<c)? b: c :x;
   System.out.println(d);
```

A. 100
B. 101
C. 102
D. 103
E. compilation fails

```
a = 100
x = 101,102,103
b = 102
c = 102

d=(100<102) ? (100<102) ? 100
d= 100
System.out.println(100);
```

Q>
```
class Alpha {
      int ns;static int s;
      Alpha(int ns) {
            if (s < ns) {
                  s = ns;
                  this.ns = ns;
              }
      }
      void doPrint() {System.out.println("ns = " + ns + " s=" + s);}
}
public class DemoApp {
      public static void main(String[] args) {
            Alpha a1 = new Alpha(50);
            Alpha a2 = new Alpha(125);
            Alpha a3 = new Alpha(100);
            a1.doPrint();
            a2.doPrint();
            a3.doPrint();
      }
}
```
Options
```
A. ns =50   s=125
   ns =125  s=125
   ns =100  s=125

B. ns =50   s=125
   ns =125  s=125
   ns =0    s=125

C. ns =50   s=50
```

```
   ns =125  s=125
   ns =100  s=100

D. ns =50   s=50
   ns =125  s=125
   ns =0    s=125

Answer:B


Q>
public class DemoApp {
      static int count = 0;int i = 0;
      public void changeCount() {
            while (i < 5) {
                  i++;
                  count++;
            }
      }
      public static void main(String[] args) {
            DemoApp demoApp1 = new DemoApp();
            DemoApp demoApp2 = new DemoApp();
            demoApp1.changeCount();
            demoApp2.changeCount();
            System.out.println(DemoApp.count + ":" + DemoApp.count);
      }
}
What is the output?
A. 10: 10
B. 5: 5
C. 5: 10
D. Compilation fails

count = 0,1,2,3,4,5,6,7,8,9,10

demoApp1
  i =0
  i =1,2,3,4,5

demoApp2
  i =0
  i =1,2,3,4,5

Answer: A


Q>
public class DemoApp{
      public static void main(String... args){
            if(arg[0].equals("hello") ? false : true)
                  System.out.println("success");
            else
                  System.out.println("failure");
      }
}
What is the output if the program is executed in the following style?
 DemoApp hello

A. success
```

B. failure
C. CE
D. ArrayIndexOutOfBoundsException
E. StringIndexOutOfBoundsException

Answer:B

++++++++++
Overriding
++++++++++
  In case of overriding, reference type is dummy and runtime object will play a vital role.
  In case of overriding, methodnames and arguments must be same , that is method signature is same.
  While overriding the parent class method in child class, return type need not be same.

Example
```
class Parent{
      public Object methodOne(){
            return null;
      }
}
class Child extends parent{
      public String methodOne(){
            return null;
      }
}
```

Example
```
class Parent{
      public String methodOne(){
            return null;
      }
}
class Child extends Parent
{
      public Object methodOne(){
            return null;
      }
}
```
co-variant type is from from Child to Parent, not from Parent to child.
It is appicable only for Object types.

Example
```
class Parent{
      public int methodOne(){
            return 0;
      }
}
class Child extends Parent
{
      public float methodOne(){
            return 10.5f;
      }
}
```

co-variant concept is not applicable for primtive types,it is applicable only Object types.

example
========
```
class Parent{
      private void methodOne(){}
}
class Child extends Parent{
      private void methodOne(){}
}
```

private methods won't be inherited,so these methods are not overriden methods they are specialized methods.

example
======
```
class Parent{
      public final void m1(){}
}
class Child extends Parent{
      public void m1(){}
}
```
Parent class final methods we can't override in child class.

example
=======
```
class Parent{
      public void m1(){}
}
class Child extends Parent{
      public final void m1(){}
}
```
Parent class non-final methods, we can mark as final in child class.

example
======
```
 public class Parent{
      public abstract void methodOne();
 }
 class Child extends Parent{
      public void methodOne(){}
 }
```

example
=======
```
 public class Parent{
      public  void methodOne(){}
 }
 abstract class Child extends Parent{
      public abstract void methodOne();
 }
```

Note: This feature is used to stop the implementation of parent class to its child classes.

example
=======

```
 class Parent{
      public static void m1(){}
 }
 class Child extends Parent{
      public void m1(){}
 }
Parent class static methods can't be made as non-static for child classes.

example
=======
class Parent{
      public  void m1(){}
}
class Child extends Parent{
      public static void m1(){}
}
Parent class non-static methods can't be made as static for child classes.

example
=======
class Parent{
      //Method-hiding(compiler)
      public static void m1(){System.out.println("parent");}
}
class Child extends Parent{
      public static void m1(){System.out.println("child");}
}
Parent p = new Child();
 p.m1();//parent
```