Inheritance

1. Find Output

```
class Base {
    public void show() {
      System.out.println("Base::show() called");
    }
}

class Derived extends Base {
    public void show() {
      System.out.println("Derived::show() called");
    }
}

public class Main {
    public static void main(String[] args) {
        Base b = new Derived();;
        b.show();
    }
}
```

Output

(A) Derived::show() called
(B) Base::show() called
c) compilation error

d) Runtime error

Answer: (A)

Explanation: In the above program, b is a reference of Base type and refers to an abject of Derived class.

In Java, functions are virtual by default. So the run time polymorphism happens and derived fun() is called.

2. Find Output

```
class Base {
   final public void show() {
      System.out.println("Base::show() called");
   }
}

class Derived extends Base {
   public void show() {
      System.out.println("Derived::show() called");
   }
}

class Main {
   public static void main(String[] args) {
       Base b = new Derived();
       b.show();
   }
}
```

Output

(A) Base::show() called
(B) Derived::show() called
(C) Compiler Error
(D) Runtime Error

Answer: (C)

Explanation: Final methods cannot be overridden.

3. Find Output

```
class Base {
    public static void show() {
        System.out.println("Base::show() called");
    }
}

class Derived extends Base {
    public static void show() {
        System.out.println("Derived::show() called");
    }
}

class Main {
    public static void main(String[] args) {
        Base b = new Derived();
        b.show();
    }
}
```

Output

(A) Base::show() called
(B) Derived::show() called
(C) Compiler Error
D) runtime error

Answer: (A)

Explanation: when a function is static, runtime polymorphism doesn’t happen.

4. Find Output

Which of the following is true about inheritance in Java?

1) Private methods are final.

2) Protected members are accessible within a package and inherited classes outside the package.

3) Protected methods are final.

4) We cannot override private methods.

Output

(A) 1, 2 and 4
(B) Only 1 and 2
(C) 1, 2 and 3
(D) 2, 3 and 4

Answer: (A)

5. Find Output

```
class Base {
   public void Print() {
      System.out.print("Base");
   }
}

class Derived extends Base {
   public void Print() {
      System.out.print("Derived");
   }
}

class Main{
   public static void DoPrint( Base o ) {
      o.Print();
   }
   public static void main(String[] args) {
      Base x = new Base();
      Base y = new Derived();
      Derived z = new Derived();
      DoPrint(x);
      DoPrint(y);
      DoPrint(z);
   }
}
```

Output

a) BaseDerivedDerived
b) BaseBaseDerived
c) BaseDerivedBase
d) Compiler Error

Ans: a

6. Find Output

```
class Base {
    public void foo() { System.out.println("Base"); }
}

class Derived extends Base {
    private void foo() { System.out.println("Derived"); }
}

public class Main {
    public static void main(String args[]) {
        Base b = new Derived();
        b.foo();
    }
}
```

Output

(A) Base
(B) Derived
(C) Compiler Error
(D) Runtime Error

Answer: (C)

Explanation: It is compiler error to give more restrictive access to a derived class function which overrides a base class function.

7. Find Output

Which of the following is true about inheritance in Java.

1) In Java all classes inherit from the Object class directly or indirectly. The Object class is root of all classes.

2) Multiple inheritance is not allowed in Java.

3) Unlike C++, there is nothing like type of inheritance in Java where we can specify whether the inheritance is protected, public or private.

Output

(A) 1, 2 and 3
(B) 1 and 2
(C) 2 and 3
(D) 1 and 3

Answer: (A)

8. Find Output

```
class Grandparent {
   public void Print() {
      System.out.print("Grandparent's Print() ");
   }
}

class Parent extends Grandparent {
   public void Print() {
      System.out.print("Parent's Print() ");
   }
}

class Child extends Parent {
   public void Print() {
      super.super.Print();
      System.out.print("Child's Print() ");
   }
}

public class Main {
   public static void main(String[] args) {
      Child c = new Child();
      c.Print();
   }
}
```

Output

(A) Compiler Error in super.super.Print()

(B) Grandparent's Print() Parent's Print() Child's Print()

(C) Runtime Error

d) Parent's Print() Child's Print()

Ans) a

Explanation: In Java, it is not allowed to do super.super. We can only access Grandparent’s members using Parent.

9. Find Output

```
final class Complex {

    private final double  re;
    private final double  im;

    public Complex(double re, double im) {
        this.re = re;
        this.im = im;
    }

    public String toString() {
        return "(" + re + " + " + im + "i)";
    }
}

class Main {
  public static void main(String args[])
  {
      Complex c = new Complex(10, 15);
      System.out.println("Complex number is " + c);
  }
}
```

Output

(A) Complex number is (10.0 + 15.0i)

(B) Compiler Error

(C) Complex number is SOME_GARBAGE

(D) Complex number is Complex@8e2fb5

Ans) a

10. Find Output

```
class A
{
   public A(String s)
   {
      System.out.print("A");
   }
}

public class B extends A
{
   public B(String s)
   {
      System.out.print("B");
   }
   public static void main(String[] args)
   {
      new B("C");
      System.out.println(" ");
   }
}
```

Output

a) Compilation error
b) Runtime error
c) AB
d) BA

Ans) a

Explanation: The implied super() call in B's constructor cannot be satisfied because there isn't a no-arg constructor in A. A default, no-arg constructor is generated by the compiler only if the class has no constructor defined explicitly.

11. Find Output

```
class Clidder
{
   private final void flipper()
   {
      System.out.println("Clidder");
   }
}

public class Clidlet extends Clidder
{
   public final void flipper()
   {
      System.out.println("Clidlet");
   }
   public static void main(String[] args)
   {
      new Clidlet().flipper();
   }
}
```

Output

a) Clidlet
b) Compilation error
c) Runtime error
d) Clidlet Clidder

Ans) a

Explanation: Although a final method cannot be overridden, in this case, the method is private, and therefore hidden. The effect is that a new, accessible, method flipper is created. Therefore, no polymorphism occurs in this example, the method invoked is simply that of the child class, and no error occurs.

12. Find Output

```
class Alpha
{
   static String s = " ";
   protected Alpha()
   {
      s += "alpha ";
   }
}
class SubAlpha extends Alpha
{
   private SubAlpha()
   {
      s += "sub ";
   }
}

public class SubSubAlpha extends Alpha
{
   private SubSubAlpha()
   {
      s += "subsub ";
   }
   public static void main(String[] args)
   {
      new SubSubAlpha();
      System.out.println(s);
   }
}
```

Output

a) alpha subsub

b) Compilation error

c) alpha subsub sub

d) Runtime error

Ans) a

Explanation: SubSubAlpha extends Alpha! Since the code doesnt attempt to make a SubAlpha, the private constructor in SubAlpha is okay.

13. Find Output

```
public class A extends B
{
   public static String sing()
   {
      return "fa";
   }
   public static void main(String[] args)
   {
      A a = new A();
      B b = new A();
   System.out.println(a.sing() + " " + b.sing());
   }
}
class B
{
   public static String sing()
   {
      return "la";
   }
}
```

Output

a) fa la

b) Compilation error

c) fa fa

d) la la

ans) a

Explanation: B b = new A(); b is object of type B, and hence b.sing() refers to the method sing of class B

14. Find Output

```
class A
{
A()
{
System.out.print("a");
}
}
class B extends A
{
B()
{
System.out.print("b");
}
}

public class Main {
   public static void main(String[] args) {
       new B();
   }
}
```

Output

a) ab
b) Compilation error
c) Runtime error
d) B

Ans: a

15. Find Output

```
class A
{
A()
{
System.out.print("a");
}
}
class B extends A
{
B()
{
this();
System.out.print("b");
}
}

public class Main {
   public static void main(String[] args) {
       new B();
   }
}
```

Output

a) Compilation error
b) Runtime error
c) ab
d) B

Ans) a

16. Find Output

```
class A
{
A()
{
System.out.print("a");
}
}
class B extends A
{
B()
{
super();
System.out.print("b");
}
}

public class Main {
   public static void main(String[] args) {
       new B();
   }
}
```

Output

a) ab
b) Compilation error
c) Runtime error
d) B

Ans: a

17. Find Output

```
class A
{
A(int n)
{
System.out.print("a");
}
}
class B extends A
{
B()
{
super(10);
System.out.print("b");
}
}

public class Main {
   public static void main(String[] args) {
       new B();
   }
}
```

Output

a) ab
b) Compilation error
c) Runtime error
d) B

Ans: a

18. Find Output

```
class A
{
A(int n)
{
System.out.print("a");
}
}
class B extends A
{
B()
{
System.out.print("b");
}
}

public class Main {
   public static void main(String[] args) {
       new B();
   }
}
```

Output

a) ab
b) Compilation error
c) Runtime error
d) B

Ans: b

19. Find Output

```
class A
{
A()
{
super();
System.out.print("a");
}
}
class B extends A
{
B()
{
System.out.print("b");
}
}

public class Main {
   public static void main(String[] args) {
       new B();
   }
}
```

Output

a) ab
b) Compilation error
c) Runtime error
d) B

Ans: a

20. Find output

```
class A
{
A()
{
this();
System.out.print("a");
}
}
class B extends A
{
B()
{
System.out.print("b");
}
}

public class Main {
   public static void main(String[] args) {
       new B();
   }
}
```

output

a) ab
b) Compilation error
c) Runtime error
d) B

Ans: b

21. Find output

```
class Base
{
int value=0;
Base()
{
addValue();
}
void addValue()
{
value+=10;
}
int getValue()
{
return value;
}
}

class Derived extends Base
{
Derived()
{
addValue();
}
void addValue()
{
value+=20;
}
}

public class Test {
public static void main(String[] args)
{
Base b = new Derived();
System.out.println(b.getValue());
}}
```

output

a) 40
b) 30
c) 20
d) 10

Ans) a

Find output

output