MCQ Question on Inheritance

Question1: what will be the output of given code?

```
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 ();
   }
}
```

Options

(A) Derived::show() called

(B) Base::show() called

Answer: (A)

Explanation: In the above program, b is a reference of Base type and refers to an object of Derived class. In Java, functions are virtual by default. So the run time polymorphism happens and derived fun() is called.

Question2: what will be the output of given code?

```
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 ();
   }
}
```

Options

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

Answer: (C)
Explanation: Final methods cannot be overridden. See the compiler error

Question 3: what will be the output of given code?

```
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();
   }
}
```

Options

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

Answer: (A)
Explanation: Like C++, when a function is static, runtime polymorphism doesn't happen

Question 4: what will be the output of given code?

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

Options

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

Answer: (A)

Question5: what will be the output of given code?

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

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

Answer: (A)

Explanation: We create an object of type Base and call DoPrint(). DoPrint calls the print function and we get the first line.DoPrint(y) causes the second line of output. Like C++, assigning a derived class reference to a base class reference is allowed in Java. Therefore, the expression Base y = new Derived() is a valid statement in Java. In DoPrint(), o starts referring to the same object as referred by y. Also, unlike C++, functions are virtual by default in Java. So, when we call o.print (), the print() method of Derived class is called due to run time polymorphism present by default in Java.
DoPrint(z) causes the third line of output, we pass a reference of Derived type and the print() method of Derived class is called again. The point to note here

is: unlike C++, object slicing doesn't happen in Java. Because non-primitive types are always assigned by reference.

Question 6: what will be the output of given code?

```
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();
   }
}
```

Options

(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.

Question 7: Imagine you define the My Interface interface as shown in the following code snippet?

```
interface MyInterface {
 //complete the code here
 final int mynum = 70;
}
```

Which of the following options will lead to compilation error?

A. public final void myMethod(); B. protected void myMethod();
C. public void myMethod(); D. private abstract void myMethod()

Answer: The correct options are A, B, and D.
Explanation: The following are the invalid method declarations in an interface:

```
public final void myMethod();
private abstract void myMethod();
protected void myMethod();
```

A method within an interface cannot be marked as final, as final modifier disallows overriding of a method and the methods declared within an interface do not have method body. Therefore, the final modifier cannot be used as method declarations within an interface. Within an interface, you cannot declare a method using both private and abstract as the private class members can be accessed only within the class in which they are declared. Therefore, a method declared within an interface is declared as public and not as private or protected. As a result, the option A, B, and D will lead to compilation error and are the correct answers

Question 8: Imagine you write the following lines of code in your program:

```
class QuesSuper
{
 public int mynum=0;
 public QuesSuper(String str) {
 mynum=10;
 }
}
public class QuesSub extends QuesSuper {
 public QuesSub(String str) {
 mynum=20;
 }
 public static void main(String args[]) {
 QuesSub sub= new QuesSub("Suchita");
 System.out.println(sub.mynum);
 }
}
```

What will be the output after the preceding program is compiled and executed?

A. The program will compile successfully and 20 will be displayed as output.
B. The program will lead to compile time error.
C. The program will compile successfully and 10 will be displayed as output.
D. The program will compile successfully and 0 will be displayed as output.

The correct option is B.

Explanation: In the preceding program, the main() method of the QuesSub class invokes the constructor while creating an instance of the QuesSub class. Now before invoking constructor of the subclass constructor of the superclass is invoked. If constructor of the superclass is explicitly declared, then it is invoked else the default constructor is invoked. However in the above program constructor is explicitly declared with a parameter in QuesSuper and the constructor of QuesSub class is not passing any value to it. Therefore the above program will lead compilation error and hence the correct option is B.

Question 9: Imagine you want to clear your concept of nested classes and so you create a program containing nested and static classes. Consider that you have created the following program

```
public class Ques43 {
 public static void main(String args[]) {
 TestOuter o = new TestOuter();
 TestOuter.TestInner i = o.new TestInner();
 TestOuter.TestStaticInner inner = new TestOuter.TestStaticInner();
 }
}
class TestOuter {
 static int num1 = 100;
 TestOuter() {
 System.out.print("Welcome to the outer class" + " ");
 }
 class TestInner {
 TestInner() {
 System.out.print(TestOuter.num1 + " ");
 }
```

```
 }
 static class TestStaticInner {
 static int staticnum = 200;
 TestStaticInner() {
 System.out.print(staticnum + " ");
 }
 }
}
```

What will be the output after you compile and execute the preceding program?

A. The program compiles successfully and displays "Welcome to the outer class 100 200" as output.
B. The program compiles successfully and displays "Welcome to the outer class 200 100" as output.
C. The program compiles successfully and displays "Welcome to the outer class 100" as output.
D. The program compiles successfully and displays "Welcome to the outer class 200" as output.

The correct option is A.

Explanation: The first statement in the main method creates an instance of the TestOuter class and then the second statement instantiates the TestInner class. The instantiation of the TestInner class is done by using the outer class instance as the TestInner class is a non-static class. Finally an instance of the TestStaticInner class is created without using the instance of the TestOuter class as the TestStaticInner class is a static class. As a result the correct option is A

Question 10 : Imagine you are working a Java programmer in the ABC Company and write the following program:

```
public class Ques48
{public static void main(String[] args) {
   Vehicle v = new Car();
    System.out.print(v.getVehicle().getClass().getName() + " ");
   System.out.print(v.getVehicle().getName());
  }
}
class Vehicle {
 public Vehicle getVehicle() {
```

```
 return this;
 }
 public String getName() {
 return "Vehicle";
 }
}
class Car extends Vehicle {
 public Vehicle getVehicle() {
 return this;
 }
 public String getName() {
 return "Car";
 }
}
```

What will be the output after you compile and execute the preceding program?

A. The program will lead to compilation errors as the Car class overloads the getVehicle method by changing its return type.
B. The program will compile successfully and display “com.kogent.Car Car” as output.
C. The program will compile successfully but lead to runtime error.
D. The program will lead to compilation error at Vehicle v = new Car();

The correct option is B.

Explanation: The program will not lead to any compilation error as the getVehicle() and getName() are examples of method overriding and not overloading. Therefore the options A and D are incorrect.The code v.getVehicle() will return an instance of the Car class and its class name is com.kogent.Car, therefore the correct option is B

Question 11: Jerry is working as a Java Developer in XYZ Solution System. He writes the following classes in a program?

```
class SuperClass
 {
   public static void method1()
  { System.out.println("Static method in super  class");
  }
}
class SubClass extends SuperClass
```

```
{
 public static void method1()
 { System.out.println("Static method in sub class"); // line 1
 }
 public void method1()
  { System.out.println("Non-static method in subclass");
    // line 2
  }
}
```

Which of the following options are true?

A. If line 1 and line 2 are commented, then class SubClass will compile.
B. If line 1 is commented, then class SubClass will compile.
C. If line 2 is commented, then class SubClass will compile.
D. None of the above options are true

Options A and C are correct

Explanation: The static method (method1() in SubClass) cannot be overridden by a non-static method (method1 in SubClass) and vice versa

Q 12. Mr. John is working in XYZ Company Ltd. He tries to compile and run the following program

```
public class ObjectTest {
 public static void main (String args[ ] )
 {
    X obj1 = new Z( );
    Y obj2 = (Y) obj1;
   System.out.println(obj1.method1( ) );
    System.out.println(obj2.a );
  }
}
class X
{ int a = 100;
  int method1( )
  { return a;
  }
}
class Y extends X
{ int a = 200; int method1( )
  { return a;
   }
```

```
 }
class Z extends Y
{ int a = 300;
  int method1( )
   { return a;
   }
 }
```

What will happen when he compiles and runs the preceding program?

A. The program displays “300” followed by “300” as an output.

B. The program displays “200” followed by “200” as an output.

C. The Class cast exception is generated at runtime.

D. The program displays “300” followed by “200” as an output.

Option D is correct

Explanation: Variable is accessed depending upon the class on which object is created, therefore, the obj2.a prints “200” as an output. The method is accessed depending upon the actual class of the object that is Referenced by the variable, therefore, the obj1.method1() prints “300” as an output.

Question 13: Mr. David is working in XYZ Software Pvt. Ltd. He tries to compile and run the following program

```
class X1 {
 void display() {
 System.out.print("display() method of X1 class");
 }
}
class X2 extends X1 {
 void display() {
 System.out.print("display() method of X2 class");
 }
}
class X3 extends X2 {
 void display() {
 System.out.print("display() method of X3 class");
 }
 public static void main(String arg[]) {
   X1 x1 = new X1();
   X2 x2 = new X2();
   X3 x3 = new X3();
```

```
  X1 obj;
   obj = x1;
  obj.display();
  obj = x2;
  obj.display();
  obj = x3;
  obj.display();
 }
}
```

What will happen when he compiles and runs the preceding program? Choose more than one options, if correct.

A. The program displays “display() method of X1 class” as an output.
B. The program displays “display() method of X3 class” as an output.
C. The program displays “display() method of X2 class” as an output.
D. The program displays “display() method of X1 class” followed by “display() method of X2 class”
followed by “display() method of X3 class” as an output

Option D is correct.

Explanation: In the preceding program, X1 is superclass of X2, and X2 is superclass of X3. All the class (X1,
X2, and X3) overrides the display() method. Therefore, the display() method is invoked based on the object
type of class. For example:

```
X1=obj;
obj=x1;
obj.display();
```

Here, in the preceding code, the reference variable obj of type X1; therefore, display () method is called on X1 class

Question 14: Akbar works as a software developer in AAA Company. He tries to compile and run the following program:

```
class Base1
{
 Base1()
  { print();
  }
 void print()
  { System.out.println("Base1");
  }
```

```
}
class Base2 extends Base1
 {
   int number = Math.round(7.4f);
 public static void main(String args[])
 {
    Base1 base = new Base2();
    base.print();
 }
 void print()
  {System.out.println (number);
  } //1
}
```

What will happen when he compiles and runs the preceding program?

A. Program displays “Base1” followed by”7” as an output.
B. Program displays “Base1” followed by “Base1” as an output.
C. Program displays”7” followed by “7” as an output.
D. Program displays “0” followed by “7” as an output.

Option D is correct.

Explanation: The Base2 class extends the Base1 class. Both the classes have the overridden method print (). Here, when an object (base) of class Base2 is created, this object calls Base1's constructor and then invokes
The print () method. Now, which print() method is called (Base1 or Base2), here base object is created by Base2 class. Therefore, the Base2's print () method is called at //1. At this point, variable number is not initialized (because we are still in Base1 class), so the default value of number (0) is printed and then final value of number (7) is printed

Question 15: what will be the output of given code?

```
class Base {
 public void doMethod1() {
 System.out.println("non static doMethod1 in Base class");
 }
 public static void doMethod2() {
 System.out.println("static doMethod2 in Base class");
 }
}
class Sub extends Base {
 public void doMethod1() {
```

```
 System.out.println("non static doMethod1 in Sub class");
 }
 public static void doMethod2() {
 System.out.println("static doMethod2 in Sub class");
 }
}
class SCJPQ05 {
 public static void main(String[] args) {
 Base obj1 = new Base(); //1
 obj1.doMethod1();
 obj1.doMethod2();
 obj1 = new Sub();
 obj1.doMethod1(); //2
 obj1.doMethod2 ();
 }
}
```

What will happen when she compiles and runs the preceding program?

A. non static doMethod1 in Base class
static doMethod2 in Base class
non static doMethod1 in Sub class
static doMethod2 in Sub class

B. non static doMethod1 in Base class
static doMethod2 in Base class
non static doMethod1 in Sub class
static doMethod2 in Base class

C. non static doMethod1 in Base class
static doMethod2 in Base class
non static doMethod1 in Base class
static doMethod2 in Sub class

D. Program generates compile-time error

Option B is correct.

Explanation: The object obj1 of the Base class is created at //1 and calls doMethod1() and doMethod2() of
Base class. After calling doMethod1() and doMethod2() of Base class, the object obj1 is reference to Sub class

at //2, but the class type is Base. This object is then called doMethod1() of Sub class and doMethod2() of Base
class (here, doMethod2 () is called on Base class rather than Sub class because static methods are not override.

Question 16: what will be the output of given code?

```
class SuperClass {
 private int i;
 public void method1() { }
 public void method2() { }
}
class SubClass extends SuperClass {
 public int j;
 public void method2() { }
}
public class Test {
 public static void main (String args []) {
   SuperClass obj1 = new SuperClass(); //1
  SubClass obj2 = new SubClass(); //2
  obj2.method1 (); //3
  obj1.j=10; //4
  obj1.method2 (); //5
  obj2.i=50; //6
  }
 }
```

Which of the following statements are invalid in the main() method?
A. Statements at //4 and //6 are invalid statements.
B. Statements at //1, //4 and //6 are invalid statements.
C. Statements at //1, //2, //5, and //6 are invalid.
D. All statements are invalid.
Option A is correct.
Explanation: The statement obj1.j=10 is invalid, because the object obj1 is created on Superclass and the instance variable j is not available in this class. Similarly, the statement obj2.i=50 is invalid because object
obj2 is created on Subclass and the instance variable i is not available in this class.

Question 17: what will be the output of given code?

```
public class TestClass extends SuperClass {
 public static void main(String[] args) {
 TestClass object1 = new TestClass();
   object1.printMsg();
 }
 public void printMsg() {
 // 1
 }
}
class SuperClass {
 ABC objABC = new ABC();
}
class ABC {
 String str = "Java World!";
}
```

Which of the following statements are used at //1 to print the message Java World!?

A. System.out.println (this.objABC.str); B. System.out.println (objABC.str);
C. System.out.println (str); D. System.out.println (ABC.str);

Options A and B are correct.

Explanation: Within the main () method, an object (object1) of the TestClass is created, which holds the creation of a SuperClass instance (objABC) because TestClass extends the SuperClass. Now, the objABC object is available in TestClass. The objABC is an instance of ABC class, this ABC class has a member variable str. Thus, the str object can be accessed directly by objABC.str or this.objABC.str. Here, this (this.objABC.str) refer to current object (object1).

Question17: what will be the output of given code?

```
public class TestClass {
 public static void main (String[] args) {
     Sub objSub = new Sub(" Inc");
 }
}
class Base {
 Base() { //2
 this("Kogent ", "Solution");
```

```
 }
 Base(String s, String t) { //3
 this(s + t);
 }
 Base(String s) { //4
 System.out.print(s);
 }
}
class Sub extends Base {
 Sub(String s) { //1
 System.out.print(s);
 }
 Sub(String s, String t) {
 this (t + s + "1");
 }
 Sub () {
 super ("2");
 };
}
```

What will happen when you compile and run the preceding program?

A. It displays “Inc” as an output. B. It displays “Inc Inc” as an output.
C. It displays “Kogent Solution Inc” as an output. D. It displays “Inc 2 Kogent Solution” as an output

Option C is correct.

Explanation: Within the main() method, the object objSub of the Sub class is created by passing Inc as an argument. This object calls Sub class constructor at //1. The default constructor of Base class is called at //2
(by default any subclass calls its superclass’s default constructor.) Within the body of default constructor at //2, it calls constructor at //3 and this constructor (Base(String s, String t)) calls constructor at //4 and add the string value Kogent Solution. Finally, the constructor at //1 is executed and displays Kogent Solution Inc as an output