QUESTION
What is the result?

```
11. public class Person {
12.         String name = "No name";
13.         public Person(String nm) { name = nm; }
14. }
15.
16. public class Employee extends Person {
17.         String empID = "0000";
18.         public Employee(String id) { empID = id; }
19. }
20.
21. public class EmployeeTest {
22.         public static void main(String[] args){
23.                     Employee e = new Employee("4321");
24.                     System.out.println(e.empID);
25.         }
26. }
```

Choose the answer
A. 4321
B. 0000
C. An exception is thrown at runtime.
D. Compilation fails because of an error in line 18.

Answer: D

QUESTION
Given:

```
1. class Atom {
2.    Atom() { System.out.print("atom "); }
3. }
4. class Rock extends Atom {
5.    Rock(String type) { System.out.print(type); }
6. }
7. public class Mountain extends Rock {
8.          Mountain() {
9.                super("granite ");
10.               new Rock("granite ");
11.         }
12. public static void main(String[] a) { new Mountain(); }
13.}
```

What is the result?
A. Compilation fails.
B. atom granite
C. granite granite
D. atom granite granite
E. An exception is thrown at runtime.
F. atom granite atom granite

Answer: F

Given:

```
21. class Money {
22.         private String country = "Canada";
23.         public String getC() { return country; }
24. }
25. class Yen extends Money {
```

```
26.         public String getC() { return super.country; }
27. }
28. public class Euro extends Money {
29.         public String getC(int x) { return super.getC(); }
30.         public static void main(String[] args) {
31.               System.out.print(new Yen().getC() + " " + new Euro().getC());
32.         }
33. }
```
What is the result?
A. Canada
B. null Canada
C. Canada null
D. Canada Canada
E. Compilation fails due to an error on line 26.
F. Compilation fails due to an error on line 29.

Answer: E

Given:
```
10. public class SuperCalc {
11.         protected static int multiply(int a, int b) { return a * b;}
12. }
```
and:
```
20. public class SubCalc extends SuperCalc{
21.         public static int multiply(int a, int b) {
22.               int c = super.multiply(a, b);
23.               return c;
24.         }
25. }
```
and:
```
30. SubCalc sc = new SubCalc ();
31. System.out.println(sc.multiply(3,4));
32. System.out.println(SubCalc.multiply(2,2));
```

What is the result?
A. 12
B. The code runs with no output.
C. An exception is thrown at runtime.
D. Compilation fails because of an error in line 21.
E. Compilation fails because of an error in line 22.
F. Compilation fails because of an error in line 31.

Answer: E

QUESTION
Click the Exhibit button.
```
1. public class A {
2.    public String doit(int x, int y){
3.          return "a";
4.    }
5.
6.    public String doit(int... vals){
7.          return "b";
8.    }
9. }
```

Given:
```
 25. A a = new A();
 26. System.out.println(a.doit(4, 5));
```

What is the result?
A. Line 26 prints "a" to System.out.
B. Line 26 prints "b" to System.out.
C. An exception is thrown at line 26 at runtime.
D. Compilation of class A will fail due to an error in line 6.

Answer: A

Given:

```
public class Yikes {
 public static void go(Long n) {
      System.out.print("Long ");
 }
 public static void go(Short n) {
      System.out.print("Short ");
 }
 public static void go(int n) {
      System.out.print("int ");
 }
 public static void main(String[] args) {
      short y = 6;
      long z = 7;
      go(y);
      go(z);
 }
}
```

What is the result?
A. int Long
B. Short Long
C. Compilation fails.
D. An exception is thrown at runtime.

Instance control flow from parent to child
++
whenever we create an object of child class the following things takes place

a. Identification of instance members from parent to child

b. Execution of instance variable assignment and instance block only in parent classs.
c. Execution of Parent constructor

d. Execution of instance variable assignment and instance block only in child classs.
e. Execution of Child constructor

eg#1.

```
class Parent
{
      int x= 10;
      {
            methodOne();
            System.out.println("Parent fist instance block...");
      }

      Parent()
      {
```

```
            System.out.println("Parent class constructor...");
      }
      public static void main(String... args)
      {
            Parent p = new Parent();
            System.out.println("Parent class main method...");
      }
      public void methodOne()
      {
            System.out.println(y);
      }
      int y =20;
}

class Child extends Parent
{
      int i= 100;

      {
            methodTwo();
            System.out.println("Child fist instance block...");
      }

      Child()
      {
            System.out.println("Child class constructor...");
      }
      public static void main(String... args)
      {
            Child  c = new Child();
            System.out.println("Child class main method...");
      }
      public void methodOne()
      {
            System.out.println(j);
      }
      int j =200;
}

public class Test
{
      public static void main(String[] args)
      {
            Child c = new Child();

      }
}
```

static control flow in inheritance
++++++++++++++++++++++++++++++++
Whenever we are executing child class the following sequence of events takes place

a. Identification of static memebers from parent to child.
b. Execution of static variable assignments and static block execution from parent to child
c. Execution of child class main().

eg#1.
class Base

```
{
	static int i = 10;
	static{
		methodOne();
		System.out.println("base static block");
	}
	public static void main(String... args)
	{
		methodOne();
		System.out.println("Base main");
	}
	static void methodOne()
	{
		System.out.println(j);
	}
	static int j = 20;
}

class Derived extends Base
{
	static int x = 100;

	static{
		methodTwo();
		System.out.println("Derived static block");
	}
	public static void main(String... args)
	{
		methodTwo();
		System.out.println("Derived main");
	}
	static void methodTwo()
	{
		System.out.println(y);
	}
	static int y = 200;
}

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


	}
}
```

```
Output
D:\OctBatchMicroservices>java Derived
0
base static block
0
Derived static block
200
Derived main

D:\OctBatchMicroservices>java Base
0
base static block
```

20
Base main

Creating an object everytime without its need is it a good programming practise?
Ans. No, it is not a good programming practise because to create an object
   a. loading of .class file should be happen[All the activities of static control flow should happen]
   b. instance control flow from parent to child should happen.
Since it is time consuming event, only when it is required we need to use "new" keyword in java.

Pillars of oops
 a. Encapsulation => Datahiding(private, setXXXX(),getXXXX()) + abstraction[abstract,interface]
 b. Inheritance   => ReUsablitiy(IS-A,HAS-A)
 c. Polymorphism => Code Flexiblity(Overloading,Overriding, MethodHiding)

What is TightCoupling and What is Loose Coupling?

TightCoupling
++++++++++++
eg#1.
```
class Forest
{
	//Method Overloading :: Tight Coupling
	public void allowAnimal(Tiger t)
	{
		t.eat();
		t.sleep();
		t.breathe();
	}
	public void allowAnimal(Deer d)
	{
		d.eat();
		d.sleep();
		d.breathe();
	}
	public void allowAnimal(Monkey m)
	{
		m.eat();
		m.sleep();
		m.breathe();
	}
}
```

Loose Coupling
++++++++++++++
eg#1.
```
//Helper class
class Forest
{
	/*
		RunTime Polymphism[1:M]

			= new Tiger();
		Animal ref = new Deer();
			   = new Monkey();
```

```
	*/
	//MethodOverriding :: LooseCoupling
	public void allowAnimal(Animal ref)
	{
		ref.eat();
		ref.sleep();
		ref.breathe();

		System.out.println();
	}
}
```

Abstract class in java
++++++++++++++++++++
=> If we don't want an object to be created for a particular class, then such class we need to mark as "abstract".
=> abstract access modifer is applicable at
	a. class level : prevents object creation.
	b. method level : prevents giving the implementation for body of a method.
	c. variable level : not applicable at variable level.
=> Through abstract keyword we can promote "abstraction".
=> By referring to abstract class, we would get to know only the services name(methodnames),but not the internal implementation given by
   developers, this mechanism only we call it as "abstraction".
=> for an abstract class,"instantiation is not possible",but we can create a reference for an "abstract class".
=> If a class contains only one abstract methods also,then we need to mark the class as "abstract".

eg#1.

```
//Parent class normally should be abstract class with abstract methods.
abstract class Plane
{
	//abstract methods : Method without implementation
	public abstract void takeOff();

	public abstract void fly();

	public abstract void land();

	public abstract void carry();

}

//Child class
final class Passenger extends Plane
{
	//Specific implementation
	@Override
	public void takeOff(){
		System.out.println("Passenger plane take off...");
	}

	@Override
	public void fly(){
		System.out.println("Passenger plane is flying...");
	}

	@Override
```

```
	public void land(){
		System.out.println("Passenger plane is landing...");
	}

	@Override
	public void carry(){
		System.out.println("Passenger plane is carrying passengers...");
	}
}

//Child class
final class Cargo extends Plane
{
	//Specific implementation
	@Override
	public void takeOff(){
		System.out.println("Cargo plane is carrying...");
	}

	@Override
	public void fly(){
		System.out.println("Cargo plane is flying...");
	}

	@Override
	public void land(){
		System.out.println("Cargo plane is landing...");
	}

	@Override
	public void carry(){
		System.out.println("Cargo plane is carrying goods...");
	}
}

//Child class
final class Fighter extends Plane
{
	//Specific implementation
	@Override
	public void takeOff(){
		System.out.println("Fighter plane is taking off...");
	}

	@Override
	public void fly(){
		System.out.println("Fighter plane is flying...");
	}

	@Override
	public void land(){
		System.out.println("Fighter plane is landing...");
	}

	@Override
	public void carry(){
		System.out.println("Fighter plane is carrying weapons..");
	}
}
```

```
//Helper class
final class Airport
{
      /*MethodOverriding :TruePolymorphsim[1:M] => Loose Coupling
                              = new Cargo();
                  Plane ref = new Passenger();
                              = new Fighter();
      */
      public void allowPlane(Plane ref)
      {
            System.out.println("Working with object called ::
"+ref.getClass().getName());
            ref.takeOff();
            ref.carry();
            ref.fly();
            ref.land();
            System.out.println();
      }
}
public class Test
{
      public static void main(String[] args)
      {
            //Creating 3 objects of Plane Type
            Cargo c = new Cargo();
            Passenger p =new Passenger();
            Fighter f = new Fighter();

            //Taking the actions for all the 3 planes
            Airport a = new Airport();
            a.allowPlane(c);
            a.allowPlane(p);
            a.allowPlane(f);
      }
}
```

```
Output
Working with object called :: Cargo
Cargo plane is carrying...
Cargo plane is carrying goods...
Cargo plane is flying...
Cargo plane is landing...

Working with object called :: Passenger
Passenger plane take off...
Passenger plane is carrying passengers...
Passenger plane is flying...
Passenger plane is landing...

Working with object called :: Fighter
Fighter plane is taking off...
Fighter plane is carrying weapons..
Fighter plane is flying...
Fighter plane is landing...
```