Conclusions of this() vs super()
++++++++++++++++++++++++++++++

Case1:
 We have to take super() or this() only in the 1st line of the constructor, if we are taking anywhere
 else it would result in "CompileTimeError".

eg#1.

```
class Demo
{
	Demo()
	{
		System.out.println("Inside constructor");
		this(10);
	}
	Demo(int i)
	{
		System.out.println("Inside One-Arg constructor");
	}
}
public class Test
{
	public static void main(String[] args)
	{
			Demo d = new Demo();
	}
}
```

eg#2.

```
class Demo
{
	Demo()
	{
		System.out.println("Inside constructor");
		super();
	}
}
public class Test
{
	public static void main(String[] args)
	{
			Demo d = new Demo();
	}
}
```

Case2:
We can't use either super() or this() both simultaneously

eg#1.

```
class Demo
{
	Demo()
	{
		super();
		this(10);
	}
```

```
	Demo(int i)
	{
		System.out.println("Inside One-Arg constructor");
	}
}
public class Test
{
	public static void main(String[] args)
	{
			Demo d = new Demo();
	}
}
```

Case3:
super() or this() should always be the first statement inside the constructor but we can't use inside the method, if we try to use it would
result in "CompileTime Error".

```
class Demo
{
	public void methodOne();
	{
		super();
	}
}
public class Test
{
	public static void main(String[] args)
	{
			Demo d = new Demo();
			d.methodOne();
	}
}
```

Which of the following statemetns are true?
 a. Whenever we are creating an object of child class, parent class constructor will be called ?
Answer: true

 b. Whenever we are creating an object of child class object then automatically parent class object will be created?
Answer: false.only child class object will be created but not the parent class.

eg#1.

```
class Parent
{
	Parent()
	{
		System.out.println(this.hashCode());//366712642
	}
}
class Child extends Parent
{
	Child()
	{
		System.out.println(this.hashCode());//366712642
	}
```

```
}
public class Test
{
	public static void main(String[] args)
	{
		Child c = new Child();
		System.out.println(c.hashCode());//366712642
	}
}
```

Recursive functions
++++++++++++++++++
A function is called using two methods types
  a. Nested call
  b. Recursive call

Nested call
1. Calling function inside another function.
2. In nested call, there is a calling function which calls another function(called function).

```
eg#1.
public static void methodOne()
{
	methodTwo();
}
public static void methodTwo()
{
	methodOne();
}
```

Recursive call
++++++++++++++
1. Calling a function within the same function is called "recursive call".
2. In recursive call called and calling function both are same.

```
eg#1.
public static void methodOne()
{
	methodOne();
}
```

Case1: recursive method call is always "RuntimeException" where as recursive constructor invocation is a "CompiletimeError".

```
eg#1
public class Test
{
	public static void methodOne()
	{
		methodTwo();
	}

	public static void methodTwo()
	{
		methodOne();
```

```
	}

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

Output:: java.lang.StackOverflowError


eg#2.
public class Test
{
	Test(int i)
	{
		this();
	}

	Test()
	{
		this(10);
	}

	public static void main(String[] args)
	{
			Test t = new Test();
			System.out.println("hello");
	}
}

CE: recursive constructor invocation


Questions on inheritance
++++++++++++++++++++++++
Q>
class Vehicle
{
	int x;
	Vehicle(){
		this(10);//line-n1
	}
	Vehicle(int x){
		this.x = x;
	}
}
class Car extends Vehicle
{
	int y;
	Car(){
		super();//line-n2
		this(20);
	}
	Car(int y){
		this.y= y;
	}
	public String toString(){
```

```
            return super.x + " " + super.y;
      }
}
public class Test {
      public static void main(String[] args) {
            Vehicle y= new Car();
            System.out.println(y);
      }
}
```

Predict the answer

A. 10:20
B. 0:20
C. Compilation fails at line n1
D. Compilation fails at line n2

Answer: D

Q>

Given:

```
1. class X {
2.    X() { System.out.print(1); }
3.    X(int x) {
4.          this(); System.out.print(2);
5.    }
6. }
7. public class Y extends X {
8.    Y() { super(6); System.out.print(3); }
9.    Y(int y) {
10.         this(); System.out.println(4);
11.   }
12.   public static void main(String[] a) { new Y(5); }
13.}
```

What is the result?

A. 13
B. 134
C. 1234
D. 2134
E. 2143
F. 4321

Answer: C

Q>

Given

```
public class Hello {
      String title;
      int value;
      public Hello() {
            title += " World";
      }
      public Hello(int value) {
            this.value = value;
            title = "Hello";
            Hello();
      }
}
```

and:

```
Hello c = new Hello(5);
```

```
System.out.println(c.title);
```

What is the result?
A. Hello
B. Hello World
C. Compilation fails.
D. Hello World 5
E. The code runs with no output.
F. An exception is thrown at runtime.

Answer: C

Types of inheritance supported by java
++++++++++++++++++++++++++++++++++++
1. Multiple inheritance : Not supported because of ambiguity in java through "classes".
2. Multilevel inheritance : Getting properties from parent to child in hierarchial way is refered as "MultiLevel Inheritance".
3. Cyclic inheritance : Not supported in java becoz of consructor invocation in loop.

Why java won't support Multiple Inheritance?
Ans. To avoid ambiguity b/w method calls coming for mulitple inheritance, java won't support mulitple inheritance.

eg#1.
```
class Parent1
{
	public void methodOne()
	{
		System.out.println("From Parent1");
	}
}

class Parent2
{
	public void methodOne()
	{
		System.out.println("From Parent2");
	}
}
class Child extends Parent1,Parent2
{

}

public class Test
{
	public static void main(String[] args)
	{
		 Child c = new Child();
		 c.methodOne();
	}
}
```
Output: CE: Parent1,Parent2

eg#2.

```
class Parent extends Child
{
	Parent()
	{
		super();
	}
}
class Child extends Parent
{
	Child()
	{
		super();
	}
}
public class Test
{
	public static void main(String[] args)
	{
		 Child c = new Child();
	}
}
```

Output:CE cyclic inheritance involing the parent

program to Demonstrate the usage of inheritance

++

```
class Plane
{
	String engine;
	float fuel;
	int wheel;

	public void takeOff()
	{
		System.out.println("Plane tookoff...");
	}
	public void fly()
	{
		System.out.println("Plane is flying...");
	}
	public void land()
	{
		System.out.println("Plane is landing...");
	}
}

class Passenger extends Plane
{
	public void carryPassengers()
	{
		System.out.println("Carrying Passengers...");
	}
}
class Cargo extends Plane
{
	public void carryCargo()
	{
		System.out.println("Carrying Cargo...");
	}
}
```

```
class Fighter extends Plane
{
	public void carryWeapons()
	{
		System.out.println("Carrying Weapons...");
	}
}
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
		c.takeOff();
		c.carryCargo();
		c.fly();
		c.land();

		System.out.println();
		p.takeOff();
		p.carryPassengers();
		p.fly();
		p.land();

		System.out.println();
		f.takeOff();
		f.carryWeapons();
		f.fly();
		f.land();



	}
}
```

```
Output
Plane tookoff...
Carrying Cargo...[Specialized  method]
Plane is flying...
Plane is landing...

Plane tookoff...
Carrying Passengers...[Specialzied method]
Plane is flying...
Plane is landing...

Plane tookoff...
Carrying Weapons...[Specialzied method]
Plane is flying...
Plane is landing...
```

Overident Methods
++++++++++++++++

```
class Animal
{
	public void eat()
	{
		System.out.println("Animal is eating...");
	}
	public void sleep()
	{
		System.out.println("Animal is sleeping...");
	}
	public void breathe()
	{
		System.out.println("Animal is breathing...");
	}
}
class Tiger extends Animal
{
	//informing compiler about overidden method
	@Override
	public void eat()
	{
		System.out.println("Tiger hunts and eat...");
	}
}
class Deer extends Animal
{
	//informing compiler about overidden method
	@Override
	public void eat()
	{
		System.out.println("Deer will graze and eat...");
	}
}

class Monkey extends Animal
{
	//informing compiler about overidden method
	@Override
	public void eat()
	{
		System.out.println("Monkey steal and eat...");
	}

}
public class Test
{
	public static void main(String[] args)
	{
		//Creating an Object of Animal Type
		Tiger t  = new Tiger();
		Deer d   = new Deer();
		Monkey m = new Monkey();

		//Invoking the behaviours of all 3 animals
		t.eat();
		t.sleep();
		t.breathe();

		System.out.println();
```

```
            d.eat();
            d.sleep();
            d.breathe();

            System.out.println();
            m.eat();
            m.sleep();
            m.breathe();

      }
}
```

```
Output
Tiger hunts and eat... [Overriden method got called]
Animal is sleeping...
Animal is breathing...

Deer will graze and eat...[Overriden method got called]
Animal is sleeping...
Animal is breathing...

Monkey steal and eat...[Overrident method got called]
Animal is sleeping...
Animal is breathing...

Note: In cas of Overriding
  1. Compiler will use reference of the type to check whether the respective method
is avaialbe in the class or not.
      In case of Overriding
  1. JVM will use the current object and respective method of that object will be
called.
```