In java abstraction can be achieved using
      a. abstract class
      b. interface

In real word, for few cases Object should not be created, to handle such scenarios in java we need go for a keyword called
 "abstract".

abstract access modifer can be applied at
      a. class level   -> yes possible, if we make class as abstract then object can't be instantiated.
      b. method level   -> yes possible, if we make method as abstract,then we can't give body for the methods.
      c. variable level -> no we can't use abstract at variable level.

Rules of abstract accessmodifier in java
++++++++++++++++++++++++++++++++++++++
1. If a class contains atleast one abstract method, then mark the class as "abstract".
2. abstract class can't be instantiated.
3. for an abstract class, we can create a reference,but not the object.
4. Inside abstract class, we can write concrete methods also.
5. If a parent class is abstract, then compulsorily the child class should give implementation for all the abstract
   methods otherwise the child class also would become "abstract".
6. Even if the class doesn't contain abstract methods/concrete method still we can mark the empty class as "abstract".

eg#1.
```
//Exposing the set of services but hiding the internal implementation
abstract class Plane
{
      public abstract void takeOff();
      public abstract void fly();
      public abstract void land();
}

class CargoPlane extends Plane
{
      public void takeOff(){
            System.out.println("CargoPlane tookOff..");
      }
      public void fly(){
            System.out.println("CargoPlane flying..");
      }
      public void land(){
            System.out.println("CargoPlane landing..");
      }
}
class PassengerPlane extends Plane
{
      public void takeOff(){
            System.out.println("PassengerPlane tookOff..");
      }
      public void fly(){
            System.out.println("PassengerPlane flying..");
      }
      public void land(){
```

```
            System.out.println("PassengerPlane landing..");
      }
}

class FighterPlane extends Plane
{
      public void takeOff(){
            System.out.println("FighterPlane tookOff..");
      }
      public void fly(){
            System.out.println("FighterPlane flying..");
      }
      public void land(){
            System.out.println("FighterPlane landing..");
      }
}

class Airport
{
      public void allowPlane(Plane ref){
            ref.takeOff();
            ref.fly();
            ref.land();
            System.out.println();
      }
}

public class Test {
      public static void main(String[] args) {
            Airport a =new Airport();
            a.allowPlane(new PassengerPlane());
            a.allowPlane(new FighterPlane());
            a.allowPlane(new CargoPlane());
      }
}

Output
D:\Decode Java1.0Batch>javac Test.java

D:\Decode Java1.0Batch>javac Test.java

D:\Decode Java1.0Batch>java Test
PassengerPlane tookOff..
PassengerPlane flying..
PassengerPlane landing..

FighterPlane tookOff..
FighterPlane flying..
FighterPlane landing..

CargoPlane tookOff..
CargoPlane flying..
CargoPlane landing..


Can we create an Object for abstract class?
Ans. No.

When will the constructor gets called?
```

Ans. During the creation of an Object.

Does abstract class has constructor?
Ans. yes

Why we need constructor in abstract class, when we can't instantiate an object?
Ans. To get the properites of parent class to child class, we need constructors in abstract class also.

eg#1.

```
//Exposing the set of services but hiding the internal implementation
abstract class Person
{
	String name;
	int age;
	float height;

	Person(String name,int age,float height){
		this.name =name;
		this.age = age;
		this.height =height;
	}
}
class Student extends Person
{
	int marks;
	float avg;

	Student(String name,int age,float height,int marks,float avg){
		//To call parameterised consturctor of parent from child class
		super(name,age,height);

		this.marks = marks;
		this.avg   = avg;
	}

	public void display(){
		System.out.println("Name   is :: "+name);
		System.out.println("Age    is :: "+age);
		System.out.println("Height is :: "+height);
		System.out.println("Marks  is :: "+marks);
		System.out.println("Avg    is :: "+avg);
	}
}
public class Test {
	public static void main(String[] args) {
		Student student = new Student("sachin",51,5.3f,100,53.5f);
		student.display();
	}
}
```

Output

```
D:\Decode Java1.0Batch>javac Test.java
D:\Decode Java1.0Batch>java Test
Name   is :: sachin
Age    is :: 51
Height is :: 5.3
Marks  is :: 100
Avg    is :: 53.5
```

eg#2.

```
//Exposing the set of services but hiding the internal implementation
abstract class Bird
{
	public abstract void fly();
	public abstract void eat();
}
class Sparrow extends Bird
{
	public void fly(){
		System.out.println("Sparrow fly @short height");
	}
	public void eat(){
		System.out.println("Sparrow eat grains...");
	}
}

//abstract class can contain concrete and abstract methods
//concrete methods : A method with implementation
//abstrct methods  : A method without implementation
abstract class Eagle extends Bird
{
	public void fly(){
		System.out.println("Eagel fly @very very height");
	}
	public abstract void eat();
}
class SerpentEagle extends Eagle
{
	public void eat(){
		System.out.println("Serpent eagel eats snakes...");
	}
}
class GoldenEagle extends Eagle
{
	public void eat(){
		System.out.println("Golden eagel catches prey over the ocean...");
	}
}

class Crow extends Bird
{
	public void fly(){
		System.out.println("Crow fly @medium height...");
	}
	public void eat(){
		System.out.println("Crow eat grains...");
	}
}

class Sky
{
	public void allowBird(Bird ref){
		ref.fly();
		ref.eat();
		System.out.println();
	}
}
```

```
public class Test {
	public static void main(String[] args) {
		Sky sky = new Sky();
		sky.allowBird(new Sparrow());
		sky.allowBird(new SerpentEagle());
		sky.allowBird(new GoldenEagle());
		sky.allowBird(new Crow());
	}
}
```

Output

```
D:\Decode Java1.0Batch>javac Test.java

D:\Decode Java1.0Batch>java Test
Sparrow fly @short height
Sparrow eat grains...

Eagly fly @very very height
Serpent eagel eats snakes...

Eagly fly @very very height
Golden eagel catches prey over the ocean...

Crow fly @medium height...
Crow eat grains...
```

Note:

1. abstract class contains concrete methods and abstract methods, so we say through abstract class 100% abstraction can't be achieved.
2. To achieve 100% abstraction, we need to go for "interfaces".