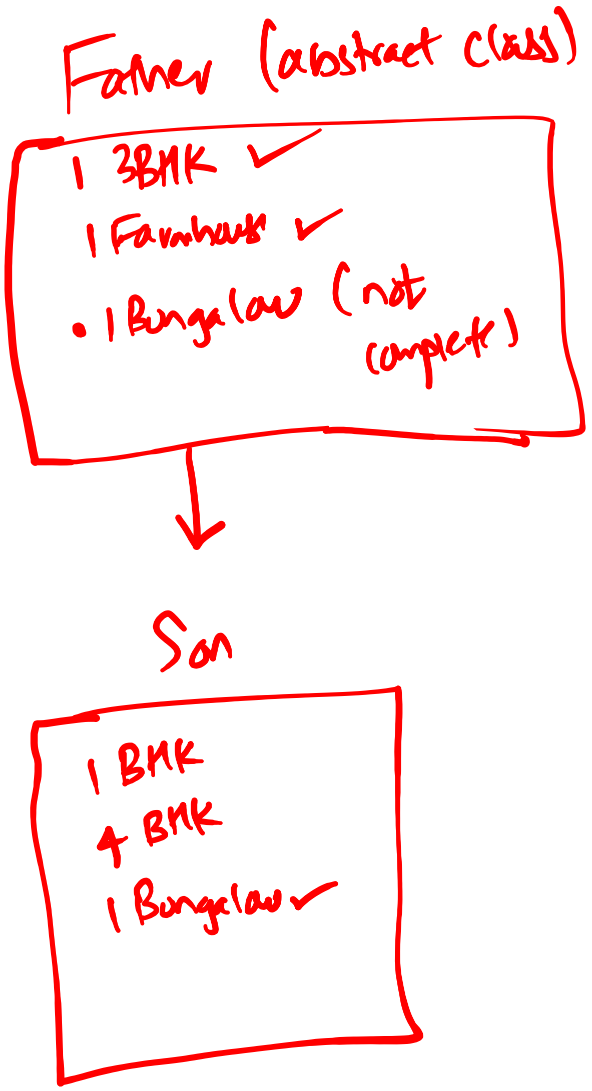
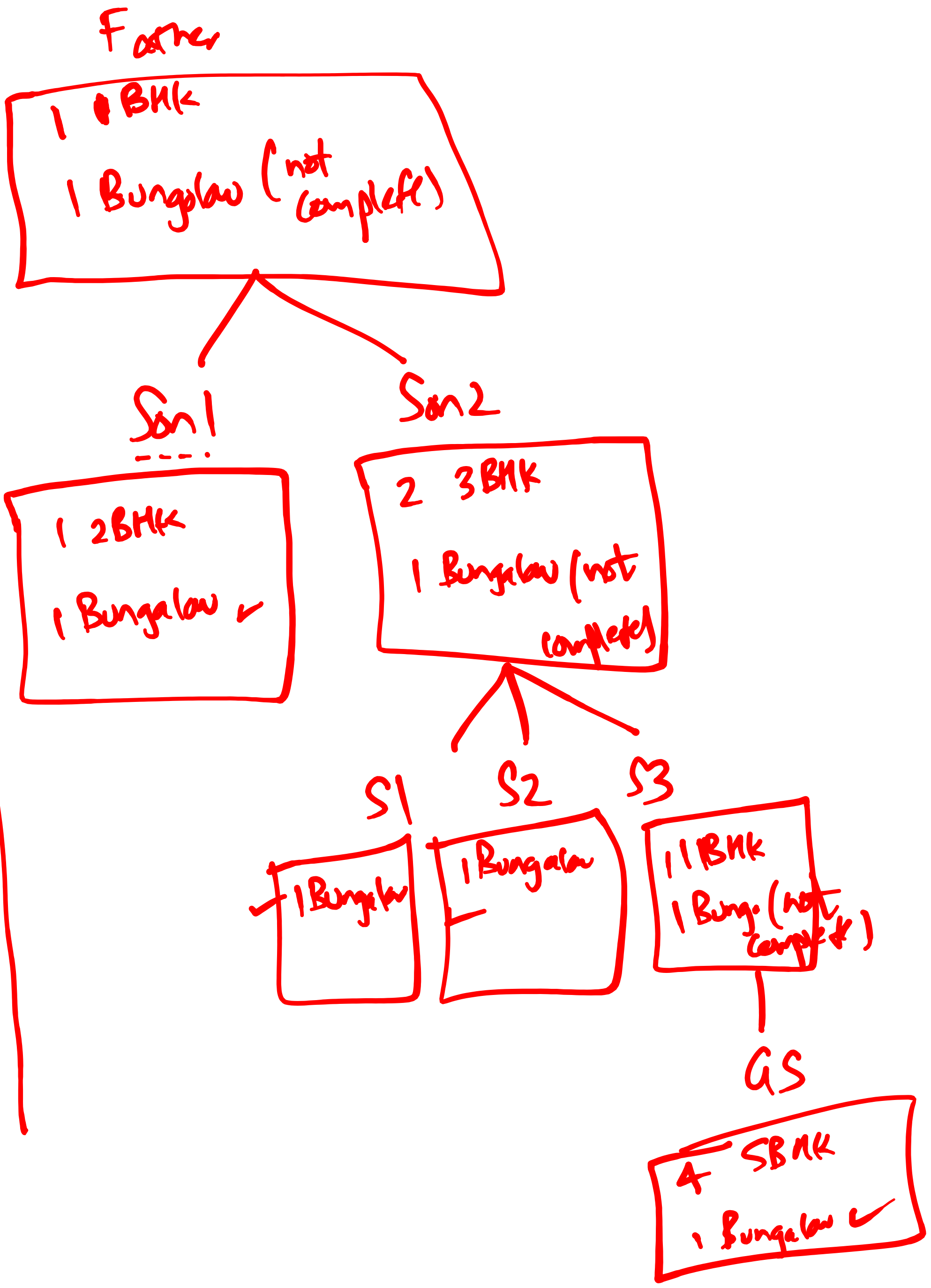
Inheritance


Father (abstract class)
1 3BHK ✓
1 Farmhouse ✓
• 1 Bungalow (not complete)
Son
1 BHK
4 BHK
1 Bungalow ✓


Father
1 BHK
1 Bungalow (not complete)
Son1
1 2BHK
1 Bungalow ✓
Son2
2 3BHK
1 Bungalow (not completed)
S1
1 Bungalow ✓
S2
1 Bungalow ✓
S3
1 1BHK
1 Bung. (not complete)
GS
4 5BHK
1 Bungalow ✓

Using Abstract Classes

Sometimes you will want to create a superclass that only defines a generalized form that will be shared by all of its subclasses, leaving it to each subclass to fill in the details. Such a class determines the nature of the methods that the subclasses must implement. One way this situation can occur is when a superclass is unable to create a meaningful implementation for a method.

This is the case with the class **Figure used in the preceding example. The definition of area( ) is simply a placeholder. It will not compute and display the area of any type of object.**

As you will see as you create your own class libraries, it is not uncommon for a method to have no meaningful definition in the context of its superclass. You can handle this situation two ways. One way, as shown in the previous example, is to simply have it report a warning message.

While this approach can be useful in certain situations—such as debugging—it is not usually appropriate. You may have methods which must be overridden by the subclass in order for the subclass to have any meaning.

Consider the class **Triangle. It has no meaning if area( ) is not defined. In this case, you want some way to ensure that a subclass does, indeed, override all necessary methods. Java's solution to this problem is the *abstract method.***

You can require that certain methods be overridden by subclasses by specifying the **abstract type modifier. These methods are sometimes referred to as *subclasser responsibility because they have no implementation specified in the superclass.***

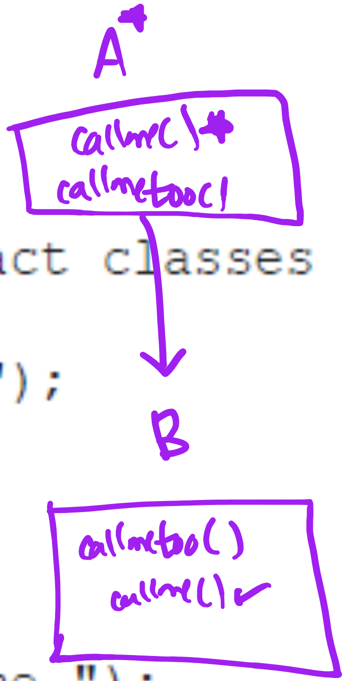
Thus, a subclass must override them—it cannot simply use the version defined in the superclass. To declare an abstract method, use this general form:

abstract *type name(parameter-list);*

As you can see, no method body is present. Any class that contains one or more abstract methods must also be declared abstract. To declare a class abstract, you simply use the **abstract keyword in front of the class keyword at the beginning of the class declaration. There can be no objects of an abstract class.**

That is, an abstract class cannot be directly instantiated with the new operator. Such objects would be useless, because an abstract class is not fully defined. Also, you cannot declare abstract constructors, or abstract static methods.
Any subclass of an abstract class must either implement all of the abstract methods in the superclass, or be itself declared abstract.

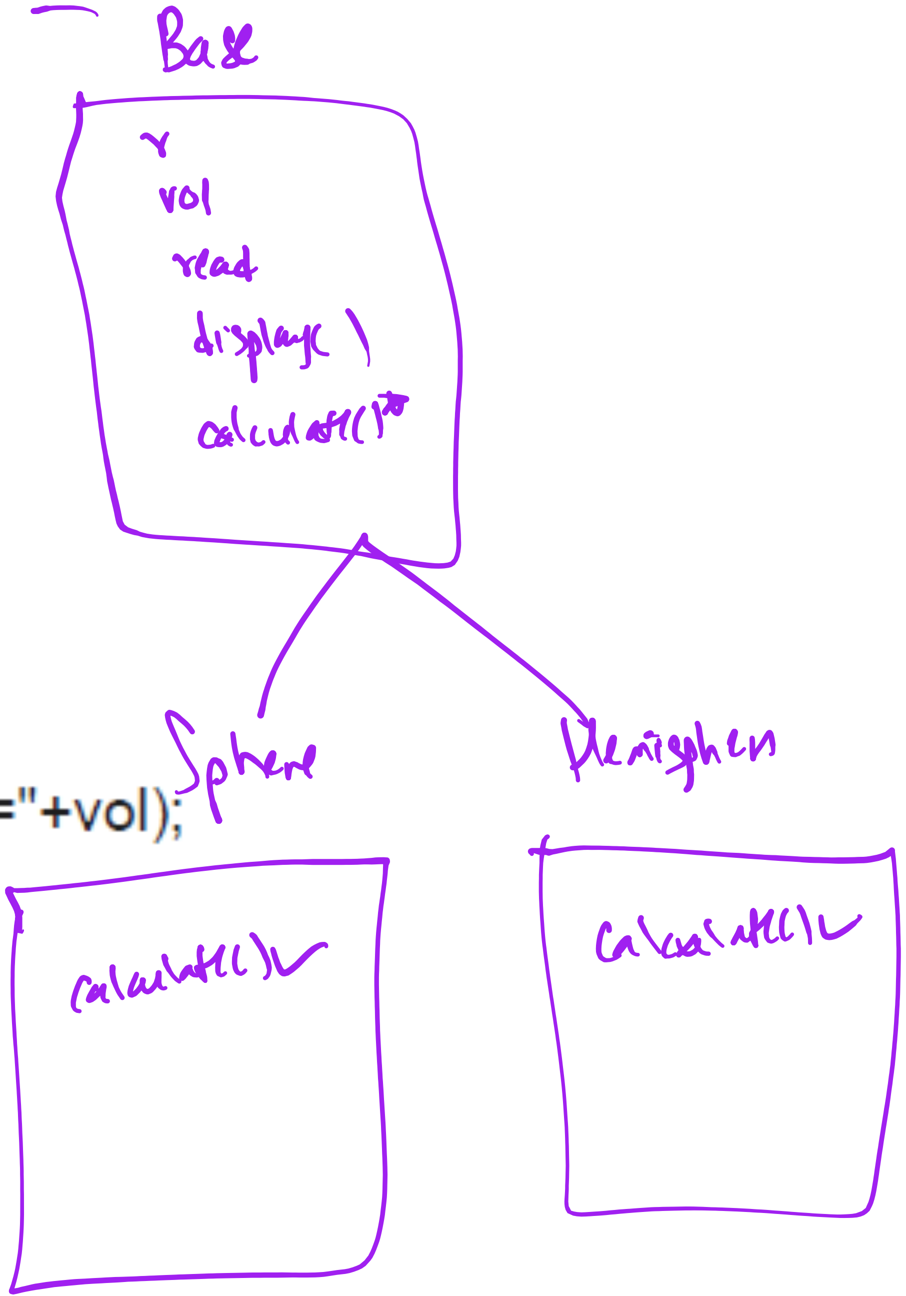
```
// A Simple demonstration of abstract.
abstract class A {
abstract void callme(); //incomplete
// concrete methods are still allowed in abstract classes
void callmetoo() {
System.out.println("This is a concrete method.");
}
}
class B extends A {
void callme() {
System.out.println("B's implementation of callme.");
}
}
class AbstractDemo {
public static void main(String args[]) {
B b = new B();
b.callme();
b.callmetoo();
}
}
```



o/p: B's impl. of callme
This is a concrete mth.

//WAP to display volume of the sphere and hemisphere. Make use of Abstract class.

```
import java.io.*;
abstract class Base{
protected float r,vol;
public void read(float x)
{
  r=x;
}
abstract void calculate();
public void display()
{
  System.out.println("Volume="+vol);
}
}
```



```
class Sphere extends Base
{
public void calculate()
{
   vol=3.14f*r*r*r*4/3;
}
}
class Hemisphere extends Base
{
public void calculate()
{
  vol=3.14f*r*r*r*2/3;
}
}
```

```
class Main{
public static void main (String args[]) throws IOException{
float x;
String str;
BufferedReader br = new BufferedReader (new InputStreamReader (System.in));
System.out.println("Enter the radius:");
str=br.readLine();
x=Float.parseFloat(str);
Sphere s=new Sphere();
s.read(x);
s.calculate();
System.out.println("Sphere:");
s.display();
Hemisphere h=new Hemisphere();
h.read(x);
h.calculate();
System.out.println("Hemisphere:");
h.display();
}
}
```

Output:

```
Enter the radius:
10
Sphere:
Volume=4186.6665
Hemisphere:
Volume=2093.3333
```

The method calculate() is declared to be ‘abstract‘ in the base class. This makes it compulsory for the derived class to override this method. Also it makes the base class to be compulsorily be declared as ‘abstract‘.

The implementation of the above program is done with method calculate() in base abstract class and in subclasses Sphere and Hemisphere. The derived classes have the definition of the method calculate(), which is an abstract method of the base class.

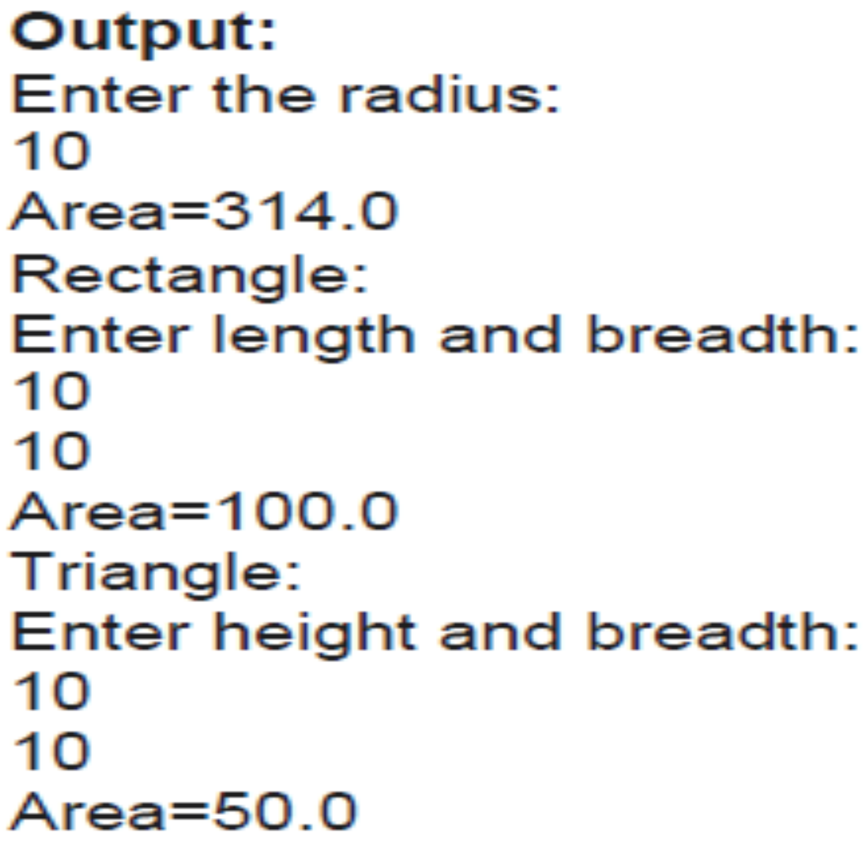
//Write a abstract class program to calculate area of circle, rectangle and triangle.

```
import java.io.*;
abstract class Base{
protected float r,l,b,area;
abstract void calculate();
public void display()
{
  System.out.println("Area="+area);
}
}
class Circle extends Base
{
public void read(float x)
{
  r=x;
}
public void calculate()
{
   area=3.14f*r*r;
}
}
```

```
class Rectangle extends Base
{
public void read(float x, float y)
{
  l=x;
  b=y;
}
public void calculate()
{
  area=l*b;
}
}
```

```
class Triangle extends Base
{
public void read(float x, float y)
{
  l=x;
  b=y;
}
public void calculate()
{
  area=0.5f*l*b;
}
}
```

```
class Main{
public static void main (String args[]) throws IOException{
float x,y;
String str;
BufferedReader br = new BufferedReader (new InputStreamReader (System.in));
System.out.println("Circle:");
System.out.println("Enter the radius:");
str=br.readLine();
x=Float.parseFloat(str);
Circle s=new Circle();
s.read(x);
s.calculate();
s.display();
System.out.println("Rectangle:");
System.out.println("Enter length and breadth:");
str=br.readLine();
x=Float.parseFloat(str);
str=br.readLine();
y=Float.parseFloat(str);
Rectangle h=new Rectangle();
h.read(x,y);
h.calculate();
h.display();
System.out.println("Triangle:");
System.out.println("Enter height and breadth:");
str=br.readLine();
x=Float.parseFloat(str);
str=br.readLine();
y=Float.parseFloat(str);
Triangle t=new Triangle();
t.read(x,y);
t.calculate();
t.display();
}
}
```



```
Output:
Enter the radius:
10
Area=314.0
Rectangle:
Enter length and breadth:
10
10
Area=100.0
Triangle:
Enter height and breadth:
10
10
Area=50.0
```

The implementation of the abstract class is done with same methods read() and calculate() in the base abstract class and in subclasses Circle, rectangle and triangle.
The base class has a method read() and calculate() declared as —abstract|| making the base class to be —abstract|| and also making it compulsory for the derived classes to override these methods.

//Write a abstract class program to calculate area of square and triangle.

```
import java.io.*;
abstract class Base{
protected float l,b,area;
abstract void calculate();
public void display()
{
 System.out.println("Area="+area);
}
}
class Square extends Base
{
public void read(float x)
{
 l=x;
}
public void calculate()
{
  area=l*l;
}
}
```

```
class Triangle extends Base
{
public void read(float x, float y)
{
  l=x;
  b=y;
}
public void calculate()
{
  area=0.5f*l*b;
}
}
```

```
class Main{
public static void main (String args[]) throws IOException{
float x,y;
String str;
BufferedReader br = new BufferedReader (new InputStreamReader (System.in));
 System.out.println("Square:");
 System.out.println("Enter the length of a side:");
 str=br.readLine();
 x=Float.parseFloat(str);
 Square s=new Square();
 s.read(x);
 s.calculate();
 s.display();
System.out.println("Triangle:");
System.out.println("Enter height and breadth:");
str=br.readLine();
x=Float.parseFloat(str);
str=br.readLine();
y=Float.parseFloat(str);
Triangle t=new Triangle();
t.read(x,y);
t.calculate();
t.display();
}
}
```

Output:

```
Square:
Enter the length of a side:
10
Area=100.0
Triangle:
Enter height and breadth:
10
10
Area=50.0
```

The implementation of the program is done with the same methods read() and calculate() in the base abstract class and in subclasses Square and Triangle.

The base class has the method read() and calculate() declared as —abstract‖ maki8ng the base class to be —abstract‖ and also making it compulsory for the derived classes to override these methods.

Notice that no objects of class A are declared in the program. As mentioned, it is not possible to instantiate an abstract class. One other point: class A implements a concrete method called callmetoo( ). This is perfectly acceptable. Abstract classes can include as much implementation as they see fit.

Although abstract classes cannot be used to instantiate objects, they can be used to create object references, because Java‘s approach to run-time polymorphism is implemented through the use of superclass references. Thus, it must be possible to create a reference to an abstract class so that it can be used to point to a subclass object.

You will see this feature put to use in the next example. Using an abstract class, you can improve the **Figure class shown earlier. Since there is no meaningful concept of area for an undefined two-dimensional figure, the following version of the program declares area( ) as abstract inside Figure. This, of course, means that all classes derived from Figure must override area( ).**

```
// Using abstract methods and classes.
abstract class Figure {
double dim1;
double dim2;
Figure(double a, double b) {
dim1 = a;
dim2 = b;
}
// area is now an abstract method
abstract double area();
}
class Rectangle extends Figure {
Rectangle(double a, double b) {
super(a, b);
}
// override area for rectangle
double area() {
System.out.println("Inside Area for Rectangle.");
return dim1 * dim2;
}
}
```

```
class Triangle extends Figure {
Triangle(double a, double b) {
super(a, b);
}
// override area for right triangle
double area() {
System.out.println("Inside Area for Triangle.");
return dim1 * dim2 / 2;
}
}
```

```
Triangle t = new Triangle(10, 8);
Figure figref; // this is OK, no object is created
figref = r;
System.out.println("Area is " + figref.area());
figref = t;
System.out.println("Area is " + figref.area());
}
}
```

As the comment inside **main( ) indicates, it is no longer possible to declare objects of type Figure, since it is now abstract. And, all subclasses of Figure must override area( ). To prove this to yourself, try creating a subclass that does not override area( ). You will receive a compile-time error.**

Although it is not possible to create an object of type Figure, you can create a reference variable of type Figure. The variable figref is declared as a reference to Figure, which means that it can be used to refer to an object of any class derived from Figure. As explained, it is through superclass reference variables that overridden methods are resolved at run time.