Practice Problem: Inheritance

Check the display of the following program.

```
public class Foo {
   public void method1() {
      System.out.println("foo 1");
   }
   public void method2() {
      System.out.println("foo 2");
   }
   public String toString() {
      return "foo";
   }
}
public class Bar extends Foo {
   public void method2() {
      System.out.println("bar 2");
   }
}

public class Baz extends Foo {
   public void method1() {
      System.out.println("baz 1");
   }
   public String toString() {
      return "baz";
   }
}
public class Mumble extends Baz {
   public void method2() {
      System.out.println("mumble 2");
   }
}

public class Polymorphism{
public static void main(String [] args){
Foo[] pity = { new Baz(), new Bar(),
            new Mumble(), new Foo() };
for (int i = 0; i < pity.length; i++) {
   System.out.println(pity[i]);
   pity[i].method1();
   pity[i].method2();
   System.out.println();
}
}
```

```
public class A {
public int x = 1;
public void setX(int a){
x=a;
}
}
public class B extends A {
public int getB(){
setX(2);
return x;}
}
public class C {
public static void main(String [] args){
A a = new A();
B b = new B();
System.out.println(a.x);
System.out.println(b.getB());
}
}
```

```
public class A {
private int x = 1;
protected void setX(int a){
x=a;
}
protected int getX(){
return x;}
}
public class B extends A {
public int getB(){
setX(2);
//return x; It does not work because private modifier, so
return getX();
}
}
public class C {
public static void main(String [] args){
A a = new A();
B b = new B();
System.out.println(a.getX());//a.x is not allowed, private!
System.out.println(b.getB());
}
}
```

```
public class A {
protected int x = 1;
protected void setX(int a){x=a;}
protected int getX(){return x;}
}
public class B extends A {
public int getB(){
setX(2);
return x;
}
}
public class C {
public static void main(String [] args){
A a = new A();
B b = new B();
System.out.println(a.getX());
System.out.println(b.x); //b.x is protected, then inherited.
System.out.println(b.getB());
}
}
```

```
public class A {
protected int x = 1;
protected void setX(int a){
x=a;
}
protected int getX(){
return x;}
}
public class B extends A {
protected int x = 3;
public int getX(){
return x; }
public int getB(){
return x;
}
}
public class C {
public static void main(String [] args){
A a = new A();
B b = new B();
System.out.println(a.getX());//target for comparison
System.out.println(b.getB());//subclass method access own attrib
System.out.println(b.getX());//overriding method, accessing sub
System.out.println(a.x); //protected
System.out.println(b.x); //overriding attribute!
}
}
```

```
//reusing the above class A and B
public class C {
public static void main(String [] args){
A a = new A();
A b = new B();  //polymorphism, making shadowing possible!
System.out.println(a.getX());
System.out.println(b.getX());//override, access subclass attri.
//System.out.println(b.getB()); not able to load subclass method!
System.out.println(a.x);
System.out.println(b.x); //variable shadowing!
}
}

// For your development:
//1) Is it good to block the use of b.getB()?
//    ANS>:  Good, because methods can be in template. In the security control, no leakage!
//2) Is it good to have the direct access of attribute such as b.x?
//    ANS>:  Better not, if it is not in your control. See how complicate it is in this program.
```

```
// reusing the above class A and C
public class B extends A {
protected int x = 3;
public int getX(){
setX(2); // call superclass method to set 2 in superclass attrib
return x; } //but return attrib of subclass
public int getB(){
return x;
}
}
```