Yesterday as we went over some competition problems, we discussed this. Since I was not properly prepared yesterday, here is a better explanation of what's going on here.

QUESTION 22

Which of the following could replace **<1*>** in the code to the right so that the A constructor works as intended?

A. `self.`
B. `this.`
C. `A.`
D. Nothing is required.
E. More than one of the above.

QUESTION 23

Which of the following terms is demonstrated by the `add` methods in the A and B classes to the right?

A. Overriding
B. Overloading
C. Encapsulation
D. A and B.
E. All of the above.

QUESTION 24

Assuming **<1*>** has been filled correctly, what is output by the lines marked `//q24` in the client code to the right?

A. `a: 4 b: 5 c: 6`
B. `a: 4 b: 9 c: 11`
C. `a: 4 b: 9 c: 6`
D. There is no output due to a compile error.
E. There is no output due to a runtime error.

QUESTION 25

Assuming **<1*>** has been filled correctly, what is output by the lines marked `//q25` in the client code to the right?

A. `a: 4 b: 9 c: 16`
B. `a: 4 b: 13 c: 16`
C. `a: 4 b: 14 c: 17`
D. There is no output due to a compile error.
E. There is no output due to a runtime error.

QUESTION 26

Assuming **<1*>** has been filled correctly, what is output by the lines marked `//q26` in the client code to the right?

A. `a: 4 b: 36 c: 80`
B. `a: 4 b: 52 c: 80`
C. `a: 4 b: 56 c: 85`
D. There is no output due to a compile error.
E. There is no output due to a runtime error.

```
class A{
 int n;
 String s;
 public A(String s, int n) {
  <1*>n = n;
  <1*>s = s;
 }
 public int add() {
  return n++;
 }
 public String toString() {
  return s+": "+n;
 }
}
class B extends A{
 public B(String s, int n) {
  super(s, n * 2);
 }
 public int add(int i) {
  n = add() + i;
  return n;
 }
 public int mult(int i) {
  n *= i;
  return i;
 }
}
//////////////client code///////////
A a = new A("a", 3);
B b = new B("b", 4);
A c = new B("c", 5);
a.add();
b.add();
c.add();
String s = a + " ";
s += b + " " + c;
out.println(s); //q24
b.add(4);
c.add(5);
s = a + " ";
s += b + " " + c;
out.println(s); //q25
b.mult(4);
c.mult(5);
s = a + " ";
s += b + " " + c;
out.println(s); //q26
```

The answers to 22 and 23 are b and b, of course. 24 is b as well.

The next two problems answer a compile error. Why? For 25 Since c is declared as a type A (is a reference of type A or x has static/ compile-time type A) BUT the actual object (runtime type / dynamic type) is of type B. Thus when we call add(5), it looks up at A for an add(int n) method which does not exist.

For problem 26 we have the same issue. mult(int n) is not in A. Note that one way to is this is to cast c as a B.

((B) c).mult(5); would compile.

This

```
c = (B) c;
c.mult(5);
```

does not work BTW.