Spring 2019 Test 2 Key w/ Explanations

1. **B::f3**

f5 is a public instruction so it is acceptable to call it. It then goes on to call f3() which is private but as we are inside the Base class when it is called this is acceptable.

2. **B::f1(i)**

Contrary to how C++ works, in Java overloading a method does not hide the base class's implementation. For this reason when it goes to call f1 it has the option to call f1(int) f1(double) or f1(long). f1(int) fits and can be widened to the other two so they are discarded, and Base's f1(int) is called.

3. **D::f1(1)**

Here once again it examines all the f1's available to it. A long can not fit in an int, and f1(long) could be widened to fit in f1(double) so f1(double) is removed, leaving D's f1(long) as the function that is called.

4. **B::f1(f)**

Here out of the three f1's neither a int or a long can hold a float, therefore B's f1(double) is what is called.

5. **D::f2**

Here D has implemented its own f2(). All Java methods are by default virtual, so Base's f2() is overridden, and D's f2() is called.

6. **D::f4**

Here f4() is a function that is only in D and as we are referencing a D object with a D reference we can call f4().

7. **B::f3**

Same as #1.

8. **B::f1(i)**

Here we have a B object referencing a D object. For this reason it will only be able to see the functions that B has available to it but if any functions are overridden by D they will be called. Here we can only see f1(int) and f1(double) which both fit the int(1) we have passed it. As f1(int) can be widened to f1(double), f1(double) can be removed and we can see that we call B's f1(int).

9. **B::f1(f)**

Here we can also only see f1(int) and f1(double) due to being a B reference. A long can not fit into an int but it can fit into a double so for this reason B's f1(double) is called.

10. **B::f1(f)**

Here we can only see f1(int) and f1(double). 1.0 can not fit into an int but it can fit into a double so f1(double) is called.

11. **D::f2**

Here we are able to call f2() but as this B reference is actually pointing to a D object it will call the overridden function which is D's f2().

12. **Err**

As we are a Base pointer we do not know of any function f4(), for this reason we do not know what to call and this will produce an error.

13. B::f3

In Java private methods are non-virtual, therefore B’s f3() function is not overwritten by D and we still call Base’s f3.

14. No

Not on current exam, will be updated later.

15. Yes

Not on current exam, will be updated later.

16. No

Not on current exam, will be updated later.

17. No

Not on current exam, will be updated later.

18. Err

This is an error, because if you decide to include super() it must be the first thing in the constructor.

19. 10

In our B constructor we set valB to 10.

20. 0

As line 18 is illegal and an error we will ignore it, and without that line and no super() call in our B constructor, what happens is that by default super() with no parameters will be called at the start of B’s constructor, therefore initializing val to 0.

21. -100

Not on current exam, will be updated later.

22. 10

Not on current exam, will be updated later.

23. 10

Not on current exam, will be updated later.

24. Ok

A B reference may be made on a D object as D derives from B.

25. Ok

As the D object has a B inside of it which contains a v we may set it here.

26. Err

Even though the D object has a dv, this is not something that would be known by a B reference which only has knowledge of things that relate to B so this is an error.

27. 5

When we say bvar = dvar we are setting bvar equal to the B part of dvar. Therefore it gets the value of v=5 from dvar.

28. 5

Set in Q25.

29. 5

When two references are set equal Ref1 = Ref2, it by default copies the information from inside Ref2 into Ref1.

30. 5

No changes made.

31. **B::f1**

The B reference calls B’s f1() function which is virtual but has not been overridden.

32. **D::f2**

The B reference calls B’s f2() function but this code has been overwritten by D’s f2() function as f2() is virtual in B (and therefore also D). For this reason D’s f2() is called.

33. **Err**

br2 is a B reference so it only knows information as it pertains to B, therefore it has no idea what a f3() is and can’t call it.

34. **4**

Ok, a > operand has been declared for the C class and functions as expected, printing the greater of the two values. Incase any of you do not recognize the notation it is called a ternary operator.
(Statement) ? (Happens if True) : (Happens if False)

35. **Err**

Here N has not defined a > operator so it has no idea how to compare the two.

36. **B() D() B(int) ~B()**

Here if the B constructor is not implicitly called in the initializer list, B’s default constructor will be called anyway before D is allowed to execute the instructions it has in its constructor. Therefore we get B(), then D(), then the B(1) being called is just a tricky part which is not a constructor for D’s base class but just a random B() object that is constructed and destroyed, hence the B(int) then ~B().

37. **D::f2(float)**

Here with how C++ works it will hide B’s f2(int) because D has declared its own f2(float) with a different parameter. Therefore due to how the scope works it will never see B’s f2() when called like this and will end up in D’s f2().

38. **~D() ~B()**

Here we delete D, where as constructors start with the base and go down, destructors start at the lowest level and work their way back up, therefore we print ~D() and then after that is done ~B().

39. **B::f3()**

Even though we have a f3() function in both B and D, it is not virtual, therefore the B pointer will call B’s f3().

40. **Err**

Even though D has a f4(), this is a B pointer which only knows things that relate to B, therefore it has no idea what a f4() is and will error out.