Fall 2018 Test 3 Key w/ Explanations

1. **A::f4 B::f5**

Here we have a A pointer pointing to a B object. f4() is not virtual so the A pointer will call the A class's f4(). When we get to calling f5(), f5() is virtual and B has its own implementation. Here due to that virtual behavior, even though it is private, we are able to call B's f5() as it has overwritten A's f5().

2. **A::f1**

Any static elements are always tied to the class. Here we have an A pointer so it will access A's static function.

3. **B::f2**

f2() is virtual and B has overwritten A's f2() so B's f2() is called when the A pointer tries to call f2().

4. **A::f3**

f3() is virtual so all that happens when an A pointer tries to call f3() is A's f3 will be called, even if B has decided to make f3() virtual (which it hasn't in this case).

5. **Err**

f5() is private and can not be called publicly in this context.

6. **Err**

Even though B has implemented f6() we are using an A pointer and the A pointer does not know what f6() is so this gives an error.

7. **0**

When "new B(2)" was done it first constructed an A(0) and then the B(2). As this is an A pointer it will point to the val belonging to the A object inside of the B object.

8. **A::f4 B::f5**

For questions 8 through 11 the reference will exhibit the same virtual behaviors as the pointer does.

9. **A::f1**

10. **B::f2**

11. **A::f3**

12. **A::f4 A::f5**

Even though we set "A a = b" all that this did was do a byte for byte copy of the A inside of b to a. This means that in every way this will still just function as a plain A object would for questions 12 through 15. It will never exhibit any characteristics of B.

13. **A::f1**

14. **A::f2**

15. **A::f3**

16. A(1)

Here when we construct A(1) it calls the constructor A(int i) and prints A(1).

17. A(3) C()

C derived from A. This means that when the constructor of C is called it must first construct A as that is what it derives from. The C() constructor calls A(3) so we see A(3) C() printed.

18. A(4) C() B() D()

The class which everything else derives from needs to be constructed first which is A(4). After that construction is done in the order that they are declared “class D : public C, public B”, therefore C() is printed next then B(). After this D() can finally be constructed.

19. 1

A had its value set to 1 in in A(1) and therefore 1 is printed.

20. 3

As C did not declare its own “val” it uses A’s val. When the default C constructor is called it calls A(3) which sets val to 3, and therefore 3 is printed.

21. 4

In the D constructor it can be seen that it explicitly has an A(4) which is constructed. The D C and B all point up to the same singular A object which was given this 4 value, so when val is printed it prints 4.

22. Ok

We are putting an A* into a vector of A*s so this is ok.

23. Err

A and B are completely unrelated classes so trying to but an A* into a B* gives an error.

24. Err

Just like A* and B* are unrelated, Vector<A>* and Vector<B>* are also unrelated, so trying to set one equal to the other makes no sense and is a compile time error.

25. if

Here we send a double where it expects a float but this is allowed as it can do a conversion from double to float. I think this may give a warning on some compilers (I think) but it wouldn’t ever cause an error.

26. ab

Here we use a C style cast to turn the int into an A as A has defined a constructor which takes an int so it can convert the int into an A. If the constructor A(int) was not there this would not be possible.

27. 0

In aaR.f(aR, i) a.val was set equal to zero, and it was passed as a Java reference which means that its value changed inside the aR present in the main when that function was called.

28. -1

val is set to -1 in the f function that was called.

29. Ok

When we set ‘a = null’ inside that function the reference ‘A a’ which was a parameter to null. While it is a reference to aR it is a copy of that reference and that copy is set to null the original reference in the main is not set to null.

30. 50

i was passed by value and therefore any modification to it in the function did not affect the value in the main.

31. foo 1 foo f 2 f1

foo(1) goes to the try which calls bar(i) which checks that i (1) is greater than zero so it throws E1. E2 is derived from E1 and you can not take a base class where you expect a derived class so instead of being caught by the first catch which expects an E2, it is caught by the catch that takes an E1, printing “foo 1”. Then even when an error is being thrown the code in the finally block will always execute no matter what. Therefore after we throw the new E2 we end up in the finally block and print “ foo f”. The E2 is caught back in the main where it is caught by the catch which expects an E2 and prints “2”. The finally code is then called and prints “f1”.

32. foo 2 foo f 1 f2

Here foo(-1) goes into the try that calls bar(-1) which sees that i is not greater than zero and throws an E2. This E2 is caught in foo’s first catch and “foo 2” is printed. A new E1 is thrown but we still go through the finally block, printing “foo f”. This then ends back up in the main were as we stated in the previous problem we can not catch an E1 as an E2 so we go on to the second catch that expects an E1. We then print “1” and go on to the finally statement where we print “f2”.

33. yes

The run function is indeed synchronized and this means that they are effectively locked based on what object they are a part of. The two threads were created on separate objects so this does nothing to serve as a lock that they must share. When t1 and t2 start they will complete their run in parallel and both attempt to modify s at the same time. s is static and therefore shared since they are instances of the same class and it will be a race condition for who is the last to write to it or what value it will end up.

34. no

We are in a similar situation here but instead of i being shared between the two objects i here is a member variable, local to each instance of a D object. Therefore there is no race here as there is no shared variable they are trying to modify.

35. any answer is ok[1]

In this context the main thread would terminate and the other threads would continue to run until they finished.

36. Err

C1 does not implement its own clone function so this does not work.

37. C2 C

This function has no compile time errors but it does not make its own arrays so it will be referencing the same array as what it copied.

38. 100

c2 and c2New point to the same array so a change made in one will affect the other.

39. 0

In C3’s clone it actually makes it own new array in the clone so they actually have independant arrays and a change to c3 is not a change to c3New.

40. A::f4 A::f5

Methods in java are usually virtual by default with a few exceptions. Here we call f4() which is not overwritten in B so of course we call A's f4(). When we call f5() it does not exhibit virtual behavior as it is an exception to the rule, as private members in Java are not virtual.

41. A::f1

Any static method is always tied to whatever class it is a part of, so the A reference will always call A's static methods.

42. B::f2

Here f2() is overwritten by B and since both methods are virtual the A reference will end up calling B's f2() method.

43. B::f3

Unless I'm missing something this seems to be the exact same situation as the problem above.

44. Err

f5() is private and therefore can't be called in this context. It would have to be public.

45. Err

A has no f6() method so even through B implements it the A reference does not now what it is so it can not be called.

46. 2

When the constructor for B was called it first had to construct the A object. When A was constructed it set val to 0, but then when B's constructor ran it overwrote that value to become 2. Therefore when it is printed it prints 2.

47. 1

When we construct A it sets val to 1 and when we print it it prints 1.