Question 1. (6 points)

a) AA
b) BA
c) CC
d) AA
e) CA
f) AC

If we explicitly show all inherited methods and color different method families, we get:

```
class A {
        void m(A a) { System.out.println("AA"); }
        void m(C a) { System.out.println("AC"); }
}
class B extends A {
        void m(A a) { System.out.println("BA"); }
        void m(B a) { System.out.println("BB"); }
        void m(C a) { System.out.println("AC"); }
}
class C extends B {
        void m(A a) { System.out.println("CA"); }
        void m(B a) { System.out.println("CB"); }
        void m(C a) { System.out.println("CC"); }
}
```

```
A a1 = new A();
A a2 = new B();
A a3 = new C();
B b1 = new B();
B b2 = new C();
C c1 = new C();
```

a) **compile time**: a1 is an A, b2 is a B
start in A, find if any method family matches exactly the parameters that we have in the call.
No exact match (we have m(A a) and m(C a) but no m(B a)). Is there any match that would make the call possible at all? We can't use m(C a) when a is a B because a hypothetical assignment C a; a = new B(); would not be valid and therefore is disallowed. m(A a) is a valid match because a hypothetical assignment A a; a = new B(); would be valid (casting B to A is type safe). So, the red family is selected.
run time: a1 is an A, so the method from the red family in A: m(A a) => "AA"

b) **compile time**: a2 is an A, b1 is a B
start in A, find if any method family matches exactly the parameters that we have in the call.
No exact match (we have m(A a) and m(C a) but no m(B a)). Is there any match that would make the call possible at all? We can't use m(C a) when a is a B because a hypothetical assignment C a; a = new B(); would not be valid and therefore is disallowed. m(A a) is a valid match because a hypothetical assignment A a; a = new B(); would be valid (casting B to A is type safe). So, the red family is selected.

run time: a2 is a B, look in B, find an overridden method from the red family in B: m(A a) => "BA"

c) **compile time**: a3 is an A, c1 is a C
start in A, find if any method family matches exactly the parameters that we have in the call. There is an exact match (we have m(C a)). So, the blue family is selected.
run time: a3 is a C, look in C, find an overridden method from the blue family in C: m(C a) => "CC"

d) **compile time**: a1 is an A, a3 is an A
start in A, find if any method family matches exactly the parameters that we have in the call.
There is an exact match (we have m(A a)). So, the red family is selected.
run time: a1 is an A, so the method from the red family in A: m(A a) => "AA"

e) **compile time**: c1 is a C, a3 is an A
start in C, find if any method family matches exactly the parameters that we have in the call.
There is an exact match (we have m(A a)). So, the red family is selected.
run time: c1 is a C, so the method from the red family in C: m(A a) => "CA"

f) **compile time**: a2 is an A, c1 is a C
start in A, find if any method family matches exactly the parameters that we have in the call.
There is an exact match (we have m(C a)). So, the blue family is selected.
run time: a2 is a B, look in B, method m(C a) is not overridden in B which means that it has m(C a) which it inherited from A, find an inherited (non overridden) method from the blue family in B (it is exactly the same as m(C a) from A): m(C a) => "AC"

Question 2 (2 points)

False

Function subtyping: true subtype must have supertype parameters and subtype return. Double f(String) has String as the parameter which is not a supertype of Object.

Question 3 (1 point)

False

Transitivity:

```
a.equals(b) && b.equals(c) => a.equals(c)  (=> is implication)
  true      && false           false   (given to us)
            false              false
```

From the implication truth table (slide 11 of Specifications2.pdf)

```
S1    S2      S1 => S2
false false   true
```

Less formally, transitivity is only broken if x.equals(y) is true and y.equals(z) is true but x.equals(z) is false. If (x.equals(y) && y.equals(z)) is false, we can't really say anything about the transitivity of equals.

Question 4 (1 point)

False

See slides 36 and 37 from Generics.pdf. Generic classes have no inheritance relationship by the type parameter. In other words, "generic classes or interfaces are not related merely because there is a relationship between their types" (from https://docs.oracle.com/javase/tutorial/java/generics/subtyping.html).