Here we discuss how function modifies its parameters.

In Java, a variable is assigned a value by “=” sign. If variable is prime type, byte, short, int, long, float, double, boolean or char, then the value of variable is prime itself. If it is not prime, include array, String and any thing extends Object, the value of variable is a reference, which is an address pointing to some location in memory.

In Java, an instance of type can create in two ways.

1. Literally in code. For example, 99.9 creates float, “abc” creates String.
2. Use new operator. For example, new Car(), new array[10], or new List<String>().

If variable type is prime, then the created prime data is stored in the variable. When it passes as parameter to a function, Java Virtual Machine (JVM) will make a copy, a new prime data of the same type with the same value, and passes it into function. The function will work on the copy within the function, not the original variable. So, no matter how function changes its parameter within the function, it cannot change the original variable outside the function. Because they are separate and different things.

If variable type is not prime, then we call it Object type, including array and List. When creating Object type variable, JVM first creates an instance in memory, then assigns the instance’s reference to the variable. When a variable is passed to a function, JVM passes the reference of the instance stored the variable. In the function, code works the instance which input parameter refers to, which is the same instance created outside the function. Let’s say we have a class ***Car*** and a function ***void doggify(Car pCar, String name) {pCar.setName(name);}*** and following code

*Car car = new Car();*

*doggify(car, “abc”);*

*System.out.println(car.getName());*

The first line creates a Car instance in memory and assigns the instance reference to variable *car*.

The second line passes the reference of Car instance created in the first line as a parameter *pCar* to doggify function. Remember *pCar* variable in doggify function refers to the same instance created in the first line.

The third line output is “abc”. Because in the whole process, there is one Car instance, which is referred by the outside variable *car* and the function parameter *pCar*. In other words, variable *car* and parameter *pCar* refer to the same instance. So, the outside variable car *can* “see” the change parameter *pCar* has in the function.

Simply say, if a variable is passed to a function as copy, then the variable cannot “see” the change in the function. But if a variable is passed to a function as reference, then the variable can see the change in the function.

String type is a special. It is not a prime type. But it is passed to a function as a copy, not a reference. That is why the right answer of Question 22 in Sample 1 is “Cuddles” not “Killer”. Question 21 is to mislead you.

Here is another tricky question. Let’s change Question 22 as below

*public static String doggify(String s, String name) {*

*s = name;*

*return s;*

*}*

*public static void main(String[] args) {*

*String cuddles = “Cuddles”;*

*cuddles = doggify(cuddles, “Killer”);*

*System.out.println(cuddles);*

What will output be? Answer is Killer. Because the original cuddles value is overwritten by the doggify’s return value.

Finally, let talk about String compare. Let’s say we have

*String s1 = “ABC”;*

*String s2 = new String(“ABC”);*

*s1 == s2 -> false.* Because s1 and s2 are two separate/different instances

*s1.equals(s2) -> true.* So, when compare to Strings, always use equals() function.