# **Primitive Types**

There are 8: ****boolean , byte , char , short , int , long , float****and****double****.

They don't have any method. Because primitive type variables are directly pointing to the ****value**** not object.

Can be compared with ==

# **Non-primitive Types**

Annotations. Class. Interface. Enum. Array.

They are pointing to the reference of an object.

Object o = new Object();Object o1 = o;// o1 and o are pointing to the save object

Compare with == : Will compare the reference value ( the pointed location)

Compare with equals(Object obj) : Will call the equals method of the object (default is ==, but we can override)

o1 == o; // true

o1.equals(o); // true, because by default equals is implemented by ==. Check the Object class.

class Apple{

String color;

Apple(String color){

this.color = color;

}}

Apple a1 = new Apple("RED");Apple a2 = new Apple("RED");

a1 == a2; // false; They are different object.

a1.equals(a2); // false; We didn't override equals().// to make a1.equals(a2) to return trueclass Apple {

//....

public boolean equals(Object obj){

// sanity check, type check

//...

Apple a = (Apple)obj;

return a.color.equals(this.color);

}}

# **Passed By Value**

All types are passed by value when calling a method.

For primitive types:

public void method1(){

int i = 100;

doSomething(i);

System.out.println(i); // 100}

public void doSomething(int i) {. // the i here is a local variable. value is copied.

i = 200; // because value is copied, changing a copied value won't change the i in

//method1();}

For reference types:

public void method1(){

Apple a1 = new Apple("RED");

doSomething(a1);

System.out.println(a1.color); // GREEN }

public void doSomething(Apple a){ // a can be any name, because it will be a copy of

// reference. a and a1 pointing to the same object, but

// they are difference references

a.color="GREEN"; // Object's content is changed. So both a and a1

//changed.

}

Reference value is difference from 'Object value', Objects are never passed, References are.

public void method1(){

Apple a1 = new Apple("RED");

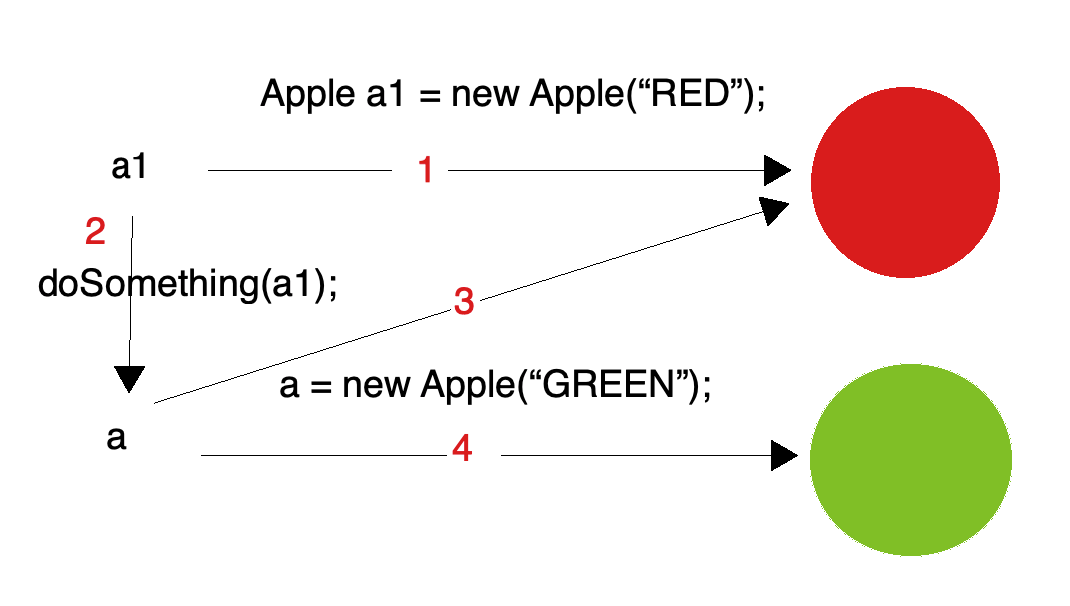
doSomething(a1);

System.out.println(a1.color); // RED }

public void doSomething(Apple a){

a = new Apple("RED"); // here we create a new Apple. and a is its reference now.

a.color="GREEN"; // so changing the new Object value won't do anything to a1.}



****Immutable Class vs Final Class****

https://www.youtube.com/watch?v=HiA5bSF\_wyA

https://www.youtube.com/watch?v=Epatg0uqwg0