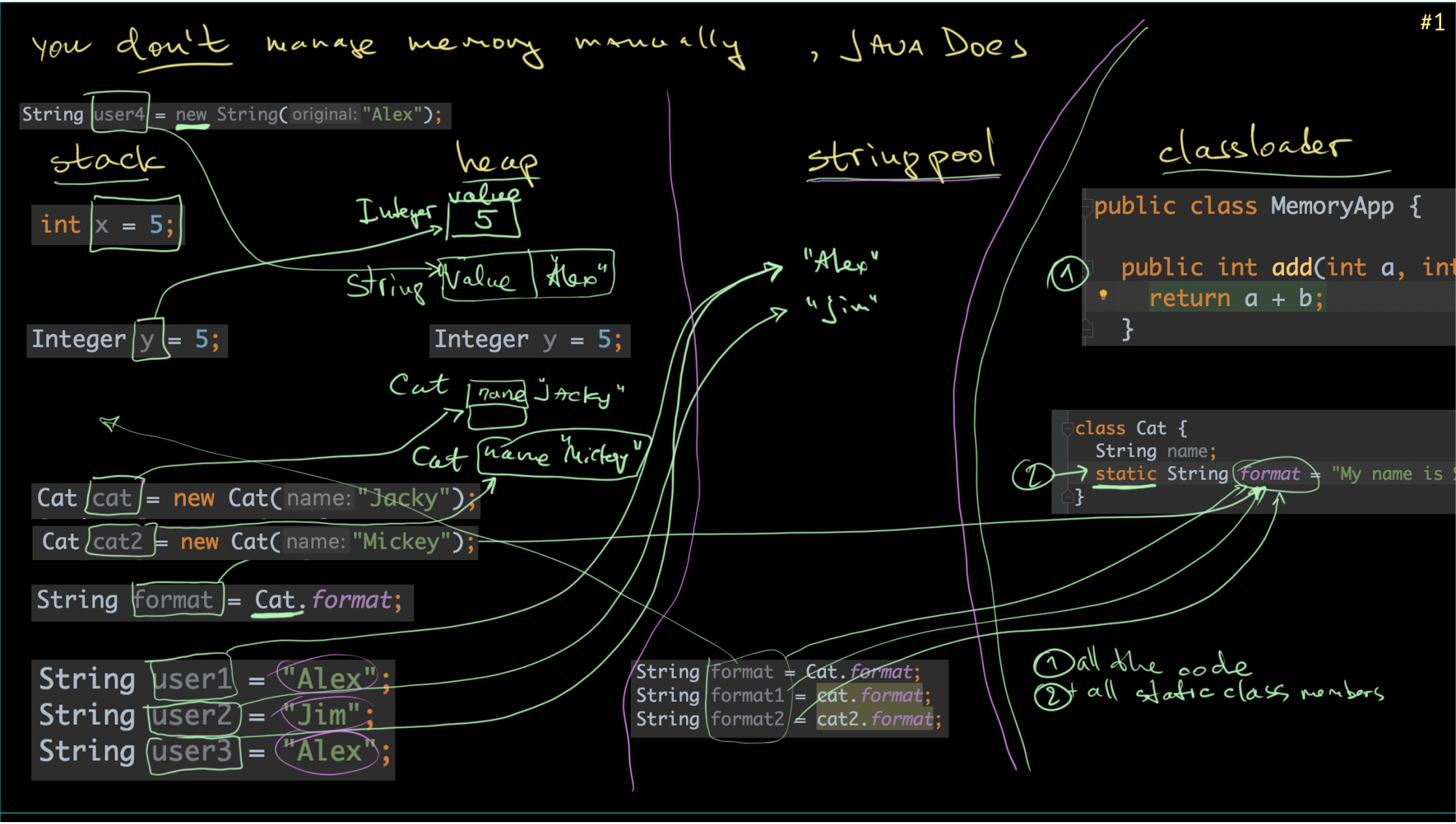
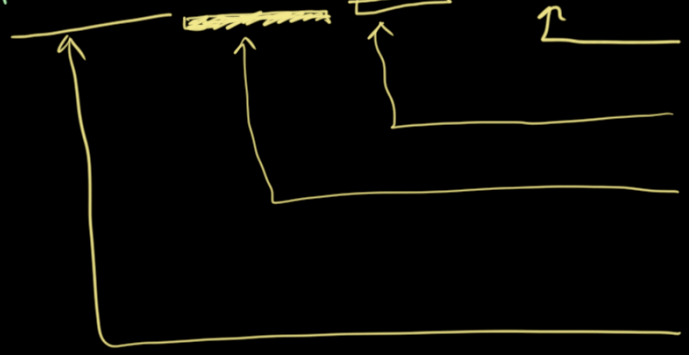
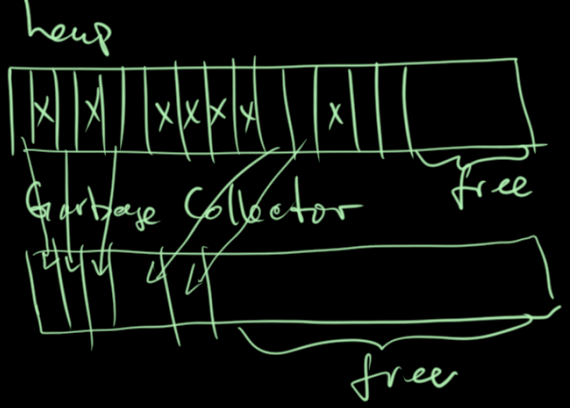

you don't manage memory manually, JAVA Does
String user4 = new String(original: "Alex");
stack
heap
string pool
classloader
int x = 5;
Integer
value
5
String
value
"Alex"
"Alex"
"Jim"
Integer y = 5;
Integer y = 5;
public class MemoryApp {
public int add(int a, int
return a + b;
}
Cat
name
"Jacky"
Cat
name
"Mickey"
class Cat {
String name;
static String format = "My name is
}
Cat cat = new Cat(name: "Jacky");
Cat cat2 = new Cat(name: "Mickey");
String format = Cat.format;
String user1 = "Alex";
String user2 = "Jim";
String user3 = "Alex";
String format = Cat.format;
String format1 = cat.format;
String format2 = cat2.format;
① all the code
② + all static class members

Java → compiling javac App.java => App.class
→ running java app.class

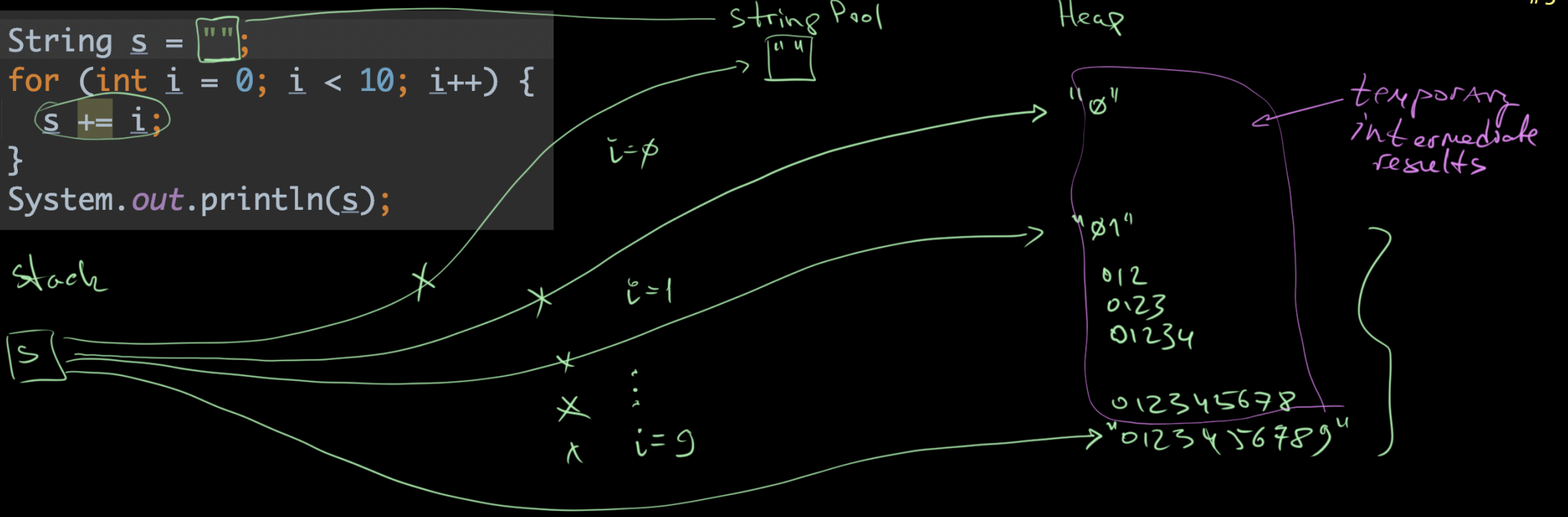
1. everything (classes) loaded to classloader area
2. public static void main. ← entry point



just a reserved word (inherited from C)
we return "nothing"
nobody creates instance of the main class
it needs to be accessible from JVM (outside)



```
String s = "";
for (int i = 0; i < 10; i++) {
  s += i;
}
System.out.println(s);
```



String is immutable

```java
StringBuilder sb = new StringBuilder();
for (int i = 0; i < 10; i++) {
  sb.append(i);
}
System.out.println(sb.toString());
```

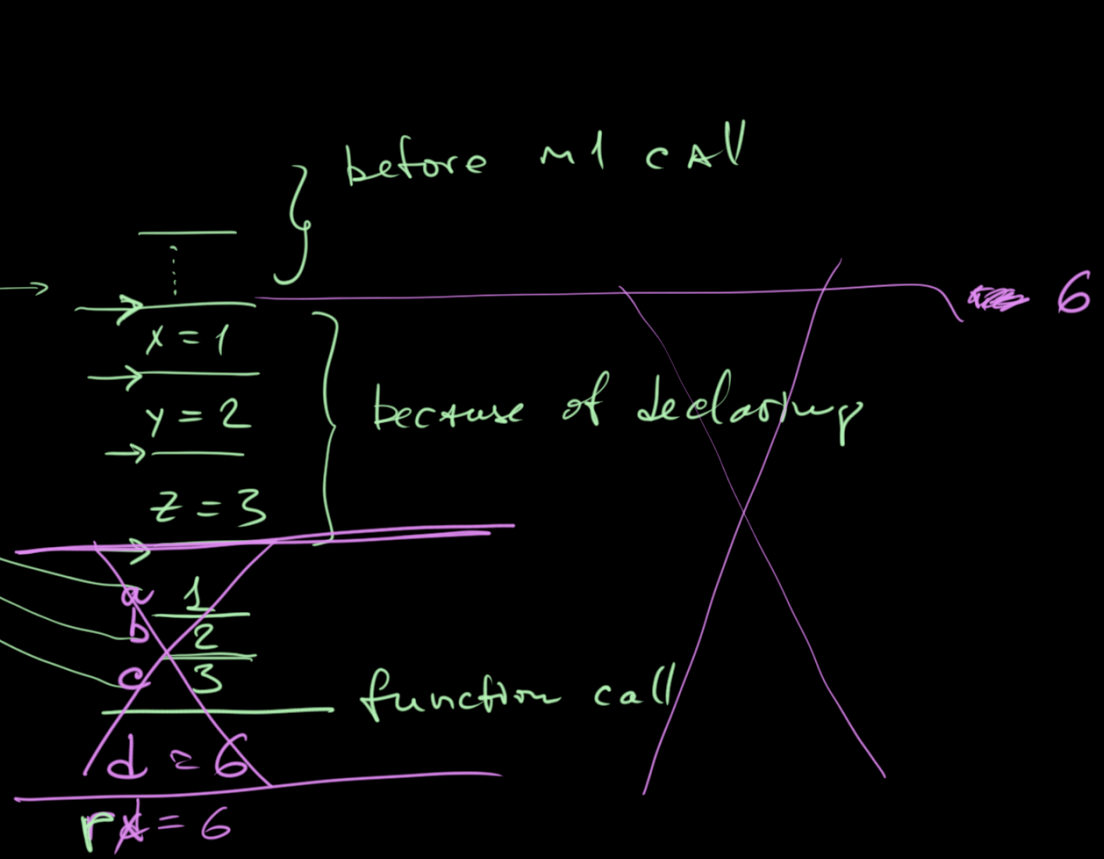
sb.append("0")
sb.append("1")

sb.toString traverses all things added and creates ONE string

stack

these parameters extracted from stack

```
public int m2(int a, int b, int c) {
  int d = a + b + c;
  return d;
}

public int m1() {
  int x = 1;
  int y = 2;
  int z = 3;
  int r = m2(x, y, z);
  return r;
}
```

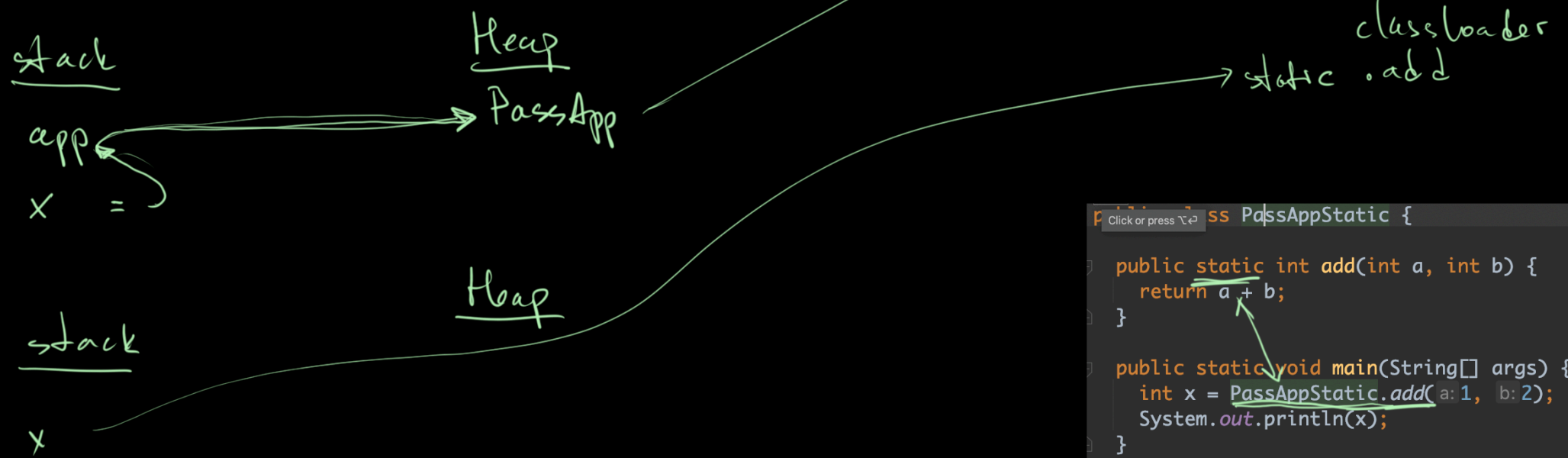


all parameters are passed via a stack

1. stack always holds everything {...}
2. stack is always cleaning when we reach ... }

pass by value / reference

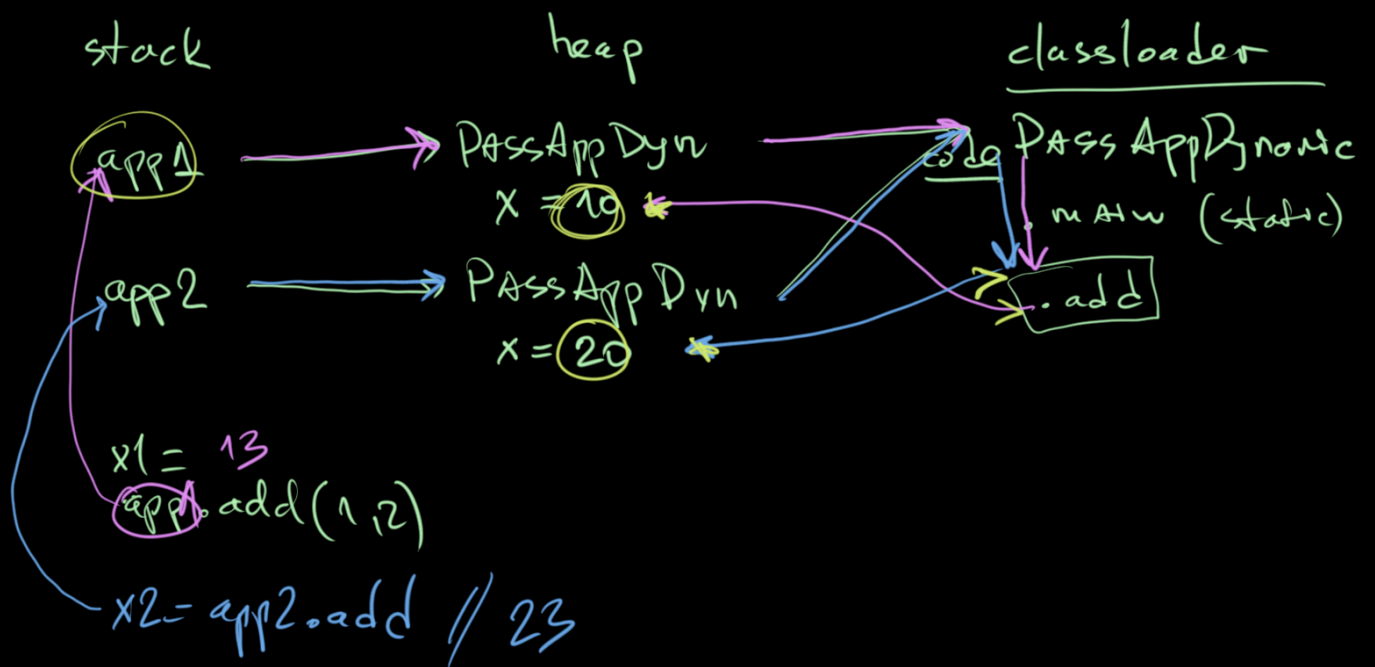
```java
public class PassApp {

  public int add(int a, int b) {
    return a + b;
  }

  public static void main(String[] args) {
    PassApp app = new PassApp();
    int x = app.add( a: 1,  b: 2);
    System.out.println(x);
  }
}
```



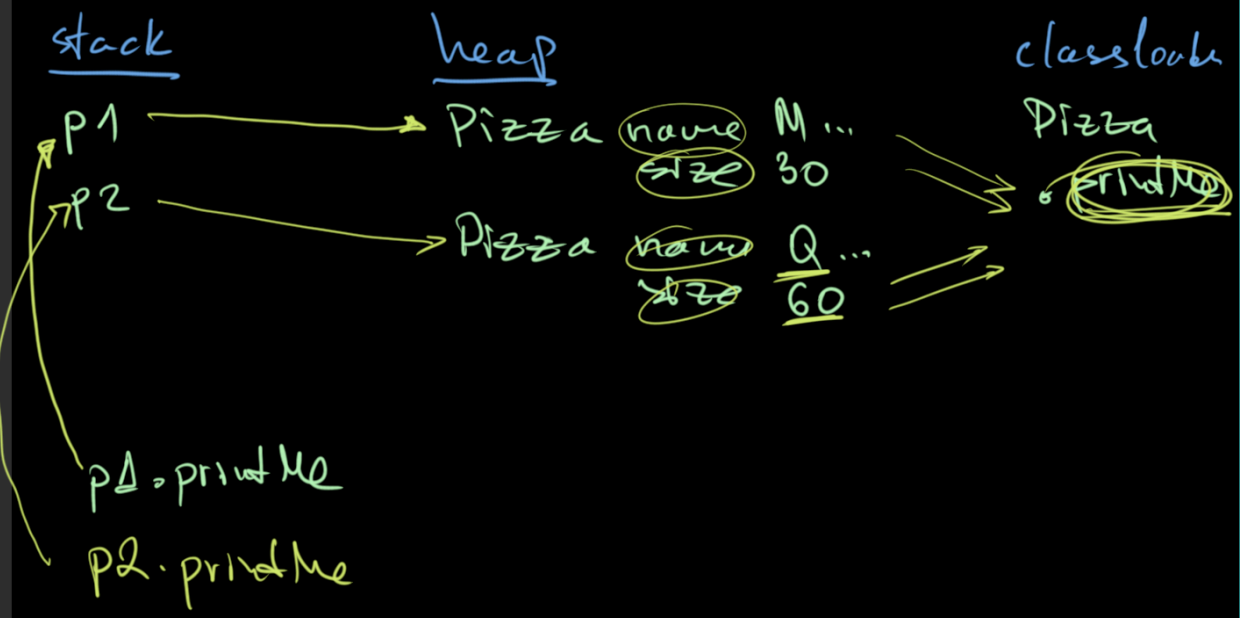

```java
Click or press ⌥⏎ ss PassAppStatic {

  public static int add(int a, int b) {
    return a + b;
  }

  public static void main(String[] args) {
    int x = PassAppStatic.add( a: 1,  b: 2);
    System.out.println(x);
  }
}
```

```java
public class PassAppDynamic {

  int x = 10;

  public int add(int a, int b) {
    return a + b + x;
  }

  public static void main(String[] args) {
    PassAppDynamic app = new PassAppDynamic();
    int x = app.add(a:1, b:2);
    System.out.println(x); //13
  }
```

```java
public class PassAppDynamic {

  int x;

  public PassAppDynamic(int x) {
    this.x = x;
  }

  public int add(int a, int b) {
    return a + b + x;
  }

  public static void main(String[] args) {
    PassAppDynamic app1 = new PassAppDynamic(x:10);
    PassAppDynamic app2 = new PassAppDynamic(x:20);

    int x1 = app1.add(a:1, b:2);
    int x2 = app2.add(a:1, b:2);
    System.out.println(x1); // 13
    System.out.println(x2); // 23
  }
```



```java
class Pizza {
  String name;
  Integer size;

  public Pizza(String n, Integer s) {...}

  @Override
  public boolean equals(Object given) {...}

  public void printMe() {
    System.out.printf(
      "Pizza name %s of size %d\n",
      name, size);
  }

  public static void main(String[] args) {
    Pizza p1 = new Pizza( n: "Margarita",  s: 30);
    Pizza p2 = new Pizza( n: "Quattroformaggi",  s: 60);
  } p1.printMe();
    p2.printMe();
}
```



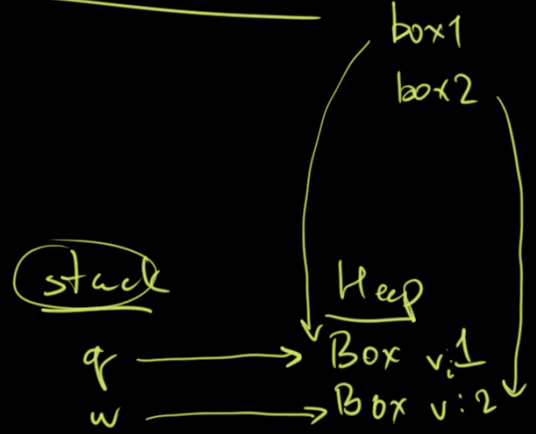
Pass by Value

```
public int add(int a, int b) {
  int r = a + b;
  a = 10;
  b = 20;
  return r;
}

public void doSomething() {
  int q = 1;
  int w = 2;
  int x = add(q, w);
  System.out.println(x); // 3
  System.out.println(q); // 1
  System.out.println(w); // 2
}
```

this change doesn't affect 'q' 'w'

any changes "allowed"

copied to stack



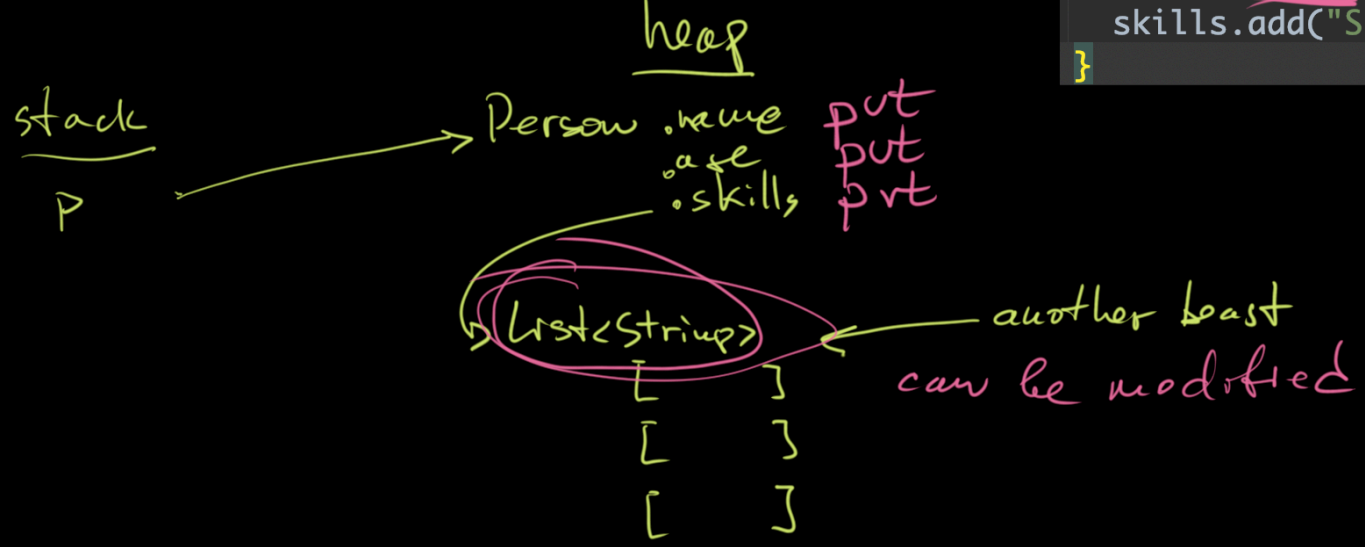
Pass By Reference

```
class Box {
  int value;

public int add(Box box1, Box box2) {
  int r = box1.value + box2.value;
  box1.value = 10;
  box2.value = 20;
  return r;
}

public void doSomething() {
  Box q = new Box( value: 1);
  Box w = new Box( value: 2);
  int x = add(q, w);
  System.out.println(x); //3
  System.out.println(q); // 10
  System.out.println(w); //20
}
```

it doesn't copy whole Box
it copies Link (Ref)

```java
class Person {
  private String name;
  private Integer age;
  private List<String> skills;

public List<String> getSkills() { return skills; }
```

```java
public static void whatever(Person p) {
  List<String> skills = p.getSkills();
  skills.clear();
  skills.add("Scala");
}
```




```java
Person p = new Person( name: "alex",  age: 33,  ...skills: "Python", "Java");
System.out.println(p);
whatever(p);
System.out.println(p);
```

```
Person{name='alex', age=33, skills=[Python, Java]}
Person{name='alex', age=33, skills=[Scala]}
```

getters

- do have more precise control on field usage

! what is the reason do have a public getter to a private field

mostly we do that to split/separate public contract and Implementation

```
public List<String> getSkills()
```

can be different

```
private List<String> skills;
public List<String> getSkills() {
  return skills;
}
```

```
private Map<String, Integer> skills;
public List<String> getSkills() {
  return new ArrayList<>(skills.keySet());
}
```

setters