Agenda:

1. Constructors
2. copy constructor
3. Deep copy / Shallow copy
4. Static keyword

Constructors

```
Student st = new Student();
```

- `Student` → Custom datatype
- `st` → variable
- `new` → keyword
- `Student()` → class Name / constructor

```
access-modifier Student() {
    // initialise
}
```

- it always returns an object of type Student.
- ↳ unparameterised constructor
  - default
  - custom

```
Student ( int age , String name) {
    // initialise the values
}
```

parameterised Constructor

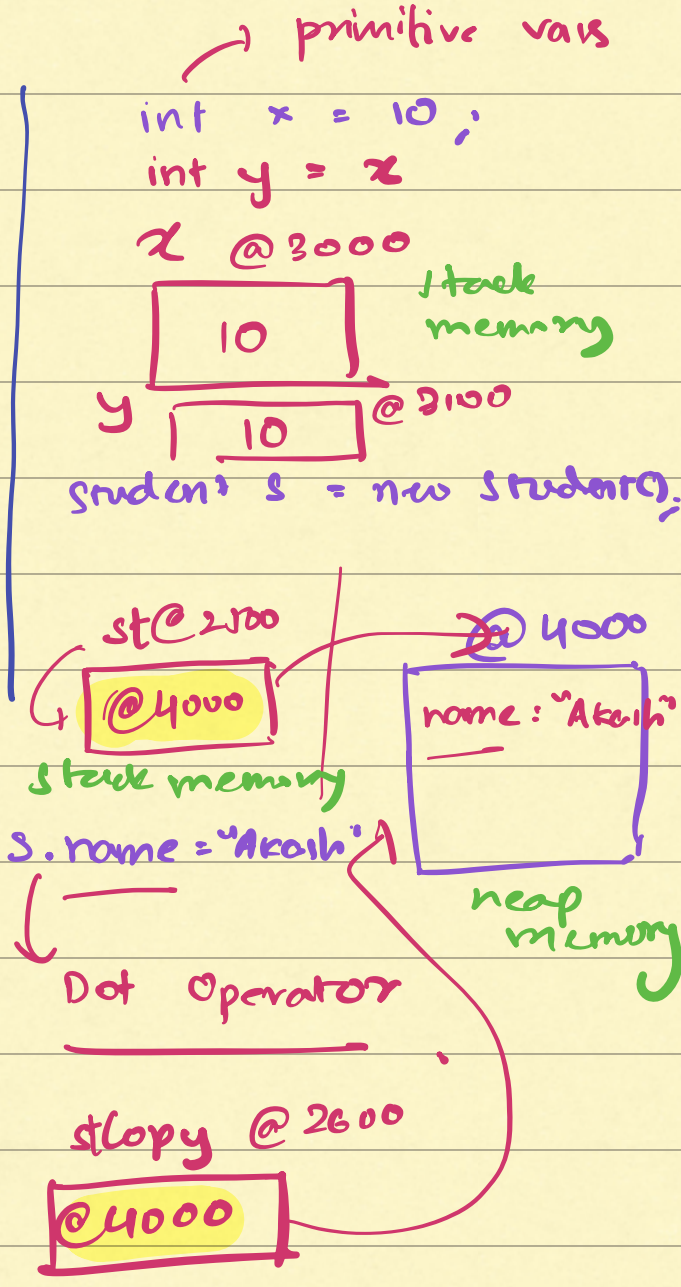
* Java only provides default constructor if no custom custructor is provided.

```
____ = new Student ( );
2    new Student( 15, "Akash")
```

constructor overloading / function overloading

copy Constructors

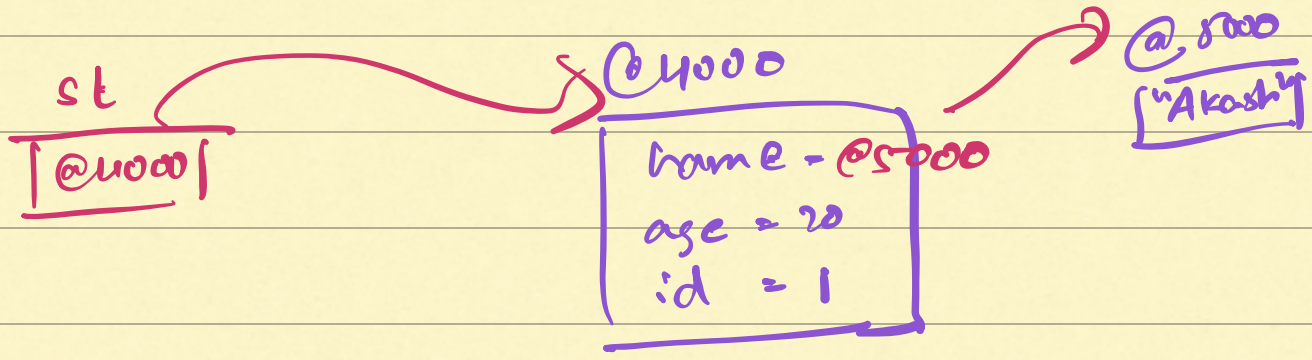
```
Student st = new Student();
Student stCopy = st;
print(stCopy.name)
```

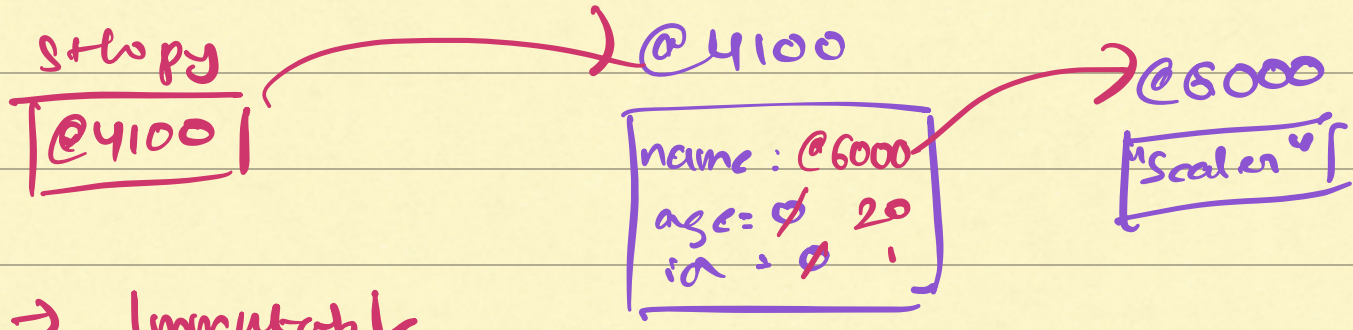


Right way to copy :

```
Student st = new Student();
Student stCopy = new Student();
{ stCopy.name = st.name; // new String(st.name)
  stCopy.age = st.age;
```

stcopy.id = st.id;





String → Immutable

stcopy.name = "Scaler"

// stCopy.name = new String("Scaler")

Problems with this approach:

1. Too much wdc everytime / not reusable
   manual errors

2. Private Variables can't be copied.

Constructor — inside class
need to create an object.

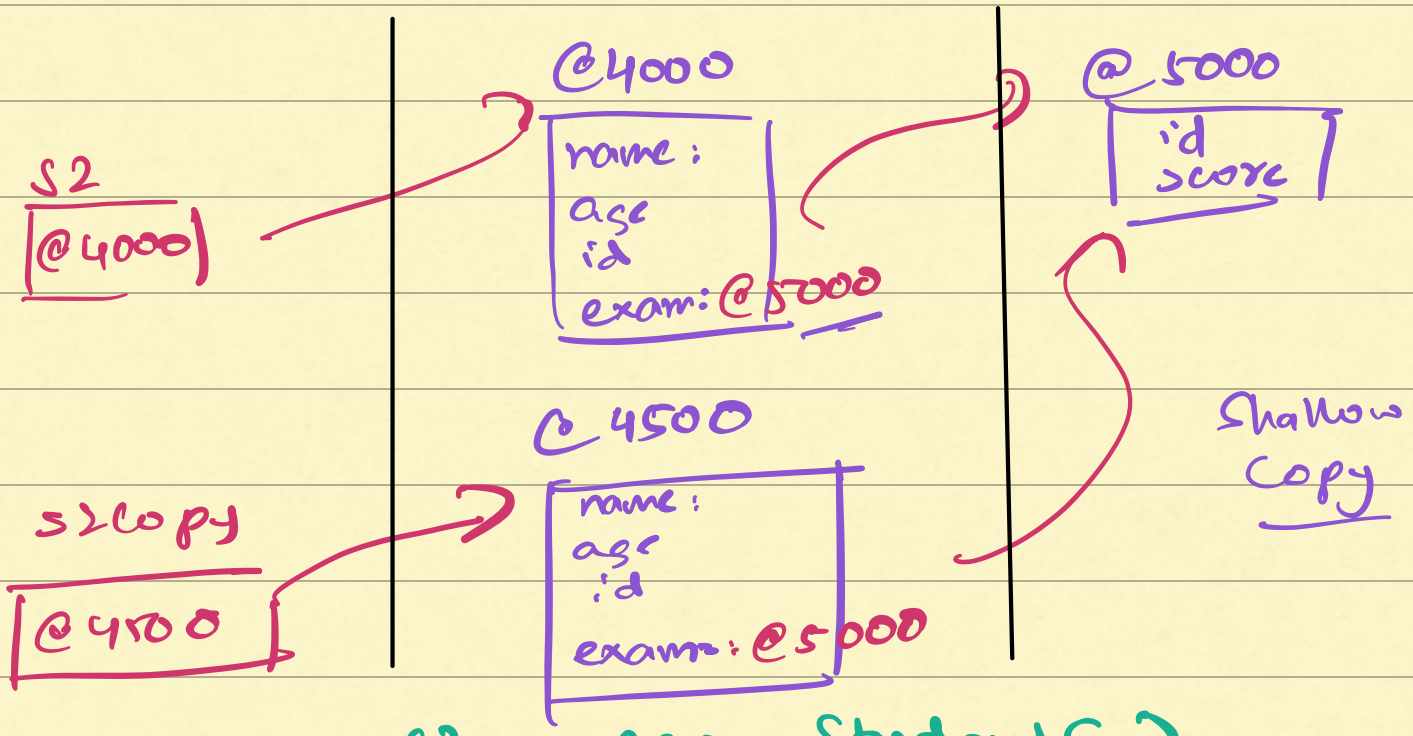
Student {

```
public Student() {
    // name = "temp";
    ⋮
}
// copy constructor:
public Student(Student st) {
    name = st.name;      | st.getName()
    age  = st.age        | st.getAge()
    id   = st.id;        | st.setId()
    gy   = st.gy;        | st.getGradYear()
}
```

inside class

```
{
    Student st = new Student();

    Student stCopy = new Student(st);
}
```

Deep Copy Vs Shallow Copy



Student s2 = new Student();

student s2Copy = new Student(s2);

```
//copy constructor
{ ...
    this.exam = s.exam;
}
```

reference variable.

this.exam = new Exam(s.exam);

copy constructor written in Exam

```
class Student {
    int id;
    :
    Exam exam;
    Database db;
}
```

Pass by Value / Pass by Reference

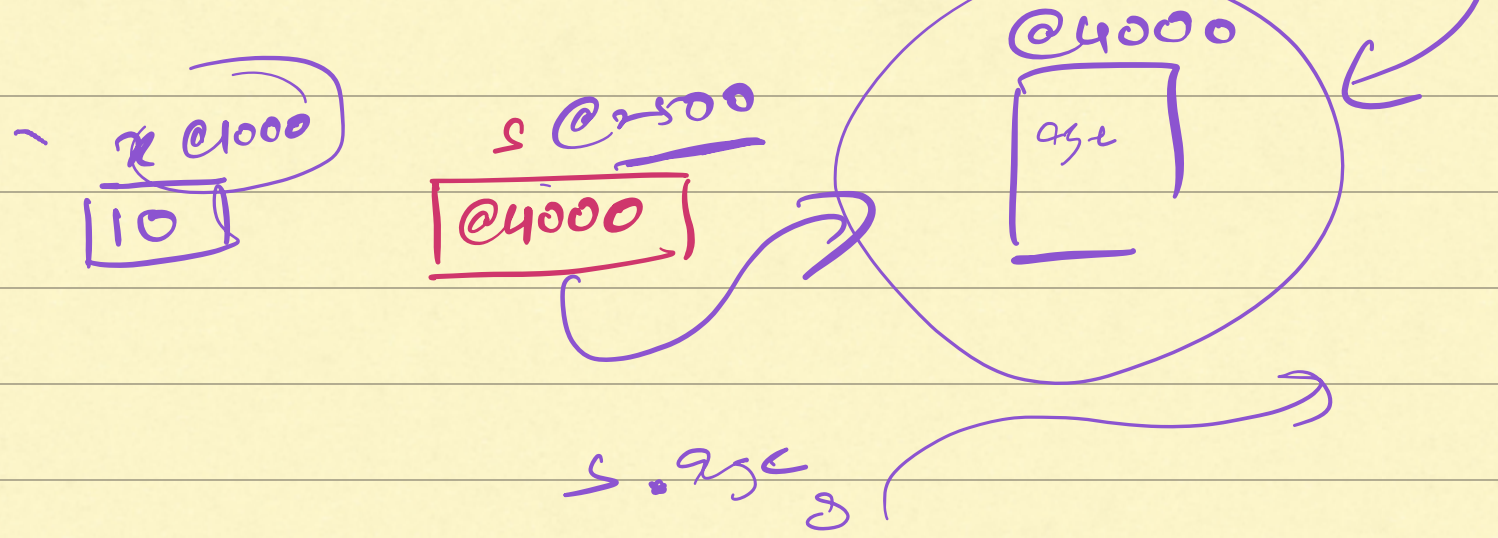
↳ Always

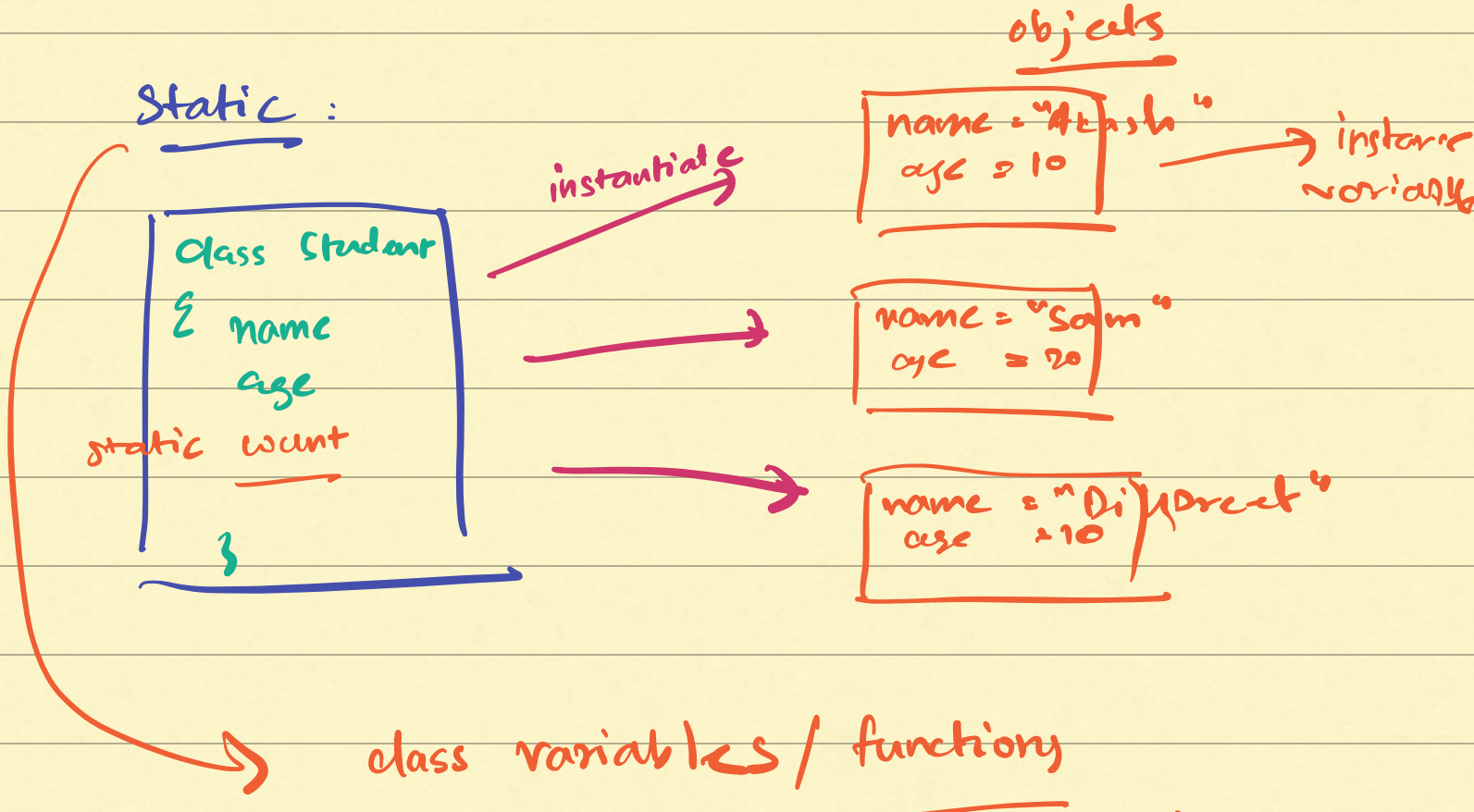
```
doSomething (int x) {
    x = 20;
}

int x = 10;
doSomething(x);
print(x); → 10
```

@4000

```
doSomething (Student s){
    s.age = 20;
}

Student s = new ...
s.age = 10;
doSomething(s);
print(s.age);
```

↳ 20





class variables / functions

Student . count = 0 ;

Classname . staticVar

. staticfunc