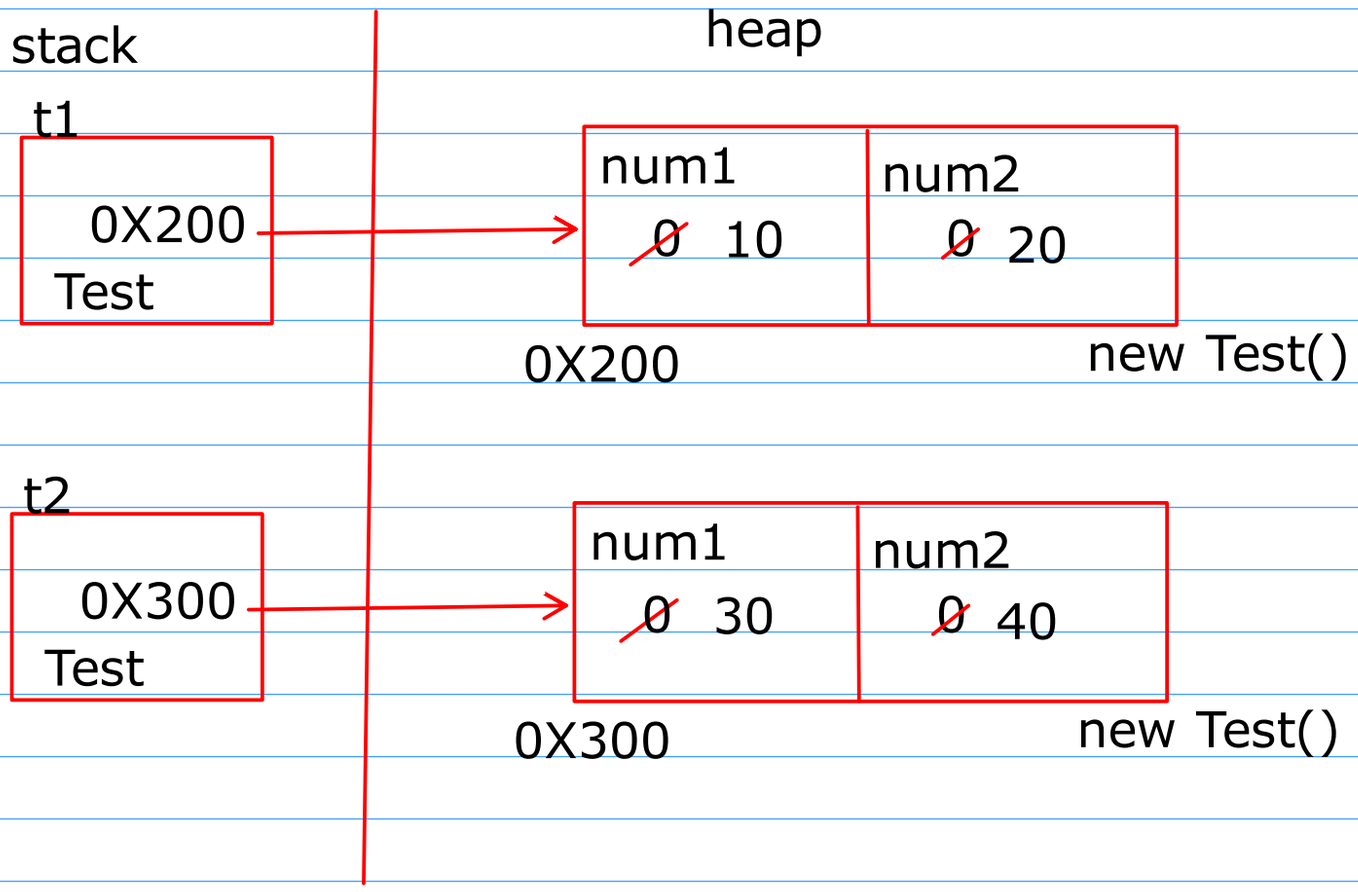
static

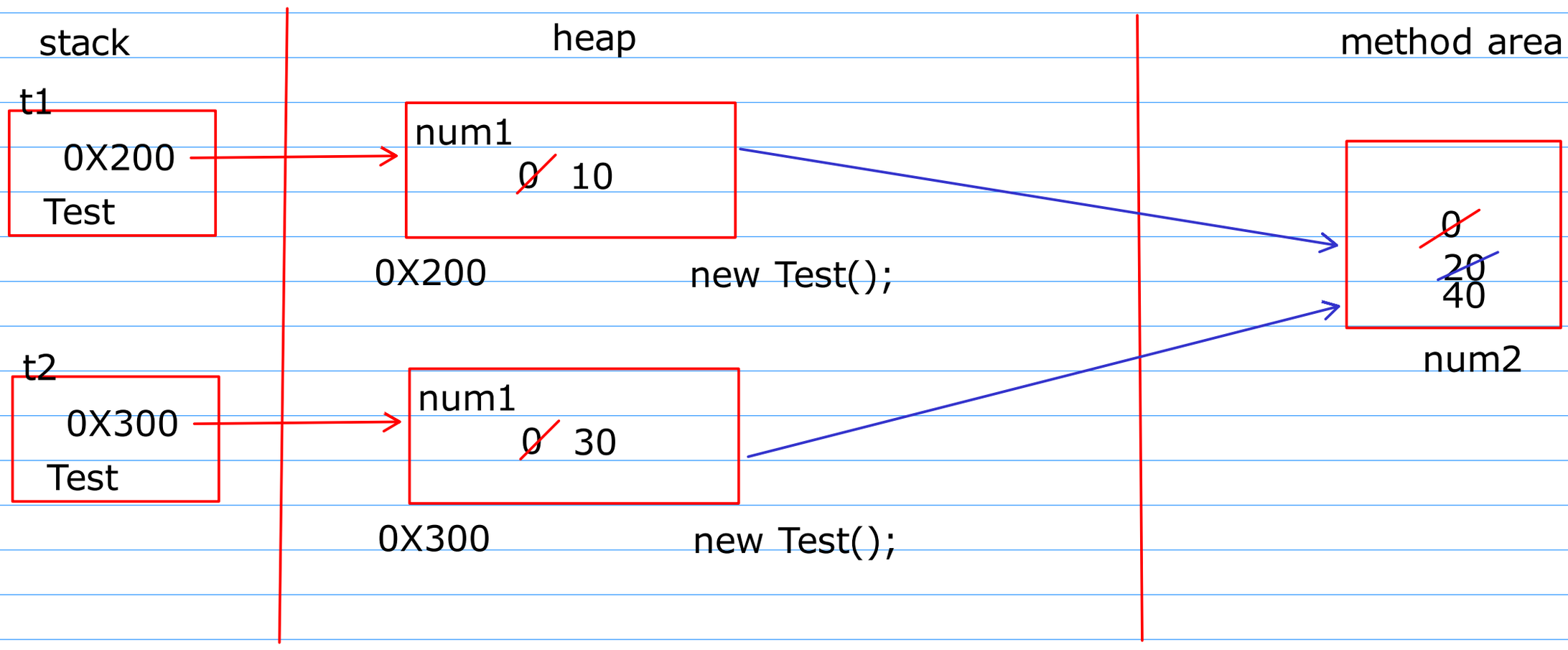
- fields
- methods
- block

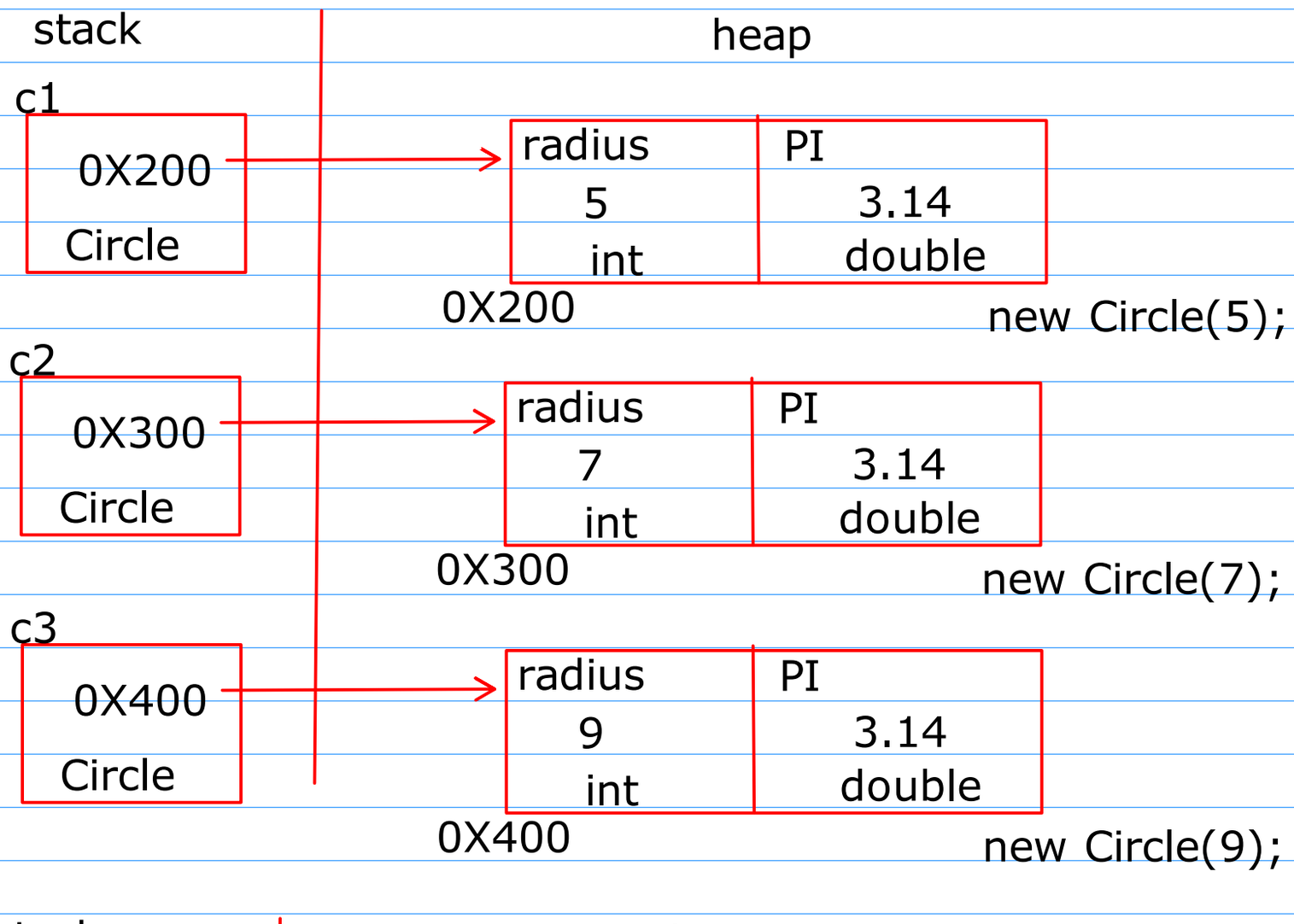
class consists of

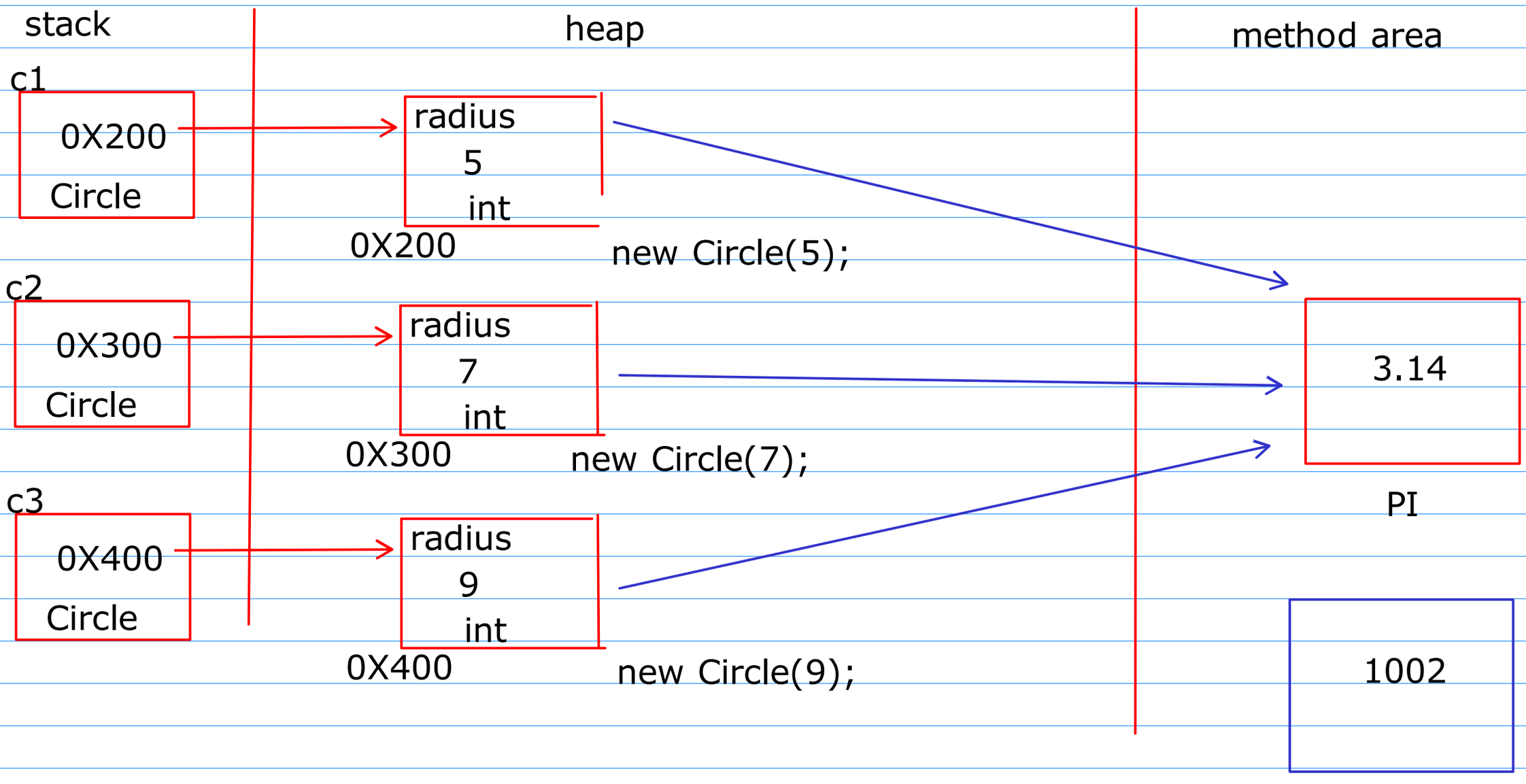
1. Fields
2. Methods

- fields and methods can be static as well as nonstatic









JVM -> Program01.main()

Program.main()

```
class One{
}
class Program{
public static void main(String []args){
        }
}
```

```
javac Program.java
Program.class One.class
java Program // name of .class
```

Static Field

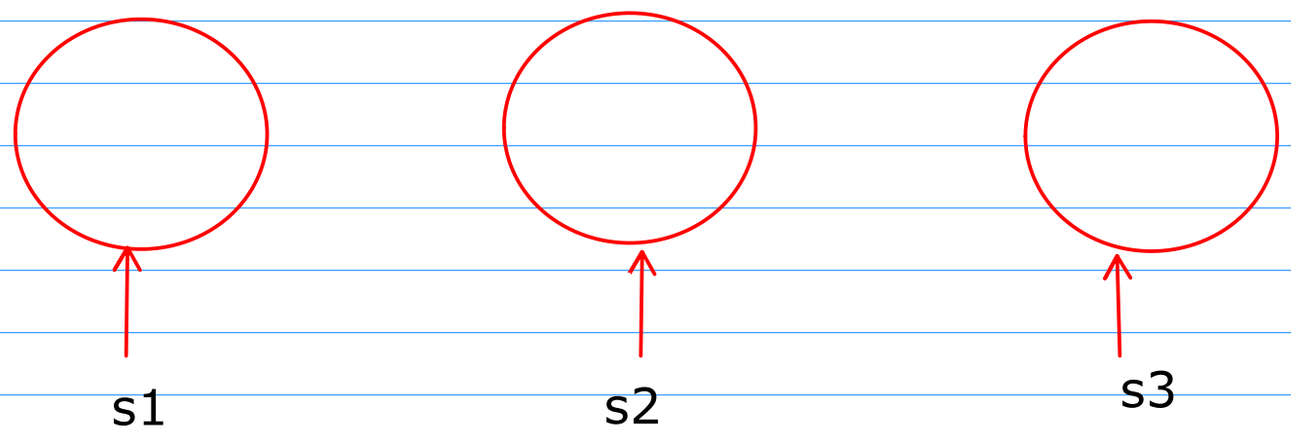
- Static fields are designed to be shared across multiple objects
- The memory for them is allocated on the method area only once at the time of class loading
- These fields can be initialized in the field initializer or in the static block
- These are designed to be accessed on class name and not on objects.

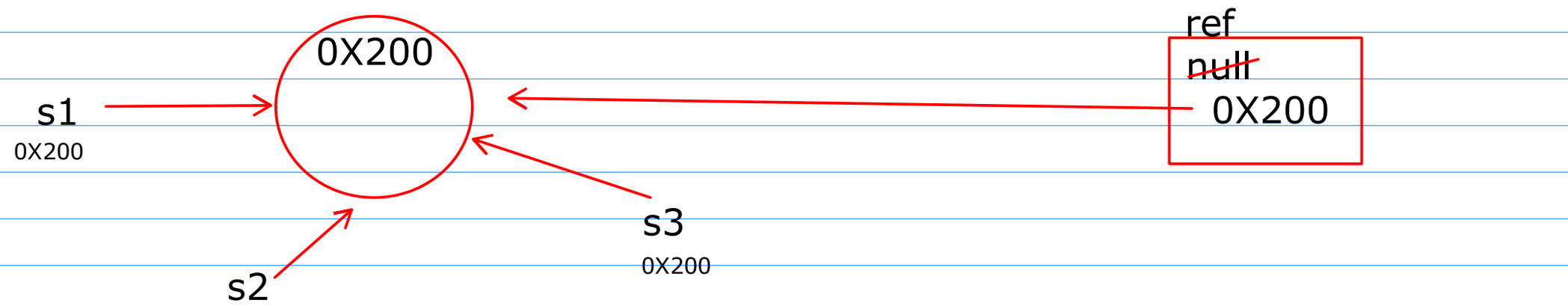
Static Methods

- The static methods are designed to be accessed on classname using . operator
- these are not designed to be accessed on objects, hence this refrence is not given for the
- static methods
- As this reference is not provided we cannot access the non static fields inside the static methods
- We can only access the static fields inisde the these methods

Singleton Design pattern

- Its a design pattern that is used to create only 1 instance of the class.
- We cannot create multiple instances of the singleton class





Assignment Question

```
class Employee{
int id; // make id as autoincrement
String name;
double salary;

accept();
display();
displayAllEmps(Employee[] arr)
findspecificEmpById(Employee[] arr)
}
```

menu driven

```
main(){
Employee[]arr = new  Employee[5];
int index = 0;
arr[index] = new Employee();

}
```

Hirerachy
- Reusability
- Their are two types of relationship
    1. has-a relationship (Association)
    2. is-a relationship (Inheritance)

Major Pillars
1. Abstraction
2. Encapsulation
3. Modularity
4. Hirerachy

Association
- If has-a relationship exists between 2 entities we use association
- It is further classified in 2 types
    1. Composition
        - If their is tight coupling between the 2 entities then use composition
            Dependent has-a Dependency
        - eg -> Human has-a Heart
            Car has-a Engine
    2. Aggegration
        - If their is loose coupling between the 2 entities then use aggegration
            Dependent has-a Dependency
        - eg -> Room has-a Window
            Employee has-a Car

```
class Date { // dependency
int day;
int month;
int year;

}
```

```
class Employee{ // dependent
int id
String name
double salary
Date doj; // Association (Composition)
Vehicle veh; // Association (Aggegration)
}
```

Date of joining
day
month
year

```
class Vehicle{
String name;
String liscence_no;
}
```

Employee has-a DateOfJoining
Dependent Dependency
Association

We create the object of dependency class as a field inside the dependent class

Employee has-a Vehicle

c++ -> by defualt it forms composition
java -> by default it forms aggegration