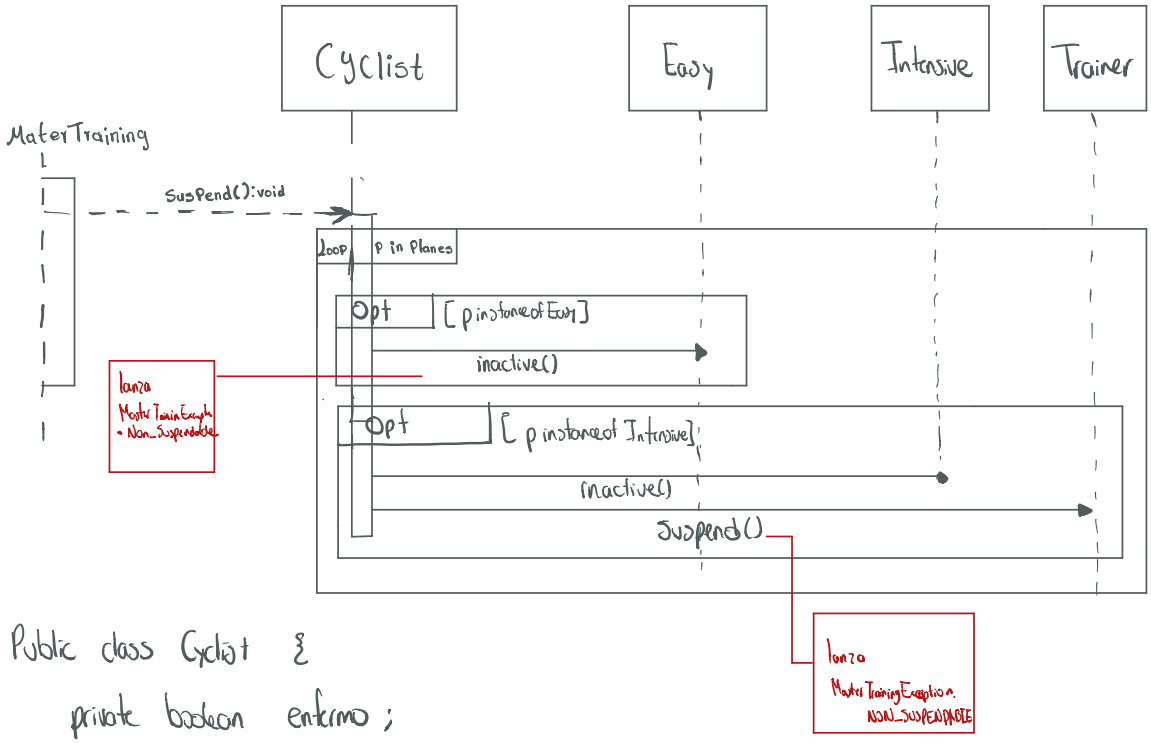
Taller

1. Implementando

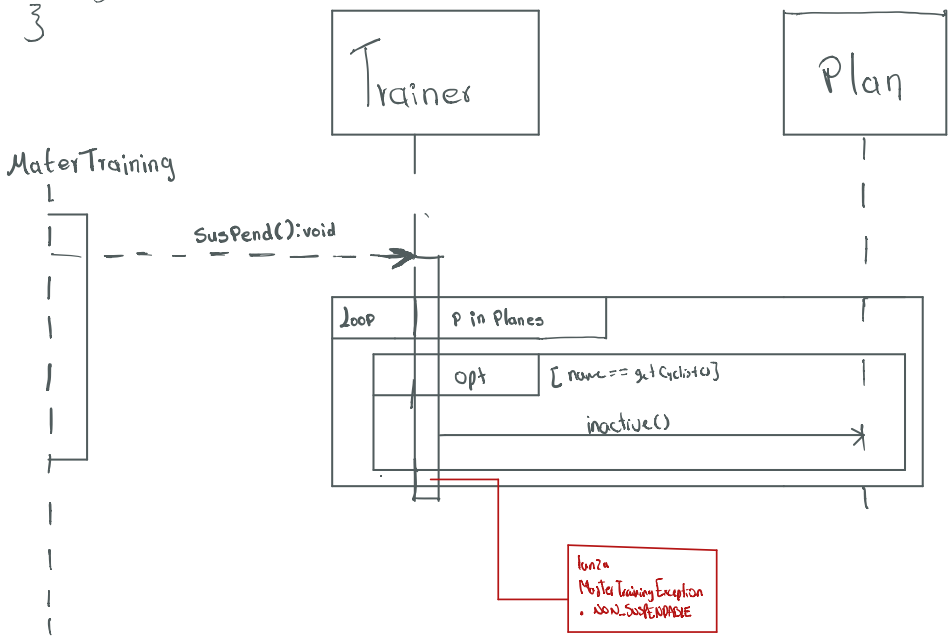
Implemente los métodos suspend de las clases Cyclist y Trainer (No olvide MDD) -

//SuspendIble <Interface>




```
Public class Cyclist {
    private boolean enFermo;
    private ArrayList<Plan> plans;
    Public Cyclist() {
        enFermo = false;
        plans = new ArrayList<Plan>();
    }
    public void suspend() throws MasterTrainingException {
        for (Plan p: plans){
            if (p instanceof Easy) p.inactive();
            else if (p instanceof Intensive) {
```

```
            p.getTrainer().rName(name);
            p.getTrainer().suspend();
            p.inactive();
        }
    else {
        throw new MasterTrainingException(MasterTrainingException.NON_SUSPENDABLE);
        }
    }
}
```




```
Public class Trainer {
    private String name;
    private ArrayList<Plan> plans;
    public Trainer() {
        plans = new ArrayList<Plan>();

    public void suspend() throws MasterTrainingException {
        for (Plan p: plans) {
            if(name == p.getCyclist()) p.inactive();
            else throw new MasterTrainingException(MasterTrainingException.NON_SUSPENDABLE);
        }}
    public void rName(String n) {
        name = n;
    }
```

2. Diseñando

①

```
Public class MasterTraining {

    private ArrayList<Plan> plans;
    public MasterTraining () {
            plans = new ArrayList<Plan>(); }

    public void evaluatePlan ( long number ) throws MasterTrainingException {

            for (Plan p: plans) {

                if (p.eval() == false ) {
                    if (p.getActive() == false ){
                            if (p.getNumber() == number) p.evaluate();
                            else throw new MasterTrainingException ( MasterTrainingException. NOT_FOUND);
                    else throw new MasterTrainingException ( MasterTrainingException. OPEN ); }
                else throw new MasterTrainingException ( MasterTrainingException. INCOMPLETE);
                }
            }
    }
}
```

```
* public abstract class Plan {
  protected ArrayList<Week> weeks;
        public plan() {
          weeks = new ArrayList<Week>();
        }
  public abstract void evaluate();

  public boolean evaluatePlan throws MasterTrainingException {
```

```
public class Intensive extends Plan' {
public boolean evaluate () {

        boolean b = true;
        for ( Week w : weeks ) {

                b = b && w.evaluateIntensive();
        }
        return b
```

```
4 public class Easy extends Plan {

  boolean b = true;
  for ( Week w : weeks ) {

          b = b && w.evaluateEasy ();
  }
  return b
```