JAVA: SINGLETON, IMMUTABLE AND WRAPPER CLASS.

1. Singleton class.

- DB CONNECTION Connection one times and operation many times.
- This class objective to create only 1 and 1 Object.
- Diference ways of creating Singleton Class.

 Eager Initialization.
 Lazy Initialization.
 Synchronization block.
 Double Check Lock (There is a memory issue, resolved through Volatile instance variable).
 Enum Singleton

1.2. Eager Initialization.

- All Static variables are pre loaded while start the application.

Disadvantage:-

- Event though we are not using getInstance method but still object is going to be created and stored in memory.

```
public class DBConnection{
  private static DBConnection conn = new DBConnection();
  private DBConnection(){
  }
  public static DBConnection getInstance(){
    return conn;
  }
}
public class Main{
  public static void main(String[] args){
    DBConnection conn = DBConnection.getInsatnce();
  }
}
```

1.3. Lazy Initialization.

- I>t's over comes the problem of Eager Initialization.

Disadvantage:-

- W>hat if at the same time or concurrent time or simultaneously or In PARALLEL two Threads T1 and T2 want access Object, If Objects IS Null , It will Created TWO OBJECTS at the same time.

```
public class DBConnection{
  private static DBConnection conn = null;
  private DBConnection(){
  }
  public static DBConnection getInstance(){
    if(conn == null){
      conn = new DBConnection();
    }
    return conn;
  }
}
```

1.4. Synchronized Method.

- I>t Over comes the problem of Lazy Initialization.
- s>ynchronized does two thing it put's **lock** and **unlock** on block of method.
- L>et's Two Threads T1 AND T2 are calling getInstance block In Parallel.
- N>ow Only one thread is going to allow in getInstance block.

Disadvantage :-

- B>ecause of synchronization it's getting very slow.
- I>f we are calling synchronized getInsatnce block at 1000 of places for every calling it put's lock and unlock and getting so slow.

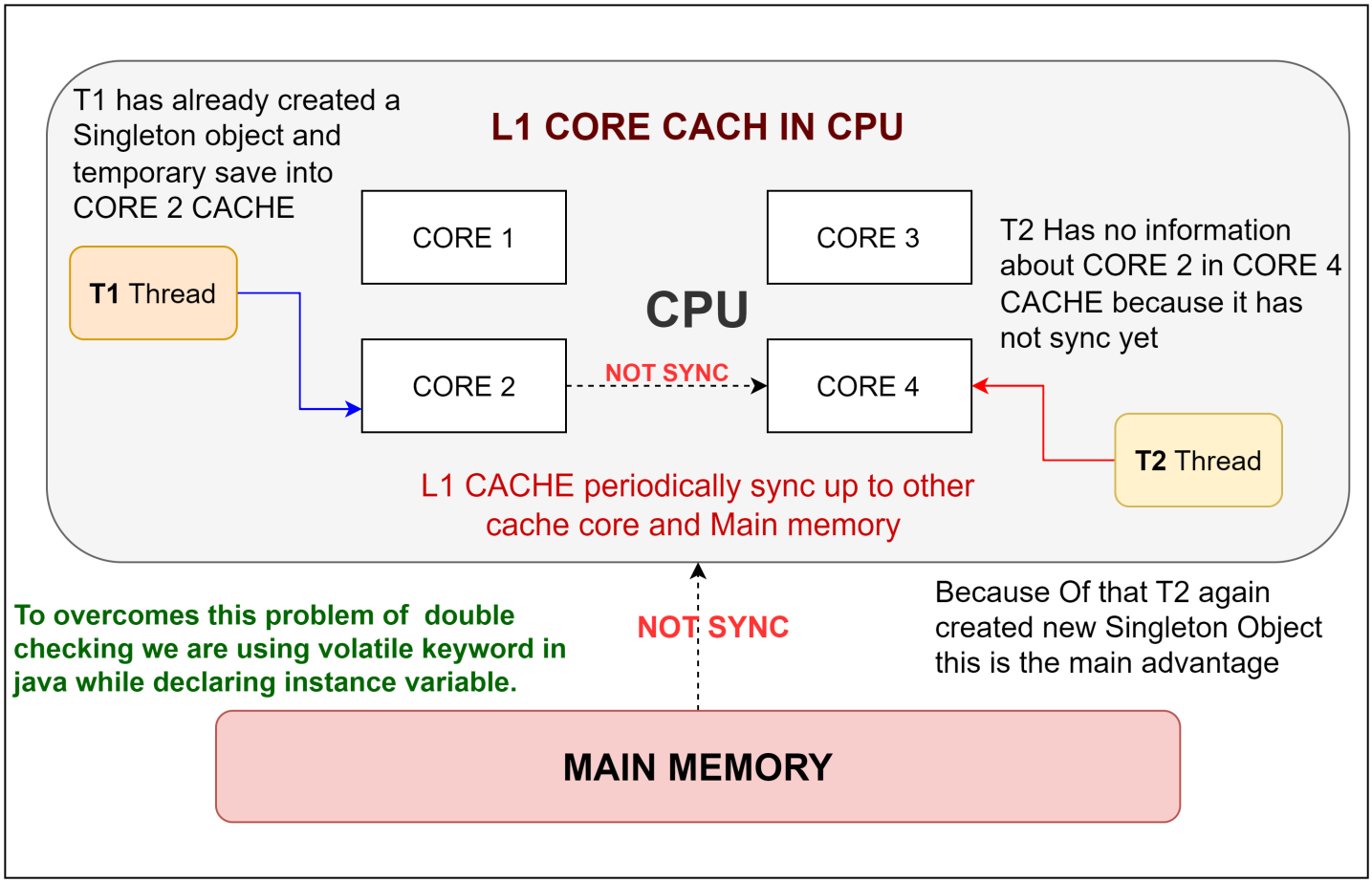
```
public class DBConnection{
  private static DBConnection conn = null;
  private DBConnection(){
  }
  synchronized public static DBConnection getInstance(){
    if(conn == null){
      conn = new DBConnection();
    }
    return conn;
  }
}
```

1.5. Double Checked Locking System.

- It's Over Comes the problem of Synchronized block.
- T1 and t2 comes in parallel in check1 if object is null then they will go inside synchronized block but it is synchronized only one thread allow. so let's t1 go inside and again check it's still null then it creates new instance.in case of t2 it will check again hey object is null it founds to be not null.

Disadvantage

- **Not proper Syncing Memory issue between cache and Main Memory.**
- **There is one more issue Memory Re-Ordering.**
- **This is also little bit slow because we are using synchronized and it's always put lock and unlock.**



volatile Keyword In Java.

> ***when you are creating object using the keyword volatile any read and write happen to the memory not CPU CORE CACHE.***

```
public class DatabaseConnection{
  private static volatile DatabaseConnection conn = null;
```

```
    private DatabaseConnection(){
    }
    public static DatabaseConnection getInstance(){
      if(conn == null){//check1
        synchronized (DatabaseConnection.class){
           if(conn==null){//check2
             conn = new DatabaseConnection();
           }
        }
      }
      return conn;
    }
  }
```

1.6. Bill Pugh Solution.

- not using volatile not using synchronized.
- Making Use Of Eager Initialization.
- Eager Initialization just put it into static nested class.
- This nested class do not loaded at the time of start of application and do not loaded into the memory.
- Nested class loaded at the time when they are going to be used

```
  public class DBConnection{
    private static DBConnection conn = null;
    private DBConnection(){
    }
    private static class DBConnectionHelper getInstance(){
      private static final DBConnection INSTANCE_OBJECT = new DBConnection();
      return conn;
    }
    public static DBConnection getInstance(){
      return DBConnectionHelper.INSTANCE_OBJECT;
    }
  }
```

1.7.Through ENUM.

- As Per JVM Only one Instance is present.
- They Are Singleton only.
- Only One Instance Per JVM.

```
enum DBConnection {
  INSTANCE;
}
```

2. IMMUTABLE CLASSES.

- We can not change the value of of an object once it is created.
- Declare class as 'final' so that it can not be extended.
- All classes members should be private. So that direct access can be avoided.
- And class members are initialized only once using constructor.
- There should not be any setters methods, which is generally used to change their member value.
- just getters method. And returns copy of the member variable.
- Example String, Wrapper classes etc.

```
final class MyImmutableClass{
  private final String name;
  private final List<Object> petNameList;
  MyImmutableClass(String name, List<Object> petNameList){
     this.name = name;
     this.petNameList = petNameList;
  }
  public String getName(){
    return name;
  }
  public List<Object> getPetNameList(){
    //this is required because making list is final.
    //means you can not now point it to new list
   //but still can add, delete values in it.
  //so that's why we send the copy of it.
  return new ArrayList<>(petNameList);
  }
}
public class Main{
  public static void main(String[] args){
    List<Object> list= new ArrayList<>();
    list.add("lola");
    list.add("pola");
    MyImmutableClass obj = new MyImmutableClass("myName", list);
    //it's only copy of petname list
    obj.getPetNameList().add("luffy");
    System.out.println(obj.getPetNameList());
  }
```

```
}
output:
[lola,pola]
```

3. Wrapper class has already been covered in depth.