## ⇒ Patterns ?

Design patterns are well established solution to commonly observed software design problems.

what ?

- ns gang of few → Design patterns (23)
- ns Head first design pattern
- 10 design patterns → most common
  Most pattern use case interviews

why

## ⇒ Creational Design pattern ?

mainly design with object creation

⇒ how many objects.

⇒ Validation

⇒ how Object Creation will work

Singleton ←

Builder

Factory ⇉

Prototype And Registry

# => Singleton

## Definition :-

Allow Creation of Single object from a Class

=

Only a single instance of a class should exist

1) Memory
2) Data Sharing
3) Concisfency
4) Should State
5) To avoid time to create new object

- DB connection
  - mysql connection
  - Radis config -


Class product
Db. db;

class usefetch
Db db.

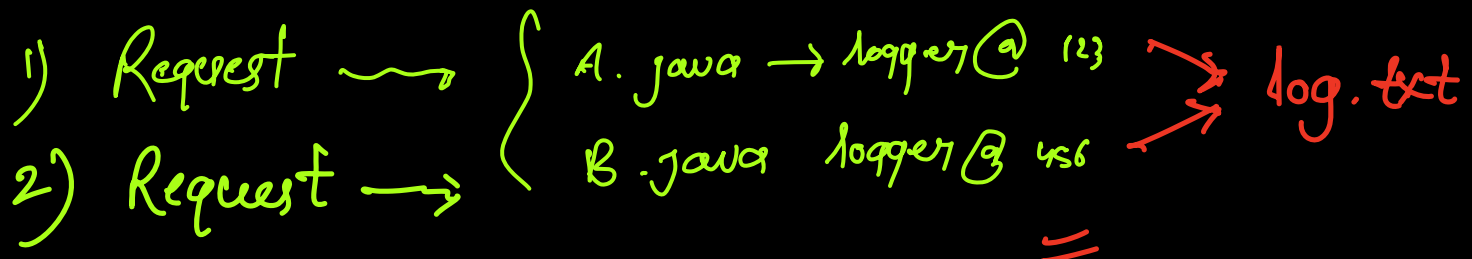
Class product
Db. db;

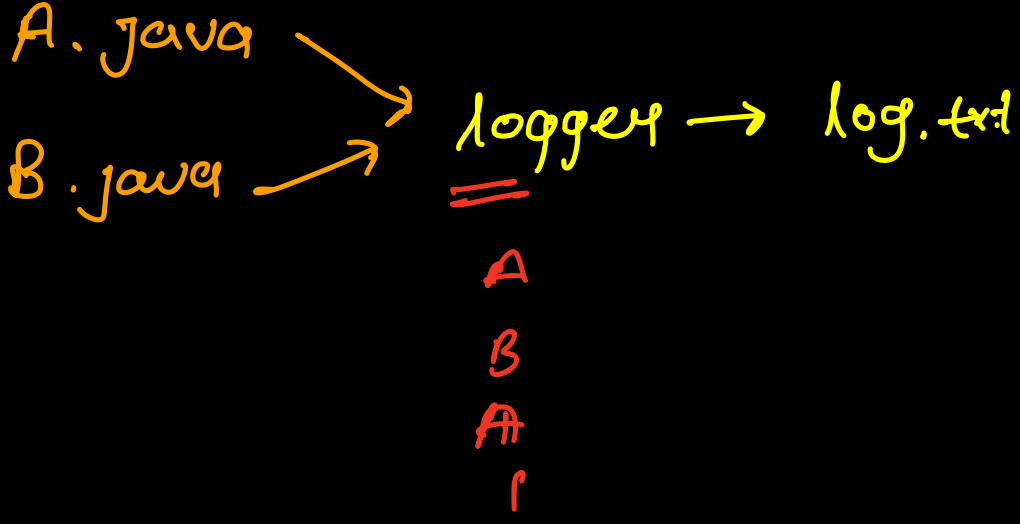
→ Applicatio

# Logger

```
Google logger logger = google logger.getInstance();
```

Only 1 Object for logger



- How to implement Singleton ?

Class RedisConfig {

String url;
String user;
String password
int port;

RedisConfig rc = new RedisConfig();

Make the constructor private !

None will be able to create object of that class

class ceti {

static getDate()

- class RedisConfig ?

String url ;
String user ;
String password
int port

private RedisConfig() { }

{ public static RedisConfig getObject() {
return new RedisConfig(); }

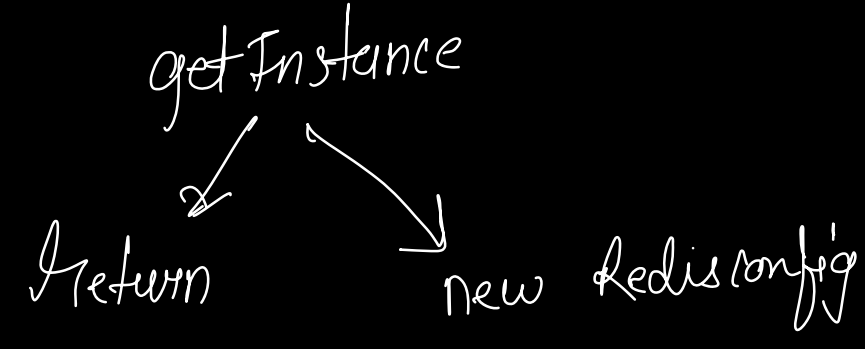
```
class RedisConfig {
    String url
    String port
    String password
    String user

    private RedisConfig() {
    }

    private static Instance = null;
    public static RedisConfig getInstance(){
        if (instance) {
            return instance
        } else {
            instance = new RedisConfig()
            return instance;
        }
    }
}
```

2 ways to implement

→ Eager Execution | Early loading

Create object application start → private static RedisConfig Instance = new RedisConfig();

→ getInstance → Create → lazy loading;




. will lazy loading methode be safe to use multithread env?

getInstance


T1


T2

```
if (Instance = null) {
    Instance = new
}
```

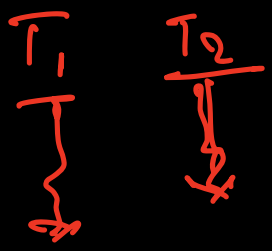
```
if (instance == null)
    instance = new
```

More than 1 object create.

```
Synchronized getInstance() {
    if (null)
        new
    Return Instance
}
```

Solves multiple object create problem

Instance = Null




public static Redisconfig getinstance() {

if (instance == Null)

{ if (instance == null) }

Instance = new Redisconfig

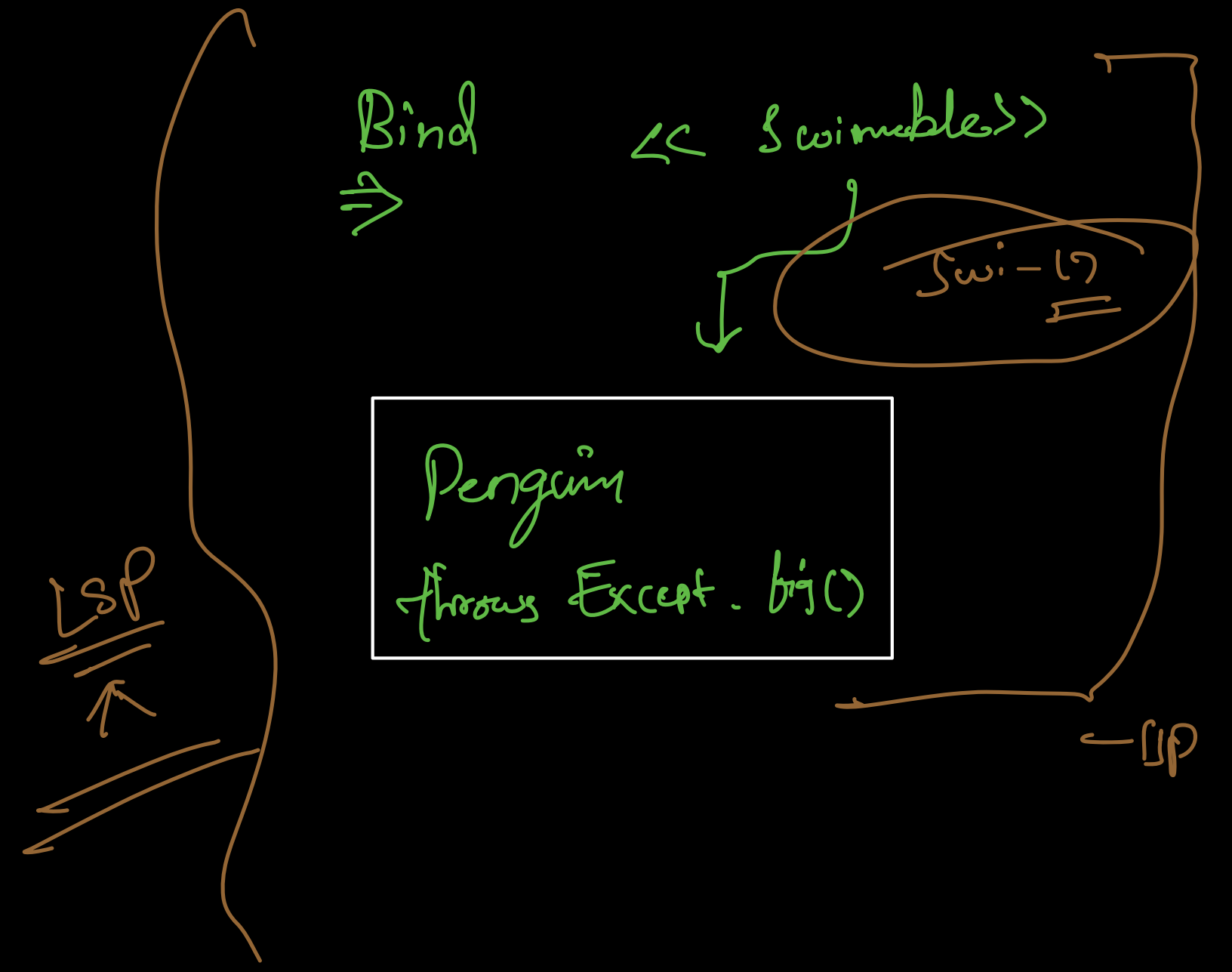
lock.unlock

}

Return Instance

Double check Locking

{ using Enum → getsingleton }


Bind
⇒
<< Swimmable >>
Swi-ly
Penguin
throws Except. big()
LSP
DIP

## a) Single-check (Lazy initialization Singleton)

```
public class RedisConfig {

    private static final RedisConfig instance =
            new RedisConfig("root", "123456", 6379, "http://127.0.0.1:6379");

    private String username;
    private String password;
    private int port;
    private String url;

    private RedisConfig(String username, String password, int port, String url) {
        this.username = username;
        this.password = password;
        this.port = port;
        this.url = url;
    }

    public static RedisConfig getInstance() {
        return instance;
    }
}
```

## ⑥ Double-checked locking (lazy and fast singleton)

```
public class RedisConfig {

    private static volatile RedisConfig
instance = null;
    private static final Lock lock = new
ReentrantLock();

    private String username;
    private String password;
    private int port;
    private String url;

    private RedisConfig(String username,
String password, int port, String url) {
        this.username = username;
        this.password = password;
        this.port = port;
        this.url = url;
    }

    public static RedisConfig getInstance() {
```

# 4) Eagerly Eager loading singleton :-

```
public class RedisConfig {

    private static final RedisConfig
instance =
            new RedisConfig("root",
"123456", 6379, "http://
127.0.0.1:6379");

    private String username;
    private String password;
    private int port;
    private String url;

    private RedisConfig(String
username, String password, int port,
String url) {
        this.username = username;
        this.password = password;
        this.port = port;
        this.url = url;
    }

    public static RedisConfig
getInstance() {
        return instance;
```