# Design Patterns

**Lazy Load:**

Tem a função de verificar se já existe uma instância da classe em uso caso não exista cria e retorna a nova instância, sendo que as próximas chamadas do método receberão sempre a mesma instância já criada.

Este pattern e utilizado para reduzir o uso de memória e CPU desnecessária pela aplicação, pois somente instância os objetos quando seu uso é realmente necessário.

Apesar de sempre trazer a mesma instância de um objeto este pattern não limita o número de instâncias possíveis de uma classe, pois não suprime o construtor.

Exemplos:

Delphi:

|  |
| --- |
| var FObject: TObject;  function getObject: TObject;  begin  if (not Assigned(FObject)) then  FObject:= TObject.create;  return:= FObject;  end; |

JS:

**var** object;

**function** **getObject** () {

**if** (!object) {

        object= **new** **Object**();

    }

**return** object;

}

Obs.: Geralmente utilizado dentro de outra classe para somente instanciar um objeto no momento em que começa a ser utilizado.

Pense em um formulário onde temos diversos processos e cada processo utiliza um objeto diferente.

Quando o usuário abre o formulário ele talvez utilize apenas um dos botões.

Então não temos a necessidade de subir instância de todos os outros objetos utilizados pelos outros botões.

Sendo assim se aplicaria este pattern para instanciar os objetos somente quando necessário.

**Singleton:**

Tem o objetivo de permitir uma única instância de uma classe dentro de uma aplicação, evitando problemas de referenciar o objeto errado na memória ao chamar um método ou ler uma propriedade.

Este tipo de design cria uma variável interna com acesso restrito para instanciar a classe, suprime o método de create e externa apenas um método getinstance que retorna sempre a mesma referencia de objeto.

Dentro do método getInstance é utilizado o pattern Lazy Load para trazer sempre buscar a mesma instância do objeto.

Exemplos:

Delphi:

|  |
| --- |
| unit uConfig;  interface  uses  System.SysUtils,  System.Classes;  type  TConfiguracoes = class(TComponent)  private  class var \_Configuracoes: TConfiguracoes;  var  FUrl: string;  protected  constructor CreateInstance(pOwner: TComponent);  public  constructor Create(AOwner: TComponent); override;  class function GetInstance(pOwner: TComponent): TConfiguracoes;  property url: string read FUrl write FUrl;  end;  implementation  constructor TConfiguracoes.CreateInstance(pOwner: TComponent);  begin  inherited Create(pOwner);  end;  constructor TConfiguracoes.Create(AOwner: TComponent);  begin  inherited;  raise Exception.Create('Erro ao instanciar a classe de Gerenciamente de Inatividade. Utilize a função GetInstance');  end;  class function TConfiguracoes.GetInstance(pOwner: TComponent): TConfiguracoes;  begin  if not Assigned(\_Configuracoes) then  begin  \_Configuracoes := TConfiguracoes.CreateInstance(pOwner);  end;  Result := \_Configuracoes;  end; |

JS:

**let** \_instance = **Symbol**();

**let** \_singletonEnforcer = **Symbol**();

**class** C**onfiguracoes** {

**constructor**(enforce) {

**if** (enforce !== \_singletonEnforcer) {

**throw**('Não utilize o constructor desta classe,

ao invés disto utilize o getInstance.');

         }

**this**.url= ””;

    }

**static** **get** **getInstance**() {

**if**(!**this**[\_instance]) {

**this**[\_instance] = **new** C**onfiguracoes**(\_singletonEnforcer);

        }

**return** **this**[\_instance];

    }

}

**var** c**onfiguracoes**= C**onfiguracoes**.getInstance;