Los archivos generado deben respetar el siguiente formato de nombre guiaDeClase01_ejercicioNumero.vhd
Si el archivo es un testbench el formato del nombre es el siguiente guiaDeClase01_ejercicioNumero_tb.vhd
Por ejemplo el archivo del ejercicio 1 debe llevar el nombre `guiaDeClase01_01.vhd` Si el ejercicio tuviera ítems a y b por ejemplo el nombre deberá ser `guiaDeClase01_01_A.vhd` para el punto A.
Todos los archivos deberán ser subidos al repositorio dentro de una carpeta con el nombre guiaDeClase01 junto con sus simulaciones (solo suba los archivos .vhd)
Ver guía holaComponente.pdf

1. Utilizando como componente el multiplexor implementado en la guia de clase 00.

```
entity myMux4_1 is
   Port ( a : in  std_logic_vector (3 downto 0);
          c : in  std_logic_vector (1 downto 0);
          s : out  std_logic);
end myMux4_1;
```

2. Implemente 4 buffer tri-State con entradas de control independientes. (Use when - else y for generate)

```
entity myTriState4 is
   Port ( a : in  std_logic_vector (3 downto 0);
          c : in  std_logic_vector (3 downto 0);
          s : out  std_logic_vector (3 downto 0));
end myTriState4;
```

3. Implemente N buffer tri-State con entradas de control independientes. (Use when - else; for generate y generic)

```
entity myTriStateN is
   Generic (N: integer := 4);
   Port ( a : in  std_logic_vector (N-1 downto 0);
          c : in  std_logic_vector (N-1 downto 0);
          s : out  std_logic_vector (N-1 downto 0));
end myTriStateN;
```

4. Utilizando como componente el sumador completo de 1 bit realizado en la guía de clase 00 implemente uno de 4 bits. (Use for-generate)

```
entity fullAdder4 is
   Port ( a, b: in std_logic_vector (3 downto 0);
          ci : in  std_logic;
          s  : out std_logic_vector (3 downto 0);
          co: out  std_logic);
end fullAdder4;
```

5. Utilizando como componente el sumador completo de 1 bit realizado en la guía de clase 00 implemente uno de N bits. (Use for-generate)

```
entity fullAdderN is
   Generic (N: integer := 4);
   Port ( a, b: in std_logic_vector (N-1 downto 0);
          ci : in  std_logic;
          s  : out std_logic_vector (N-1 downto 0);
          co: out  std_logic);
end fullAdderN;
```

6. Implemente un conversor gray a binario de N bits.

```
entity grayBinario is
   Generic (N: integer := 4);
   Port ( gray: in std_logic_vector (N-1 downto 0);
          binario  : out std_logic_vector (N-1 downto 0));
end grayBinario ;
```

7. Implemente un conversor binario a gray de N bits.

```
entity binarioGray is
   Generic (N: integer := 4);
   Port ( binario: in std_logic_vector (N-1 downto 0);
          gray  : out std_logic_vector (N-1 downto 0));
end binarioGray;
```

8. Utilizando como componente los dos VHDL creados en el punto anterior cree un conversor binario - gray cuando sel = 1 y gray - binario cuando sel = 0 seleccionable por una entrada de control.

```
entity conversiorBinarioGray is
   Generic (N: integer := 8);
   Port ( binario: in std_logic_vector (N-1 downto 0);
          gray  : out std_logic_vector (N-1 downto 0);
          sel: in std_logic);
end conversiorBinarioGray ;
```

9. (Integrador) Utilizando como componente fullAdderN y conversiorBinarioGray implemente un circuito que incremente en uno el número gray de la entrada.

```
entity incGray is
   Generic (N: integer := 16);
   Port ( entrada: in std_logic_vector (N-1 downto 0);
          salida  : out std_logic_vector (N-1 downto 0));
end incGray ;
```