**RESUMEN DESAFIOS FINAL PGE**

**Desafío 1 - Clase Listado:**

* proyecto vacio
* listado.h (clase genérica)
* cantidad
* libre
* v (puntero vector genérico)
* constructor con valor de cantidad por defecto (poner al vector como un new vector de tamaño cantidad)
* método get (devuelve un elemento del array)
* método size (devuelve el libre)
* método getCantidad (devuelve la cantidad)
* add (agrega un nuevo elemento al vector):

verificar que libre < cantidad, le asigna la variable nueva al vector en la posición de libre e incrementa libre.

* main.cpp
* creamos objeto Listado. (no olvidarse que van los dos <int>)
* añadimos elementos con la función add.
* for para recorrer los elementos.

**Desafío 2 - Clase MiVector:**

* proyecto vacío
* mivector.h (clase genérica, herede de QVector (privado))
* constructor (Usar el constructor de QVector. No olvidarse los paréntesis al final)
* método get (devuelve un elemento del vector (usar métodos de QVector))
* método add (agrega un elemento al vector (usar métodos de QVector))
* método size (invocamos al método size de QVector. No olvidarse el <T> antes de ::)
* main.cpp
* crear objeto MiVector.
* Añadir elementos con función add.
* for para recorrer los elementos.

**Desafío 3 - Clase ListadoEnteros:**

* proyecto vacío
* listadoenteros.h (clase normal que herede de QVector<int>)
* constructor (Usar constructor de QVector<int>)
* main.cpp
* Creamos objeto.
* usamos método QVector para agregar elementos.
* recorrer elementos con un for.

**Desafío 4 - Sobrecarga de operadores en MiVector:**

* Descargamos código del desafío 2.
* mivector.h
* sobrecargamos el operador + (recibe un nuevo dato y lo agrega al vector usando la función add)
* sobrecargamos el operador \* (agregar n veces el último elemento al vector)
  + usar un while y que llame n veces a la función add insertando el último elemento (this->last())
* main.cpp
* borramos todo menos el for
* creamos un objeto sin puntero (no olvidarse del <int>)
* llamamos a los operadores
* corregimos el for de que en vez de flecha son puntos

**Desafío 5 - Clase LineaDeTexto:**

* creamos un QT Widgets Application
* Class Ventana, herede de QWidget
* creamos lineadetexto.h / .cpp (hereda de QLineEdit, Q\_OBJECT)
* constructor (QWidget \* parent = nullptr)
* constructor copia (igual que el constructor normal pero este recibe un const LineaDeTexto & lineaDeTexto)
  + queremos copiar el texto que tiene el QLineEdit
* operador de asignación
  + mismo que constructor copia, pero retorna puntero a this
* ventana.ui
* hacemos un login pero sin los labels
* promocionar los lineEdit a la clase LineaDeTexto
* main.cpp
* crear un objeto LineaDeTexto (sin punteros)
* crear un QVector de LineaDeTexto (sin punteros)
* hacerle un push\_bach del objeto l1

**Desafío 6 - LineaDeTexto con restricciones:**

* Descargamos el código del desafío 5 y lo abrimos en el qt
* lineadetexto.h
* definimos una enum (SIN\_RESTRICCION, SOLO\_NUMEROS)
* (publico) void setTipo(Tipo tipo){

this->tipo = tipo

* (privado) Definimos variable de tipo enumeración llamada tipo
* (protected) definimos el eventFilter (bool eventFilter( QObject\*, QEvent \* e);) -> hacerle un refactor
* lineadetexto.cpp
* eventFilter

if( e->type() == QEvent::KeyPress && tipo == SOLO\_NUMEROS )

QKeyEvent \* keyEvent = static\_cast < QKeyEvent \* >( e )

switch( keyEvent->key() )

case Qt::Key\_0: //repetir con las 10 teclas

return false;

default:

return true;

return false

* constructor
  + definir el tipo (SIN\_RESTRICCION)
  + this->installEventFilter( this )
* ventana.cpp
* en el constructor llamar a la funcion setTipo de cada uno de los lineEdit para definirle el tipo de texto.

**Desafío 7 - Login como singleton:**

* creamos un QT Widgets Application
* Class Login, herede de QWidget
* login.h
* private:
  + poner como privado el constructor.
  + puntero **estático** de Login llamado instancia.
* public:
  + definir método **static** Login \* getInstancia() (refactor)
* login.cpp
* definir como global la instancia como puntero nulo (Clase \* Clase::variable = nullptr)
* getInstancia():

preguntar si instancia es un nullptr

instancia = new Login;

retornar instancia

* login.ui
* crear la interfaz de un login (solo los lineEdit y el pb)
* poner como password al lineEdit
* main.cpp
* Borrar lo que tira error
* Login::getInstancia()->show();

**Desafío 8 - Personas comparables:**

* proyecto vacío
* persona.h (clase Persona)
* en privado tiene:
  + edad
  + nombre
* constructor explicito
  + recibe nombre y edad, poner valores por defecto
* hacemos los getters and setters de edad y nombre (usar el refactor de QT)
* definimos el operador <, que reciba otra persona:

compara si la edad de this es menor a la edad de la otra persona

devolver true

devolver false

* definimos el operador >, es lo mismo que el <
* main.cpp
* creamos dos objetos Persona
* usar los operadores con un if para que muestre con el QDebug un texto

**Desafío 9 - Punteros a funciones:**

* creamos un QT Widgets Application
* Clase Login, herede de QWidget
* login.ui
* crear la interfaz de un login (solo los lineEdit y el pb)
* poner como password al lineEdit.
* poner orden de los tabs.
* login.h
* definir un QWidget \* ventana (como privado) -> definirlo en el constructor
* definir slot\_validar(){

valide el usuario admin:123

mostrar ventana

ocultar la del login

si no

cierra la app

* login.cpp
* constructor:
  + definir dos punteros a funciones (uno para la signal y otro para el slot)
    - void ( QPushButton::\*puntero )();
  + asignarle a que funcion van a apuntar
    - puntero = &QPushButton::pressed
  + connect del boton con el slot
    - en la signal y slot usar los punteros a funciones

**Desafío 10 – Excepciones:**

* Descargamos el código del desafío 1.
* excepciones.h (creamos clase ExcFueraRango)
* atributos privados:
  + mensaje (QString)
  + index (int)
* metodos publicos:
  + constructor (recibe el mensaje y el index)
* hacemos los getters and setters de mensaje e index (usar el refactor de QT)
* listado.h
* en la funcion get (mandar una excepcion si i >= libre)

sí i es mayor o igual que libre

- crear un objeto de ExcFueraRango

valores del contructor: poner mensaje y el index es el valor de i

- lanzamos la excepcion con throw (throw excepcion)

* main.cpp

try {

tratamos de acceder a un valor en la función get que no exista

}

Catch ( ExcFueraRango e ){

mostramos el mensaje con e.getMensaje()

}