РУ „Ангел Кънчев“

КУРСОВА РАБОТА

ПО

ООП

Име: Алекс Георгиев Иванов

Група: 25р

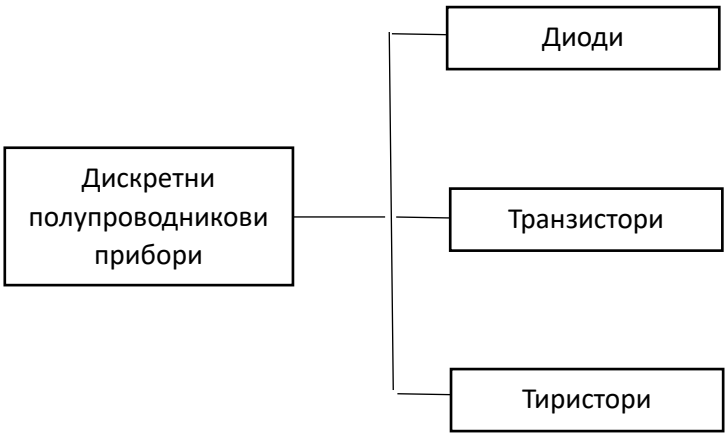
Курс: Първи

Факултетен номер: 233107

Дата:..... Проверил:.....

1. Задание №6.

Дадена е следната класификация:



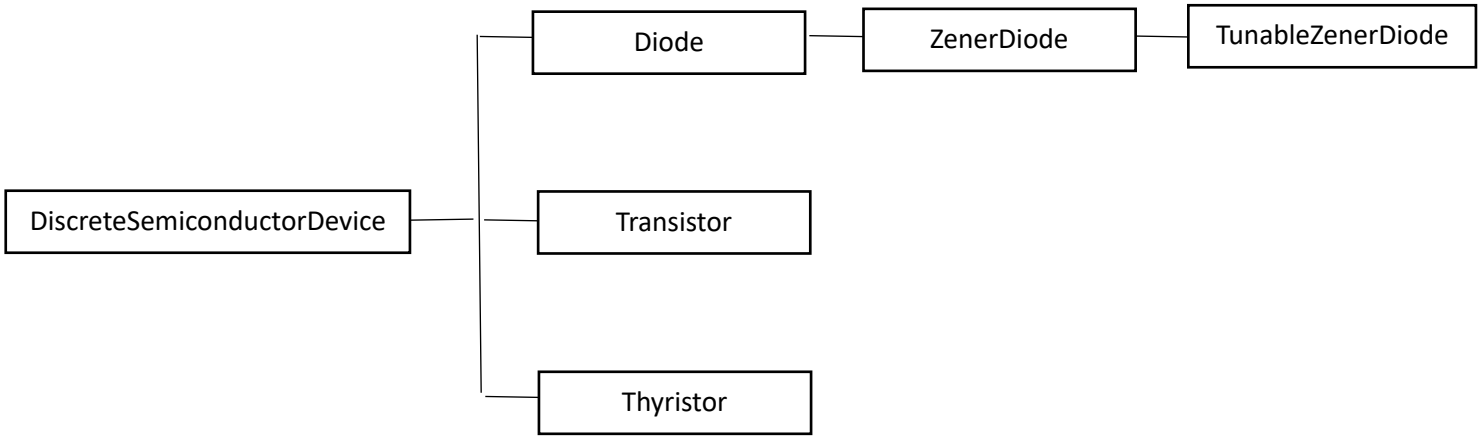
Класификацията да се продължи поне на още две нива. Да се състави йерархия от класове, отразяваща създадената класификация. Да се декларират съответните класове, като всеки клас, с изключение на базовия, да имат поне по 2 собствени атрибута.

Да се дефинира виртуална функция, която извежда характеристиките на обект от всеки клас на йерархията. Във функцията main да се изгради двусвързан списък от обекти от произволни класове в йерархията. Да се разработят функции, които обхождат двусвързания (в двете посоки) списък и извеждат информация за признаците на включените в него обекти.

Декларациите на всеки клас от йерархията да бъдат оформени в отделни заглавни (.h) файлове.

Дефинициите на всеки клас и функцията “main” да бъдат оформени в отделни модули (.cpp файлове). Във всички файлове, съдържащи дефинициите на класовете и функцията “main”, чрез директивата #include да се включат съответните заглавни файлове, съдържащи декларациите на класовете. Да се създаде проект, състоящ се от създадените модули.

2. Схема на наследяване на класовете.



3. Пълно описание на декларираните класове.

ИМЕ НА КЛАСА: DiscreteSemiconductorDevice

БАЗОВ ЗА: Diode, ZenerDiode, TunableZenerDiode, Transistor, Thyristor

ЧЛЕНОВЕ ДАННИ:
име: brand; тип: string
име: model; тип: string

ФУНКЦИИ-ЧЛЕНОВЕ (МЕТОДИ):
DiscreteSemiconductorDevice(const string &brand, const string &model) – инициализиращ конструктор
virtual string ToString() const; – не приема параметри и връща текс с информация за обекта:

Class DiscreteSemiconductorDevice:

- Brand: {brand}

- Model: {model}

ИМЕ НА КЛАСА: Diode

НАСЛЕДЯВА ОТ: DiscreteSemiconductorDevice

БАЗОВ ЗА: ZenerDiode

ЧЛЕНОВЕ ДАННИ:
име: brand; тип: string
име: model; тип: string
име: forwardVoltage; тип: double
име: maxCurrent; тип: double

ФУНКЦИИ-ЧЛЕНОВЕ (МЕТОДИ):
Diode(const string &brand, const string &model, double forwardVoltage, double maxCurrent);– инициализиращ конструктор
virtual string ToString() const override;– не приема параметри и връща текс с информация за обекта:

Class Diode, based on:

Class DiscreteSemiconductorDevice:

- Brand: {brand}

- Model: {model}

- ForwardVoltage: { forwardVoltage }

- MaxCurrent: { maxCurrent }

ИМЕ НА КЛАСА: ZenerDiode

НАСЛЕДЯВА ОТ: Diode

БАЗОВ ЗА: TunableZenerDiode

ЧЛЕНОВЕ ДАННИ:
име: brand; тип: string
име: model; тип: string
име: forwardVoltage; тип: double
име: maxCurrent; тип: double
име: reverseBreakdownVoltage; тип: double
име: temperatureCoefficient; тип: double

ФУНКЦИИ-ЧЛЕНОВЕ (МЕТОДИ):
ZenerDiode(const string &brand, const string &model, double forwardVoltage, double maxCurrent, double reverseBreakdownVoltage, double temperatureCoefficient);– инициализиращ конструктор
virtual string ToString() const override;– не приема параметри и връща текс с информация за обекта:

Class ZenerDiode, based on:

Class Diode:

- Brand: {brand}

- Model: {model}

- ForwardVoltage: { forwardVoltage }

- MaxCurrent: { maxCurrent }

- ReverseBreakdownVoltage: { reverseBreakdownVoltage }

- TemperatureCoefficient: { temperatureCoefficient }

ИМЕ НА КЛАСА: TunableZenerDiode

НАСЛЕДЯВА ОТ: ZenerDiode

ЧЛЕНОВЕ ДАННИ:
име: brand; тип: string
име: model; тип: string
име: forwardVoltage; тип: double
име: maxCurrent; тип: double
име: reverseBreakdownVoltage; тип: double
име: temperatureCoefficient; тип: double
име: controlVoltage; тип: double
име: adjustmentRange; тип: double

ФУНКЦИИ-ЧЛЕНОВЕ (МЕТОДИ):
TunableZenerDiode(const string &brand, const string &model, double forwardVoltage, double maxCurrent, double reverseBreakdownVoltage, double temperatureCoefficient, double controlVoltage, double adjustmentRange); – инициализиращ конструктор
virtual string ToString() const override;– не приема параметри и връща текс с информация за обекта:

Class TunableZenerDiode, based on:

Class ZenerDiode:

- Brand: {brand}

- Model: {model}

- ForwardVoltage: { forwardVoltage }

- MaxCurrent: { maxCurrent }

- ReverseBreakdownVoltage: { reverseBreakdownVoltage }

- TemperatureCoefficient: { temperatureCoefficient }

- ControlVoltage: { controlVoltage}

- AdjustmentRange: { adjustmentRange}

ИМЕ НА КЛАСА: Transistor

НАСЛЕДЯВА ОТ: DiscreteSemiconductorDevice

ЧЛЕНОВЕ ДАННИ:
име: brand; тип: string
име: model; тип: string
име: baseEmitterVoltage; тип: double
име: collectorCurrent; тип: double

ФУНКЦИИ-ЧЛЕНОВЕ (МЕТОДИ):
Transistor(const string &brand, const string &model, double baseEmitterVoltage, double collectorCurrent); – инициализиращ конструктор
virtual string ToString() const ovveride; – не приема параметри и връща текс с информация за обекта:

Class Transistor based on DiscreteSemiconductorDevice:

- Brand: {brand}

- Model: {model}

- BaseEmitterVoltage: { baseEmitterVoltage }

- CollectorCurrent: { collectorCurrent }

ИМЕ НА КЛАСА: Thyristor
НАСЛЕДЯВА ОТ: DiscreteSemiconductorDevice
ЧЛЕНОВЕ ДАННИ:
име: brand; тип: string
име: model; тип: string
име: gateTriggerVoltage; тип: double
име: holdingCurrent; тип: double

ФУНКЦИИ-ЧЛЕНОВЕ (МЕТОДИ):
Thyristor (const string &brand, const string &model, double gateTriggerVoltage, double holdingCurrent); – инициализиращ конструктор
virtual string ToString() const ovveride; – не приема параметри и връща текс с информация за обекта:

Class Thyristor based on DiscreteSemiconductorDevice:

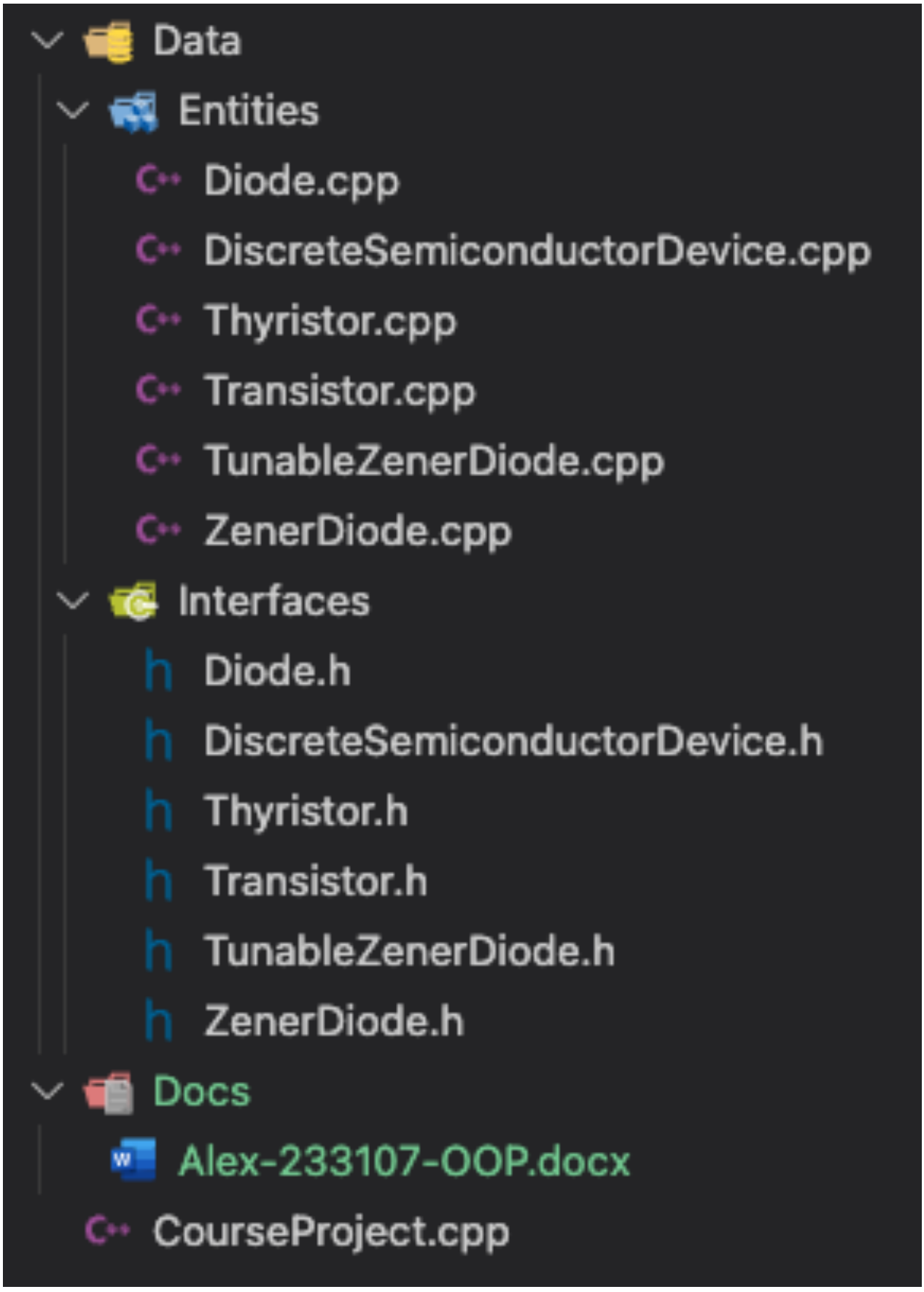
- Brand: {brand}

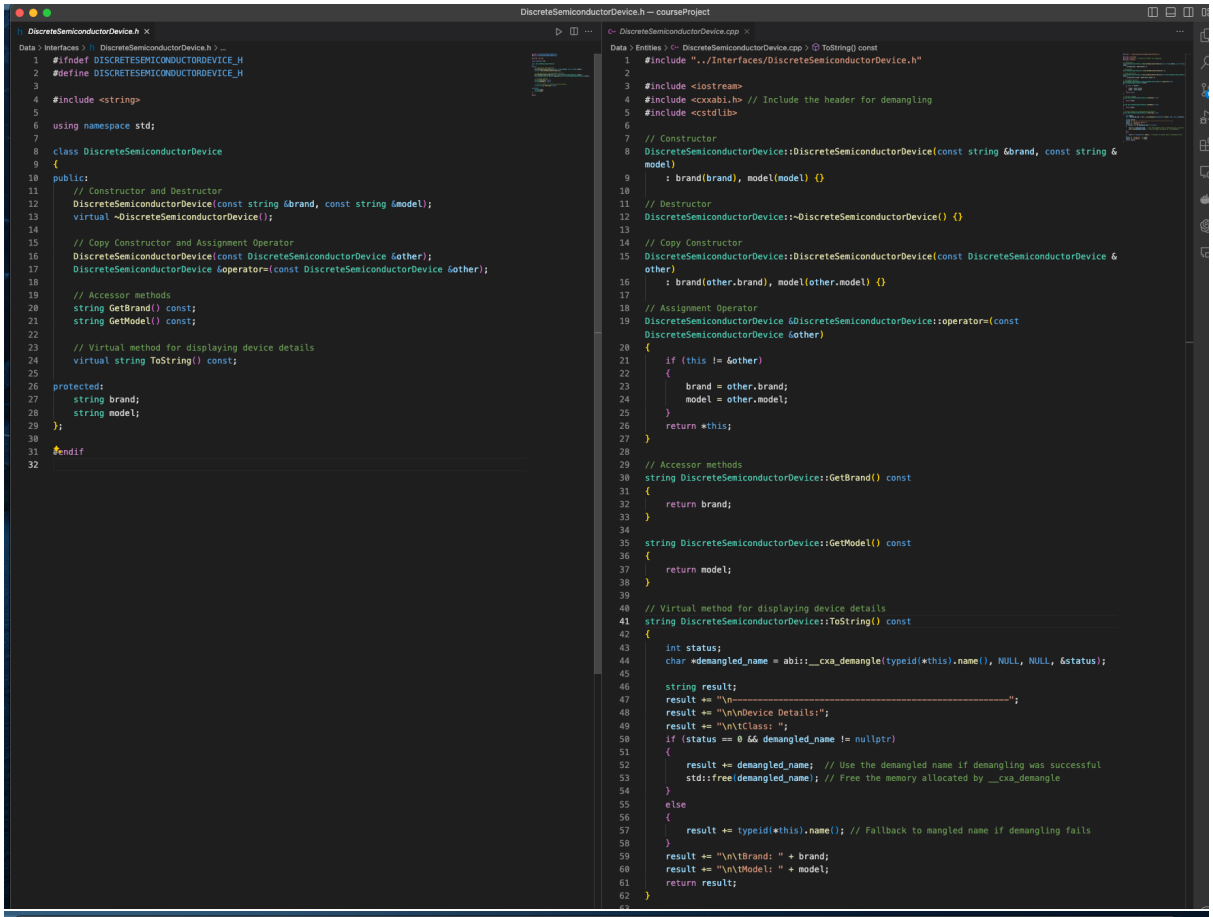
- Model: {model}

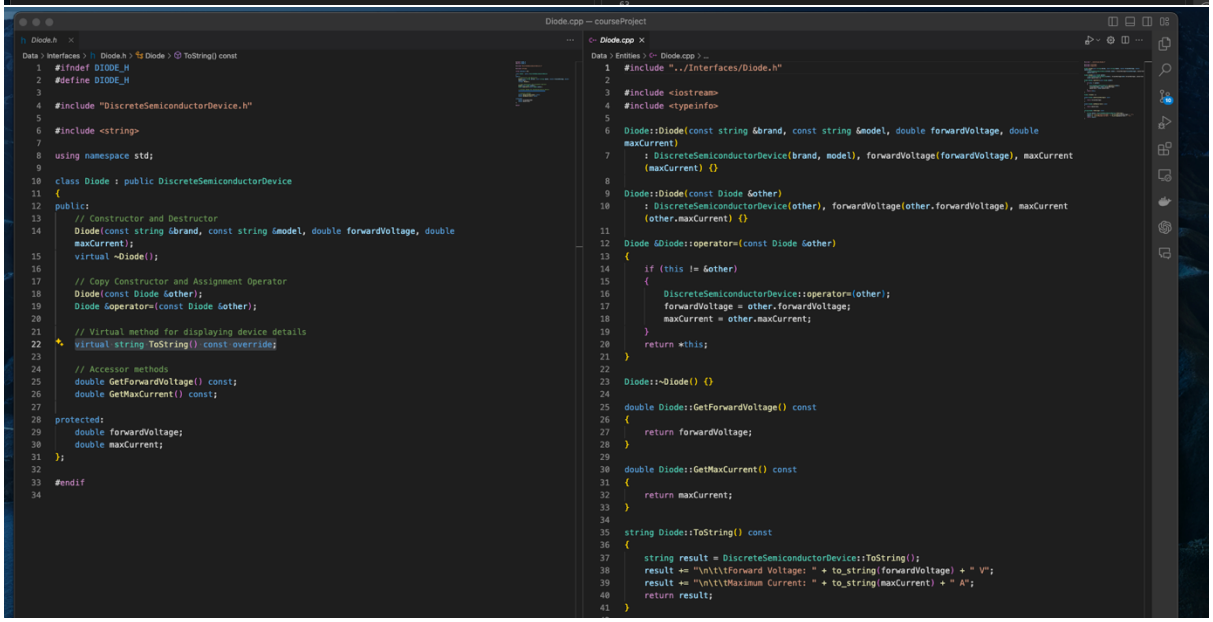
- GateTriggerVoltage: { gateTriggerVoltage }

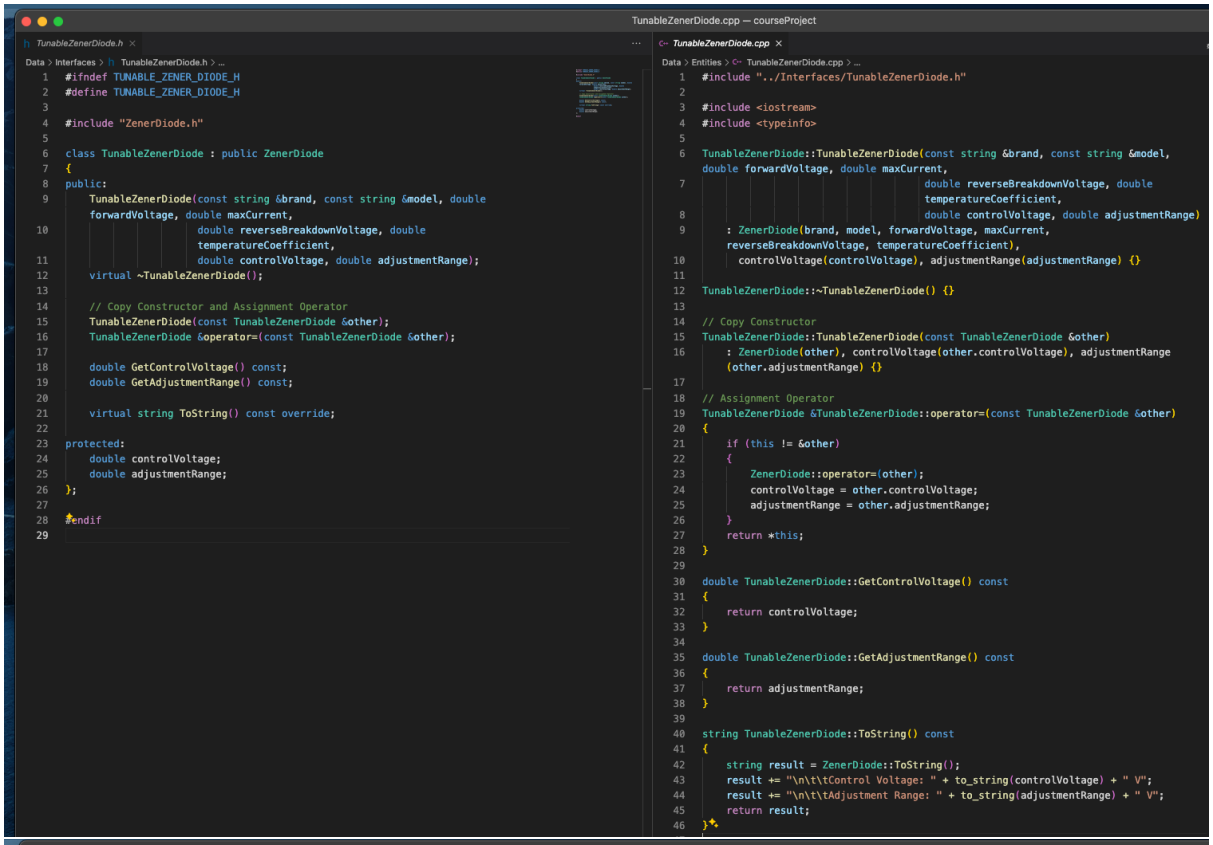
- HoldingCurrent: { holdingCurrent }

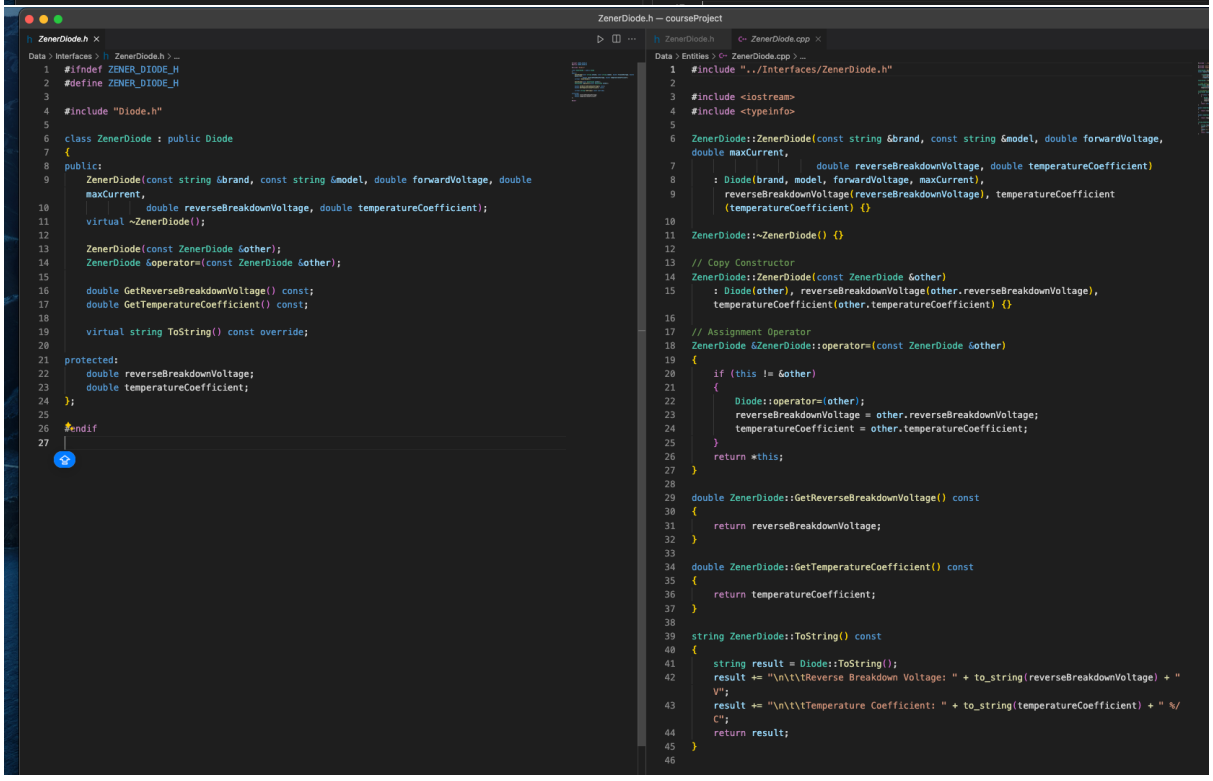
4. Код.

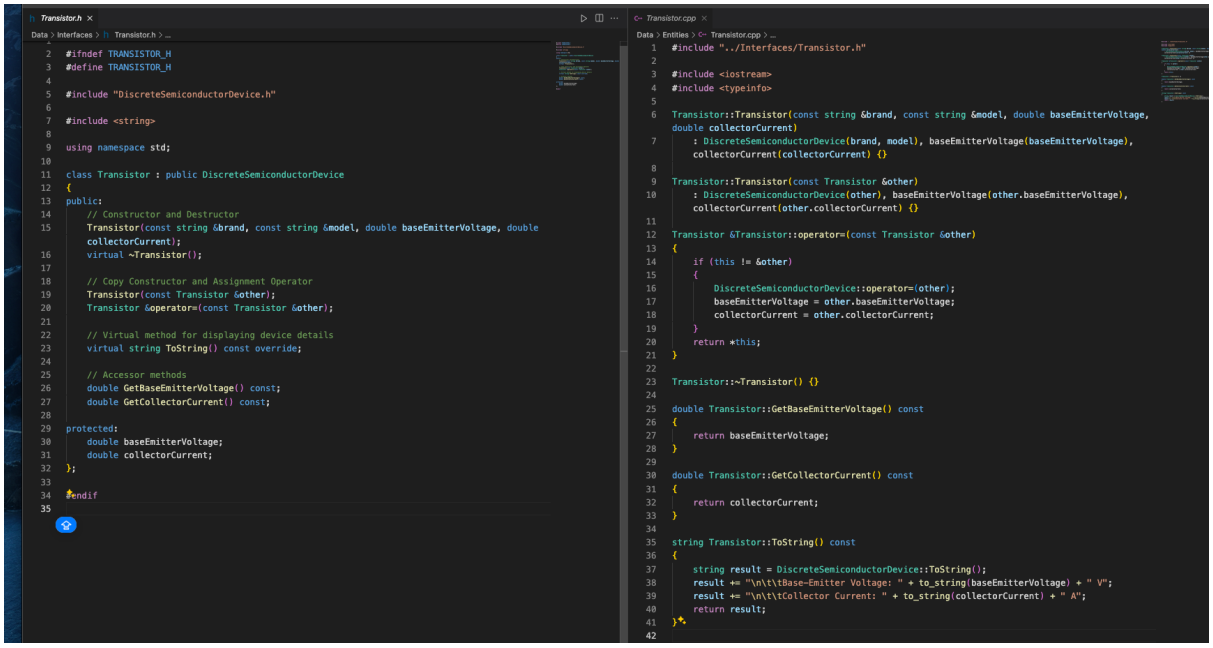

Data
Entities
Diode.cpp
DiscreteSemiconductorDevice.cpp
Thyristor.cpp
Transistor.cpp
TunableZenerDiode.cpp
ZenerDiode.cpp
Interfaces
Diode.h
DiscreteSemiconductorDevice.h
Thyristor.h
Transistor.h
TunableZenerDiode.h
ZenerDiode.h
Docs
Alex-233107-OOP.docx
CourseProject.cpp


DiscreteSemiconductorDevice.h — courseProject
DiscreteSemiconductorDevice.h
DiscreteSemiconductorDevice.cpp
Data > Interfaces > DiscreteSemiconductorDevice.h > ...
Data > Entities > DiscreteSemiconductorDevice.cpp > ToString() const
#ifndef DISCRETESEMICONDUCTORDEVICE_H
#define DISCRETESEMICONDUCTORDEVICE_H
#include <string>
using namespace std;
class DiscreteSemiconductorDevice
{
public:
// Constructor and Destructor
DiscreteSemiconductorDevice(const string &brand, const string &model);
virtual ~DiscreteSemiconductorDevice();
// Copy Constructor and Assignment Operator
DiscreteSemiconductorDevice(const DiscreteSemiconductorDevice &other);
DiscreteSemiconductorDevice &operator=(const DiscreteSemiconductorDevice &other);
// Accessor methods
string GetBrand() const;
string GetModel() const;
// Virtual method for displaying device details
virtual string ToString() const;
protected:
string brand;
string model;
};
#endif
#include "../Interfaces/DiscreteSemiconductorDevice.h"
#include <iostream>
#include <cxxabi.h> // Include the header for demangling
#include <cstdlib>
// Constructor
DiscreteSemiconductorDevice::DiscreteSemiconductorDevice(const string &brand, const string &model)
: brand(brand), model(model) {}
// Destructor
DiscreteSemiconductorDevice::~DiscreteSemiconductorDevice() {}
// Copy Constructor
DiscreteSemiconductorDevice::DiscreteSemiconductorDevice(const DiscreteSemiconductorDevice &other)
: brand(other.brand), model(other.model) {}
// Assignment Operator
DiscreteSemiconductorDevice &DiscreteSemiconductorDevice::operator=(const DiscreteSemiconductorDevice &other)
{
if (this != &other)
{
brand = other.brand;
model = other.model;
}
return *this;
}
// Accessor methods
string DiscreteSemiconductorDevice::GetBrand() const
{
return brand;
}
string DiscreteSemiconductorDevice::GetModel() const
{
return model;
}
// Virtual method for displaying device details
string DiscreteSemiconductorDevice::ToString() const
{
int status;
char *demangled_name = abi::__cxa_demangle(typeid(*this).name(), NULL, NULL, &status);
string result;
result += "\n----------";
result += "\n\nDevice Details:";
result += "\n\tClass: ";
if (status == 0 && demangled_name != nullptr)
{
result += demangled_name; // Use the demangled name if demangling was successful
std::free(demangled_name); // Free the memory allocated by __cxa_demangle
}
else
{
result += typeid(*this).name(); // Fallback to mangled name if demangling fails
}
result += "\n\tBrand: " + brand;
result += "\n\tModel: " + model;
return result;
}


Diode.cpp — courseProject
Diode.h
Diode.cpp
Data > Interfaces > Diode.h > Diode > ToString() const
Data > Entities > Diode.cpp > ...
#ifndef DIODE_H
#define DIODE_H
#include "DiscreteSemiconductorDevice.h"
#include <string>
using namespace std;
class Diode : public DiscreteSemiconductorDevice
{
public:
// Constructor and Destructor
Diode(const string &brand, const string &model, double forwardVoltage, double maxCurrent);
virtual ~Diode();
// Copy Constructor and Assignment Operator
Diode(const Diode &other);
Diode &operator=(const Diode &other);
// Virtual method for displaying device details
virtual string ToString() const override;
// Accessor methods
double GetForwardVoltage() const;
double GetMaxCurrent() const;
protected:
double forwardVoltage;
double maxCurrent;
};
#endif
#include "../Interfaces/Diode.h"
#include <iostream>
#include <typeinfo>
Diode::Diode(const string &brand, const string &model, double forwardVoltage, double maxCurrent)
: DiscreteSemiconductorDevice(brand, model), forwardVoltage(forwardVoltage), maxCurrent(maxCurrent) {}
Diode::Diode(const Diode &other)
: DiscreteSemiconductorDevice(other), forwardVoltage(other.forwardVoltage), maxCurrent(other.maxCurrent) {}
Diode &Diode::operator=(const Diode &other)
{
if (this != &other)
{
DiscreteSemiconductorDevice::operator=(other);
forwardVoltage = other.forwardVoltage;
maxCurrent = other.maxCurrent;
}
return *this;
}
Diode::~Diode() {}
double Diode::GetForwardVoltage() const
{
return forwardVoltage;
}
double Diode::GetMaxCurrent() const
{
return maxCurrent;
}
string Diode::ToString() const
{
string result = DiscreteSemiconductorDevice::ToString();
result += "\n\t\tForward Voltage: " + to_string(forwardVoltage) + " V";
result += "\n\t\tMaximum Current: " + to_string(maxCurrent) + " A";
return result;
}


TunableZenerDiode.cpp — courseProject
TunableZenerDiode.h
TunableZenerDiode.cpp
Data > Interfaces > TunableZenerDiode.h > ...
Data > Entities > TunableZenerDiode.cpp > ...
#ifndef TUNABLE_ZENER_DIODE_H
#define TUNABLE_ZENER_DIODE_H
#include "ZenerDiode.h"
class TunableZenerDiode : public ZenerDiode
{
public:
TunableZenerDiode(const string &brand, const string &model, double forwardVoltage, double maxCurrent, double reverseBreakdownVoltage, double temperatureCoefficient, double controlVoltage, double adjustmentRange);
virtual ~TunableZenerDiode();
// Copy Constructor and Assignment Operator
TunableZenerDiode(const TunableZenerDiode &other);
TunableZenerDiode &operator=(const TunableZenerDiode &other);
double GetControlVoltage() const;
double GetAdjustmentRange() const;
virtual string ToString() const override;
protected:
double controlVoltage;
double adjustmentRange;
};
#endif
#include "../Interfaces/TunableZenerDiode.h"
#include <iostream>
#include <typeinfo>
TunableZenerDiode::TunableZenerDiode(const string &brand, const string &model, double forwardVoltage, double maxCurrent, double reverseBreakdownVoltage, double temperatureCoefficient, double controlVoltage, double adjustmentRange)
: ZenerDiode(brand, model, forwardVoltage, maxCurrent, reverseBreakdownVoltage, temperatureCoefficient), controlVoltage(controlVoltage), adjustmentRange(adjustmentRange) {}
TunableZenerDiode::~TunableZenerDiode() {}
// Copy Constructor
TunableZenerDiode::TunableZenerDiode(const TunableZenerDiode &other)
: ZenerDiode(other), controlVoltage(other.controlVoltage), adjustmentRange(other.adjustmentRange) {}
// Assignment Operator
TunableZenerDiode &TunableZenerDiode::operator=(const TunableZenerDiode &other)
{
if (this != &other)
{
ZenerDiode::operator=(other);
controlVoltage = other.controlVoltage;
adjustmentRange = other.adjustmentRange;
}
return *this;
}
double TunableZenerDiode::GetControlVoltage() const
{
return controlVoltage;
}
double TunableZenerDiode::GetAdjustmentRange() const
{
return adjustmentRange;
}
string TunableZenerDiode::ToString() const
{
string result = ZenerDiode::ToString();
result += "\n\t\tControl Voltage: " + to_string(controlVoltage) + " V";
result += "\n\t\tAdjustment Range: " + to_string(adjustmentRange) + " V";
return result;
}


ZenerDiode.h — courseProject
ZenerDiode.h
ZenerDiode.cpp
Data > Interfaces > ZenerDiode.h > ...
Data > Entities > ZenerDiode.cpp > ...
#ifndef ZENER_DIODE_H
#define ZENER_DIODE_H
#include "Diode.h"
class ZenerDiode : public Diode
{
public:
ZenerDiode(const string &brand, const string &model, double forwardVoltage, double maxCurrent, double reverseBreakdownVoltage, double temperatureCoefficient);
virtual ~ZenerDiode();
ZenerDiode(const ZenerDiode &other);
ZenerDiode &operator=(const ZenerDiode &other);
double GetReverseBreakdownVoltage() const;
double GetTemperatureCoefficient() const;
virtual string ToString() const override;
protected:
double reverseBreakdownVoltage;
double temperatureCoefficient;
};
#endif
#include "../Interfaces/ZenerDiode.h"
#include <iostream>
#include <typeinfo>
ZenerDiode::ZenerDiode(const string &brand, const string &model, double forwardVoltage, double maxCurrent, double reverseBreakdownVoltage, double temperatureCoefficient)
: Diode(brand, model, forwardVoltage, maxCurrent), reverseBreakdownVoltage(reverseBreakdownVoltage), temperatureCoefficient(temperatureCoefficient) {}
ZenerDiode::~ZenerDiode() {}
// Copy Constructor
ZenerDiode::ZenerDiode(const ZenerDiode &other)
: Diode(other), reverseBreakdownVoltage(other.reverseBreakdownVoltage), temperatureCoefficient(other.temperatureCoefficient) {}
// Assignment Operator
ZenerDiode &ZenerDiode::operator=(const ZenerDiode &other)
{
if (this != &other)
{
Diode::operator=(other);
reverseBreakdownVoltage = other.reverseBreakdownVoltage;
temperatureCoefficient = other.temperatureCoefficient;
}
return *this;
}
double ZenerDiode::GetReverseBreakdownVoltage() const
{
return reverseBreakdownVoltage;
}
double ZenerDiode::GetTemperatureCoefficient() const
{
return temperatureCoefficient;
}
string ZenerDiode::ToString() const
{
string result = Diode::ToString();
result += "\n\t\tReverse Breakdown Voltage: " + to_string(reverseBreakdownVoltage) + " V";
result += "\n\t\tTemperature Coefficient: " + to_string(temperatureCoefficient) + " %/C";
return result;
}

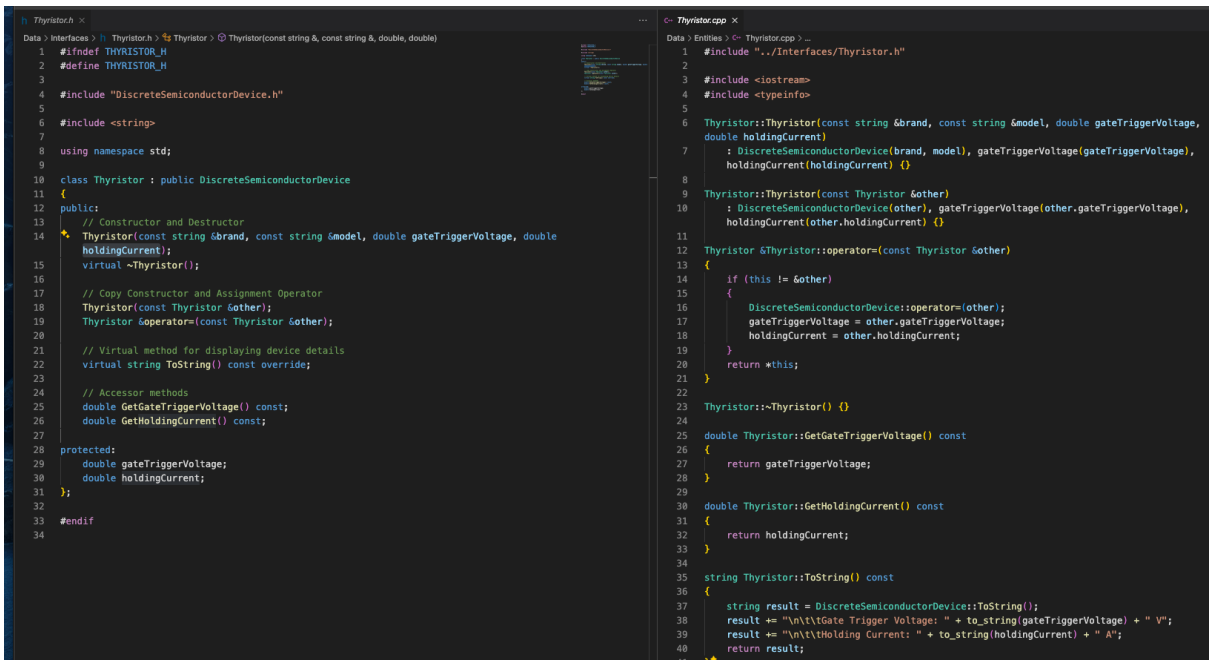


Transistor.h

```
#ifndef TRANSISTOR_H
#define TRANSISTOR_H

#include "DiscreteSemiconductorDevice.h"

#include <string>

using namespace std;

class Transistor : public DiscreteSemiconductorDevice
{
public:
    // Constructor and Destructor
    Transistor(const string &brand, const string &model, double baseEmitterVoltage, double
    collectorCurrent);
    virtual ~Transistor();

    // Copy Constructor and Assignment Operator
    Transistor(const Transistor &other);
    Transistor &operator=(const Transistor &other);

    // Virtual method for displaying device details
    virtual string ToString() const override;

    // Accessor methods
    double GetBaseEmitterVoltage() const;
    double GetCollectorCurrent() const;

protected:
    double baseEmitterVoltage;
    double collectorCurrent;
};

#endif

```

Transistor.cpp

```
#include "../Interfaces/Transistor.h"

#include <iostream>
#include <typeinfo>

Transistor::Transistor(const string &brand, const string &model, double baseEmitterVoltage,
double collectorCurrent)
    : DiscreteSemiconductorDevice(brand, model), baseEmitterVoltage(baseEmitterVoltage),
    collectorCurrent(collectorCurrent) {}

Transistor::Transistor(const Transistor &other)
    : DiscreteSemiconductorDevice(other), baseEmitterVoltage(other.baseEmitterVoltage),
    collectorCurrent(other.collectorCurrent) {}

Transistor &Transistor::operator=(const Transistor &other)
{
    if (this != &other)
    {
        DiscreteSemiconductorDevice::operator=(other);
        baseEmitterVoltage = other.baseEmitterVoltage;
        collectorCurrent = other.collectorCurrent;
    }
    return *this;
}

Transistor::~Transistor() {}

double Transistor::GetBaseEmitterVoltage() const
{
    return baseEmitterVoltage;
}

double Transistor::GetCollectorCurrent() const
{
    return collectorCurrent;
}

string Transistor::ToString() const
{
    string result = DiscreteSemiconductorDevice::ToString();
    result += "\n\t\tBase-Emitter Voltage: " + to_string(baseEmitterVoltage) + " V";
    result += "\n\t\tCollector Current: " + to_string(collectorCurrent) + " A";
    return result;
}

```



Thyristor.h

```
#ifndef THYRISTOR_H
#define THYRISTOR_H

#include "DiscreteSemiconductorDevice.h"

#include <string>

using namespace std;

class Thyristor : public DiscreteSemiconductorDevice
{
public:
    // Constructor and Destructor
    Thyristor(const string &brand, const string &model, double gateTriggerVoltage, double
    holdingCurrent);
    virtual ~Thyristor();

    // Copy Constructor and Assignment Operator
    Thyristor(const Thyristor &other);
    Thyristor &operator=(const Thyristor &other);

    // Virtual method for displaying device details
    virtual string ToString() const override;

    // Accessor methods
    double GetGateTriggerVoltage() const;
    double GetHoldingCurrent() const;

protected:
    double gateTriggerVoltage;
    double holdingCurrent;
};

#endif

```

Thyristor.cpp

```
#include "../Interfaces/Thyristor.h"

#include <iostream>
#include <typeinfo>

Thyristor::Thyristor(const string &brand, const string &model, double gateTriggerVoltage,
double holdingCurrent)
    : DiscreteSemiconductorDevice(brand, model), gateTriggerVoltage(gateTriggerVoltage),
    holdingCurrent(holdingCurrent) {}

Thyristor::Thyristor(const Thyristor &other)
    : DiscreteSemiconductorDevice(other), gateTriggerVoltage(other.gateTriggerVoltage),
    holdingCurrent(other.holdingCurrent) {}

Thyristor &Thyristor::operator=(const Thyristor &other)
{
    if (this != &other)
    {
        DiscreteSemiconductorDevice::operator=(other);
        gateTriggerVoltage = other.gateTriggerVoltage;
        holdingCurrent = other.holdingCurrent;
    }
    return *this;
}

Thyristor::~Thyristor() {}

double Thyristor::GetGateTriggerVoltage() const
{
    return gateTriggerVoltage;
}

double Thyristor::GetHoldingCurrent() const
{
    return holdingCurrent;
}

string Thyristor::ToString() const
{
    string result = DiscreteSemiconductorDevice::ToString();
    result += "\n\t\tGate Trigger Voltage: " + to_string(gateTriggerVoltage) + " V";
    result += "\n\t\tHolding Current: " + to_string(holdingCurrent) + " A";
    return result;
}

```

CourseProject.cpp

```cpp
#include <iostream>
#include <typeinfo>

#include "./Data/Interfaces/DiscreteSemiconductorDevice.h"
#include "./Data/Entities/DiscreteSemiconductorDevice.cpp"
#include "./Data/Interfaces/Diode.h"
#include "./Data/Entities/Diode.cpp"
#include "./Data/Interfaces/Transistor.h"
#include "./Data/Entities/Transistor.cpp"
#include "./Data/Interfaces/Thyristor.h"
#include "./Data/Entities/Thyristor.cpp"
#include "./Data/Interfaces/ZenerDiode.h"
#include "./Data/Entities/ZenerDiode.cpp"
#include "./Data/Interfaces/TunableZenerDiode.h"
#include "./Data/Entities/TunableZenerDiode.cpp"

using namespace std;

int main()
{
    DiscreteSemiconductorDevice *devices[6];

    devices[0] = new DiscreteSemiconductorDevice("NXP", "BC547");
    devices[1] = new Diode("NXP", "BC557", 0.7, 0.1);
    devices[2] = new ZenerDiode("Philips", "BZX55C5V1", 0.7, 0.05, 5.1, 0.03);
    devices[3] = new TunableZenerDiode("Sony", "TZD100", 0.6, 0.04, 10.0, 0.02, 5, 1.0);
    devices[4] = new Transistor("Motorola", "2N3904", 0.6, 0.2);
    devices[5] = new Thyristor("STMicroelectronics", "T1234", 1.0, 1.5);

    // Demonstrating copy constructor
    Transistor transistorCopy = *(static_cast<Transistor *>(devices[2]));
    Thyristor thyristorCopy = *(static_cast<Thyristor *>(devices[3]));

    // Showing all device info
    for (int i = 0; i < 6; i++)
    {
        cout << devices[i]->ToString() << endl;
        if (auto di = dynamic_cast<DiscreteSemiconductorDevice *>(devices[i]))
        {
            cout << "\nAccessor Methods: " << endl;
            cout << "Brand: " << di->GetBrand() << endl;
            cout << "Model: " << di->GetModel() << endl;
        }
        else if (auto di = dynamic_cast<Diode *>(devices[i]))
        {
            cout << "\nAccessor Methods: " << endl;
            cout << "Forward Voltage: " << di->GetForwardVoltage() << " V" << endl;
            cout << "Max Current: " << di->GetMaxCurrent() << " A" << endl;
            if (auto zd = dynamic_cast<ZenerDiode *>(devices[i]))
            {
                cout << "Reverse Breakdown Voltage: " << zd->GetReverseBreakdownVoltage() << " V" << endl;
                cout << "Temperature Coefficient: " << zd->GetTemperatureCoefficient() << " %/C" << endl;
                if (auto tzd = dynamic_cast<TunableZenerDiode *>(devices[i]))
                {
                    cout << "Control Voltage: " << tzd->GetControlVoltage() << " V" << endl;
                    cout << "Adjustment Range: " << tzd->GetAdjustmentRange() << " V" << endl;
                }
            }
        }
        else if (auto tr = dynamic_cast<Transistor *>(devices[i]))
        {
            cout << "\nAccessor Methods: " << endl;
            cout << "Base-Emitter Voltage: " << tr->GetBaseEmitterVoltage() << " V" << endl;
            cout << "Collector Current: " << tr->GetCollectorCurrent() << " A" << endl;
        }
        else if (auto th = dynamic_cast<Thyristor *>(devices[i]))
        {
            cout << "\nAccessor Methods: " << endl;
            cout << "Gate Trigger Voltage: " << th->GetGateTriggerVoltage() << " V" << endl;
            cout << "Holding Current: " << th->GetHoldingCurrent() << " A" << endl;
        }
    }

    // Demonstrating the copy's independence
    cout << "\n\nCopy constructor examples:" << endl;
    cout << "\nOriginal Transistor:\n"
         << devices[4]->ToString() << endl;
    cout << "\nCopy of Transistor:\n"
         << transistorCopy.ToString() << endl;

    for (int i = 0; i < 6; i++)
    {
        if (devices[i])
        {
            delete devices[i];
        }
    }
    return 0;
}
```

Всичко може да бъде видяно на тук:
GitHub