**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра ВТ**

Курсовая РАБОТА

**по дисциплине «Объектно-ориентированное программирование»**

Тема: Разработка ПК на объектно-ориентированном языке программирования.

|  |  |  |
| --- | --- | --- |
| Студент гр. 1308 |  | Лепов А. В. |
| Преподаватель |  | Гречухин М. Н. |

**ЗАДАНИЕ**

**на курсовую работу**

|  |
| --- |
| Студент Лепов А. В. |
| Группа 1308 |
| Тема работы: Разработка ПК на объектно-ориентированном языке программирования. |
| Исходные данные:  Разработать ПК для администратора Интернет-магазина компьютерной техники (компонентов системного блока). В ПК должны храниться сведения о центральных процессорах, материнских платах и видеоадаптерах. Администратор Интернет-магазина может добавлять, изменять и удалять эти сведения. Ему может потребоваться следующая информация:   * название чипсета и сокета у материнской платы; * название сокета у центрального процессора; * наименование производителей видеоадаптеров и материнских плат; * основные характеристики и стоимость компонентов системного блока. |

**СОДЕРЖАНИЕ**

[1. ДИАГРАММА КЛАССОВ 4](#_Toc137193673)

[1.1. Класс «ClassBrand» 5](#_Toc137193674)

[1.2. Класс «ClassChipset» 5](#_Toc137193675)

[1.3. Класс «ClassSocket» 6](#_Toc137193676)

[1.4. Класс «ClassCPU» 7](#_Toc137193677)

[1.5. Класс «ClassGPU» 8](#_Toc137193678)

[1.6. Класс «ClassPCB» 9](#_Toc137193679)

[1.7. Исходный код для классов 10](#_Toc137193680)

[1.7.1. ClassBrand.java 10](#_Toc137193681)

[1.7.2. ClassChipset.java 11](#_Toc137193682)

[1.7.3. ClassSocket.java 12](#_Toc137193683)

[1.7.4. ClassCPU.java 13](#_Toc137193684)

[1.7.5. ClassGPU.java 15](#_Toc137193685)

[1.7.6. ClassPCB.java 16](#_Toc137193686)

[2. СХЕМА ДАННЫХ БД 19](#_Toc137193687)

[3. ИНТЕРФЕЙС ПРОГРАММЫ 22](#_Toc137193688)

[4. ГЕНЕРАЦИЯ ОТЧЕТОВ 28](#_Toc137193689)

[5. КРАТКОЕ РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 31](#_Toc137193690)

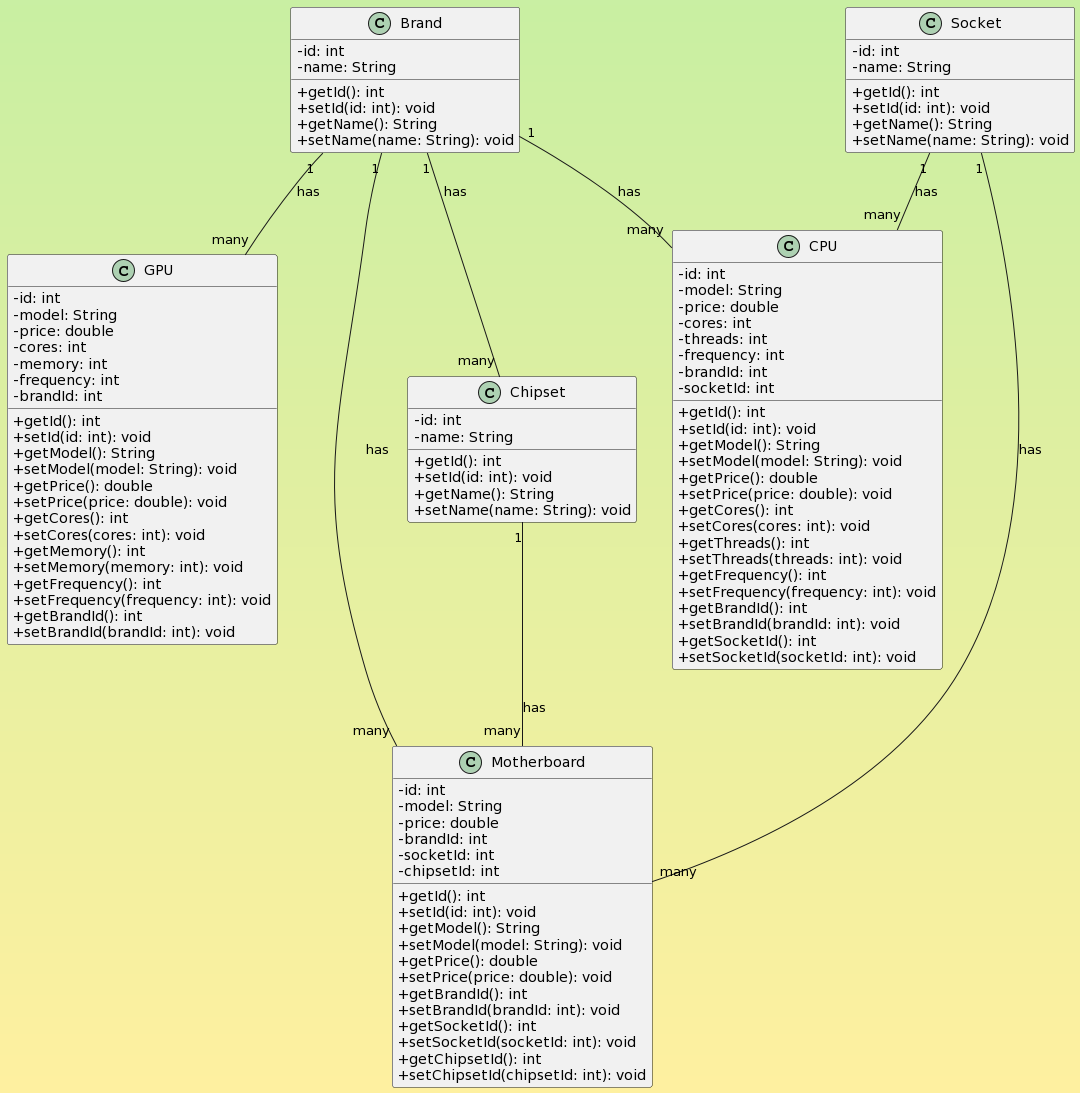
[6. СПИСОК ИСТОЧНИКОВ 35](#_Toc137193691)

[приложение А 36](#_Toc137193692)

[приложение б 38](#_Toc137193693)

1. **ДИАГРАММА КЛАССОВ**

Ниже продемонстрирована UML диаграмма классов для данного курсового проекта (UML код представлен в приложении А).

****

*Рисунок 1. UML-диаграмма классов*

* 1. **Класс «ClassBrand»**

Данный класс ClassBrand представляет таблицу brand в базе данных и имеет следующие поля и методы:

Таблица № 1.   
Свойства класса ClassBrand

|  |  |
| --- | --- |
| **Наименование** | **Описание** |
| **Поля** | |
| id | Хранит уникальный идентификатор (primary key) для каждой записи в таблице "brand". Он имеет тип int. |
| name | Хранит имя бренда. Он имеет тип String. |
| **Геттеры** | |
| getId() | Возвращает значение поля id, представляющее идентификатор бренда. |
| getName() | Возвращает значение поля name, представляющее имя бренда. |
| **Сеттеры** | |
| setId(int id) | Устанавливает значение поля id в заданное целочисленное значение. |
| setName(String name) | Устанавливает значение поля name в заданную строку. |

* 1. **Класс «ClassChipset»**

Этот класс ClassChipset представляет таблицу chipset в базе данных с помощью аннотаций JPA (Java Persistence API). Он содержит следующие поля и методы:

Таблица № 2.   
Свойства класса ClassChipset

|  |  |
| --- | --- |
| **Наименование** | **Описание** |
| **Поля** | |
| id | Целочисленное поле, представляющее идентификатор (primary key) для записи в таблице chipset. |
| name | Строковое поле, представляющее имя чипсета. |
| **Геттеры** | |
| getId() | Возвращает значение поля id, представляющее идентификатор сокета. |
| getName() | Возвращает значение поля name, представляющее имя сокета. |
| **Сеттеры** | |
| setId(int id) | Устанавливает значение поля id в заданное целочисленное значение. |
| setName(String name) | Устанавливает значение поля name в заданную строку. |

* 1. **Класс «ClassSocket»**

Класс ClassSocket представляет сущность сокета и используется для отображения данных в базе данных с помощью Hibernate и JPA:

Таблица № 3.   
Свойства класса ClassChipset

|  |  |
| --- | --- |
| **Наименование** | **Описание** |
| **Поля** | |
| id | Тип int. Хранит уникальный идентификатор сокета. Аннотация @Id указывает, что это поле является первичным ключом, @GeneratedValue указывает на автоматическую генерацию значения для данного поля, а @Column определяет соответствующее имя столбца в базе данных. |
| name | Тип String. Хранит название сокета. |
| **Геттеры** | |
| getId() | Возвращает значение поля id, представляющее идентификатор чипсета. |
| getName() | Возвращает значение поля name, представляющее имя чипсета. |
| **Сеттеры** | |
| setId(int id) | Устанавливает значение поля id в соответствии с переданным аргументом. |
| setName(String name) | Устанавливает значение поля name в соответствии с переданным аргументом. |

* 1. **Класс «ClassCPU»**

Этот класс, названный ClassCPU, представляет процессор (CPU) компьютера. Вот описание каждого поля и методов класса:

Таблица № 4.   
Свойства класса ClassCPU

|  |  |
| --- | --- |
| **Наименование** | **Описание** |
| **Поля** | |
| id | Целочисленное поле, представляющее идентификатор. |
| model | (тип: String): Хранит модель процессора. |
| price | (тип: double): Хранит цену процессора. |
| cores | (тип: int): Хранит количество ядер процессора. |
| threads | (тип: int): Хранит количество потоков процессора. |
| frequency | (тип: int): Хранит тактовую частоту процессора. |
| brand | (тип: ClassBrand): Хранит информацию о бренде процессора (класс ClassBrand содержит дополнительные детали о бренде). |
| socket | (тип: ClassSocket): Хранит информацию о разъеме процессора (класс ClassSocket содержит дополнительные детали о разъеме). |
| **Геттеры** | |
| getId() | Возвращает идентификатор (id) процессора. |
| getModel() | Возвращает модель процессора. |
| getPrice() | Возвращает цену процессора. |
| getCores() | Возвращает количество ядер процессора. |
| getThreads() | Возвращает количество потоков процессора. |
| getFrequency() | Возвращает тактовую частоту процессора. |
| getBrand() | Возвращает объект ClassBrand, который представляет бренд процессора. |
| getSocket() | Возвращает объект ClassSocket, который представляет разъем процессора. |
| **Сеттеры** | |
| setId(int id) | Устанавливает значение поля id в заданное целочисленное значение. |
| setName(String name) | Устанавливает значение поля name в заданную строку. |
| setPrice(double price) | Устанавливает цену процессора. |
| setCores(int cores) | Устанавливает количество ядер процессора. |
| setThreads(int threads) | Устанавливает количество потоков процессора. |
| setFrequency(int frequency) | Устанавливает тактовую частоту процессора. |
| setBrand(ClassBrand brand) | Устанавливает объект ClassBrand, который представляет бренд процессора. |
| setSocket(ClassSocket socket) | Устанавливает объект ClassSocket, который представляет разъем процессора. |

* 1. **Класс «ClassGPU»**

Этот класс ClassGPU представляет графический процессор (GPU) и имеет следующие поля и методы:

Таблица № 5.   
Свойства класса ClassGPU

|  |  |
| --- | --- |
| **Наименование** | **Описание** |
| **Поля** | |
| id | Целочисленное поле, представляющее идентификатор (primary key) для записи в таблице. |
| model | (тип данных: String): Хранит модель графического процессора. |
| price | (тип данных: double): Хранит цену графического процессора. |
| cores | (тип данных: int): Хранит количество ядер (ядерных блоков) графического процессора. |
| memory | (тип данных: int): Хранит объем памяти графического процессора. |
| frequency | (тип данных: int): Хранит частоту работы графического процессора. |
| brand | (тип данных: ClassBrand): Хранит бренд (производителя) графического процессора. |
| **Геттеры** | |
| getId() | Возвращает значение поля id. |
| getModel() | Возвращает значение поля model. |
| getPrice() | Возвращает значение поля price. |
| getCores() | Возвращает значение поля cores. |
| getMemory() | Возвращает значение поля memory. |
| getFrequency() | Возвращает значение поля frequency. |
| getBrand() | Возвращает значение поля brand. |
| **Сеттеры** | |
| setId(int id) | Устанавливает значение поля id на основе переданного аргумента id. |
| setModel(String name) | Устанавливает значение поля model на основе переданного аргумента model. |
| setPrice(double price) | Устанавливает значение поля price на основе переданного аргумента price. |
| setCores(int cores) | Устанавливает значение поля cores на основе переданного аргумента cores. |
| setMemory(int memory) | Устанавливает значение поля memory на основе переданного аргумента memory. |
| setFrequency(int frequency) | Устанавливает значение поля frequency на основе переданного аргумента frequency. |
| setBrand(ClassBrand brand) | Устанавливает значение поля brand на основе переданного аргумента brand. |

* 1. **Класс «ClassPCB»**

Данный класс, названный "ClassPCB", представляет печатную плату (PCB) компьютера. Вот описание каждого поля и метода класса:

Таблица № 6.   
Свойства класса ClassPCB

|  |  |
| --- | --- |
| **Наименование** | **Описание** |
| **Поля** | |
| id | Целочисленное поле, представляющее идентификатор (primary key) для записи в таблице. |
| model | (тип данных: String): Хранит модель печатной платы. |
| price | (тип данных: double): Хранит цену печатной платы. |
| brand | (тип данных: ClassBrand): Хранит информацию о бренде печатной платы. (Предполагается, что "ClassBrand" - это класс, представляющий бренд компонента.) |
| socket | (тип данных: ClassSocket): Хранит информацию о сокете печатной платы. |
| chipset | (тип данных: ClassChipset): Хранит информацию о чипсете печатной платы. |

|  |  |
| --- | --- |
| **Геттеры** | |
| getId() | Возвращает идентификатор платы. |
| getModel() | Возвращает модель платы. |
| getPrice() | Возвращает цену платы. |
| getBrand() | Возвращает объект класса ClassBrand, представляющий бренд платы. |
| getSocket() | Возвращает объект класса ClassSocket, представляющий сокет платы. |
| getChipset() | Возвращает объект класса ClassChipset, представляющий чипсет платы. |
| **Сеттеры** | |
| setId(int id) | Устанавливает значение поля id в заданное целочисленное значение. |
| setModel(String name) | Устанавливает модель платы. |
| setPrice(double price) | Устанавливает цену платы. |
| setBrand(ClassBrand brand) | Устанавливает объект класса ClassBrand, представляющий бренд платы. |
| setSocket(ClassSocket socket) | Устанавливает объект класса ClassSocket, представляющий сокет платы. |
| setChipset(ClassChipset chipset) | Устанавливает объект класса ClassChipset, представляющий чипсет платы. |

* 1. **Исходный код для классов**

Ниже представлен код структуры и методов классов проекта:

Полный код самой программы представлен в приложении Б.

* + 1. **ClassBrand.java**

package al.exe;

import javax.persistence.\*;

@Entity

@Table(name = "brand")

public class ClassBrand

{

    @Id

    @GeneratedValue(strategy = GenerationType.IDENTITY)

    @Column(name = "id")

    private int id;

    @Column(name = "name")

    private String name;

    // Default constructor

    public ClassBrand()

    {

    }

    // Parameterized constructor

    public ClassBrand(String name)

    {

        this.name = name;

    }

    // Getters

    public int getId()

    {

        return id;

    }

    public String getName()

    {

        return name;

    }

    // Setters

    public void setId(int id)

    {

        this.id = id;

    }

    public void setName(String name)

    {

        this.name = name;

    }

}

* + 1. **ClassChipset.java**

package al.exe;

import javax.persistence.\*;

@Entity

@Table(name = "chipset")

public class ClassChipset

{

    @Id

    @GeneratedValue(strategy = GenerationType.IDENTITY)

    @Column(name = "id")

    private int id;

    @Column(name = "name")

    private String name;

    // Default constructor

    public ClassChipset()

    {

    }

    // Parameterized constructor

    public ClassChipset(String name)

    {

        this.name = name;

    }

    // Getters

    public int getId()

    {

        return id;

    }

    public String getName()

    {

        return name;

    }

    // Setters

    public void setId(int id)

    {

        this.id = id;

    }

    public void setName(String name)

    {

        this.name = name;

    }

}

* + 1. **ClassSocket.java**

package al.exe;

import javax.persistence.\*;

@Entity

@Table(name = "socket")

public class ClassSocket

{

    @Id

    @GeneratedValue(strategy = GenerationType.IDENTITY)

    @Column(name = "id")

    private int id;

    @Column(name = "name")

    private String name;

    // Default constructor

    public ClassSocket()

    {

    }

    // Parameterized constructor

    public ClassSocket(String name)

    {

        this.name = name;

    }

    // Getters

    public int getId()

    {

        return id;

    }

    public String getName()

    {

        return name;

    }

    // Setters

    public void setId(int id)

    {

        this.id = id;

    }

    public void setName(String name)

    {

        this.name = name;

    }

}

* + 1. **ClassCPU.java**

package al.exe;

import javax.persistence.\*;

@Entity

@Table(name = "cpu")

public class ClassCPU

{

    @Id

    @GeneratedValue(strategy = GenerationType.IDENTITY)

    @Column(name = "id")

    private int id;

    @Column(name = "model")

    private String model;

    @Column(name = "price")

    private double price;

    @Column(name = "cores")

    private int cores;

    @Column(name = "threads")

    private int threads;

    @Column(name = "frequency")

    private int frequency;

    @ManyToOne

    @JoinColumn(name = "brand\_id")

    private ClassBrand brand;

    @ManyToOne

    @JoinColumn(name = "socket\_id")

    private ClassSocket socket;

    // Default constructor

    public ClassCPU()

    {

    }

    // Parameterized constructor

    public ClassCPU(String model, double price, int cores, int threads, int frequency, ClassBrand brand, ClassSocket socket)

    {

        this.model = model;

        this.price = price;

        this.cores = cores;

        this.threads = threads;

        this.frequency = frequency;

        this.brand = brand;

        this.socket = socket;

    }

    // Getters

    public int getId()

    {

        return id;

    }

    public String getModel()

    {

        return model;

    }

    public double getPrice()

    {

        return price;

    }

    public int getCores()

    {

        return cores;

    }

    public int getThreads()

    {

        return threads;

    }

    public int getFrequency()

    {

        return frequency;

    }

    public ClassBrand getBrand()

    {

        return brand;

    }

    public ClassSocket getSocket()

    {

        return socket;

    }

    // Setters

    public void setId(int id)

    {

        this.id = id;

    }

    public void setModel(String model)

    {

        this.model = model;

    }

    public void setPrice(double price)

    {

        this.price = price;

    }

    public void setCores(int cores)

    {

        this.cores = cores;

    }

    public void setThreads(int threads)

    {

        this.threads = threads;

    }

    public void setFrequency(int frequency)

    {

        this.frequency = frequency;

    }

    public void setBrand(ClassBrand brand)

    {

        this.brand = brand;

    }

    public void setSocket(ClassSocket socket)

    {

        this.socket = socket;

    }

}

* + 1. **ClassGPU.java**

package al.exe;

import javax.persistence.\*;

@Entity

@Table(name = "gpu")

public class ClassGPU

{

    @Id

    @GeneratedValue(strategy = GenerationType.IDENTITY)

    @Column(name = "id")

    private int id;

    @Column(name = "model")

    private String model;

    @Column(name = "price")

    private double price;

    @Column(name = "cores")

    private int cores;

    @Column(name = "memory")

    private int memory;

    @Column(name = "frequency")

    private int frequency;

    @ManyToOne

    @JoinColumn(name = "brand\_id")

    private ClassBrand brand;

    // Default constructor

    public ClassGPU()

    {

    }

    // Parameterized constructor

    public ClassGPU(String model, double price, int cores, int memory, int frequency, ClassBrand brand) {

        this.model = model;

        this.price = price;

        this.cores = cores;

        this.memory = memory;

        this.frequency = frequency;

        this.brand = brand;

    }

    // Getters

    public int getId()

    {

        return id;

    }

    public String getModel()

    {

        return model;

    }

    public double getPrice()

    {

        return price;

    }

    public int getCores()

    {

        return cores;

    }

    public int getMemory()

    {

        return memory;

    }

    public int getFrequency()

    {

        return frequency;

    }

    public ClassBrand getBrand()

    {

        return brand;

    }

    // Setters

    public void setId(int id)

    {

        this.id = id;

    }

    public void setModel(String model)

    {

        this.model = model;

    }

    public void setPrice(double price)

    {

        this.price = price;

    }

    public void setCores(int cores)

    {

        this.cores = cores;

    }

    public void setMemory(int memory)

    {

        this.memory = memory;

    }

    public void setFrequency(int frequency)

    {

        this.frequency = frequency;

    }

    public void setBrand(ClassBrand brand)

    {

        this.brand = brand;

    }

}

* + 1. **ClassPCB.java**

package al.exe;

import javax.persistence.\*;

@Entity

@Table(name = "mbrd")

public class ClassPCB

{

    @Id

    @GeneratedValue(strategy = GenerationType.IDENTITY)

    @Column(name = "id")

    private int id;

    @Column(name = "model")

    private String model;

    @Column(name = "price")

    private double price;

    @ManyToOne

    @JoinColumn(name = "brand\_id")

    private ClassBrand brand;

    @ManyToOne

    @JoinColumn(name = "socket\_id")

    private ClassSocket socket;

    @ManyToOne

    @JoinColumn(name = "chipset\_id")

    private ClassChipset chipset;

    // Default constructor

    public ClassPCB()

    {

    }

    // Parameterized constructor

    public ClassPCB(String model, double price, ClassBrand brand, ClassSocket socket, ClassChipset chipset)

    {

        this.model = model;

        this.price = price;

        this.brand = brand;

        this.socket = socket;

        this.chipset = chipset;

    }

    // Getters

    public int getId()

    {

        return id;

    }

    public String getModel()

    {

        return model;

    }

    public double getPrice()

    {

        return price;

    }

    public ClassBrand getBrand()

    {

        return brand;

    }

    public ClassSocket getSocket()

    {

        return socket;

    }

    public ClassChipset getChipset()

    {

        return chipset;

    }

    // Setters

    public void setModel(String model)

    {

        this.model = model;

    }

    public void setPrice(double price)

    {

        this.price = price;

    }

    public void setBrand(ClassBrand brand)

    {

        this.brand = brand;

    }

    public void setSocket(ClassSocket socket)

    {

        this.socket = socket;

    }

    public void setChipset(ClassChipset chipset)

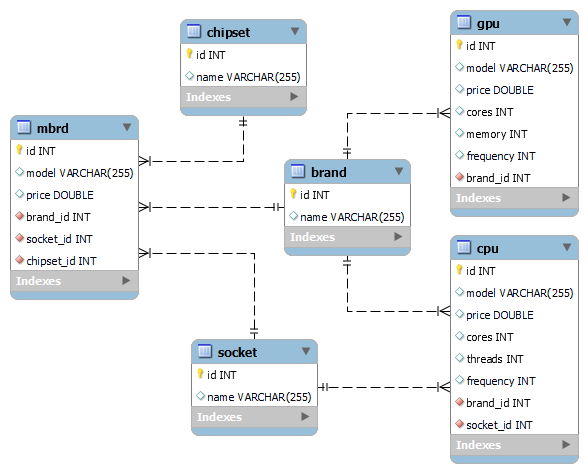
    {

        this.chipset = chipset;

    }

}

**2. СХЕМА ДАННЫХ БД**

****

*Рисунок 2. EER-диаграмма базы данных*

Таблица № 7.   
Свойства таблицы brand

|  |  |  |
| --- | --- | --- |
| Наименование | Тип данных | Описание |
| id | INT | уникальный идентификатор бренда |
| name | VARCHAR(255) | название бренда |

Таблица № 7.   
Свойства таблицы chipset

|  |  |  |
| --- | --- | --- |
| Наименование | Тип данных | Описание |
| id | INT | уникальный идентификатор чипсета |
| name | VARCHAR(255) | название чипсета |

Таблица № 7.   
Свойства таблицы cpu

|  |  |  |
| --- | --- | --- |
| Наименование | Тип данных | Описание |
| id | INT | уникальный идентификатор процессора |
| model | VARCHAR(255) | модель процессора |
| price | DOUBLE | цена процессора |
| cores | INT | количество ядер процессора |
| threads | INT | количество потоков процессора |
| frequency | INT | тактовая частота процессора |
| Brand\_id | INT | идентификатор бренда процессора (ссылка на таблицу brand) |
| socket\_id | INT | идентификатор сокета процессора (ссылка на таблицу socket) |

Таблица № 7.   
Свойства таблицы gpu

|  |  |  |
| --- | --- | --- |
| Наименование | Тип данных | Описание |
| id | INT | уникальный идентификатор графического процессора |
| name | VARCHAR(255) | модель графического процессора |
| price | DOUBLE | цена графического процессора |
| cores | INT | количество ядер графического процессора |
| memory | INT | объем памяти графического процессора |
| frequency | INT | тактовая частота графического процессора |
| brand\_id | INT | идентификатор бренда графического процессора (ссылка на таблицу brand) |

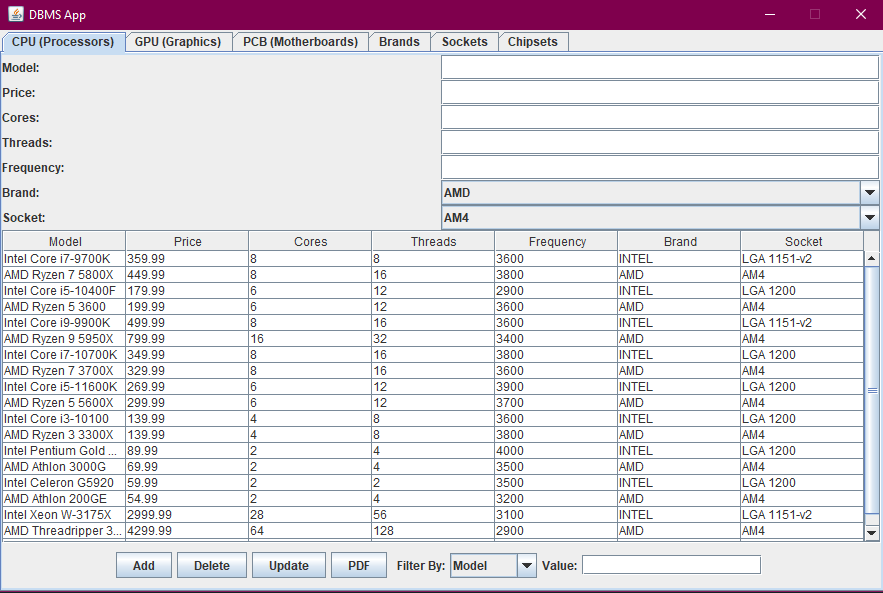
Таблица № 7.   
Свойства таблицы mbrd

|  |  |  |
| --- | --- | --- |
| Наименование | Тип данных | Описание |
| id | INT | уникальный идентификатор материнской платы |
| model | VARCHAR(255) | модель материнской платы |
| price | DOUBLE | цена материнской платы |
| brand\_id | INT | идентификатор бренда материнской платы (ссылка на таблицу brand) |
| socket\_id | INT | идентификатор сокета материнской платы (ссылка на таблицу socket) |
| chipset\_id | INT | идентификатор чипсета материнской платы (ссылка на таблицу chipset) |

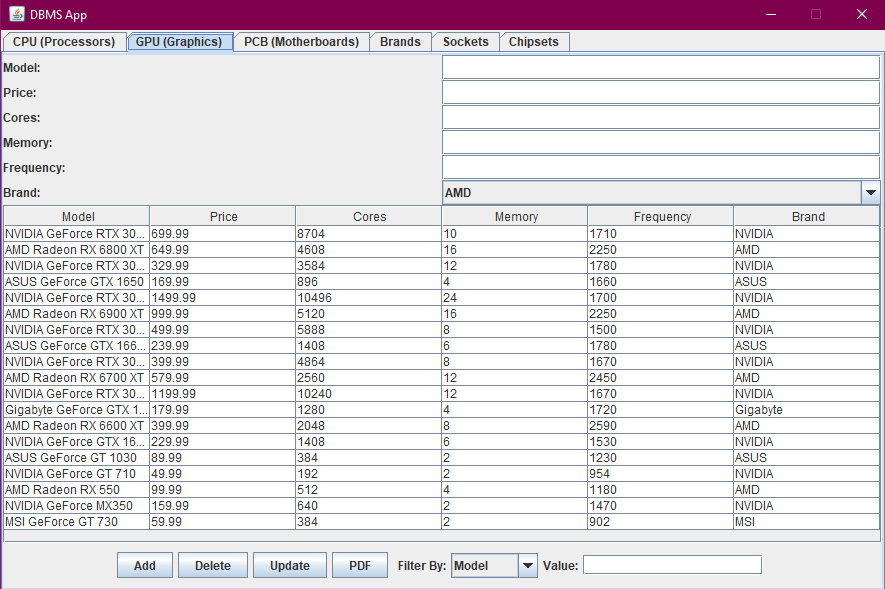
Таблица № 7.   
Свойства таблицы socket

|  |  |  |
| --- | --- | --- |
| Наименование | Тип данных | Описание |
| id | INT | уникальный идентификатор сокета |
| name | VARCHAR(255) | название сокета |

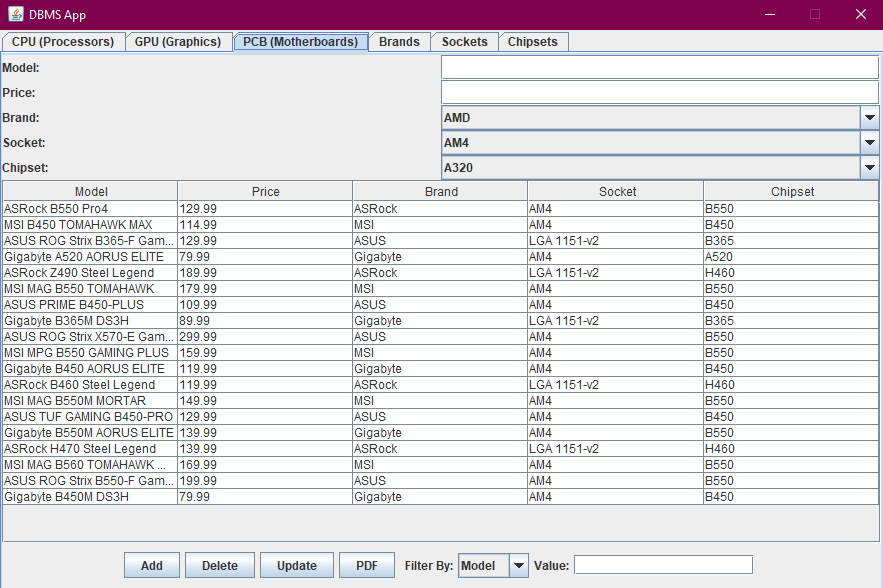
**3. ИНТЕРФЕЙС ПРОГРАММЫ**

**

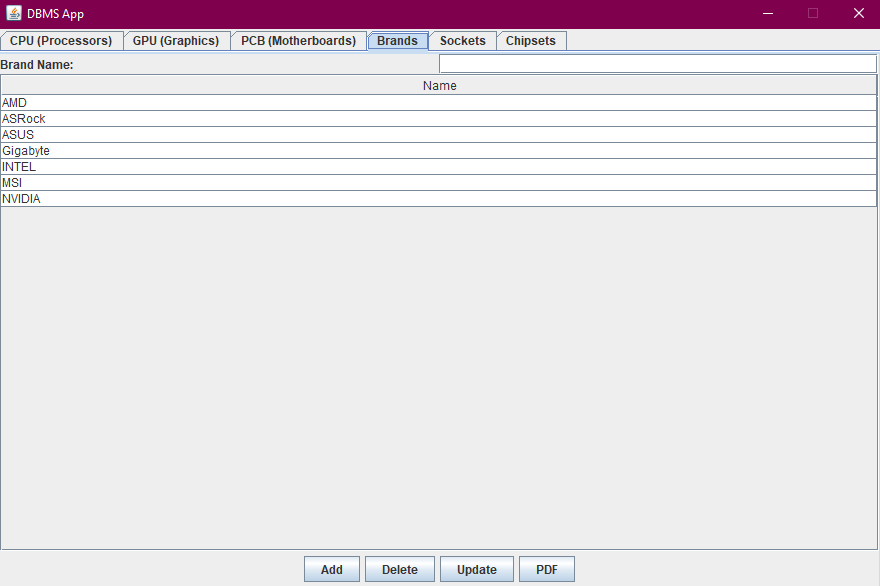
*Рисунок 3. Интерфейс программы. Таблица процессоров*

**

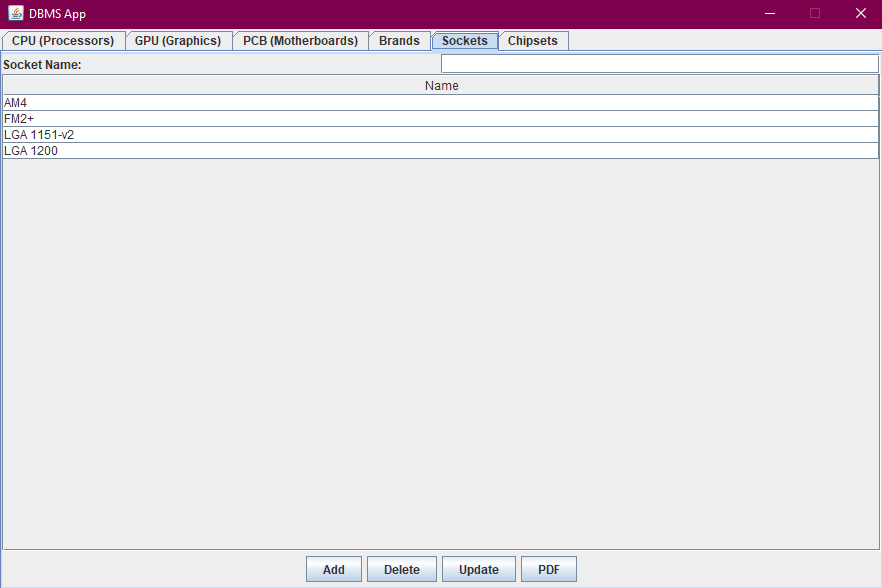
*Рисунок 4. Интерфейс программы. Таблица графических процессоров*

**

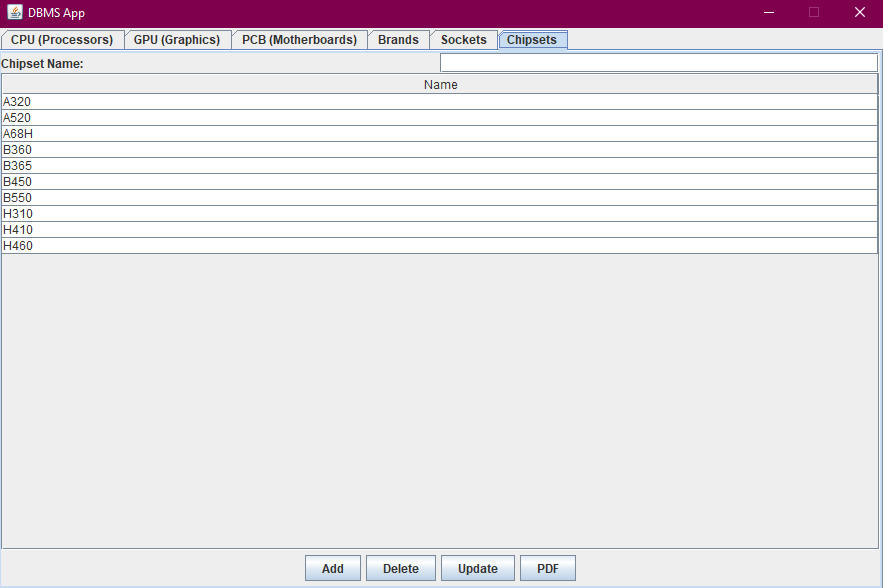
*Рисунок 5. Интерфейс программы. Таблица материнских плат*

**

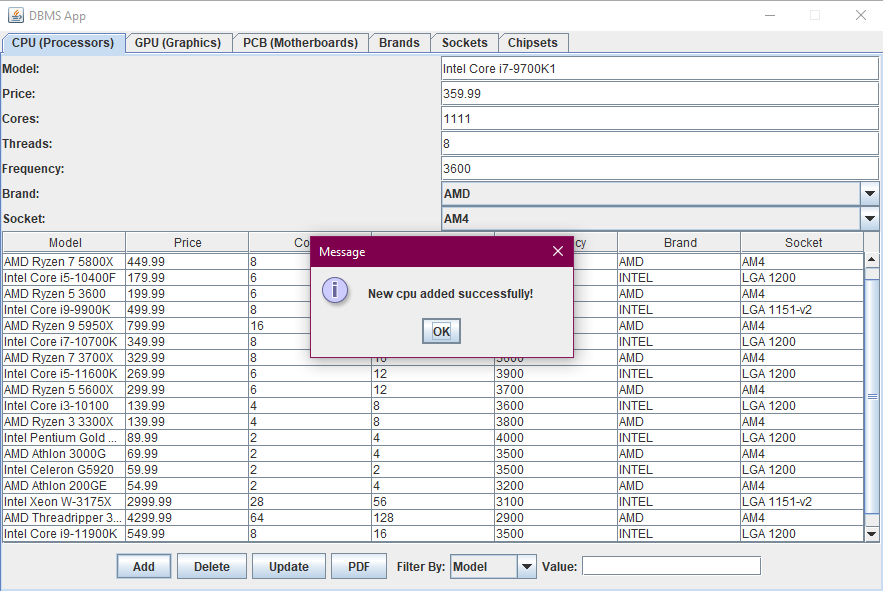
*Рисунок 6. Интерфейс программы. Таблица брендов*

**

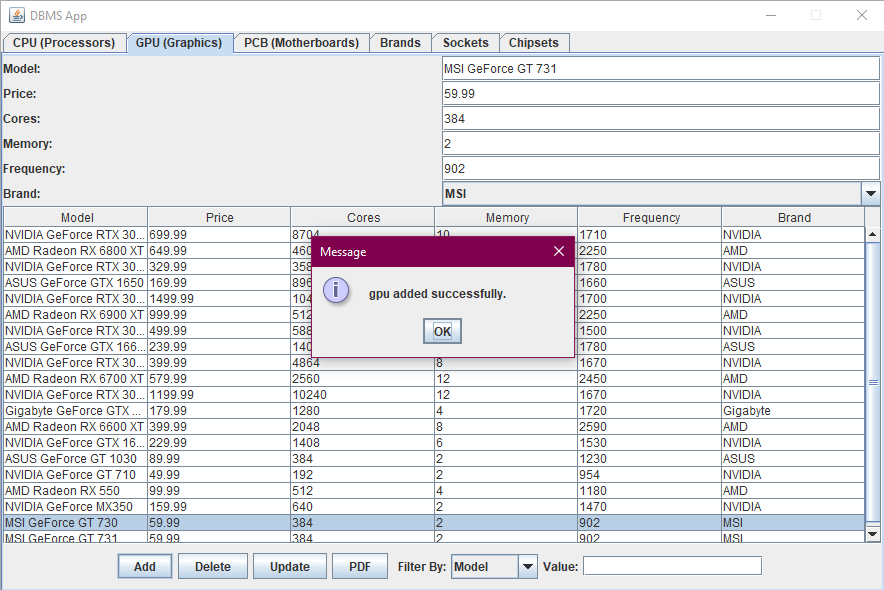
*Рисунок 7. Интерфейс программы. Таблица сокетов*

**

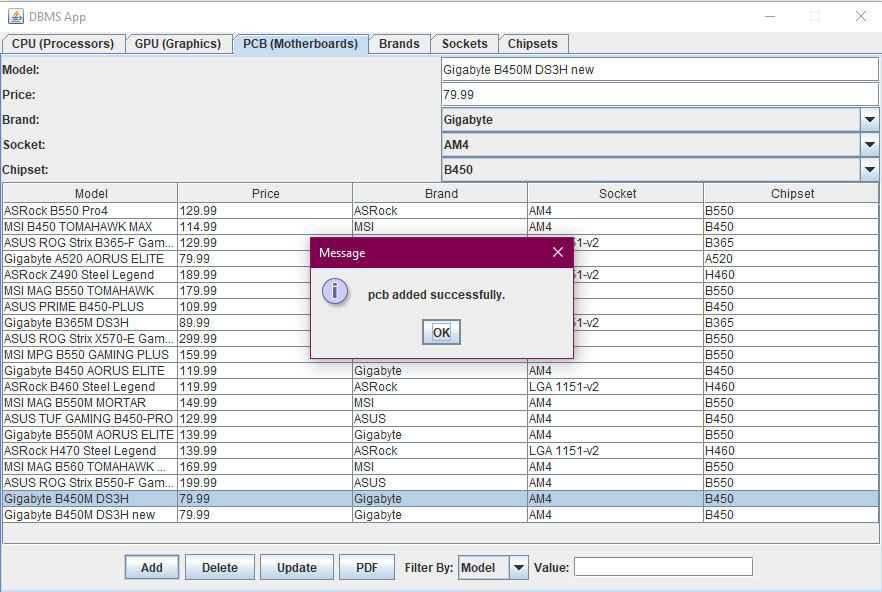
*Рисунок 8. Интерфейс программы. Таблица чипсетов*

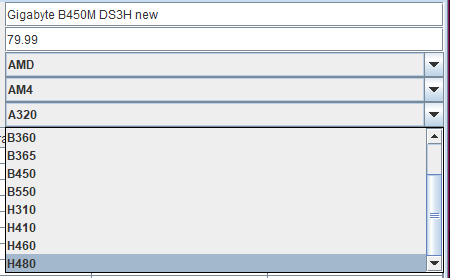
****

*Рисунок 9. Пример добавления новой записи*

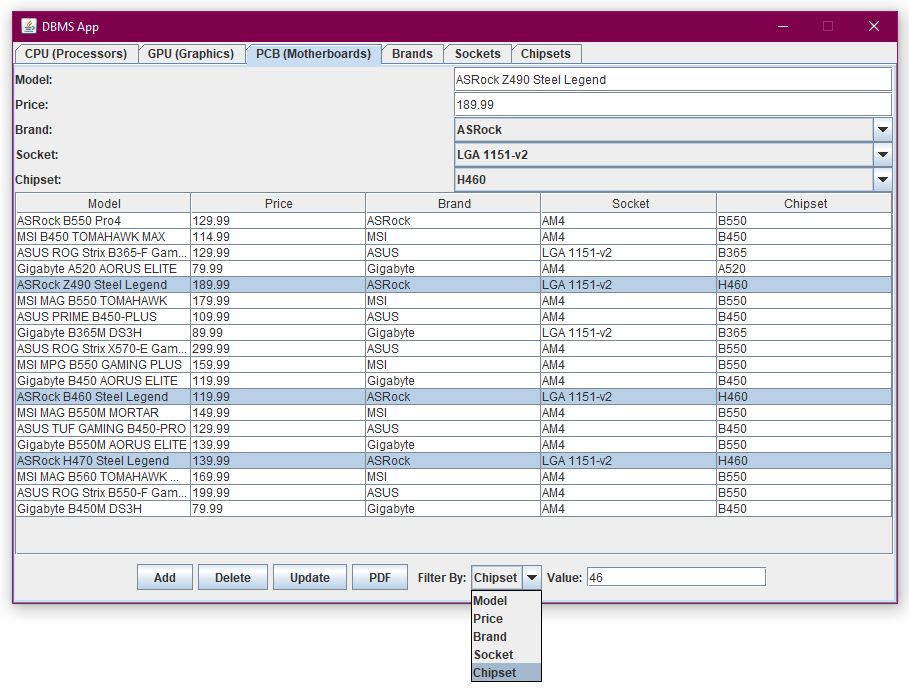
****

*Рисунок 10. Пример добавления новой записи*

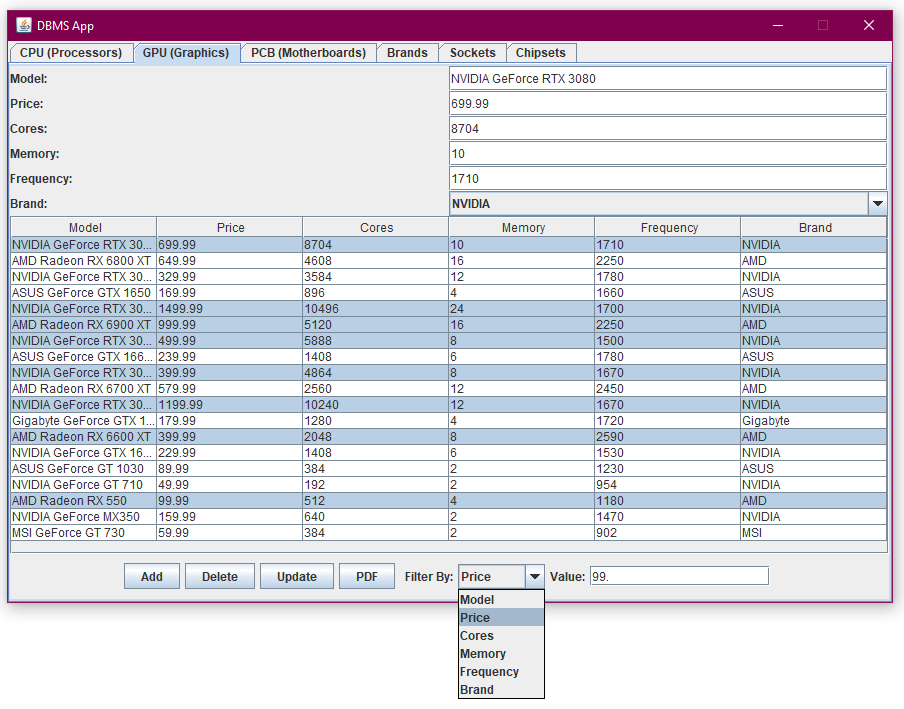
****

*Рисунок 11. Пример добавления новой записи*

*Рисунок 12. Пример обновления элемента ComboBox*

**

*Рисунок 13. Пример работы фильтра по полю Chipset*

****

*Рисунок 13. Пример работы фильтра по полю Price*

**4. ГЕНЕРАЦИЯ ОТЧЕТОВ**

В данной программной реализации есть возможность экспорта таблиц в виде PDF документа.



*Рисунок 11. Отчет по процессорам*



*Рисунок 12. Отчет по видеокартам*

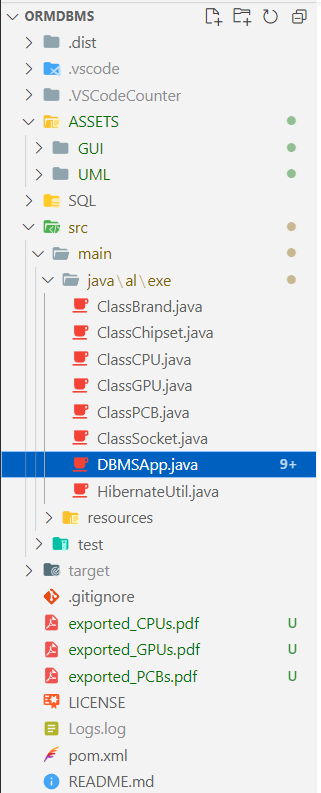


*Рисунок 13. Отчет по материнским платам*

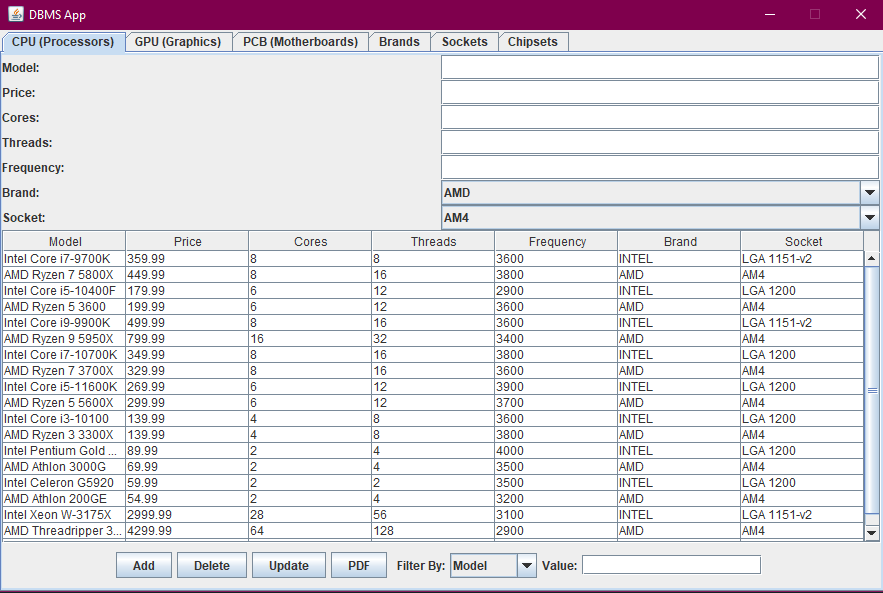
**5. КРАТКОЕ РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

DBMSApp - это приложение для управления базой данных с использованием графического интерфейса на основе Java Swing. Приложение позволяет добавлять, удалять и обновлять данные в таблицах базы данных, основанных на структуре "pc".

1. Установка и запуск приложения
   * Убедитесь, что на вашем компьютере установлена Java Runtime Environment (JRE).
   * Скомпилируйте и запустите приложение DBMSApp.java.



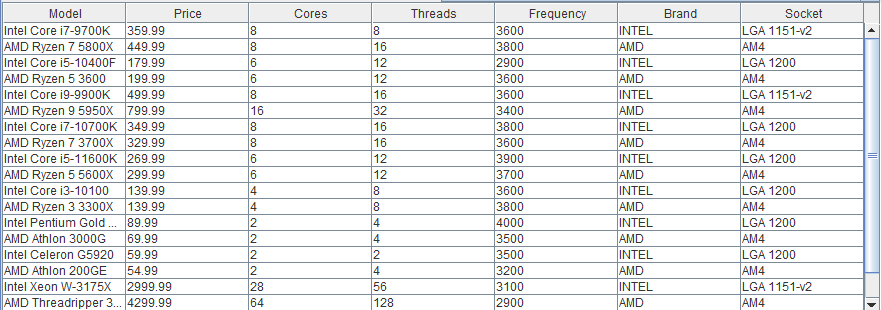
* + После запуска откроется главное окно приложения.



1. Интерфейс пользователя
   * Главное окно приложения состоит из вкладок для каждой таблицы базы данных "pc".



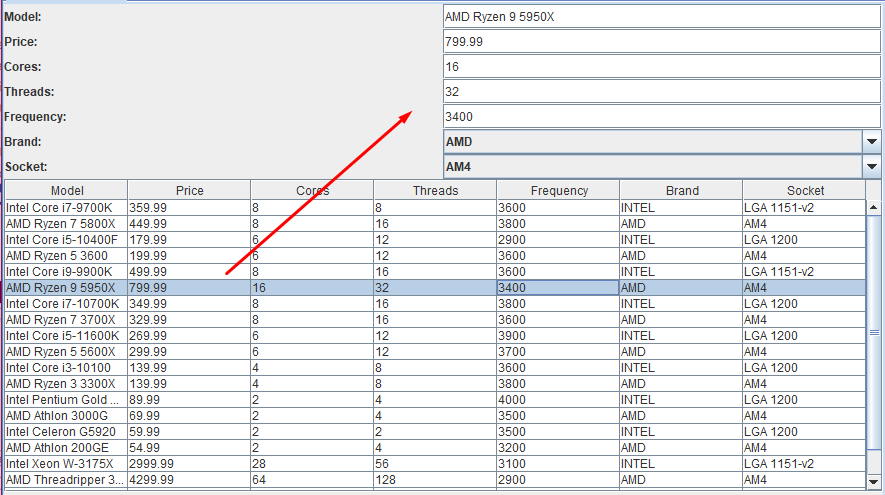
* + В каждой вкладке таблица отображает данные из соответствующей таблицы базы данных.



* + Для каждой таблицы доступны кнопки "Добавить", "Удалить" и "Обновить".



* + При выборе строки в таблице срабатывает событие выбора, которое можно использовать для дополнительной логики.



1. Добавление данных
   * Чтобы добавить запись в таблицу, выберите соответствующую вкладку и нажмите кнопку "Add".



* + В появившемся диалоговом окне введите значения для всех полей и нажмите кнопку "ОК".
  + Запись будет добавлена в таблицу и отображена в графическом интерфейсе.

1. Удаление данных
   * Чтобы удалить запись из таблицы, выберите соответствующую вкладку и выделите нужную строку в таблице.
   * Нажмите кнопку "Delete".



* + Запись будет удалена из таблицы и удалится из графического интерфейса.

1. Обновление данных
   * Чтобы обновить запись в таблице, выберите соответствующую вкладку и выделите нужную строку в таблице.
   * Нажмите кнопку "Update".

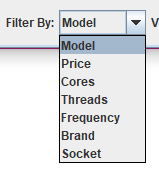


* + В появившемся диалоговом окне введите новые значения для полей и нажмите кнопку "ОК".
  + Запись будет обновлена в таблице и изменится в графическом интерфейсе.

1. Так же, чтобы выгрузить текущую таблицу в виде документа, необходимо нажать на кнопку "PDF".



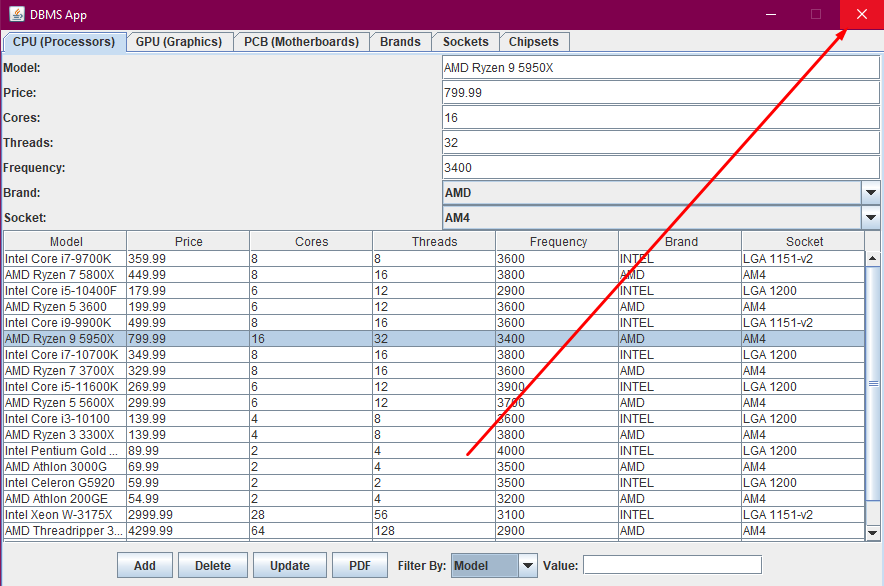
1. Для использования функции фильтрации таблиц
   * выберите необходимое поле, по которому вы хотите произвести фильтрацию данных:



* + затем введите в рядом расположенное поле необходимый текст для отбора данных



1. Выход из приложения
   * Чтобы выйти из приложения, закройте главное окно приложения.



**6. СПИСОК ИСТОЧНИКОВ**

1. Java Swing официальная документация:
   * Ссылка: <https://docs.oracle.com/javase/tutorial/uiswing/index.html>
2. Maven официальная документация:
   * Ссылка: <https://maven.apache.org/guides/index.html>
3. Hibernate официальная документация:
   * Ссылка: <https://hibernate.org/orm/documentation/>
4. Базовые концепции и примеры работы с Java Swing:
   * Java Swing Tutorial by Oracle: <https://docs.oracle.com/javase/tutorial/uiswing/>
   * The Java Tutorials - Creating a GUI With JFC/Swing: <https://docs.oracle.com/javase/tutorial/uiswing/>
5. Примеры использования Hibernate ORM:
   * Hibernate Getting Started Guide: <https://docs.jboss.org/hibernate/orm/5.6/quickstart/html_single/>
   * Hibernate Tutorial by Tutorialspoint: <https://www.tutorialspoint.com/hibernate/>
6. Примеры работы с базами данных в Java с использованием Hibernate:
   * Hibernate ORM with MySQL - Tutorial: <https://www.baeldung.com/hibernate-orm-mysql>
   * Hibernate Basics - Inserting Data into Database: <https://www.tutorialspoint.com/hibernate/hibernate_inserting_data.htm>
7. Руководства по использованию Maven:
   * Maven Tutorial by Mkyong: <https://www.mkyong.com/tutorials/maven-tutorials/>
8. Примеры работы с таблицами и кнопками в Java Swing:
   * Creating a Simple Table with Java Swing: <https://www.baeldung.com/java-swing-jtable>
   * Java Swing - JButton ActionListener Example: <https://www.javatpoint.com/java-swing-jbutton-actionlistener>

**приложение А**

'left to right direction

skinparam{

componentStyle uml2

classAttributeIconSize 0

handwritten false

backgroundcolor #c9efa3-fff1a0

}

class Brand {

- id: int

- name: String

+ getId(): int

+ setId(id: int): void

+ getName(): String

+ setName(name: String): void

}

class Chipset {

- id: int

- name: String

+ getId(): int

+ setId(id: int): void

+ getName(): String

+ setName(name: String): void

}

class CPU {

- id: int

- model: String

- price: double

- cores: int

- threads: int

- frequency: int

- brandId: int

- socketId: int

+ getId(): int

+ setId(id: int): void

+ getModel(): String

+ setModel(model: String): void

+ getPrice(): double

+ setPrice(price: double): void

+ getCores(): int

+ setCores(cores: int): void

+ getThreads(): int

+ setThreads(threads: int): void

+ getFrequency(): int

+ setFrequency(frequency: int): void

+ getBrandId(): int

+ setBrandId(brandId: int): void

+ getSocketId(): int

+ setSocketId(socketId: int): void

}

class GPU {

- id: int

- model: String

- price: double

- cores: int

- memory: int

- frequency: int

- brandId: int

+ getId(): int

+ setId(id: int): void

+ getModel(): String

+ setModel(model: String): void

+ getPrice(): double

+ setPrice(price: double): void

+ getCores(): int

+ setCores(cores: int): void

+ getMemory(): int

+ setMemory(memory: int): void

+ getFrequency(): int

+ setFrequency(frequency: int): void

+ getBrandId(): int

+ setBrandId(brandId: int): void

}

class Motherboard {

- id: int

- model: String

- price: double

- brandId: int

- socketId: int

- chipsetId: int

+ getId(): int

+ setId(id: int): void

+ getModel(): String

+ setModel(model: String): void

+ getPrice(): double

+ setPrice(price: double): void

+ getBrandId(): int

+ setBrandId(brandId: int): void

+ getSocketId(): int

+ setSocketId(socketId: int): void

+ getChipsetId(): int

+ setChipsetId(chipsetId: int): void

}

class Socket {

- id: int

- name: String

+ getId(): int

+ setId(id: int): void

+ getName(): String

+ setName(name: String): void

}

Brand "1" -- "many" CPU : has

Brand "1" -- "many" GPU : has

Brand "1" -- "many" Motherboard : has

Brand "1" -- "many" Chipset : has

Socket "1" -- "many" CPU : has

Socket "1" -- "many" Motherboard : has

Chipset "1" -- "many" Motherboard : has

**приложение б**

Код программы также представлен на интернет-ресурсе github: <https://github.com/alexeylepov/ormdbmsapp>.

package al.exe;

import java.io.\*;

import java.io.File;

import java.io.FileReader;

import java.io.IOException;

import java.io.BufferedReader;

import java.io.FileOutputStream;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.util.List;

import java.util.Vector;

import java.util.ArrayList;

import java.util.stream.Collectors;

import java.util.logging.Logger;

import java.util.logging.FileHandler;

import java.util.logging.SimpleFormatter;

import javax.swing.\*;

import javax.swing.JTable;

import javax.swing.JButton;

import javax.swing.JOptionPane;

import javax.swing.JFileChooser;

import javax.swing.event.DocumentEvent;

import javax.swing.event.DocumentListener;

import javax.swing.event.ListSelectionEvent;

import javax.swing.event.ListSelectionListener;

import javax.swing.table.DefaultTableModel;

import com.itextpdf.text.Font;

import com.itextpdf.text.Phrase;

import com.itextpdf.text.Element;

import com.itextpdf.text.Document;

import com.itextpdf.text.BaseColor;

import com.itextpdf.text.FontFactory;

import com.itextpdf.text.pdf.PdfPCell;

import com.itextpdf.text.pdf.PdfPTable;

import com.itextpdf.text.pdf.PdfWriter;

// Hibernate

import org.hibernate.Session;

import org.hibernate.Transaction;

import org.hibernate.SessionFactory;

import org.hibernate.cfg.Configuration;

// Necessary classes

import al.exe.ClassCPU;

import al.exe.ClassGPU;

import al.exe.ClassPCB;

import al.exe.ClassBrand;

import al.exe.ClassSocket;

import al.exe.ClassChipset;

import al.exe.HibernateUtil;

public class DBMSApp extends JFrame

{

private static final Logger LOGGER = Logger.getLogger(DBMSApp.class.getName());

private static FileHandler fileHandler;

// Hibernate

private Session session;

private Transaction transaction;

private SessionFactory sessionFactory;

// Swing gui components

private JTabbedPane tabbedPane;

private JTable brandTable, chipsetTable, cpuTable, gpuTable, pcbTable, socketTable;

private DefaultTableModel brandTableModel, chipsetTableModel, cpuTableModel, gpuTableModel, pcbTableModel, socketTableModel;

// Buttons

private JButton addCPUButton, deleteCPUButton, updateCPUButton, pdfExportCPUButton;

private JButton addGPUButton, deleteGPUButton, updateGPUButton, pdfExportGPUButton;

private JButton addPCBButton, deletePCBButton, updatePCBButton, pdfExportPCBButton;

private JButton addBrandButton, deleteBrandButton, updateBrandButton, pdfExportBrandButton;

private JButton addSocketButton, deleteSocketButton, updateSocketButton, pdfExportSocketButton;

private JButton addChipsetButton, deleteChipsetButton, updateChipsetButton, pdfExportChipsetButton;

// CPU Fields

private JTextField cpuModelField;

private JTextField cpuPriceField;

private JTextField cpuCoresField;

private JTextField cpuThreadsField;

private JTextField cpuFrequencyField;

private JComboBox<String> cpuBrandComboBox;

private JComboBox<String> cpuSocketComboBox;

// GPU Fields

private JTextField gpuModelField;

private JTextField gpuPriceField;

private JTextField gpuCoresField;

private JTextField gpuMemoryField;

private JTextField gpuFrequencyField;

private JComboBox<String> gpuBrandComboBox;

// PCB Fields

private JTextField pcbModelField;

private JTextField pcbPriceField;

private JComboBox<String> pcbBrandComboBox;

private JComboBox<String> pcbSocketComboBox;

private JComboBox<String> pcbChipsetComboBox;

// Brand Fields

private JTextField brandNameField;

// Socket Fields

private JTextField socketNameField;

// Chipset Fields

private JTextField chipsetNameField;

// Create filter components

private JComboBox<String> filterCPUComboBox, filterPCBComboBox, filterGPUComboBox;

private JTextField filterCPUTextField, filterPCBTextField, filterGPUTextField;

// Create regular expression

// Explanation of the pattern:

// ^ indicates the start of the string.

// [A-Za-z0-9\\s] matches any uppercase or lowercase letter, digit, or whitespace character.

// + ensures that there is at least one or more of the preceding characters.

// $ indicates the end of the string.

private String modelRegex = "^[A-Z][a-zA-Z0-9\\s-+]\*$";

private String priceRegex = "\\d+(\\.\\d{1,2})?";

private String coresRegex = "\\d+";

private String threadsRegex = "\\d+";

private String frequencyRegex = "\\d+";

private String memoryRegex = "\\d+";

private String brandNameRegex = "^[A-Z][a-zA-Z0-9\\s-+]\*$";

private String socketNameRegex = "^[A-Z][a-zA-Z0-9\\s-+]\*$";

private String chipsetNameRegex = "^[A-Z][a-zA-Z0-9\\s-+]\*$";

//exception class

private class TextFieldException extends Exception

{

public TextFieldException()

{

super ("Fill all text fields!");

}

}

// exception function

public void checkIfEmpty (JTextField field) throws TextFieldException,NullPointerException

{

String sName = field.getText();

if (sName.isBlank()) throw new TextFieldException();

if (sName.length() == 0) throw new NullPointerException();

}

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

// \_\_\_\_\_\_\_ \_\_\_\_\_\_\_ \_\_ \_\_ \_\_\_\_\_\_ \_\_\_\_\_\_ \_\_\_\_\_\_\_ \_\_\_\_\_\_\_ \_\_\_\_\_\_ \_\_ \_\_ \_\_\_\_\_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_ \_\_\_\_\_\_ \_\_ \_\_\_\_\_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_ \_\_\_\_\_\_ \_\_ \_\_ //

// / \ / \ / \ / | / \ / \ / \ / \ / |/ \ / |/ |/ |/ | / \ / | / |/ | / \ / |/ | / \ / \ / | //

// $$$$$$$ |$$$$$$$ |$$ \ /$$ |/$$$$$$ |/$$$$$$ |$$$$$$$ |$$$$$$$ | $$$$$$/ $$ \ $$ |$$$$$$/ $$$$$$$$/ $$$$$$/ /$$$$$$ |$$ | $$$$$$/ $$$$$$$$/ /$$$$$$ |$$$$$$$$/ $$$$$$/ /$$$$$$ |$$ \ $$ | //

// $$ | $$ |$$ |\_\_$$ |$$$ \ /$$$ |$$ \\_\_$$/ $$ |\_\_$$ |$$ |\_\_$$ |$$ |\_\_$$ | $$ | $$$ \$$ | $$ | $$ | $$ | $$ |\_\_$$ |$$ | $$ | /$$/ $$ |\_\_$$ | $$ | $$ | $$ | $$ |$$$ \$$ | //

// $$ | $$ |$$ $$< $$$$ /$$$$ |$$ \ $$ $$ |$$ $$/ $$ $$/ $$ | $$$$ $$ | $$ | $$ | $$ | $$ $$ |$$ | $$ | /$$/ $$ $$ | $$ | $$ | $$ | $$ |$$$$ $$ | //

// $$ | $$ |$$$$$$$ |$$ $$ $$/$$ | $$$$$$ |$$$$$$$$ |$$$$$$$/ $$$$$$$/ $$ | $$ $$ $$ | $$ | $$ | $$ | $$$$$$$$ |$$ | $$ | /$$/ $$$$$$$$ | $$ | $$ | $$ | $$ |$$ $$ $$ | //

// $$ |\_\_$$ |$$ |\_\_$$ |$$ |$$$/ $$ |/ \\_\_$$ |$$ | $$ |$$ | $$ | \_$$ |\_ $$ |$$$$ | \_$$ |\_ $$ | \_$$ |\_ $$ | $$ |$$ |\_\_\_\_\_ \_$$ |\_ /$$/\_\_\_\_ $$ | $$ | $$ | \_$$ |\_ $$ \\_\_$$ |$$ |$$$$ | //

// $$ $$/ $$ $$/ $$ | $/ $$ |$$ $$/ $$ | $$ |$$ | $$ | / $$ |$$ | $$$ |/ $$ | $$ | / $$ |$$ | $$ |$$ |/ $$ |/$$ |$$ | $$ | $$ | / $$ |$$ $$/ $$ | $$$ | //

// $$$$$$$/ $$$$$$$/ $$/ $$/ $$$$$$/ $$/ $$/ $$/ $$/ $$$$$$/ $$/ $$/ $$$$$$/ $$/ $$$$$$/ $$/ $$/ $$$$$$$$/ $$$$$$/ $$$$$$$$/ $$/ $$/ $$/ $$$$$$/ $$$$$$/ $$/ $$/ //

// //

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

public DBMSApp()

{ // Initialize window

setTitle("DBMS App");

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setSize(900, 600);

setLocationRelativeTo(null);

setResizable(false);

// Initialize

tabbedPane = new JTabbedPane();

// Initialize Hibernate session factory

sessionFactory = new Configuration().configure().buildSessionFactory();

session = sessionFactory.openSession();

transaction = null;

// Create table models

cpuTableModel = new DefaultTableModel(new Object[]{"Model", "Price", "Cores", "Threads", "Frequency", "Brand", "Socket"}, 0);

gpuTableModel = new DefaultTableModel(new Object[]{"Model", "Price", "Cores", "Memory", "Frequency", "Brand"}, 0);

pcbTableModel = new DefaultTableModel(new Object[]{"Model", "Price", "Brand", "Socket", "Chipset"}, 0);

brandTableModel = new DefaultTableModel(new Object[]{"Name"}, 0);

socketTableModel = new DefaultTableModel(new Object[]{"Name"}, 0);

chipsetTableModel = new DefaultTableModel(new Object[]{"Name"}, 0);

// Create tables

cpuTable = new JTable(cpuTableModel);

gpuTable = new JTable(gpuTableModel);

pcbTable = new JTable(pcbTableModel);

brandTable = new JTable(brandTableModel);

socketTable = new JTable(socketTableModel);

chipsetTable = new JTable(chipsetTableModel);

cpuTable.setDefaultEditor(Object.class, null);

gpuTable.setDefaultEditor(Object.class, null);

pcbTable.setDefaultEditor(Object.class, null);

brandTable.setDefaultEditor(Object.class, null);

socketTable.setDefaultEditor(Object.class, null);

chipsetTable.setDefaultEditor(Object.class, null);

// Add tables to scroll panes

JScrollPane cpuScrollPane = new JScrollPane(cpuTable);

JScrollPane gpuScrollPane = new JScrollPane(gpuTable);

JScrollPane pcbScrollPane = new JScrollPane(pcbTable);

JScrollPane brandScrollPane = new JScrollPane(brandTable);

JScrollPane socketScrollPane = new JScrollPane(socketTable);

JScrollPane chipsetScrollPane = new JScrollPane(chipsetTable);

// Create buttons

addBrandButton = new JButton("Add");

deleteBrandButton = new JButton("Delete");

updateBrandButton = new JButton("Update");

pdfExportBrandButton = new JButton("PDF");

addChipsetButton = new JButton("Add");

deleteChipsetButton = new JButton("Delete");

updateChipsetButton = new JButton("Update");

pdfExportChipsetButton = new JButton("PDF");

addCPUButton = new JButton("Add");

deleteCPUButton = new JButton("Delete");

updateCPUButton = new JButton("Update");

pdfExportCPUButton = new JButton("PDF");

addGPUButton = new JButton("Add");

deleteGPUButton = new JButton("Delete");

updateGPUButton = new JButton("Update");

pdfExportGPUButton = new JButton("PDF");

addPCBButton = new JButton("Add");

deletePCBButton = new JButton("Delete");

updatePCBButton = new JButton("Update");

pdfExportPCBButton = new JButton("PDF");

addSocketButton = new JButton("Add");

deleteSocketButton = new JButton("Delete");

updateSocketButton = new JButton("Update");

pdfExportSocketButton = new JButton("PDF");

/////////////////////////////////////////

// //

// Create Fields and Combo Boxes //

// //

/////////////////////////////////////////

// Create CPU Fields

cpuModelField = new JTextField(255);

cpuPriceField = new JTextField(10);

cpuCoresField = new JTextField(5);

cpuThreadsField = new JTextField(5);

cpuFrequencyField = new JTextField(10);

cpuBrandComboBox = new JComboBox<>();

cpuSocketComboBox = new JComboBox<>();

// Create GPU Fields

gpuModelField = new JTextField(255);

gpuPriceField = new JTextField(10);

gpuCoresField = new JTextField(5);

gpuMemoryField = new JTextField(5);

gpuFrequencyField = new JTextField(10);

gpuBrandComboBox = new JComboBox<>();

// Create PCB Fields

pcbModelField = new JTextField(255);

pcbPriceField = new JTextField(10);

pcbBrandComboBox = new JComboBox<>();

pcbSocketComboBox = new JComboBox<>();

pcbChipsetComboBox = new JComboBox<>();

// Create Brand fields

brandNameField = new JTextField(255);

// Create Socket fields

socketNameField = new JTextField(255);

// Create Chipset fields

chipsetNameField = new JTextField(255);

/////////////////////////////

// //

// Filter Components //

// //

/////////////////////////////

// Create filter components

filterCPUComboBox = new JComboBox<>(new String[]{"Model", "Price", "Cores", "Threads", "Frequency", "Brand", "Socket"});

filterCPUTextField = new JTextField(16);

filterGPUComboBox = new JComboBox<>(new String[]{"Model", "Price", "Cores", "Memory", "Frequency", "Brand"});

filterGPUTextField = new JTextField(16);

filterPCBComboBox = new JComboBox<>(new String[]{"Model", "Price", "Brand", "Socket", "Chipset"});

filterPCBTextField = new JTextField(16);

// Add elements to the panel

JPanel filterCPUPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));

filterCPUPanel.add(new JLabel("Filter By:"));

filterCPUPanel.add(filterCPUComboBox);

filterCPUPanel.add(new JLabel("Value:"));

filterCPUPanel.add(filterCPUTextField);

// Add elements to the panel

JPanel filterGPUPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));

filterGPUPanel.add(new JLabel("Filter By:"));

filterGPUPanel.add(filterGPUComboBox);

filterGPUPanel.add(new JLabel("Value:"));

filterGPUPanel.add(filterGPUTextField);

// Add elements to the panel

JPanel filterPCBPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));

filterPCBPanel.add(new JLabel("Filter By:"));

filterPCBPanel.add(filterPCBComboBox);

filterPCBPanel.add(new JLabel("Value:"));

filterPCBPanel.add(filterPCBTextField);

////////////////////////

// //

// Input Panels //

// //

////////////////////////

// For combobox filling

List<ClassBrand> tempbrands = retrieveBrands();

List<ClassSocket> tempsockets = retrieveSockets();

List<ClassChipset> tempchipsets = retrieveChipsets();

// cpu panel

JPanel cpuInputPanel = new JPanel(new GridLayout(7, 2));

cpuInputPanel.add(new JLabel("Model:"));

cpuInputPanel.add(cpuModelField);

cpuInputPanel.add(new JLabel("Price:"));

cpuInputPanel.add(cpuPriceField);

cpuInputPanel.add(new JLabel("Cores:"));

cpuInputPanel.add(cpuCoresField);

cpuInputPanel.add(new JLabel("Threads:"));

cpuInputPanel.add(cpuThreadsField);

cpuInputPanel.add(new JLabel("Frequency:"));

cpuInputPanel.add(cpuFrequencyField);

cpuInputPanel.add(new JLabel("Brand:"));

cpuInputPanel.add(cpuBrandComboBox);

cpuInputPanel.add(new JLabel("Socket:"));

cpuInputPanel.add(cpuSocketComboBox);

for (ClassBrand tempbrand : tempbrands)

{

cpuBrandComboBox.addItem(tempbrand.getName());

}

for (ClassSocket tempsocket : tempsockets)

{

cpuSocketComboBox.addItem(tempsocket.getName());

}

// gpu panel

JPanel gpuInputPanel = new JPanel(new GridLayout(6, 2));

gpuInputPanel.add(new JLabel("Model:"));

gpuInputPanel.add(gpuModelField);

gpuInputPanel.add(new JLabel("Price:"));

gpuInputPanel.add(gpuPriceField);

gpuInputPanel.add(new JLabel("Cores:"));

gpuInputPanel.add(gpuCoresField);

gpuInputPanel.add(new JLabel("Memory:"));

gpuInputPanel.add(gpuMemoryField);

gpuInputPanel.add(new JLabel("Frequency:"));

gpuInputPanel.add(gpuFrequencyField);

gpuInputPanel.add(new JLabel("Brand:"));

gpuInputPanel.add(gpuBrandComboBox);

for (ClassBrand tempbrand : tempbrands)

{

gpuBrandComboBox.addItem(tempbrand.getName());

}

// motherboard panel

JPanel pcbInputPanel = new JPanel(new GridLayout(5, 2));

pcbInputPanel.add(new JLabel("Model:"));

pcbInputPanel.add(pcbModelField);

pcbInputPanel.add(new JLabel("Price:"));

pcbInputPanel.add(pcbPriceField);

pcbInputPanel.add(new JLabel("Brand:"));

pcbInputPanel.add(pcbBrandComboBox);

pcbInputPanel.add(new JLabel("Socket:"));

pcbInputPanel.add(pcbSocketComboBox);

pcbInputPanel.add(new JLabel("Chipset:"));

pcbInputPanel.add(pcbChipsetComboBox);

for (ClassBrand tempbrand : tempbrands)

{

pcbBrandComboBox.addItem(tempbrand.getName());

}

for (ClassSocket tempsocket : tempsockets)

{

pcbSocketComboBox.addItem(tempsocket.getName());

}

for (ClassChipset tempchipset : tempchipsets)

{

pcbChipsetComboBox.addItem(tempchipset.getName());

}

// brand panel

JPanel brandInputPanel = new JPanel(new GridLayout(1, 2));

brandInputPanel.add(new JLabel("Brand Name:"));

brandInputPanel.add(brandNameField);

// socket panel

JPanel socketInputPanel = new JPanel(new GridLayout(1, 2));

socketInputPanel.add(new JLabel("Socket Name:"));

socketInputPanel.add(socketNameField);

// chipset panel

JPanel chipsetInputPanel = new JPanel(new GridLayout(1, 2));

chipsetInputPanel.add(new JLabel("Chipset Name:"));

chipsetInputPanel.add(chipsetNameField);

/////////////////////////

// //

// Button panels //

// //

/////////////////////////

JPanel cpuButtonPanel = createButtonPanel(addCPUButton, deleteCPUButton, updateCPUButton, pdfExportCPUButton);

cpuButtonPanel.add(filterCPUPanel, BorderLayout.NORTH);

JPanel gpuButtonPanel = createButtonPanel(addGPUButton, deleteGPUButton, updateGPUButton, pdfExportGPUButton);

gpuButtonPanel.add(filterGPUPanel, BorderLayout.NORTH);

JPanel pcbButtonPanel = createButtonPanel(addPCBButton, deletePCBButton, updatePCBButton, pdfExportPCBButton);

pcbButtonPanel.add(filterPCBPanel, BorderLayout.NORTH);

JPanel brandButtonPanel = createButtonPanel(addBrandButton, deleteBrandButton, updateBrandButton, pdfExportBrandButton);

JPanel socketButtonPanel = createButtonPanel(addSocketButton, deleteSocketButton, updateSocketButton, pdfExportSocketButton);

JPanel chipsetButtonPanel = createButtonPanel(addChipsetButton, deleteChipsetButton, updateChipsetButton, pdfExportChipsetButton);

////////////////////////////////////////////

// //

// Add tables and buttons to panels //

// //

////////////////////////////////////////////

// Add CPU panel

JPanel cpuPanel = new JPanel(new BorderLayout());

cpuPanel.add(cpuScrollPane, BorderLayout.CENTER);

cpuPanel.add(cpuButtonPanel, BorderLayout.SOUTH);

cpuPanel.add(cpuInputPanel, BorderLayout.NORTH);

// Add GPU panel

JPanel gpuPanel = new JPanel(new BorderLayout());

gpuPanel.add(gpuScrollPane, BorderLayout.CENTER);

gpuPanel.add(gpuButtonPanel, BorderLayout.SOUTH);

gpuPanel.add(gpuInputPanel, BorderLayout.NORTH);

// Add PCB panel

JPanel pcbPanel = new JPanel(new BorderLayout());

pcbPanel.add(pcbScrollPane, BorderLayout.CENTER);

pcbPanel.add(pcbButtonPanel, BorderLayout.SOUTH);

pcbPanel.add(pcbInputPanel, BorderLayout.NORTH);

// Add Brand panel

JPanel brandPanel = new JPanel(new BorderLayout());

brandPanel.add(brandScrollPane, BorderLayout.CENTER);

brandPanel.add(brandButtonPanel, BorderLayout.SOUTH);

brandPanel.add(brandInputPanel, BorderLayout.NORTH);

// Add Chipset panel

JPanel chipsetPanel = new JPanel(new BorderLayout());

chipsetPanel.add(chipsetScrollPane, BorderLayout.CENTER);

chipsetPanel.add(chipsetButtonPanel, BorderLayout.SOUTH);

chipsetPanel.add(chipsetInputPanel, BorderLayout.NORTH);

// Add Socket panel

JPanel socketPanel = new JPanel(new BorderLayout());

socketPanel.add(socketScrollPane, BorderLayout.CENTER);

socketPanel.add(socketButtonPanel, BorderLayout.SOUTH);

socketPanel.add(socketInputPanel, BorderLayout.NORTH);

////////////////////

// //

// Add tabs //

// //

////////////////////

// Add panels to tabbed pane

tabbedPane.addTab("CPU (Processors)", cpuPanel);

tabbedPane.addTab("GPU (Graphics)", gpuPanel);

tabbedPane.addTab("PCB (Motherboards)", pcbPanel);

tabbedPane.addTab("Brands", brandPanel);

tabbedPane.addTab("Sockets", socketPanel);

tabbedPane.addTab("Chipsets", chipsetPanel);

// Add tabbed pane to content pane

add(tabbedPane);

// Populate tables

populateTables();

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

// \_\_\_\_\_\_\_\_ \_\_\_\_\_\_ \_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_ \_\_ \_\_\_\_\_\_ \_\_\_\_\_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_ \_\_ \_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_ \_\_\_\_\_\_ //

// / |/ |/ | / |/ |/ \ / | / | / \ / |/ |/ \ / |/ |/ \ / \ //

// $$$$$$$$/ $$$$$$/ $$ | $$$$$$$$/ $$$$$$$$/ $$$$$$$ | $$ | $$$$$$/ /$$$$$$ |$$$$$$$$/ $$$$$$$$/ $$ \ $$ |$$$$$$$$/ $$$$$$$ |/$$$$$$ | //

// $$ |\_\_ $$ | $$ | $$ | $$ |\_\_ $$ |\_\_$$ | $$ | $$ | $$ \\_\_$$/ $$ | $$ |\_\_ $$$ \$$ |$$ |\_\_ $$ |\_\_$$ |$$ \\_\_$$/ //

// $$ | $$ | $$ | $$ | $$ | $$ $$< $$ | $$ | $$ \ $$ | $$ | $$$$ $$ |$$ | $$ $$< $$ \ //

// $$$$$/ $$ | $$ | $$ | $$$$$/ $$$$$$$ | $$ | $$ | $$$$$$ | $$ | $$$$$/ $$ $$ $$ |$$$$$/ $$$$$$$ | $$$$$$ | //

// $$ | \_$$ |\_ $$ |\_\_\_\_\_ $$ | $$ |\_\_\_\_\_ $$ | $$ | $$ |\_\_\_\_\_ \_$$ |\_ / \\_\_$$ | $$ | $$ |\_\_\_\_\_ $$ |$$$$ |$$ |\_\_\_\_\_ $$ | $$ |/ \\_\_$$ | //

// $$ | / $$ |$$ |$$ | $$ |$$ | $$ | $$ |/ $$ |$$ $$/ $$ | $$ |$$ | $$$ |$$ |$$ | $$ |$$ $$/ //

// $$/ $$$$$$/ $$$$$$$$/ $$/ $$$$$$$$/ $$/ $$/ $$$$$$$$/ $$$$$$/ $$$$$$/ $$/ $$$$$$$$/ $$/ $$/ $$$$$$$$/ $$/ $$/ $$$$$$/ //

// //

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

filterCPUComboBox.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

applyCPUFilter();

}

});

filterCPUTextField.getDocument().addDocumentListener(new DocumentListener() {

@Override

public void insertUpdate(DocumentEvent e) {

applyCPUFilter();

}

@Override

public void removeUpdate(DocumentEvent e) {

applyCPUFilter();

}

@Override

public void changedUpdate(DocumentEvent e) {

applyCPUFilter();

}

});

filterGPUComboBox.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

applyGPUFilter();

}

});

filterGPUTextField.getDocument().addDocumentListener(new DocumentListener() {

@Override

public void insertUpdate(DocumentEvent e) {

applyGPUFilter();

}

@Override

public void removeUpdate(DocumentEvent e) {

applyGPUFilter();

}

@Override

public void changedUpdate(DocumentEvent e) {

applyGPUFilter();

}

});

filterPCBComboBox.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

applyPCBFilter();

}

});

filterPCBTextField.getDocument().addDocumentListener(new DocumentListener() {

@Override

public void insertUpdate(DocumentEvent e) {

applyPCBFilter();

}

@Override

public void removeUpdate(DocumentEvent e) {

applyPCBFilter();

}

@Override

public void changedUpdate(DocumentEvent e) {

applyPCBFilter();

}

});

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

// \_\_\_\_\_\_ \_\_\_\_\_\_\_ \_\_ \_\_ \_\_ \_\_\_\_\_\_ \_\_\_\_\_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_ \_\_ \_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_ \_\_\_\_\_\_ //

// / \ / \ / | / | / | / | / \ / |/ |/ \ / |/ |/ \ / \ //

// /$$$$$$ |$$$$$$$ |$$ | $$ | $$ | $$$$$$/ /$$$$$$ |$$$$$$$$/ $$$$$$$$/ $$ \ $$ |$$$$$$$$/ $$$$$$$ |/$$$$$$ | //

// $$ | $$/ $$ |\_\_$$ |$$ | $$ | $$ | $$ | $$ \\_\_$$/ $$ | $$ |\_\_ $$$ \$$ |$$ |\_\_ $$ |\_\_$$ |$$ \\_\_$$/ //

// $$ | $$ $$/ $$ | $$ | $$ | $$ | $$ \ $$ | $$ | $$$$ $$ |$$ | $$ $$< $$ \ //

// $$ | \_\_ $$$$$$$/ $$ | $$ | $$ | $$ | $$$$$$ | $$ | $$$$$/ $$ $$ $$ |$$$$$/ $$$$$$$ | $$$$$$ | //

// $$ \\_\_/ |$$ | $$ \\_\_$$ | $$ |\_\_\_\_\_ \_$$ |\_ / \\_\_$$ | $$ | $$ |\_\_\_\_\_ $$ |$$$$ |$$ |\_\_\_\_\_ $$ | $$ |/ \\_\_$$ | //

// $$ $$/ $$ | $$ $$/ $$ |/ $$ |$$ $$/ $$ | $$ |$$ | $$$ |$$ |$$ | $$ |$$ $$/ //

// $$$$$$/ $$/ $$$$$$/ $$$$$$$$/ $$$$$$/ $$$$$$/ $$/ $$$$$$$$/ $$/ $$/ $$$$$$$$/ $$/ $$/ $$$$$$/ //

// //

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

{

addCPUButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e)

{

if (cpuModelField.getText().isBlank())

{

JOptionPane.showMessageDialog(cpuPanel, "You must fill all text fields first!");

}

else

{

if (cpuModelField.getText().matches(modelRegex) && cpuPriceField.getText().matches(priceRegex) && cpuCoresField.getText().matches(coresRegex) && cpuThreadsField.getText().matches(threadsRegex) && cpuFrequencyField.getText().matches(frequencyRegex))

{

String model = cpuModelField.getText();

double price = Double.parseDouble(cpuPriceField.getText());

int cores = Integer.parseInt(cpuCoresField.getText());

int threads = Integer.parseInt(cpuThreadsField.getText());

int frequency = Integer.parseInt(cpuFrequencyField.getText());

String brand = cpuBrandComboBox.getSelectedItem().toString();

String socket = cpuSocketComboBox.getSelectedItem().toString();

ClassBrand brandObj = (ClassBrand) session.createQuery("FROM ClassBrand WHERE name = :name").setParameter("name", brand).uniqueResult();

ClassSocket socketObj = (ClassSocket) session.createQuery("FROM ClassSocket WHERE name = :name").setParameter("name", socket).uniqueResult();

Object[] rowData = {model, price, cores, threads, frequency, brand, socket};

ClassCPU cpu = new ClassCPU();

cpu.setModel(model);

cpu.setPrice(price);

cpu.setCores(cores);

cpu.setThreads(threads);

cpu.setFrequency(frequency);

cpu.setBrand(brandObj);

cpu.setSocket(socketObj);

try (Session session = getSession())

{

Transaction transaction = session.beginTransaction();

session.save(cpu);

transaction.commit();

cpuTableModel.addRow(rowData);

JOptionPane.showMessageDialog(null, "New cpu added successfully!");

updateAllDropBoxes();

}

catch (Exception ex)

{

ex.printStackTrace();

JOptionPane.showMessageDialog(null, "Failed to add cpu: " + ex);

}

}

else

{

JOptionPane.showMessageDialog(null, "Invalid data format!");

}

}

}

});

updateCPUButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e)

{

int row = cpuTable.getSelectedRow();

if (row != -1) {

// create the popup window with yes/no options

int choice = JOptionPane.showConfirmDialog(cpuPanel, "Do you wish to continue? ", "Confirmation", JOptionPane.YES\_NO\_OPTION);

// handle the user's choice

if (choice == JOptionPane.YES\_OPTION)

{

if (cpuModelField.getText().matches(modelRegex) && cpuPriceField.getText().matches(priceRegex) && cpuCoresField.getText().matches(coresRegex) && cpuThreadsField.getText().matches(threadsRegex) && cpuFrequencyField.getText().matches(frequencyRegex))

{

int selectedRow = cpuTable.getSelectedRow();

if (selectedRow != -1) {

String oldModelName = (String) cpuTableModel.getValueAt(selectedRow, 0);

String model = cpuModelField.getText();

double price = Double.parseDouble(cpuPriceField.getText());

int cores = Integer.parseInt(cpuCoresField.getText());

int threads = Integer.parseInt(cpuThreadsField.getText());

int frequency = Integer.parseInt(cpuFrequencyField.getText());

String brand = cpuBrandComboBox.getSelectedItem().toString();

String socket = cpuSocketComboBox.getSelectedItem().toString();

ClassBrand brandObj = (ClassBrand) session.createQuery("FROM ClassBrand WHERE name = :name").setParameter("name", brand).uniqueResult();

ClassSocket socketObj = (ClassSocket) session.createQuery("FROM ClassSocket WHERE name = :name").setParameter("name", socket).uniqueResult();

if (model != null && !model.isEmpty()) {

try (Session session = getSession()) {

Transaction transaction = session.beginTransaction();

ClassCPU cpu = (ClassCPU) session.createQuery("FROM ClassCPU WHERE model = :model").setParameter("model", oldModelName).uniqueResult();

if (cpu != null) {

cpu.setModel(model);

cpu.setPrice(price);

cpu.setCores(cores);

cpu.setThreads(threads);

cpu.setFrequency(frequency);

cpu.setBrand(brandObj);

cpu.setSocket(socketObj);

session.update(cpu);

populateTables();

transaction.commit();

populateTables();

updateAllDropBoxes();

JOptionPane.showMessageDialog(null, "cpu updated successfully.");

} else {

JOptionPane.showMessageDialog(null, "cpu not found.");

}

} catch (Exception ex) {

ex.printStackTrace();

JOptionPane.showMessageDialog(null, "Failed to update cpu: " + ex);

}

} else {

JOptionPane.showMessageDialog(null, "Invalid cpu name.");

}

} else {

JOptionPane.showMessageDialog(null, "No cpu selected.");

}

} else {

JOptionPane.showMessageDialog(null, "Invalid data format!");

}

} else {

System.out.println("User clicked NO");

}

} else {

JOptionPane.showMessageDialog(cpuPanel, "Сan't update any record! Please select one!", "Error", row);

}

}

});

deleteCPUButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e)

{

int selectedRow = cpuTable.getSelectedRow();

if (selectedRow != -1) {

// create the popup window with yes/no options

int choice = JOptionPane.showConfirmDialog(cpuPanel, "Do you wish to continue? ", "Confirmation", JOptionPane.YES\_NO\_OPTION);

// handle the user's choice

if (choice == JOptionPane.YES\_OPTION)

{

String cpuModel = (String) cpuTableModel.getValueAt(selectedRow, 0);

try (Session session = getSession()) {

Transaction transaction = session.beginTransaction();

ClassCPU cpu = (ClassCPU) session.createQuery("FROM ClassCPU WHERE model = :model").setParameter("model", cpuModel).uniqueResult();

if (cpu != null) {

session.delete(cpu);

transaction.commit();

populateTables(); // Refresh the table data after deleting the cpu

updateAllDropBoxes();

JOptionPane.showMessageDialog(null, "cpu deleted successfully.");

} else {

JOptionPane.showMessageDialog(null, "cpu not found.");

}

} catch (Exception ex) {

ex.printStackTrace();

JOptionPane.showMessageDialog(null, "Failed to delete cpu: " + ex);

}

}

} else {

JOptionPane.showMessageDialog(null, "No cpu selected.");

}

}

});

pdfExportCPUButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e)

{

LOGGER.addHandler(fileHandler);

LOGGER.info("Trying to export data to PDF document");

try

{

JFileChooser fileChooser = new JFileChooser();

// Set default folder to current directory

fileChooser.setCurrentDirectory(new File("."));

// Set default file name

fileChooser.setSelectedFile(new File("exported\_CPUs.pdf"));

int result = fileChooser.showSaveDialog(null);

if (result == JFileChooser.APPROVE\_OPTION)

{

File selectedFile = fileChooser.getSelectedFile();

String fileName = selectedFile.getAbsolutePath();

// Append .pdf extension if necessary

if (!fileName.endsWith(".pdf"))

{

fileName += ".pdf";

}

Document document = new Document();

PdfWriter.getInstance(document, new FileOutputStream(fileName));

document.open();

PdfPTable pdfTable = new PdfPTable(cpuTable.getColumnCount());

// Create font for table headers

Font headerFont = FontFactory.getFont(FontFactory.HELVETICA\_BOLD, 10, BaseColor.BLACK);

String[] headersPdfExport = {"\nModel\n\n", "\nPrice", "\nCores", "\nThreads", "\nFrequency", "\nBrand", "\nSocket"};

// Set column headers

for (int i = 0; i < cpuTable.getColumnCount(); i++)

{

PdfPCell header = new PdfPCell(new Phrase(headersPdfExport[i], headerFont));

header.setBackgroundColor(BaseColor.ORANGE);

header.setBorderWidth(2);

header.setHorizontalAlignment(Element.ALIGN\_CENTER);

// Give more weight to the first row

pdfTable.addCell(header);

}

// Create font for table data

Font dataFont = FontFactory.getFont(FontFactory.HELVETICA, 10, BaseColor.BLACK);

// Set custom widths for each row

float[] columnWidths = {0.25f, 0.12f, 0.1f, 0.12f, 0.15f, 0.1f, 0.2f};

pdfTable.setWidths(columnWidths);

// Add table data

for (int i = 0; i < cpuTable.getRowCount(); i++)

{

for (int j = 0; j < cpuTable.getColumnCount(); j++)

{

PdfPCell data = new PdfPCell(new Phrase(cpuTable.getValueAt(i, j).toString(), dataFont));

if (i % 2 == 1)

{

data.setBackgroundColor(BaseColor.LIGHT\_GRAY);

}

else

{

data.setBackgroundColor(BaseColor.WHITE);

}

data.setBorderWidth(1);

data.setHorizontalAlignment(Element.ALIGN\_LEFT);

pdfTable.addCell(data);

}

}

document.add(pdfTable);

document.close();

JOptionPane.showMessageDialog(cpuPanel, "Exported table data to " + fileName);

}

}

catch (Exception ex)

{

ex.printStackTrace();

JOptionPane.showMessageDialog(cpuPanel, "Error exporting table data to PDF");

}

}

});

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

// \_\_\_\_\_\_ \_\_\_\_\_\_\_ \_\_ \_\_ \_\_ \_\_\_\_\_\_ \_\_\_\_\_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_ \_\_ \_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_ \_\_\_\_\_\_ //

// / \ / \ / | / | / | / | / \ / |/ |/ \ / |/ |/ \ / \ //

// /$$$$$$ |$$$$$$$ |$$ | $$ | $$ | $$$$$$/ /$$$$$$ |$$$$$$$$/ $$$$$$$$/ $$ \ $$ |$$$$$$$$/ $$$$$$$ |/$$$$$$ | //

// $$ | \_$$/ $$ |\_\_$$ |$$ | $$ | $$ | $$ | $$ \\_\_$$/ $$ | $$ |\_\_ $$$ \$$ |$$ |\_\_ $$ |\_\_$$ |$$ \\_\_$$/ //

// $$ |/ |$$ $$/ $$ | $$ | $$ | $$ | $$ \ $$ | $$ | $$$$ $$ |$$ | $$ $$< $$ \ //

// $$ |$$$$ |$$$$$$$/ $$ | $$ | $$ | $$ | $$$$$$ | $$ | $$$$$/ $$ $$ $$ |$$$$$/ $$$$$$$ | $$$$$$ | //

// $$ \\_\_$$ |$$ | $$ \\_\_$$ | $$ |\_\_\_\_\_ \_$$ |\_ / \\_\_$$ | $$ | $$ |\_\_\_\_\_ $$ |$$$$ |$$ |\_\_\_\_\_ $$ | $$ |/ \\_\_$$ | //

// $$ $$/ $$ | $$ $$/ $$ |/ $$ |$$ $$/ $$ | $$ |$$ | $$$ |$$ |$$ | $$ |$$ $$/ //

// $$$$$$/ $$/ $$$$$$/ $$$$$$$$/ $$$$$$/ $$$$$$/ $$/ $$$$$$$$/ $$/ $$/ $$$$$$$$/ $$/ $$/ $$$$$$/ //

// //

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

addGPUButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e)

{

if (gpuModelField.getText().isBlank())

{

JOptionPane.showMessageDialog(gpuPanel, "You must fill all text fields first!");

}

else

{

if (gpuModelField.getText().matches(modelRegex) && gpuPriceField.getText().matches(priceRegex) && gpuCoresField.getText().matches(coresRegex) && gpuMemoryField.getText().matches(memoryRegex) && gpuFrequencyField.getText().matches(frequencyRegex))

{

String model = gpuModelField.getText();

double price = Double.parseDouble(gpuPriceField.getText());

int cores = Integer.parseInt(gpuCoresField.getText());

int memory = Integer.parseInt(gpuMemoryField.getText());

int frequency = Integer.parseInt(gpuFrequencyField.getText());

String brand = gpuBrandComboBox.getSelectedItem().toString();

ClassBrand brandObj = (ClassBrand) session.createQuery("FROM ClassBrand WHERE name = :name").setParameter("name", brand).uniqueResult();

Object[] rowData = {model, price, cores, memory, frequency, brand};

ClassGPU gpu = new ClassGPU();

gpu.setModel(model);

gpu.setPrice(price);

gpu.setCores(cores);

gpu.setMemory(memory);

gpu.setFrequency(frequency);

gpu.setBrand(brandObj);

try (Session session = getSession())

{

Transaction transaction = session.beginTransaction();

session.save(gpu);

transaction.commit();

gpuTableModel.addRow(rowData);

JOptionPane.showMessageDialog(null, "gpu added successfully.");

updateAllDropBoxes();

}

catch (Exception ex)

{

ex.printStackTrace();

JOptionPane.showMessageDialog(null, "Failed to add gpu: " + ex);

}

} else {

JOptionPane.showMessageDialog(null, "Invalid data format!");

}

}

}

});

updateGPUButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e)

{

int row = gpuTable.getSelectedRow();

if (row != -1) {

// create the popup window with yes/no options

int choice = JOptionPane.showConfirmDialog(gpuPanel, "Do you wish to continue? ", "Confirmation", JOptionPane.YES\_NO\_OPTION);

// handle the user's choice

if (choice == JOptionPane.YES\_OPTION)

{

if (gpuModelField.getText().matches(modelRegex) && gpuPriceField.getText().matches(priceRegex) && gpuCoresField.getText().matches(coresRegex) && gpuMemoryField.getText().matches(memoryRegex) && gpuFrequencyField.getText().matches(frequencyRegex))

{

int selectedRow = gpuTable.getSelectedRow();

if (selectedRow != -1) {

String oldModelName = (String) gpuTableModel.getValueAt(selectedRow, 0);

String model = gpuModelField.getText();

double price = Double.parseDouble(gpuPriceField.getText());

int cores = Integer.parseInt(gpuCoresField.getText());

int memory = Integer.parseInt(gpuMemoryField.getText());

int frequency = Integer.parseInt(gpuFrequencyField.getText());

String brand = gpuBrandComboBox.getSelectedItem().toString();

ClassBrand brandObj = (ClassBrand) session.createQuery("FROM ClassBrand WHERE name = :name").setParameter("name", brand).uniqueResult();

if (model != null && !model.isEmpty()) {

try (Session session = getSession()) {

Transaction transaction = session.beginTransaction();

ClassGPU gpu = (ClassGPU) session.createQuery("FROM ClassGPU WHERE model = :model").setParameter("model", oldModelName).uniqueResult();

if (gpu != null) {

gpu.setModel(model);

gpu.setPrice(price);

gpu.setCores(cores);

gpu.setMemory(memory);

gpu.setFrequency(frequency);

gpu.setBrand(brandObj);

session.update(gpu);

populateTables();

transaction.commit();

populateTables();

updateAllDropBoxes();

JOptionPane.showMessageDialog(null, "gpu updated successfully.");

} else {

JOptionPane.showMessageDialog(null, "gpu not found.");

}

} catch (Exception ex) {

ex.printStackTrace();

JOptionPane.showMessageDialog(null, "Failed to update gpu: " + ex);

}

} else {

JOptionPane.showMessageDialog(null, "Invalid gpu name.");

}

} else {

JOptionPane.showMessageDialog(null, "No gpu selected.");

}

} else {

JOptionPane.showMessageDialog(null, "Invalid data format!");

}

} else {

System.out.println("User clicked NO");

}

} else {

JOptionPane.showMessageDialog(gpuPanel, "Сan't update any record! Please select one!", "Error", row);

}

}

});

deleteGPUButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e)

{

int selectedRow = gpuTable.getSelectedRow();

if (selectedRow != -1) {

// create the popup window with yes/no options

int choice = JOptionPane.showConfirmDialog(gpuPanel, "Do you wish to continue? ", "Confirmation", JOptionPane.YES\_NO\_OPTION);

// handle the user's choice

if (choice == JOptionPane.YES\_OPTION)

{

String gpuModel = (String) gpuTableModel.getValueAt(selectedRow, 0);

try (Session session = getSession()) {

Transaction transaction = session.beginTransaction();

ClassGPU gpu = (ClassGPU) session.createQuery("FROM ClassGPU WHERE model = :model").setParameter("model", gpuModel).uniqueResult();

if (gpu != null) {

session.delete(gpu);

transaction.commit();

populateTables(); // Refresh the table data after deleting the gpu

updateAllDropBoxes();

JOptionPane.showMessageDialog(null, "gpu deleted successfully.");

} else {

JOptionPane.showMessageDialog(null, "gpu not found.");

}

} catch (Exception ex) {

ex.printStackTrace();

JOptionPane.showMessageDialog(null, "Failed to delete gpu: " + ex);

}

}

} else {

JOptionPane.showMessageDialog(null, "No gpu selected.");

}

}

});

pdfExportGPUButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e)

{

LOGGER.addHandler(fileHandler);

LOGGER.info("Trying to export data to PDF document");

try

{

JFileChooser fileChooser = new JFileChooser();

// Set default folder to current directory

fileChooser.setCurrentDirectory(new File("."));

// Set default file name

fileChooser.setSelectedFile(new File("exported\_GPUs.pdf"));

int result = fileChooser.showSaveDialog(null);

if (result == JFileChooser.APPROVE\_OPTION)

{

File selectedFile = fileChooser.getSelectedFile();

String fileName = selectedFile.getAbsolutePath();

// Append .pdf extension if necessary

if (!fileName.endsWith(".pdf"))

{

fileName += ".pdf";

}

Document document = new Document();

PdfWriter.getInstance(document, new FileOutputStream(fileName));

document.open();

PdfPTable pdfTable = new PdfPTable(gpuTable.getColumnCount());

// Create font for table headers

Font headerFont = FontFactory.getFont(FontFactory.HELVETICA\_BOLD, 10, BaseColor.BLACK);

String[] headersPdfExport = {"\nModel\n\n", "\nPrice", "\nCores", "\nThreads", "\nFrequency", "\nBrand"};

// Set column headers

for (int i = 0; i < gpuTable.getColumnCount(); i++)

{

PdfPCell header = new PdfPCell(new Phrase(headersPdfExport[i], headerFont));

header.setBackgroundColor(BaseColor.ORANGE);

header.setBorderWidth(2);

header.setHorizontalAlignment(Element.ALIGN\_CENTER);

// Give more weight to the first row

pdfTable.addCell(header);

}

// Create font for table data

Font dataFont = FontFactory.getFont(FontFactory.HELVETICA, 10, BaseColor.BLACK);

// Set custom widths for each row

float[] columnWidths = {0.25f, 0.12f, 0.1f, 0.12f, 0.15f, 0.1f};

pdfTable.setWidths(columnWidths);

// Add table data

for (int i = 0; i < gpuTable.getRowCount(); i++)

{

for (int j = 0; j < gpuTable.getColumnCount(); j++)

{

PdfPCell data = new PdfPCell(new Phrase(gpuTable.getValueAt(i, j).toString(), dataFont));

if (i % 2 == 1)

{

data.setBackgroundColor(BaseColor.LIGHT\_GRAY);

}

else

{

data.setBackgroundColor(BaseColor.WHITE);

}

data.setBorderWidth(1);

data.setHorizontalAlignment(Element.ALIGN\_LEFT);

pdfTable.addCell(data);

}

}

document.add(pdfTable);

document.close();

JOptionPane.showMessageDialog(gpuPanel, "Exported table data to " + fileName);

}

}

catch (Exception ex)

{

ex.printStackTrace();

JOptionPane.showMessageDialog(gpuPanel, "Error exporting table data to PDF");

}

}

});

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

// \_\_\_\_\_\_\_ \_\_\_\_\_\_ \_\_\_\_\_\_\_ \_\_ \_\_\_\_\_\_ \_\_\_\_\_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_ \_\_ \_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_ \_\_\_\_\_\_ //

// / \ / \ / \ / | / | / \ / |/ |/ \ / |/ |/ \ / \ //

// $$$$$$$ |/$$$$$$ |$$$$$$$ | $$ | $$$$$$/ /$$$$$$ |$$$$$$$$/ $$$$$$$$/ $$ \ $$ |$$$$$$$$/ $$$$$$$ |/$$$$$$ | //

// $$ |\_\_$$ |$$ | $$/ $$ |\_\_$$ | $$ | $$ | $$ \\_\_$$/ $$ | $$ |\_\_ $$$ \$$ |$$ |\_\_ $$ |\_\_$$ |$$ \\_\_$$/ //

// $$ $$/ $$ | $$ $$< $$ | $$ | $$ \ $$ | $$ | $$$$ $$ |$$ | $$ $$< $$ \ //

// $$$$$$$/ $$ | \_\_ $$$$$$$ | $$ | $$ | $$$$$$ | $$ | $$$$$/ $$ $$ $$ |$$$$$/ $$$$$$$ | $$$$$$ | //

// $$ | $$ \\_\_/ |$$ |\_\_$$ | $$ |\_\_\_\_\_ \_$$ |\_ / \\_\_$$ | $$ | $$ |\_\_\_\_\_ $$ |$$$$ |$$ |\_\_\_\_\_ $$ | $$ |/ \\_\_$$ | //

// $$ | $$ $$/ $$ $$/ $$ |/ $$ |$$ $$/ $$ | $$ |$$ | $$$ |$$ |$$ | $$ |$$ $$/ //

// $$/ $$$$$$/ $$$$$$$/ $$$$$$$$/ $$$$$$/ $$$$$$/ $$/ $$$$$$$$/ $$/ $$/ $$$$$$$$/ $$/ $$/ $$$$$$/ //

// //

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

addPCBButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e)

{

if (pcbModelField.getText().isBlank())

{

JOptionPane.showMessageDialog(pcbPanel, "You must fill all text fields first!");

}

else

{

if (pcbModelField.getText().matches(modelRegex) && pcbPriceField.getText().matches(priceRegex))

{

String model = pcbModelField.getText();

double price = Double.parseDouble(pcbPriceField.getText());

String brand = pcbBrandComboBox.getSelectedItem().toString();

String socket = pcbSocketComboBox.getSelectedItem().toString();

String chipset= pcbChipsetComboBox.getSelectedItem().toString();

ClassBrand brandObj = (ClassBrand) session.createQuery("FROM ClassBrand WHERE name = :name").setParameter("name", brand).uniqueResult();

ClassSocket socketObj = (ClassSocket) session.createQuery("FROM ClassSocket WHERE name = :name").setParameter("name", socket).uniqueResult();

ClassChipset chipsetObj = (ClassChipset) session.createQuery("FROM ClassChipset WHERE name = :name").setParameter("name", chipset).uniqueResult();

Object[] rowData = {model, price, brand, socket, chipset};

ClassPCB pcb = new ClassPCB();

pcb.setModel(model);

pcb.setPrice(price);

pcb.setBrand(brandObj);

pcb.setSocket(socketObj);

pcb.setChipset(chipsetObj);

try (Session session = getSession())

{

Transaction transaction = session.beginTransaction();

session.save(pcb);

transaction.commit();

pcbTableModel.addRow(rowData);

JOptionPane.showMessageDialog(null, "pcb added successfully.");

updateAllDropBoxes();

}

catch (Exception ex)

{

ex.printStackTrace();

JOptionPane.showMessageDialog(null, "Failed to add pcb: " + ex);

}

} else {

JOptionPane.showMessageDialog(null, "Invalid data format!");

}

}

}

});

updatePCBButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e)

{

int row = pcbTable.getSelectedRow();

if (row != -1) {

// create the popup window with yes/no options

int choice = JOptionPane.showConfirmDialog(pcbPanel, "Do you wish to continue? ", "Confirmation", JOptionPane.YES\_NO\_OPTION);

// handle the user's choice

if (choice == JOptionPane.YES\_OPTION)

{

if (pcbModelField.getText().matches(modelRegex) && pcbPriceField.getText().matches(priceRegex))

{

int selectedRow = pcbTable.getSelectedRow();

if (selectedRow != -1) {

String oldModelName = (String) pcbTableModel.getValueAt(selectedRow, 0);

String model = pcbModelField.getText();

double price = Double.parseDouble(pcbPriceField.getText());

String brand = pcbBrandComboBox.getSelectedItem().toString();

String socket = pcbSocketComboBox.getSelectedItem().toString();

String chipset= pcbChipsetComboBox.getSelectedItem().toString();

ClassBrand brandObj = (ClassBrand) session.createQuery("FROM ClassBrand WHERE name = :name").setParameter("name", brand).uniqueResult();

ClassSocket socketObj = (ClassSocket) session.createQuery("FROM ClassSocket WHERE name = :name").setParameter("name", socket).uniqueResult();

ClassChipset chipsetObj = (ClassChipset) session.createQuery("FROM ClassChipset WHERE name = :name").setParameter("name", chipset).uniqueResult();

if (model != null && !model.isEmpty()) {

try (Session session = getSession()) {

Transaction transaction = session.beginTransaction();

ClassPCB pcb = (ClassPCB) session.createQuery("FROM ClassPCB WHERE model = :model").setParameter("model", oldModelName).uniqueResult();

if (pcb != null) {

pcb.setModel(model);

pcb.setPrice(price);

pcb.setBrand(brandObj);

pcb.setSocket(socketObj);

pcb.setChipset(chipsetObj);

session.update(pcb);

populateTables();

transaction.commit();

populateTables();

updateAllDropBoxes();

JOptionPane.showMessageDialog(null, "pcb updated successfully.");

} else {

JOptionPane.showMessageDialog(null, "pcb not found.");

}

} catch (Exception ex) {

ex.printStackTrace();

JOptionPane.showMessageDialog(null, "Failed to update pcb: " + ex);

}

} else {

JOptionPane.showMessageDialog(null, "Invalid pcb name.");

}

} else {

JOptionPane.showMessageDialog(null, "No pcb selected.");

}

} else {

JOptionPane.showMessageDialog(null, "Invalid data format!");

}

} else {

System.out.println("User clicked NO");

}

} else {

JOptionPane.showMessageDialog(pcbPanel, "Сan't update any record! Please select one!", "Error", row);

}

}

});

deletePCBButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e)

{

int selectedRow = pcbTable.getSelectedRow();

if (selectedRow != -1) {

// create the popup window with yes/no options

int choice = JOptionPane.showConfirmDialog(pcbPanel, "Do you wish to continue? ", "Confirmation", JOptionPane.YES\_NO\_OPTION);

// handle the user's choice

if (choice == JOptionPane.YES\_OPTION)

{

String pcbModel = (String) pcbTableModel.getValueAt(selectedRow, 0);

try (Session session = getSession()) {

Transaction transaction = session.beginTransaction();

ClassPCB pcb = (ClassPCB) session.createQuery("FROM ClassPCB WHERE model = :model").setParameter("model", pcbModel).uniqueResult();

if (pcb != null) {

session.delete(pcb);

transaction.commit();

populateTables(); // Refresh the table data after deleting the pcb

updateAllDropBoxes();

JOptionPane.showMessageDialog(null, "pcb deleted successfully.");

} else {

JOptionPane.showMessageDialog(null, "pcb not found.");

}

} catch (Exception ex) {

ex.printStackTrace();

JOptionPane.showMessageDialog(null, "Failed to delete pcb: " + ex);

}

}

} else {

JOptionPane.showMessageDialog(null, "No pcb selected.");

}

}

});

pdfExportPCBButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e)

{

LOGGER.addHandler(fileHandler);

LOGGER.info("Trying to export data to PDF document");

try

{

JFileChooser fileChooser = new JFileChooser();

// Set default folder to current directorys

fileChooser.setCurrentDirectory(new File("."));

// Set default file name

fileChooser.setSelectedFile(new File("exported\_PCBs.pdf"));

int result = fileChooser.showSaveDialog(null);

if (result == JFileChooser.APPROVE\_OPTION)

{

File selectedFile = fileChooser.getSelectedFile();

String fileName = selectedFile.getAbsolutePath();

// Append .pdf extension if necessary

if (!fileName.endsWith(".pdf"))

{

fileName += ".pdf";

}

Document document = new Document();

PdfWriter.getInstance(document, new FileOutputStream(fileName));

document.open();

PdfPTable pdfTable = new PdfPTable(pcbTable.getColumnCount());

// Create font for table headers

Font headerFont = FontFactory.getFont(FontFactory.HELVETICA\_BOLD, 10, BaseColor.BLACK);

String[] headersPdfExport = {"\nModel\n\n", "\nPrice", "\nBrand", "\nSocket", "\nChipset"};

// Set column headers

for (int i = 0; i < pcbTable.getColumnCount(); i++)

{

PdfPCell header = new PdfPCell(new Phrase(headersPdfExport[i], headerFont));

header.setBackgroundColor(BaseColor.ORANGE);

header.setBorderWidth(2);

header.setHorizontalAlignment(Element.ALIGN\_CENTER);

// Give more weight to the first row

pdfTable.addCell(header);

}

// Create font for table data

Font dataFont = FontFactory.getFont(FontFactory.HELVETICA, 10, BaseColor.BLACK);

// Set custom widths for each row

float[] columnWidths = {0.26f, 0.13f, 0.11f, 0.13f, 0.16f};

pdfTable.setWidths(columnWidths);

// Add table data

for (int i = 0; i < pcbTable.getRowCount(); i++)

{

for (int j = 0; j < pcbTable.getColumnCount(); j++)

{

PdfPCell data = new PdfPCell(new Phrase(pcbTable.getValueAt(i, j).toString(), dataFont));

if (i % 2 == 1)

{

data.setBackgroundColor(BaseColor.LIGHT\_GRAY);

}

else

{

data.setBackgroundColor(BaseColor.WHITE);

}

data.setBorderWidth(1);

data.setHorizontalAlignment(Element.ALIGN\_LEFT);

pdfTable.addCell(data);

}

}

document.add(pdfTable);

document.close();

JOptionPane.showMessageDialog(pcbPanel, "Exported table data to " + fileName);

}

}

catch (Exception ex)

{

ex.printStackTrace();

JOptionPane.showMessageDialog(pcbPanel, "Error exporting table data to PDF");

}

}

});

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

// \_\_\_\_\_\_\_ \_\_\_\_\_\_\_ \_\_\_\_\_\_ \_\_ \_\_ \_\_\_\_\_\_\_ \_\_ \_\_\_\_\_\_ \_\_\_\_\_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_ \_\_ \_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_ \_\_\_\_\_\_ //

// / \ / \ / \ / \ / |/ \ / | / | / \ / |/ |/ \ / |/ |/ \ / \ //

// $$$$$$$ |$$$$$$$ |/$$$$$$ |$$ \ $$ |$$$$$$$ | $$ | $$$$$$/ /$$$$$$ |$$$$$$$$/ $$$$$$$$/ $$ \ $$ |$$$$$$$$/ $$$$$$$ |/$$$$$$ | //

// $$ |\_\_$$ |$$ |\_\_$$ |$$ |\_\_$$ |$$$ \$$ |$$ | $$ | $$ | $$ | $$ \\_\_$$/ $$ | $$ |\_\_ $$$ \$$ |$$ |\_\_ $$ |\_\_$$ |$$ \\_\_$$/ //

// $$ $$< $$ $$< $$ $$ |$$$$ $$ |$$ | $$ | $$ | $$ | $$ \ $$ | $$ | $$$$ $$ |$$ | $$ $$< $$ \ //

// $$$$$$$ |$$$$$$$ |$$$$$$$$ |$$ $$ $$ |$$ | $$ | $$ | $$ | $$$$$$ | $$ | $$$$$/ $$ $$ $$ |$$$$$/ $$$$$$$ | $$$$$$ | //

// $$ |\_\_$$ |$$ | $$ |$$ | $$ |$$ |$$$$ |$$ |\_\_$$ | $$ |\_\_\_\_\_ \_$$ |\_ / \\_\_$$ | $$ | $$ |\_\_\_\_\_ $$ |$$$$ |$$ |\_\_\_\_\_ $$ | $$ |/ \\_\_$$ | //

// $$ $$/ $$ | $$ |$$ | $$ |$$ | $$$ |$$ $$/ $$ |/ $$ |$$ $$/ $$ | $$ |$$ | $$$ |$$ |$$ | $$ |$$ $$/ //

// $$$$$$$/ $$/ $$/ $$/ $$/ $$/ $$/ $$$$$$$/ $$$$$$$$/ $$$$$$/ $$$$$$/ $$/ $$$$$$$$/ $$/ $$/ $$$$$$$$/ $$/ $$/ $$$$$$/ //

// //

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

addBrandButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e)

{

if (brandNameField.getText().isBlank())

{

JOptionPane.showMessageDialog(brandPanel, "You must fill all text fields first!");

}

else

{

if (brandNameField.getText().matches(brandNameRegex))

{

String name = brandNameField.getText();

Object[] rowData = {name};

ClassBrand brand = new ClassBrand();

brand.setName(name);

try (Session session = getSession())

{

Transaction transaction = session.beginTransaction();

session.save(brand);

transaction.commit();

brandTableModel.addRow(rowData);

JOptionPane.showMessageDialog(null, "Brand added successfully.");

updateAllDropBoxes();

}

catch (Exception ex)

{

ex.printStackTrace();

JOptionPane.showMessageDialog(null, "Failed to add brand: " + ex);

}

} else {

JOptionPane.showMessageDialog(null, "Invalid data format!");

}

}

}

});

updateBrandButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

int row = brandTable.getSelectedRow();

if (row != -1) {

// create the popup window with yes/no options

int choice = JOptionPane.showConfirmDialog(brandPanel, "Do you wish to continue? ", "Confirmation", JOptionPane.YES\_NO\_OPTION);

// handle the user's choice

if (choice == JOptionPane.YES\_OPTION)

{

if (brandNameField.getText().matches(brandNameRegex))

{

int selectedRow = brandTable.getSelectedRow();

if (selectedRow != -1) {

String oldBrandName = (String) brandTableModel.getValueAt(selectedRow, 0);

String newBrandName = brandNameField.getText();

if (newBrandName != null && !newBrandName.isEmpty()) {

try (Session session = getSession()) {

Transaction transaction = session.beginTransaction();

ClassBrand brand = (ClassBrand) session.createQuery("FROM ClassBrand WHERE name = :name").setParameter("name", oldBrandName).uniqueResult();

if (brand != null) {

brand.setName(newBrandName);

session.update(brand);

populateTables();

transaction.commit();

populateTables();

updateAllDropBoxes();

JOptionPane.showMessageDialog(null, "Brand updated successfully.");

} else {

JOptionPane.showMessageDialog(null, "Brand not found.");

}

} catch (Exception ex) {

ex.printStackTrace();

JOptionPane.showMessageDialog(null, "Failed to update brand: " + ex);

}

} else {

JOptionPane.showMessageDialog(null, "Invalid brand name.");

}

} else {

JOptionPane.showMessageDialog(null, "No brand selected.");

}

} else {

JOptionPane.showMessageDialog(null, "Invalid data format!");

}

} else {

System.out.println("User clicked NO");

}

} else {

JOptionPane.showMessageDialog(brandPanel, "Сan't update any record! Please select one!", "Error", row);

}

}

});

deleteBrandButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e)

{

int selectedRow = brandTable.getSelectedRow();

if (selectedRow != -1) {

// create the popup window with yes/no options

int choice = JOptionPane.showConfirmDialog(brandPanel, "Do you wish to continue? ", "Confirmation", JOptionPane.YES\_NO\_OPTION);

// handle the user's choice

if (choice == JOptionPane.YES\_OPTION)

{

String brandName = (String) brandTableModel.getValueAt(selectedRow, 0);

try (Session session = getSession()) {

Transaction transaction = session.beginTransaction();

ClassBrand brand = (ClassBrand) session.createQuery("FROM ClassBrand WHERE name = :name").setParameter("name", brandName).uniqueResult();

if (brand != null) {

session.delete(brand);

transaction.commit();

populateTables(); // Refresh the table data after deleting the brand

updateAllDropBoxes();

JOptionPane.showMessageDialog(null, "Brand deleted successfully.");

} else {

JOptionPane.showMessageDialog(null, "Brand not found.");

}

} catch (Exception ex) {

ex.printStackTrace();

JOptionPane.showMessageDialog(null, "Failed to delete brand: " + ex);

}

}

} else {

JOptionPane.showMessageDialog(null, "No brand selected.");

}

}

});

pdfExportBrandButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e)

{

LOGGER.addHandler(fileHandler);

LOGGER.info("Trying to export data to PDF document");

try

{

JFileChooser fileChooser = new JFileChooser();

// Set default folder to current directory

fileChooser.setCurrentDirectory(new File("."));

// Set default file name

fileChooser.setSelectedFile(new File("exported\_Brands.pdf"));

int result = fileChooser.showSaveDialog(null);

if (result == JFileChooser.APPROVE\_OPTION)

{

File selectedFile = fileChooser.getSelectedFile();

String fileName = selectedFile.getAbsolutePath();

// Append .pdf extension if necessary

if (!fileName.endsWith(".pdf"))

{

fileName += ".pdf";

}

Document document = new Document();

PdfWriter.getInstance(document, new FileOutputStream(fileName));

document.open();

PdfPTable pdfTable = new PdfPTable(brandTable.getColumnCount());

// Create font for table headers

Font headerFont = FontFactory.getFont(FontFactory.HELVETICA\_BOLD, 10, BaseColor.BLACK);

String[] headersPdfExport = {"\nBrand name\n\n"};

// Set column headers

for (int i = 0; i < brandTable.getColumnCount(); i++)

{

PdfPCell header = new PdfPCell(new Phrase(headersPdfExport[i], headerFont));

header.setBackgroundColor(BaseColor.ORANGE);

header.setBorderWidth(2);

header.setHorizontalAlignment(Element.ALIGN\_CENTER);

// Give more weight to the first row

pdfTable.addCell(header);

}

// Create font for table data

Font dataFont = FontFactory.getFont(FontFactory.HELVETICA, 10, BaseColor.BLACK);

// Set custom widths for each row

float[] columnWidths = {1f};

pdfTable.setWidths(columnWidths);

// Add table data

for (int i = 0; i < brandTable.getRowCount(); i++)

{

for (int j = 0; j < brandTable.getColumnCount(); j++)

{

PdfPCell data = new PdfPCell(new Phrase(brandTable.getValueAt(i, j).toString(), dataFont));

if (i % 2 == 1)

{

data.setBackgroundColor(BaseColor.LIGHT\_GRAY);

}

else

{

data.setBackgroundColor(BaseColor.WHITE);

}

data.setBorderWidth(1);

data.setHorizontalAlignment(Element.ALIGN\_LEFT);

pdfTable.addCell(data);

}

}

document.add(pdfTable);

document.close();

JOptionPane.showMessageDialog(brandPanel, "Exported table data to " + fileName);

}

}

catch (Exception ex)

{

ex.printStackTrace();

JOptionPane.showMessageDialog(brandPanel, "Error exporting table data to PDF");

}

}

});

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

// \_\_\_\_\_\_ \_\_\_\_\_\_ \_\_\_\_\_\_ \_\_ \_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_ \_\_ \_\_\_\_\_\_ \_\_\_\_\_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_ \_\_ \_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_ \_\_\_\_\_\_ //

// / \ / \ / \ / | / |/ |/ | / | / | / \ / |/ |/ \ / |/ |/ \ / \ //

// /$$$$$$ |/$$$$$$ |/$$$$$$ |$$ | /$$/ $$$$$$$$/ $$$$$$$$/ $$ | $$$$$$/ /$$$$$$ |$$$$$$$$/ $$$$$$$$/ $$ \ $$ |$$$$$$$$/ $$$$$$$ |/$$$$$$ | //

// $$ \\_\_$$/ $$ | $$ |$$ | $$/ $$ |/$$/ $$ |\_\_ $$ | $$ | $$ | $$ \\_\_$$/ $$ | $$ |\_\_ $$$ \$$ |$$ |\_\_ $$ |\_\_$$ |$$ \\_\_$$/ //

// $$ \ $$ | $$ |$$ | $$ $$< $$ | $$ | $$ | $$ | $$ \ $$ | $$ | $$$$ $$ |$$ | $$ $$< $$ \ //

// $$$$$$ |$$ | $$ |$$ | \_\_ $$$$$ \ $$$$$/ $$ | $$ | $$ | $$$$$$ | $$ | $$$$$/ $$ $$ $$ |$$$$$/ $$$$$$$ | $$$$$$ | //

// / \\_\_$$ |$$ \\_\_$$ |$$ \\_\_/ |$$ |$$ \ $$ |\_\_\_\_\_ $$ | $$ |\_\_\_\_\_ \_$$ |\_ / \\_\_$$ | $$ | $$ |\_\_\_\_\_ $$ |$$$$ |$$ |\_\_\_\_\_ $$ | $$ |/ \\_\_$$ | //

// $$ $$/ $$ $$/ $$ $$/ $$ | $$ |$$ | $$ | $$ |/ $$ |$$ $$/ $$ | $$ |$$ | $$$ |$$ |$$ | $$ |$$ $$/ //

// $$$$$$/ $$$$$$/ $$$$$$/ $$/ $$/ $$$$$$$$/ $$/ $$$$$$$$/ $$$$$$/ $$$$$$/ $$/ $$$$$$$$/ $$/ $$/ $$$$$$$$/ $$/ $$/ $$$$$$/ //

// //

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

addSocketButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e)

{

if (socketNameField.getText().isBlank())

{

JOptionPane.showMessageDialog(null, "You must fill all text fields first!");

}

else

{

if (socketNameField.getText().matches(socketNameRegex))

{

String name = socketNameField.getText();

Object[] rowData = {name};

ClassSocket socket = new ClassSocket();

socket.setName(name);

try (Session session = getSession())

{

Transaction transaction = session.beginTransaction();

session.save(socket);

transaction.commit();

socketTableModel.addRow(rowData);

JOptionPane.showMessageDialog(null, "Socket added successfully.");

updateAllDropBoxes();

}

catch (Exception ex)

{

ex.printStackTrace();

JOptionPane.showMessageDialog(null, "Failed to add socket: " + ex);

}

} else {

JOptionPane.showMessageDialog(null, "Invalid data format!");

}

}

}

});

updateSocketButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e)

{

int row = socketTable.getSelectedRow();

if (row != -1) {

// create the popup window with yes/no options

int choice = JOptionPane.showConfirmDialog(socketPanel, "Do you wish to continue? ", "Confirmation", JOptionPane.YES\_NO\_OPTION);

// handle the user's choice

if (choice == JOptionPane.YES\_OPTION)

{

if (socketNameField.getText().matches(socketNameRegex))

{

int selectedRow = socketTable.getSelectedRow();

if (selectedRow != -1) {

String oldSocketName = (String) socketTableModel.getValueAt(selectedRow, 0);

String newSocketName = socketNameField.getText();

if (newSocketName != null && !newSocketName.isEmpty()) {

try (Session session = getSession()) {

Transaction transaction = session.beginTransaction();

ClassSocket socket = (ClassSocket) session.createQuery("FROM ClassSocket WHERE name = :name").setParameter("name", oldSocketName).uniqueResult();

if (socket != null) {

socket.setName(newSocketName);

session.update(socket);

populateTables();

transaction.commit();

populateTables();

updateAllDropBoxes();

JOptionPane.showMessageDialog(null, "socket updated successfully.");

} else {

JOptionPane.showMessageDialog(null, "socket not found.");

}

} catch (Exception ex) {

ex.printStackTrace();

JOptionPane.showMessageDialog(null, "Failed to update socket: " + ex);

}

} else {

JOptionPane.showMessageDialog(null, "Invalid socket name.");

}

} else {

JOptionPane.showMessageDialog(null, "No socket selected.");

}

} else {

JOptionPane.showMessageDialog(null, "Invalid data format!");

}

} else {

System.out.println("User clicked NO");

}

} else {

JOptionPane.showMessageDialog(socketPanel, "Сan't update any record! Please select one!", "Error", row);

}

}

});

deleteSocketButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e)

{

int selectedRow = socketTable.getSelectedRow();

if (selectedRow != -1) {

// create the popup window with yes/no options

int choice = JOptionPane.showConfirmDialog(socketPanel, "Do you wish to continue? ", "Confirmation", JOptionPane.YES\_NO\_OPTION);

// handle the user's choice

if (choice == JOptionPane.YES\_OPTION)

{

String socketName = (String) socketTableModel.getValueAt(selectedRow, 0);

try (Session session = getSession()) {

Transaction transaction = session.beginTransaction();

ClassSocket socket = (ClassSocket) session.createQuery("FROM ClassSocket WHERE name = :name").setParameter("name", socketName).uniqueResult();

if (socket != null) {

session.delete(socket);

transaction.commit();

populateTables(); // Refresh the table data after deleting the brand

updateAllDropBoxes();

JOptionPane.showMessageDialog(null, "socket deleted successfully.");

} else {

JOptionPane.showMessageDialog(null, "socket not found.");

}

} catch (Exception ex) {

ex.printStackTrace();

JOptionPane.showMessageDialog(null, "Failed to delete socket: " + ex);

}

}

} else {

JOptionPane.showMessageDialog(null, "No socket selected.");

}

}

});

pdfExportSocketButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e)

{

LOGGER.addHandler(fileHandler);

LOGGER.info("Trying to export data to PDF document");

try

{

JFileChooser fileChooser = new JFileChooser();

// Set default folder to current directory

fileChooser.setCurrentDirectory(new File("."));

// Set default file name

fileChooser.setSelectedFile(new File("exported\_Sockets.pdf"));

int result = fileChooser.showSaveDialog(null);

if (result == JFileChooser.APPROVE\_OPTION)

{

File selectedFile = fileChooser.getSelectedFile();

String fileName = selectedFile.getAbsolutePath();

// Append .pdf extension if necessary

if (!fileName.endsWith(".pdf"))

{

fileName += ".pdf";

}

Document document = new Document();

PdfWriter.getInstance(document, new FileOutputStream(fileName));

document.open();

PdfPTable pdfTable = new PdfPTable(socketTable.getColumnCount());

// Create font for table headers

Font headerFont = FontFactory.getFont(FontFactory.HELVETICA\_BOLD, 10, BaseColor.BLACK);

String[] headersPdfExport = {"\nSocket name\n\n"};

// Set column headers

for (int i = 0; i < socketTable.getColumnCount(); i++)

{

PdfPCell header = new PdfPCell(new Phrase(headersPdfExport[i], headerFont));

header.setBackgroundColor(BaseColor.ORANGE);

header.setBorderWidth(2);

header.setHorizontalAlignment(Element.ALIGN\_CENTER);

// Give more weight to the first row

pdfTable.addCell(header);

}

// Create font for table data

Font dataFont = FontFactory.getFont(FontFactory.HELVETICA, 10, BaseColor.BLACK);

// Set custom widths for each row

float[] columnWidths = {1f};

pdfTable.setWidths(columnWidths);

// Add table data

for (int i = 0; i < socketTable.getRowCount(); i++)

{

for (int j = 0; j < socketTable.getColumnCount(); j++)

{

PdfPCell data = new PdfPCell(new Phrase(socketTable.getValueAt(i, j).toString(), dataFont));

if (i % 2 == 1)

{

data.setBackgroundColor(BaseColor.LIGHT\_GRAY);

}

else

{

data.setBackgroundColor(BaseColor.WHITE);

}

data.setBorderWidth(1);

data.setHorizontalAlignment(Element.ALIGN\_LEFT);

pdfTable.addCell(data);

}

}

document.add(pdfTable);

document.close();

JOptionPane.showMessageDialog(socketPanel, "Exported table data to " + fileName);

}

}

catch (Exception ex)

{

ex.printStackTrace();

JOptionPane.showMessageDialog(socketPanel, "Error exporting table data to PDF");

}

}

});

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

// \_\_\_\_\_\_ \_\_ \_\_ \_\_\_\_\_\_ \_\_\_\_\_\_\_ \_\_\_\_\_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_ \_\_ \_\_\_\_\_\_ \_\_\_\_\_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_ \_\_ \_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_ \_\_\_\_\_\_ //

// / \ / | / |/ |/ \ / \ / |/ | / | / | / \ / |/ |/ \ / |/ |/ \ / \ //

// /$$$$$$ |$$ | $$ |$$$$$$/ $$$$$$$ |/$$$$$$ |$$$$$$$$/ $$$$$$$$/ $$ | $$$$$$/ /$$$$$$ |$$$$$$$$/ $$$$$$$$/ $$ \ $$ |$$$$$$$$/ $$$$$$$ |/$$$$$$ | //

// $$ | $$/ $$ |\_\_$$ | $$ | $$ |\_\_$$ |$$ \\_\_$$/ $$ |\_\_ $$ | $$ | $$ | $$ \\_\_$$/ $$ | $$ |\_\_ $$$ \$$ |$$ |\_\_ $$ |\_\_$$ |$$ \\_\_$$/ //

// $$ | $$ $$ | $$ | $$ $$/ $$ \ $$ | $$ | $$ | $$ | $$ \ $$ | $$ | $$$$ $$ |$$ | $$ $$< $$ \ //

// $$ | \_\_ $$$$$$$$ | $$ | $$$$$$$/ $$$$$$ |$$$$$/ $$ | $$ | $$ | $$$$$$ | $$ | $$$$$/ $$ $$ $$ |$$$$$/ $$$$$$$ | $$$$$$ | //

// $$ \\_\_/ |$$ | $$ | \_$$ |\_ $$ | / \\_\_$$ |$$ |\_\_\_\_\_ $$ | $$ |\_\_\_\_\_ \_$$ |\_ / \\_\_$$ | $$ | $$ |\_\_\_\_\_ $$ |$$$$ |$$ |\_\_\_\_\_ $$ | $$ |/ \\_\_$$ | //

// $$ $$/ $$ | $$ |/ $$ |$$ | $$ $$/ $$ | $$ | $$ |/ $$ |$$ $$/ $$ | $$ |$$ | $$$ |$$ |$$ | $$ |$$ $$/ //

// $$$$$$/ $$/ $$/ $$$$$$/ $$/ $$$$$$/ $$$$$$$$/ $$/ $$$$$$$$/ $$$$$$/ $$$$$$/ $$/ $$$$$$$$/ $$/ $$/ $$$$$$$$/ $$/ $$/ $$$$$$/ //

// //

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

addChipsetButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e)

{

if (chipsetNameField.getText().isBlank())

{

JOptionPane.showMessageDialog(null, "You must fill all text fields first!");

}

else

{

if (chipsetNameField.getText().matches(chipsetNameRegex))

{

String name = chipsetNameField.getText();

Object[] rowData = {name};

ClassChipset chipset = new ClassChipset();

chipset.setName(name);

try (Session session = getSession())

{

Transaction transaction = session.beginTransaction();

session.save(chipset);

transaction.commit();

chipsetTableModel.addRow(rowData);

JOptionPane.showMessageDialog(null, "chipset added successfully.");

updateAllDropBoxes();

}

catch (Exception ex)

{

ex.printStackTrace();

JOptionPane.showMessageDialog(null, "Failed to add chipset: " + ex);

}

} else {

JOptionPane.showMessageDialog(null, "Invalid data format!");

}

}

}

});

updateChipsetButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e)

{

int row = chipsetTable.getSelectedRow();

if (row != -1) {

// create the popup window with yes/no options

int choice = JOptionPane.showConfirmDialog(chipsetPanel, "Do you wish to continue? ", "Confirmation", JOptionPane.YES\_NO\_OPTION);

// handle the user's choice

if (choice == JOptionPane.YES\_OPTION)

{

if (chipsetNameField.getText().matches(chipsetNameRegex))

{

int selectedRow = chipsetTable.getSelectedRow();

if (selectedRow != -1) {

String oldChipsetName = (String) chipsetTableModel.getValueAt(selectedRow, 0);

String newChipsetName = chipsetNameField.getText();

if (newChipsetName != null && !newChipsetName.isEmpty()) {

try (Session session = getSession()) {

Transaction transaction = session.beginTransaction();

ClassChipset chipset = (ClassChipset) session.createQuery("FROM ClassChipset WHERE name = :name").setParameter("name", oldChipsetName).uniqueResult();

if (chipset != null) {

chipset.setName(newChipsetName);

session.update(chipset);

populateTables();

transaction.commit();

populateTables();

updateAllDropBoxes();

JOptionPane.showMessageDialog(null, "chipset updated successfully.");

} else {

JOptionPane.showMessageDialog(null, "chipset not found.");

}

} catch (Exception ex) {

ex.printStackTrace();

JOptionPane.showMessageDialog(null, "Failed to update chipset: " + ex);

}

} else {

JOptionPane.showMessageDialog(null, "Invalid chipset name.");

}

} else {

JOptionPane.showMessageDialog(null, "No chipset selected.");

}

} else {

JOptionPane.showMessageDialog(null, "Invalid data format!");

}

} else {

System.out.println("User clicked NO");

}

} else {

JOptionPane.showMessageDialog(chipsetPanel, "Сan't update any record! Please select one!", "Error", row);

}

}

});

deleteChipsetButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e)

{

int selectedRow = chipsetTable.getSelectedRow();

if (selectedRow != -1) {

// create the popup window with yes/no options

int choice = JOptionPane.showConfirmDialog(chipsetPanel, "Do you wish to continue? ", "Confirmation", JOptionPane.YES\_NO\_OPTION);

// handle the user's choice

if (choice == JOptionPane.YES\_OPTION)

{

String chipsetName = (String) chipsetTableModel.getValueAt(selectedRow, 0);

try (Session session = getSession()) {

Transaction transaction = session.beginTransaction();

ClassChipset chipset = (ClassChipset) session.createQuery("FROM ClassChipset WHERE name = :name").setParameter("name", chipsetName).uniqueResult();

if (chipset != null) {

session.delete(chipset);

transaction.commit();

populateTables(); // Refresh the table data after deleting the brand

updateAllDropBoxes();

JOptionPane.showMessageDialog(null, "chipset deleted successfully.");

} else {

JOptionPane.showMessageDialog(null, "chipset not found.");

}

} catch (Exception ex) {

ex.printStackTrace();

JOptionPane.showMessageDialog(null, "Failed to delete chipset: " + ex);

}

}

} else {

JOptionPane.showMessageDialog(null, "No chipset selected.");

}

}

});

pdfExportChipsetButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e)

{

LOGGER.addHandler(fileHandler);

LOGGER.info("Trying to export data to PDF document");

try

{

JFileChooser fileChooser = new JFileChooser();

// Set default folder to current directory

fileChooser.setCurrentDirectory(new File("."));

// Set default file name

fileChooser.setSelectedFile(new File("exported\_Chipsets.pdf"));

int result = fileChooser.showSaveDialog(null);

if (result == JFileChooser.APPROVE\_OPTION)

{

File selectedFile = fileChooser.getSelectedFile();

String fileName = selectedFile.getAbsolutePath();

// Append .pdf extension if necessary

if (!fileName.endsWith(".pdf"))

{

fileName += ".pdf";

}

Document document = new Document();

PdfWriter.getInstance(document, new FileOutputStream(fileName));

document.open();

PdfPTable pdfTable = new PdfPTable(chipsetTable.getColumnCount());

// Create font for table headers

Font headerFont = FontFactory.getFont(FontFactory.HELVETICA\_BOLD, 10, BaseColor.BLACK);

String[] headersPdfExport = {"\nChipset name\n\n"};

// Set column headers

for (int i = 0; i < chipsetTable.getColumnCount(); i++)

{

PdfPCell header = new PdfPCell(new Phrase(headersPdfExport[i], headerFont));

header.setBackgroundColor(BaseColor.ORANGE);

header.setBorderWidth(2);

header.setHorizontalAlignment(Element.ALIGN\_CENTER);

// Give more weight to the first row

pdfTable.addCell(header);

}

// Create font for table data

Font dataFont = FontFactory.getFont(FontFactory.HELVETICA, 10, BaseColor.BLACK);

// Set custom widths for each row

float[] columnWidths = {1f};

pdfTable.setWidths(columnWidths);

// Add table data

for (int i = 0; i < chipsetTable.getRowCount(); i++)

{

for (int j = 0; j < chipsetTable.getColumnCount(); j++)

{

PdfPCell data = new PdfPCell(new Phrase(chipsetTable.getValueAt(i, j).toString(), dataFont));

if (i % 2 == 1)

{

data.setBackgroundColor(BaseColor.LIGHT\_GRAY);

}

else

{

data.setBackgroundColor(BaseColor.WHITE);

}

data.setBorderWidth(1);

data.setHorizontalAlignment(Element.ALIGN\_LEFT);

pdfTable.addCell(data);

}

}

document.add(pdfTable);

document.close();

// JOptionPane.showMessageDialog(chipsetPanel, "Exported table data to " + fileName);

}

}

catch (Exception ex)

{

ex.printStackTrace();

JOptionPane.showMessageDialog(chipsetPanel, "Error exporting table data to PDF");

}

}

});

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

// \_\_\_\_\_\_ \_\_\_\_\_\_\_\_ \_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_ \_\_\_\_\_\_ \_\_ \_\_ \_\_ \_\_\_\_\_\_ \_\_\_\_\_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_ \_\_ \_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_ \_\_\_\_\_\_ //

// / \ / |/ | / | / \ / |/ | / \ / \ / | / | / | / \ / |/ |/ \ / |/ |/ \ / \ //

// /$$$$$$ |$$$$$$$$/ $$ | $$$$$$$$/ /$$$$$$ |$$$$$$$$/ $$$$$$/ /$$$$$$ |$$ \ $$ | $$ | $$$$$$/ /$$$$$$ |$$$$$$$$/ $$$$$$$$/ $$ \ $$ |$$$$$$$$/ $$$$$$$ |/$$$$$$ | //

// $$ \\_\_$$/ $$ |\_\_ $$ | $$ |\_\_ $$ | $$/ $$ | $$ | $$ | $$ |$$$ \$$ | $$ | $$ | $$ \\_\_$$/ $$ | $$ |\_\_ $$$ \$$ |$$ |\_\_ $$ |\_\_$$ |$$ \\_\_$$/ //

// $$ \ $$ | $$ | $$ | $$ | $$ | $$ | $$ | $$ |$$$$ $$ | $$ | $$ | $$ \ $$ | $$ | $$$$ $$ |$$ | $$ $$< $$ \ //

// $$$$$$ |$$$$$/ $$ | $$$$$/ $$ | \_\_ $$ | $$ | $$ | $$ |$$ $$ $$ | $$ | $$ | $$$$$$ | $$ | $$$$$/ $$ $$ $$ |$$$$$/ $$$$$$$ | $$$$$$ | //

// / \\_\_$$ |$$ |\_\_\_\_\_ $$ |\_\_\_\_\_ $$ |\_\_\_\_\_ $$ \\_\_/ | $$ | \_$$ |\_ $$ \\_\_$$ |$$ |$$$$ | $$ |\_\_\_\_\_ \_$$ |\_ / \\_\_$$ | $$ | $$ |\_\_\_\_\_ $$ |$$$$ |$$ |\_\_\_\_\_ $$ | $$ |/ \\_\_$$ | //

// $$ $$/ $$ |$$ |$$ |$$ $$/ $$ | / $$ |$$ $$/ $$ | $$$ | $$ |/ $$ |$$ $$/ $$ | $$ |$$ | $$$ |$$ |$$ | $$ |$$ $$/ //

// $$$$$$/ $$$$$$$$/ $$$$$$$$/ $$$$$$$$/ $$$$$$/ $$/ $$$$$$/ $$$$$$/ $$/ $$/ $$$$$$$$/ $$$$$$/ $$$$$$/ $$/ $$$$$$$$/ $$/ $$/ $$$$$$$$/ $$/ $$/ $$$$$$/ //

// //

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

// Set table selection listeners

brandTable.getSelectionModel().addListSelectionListener(new ListSelectionListener() {

@Override

public void valueChanged(ListSelectionEvent e)

{

int selectedRow = brandTable.getSelectedRow();

if (selectedRow >= 0)

{

String name = brandTableModel.getValueAt(selectedRow, 0).toString();

brandNameField.setText(name);

}

}

});

socketTable.getSelectionModel().addListSelectionListener(new ListSelectionListener() {

@Override

public void valueChanged(ListSelectionEvent e)

{

int selectedRow = socketTable.getSelectedRow();

if (selectedRow >= 0)

{

String name = socketTableModel.getValueAt(selectedRow, 0).toString();

socketNameField.setText(name);

}

}

});

chipsetTable.getSelectionModel().addListSelectionListener(new ListSelectionListener() {

@Override

public void valueChanged(ListSelectionEvent e)

{

int selectedRow = chipsetTable.getSelectedRow();

if (selectedRow >= 0)

{

String name = chipsetTableModel.getValueAt(selectedRow, 0).toString();

chipsetNameField.setText(name);

}

}

});

cpuTable.getSelectionModel().addListSelectionListener(new ListSelectionListener() {

@Override

public void valueChanged(ListSelectionEvent e)

{

int selectedRow = cpuTable.getSelectedRow();

if (selectedRow != -1)

{ // Update CPU fields with selected row data

cpuModelField.setText(cpuTableModel.getValueAt(selectedRow, 0).toString());

cpuPriceField.setText(cpuTableModel.getValueAt(selectedRow, 1).toString());

cpuCoresField.setText(cpuTableModel.getValueAt(selectedRow, 2).toString());

cpuThreadsField.setText(cpuTableModel.getValueAt(selectedRow, 3).toString());

cpuFrequencyField.setText(cpuTableModel.getValueAt(selectedRow, 4).toString());

// Get the values from the table model

String brand = cpuTableModel.getValueAt(selectedRow, 5).toString();

String socket = cpuTableModel.getValueAt(selectedRow, 6).toString();

// Set the selected item in the comboboxes

for (int i = 0; i < cpuBrandComboBox.getItemCount(); i++)

{

String item = cpuBrandComboBox.getItemAt(i).toString();

if (item.equals(brand))

{

cpuBrandComboBox.setSelectedItem(item);

break;

}

}

for (int i = 0; i < cpuSocketComboBox.getItemCount(); i++)

{

String item = cpuSocketComboBox.getItemAt(i).toString();

if (item.equals(socket))

{

cpuSocketComboBox.setSelectedItem(item);

break;

}

}

}

}

});

gpuTable.getSelectionModel().addListSelectionListener(new ListSelectionListener() {

@Override

public void valueChanged(ListSelectionEvent e)

{

int selectedRow = gpuTable.getSelectedRow();

if (selectedRow != -1)

{ // Update CPU fields with selected row data

gpuModelField.setText(gpuTableModel.getValueAt(selectedRow, 0).toString());

gpuPriceField.setText(gpuTableModel.getValueAt(selectedRow, 1).toString());

gpuCoresField.setText(gpuTableModel.getValueAt(selectedRow, 2).toString());

gpuMemoryField.setText(gpuTableModel.getValueAt(selectedRow, 3).toString());

gpuFrequencyField.setText(gpuTableModel.getValueAt(selectedRow, 4).toString());

// Get the values from the table model

String brand = gpuTableModel.getValueAt(selectedRow, 5).toString();

// Set the selected item in the comboboxes

for (int i = 0; i < gpuBrandComboBox.getItemCount(); i++)

{

String item = gpuBrandComboBox.getItemAt(i).toString();

if (item.equals(brand))

{

gpuBrandComboBox.setSelectedItem(item);

break;

}

}

}

}

});

pcbTable.getSelectionModel().addListSelectionListener(new ListSelectionListener() {

@Override

public void valueChanged(ListSelectionEvent e)

{

int selectedRow = pcbTable.getSelectedRow();

if (selectedRow != -1)

{ // Update CPU fields with selected row data

pcbModelField.setText(pcbTableModel.getValueAt(selectedRow, 0).toString());

pcbPriceField.setText(pcbTableModel.getValueAt(selectedRow, 1).toString());

// Get the values from the table model

String brand = pcbTableModel.getValueAt(selectedRow, 2).toString();

String socket = pcbTableModel.getValueAt(selectedRow, 3).toString();

String chipset = pcbTableModel.getValueAt(selectedRow, 4).toString();

// Set the selected item in the comboboxes

for (int i = 0; i < pcbBrandComboBox.getItemCount(); i++)

{

String item = pcbBrandComboBox.getItemAt(i).toString();

if (item.equals(brand))

{

pcbBrandComboBox.setSelectedItem(item);

break;

}

}

for (int i = 0; i < pcbSocketComboBox.getItemCount(); i++)

{

String item = pcbSocketComboBox.getItemAt(i).toString();

if (item.equals(socket))

{

pcbSocketComboBox.setSelectedItem(item);

break;

}

}

for (int i = 0; i < pcbChipsetComboBox.getItemCount(); i++)

{

String item = pcbChipsetComboBox.getItemAt(i).toString();

if (item.equals(chipset))

{

pcbChipsetComboBox.setSelectedItem(item);

break;

}

}

}

}

});

// Set the selected tab to the first tab

tabbedPane.setSelectedIndex(0);

setVisible(true);

}

}

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

// \_\_\_\_\_\_\_ \_\_\_\_\_\_ \_\_\_\_\_\_\_ \_\_ \_\_ \_\_ \_\_\_\_\_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_ \_\_\_\_\_\_\_ \_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_ \_\_ \_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_ \_\_ \_\_ \_\_\_\_\_\_ \_\_\_\_\_\_\_ \_\_\_\_\_\_ //

// / \ / \ / \ / | / |/ | / \ / |/ | / |/ \ / \ / | / | / \ / \ / |/ |/ |/ | / | / \ / \ / \ //

// $$$$$$$ |/$$$$$$ |$$$$$$$ |$$ | $$ |$$ | /$$$$$$ |$$$$$$$$/ $$$$$$$$/ $$$$$$$$//$$$$$$ |$$$$$$$ |$$ | $$$$$$$$/ /$$$$$$ | $$ \ /$$ |$$$$$$$$/ $$$$$$$$/ $$ | $$ |/$$$$$$ |$$$$$$$ |/$$$$$$ | //

// $$ |\_\_$$ |$$ | $$ |$$ |\_\_$$ |$$ | $$ |$$ | $$ |\_\_$$ | $$ | $$ |\_\_ $$ | $$ |\_\_$$ |$$ |\_\_$$ |$$ | $$ |\_\_ $$ \\_\_$$/ $$$ \ /$$$ |$$ |\_\_ $$ | $$ |\_\_$$ |$$ | $$ |$$ | $$ |$$ \\_\_$$/ //

// $$ $$/ $$ | $$ |$$ $$/ $$ | $$ |$$ | $$ $$ | $$ | $$ | $$ | $$ $$ |$$ $$< $$ | $$ | $$ \ $$$$ /$$$$ |$$ | $$ | $$ $$ |$$ | $$ |$$ | $$ |$$ \ //

// $$$$$$$/ $$ | $$ |$$$$$$$/ $$ | $$ |$$ | $$$$$$$$ | $$ | $$$$$/ $$ | $$$$$$$$ |$$$$$$$ |$$ | $$$$$/ $$$$$$ | $$ $$ $$/$$ |$$$$$/ $$ | $$$$$$$$ |$$ | $$ |$$ | $$ | $$$$$$ | //

// $$ | $$ \\_\_$$ |$$ | $$ \\_\_$$ |$$ |\_\_\_\_\_ $$ | $$ | $$ | $$ |\_\_\_\_\_ $$ | $$ | $$ |$$ |\_\_$$ |$$ |\_\_\_\_\_ $$ |\_\_\_\_\_ / \\_\_$$ | $$ |$$$/ $$ |$$ |\_\_\_\_\_ $$ | $$ | $$ |$$ \\_\_$$ |$$ |\_\_$$ |/ \\_\_$$ | //

// $$ | $$ $$/ $$ | $$ $$/ $$ |$$ | $$ | $$ | $$ | $$ | $$ | $$ |$$ $$/ $$ |$$ |$$ $$/ $$ | $/ $$ |$$ | $$ | $$ | $$ |$$ $$/ $$ $$/ $$ $$/ //

// $$/ $$$$$$/ $$/ $$$$$$/ $$$$$$$$/ $$/ $$/ $$/ $$$$$$$$/ $$/ $$/ $$/ $$$$$$$/ $$$$$$$$/ $$$$$$$$/ $$$$$$/ $$/ $$/ $$$$$$$$/ $$/ $$/ $$/ $$$$$$/ $$$$$$$/ $$$$$$/ //

// //

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

// Populate the tables with data from the database

private void populateTables()

{

// Clear tables

brandTableModel.setRowCount(0);

socketTableModel.setRowCount(0);

chipsetTableModel.setRowCount(0);

cpuTableModel.setRowCount(0);

gpuTableModel.setRowCount(0);

pcbTableModel.setRowCount(0);

// Populate the tables with data from the database

try (Session session = getSession()) {

List<ClassBrand> brands = session.createQuery("FROM ClassBrand", ClassBrand.class).list();

for (ClassBrand brand : brands) {

brandTableModel.addRow(new Object[]{brand.getName()});

}

} catch (Exception e) {

e.printStackTrace();

}

try (Session session = getSession()) {

List<ClassChipset> chipsets = session.createQuery("FROM ClassChipset", ClassChipset.class).list();

for (ClassChipset chipset : chipsets) {

chipsetTableModel.addRow(new Object[]{chipset.getName()});

}

} catch (Exception e) {

e.printStackTrace();

}

try (Session session = getSession()) {

List<ClassSocket> sockets = session.createQuery("FROM ClassSocket", ClassSocket.class).list();

for (ClassSocket socket : sockets) {

socketTableModel.addRow(new Object[]{socket.getName()});

}

} catch (Exception e) {

e.printStackTrace();

}

try (Session session = getSession()) {

List<ClassCPU> cpus = session.createQuery("FROM ClassCPU", ClassCPU.class).list();

for (ClassCPU cpu : cpus) {

cpuTableModel.addRow(new Object[]{cpu.getModel(), cpu.getPrice(), cpu.getCores(), cpu.getThreads(), cpu.getFrequency(), cpu.getBrand().getName(), cpu.getSocket().getName()});

}

} catch (Exception e) {

e.printStackTrace();

}

try (Session session = getSession()) {

List<ClassGPU> gpus = session.createQuery("FROM ClassGPU", ClassGPU.class).list();

for (ClassGPU gpu : gpus) {

gpuTableModel.addRow(new Object[]{gpu.getModel(), gpu.getPrice(), gpu.getCores(), gpu.getMemory(), gpu.getFrequency(), gpu.getBrand().getName()});

}

} catch (Exception e) {

e.printStackTrace();

}

try (Session session = getSession()) {

List<ClassPCB> motherboards = session.createQuery("FROM ClassPCB", ClassPCB.class).list();

for (ClassPCB motherboard : motherboards) {

pcbTableModel.addRow(new Object[]{motherboard.getModel(), motherboard.getPrice(), motherboard.getBrand().getName(), motherboard.getSocket().getName(), motherboard.getChipset().getName()});

}

} catch (Exception e) {

e.printStackTrace();

}

}

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

// \_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_ \_\_\_\_\_\_ \_\_ \_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_ \_\_\_\_\_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_ \_\_ \_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_ \_\_ \_\_ \_\_\_\_\_\_ \_\_\_\_\_\_\_ \_\_\_\_\_\_ //

// / \ / |/ |/ \ / |/ | / |/ | / \ / \ / |/ \ / \ / |/ |/ |/ | / | / \ / \ / \ //

// $$$$$$$ |$$$$$$$$/ $$$$$$$$/ $$$$$$$ |$$$$$$/ $$ | $$ |$$$$$$$$/ $$$$$$$ |/$$$$$$ |$$$$$$$$//$$$$$$ | $$ \ /$$ |$$$$$$$$/ $$$$$$$$/ $$ | $$ |/$$$$$$ |$$$$$$$ |/$$$$$$ | //

// $$ |\_\_$$ |$$ |\_\_ $$ | $$ |\_\_$$ | $$ | $$ | $$ |$$ |\_\_ $$ | $$ |$$ |\_\_$$ | $$ | $$ |\_\_$$ | $$$ \ /$$$ |$$ |\_\_ $$ | $$ |\_\_$$ |$$ | $$ |$$ | $$ |$$ \\_\_$$/ //

// $$ $$< $$ | $$ | $$ $$< $$ | $$ \ /$$/ $$ | $$ | $$ |$$ $$ | $$ | $$ $$ | $$$$ /$$$$ |$$ | $$ | $$ $$ |$$ | $$ |$$ | $$ |$$ \ //

// $$$$$$$ |$$$$$/ $$ | $$$$$$$ | $$ | $$ /$$/ $$$$$/ $$ | $$ |$$$$$$$$ | $$ | $$$$$$$$ | $$ $$ $$/$$ |$$$$$/ $$ | $$$$$$$$ |$$ | $$ |$$ | $$ | $$$$$$ | //

// $$ | $$ |$$ |\_\_\_\_\_ $$ | $$ | $$ | \_$$ |\_ $$ $$/ $$ |\_\_\_\_\_ $$ |\_\_$$ |$$ | $$ | $$ | $$ | $$ | $$ |$$$/ $$ |$$ |\_\_\_\_\_ $$ | $$ | $$ |$$ \\_\_$$ |$$ |\_\_$$ |/ \\_\_$$ | //

// $$ | $$ |$$ | $$ | $$ | $$ |/ $$ | $$$/ $$ | $$ $$/ $$ | $$ | $$ | $$ | $$ | $$ | $/ $$ |$$ | $$ | $$ | $$ |$$ $$/ $$ $$/ $$ $$/ //

// $$/ $$/ $$$$$$$$/ $$/ $$/ $$/ $$$$$$/ $/ $$$$$$$$/ $$$$$$$/ $$/ $$/ $$/ $$/ $$/ $$/ $$/ $$$$$$$$/ $$/ $$/ $$/ $$$$$$/ $$$$$$$/ $$$$$$/ //

// //

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

private List<ClassBrand> retrieveBrands()

{

List<ClassBrand> brands = null;

try

{

transaction = session.beginTransaction();

brands = session.createQuery("FROM ClassBrand", ClassBrand.class).list();

transaction.commit();

}

catch (Exception e)

{

if (transaction != null)

{

transaction.rollback();

}

e.printStackTrace();

}

return brands;

}

private List<ClassChipset> retrieveChipsets()

{

List<ClassChipset> chipsets = null;

try

{

transaction = session.beginTransaction();

chipsets = session.createQuery("FROM ClassChipset", ClassChipset.class).list();

transaction.commit();

}

catch (Exception e)

{

if (transaction != null)

{

transaction.rollback();

}

e.printStackTrace();

}

return chipsets;

}

private List<ClassCPU> retrieveCPUs()

{

List<ClassCPU> cpus = null;

try

{

transaction = session.beginTransaction();

cpus = session.createQuery("FROM ClassCPU", ClassCPU.class).list();

transaction.commit();

}

catch (Exception e)

{

if (transaction != null)

{

transaction.rollback();

}

e.printStackTrace();

}

return cpus;

}

private List<ClassGPU> retrieveGPUs()

{

List<ClassGPU> gpus = null;

try

{

transaction = session.beginTransaction();

gpus = session.createQuery("FROM ClassGPU", ClassGPU.class).list();

transaction.commit();

}

catch (Exception e)

{

if (transaction != null)

{

transaction.rollback();

}

e.printStackTrace();

}

return gpus;

}

private List<ClassPCB> retrievePCBs()

{

List<ClassPCB> pcbs = null;

try

{

transaction = session.beginTransaction();

pcbs = session.createQuery("FROM ClassPCB", ClassPCB.class).list();

transaction.commit();

}

catch (Exception e)

{

if (transaction != null)

{

transaction.rollback();

}

e.printStackTrace();

}

return pcbs;

}

private List<ClassSocket> retrieveSockets()

{

List<ClassSocket> sockets = null;

try

{

transaction = session.beginTransaction();

sockets = session.createQuery("FROM ClassSocket", ClassSocket.class).list();

transaction.commit();

}

catch (Exception e)

{

if (transaction != null)

{

transaction.rollback();

}

e.printStackTrace();

}

return sockets;

}

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

// \_\_ \_\_ \_\_\_\_\_\_\_\_ \_\_ \_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_ \_\_ \_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_ \_\_ \_\_ \_\_\_\_\_\_ \_\_\_\_\_\_\_ \_\_\_\_\_\_ //

// / | / |/ |/ | / \ / |/ \ / \ / |/ |/ |/ | / | / \ / \ / \ //

// $$ | $$ |$$$$$$$$/ $$ | $$$$$$$ |$$$$$$$$/ $$$$$$$ | $$ \ /$$ |$$$$$$$$/ $$$$$$$$/ $$ | $$ |/$$$$$$ |$$$$$$$ |/$$$$$$ | //

// $$ |\_\_$$ |$$ |\_\_ $$ | $$ |\_\_$$ |$$ |\_\_ $$ |\_\_$$ | $$$ \ /$$$ |$$ |\_\_ $$ | $$ |\_\_$$ |$$ | $$ |$$ | $$ |$$ \\_\_$$/ //

// $$ $$ |$$ | $$ | $$ $$/ $$ | $$ $$< $$$$ /$$$$ |$$ | $$ | $$ $$ |$$ | $$ |$$ | $$ |$$ \ //

// $$$$$$$$ |$$$$$/ $$ | $$$$$$$/ $$$$$/ $$$$$$$ | $$ $$ $$/$$ |$$$$$/ $$ | $$$$$$$$ |$$ | $$ |$$ | $$ | $$$$$$ | //

// $$ | $$ |$$ |\_\_\_\_\_ $$ |\_\_\_\_\_ $$ | $$ |\_\_\_\_\_ $$ | $$ | $$ |$$$/ $$ |$$ |\_\_\_\_\_ $$ | $$ | $$ |$$ \\_\_$$ |$$ |\_\_$$ |/ \\_\_$$ | //

// $$ | $$ |$$ |$$ |$$ | $$ |$$ | $$ | $$ | $/ $$ |$$ | $$ | $$ | $$ |$$ $$/ $$ $$/ $$ $$/ //

// $$/ $$/ $$$$$$$$/ $$$$$$$$/ $$/ $$$$$$$$/ $$/ $$/ $$/ $$/ $$$$$$$$/ $$/ $$/ $$/ $$$$$$/ $$$$$$$/ $$$$$$/ //

// //

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

// Helper method to create a panel with buttons

private JPanel createButtonPanel(JButton addButton, JButton deleteButton, JButton updateButton, JButton pdfExportButton) {

JPanel buttonPanel = new JPanel(new FlowLayout());

buttonPanel.add(addButton);

buttonPanel.add(deleteButton);

buttonPanel.add(updateButton);

buttonPanel.add(pdfExportButton);

return buttonPanel;

}

// Helper method to retrieve Hibernate session

private Session getSession() {

Configuration configuration = new Configuration().configure();

SessionFactory sessionFactory = configuration.buildSessionFactory();

return sessionFactory.openSession();

}

private void updateAllDropBoxes() {

cpuBrandComboBox.removeAllItems();

cpuSocketComboBox.removeAllItems();

gpuBrandComboBox.removeAllItems();

pcbBrandComboBox .removeAllItems();

pcbSocketComboBox.removeAllItems();

pcbChipsetComboBox.removeAllItems();

try (Session session = getSession()) {

List<ClassBrand> brands = session.createQuery("FROM ClassBrand", ClassBrand.class).list();

for (ClassBrand brand : brands) {

cpuBrandComboBox.addItem(brand.getName());

}

} catch (Exception e) {

e.printStackTrace();

}

try (Session session = getSession()) {

List<ClassSocket> sockets = session.createQuery("FROM ClassSocket", ClassSocket.class).list();

for (ClassSocket socket : sockets) {

cpuSocketComboBox.addItem(socket.getName());

}

} catch (Exception e) {

e.printStackTrace();

}

try (Session session = getSession()) {

List<ClassBrand> brands = session.createQuery("FROM ClassBrand", ClassBrand.class).list();

for (ClassBrand brand : brands) {

gpuBrandComboBox.addItem(brand.getName());

}

} catch (Exception e) {

e.printStackTrace();

}

try (Session session = getSession()) {

List<ClassBrand> brands = session.createQuery("FROM ClassBrand", ClassBrand.class).list();

for (ClassBrand brand : brands) {

pcbBrandComboBox.addItem(brand.getName());

}

} catch (Exception e) {

e.printStackTrace();

}

try (Session session = getSession()) {

List<ClassSocket> sockets = session.createQuery("FROM ClassSocket", ClassSocket.class).list();

for (ClassSocket socket : sockets) {

pcbSocketComboBox.addItem(socket.getName());

}

} catch (Exception e) {

e.printStackTrace();

}

try (Session session = getSession()) {

List<ClassChipset> chipsets = session.createQuery("FROM ClassChipset", ClassChipset.class).list();

for (ClassChipset chipset : chipsets) {

pcbChipsetComboBox.addItem(chipset.getName());

}

} catch (Exception e) {

e.printStackTrace();

}

}

private void applyCPUFilter() {

String column = filterCPUComboBox.getSelectedItem().toString();

String value = filterCPUTextField.getText(); // .getText().trim();

if (column.isEmpty() || value.isEmpty()) {

// If either the column or value is empty, show all rows

showAllRows(cpuTable, cpuTableModel);

} else {

// Filter the rows based on the selected column and value

List<Integer> filteredRows = new ArrayList<>();

for (int i = 0; i < cpuTableModel.getRowCount(); i++)

{

Object cellValue = cpuTableModel.getValueAt(i, getCPUColumnIndex(column));

if (cellValue != null && cellValue.toString().toLowerCase().contains(value.toLowerCase()))

{

filteredRows.add(i);

}

}

// Show only the filtered rows

showFilteredRows(cpuTable, filteredRows);

}

}

private void applyGPUFilter() {

String column = filterGPUComboBox.getSelectedItem().toString();

String value = filterGPUTextField.getText(); // .getText().trim();

if (column.isEmpty() || value.isEmpty()) {

// If either the column or value is empty, show all rows

showAllRows(gpuTable, gpuTableModel);

} else {

// Filter the rows based on the selected column and value

List<Integer> filteredRows = new ArrayList<>();

for (int i = 0; i < gpuTableModel.getRowCount(); i++)

{

Object cellValue = gpuTableModel.getValueAt(i, getGPUColumnIndex(column));

if (cellValue != null && cellValue.toString().toLowerCase().contains(value.toLowerCase()))

{

filteredRows.add(i);

}

}

// Show only the filtered rows

showFilteredRows(gpuTable, filteredRows);

}

}

private void applyPCBFilter() {

String column = filterPCBComboBox.getSelectedItem().toString();

String value = filterPCBTextField.getText(); // .getText().trim();

if (column.isEmpty() || value.isEmpty()) {

// If either the column or value is empty, show all rows

showAllRows(pcbTable, pcbTableModel);

} else {

// Filter the rows based on the selected column and value

List<Integer> filteredRows = new ArrayList<>();

for (int i = 0; i < pcbTableModel.getRowCount(); i++)

{

Object cellValue = pcbTableModel.getValueAt(i, getPCBColumnIndex(column));

if (cellValue != null && cellValue.toString().toLowerCase().contains(value.toLowerCase()))

{

filteredRows.add(i);

}

}

// Show only the filtered rows

showFilteredRows(pcbTable, filteredRows);

}

}

private int getCPUColumnIndex(String column) {

switch (column) {

case "Model":

return 0;

case "Price":

return 1;

case "Cores":

return 2;

case "Threads":

return 3;

case "Frequency":

return 4;

case "Brand":

return 5;

case "Socket":

return 6;

default:

return -1; // Return -1 for unknown options

}

}

private int getGPUColumnIndex(String column) {

switch (column) {

case "Model":

return 0;

case "Price":

return 1;

case "Cores":

return 2;

case "Memory":

return 3;

case "Frequency":

return 4;

case "Brand":

return 5;

default:

return -1; // Return -1 for unknown options

}

}

private int getPCBColumnIndex(String column) {

switch (column) {

case "Model":

return 0;

case "Price":

return 1;

case "Brand":

return 2;

case "Socket":

return 3;

case "Chipset":

return 4;

default:

return -1; // Return -1 for unknown options

}

}

private void showAllRows(JTable table, DefaultTableModel model) {

table.clearSelection();

for (int i = 0; i < model.getRowCount(); i++) {

table.addRowSelectionInterval(i, i);

}

}

private void showFilteredRows(JTable table, List<Integer> rows) {

table.clearSelection();

for (int row : rows) {

table.addRowSelectionInterval(row, row);

}

}

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

// \_\_\_\_\_\_\_ \_\_ \_\_ \_\_ \_\_ \_\_ \_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_ \_\_ \_\_ \_\_\_\_\_\_ \_\_\_\_\_\_\_ \_\_\_\_\_\_\_ \_\_\_\_\_\_ \_\_\_\_\_\_ \_\_\_\_\_\_\_ \_\_\_\_\_\_ \_\_ \_\_ //

// / \ / | / |/ \ / | / |/ \ / |/ \ / | / | / \ / \ / \ / \ / \ / \ / \ / \ / | //

// $$$$$$$ |$$ | $$ |$$ \ $$ | /$$/ $$$$$$$ |$$$$$$$$/ $$$$$$$ |$$ | $$ |/$$$$$$ | $$$$$$$ |$$$$$$$ |/$$$$$$ |/$$$$$$ |$$$$$$$ |/$$$$$$ |$$ \ /$$ | //

// $$ |\_\_$$ |$$ | $$ |$$$ \$$ | /$$/ $$ | $$ |$$ |\_\_ $$ |\_\_$$ |$$ | $$ |$$ | \_$$/ $$ |\_\_$$ |$$ |\_\_$$ |$$ | $$ |$$ | \_$$/ $$ |\_\_$$ |$$ |\_\_$$ |$$$ \ /$$$ | //

// $$ $$< $$ | $$ |$$$$ $$ | /$$/ $$ | $$ |$$ | $$ $$< $$ | $$ |$$ |/ | $$ $$/ $$ $$< $$ | $$ |$$ |/ |$$ $$< $$ $$ |$$$$ /$$$$ | //

// $$$$$$$ |$$ | $$ |$$ $$ $$ | /$$/ $$ | $$ |$$$$$/ $$$$$$$ |$$ | $$ |$$ |$$$$ | $$$$$$$/ $$$$$$$ |$$ | $$ |$$ |$$$$ |$$$$$$$ |$$$$$$$$ |$$ $$ $$/$$ | //

// $$ | $$ |$$ \\_\_$$ |$$ |$$$$ | /$$/ $$ |\_\_$$ |$$ |\_\_\_\_\_ $$ |\_\_$$ |$$ \\_\_$$ |$$ \\_\_$$ | $$ | $$ | $$ |$$ \\_\_$$ |$$ \\_\_$$ |$$ | $$ |$$ | $$ |$$ |$$$/ $$ | //

// $$ | $$ |$$ $$/ $$ | $$$ |/$$/ $$ $$/ $$ |$$ $$/ $$ $$/ $$ $$/ $$ | $$ | $$ |$$ $$/ $$ $$/ $$ | $$ |$$ | $$ |$$ | $/ $$ | //

// $$/ $$/ $$$$$$/ $$/ $$/ $$/ $$$$$$$/ $$$$$$$$/ $$$$$$$/ $$$$$$/ $$$$$$/ $$/ $$/ $$/ $$$$$$/ $$$$$$/ $$/ $$/ $$/ $$/ $$/ $$/ //

// //

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

public static void main(String[] args)

{

SwingUtilities.invokeLater(new Runnable()

{

public void run()

{

try

{

fileHandler = new FileHandler("Logs.log"); // Initialize the file handler

fileHandler.setFormatter(new SimpleFormatter()); // Set the formatter for the file handler

LOGGER.addHandler(fileHandler); // Add the file handler to the logger

LOGGER.info("Logging started"); // Log some messages

// LOGGER.warning("This is a warning message");

// LOGGER.severe("This is a severe error message");

}

catch (IOException e)

{

e.printStackTrace();

}

new DBMSApp();

}

});

}

}