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.

Санкт-Петербург

2023

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

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

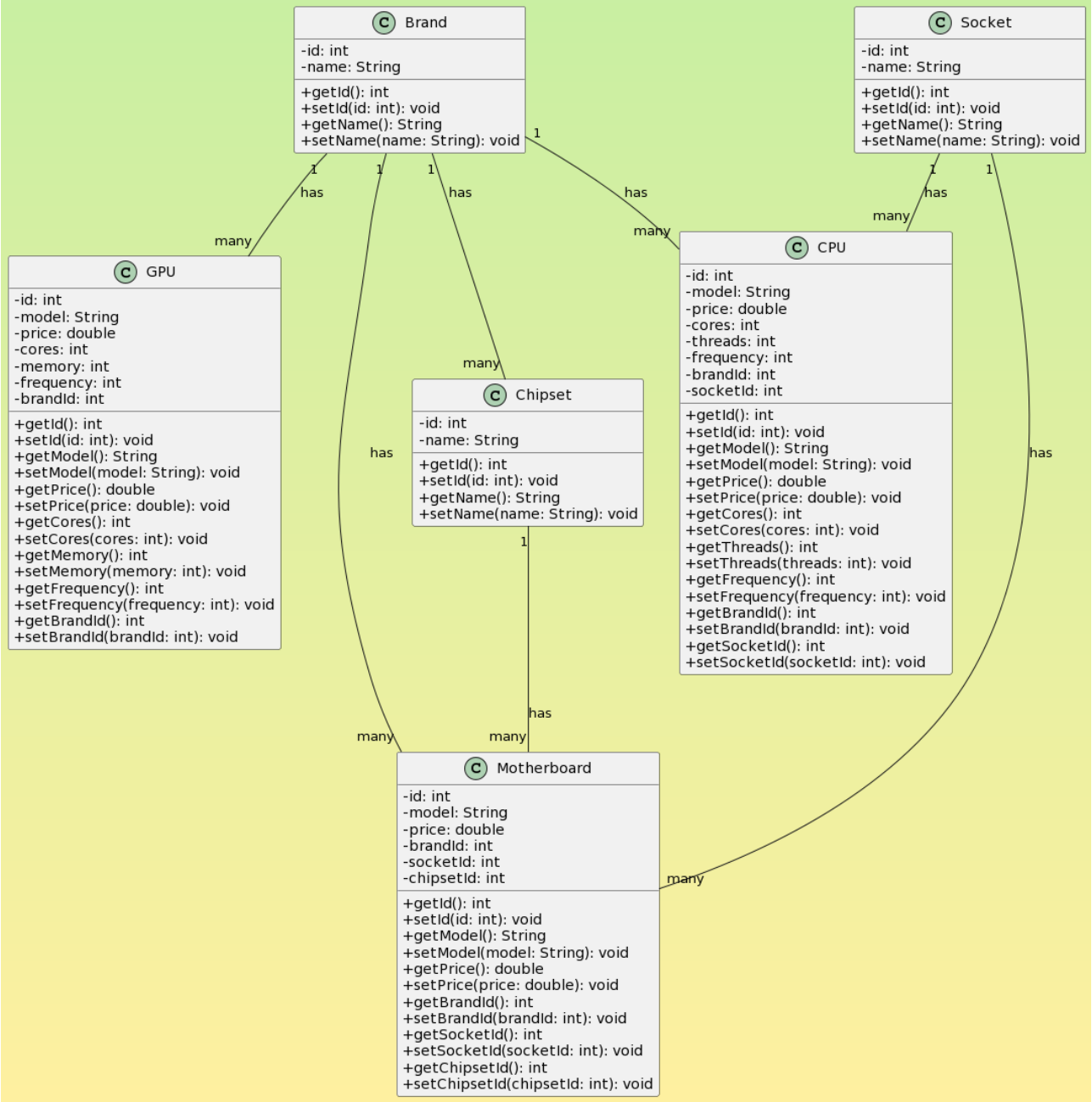


Рисунок 1. UML-диаграмма классов

1.1. 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.

1.2. 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.

1.3. 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.

1.4. 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.

1.5. 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.

1.6. 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.

1.7. Исходный код для классов

Ниже представлен код структуры и методов классов проекта:

Полный код самой программы представлен в приложении Б.

1.7.1. 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;
    }
}
```

1.7.2. 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;
    }
}
```

1.7.3. 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;
    }
}
```

1.7.4. 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;
    }
}
```

1.7.5. 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;
    }
}
```

1.7.6. 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

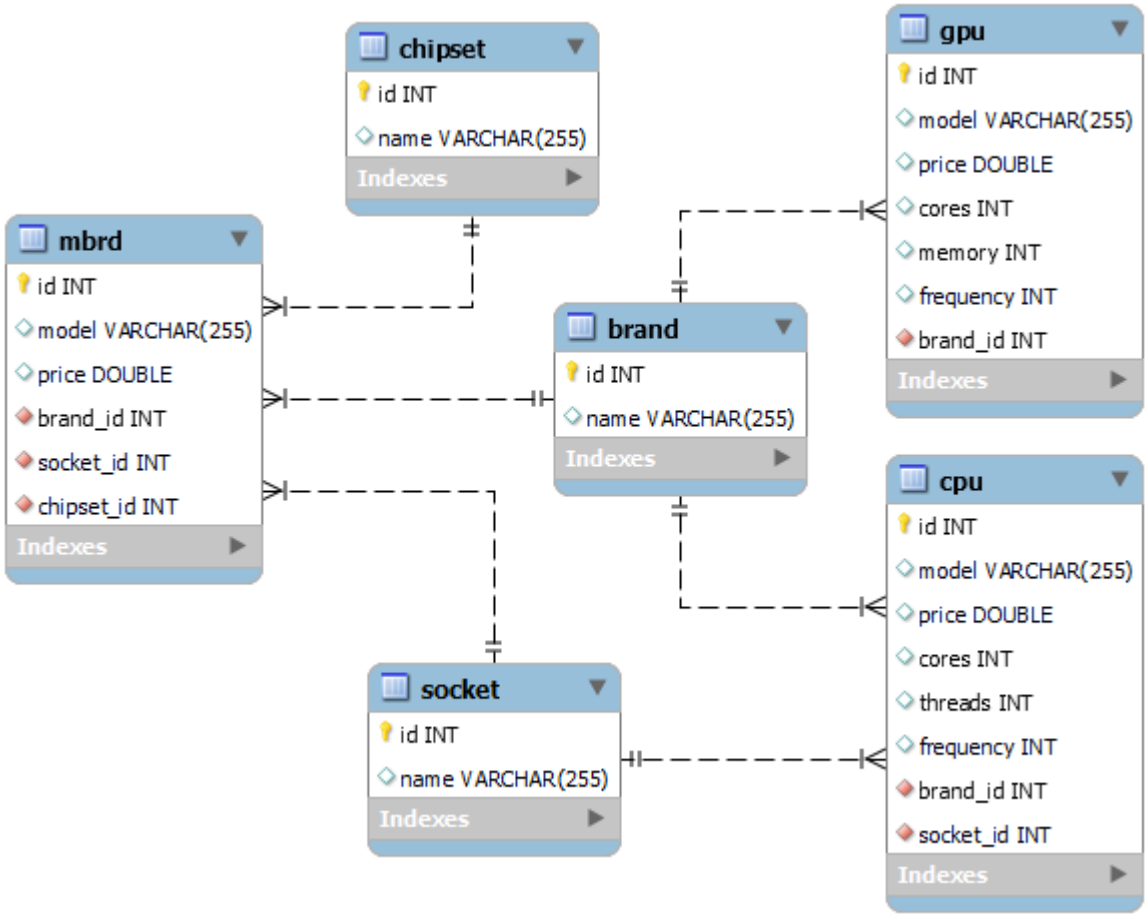


Рисунок 2. EER-диаграмма базы данных

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

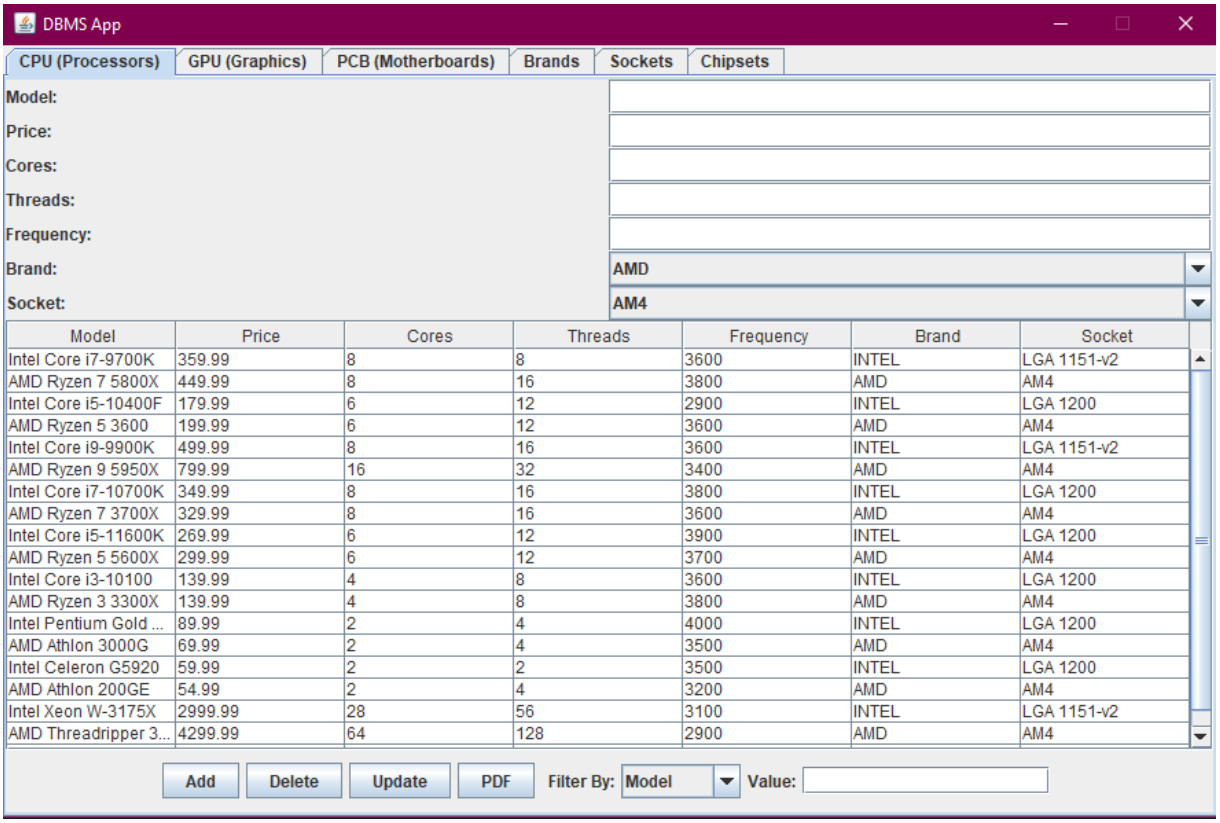
Наименование	Тип данных	Описание
id	INT	уникальный идентификатор материнской платы
model	VARCHAR(255)	модель материнской платы
price	DOUBLE	цена материнской платы
brand_id	INT	идентификатор бренда материнской платы (ссылка на таблицу brand)
socket_id	INT	идентификатор сокета материнской платы (ссылка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

Таблица № 7.

Свойства таблицы socket

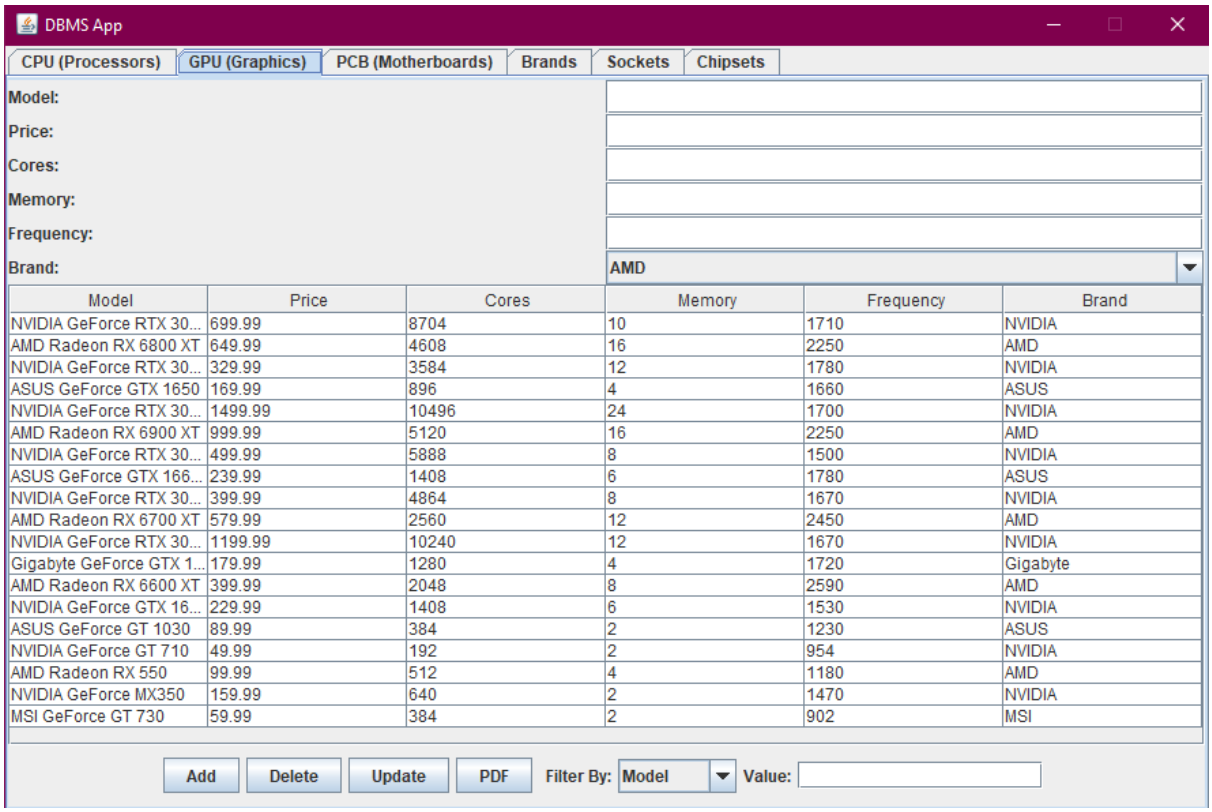
Наименование	Тип данных	Описание
id	INT	уникальный идентификатор сокета
name	VARCHAR(255)	название сокета

3. ИНТЕРФЕЙС ПРОГРАММЫ



Model	Price	Cores	Threads	Frequency	Brand	Socket
Intel Core i7-9700K	359.99	8	8	3600	INTEL	LGA 1151-v2
AMD Ryzen 7 5800X	449.99	8	16	3800	AMD	AM4
Intel Core i5-10400F	179.99	6	12	2900	INTEL	LGA 1200
AMD Ryzen 5 3600	199.99	6	12	3600	AMD	AM4
Intel Core i9-9900K	499.99	8	16	3600	INTEL	LGA 1151-v2
AMD Ryzen 9 5950X	799.99	16	32	3400	AMD	AM4
Intel Core i7-10700K	349.99	8	16	3800	INTEL	LGA 1200
AMD Ryzen 7 3700X	329.99	8	16	3600	AMD	AM4
Intel Core i5-11600K	269.99	6	12	3900	INTEL	LGA 1200
AMD Ryzen 5 5600X	299.99	6	12	3700	AMD	AM4
Intel Core i3-10100	139.99	4	8	3600	INTEL	LGA 1200
AMD Ryzen 3 3300X	139.99	4	8	3800	AMD	AM4
Intel Pentium Gold ...	89.99	2	4	4000	INTEL	LGA 1200
AMD Athlon 3000G	69.99	2	4	3500	AMD	AM4
Intel Celeron G5920	59.99	2	2	3500	INTEL	LGA 1200
AMD Athlon 200GE	54.99	2	4	3200	AMD	AM4
Intel Xeon W-3175X	2999.99	28	56	3100	INTEL	LGA 1151-v2
AMD Threadripper 3...	4299.99	64	128	2900	AMD	AM4

Рисунок 3. Интерфейс программы. Таблица процессоров



Model	Price	Cores	Memory	Frequency	Brand
NVIDIA GeForce RTX 30...	699.99	8704	10	1710	NVIDIA
AMD Radeon RX 6800 XT	649.99	4608	16	2250	AMD
NVIDIA GeForce RTX 30...	329.99	3584	12	1780	NVIDIA
ASUS GeForce GTX 1650	169.99	896	4	1660	ASUS
NVIDIA GeForce RTX 30...	1499.99	10496	24	1700	NVIDIA
AMD Radeon RX 6900 XT	999.99	5120	16	2250	AMD
NVIDIA GeForce RTX 30...	499.99	5888	8	1500	NVIDIA
ASUS GeForce GTX 166...	239.99	1408	6	1780	ASUS
NVIDIA GeForce RTX 30...	399.99	4864	8	1670	NVIDIA
AMD Radeon RX 6700 XT	579.99	2560	12	2450	AMD
NVIDIA GeForce RTX 30...	1199.99	10240	12	1670	NVIDIA
Gigabyte GeForce GTX 1...	179.99	1280	4	1720	Gigabyte
AMD Radeon RX 6600 XT	399.99	2048	8	2590	AMD
NVIDIA GeForce GTX 16...	229.99	1408	6	1530	NVIDIA
ASUS GeForce GT 1030	89.99	384	2	1230	ASUS
NVIDIA GeForce GT 710	49.99	192	2	954	NVIDIA
AMD Radeon RX 550	99.99	512	4	1180	AMD
NVIDIA GeForce MX350	159.99	640	2	1470	NVIDIA
MSI GeForce GT 730	59.99	384	2	902	MSI

Рисунок 4. Интерфейс программы. Таблица графических процессоров

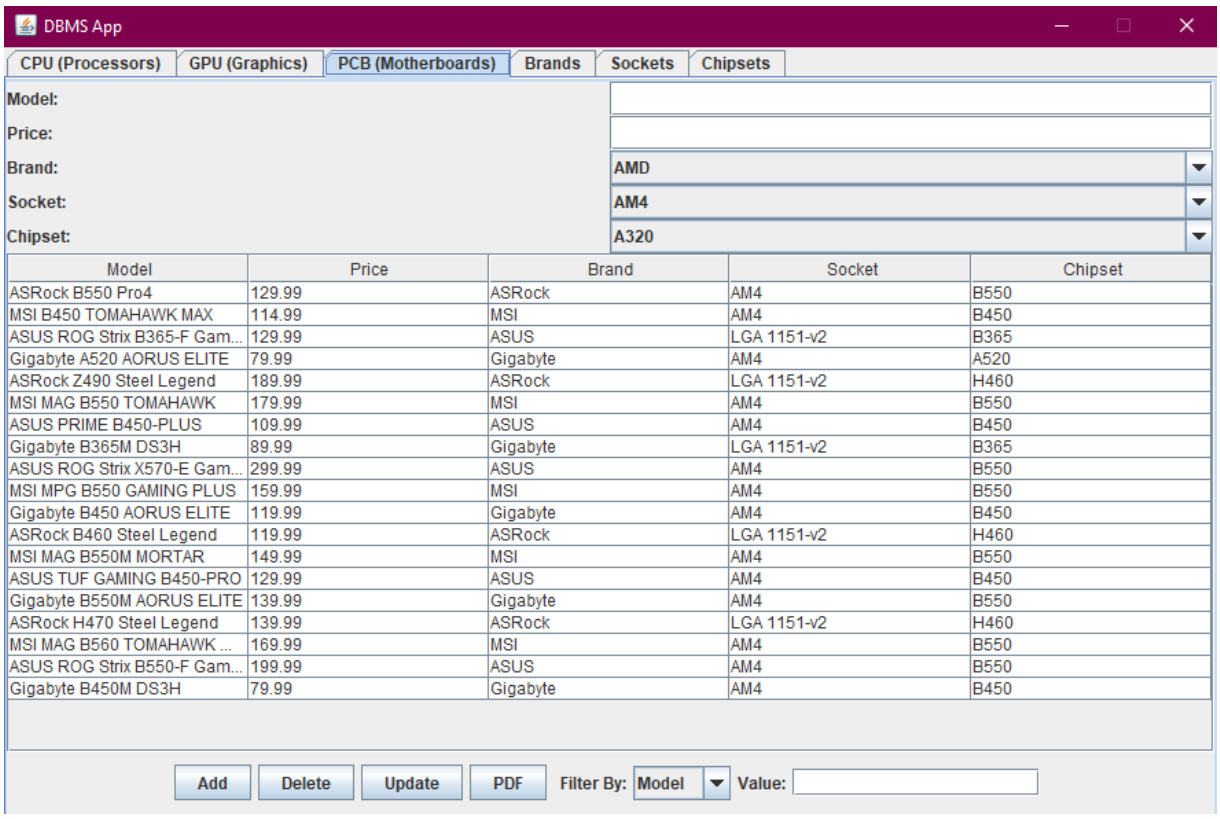


Рисунок 5. Интерфейс программы. Таблица материнских плат

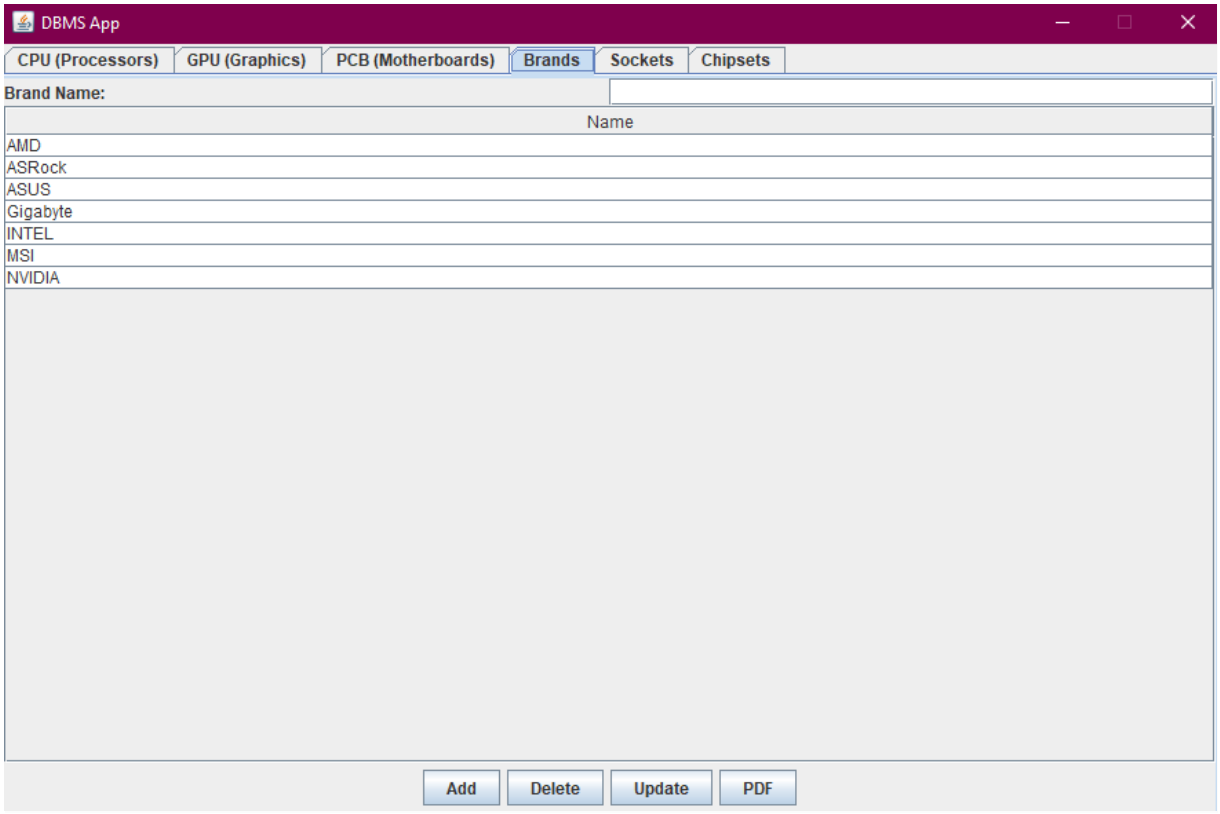


Рисунок 6. Интерфейс программы. Таблица брендов

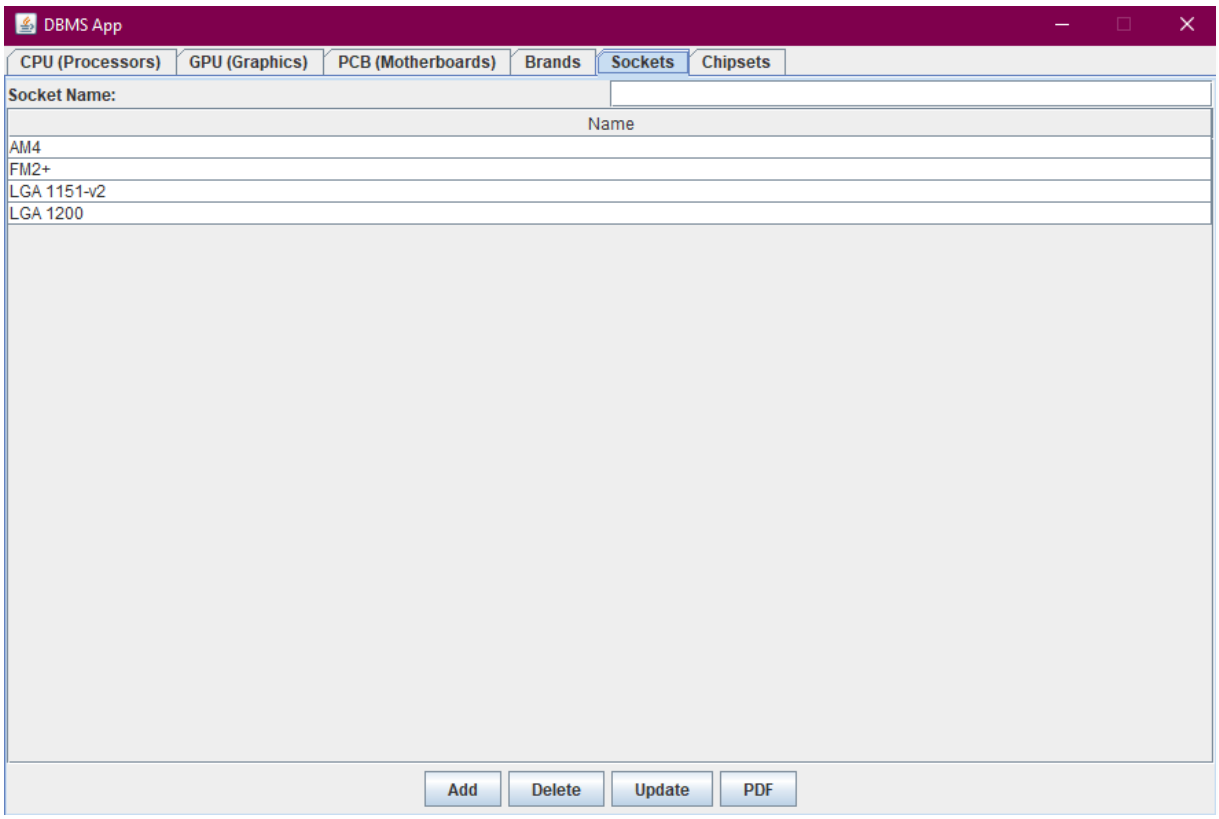


Рисунок 7. Интерфейс программы. Таблица сокетов

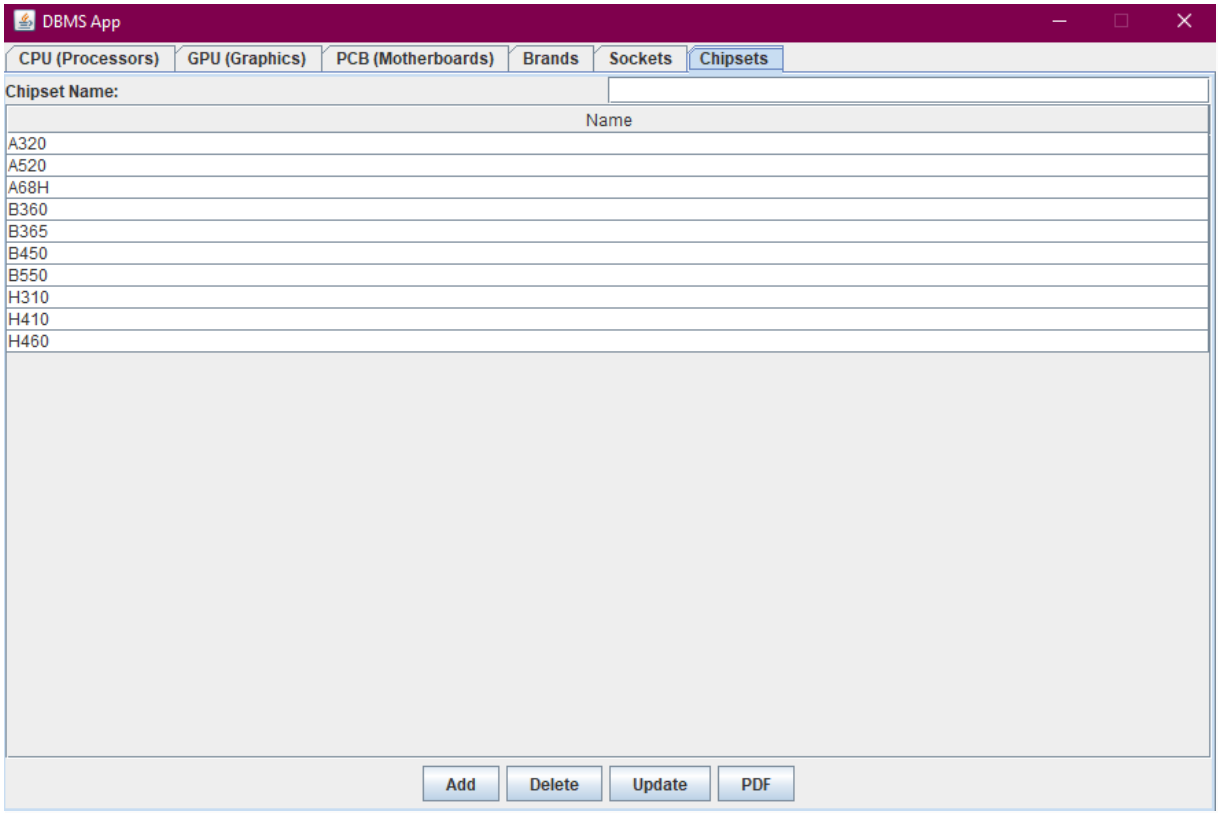


Рисунок 8. Интерфейс программы. Таблица чипсетов

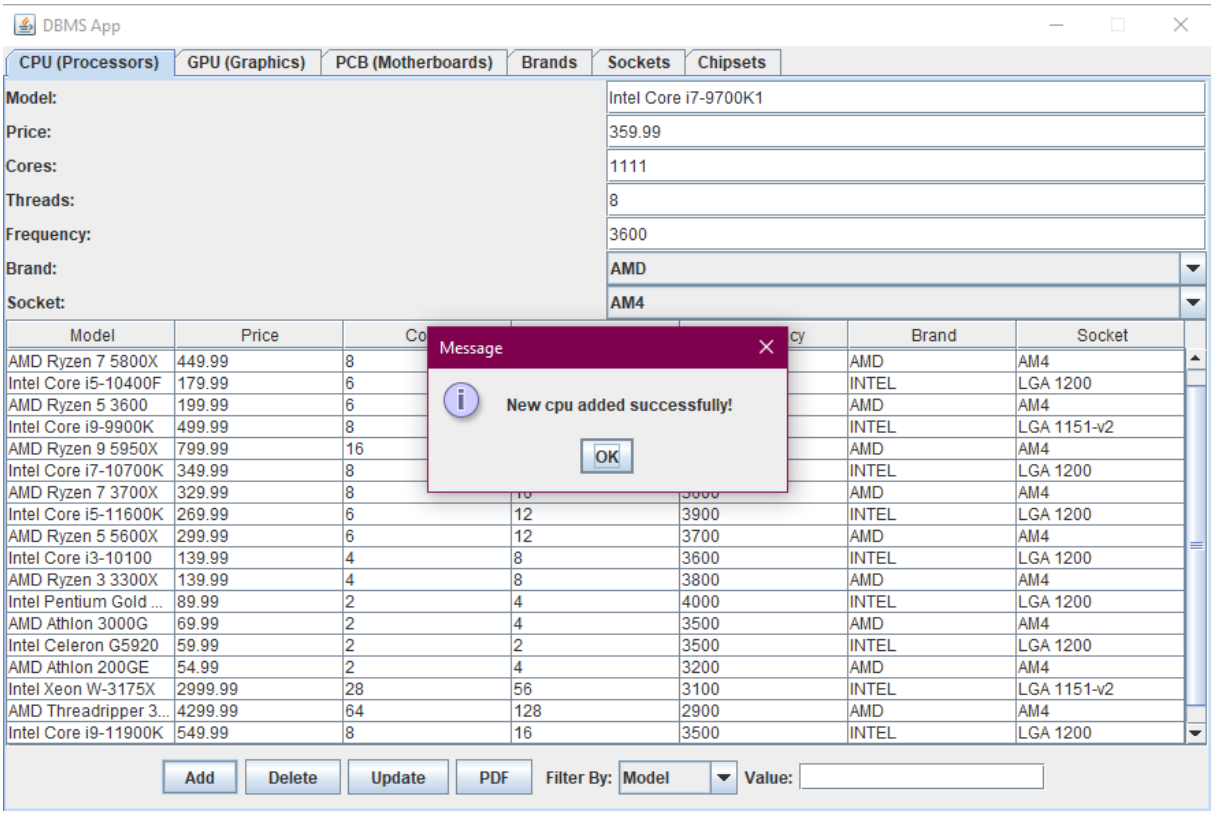


Рисунок 9. Пример добавления новой записи

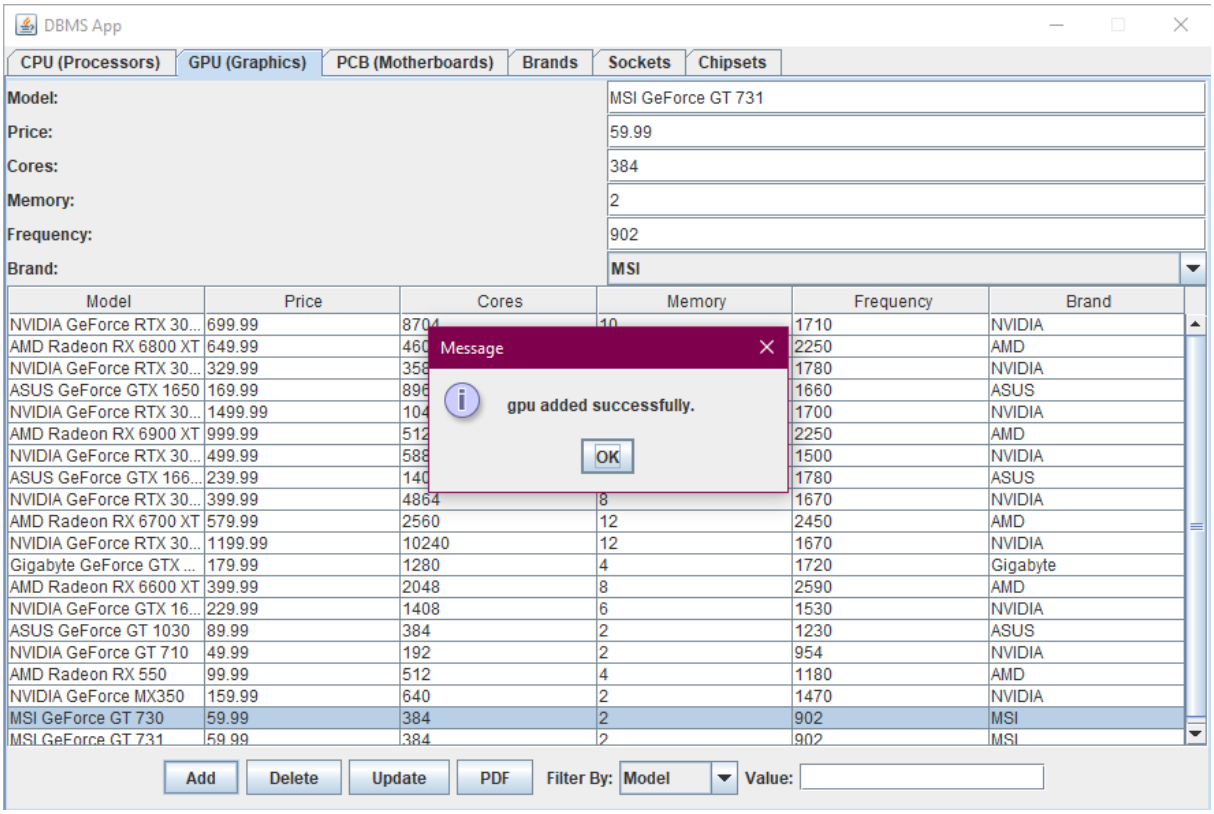


Рисунок 10. Пример добавления новой записи

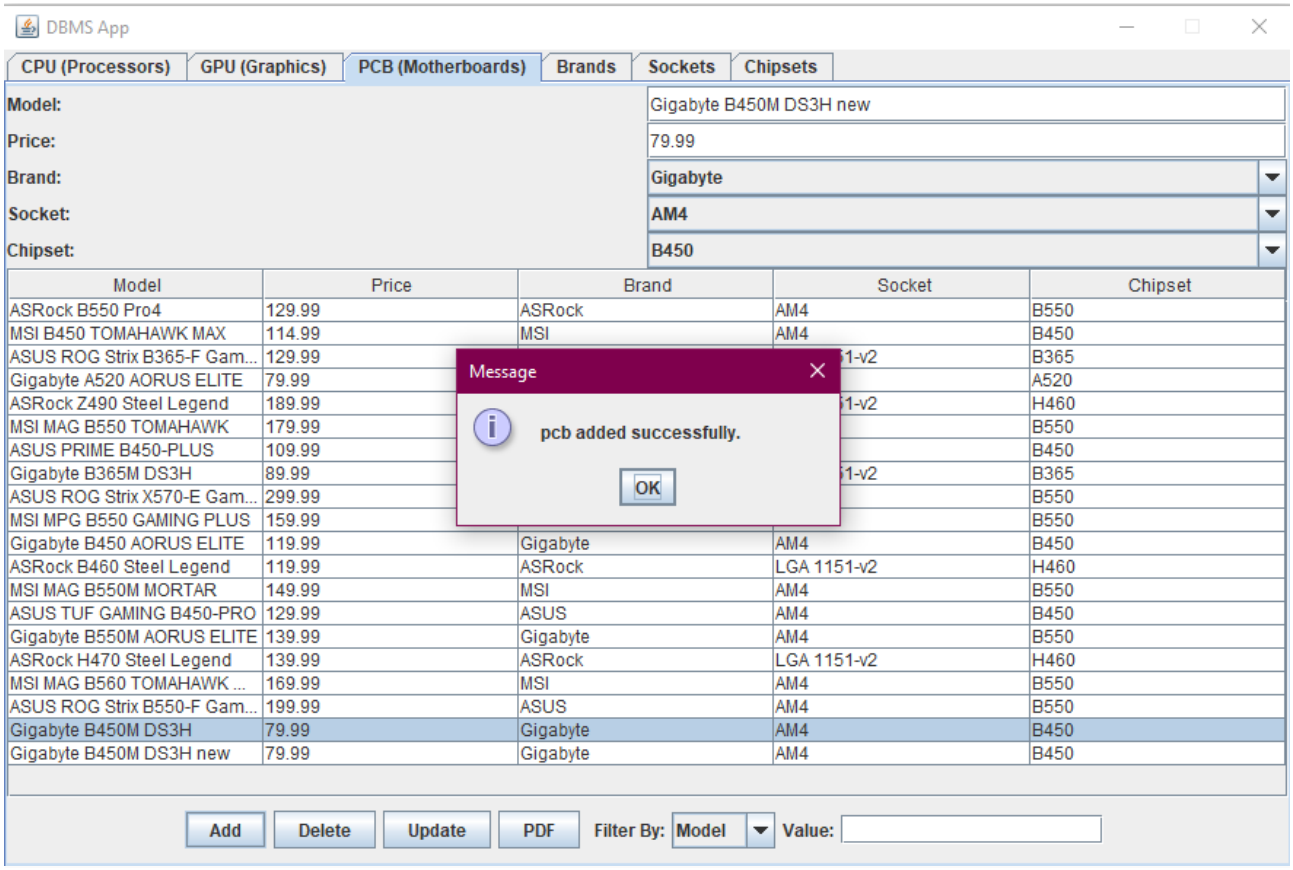


Рисунок 11. Пример добавления новой записи

Рисунок 12. Пример обновления элемента ComboBox

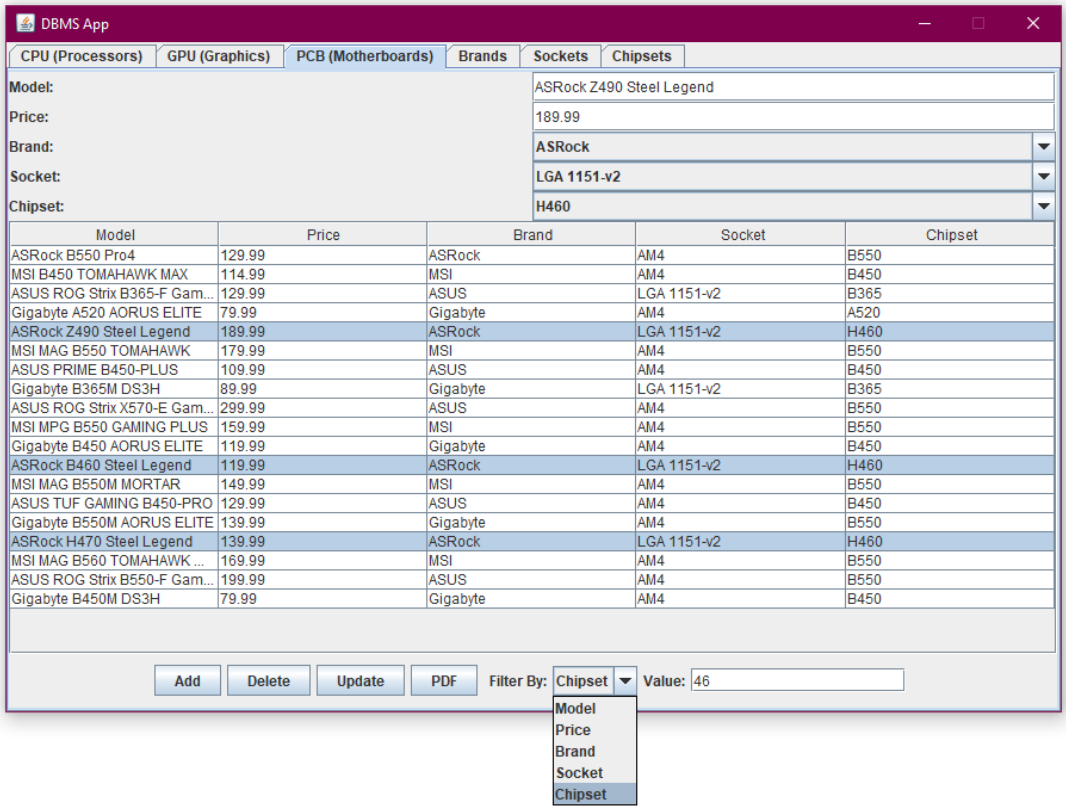


Рисунок 13. Пример работы фильтра по полю Chipset

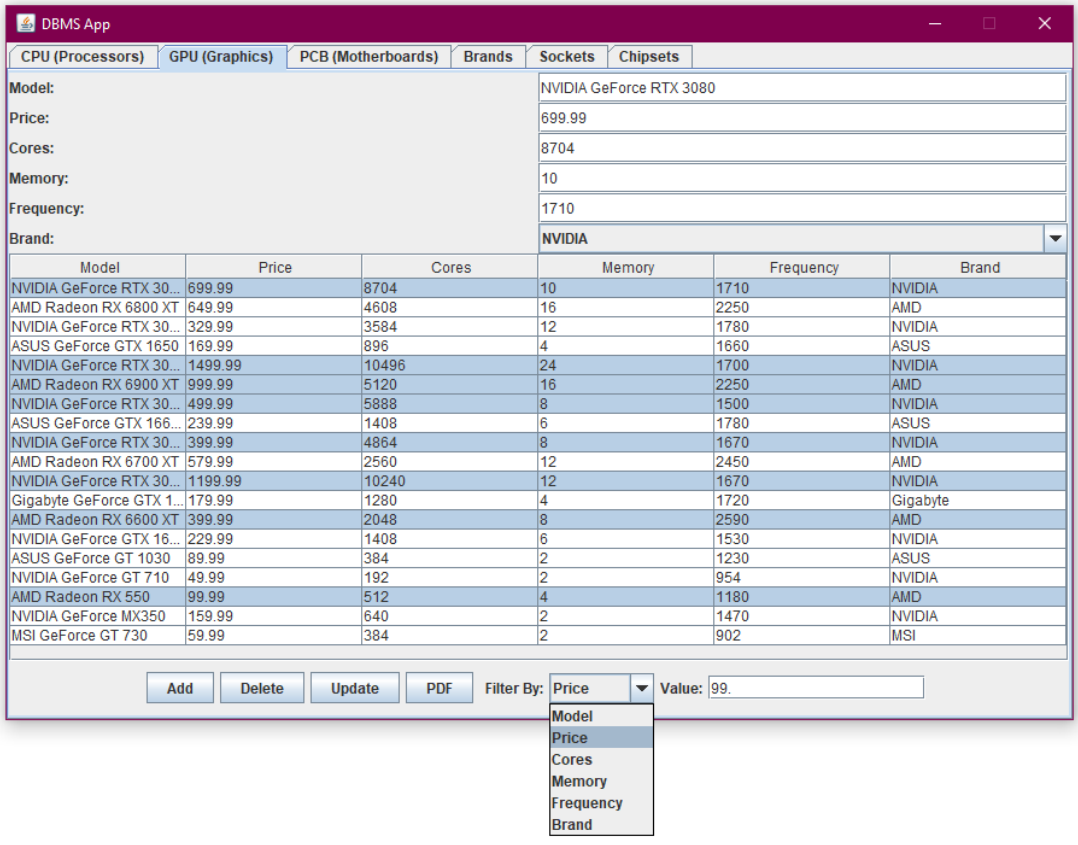


Рисунок 13. Пример работы фильтра по полю Price

4. ГЕНЕРАЦИЯ ОТЧЕТОВ

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

Model	Price	Cores	Threads	Frequency	Brand	Socket
Intel Core i7-9700K	359.99	8	8	3600	INTEL	LGA 1151-v2
AMD Ryzen 7 5800X	449.99	8	16	3800	AMD	AM4
Intel Core i5-10400F	179.99	6	12	2900	INTEL	LGA 1200
AMD Ryzen 5 3600	199.99	6	12	3600	AMD	AM4
Intel Core i9-9900K	499.99	8	16	3600	INTEL	LGA 1151-v2
AMD Ryzen 9 5950X	799.99	16	32	3400	AMD	AM4
Intel Core i7-10700K	349.99	8	16	3800	INTEL	LGA 1200
AMD Ryzen 7 3700X	329.99	8	16	3600	AMD	AM4
Intel Core i5-11600K	269.99	6	12	3900	INTEL	LGA 1200
AMD Ryzen 5 5600X	299.99	6	12	3700	AMD	AM4
Intel Core i3-10100	139.99	4	8	3600	INTEL	LGA 1200
AMD Ryzen 3 3300X	139.99	4	8	3800	AMD	AM4
Intel Pentium Gold G6400	89.99	2	4	4000	INTEL	LGA 1200
AMD Athlon 3000G	69.99	2	4	3500	AMD	AM4
Intel Celeron G5920	59.99	2	2	3500	INTEL	LGA 1200
AMD Athlon 200GE	54.99	2	4	3200	AMD	AM4
Intel Xeon W-3175X	2999.99	28	56	3100	INTEL	LGA 1151-v2
AMD Threadripper 3990X	4299.99	64	128	2900	AMD	AM4
Intel Core i9-11900K	549.99	8	16	3500	INTEL	LGA 1200
Intel Core i7-9700K1	359.99	1111	8	3600	AMD	AM4

Рисунок 11. Отчет по процессорам

Model	Price	Cores	Threads	Frequency	Brand
NVIDIA GeForce RTX 3080	699.99	8704	10	1710	NVIDIA
AMD Radeon RX 6800 XT	649.99	4608	16	2250	AMD
NVIDIA GeForce RTX 3060	329.99	3584	12	1780	NVIDIA
ASUS GeForce GTX 1650	169.99	896	4	1660	ASUS
NVIDIA GeForce RTX 3090	1499.99	10496	24	1700	NVIDIA
AMD Radeon RX 6900 XT	999.99	5120	16	2250	AMD
NVIDIA GeForce RTX 3070	499.99	5888	8	1500	NVIDIA
ASUS GeForce GTX 1660 Super	239.99	1408	6	1780	ASUS
NVIDIA GeForce RTX 3060 Ti	399.99	4864	8	1670	NVIDIA
AMD Radeon RX 6700 XT	579.99	2560	12	2450	AMD
NVIDIA GeForce RTX 3080 Ti	1199.99	10240	12	1670	NVIDIA
Gigabyte GeForce GTX 1650 Super	179.99	1280	4	1720	Gigabyte
AMD Radeon RX 6600 XT	399.99	2048	8	2590	AMD
NVIDIA GeForce GTX 1660	229.99	1408	6	1530	NVIDIA
ASUS GeForce GT 1030	89.99	384	2	1230	ASUS
NVIDIA GeForce GT 710	49.99	192	2	954	NVIDIA
AMD Radeon RX 550	99.99	512	4	1180	AMD
NVIDIA GeForce MX350	159.99	640	2	1470	NVIDIA
MSI GeForce GT 730	59.99	384	2	902	MSI
MSI GeForce GT 731	59.99	384	2	902	MSI

Рисунок 12. Отчет по видеокартам

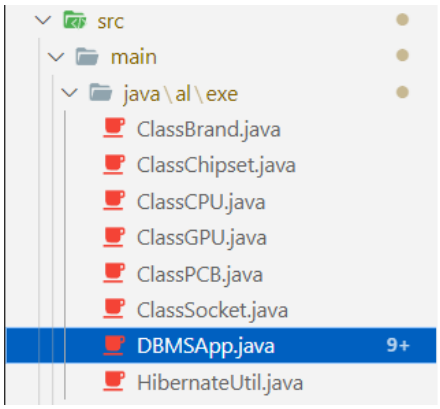
Model	Price	Brand	Socket	Chipset
ASRock B550 Pro4	129.99	ASRock	AM4	B550
MSI B450 TOMAHAWK MAX	114.99	MSI	AM4	B450
ASUS ROG Strix B365-F Gaming	129.99	ASUS	LGA 1151-v2	B365
Gigabyte A520 AORUS ELITE	79.99	Gigabyte	AM4	A520
ASRock Z490 Steel Legend	189.99	ASRock	LGA 1151-v2	H460
MSI MAG B550 TOMAHAWK	179.99	MSI	AM4	B550
ASUS PRIME B450-PLUS	109.99	ASUS	AM4	B450
Gigabyte B365M DS3H	89.99	Gigabyte	LGA 1151-v2	B365
ASUS ROG Strix X570-E Gaming	299.99	ASUS	AM4	B550
MSI MPG B550 GAMING PLUS	159.99	MSI	AM4	B550
Gigabyte B450 AORUS ELITE	119.99	Gigabyte	AM4	B450
ASRock B460 Steel Legend	119.99	ASRock	LGA 1151-v2	H460
MSI MAG B550M MORTAR	149.99	MSI	AM4	B550
ASUS TUF GAMING B450-PRO	129.99	ASUS	AM4	B450
Gigabyte B550M AORUS ELITE	139.99	Gigabyte	AM4	B550
ASRock H470 Steel Legend	139.99	ASRock	LGA 1151-v2	H460
MSI MAG B560 TOMAHAWK WIFI	169.99	MSI	AM4	B550
ASUS ROG Strix B550-F Gaming	199.99	ASUS	AM4	B550
Gigabyte B450M DS3H	79.99	Gigabyte	AM4	B450
Gigabyte B450M DS3H new	79.99	Gigabyte	AM4	B450

Рисунок 13. Отчет по материнским платам

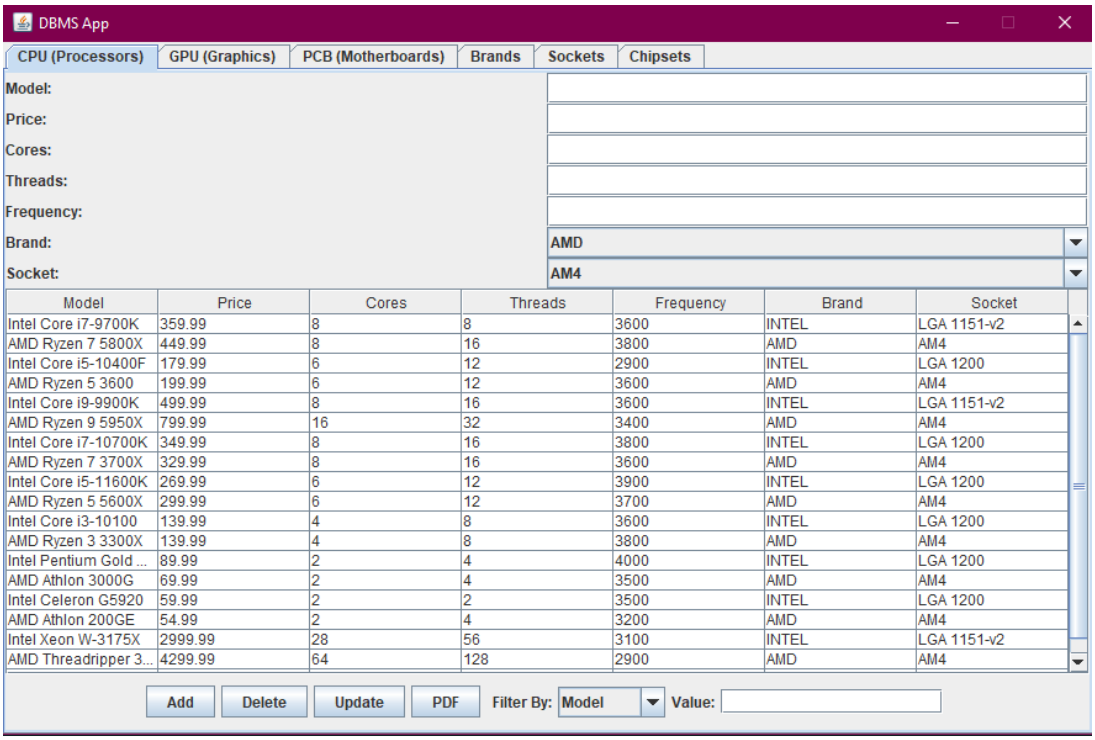
5. КРАТКОЕ РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

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.



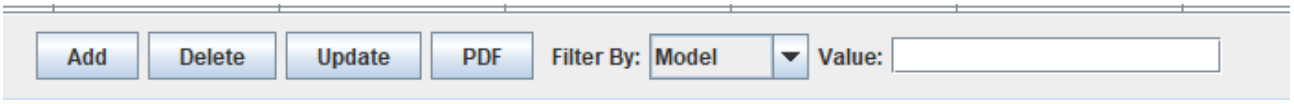
2. Интерфейс пользователя

- Главное окно приложения состоит из вкладок для каждой таблицы базы данных "pc".

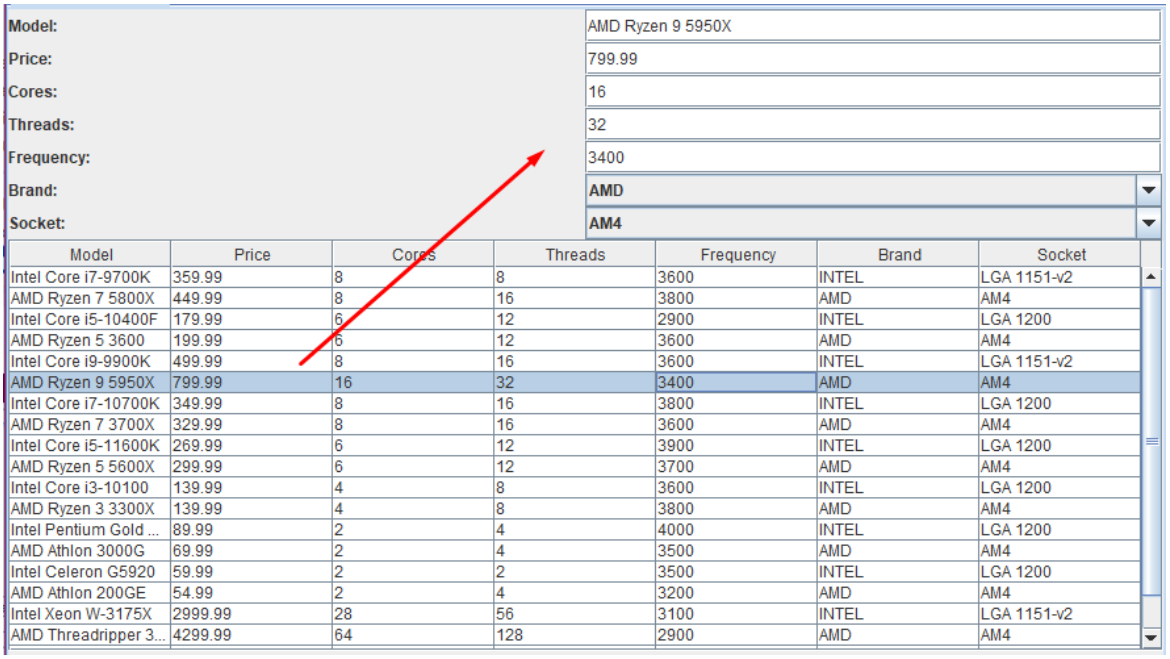
- В каждой вкладке таблица отображает данные из соответствующей таблицы базы данных.

Model	Price	Cores	Threads	Frequency	Brand	Socket
Intel Core i7-9700K	359.99	8	8	3600	INTEL	LGA 1151-v2
AMD Ryzen 7 5800X	449.99	8	16	3800	AMD	AM4
Intel Core i5-10400F	179.99	6	12	2900	INTEL	LGA 1200
AMD Ryzen 5 3600	199.99	6	12	3600	AMD	AM4
Intel Core i9-9900K	499.99	8	16	3600	INTEL	LGA 1151-v2
AMD Ryzen 9 5950X	799.99	16	32	3400	AMD	AM4
Intel Core i7-10700K	349.99	8	16	3800	INTEL	LGA 1200
AMD Ryzen 7 3700X	329.99	8	16	3600	AMD	AM4
Intel Core i5-11600K	269.99	6	12	3900	INTEL	LGA 1200
AMD Ryzen 5 5600X	299.99	6	12	3700	AMD	AM4
Intel Core i3-10100	139.99	4	8	3600	INTEL	LGA 1200
AMD Ryzen 3 3300X	139.99	4	8	3800	AMD	AM4
Intel Pentium Gold ...	89.99	2	4	4000	INTEL	LGA 1200
AMD Athlon 3000G	69.99	2	4	3500	AMD	AM4
Intel Celeron G5920	59.99	2	2	3500	INTEL	LGA 1200
AMD Athlon 200GE	54.99	2	4	3200	AMD	AM4
Intel Xeon W-3175X	2999.99	28	56	3100	INTEL	LGA 1151-v2
AMD Threadripper 3...	4299.99	64	128	2900	AMD	AM4

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

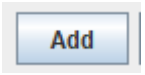


- При выборе строки в таблице срабатывает событие выбора, которое можно использовать для дополнительной логики.



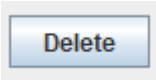
3. Добавление данных

- Чтобы добавить запись в таблицу, выберите соответствующую вкладку и нажмите кнопку "Add".

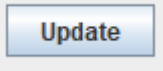


- В появившемся диалоговом окне введите значения для всех полей и нажмите кнопку "ОК".
- Запись будет добавлена в таблицу и отображена в графическом интерфейсе.

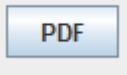
4. Удаление данных
    - Чтобы удалить запись из таблицы, выберите соответствующую вкладку и выделите нужную строку в таблице.
    - Нажмите кнопку "Delete".



    - Запись будет удалена из таблицы и удалится из графического интерфейса.
5. Обновление данных
    - Чтобы обновить запись в таблице, выберите соответствующую вкладку и выделите нужную строку в таблице.
    - Нажмите кнопку "Update".

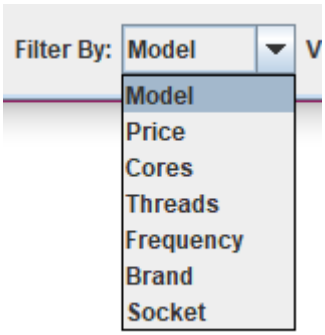


    - В появившемся диалоговом окне введите новые значения для полей и нажмите кнопку "ОК".
    - Запись будет обновлена в таблице и изменится в графическом интерфейсе.
6. Так же, чтобы выгрузить текущую таблицу в виде документа, необходимо нажать на кнопку "PDF".



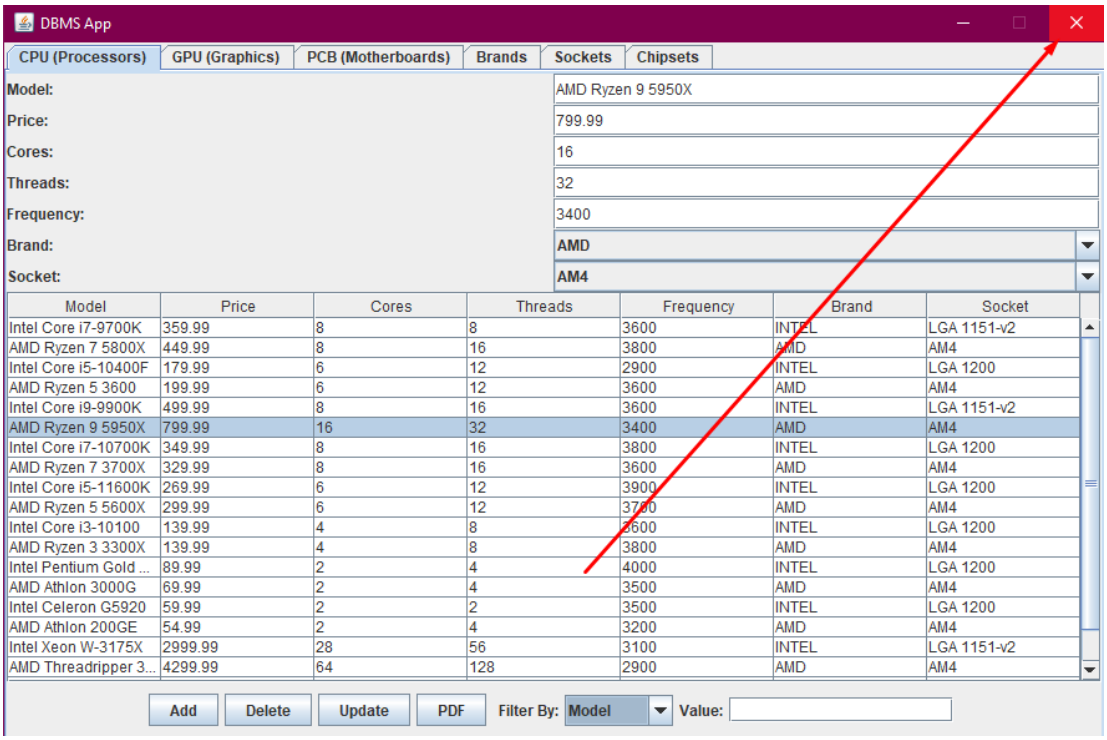
7. Для использования функции фильтрации таблиц

- выберите необходимое поле, по которому вы хотите произвести фильтрацию данных:



- затем введите в рядом расположенное поле необходимый текст для отбора данных

8. 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, "CAN'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, "CAN'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, "CAN'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, "CAN'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, "CAN'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, "CAN'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
```

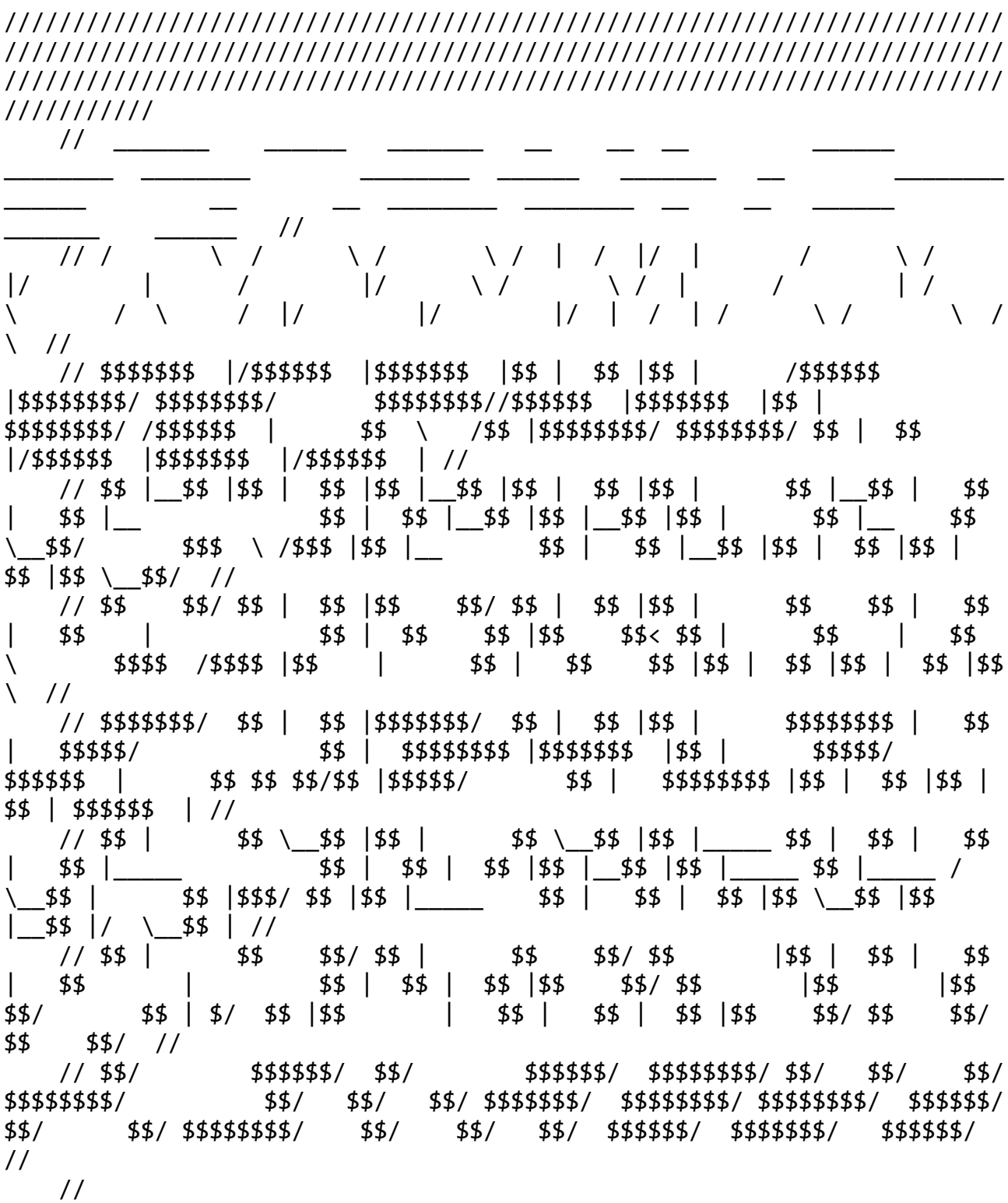
```
        BRANDTABLE.GETSELECTIONMODEL().ADDLISTSELECTIONLISTENER(NEW
LISTSELECTIONLISTENER() {
            @OVERRIDE
            PUBLIC VOID VALUECHANGED(LISTSELECTIONEVENT E)
            {
                INT SELECTEDROW = BRANDTABLE.GETSELECTEDROW();
                IF (SELECTEDROW >= 0)
                {
                    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();
            }
        });
    }
}
```