SE 311 Term Project – Livestock Farm

1 Group Information

- Yiğit Can Dündar
  ID: 20130601019
  E-Mail: yigitcandundar95@gmail.com

- Selin Önal
  ID: 20120603042
  E-Mail: onalselinn@gmail.com

2 Patterns Used & Thought Processes

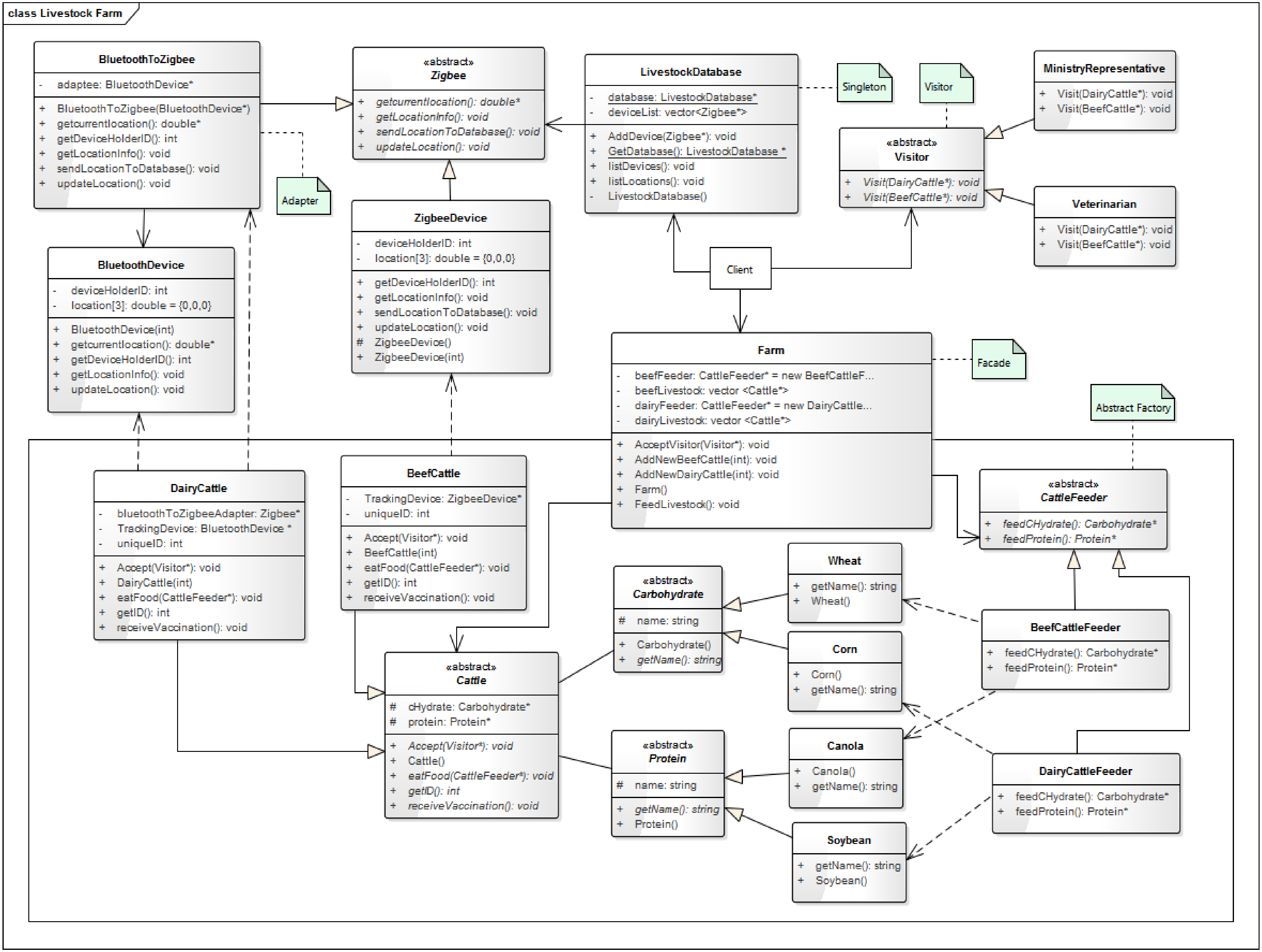
1) Singleton Pattern: This pattern was selected with the notion of having a single database that could later be used by other classes/clients without having to create a new database instance.

2) Façade Pattern: This pattern was selected with the notion of providing the client an interface that could be used to handle Farm specific tasks such as: adding new cattle animals to cattle lists, controlling cattle feeder relations and also visitor relations, without having to manually keep track of every façade covered classes and relations.

3) Abstract Factory Pattern: This pattern was selected so that the feeding constraints of each cattle would be separated into their own food product and feeder categories. That is later used by the Façade client to complete the feeding process.

4) Adapter Pattern: This pattern was selected due to the database requiring only Zigbee signals. Since there is another signal transmitter that transmits Bluetooth signals, this pattern serves as an adapter from Bluetooth to Zigbee for the database to register Bluetooth signals.

5) Visitor Pattern: This pattern was selected since the Veterinary Physician and The Ministry of Food, Agriculture and Livestock need to visit the cattle animals to perform specific tasks through the Farm Façade.


class Livestock Farm
BluetoothToZigbee
- adaptee: BluetoothDevice*
+ BluetoothToZigbee(BluetoothDevice*)
+ getcurrentlocation(): double*
+ getDeviceHolderID(): int
+ getLocationInfo(): void
+ sendLocationToDatabase(): void
+ updateLocation(): void
Adapter
«abstract»
Zigbee
+ getcurrentlocation(): double*
+ getLocationInfo(): void
+ sendLocationToDatabase(): void
+ updateLocation(): void
LivestockDatabase
- database: LivestockDatabase*
- deviceList: vector<Zigbee*>
+ AddDevice(Zigbee*): void
+ GetDatabase(): LivestockDatabase *
+ listDevices(): void
+ listLocations(): void
- LivestockDatabase()
Singleton
Visitor
MinistryRepresentative
+ Visit(DairyCattle*): void
+ Visit(BeefCattle*): void
«abstract»
Visitor
+ Visit(DairyCattle*): void
+ Visit(BeefCattle*): void
Veterinarian
+ Visit(DairyCattle*): void
+ Visit(BeefCattle*): void
BluetoothDevice
- deviceHolderID: int
- location[3]: double = {0,0,0}
+ BluetoothDevice(int)
+ getcurrentlocation(): double*
+ getDeviceHolderID(): int
+ getLocationInfo(): void
+ updateLocation(): void
ZigbeeDevice
- deviceHolderID: int
- location[3]: double = {0,0,0}
+ getDeviceHolderID(): int
+ getLocationInfo(): void
+ sendLocationToDatabase(): void
+ updateLocation(): void
# ZigbeeDevice()
+ ZigbeeDevice(int)
Client
Farm
- beefFeeder: CattleFeeder* = new BeefCattleF...
- beefLivestock: vector <Cattle*>
- dairyFeeder: CattleFeeder* = new DairyCattle...
- dairyLivestock: vector <Cattle*>
+ AcceptVisitor(Visitor*): void
+ AddNewBeefCattle(int): void
+ AddNewDairyCattle(int): void
+ Farm()
+ FeedLivestock(): void
Facade
Abstract Factory
DairyCattle
- bluetoothToZigbeeAdapter: Zigbee*
- TrackingDevice: BluetoothDevice *
- uniqueID: int
+ Accept(Visitor*): void
+ DairyCattle(int)
+ eatFood(CattleFeeder*): void
+ getID(): int
+ receiveVaccination(): void
BeefCattle
- TrackingDevice: ZigbeeDevice*
- uniqueID: int
+ Accept(Visitor*): void
+ BeefCattle(int)
+ eatFood(CattleFeeder*): void
+ getID(): int
+ receiveVaccination(): void
«abstract»
CattleFeeder
+ feedCHydrate(): Carbohydrate*
+ feedProtein(): Protein*
Wheat
+ getName(): string
+ Wheat()
«abstract»
Carbohydrate
# name: string
+ Carbohydrate()
+ getName(): string
BeefCattleFeeder
+ feedCHydrate(): Carbohydrate*
+ feedProtein(): Protein*
Corn
+ Corn()
+ getName(): string
«abstract»
Cattle
# cHydrate: Carbohydrate*
# protein: Protein*
+ Accept(Visitor*): void
+ Cattle()
+ eatFood(CattleFeeder*): void
+ getID(): int
+ receiveVaccination(): void
Canola
+ Canola()
+ getName(): string
«abstract»
Protein
# name: string
+ getName(): string
+ Protein()
DairyCattleFeeder
+ feedCHydrate(): Carbohydrate*
+ feedProtein(): Protein*
Soybean
+ getName(): string
+ Soybean()

3 CLASS EXPLANATIONS

LivestockDatabase: A Singleton class that holds the location information of all farm animals using Zigbee Devices.

- GetDatabase(): Returns the unique instance. If the unique instance is NULL instantiates the Database instance.
- listLocations(): Prints out all animal's location information. If the location's x or z values are above 15 prints out a Warning message to notify the user.

Zigbee: An Abstract Class that provides an interface for the Adapter and ZigbeeDevice classes.

- updateLocation(): Randomly selects values within the range[0,19] for x and z axes of the animal's location.

ZigbeeDevice: A Concrete Zigbee Class that is used for tracking Beef Cattle locations.

BluetoothDevice: An Adaptee Class that is used for tracking Dairy Cattle locations.

BluetoothToZigbee: A Concrete Zigbee Adapter Class that is used for adapting the Bluetooth device to Zigbee so that the Database can process it correctly.

- BluetoothToZigbee(BluetoothDevice* bTooth): The constructor that initializes the adaptee.

Carbohydrate: An Abstract Product Class that serves as an interface for Wheat and Corn Classes.

Wheat, Corn: Two Concrete Carbohydrate Product Classes that are used for feeding the farm animals.

Protein: An Abstract Product Class that serves as an interface for Canola and Soybean Classes.

Canola, Soybean: Two Concrete Protein Product Classes that are used for feeding the farm animals.

CattleFeeder: An Abstract Factory Class that serves as an interface for BeefCattleFeeder and DairyCattleFeeder Classes.

- feedCHydrate(): A Factory Method that creates Carbohydrate objects.
- feedProtein(): A Factory Method that creates Protein objects.

BeefCattleFeeder: A Concrete CattleFeeder Class that is used to fulfill Beef Cattle feeding constraints.

DairyCattleFeeder: A Concrete CattleFeeder Class that is used to fulfill Dairy Cattle feeding constraints.

Cattle: An Abstract Class that serves as an interface for DairyCattle and BeefCattle classes.

- Accept(Visitor* visitor): The method that accepts visitors.
- eatFood(CattleFeeder* feeder): The Method that uses the CattleFeeder object to feed the animal.

DairyCattle: A Concrete Cattle Class that represents Dairy Farm animals. They are tracked using Bluetooth Devices.

BeefCattle: A Concrete Cattle Class that represents Beef Farm animals. They are tracked using Zigbee Devices.

Visitor: An Abstract Visitor Class that serves as an interface for Veterinarian and MinistryRepresentative Classes.

- Visit(DairyCattle* cattle): The Visit Method that is used to visit Dairy Cattle.
- Visit(BeefCattle* cattle): The Visit Method that is used to visit Beef Cattle.

Veterinarian: A Concrete Visitor Class that visits farm animals to give them vaccination.

MinistryRepresentative: A Concrete Visitor Class that visits farm animals to check if they have an ear tag or not.

Farm: A Façade Class that keeps a list of all farm animals, controls feeders and accepts visitors for all animals.

- FeedLivestock(): The Method that feeds all farm animals using appropriate CattleFeeders.
- AcceptVisitor(Visitor* visitor): The Method that accepts the visitor for all farm animals.
- AddNewDairyCattle(int id): Adds a new Dairy Cattle to the list with unique ID “id”.
- AddNewBeefCattle(int id): Adds a new Dairy Cattle to the list with unique ID “id”.

4 CODE

4.1 HEADER

```
#pragma once
#include <iostream>
#include <vector>
#include <string>
#include <ctime>
#include<cstdlib>
using namespace std;
//Forward declarations for Zigbee, Cattle, DairyCattle and BeefCattle classes.
class Zigbee;
class Cattle;
class DairyCattle;
class BeefCattle;
//Singleton Database that holds the Location information of all farm animals using Zigbee signals.
class LivestockDatabase {
private:
        LivestockDatabase() {}; //Private constructor.
        vector<Zigbee*> deviceList; //The list that contains ONLY Zigbee Devices.
        static LivestockDatabase* database; //Singleton unique instance.
public:
        //Method that Adds the ZBDevice to the deviceList.
        void AddDevice(Zigbee* ZBDevice) {
                deviceList.push_back(ZBDevice);
        }
        //Database Method that lists all the connected devices.
        void listDevices();
        //Database Method that lists the location info for each monitored cattle.
        void listLocations();
        //Accessor method for the unique instance.
        static LivestockDatabase* GetDatabase() {
                if (database == NULL) {
                        database = new LivestockDatabase();
                }
                return database;
        }
};

//Abstract Zigbee class that provides an interface for Zigbee Devices and Bluetooth adapter.
class Zigbee {
public:
        //Virtual methods that MUST be overridden by subclasses.
        virtual void sendLocationToDatabase() = 0;
        virtual void printLocationInfo() = 0;
        virtual int getDeviceHolderID() = 0;
        virtual void updateLocation() = 0;
        virtual double * getCurrentLocation() = 0;
};

//Concrete Zigbee Class that is used on Beef Cattle.
class ZigbeeDevice :public Zigbee {
private:
        int deviceHolderID; //The ID of the cattle that this device is attached to.
        double location[3] = { 0,1,0 }; //Location information of the cattle.
public:
        //Unique ID is set and the device info is sent to the Database
        ZigbeeDevice(int id) { deviceHolderID = id; sendLocationToDatabase(); }
        //Accessor method for the private data member "deviceHolderID".
        int getDeviceHolderID() { return deviceHolderID; }
        //Updates the location of the cattle based on a random number within the range [0,19], for x and z axes.
        void updateLocation() {
                double x = rand() % 20;
```

```
		location[0] = x;
		x = rand() % 20;
		location[2] = x;
	}
	//A pretty print method for the location info.
	void printLocationInfo() {
		cout << "(" << location[0] << "," << location[1] << "," << location[2] << ")";
	}
	//Accessor method for the private data member "location".
	double * getCurrentLocation() {
		return location;
	}
	//Method for sending the device info to the Database.
	void sendLocationToDatabase() {
		LivestockDatabase::GetDatabase()->AddDevice(this);
	}
};

//Bluetooth Class that is used on Dairy Cattle.
class BluetoothDevice {
private:
	int deviceHolderID; //The ID of the cattle that this device is attached to.
	double location[3] = { 0,1,0 }; //Location information of the cattle.
public:
	//Unique ID is set.
	BluetoothDevice(int id) { deviceHolderID = id; }
	//Accessor method for the private data member "deviceHolderID".
	int getDeviceHolderID() { return deviceHolderID; }
	//Updates the location of the cattle based on a random number within the range [0,19], for x and z axes.
	void updateLocation() {
		double x = rand() % 20;
		location[0] = x;
		x = rand() % 20;
		location[2] = x;
	}
	//A pretty print method for the location info.
	void printLocationInfo() {
		cout << "(" << location[0] << "," << location[1] << "," << location[2] << ")";
	}
	//Accessor method for the private data member "location".
	double * getCurrentLocation() { return location; }
};

//Bluetooth signal to Zigbee signal Adapter
class BluetoothToZigbee :public Zigbee {
private:
	BluetoothDevice* adaptee; //Adaptee Bluetooth Device instance.
public:
	//Adaptee is set and the adapters information is sent to the database.
	BluetoothToZigbee(BluetoothDevice* bTooth) { adaptee = bTooth; sendLocationToDatabase(); };

	//Adapter method to print the location info from the adaptee.
	void printLocationInfo() { adaptee->printLocationInfo(); }
	//Adapter method to update the location info from the adaptee
	void updateLocation() { adaptee->updateLocation(); }
	//Sends the adapter information to the Database.
	void sendLocationToDatabase() { LivestockDatabase::GetDatabase()->AddDevice(this);  }
	//Adapter method to get the deviceHolderID from the adaptee.
	int getDeviceHolderID() { return adaptee->getDeviceHolderID(); }
	//Adapter method to get the location info from the adaptee.
	double * getCurrentLocation() { return adaptee->getCurrentLocation(); }
};

//Abstract Product Class Carbohydrate
class Carbohydrate {
public:
        Carbohydrate() {}
        //Virtual method that must be overridden by subclasses.
        virtual string getName() = 0;
protected:
        //Protected data member that holds the name of the product.
        string name;
};

//Concrete Carbohydrate product class Corn
class Corn : public Carbohydrate {
public:
        //Name is set.
        Corn() { name = "Corn"; }
        //Accessor method for the protected data member "name".
        string getName() { return name; }
};

//Concrete Carbohydrate product class Wheat
class Wheat :public Carbohydrate {
public:
        Wheat() { name = "Wheat"; }
        string getName() { return name; }
};

//Abstract Product Class Protein
class Protein {
public:
        Protein() {}
        virtual string getName() = 0;
protected:
        string name;
};

//Concrete Protein product class Soybean
class Soybean :public Protein {
public:
        Soybean() { name = "Soybean"; }
        string getName() { return name; }
};

//Concrete Protein product class Canola
class Canola : public Protein {
public:
        Canola() { name = "Canola"; }
        string getName() { return name; }
};

//Abstract Factory Class CattleFeeder
class CattleFeeder {
public:
        //Virtual Methods that must be overridden by subclasses.
        virtual Carbohydrate* feedCHydrate() = 0;
        virtual Protein* feedProtein() = 0;
};

//Concrete Factory Class Dairy Feeder
class DairyCattleFeeder : public CattleFeeder {
public:
        //Factory method that returns the appropriate Carbohydrate object.
        Carbohydrate* feedCHydrate() {
                return new Corn();
        }

        //Factory method that returns the appropriate Protein object.
        Protein* feedProtein() {
                return new Soybean();
```

```
		}
};

//Concrete Factory Class Beef Feeder
class BeefCattleFeeder : public CattleFeeder {
public:
		//Factory method that returns the appropriate Carbohydrate object.
		Carbohydrate* feedCHydrate() {
				return new Wheat();
		}
		//Factory method that returns the appropriate Carbohydrate object.
		Protein* feedProtein() {
				return new Canola();
		}
};

//Abstract Visitor Class
class Visitor {
public:
		//Virtual methods that must be overridden by subclasses.
		virtual void Visit(DairyCattle* cattle) = 0;
		virtual void Visit(BeefCattle* cattle) = 0;
};

//Abstract Cattle Class
class Cattle {
public:
		Cattle() {}
		//Virtual methods that must be overridden by subclasses.
		virtual void eatFood(CattleFeeder* feeder) = 0;
		virtual void Accept(Visitor* visitor) = 0;
		virtual int getID() = 0;
		virtual void receiveVaccination() = 0;
protected:
		Carbohydrate* cHydrate; //Carbohydrate product instance.
		Protein* protein; //Protein product instance.
};

//Concrete Dairy Cattle Class. Tracked using Bluetooth. That also serves as the Abstract Factory methods client.
class DairyCattle :public Cattle {
private:
		int uniqueID; //Unique identifier.
		BluetoothDevice* TrackingDevice; //Tracker for dairy cattle.
		Zigbee* bluetoothToZigbeeAdapter; //Bluetooth to Zigbee adapter.
public:
		//Unique id is set, Bluetooth tracker is initialized, adapter is initialized with the tracker.
		DairyCattle(int id) {
				uniqueID = id;
				TrackingDevice = new BluetoothDevice(uniqueID);
				bluetoothToZigbeeAdapter = new BluetoothToZigbee(TrackingDevice);
		}

		//Method that simulates eating habits of dairy cattle.
		void eatFood(CattleFeeder* feeder) {
				cHydrate = feeder->feedCHydrate();
				protein = feeder->feedProtein();
				cout << "Dairy Cattle #" << uniqueID << " is eating " << cHydrate->getName() << " and " << protein->getName() << endl;
		}

		//Visitor accept method.
		void Accept(Visitor * visitor) {
				visitor->Visit(this);
		}
		//Accessor method for the private data member "uniqueID".
		int getID() {
				return uniqueID;
		}

		//Method that the Veterinarian visitor can use.
		void receiveVaccination() {
			cout << "Dairy Cattle #" << uniqueID << " is receiving Vaccination." << endl;
		}
};

//Concrete Beef Cattle Class. Tracked using Zigbee. That also serves as the Abstract Factory methods client.
class BeefCattle :public Cattle {
private:
		int uniqueID; //Unique identifier.
		ZigbeeDevice* TrackingDevice; //Tracker for beef cattle.
public:
		//Unique ID is set, tracker is initialized.
		BeefCattle(int id) { uniqueID = id; TrackingDevice = new ZigbeeDevice(uniqueID); }

		//Method that simulates eating habits of beef cattle.
		void eatFood(CattleFeeder* feeder) {
			cHydrate = feeder->feedCHydrate();
			protein = feeder->feedProtein();
			cout << "Beef Cattle #" << uniqueID << " is eating " << cHydrate->getName() << " and " << protein->getName() << endl;
		}

		//Visitor accept method.
		void Accept(Visitor * visitor) {
			visitor->Visit(this);
		}

		//Accessor method for the private data member "uniqueID".
		int getID() {
			return uniqueID;
		}

		//Method that the Veterinarian visitor can use.
		void receiveVaccination() {
			cout << "Beef Cattle #" << uniqueID << " is receiving Vaccination." << endl;
		}
};

//Concrete Veterinary Physician Visitor Class
class Veterinarian :public Visitor {
public:
		//Visit method for Dairy Cattle.
		void Visit(DairyCattle* cattle) {
			cattle->receiveVaccination();
		}
		//Visit method for Beef Cattle.
		void Visit(BeefCattle* cattle) {
			cattle->receiveVaccination();
		}
};

//Concrete Ministry of Agriculture Visitor Class
class MinistryRepresentative :public Visitor {
public:
		//Visit Method for Dairy Cattle.
		void Visit(DairyCattle* cattle) {
			cout << "Dairy Cattle #" << cattle->getID() << " has an ear tag." << endl;
		}
		//Visit Method for Beef Cattle.
		void Visit(BeefCattle* cattle) {
			cout << "Beef Cattle #" << cattle->getID() << " has an ear tag." << endl;
		}
};
```

```
//Facade Farm Class
class Farm {
private:
        vector <Cattle*> dairyLivestock; //List that holds Dairy Cattle informations.
        vector <Cattle*> beefLivestock; //List that holds Beef Cattle informations.
        CattleFeeder * dairyFeeder = new DairyCattleFeeder(); //Dairy Feeder instance.
        CattleFeeder * beefFeeder = new BeefCattleFeeder(); //Beef Feeder instance.
public:
        //Method that adds a new Dairy Cattle to the dairyLiveStock list with id.
        void AddNewDairyCattle(int id) {
                dairyLivestock.push_back(new DairyCattle(id));
        }
        //Method that adds a new Beef Cattle to the dairyLiveStock list with id.
        void AddNewBeefCattle(int id) {
                beefLivestock.push_back(new BeefCattle(id));
        }
        //Method that feeds all farm animals.
        void FeedLivestock() {
                for (int i = 0; i<dairyLivestock.size(); i++) {
                        dairyLivestock[i]->eatFood(dairyFeeder);
                }
                for (int i = 0; i<beefLivestock.size(); i++) {

                        beefLivestock[i]->eatFood(beefFeeder);
                }
        }
        //Method that accepts a visitor for all farm animals.
        void AcceptVisitor(Visitor* visitor) {
                for (int i = 0; i<dairyLivestock.size(); i++) {
                        dairyLivestock[i]->Accept(visitor);
                }
                for (int i = 0; i<beefLivestock.size(); i++) {
                        beefLivestock[i]->Accept(visitor);
                }
        }
};
```

4.2 SOURCE

```
#include "dundar-onalProject.h"
#include <iostream>
#include <vector>
#include <string>
#include <ctime>

using namespace std;

//Database singleton initialization.
LivestockDatabase* LivestockDatabase::database = LivestockDatabase::GetDatabase();
//Definition for Database Method "listDevices".
void LivestockDatabase::listDevices() {
	for (unsigned int i = 0; i < deviceList.size(); i++) {
		cout << "Device #" << i + 1 << " is on Cattle ID: " << deviceList[i]->getDeviceHolderID() << endl;
	}
}
//Definition for Database Method "listLocations".
void LivestockDatabase::listLocations() {
	double * p;
	for (unsigned int i = 0; i < deviceList.size(); i++) {
		deviceList[i]->updateLocation();
		cout << "Cattle #" << deviceList[i]->getDeviceHolderID() << " is at location: ";
		deviceList[i]->printLocationInfo();
		cout << endl;
		p = deviceList[i]->getCurrentLocation();
		if (*p>15 || *(p + 2) >15) {
			cout << "--------->" << "WARNING: Cattle #" << deviceList[i]->getDeviceHolderID() << " is outside the farm
boundaries!!!" << endl;
		}
	}
}
//The Farmer Client
int main() {
	srand(time(NULL)); //Seed for location randomizer
	Farm* farm = new Farm(); // Farm facade is created
	//Cattle are added to the farm
	farm->AddNewBeefCattle(31234);
	farm->AddNewBeefCattle(43569);
	farm->AddNewDairyCattle(45435);
	farm->AddNewDairyCattle(35433);
	cout << "--->Connected devices are listed." << endl;
	LivestockDatabase::GetDatabase()->listDevices();
	cout << endl;
	cout << "--->Locations of Cattle animals are listed." << endl;
	LivestockDatabase::GetDatabase()->listLocations();
	cout << endl;
	cout << "--->Catlle is being fed." << endl;
	farm->FeedLivestock();
	cout << endl;
	cout << "--->Veterinary Physician visits the farm." << endl;
	farm->AcceptVisitor(new Veterinarian());
	cout << endl;
	cout << "--->Ministry of Food, Agriculture and Livestock visits the farm." << endl;
	farm->AcceptVisitor(new MinistryRepresentative());

	return 0;
}
```

4.3 Output

```
--->Connected devices are listed.
Device #1 is on Cattle ID: 31234
Device #2 is on Cattle ID: 43569
Device #3 is on Cattle ID: 45435
Device #4 is on Cattle ID: 35433

--->Locations of Cattle animals are listed.
Cattle #31234 is at location: (10,1,8)
Cattle #43569 is at location: (8,1,1)
Cattle #45435 is at location: (0,1,7)
Cattle #35433 is at location: (19,1,13)
--------->WARNING: Cattle #35433 is outside the farm boundaries!!!

--->Catlle is being fed.
Dairy Cattle #45435 is eating Corn and Soybean
Dairy Cattle #35433 is eating Corn and Soybean
Beef Cattle #31234 is eating Wheat and Canola
Beef Cattle #43569 is eating Wheat and Canola

--->Veterinary Physician visits the farm.
Dairy Cattle #45435 is receiving Vaccination.
Dairy Cattle #35433 is receiving Vaccination.
Beef Cattle #31234 is receiving Vaccination.
Beef Cattle #43569 is receiving Vaccination.

--->Ministry of Food, Agriculture and Livestock visits the farm.
Dairy Cattle #45435 has an ear tag.
Dairy Cattle #35433 has an ear tag.
Beef Cattle #31234 has an ear tag.
Beef Cattle #43569 has an ear tag.
```

Important Note about Output: Location info may differ from output to output since it is set randomly.