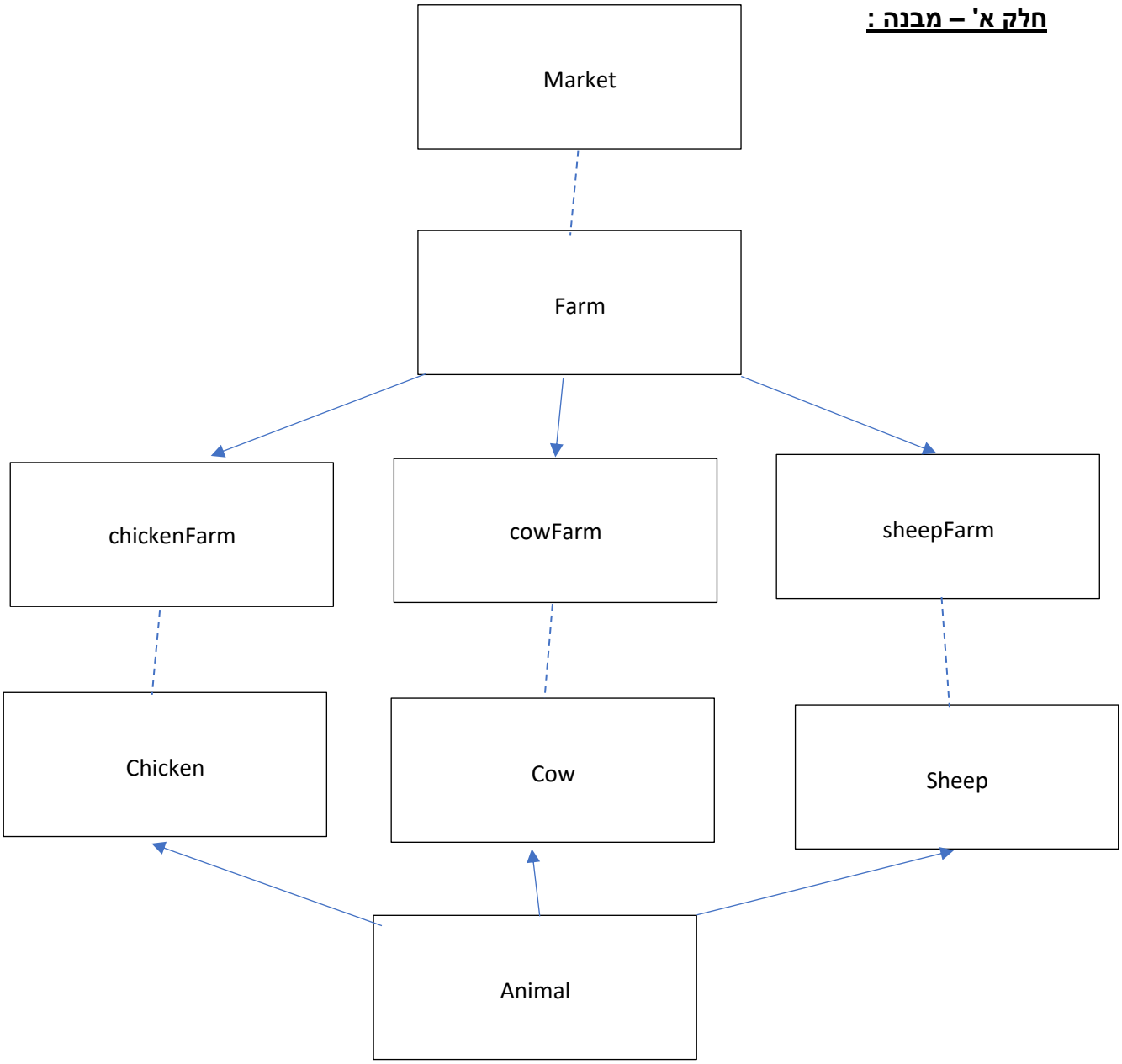
דו"ח על תרגיל בית מספר 4

מגיש : ירין בנימין

חלק א' – מבנה :



ניתן לראות את התלויות בין המחלקות בקובץ המake בצורה הבאה (מקוצרת) :

```
Market.o: Market.cpp Market.h Farm.h cowFarm.h sheepFarm.h chickenFarm.h exp.h
Farm.o: Farm.cpp Farm.h Animal.h
chickenFarm.o: chickenFarm.cpp chickenFarm.h Farm.h Chicken.h
cowFarm.o: cowFarm.cpp cowFarm.h Farm.h Cow.h
sheepFarm.o: sheepFarm.cpp sheepFarm.h Farm.h Sheep.h
Animal.o: Animal.cpp Animal.h priceList.h
Chicken.o: Chicken.cpp Chicken.h Animal.h
Cow.o: Cow.cpp Cow.h Animal.h
Sheep.o: Sheep.cpp Sheep.h Animal.h
```

ניתן לראות שימוש בפולימורפיזם ע"י מחלקות הבנים sheepFarm, cowFarm, chickenFarm למחלקת אב Farm.

במחלקת Farm ניתן לראות את הפונקציות הווירטואלי הבאות :

```
Farm();
virtual ~Farm();
virtual void production()=0;
virtual void buy_animals()=0;
friend ostream& operator<<(ostream& os, const Farm& me);
virtual void addClient(Farm* farm)=0;
virtual void buyProduct(const int seller_id, int &seller_
virtual void sellProduct()=0;
virtual bool want_milk()=0;
virtual bool want_wool()=0;
virtual bool want_eggs()=0;
void nextYear();
int getID();
```

כאשר כול פונקציה כזו ממומשת שונה בין הבנים. לדוגמא הוספת חווה לרשימת לקוחות של חווה אחרת תבוצע ע"י מעבר על כלל החוות ושאלה האם את רוצה לקנות את המוצרים שלי :

```
for(it=Farms.begin(); it != Farms.end(); ++it){
    farm->addClient(*it);
    (*it)->addClient(farm);
}
```

מימוש שונה אצל כול אחת ממחלקות הבנים (sheepFarm, cowFarm, chickenFarm) תגרום להוספה / אי הוספה שלה לרשימת הלקוחות של מחלקה אחרת ובכך בעזרת פולימורפיזם ללא ידיעת סוג המחלקה נוכל להוסיף אותה לרשימה.

לדוגמא מחלקת cowFarm:

```
virtual bool want_milk(){return false;};
virtual bool want_wool(){return false;};
virtual bool want_eggs(){return true;};
```

רוצה לקנות ביצים בלבד ולכן היא תחזיר אמת כאשר נשאל אותה האם להוסיף אותה לרשימת הלקוחות שרוצים לקנות ביצים.

מכאן ניתן לראות שתבנית העיצוב observer נעשית בעזרת פולימורפיזם (הוספה לרשימת הלקוחות) והודעה לרשימה זו נעשית על ידי המוכר.

דוגמת קוד :

כאשר מתבצע סחר בין חוות נעבור על כול החוות ונבקש מהן לעשות מסחר עם רשימת הלקוחות שלהן :

```
void Market::tradeWithOthers(){
    cout << "----Begin Market----" << endl;
    vector<Farm*>::iterator it;
    for(it=Farms.begin(); it != Farms.end(); ++it)
            (*it)->sellProduct();
}
```

נשתמש בפולימורפיזם כדי לעבור על רשימת הלקוחות ולמכור את המוצר שאנחנו רוצים ולשם כך במחלקת האב הפונקציה תהיה וירטואלית :

```
virtual void sellProduct()=0;
```

מימוש לדוגמא אצל מחלקת חוות פרות שמוכרת חלב :

```
void cowFarm::sellProduct(){
    vector<Farm*>::iterator it;
    for(it=Clients.begin(); it != Clients.end(); ++it){
        (*it)->buyProduct(ID, Milk, Money);
        if(Milk == 0)
            break;
    }
}
```

חלק ב' – תרשים זרימה ליצירת חווה :

לדוגמא יצירת חווה חדשה של פרות נעשית בצורה הבאה :

```
farm = new cowFarm();
for(it=Farms.begin(); it != Farms.end(); ++it){
    farm->addClient(*it);
    (*it)->addClient(farm);
}
Farms.push_back(farm);
```

כאשר חווה חדשה נוצרת היא קודם עוברת בבנאי של מחלקת האב ולאחר מכן בבנאי שלה:

```
Farm::Farm() {
    count++;
    ID = count;
    Money = 10;
    Milk = 0;
    Wool = 0;
    Eggs = 0;
}
```

```
cowFarm::cowFarm() {
    for(int i=0; i<start_animals; i++)
        Animals.push_back(new Cow());
}
```

נוסיף לחווה החדשה את כול הלקוחות שהיו במארקט כבר ולחוות במארקט את החווה החדשה שיצרנו כלקוחה. כמו שנאמר בסעיף הקודם רק חוות שקונות חלב יתווספו לרשימת הלקוחות של החווה החדשה שיצרנו ורק חוות שמוכרות ביצים יוסיפו את החווה הנוכחית שנוצרה.

בסיום נוסיף את החווה החדשה לכלל החוות במארקט.

מיד לאחר הוספת כלל החוות החדשות למארקט כול חווה יכולה להתחיל למכור את המוצרים שלה לשאר החוות, כאשר במקרה של חווה חדשה אין לה מוצרים התחלתיים אז רק לאחר תפוקה תוכל למכור את מוצריה (כפי שנראה בסעיף הקודם).

חלק ג' – תכנון נכון :

השתמשתי בקבועים כאשר לא רציתי שתהיה למשתמש אפשרות לשנות את תוכן המשתנה:

```
virtual void buyProduct(const int seller_id, int &seller_inventory, int &seller_money)=0;  // get the id, max inventory and money of the selle
```

במכירת מוצר לא ארצה שהלקוח ישנה את הID של המוכר.

בנוסף ישנו קובץ מחירים בו כלל המחירים הינם קבועים סטטיים כדי שתהיה גישה אליהם מכלל המחלקות ללא אפשרות (כי אין צורך) לשום מחלקה לשנות אותם :

```
static const int milk_price = 3;
static const int wool_price = 2;
static const int eggs_price = 1;

static const int cow_price = 10;
static const int sheep_price = 5;
static const int chicken_price = 3;
```

כאשר מחיר החיות נשמר בקובץ זה כדי שחווה שרוצה לדעת את מחיר חייה תוכל לבוא לקובץ זה ו"לשאול".

בנוסף משתנה נוסף קבוע סטטי במחלקת חווה :

```
static const int start_animals;  // how much animals each farm start with
```

משומש לשם כך שכול סוג של חווה חדשה שנרצה לבנות נוכל להשתמש במשתנה זה, כאשר המחשבה במשתנה זה ובקודמים היא על שינוי קל יותר לפי בקשה בשינוי במקום אחד במקום בקבצים שונים.

בנוסף למשתנים סטטיים אלה השתמשתי במשתנה סטטי נוסף במחלקת חווה :

```
private:
    static int count;
```

שימוש משתנה זה הוא מספור החוות לפי סדר עולה ללא אפשרות למחלקת בן לשנות אותה (בגלל שהיא פרטית).

טיפול בשגיאות נעשה כאשר מוכנס מספר שלילי של כמות חוות, נתפוס את השגיאה נדפיס אותה ונבקש שוב כמות חוות :

```
while(true){
try{
    cout << "How many new cow farms will be added this year?" << endl;
    cin >> cow_farms;
    if(cow_farms < 0)
        throw CowException();
    cout << "How many new sheep farms will be added this year?" << endl;
    cin >> sheep_farms;
    if(sheep_farms < 0)
        throw SheepException();
    cout << "How many new chicken farms will be added this year?" << endl;
    cin >> chicken_farms;
    if(chicken_farms < 0)
        throw ChickenException();
    break;
}
catch( exception& e ){
    cout << e.what();
}
}
```

תודה.