The Abstract Factory Pattern

Design Patterns

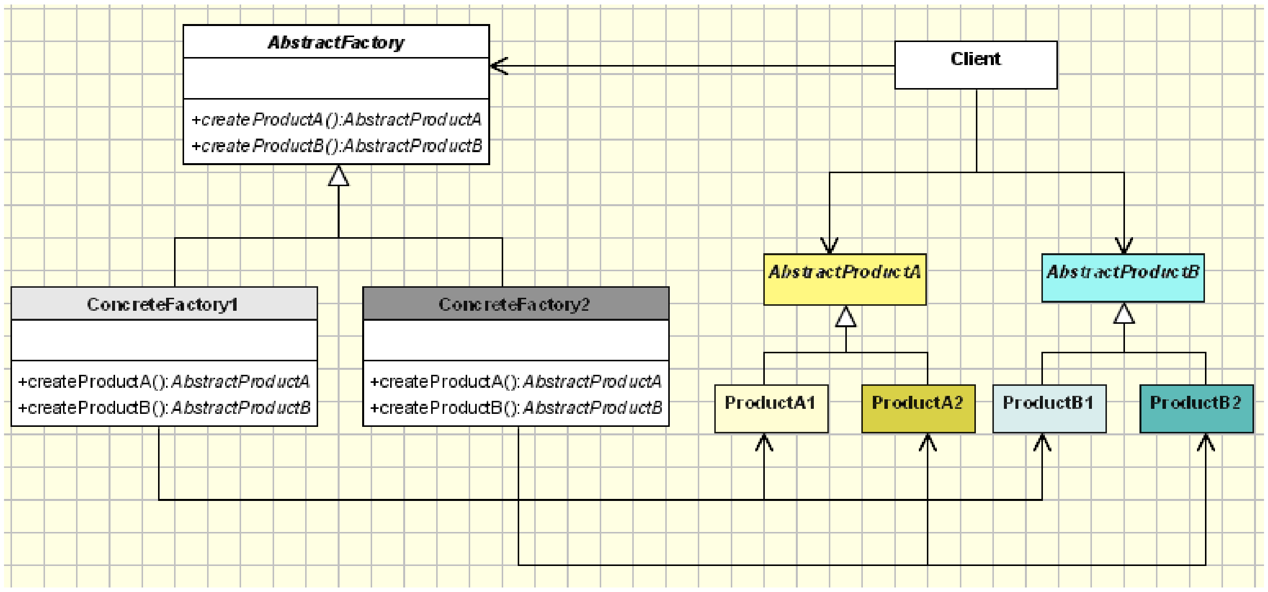
Andreya Grzegorzewski – Ohio Northern University

Fall 2016-2017

Introduction

This assignment requires an application that makes use of the abstract factory pattern. This pattern is similar to the factory method pattern used earlier in the year, except that instead of one factory, there are multiple factories, and there must be multiple products as well. My submission is called “Human Factory,” and it can create Person objects in two different ways: the first is the “test tube” method, which does not require Person objects to be parents, and the second is the “organic” way, where two Person objects mate.

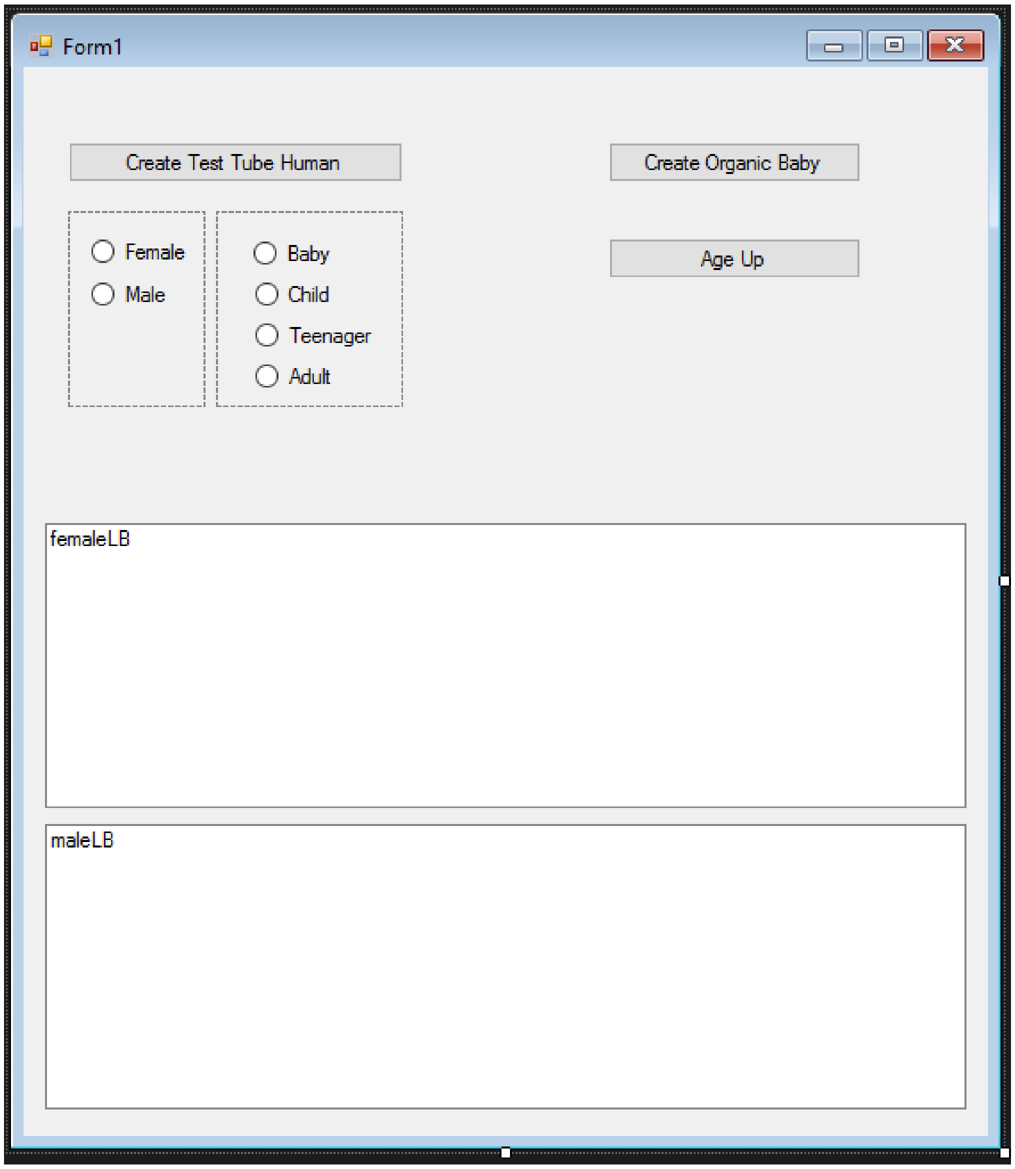
The UML Diagram for the Abstract Factory Pattern


AbstractFactory
+createProductA():AbstractProductA
+createProductB():AbstractProductB
Client
ConcreteFactory1
+createProductA():AbstractProductA
+createProductB():AbstractProductB
ConcreteFactory2
+createProductA():AbstractProductA
+createProductB():AbstractProductB
AbstractProductA
AbstractProductB
ProductA1
ProductA2
ProductB1
ProductB2

Above is the UML diagram for this pattern, courtesy of oodesign.com. This structure is very similar to the structure of the factory method pattern’s UML diagram. The only differences are that in this diagram, there are two concrete factories, two abstract products, and multiple concrete products. Other than that, the pattern is much the same. Concrete factories are used to create concrete products, which the client manipulates. Below is a table showing the classes I used in this program.

AbstractFactory	My abstract factory is called HumanFactory. The two concrete factories inherit from it. It provides signatures for two methods that, when implemented, create concrete products.
Concrete Factories 1 and 2	The classes MaleFactory and FemaleFactory inherit from the HumanFactory class and implement the methods that create concrete products.
Abstract Products A and B	I created an abstract Male class and an abstract Female class to be my abstract products. They both inherit from the Person class, and class inheritance is used throughout the program to simplify some code.
Concrete Products A1, A2, B1, and B2	The concrete classes that inherit from the Male and Female classes are called TestTubeMale, TestTubeFemale, OrganicMale, and OrganicFemale. They are created in different ways but are very similar to each other.

<u>Narrative and Code</u>



I started writing my program by designing the form in its entirety. You can see the GUI to the left. The two Create buttons make new humans; the create test tube human uses the radio buttons to make a hew human, and the create organic baby button uses a male parent and a female parent, as selected from the two list boxes, to create a new human. The age up button increases the age of every person listed by one year and updates their height and weight as necessary.

After I created the GUI, I began to work on the products. I created an abstract Person class first. This class has no contents. It is just used to unify the Male and Female classes created later so that concrete products can be more easily created. The Person class is as follows:

```
    public abstract class Person
    {
    }
```

I then created the Male and Female classes, which are the abstract product classes in the UML diagram. These two classes are identical to each other, except in name. Both are implemented in the following way:

```
    public abstract class Male : Person // This is an abstract product
    {
        public abstract string getEyeColor();
        public abstract string getHairColor();
        public abstract void kill();
        public abstract bool canBeAParent();
        public abstract bool isDead();
        public abstract int ageUp();
    }
```

The meanings of each of the methods will be described in the following section, which describes the implementation of the concrete products. Each of the four concrete product classes is implemented in a very similar way. The only differences are in height and weight equations for males and females and constructors for organic and test tube humans. I will describe just one of the four concrete classes below: the OrganicMale class. The header for the class is as follows:

```
public class TestTubeMale : Male // this is a concrete product
```

I began creating the OrganicMale class by creating a list of variables:

```
string name;
int height; // in inches
int weight = 0; // in pounds
int age;    // in months
string hairColor;
string eyeColor;
string[] hairColors = { "brown", "blonde", "black", "red" };
string[] eyeColors = { "brown", "hazel", "green", "blue" };
bool deceased = false;

Random rng = new Random();
```

I then implemented each of the functions whose signatures were provided in the abstract Male class. getEyeColor, getHairColor, and isDead are all simple getters:

```
        public override string getEyeColor()
        {
            return this.eyeColor;
        }

        public override string getHairColor()
        {
            return this.hairColor;
        }

        public override bool isDead()
        {
            return this.deceased;
        }
```

The method kill is used to mark a person as deceased when he dies as the result of aging up.

```
        public override void kill()
        {
            this.deceased = true;
        }
```

canBeAParent checks to make sure that the selected person is both old enough to have children and not dead.

```
public override bool canBeAParent()
{
    return (age >= 216 && !this.deceased); // The person can have babies if he is at
       least 18 years old
}
```

ageUp increases the age of a person by one year and updates heights and weights as appropriate for the person’s age.

```
public override int ageUp()
{
    age += 12;

    // If the person is over 18, only update his weight
    if (age >= 216)
    weight += rng.Next(-3, 5);

    // Otherwise, update height and weight
    else
    {
        height = Convert.ToInt32(Math.Floor((-.0005 * age * age) + (.3068 * age) +
              25.376));
        weight = Convert.ToInt32(Math.Floor((.0016 * age * age) + (.2881 * age) +
              16.664));
        height += rng.Next(-5, 5);
        weight += rng.Next(-(age / 12), (age / 12));
    }
    return age;
}
```

With all of the abstract functions overridden and implemented, I moved on and created a constructor.

```
public OrganicMale(Female mom, Male dad, string name, int age)
{
    this.name = name;
    this.age = age;

    // Assign eye color based on parents' traits
    if (rng.Next(2) == 0)
        this.eyeColor = mom.getEyeColor();
    else
        this.eyeColor = dad.getEyeColor();

    // Assign hair color based on parents' traits
    if (rng.Next(2) == 0)
        this.hairColor = mom.getHairColor();
    else
        this.hairColor = dad.getHairColor();

    // Generated these equations by plotting average height and weight values versus age
       in months in excel, then generating trendlines
    height = Convert.ToInt32(Math.Floor((-.0005 * age * age) + (.3068 * age) + 25.376));
    weight = Convert.ToInt32(Math.Floor((.0016 * age * age) + (.2881 * age) + 16.664));

    // Randomizes heights and weights slightly
    height += rng.Next(-5, 5);
    weight += rng.Next(-(age / 12), (age / 12));
}
```

The difference between this constructor and a constructor for a test tube human is that a test tube human does not have a mom or dad, so the hair color and eye color are just randomized as follows:

```
hairColor = hairColors[rng.Next(hairColors.Length)];
eyeColor = eyeColors[rng.Next(eyeColors.Length)];
```

After creating the constructor, I overrode the ToString method to describe the person textually.

```
public override string ToString()
{
    string description = name + " - ";
    int feet = 0;
    int inches = height; // Use this so we don't destroy the height variable

    while (inches >= 12)
    {
        feet++;
        inches -= 12;
    }

    description += feet + " feet, " + inches + " inches - ";
    description += weight + " pounds - ";

    int months = age;
    int years = 0;

    while (months >= 12)
    {
        months -= 12;
        years++;
    }

    if (years == 0)
        description += months + " months old. ";
    else
        description += years + " years old. ";

    description += "He has " + hairColor + " hair and " + eyeColor + " eyes.";

    if (this.deceased)
        description = name + " died at " + years + " years old.";

    return description;
}
```

This is the end of the OrganicMale class. The other three concrete product classes were implemented in very similar ways, so the code will not be included here, but it is available in my Design-Patterns repository on GitHub at https://github.com/andreya-grzegorzewski/Design-Patterns.

With my products implemented, I moved on to the factories. I started with the abstract factory, HumanFactory. This class simply defined two abstract factory methods:

```
    public abstract class HumanFactory // This is the AbstractFactory class
    {
        public abstract Person createTestTubePerson(int age);
        public abstract Person createBaby(Female mom, Male dad);
    }
```

Then I defined the MaleFactory and FemaleFactory classes. Much like the Male and Female classes, the MaleFactory and FemaleFactory classes are implemented very similarly, so only one will be described here. The class header is as follows:

```
class MaleFactory : HumanFactory // This is a concrete factory
```

I first created three variables: a random number generator, a list of boys' names to be used for organic babies, and an index to be used for test tube humans.

```
Random rng = new Random();
string[] boyNames = { "Jaired", "Matt", "Jack", "Nathan", "Chris", "Paul", "Adam",
       "Brayden", "Mike" };
int maleIndex = 1001;
```

Then, I implemented the two factory methods. The first sets a generic male name, then calls the appropriate constructor:

```
public override Person createTestTubePerson(int age)
{
    string name = "Male " + maleIndex;
    maleIndex++;
    return new TestTubeMale(name, age);
}
```

The second randomly selects a name from the list, sets the age to zero months, and then calls the appropriate constructor.

```
public override Person createBaby(Female mom, Male dad)
{
    string name = boyNames[rng.Next(boyNames.Length)];
    int age = 0;

    return new OrganicMale(mom, dad, name, age);
}
```

This is all of the code used in the MaleFactory class. The FemaleFactory class is much the same; the only difference is that the list of names contains girls' names instead of boys' names. With the code for the factories written, I moved on to the code for the form. I started by creating my variables:

```
MaleFactory mf;
FemaleFactory ff;
Random rng = new Random();
List<Female> females = new List<Female>();
List<Male> males = new List<Male>();
```

Then, in the Form1 constructor, I initialized the mf and ff variables.

```
public Form1()
{
    InitializeComponent();
    mf = new MaleFactory();
    ff = new FemaleFactory();
}
```

The only remaining code that I found necessary to implement in the form was the code for each of the buttons. First, I implemented the Create Test Tube Human button. This code consists of several if-else statements to create a new human that matches the specified parameters.

```
private void testTubeBabyButton_Click(object sender, EventArgs e)
{
    if (femaleRB.Checked)
    {
        // Creates the female by checking radio boxes
        Female newFemale = null;
        if (babyRB.Checked)     // Create a female two years or younger
            newFemale = (Female) ff.createTestTubePerson(rng.Next(24));
        else if (childRB.Checked)    // Create a female from 2-12 years
            newFemale = (Female) ff.createTestTubePerson(rng.Next(24, 144));
        else if (teenagerRB.Checked) // Create a female from 12-18 years
            newFemale = (Female) ff.createTestTubePerson(rng.Next(144, 216));
        else if (adultRB.Checked)    // Create a female from 18-80 years
            newFemale = (Female) ff.createTestTubePerson(rng.Next(216, 960));
        else
            MessageBox.Show("Please select an age.");

        // Add the female to the appropriate lists
        if (newFemale != null)
        {
            females.Add(newFemale);
            femaleLB.Items.Add(newFemale.ToString());
        }
    }
    // Repeat
    else if (maleRB.Checked)
    {
        Male newMale = null;
        if (babyRB.Checked)
            newMale = (Male) mf.createTestTubePerson(rng.Next(24));
        else if (childRB.Checked)
            newMale = (Male) mf.createTestTubePerson(rng.Next(24, 144));
        else if (teenagerRB.Checked)
            newMale = (Male) mf.createTestTubePerson(rng.Next(144, 216));
        else if (adultRB.Checked)
            newMale = (Male) mf.createTestTubePerson(rng.Next(216, 960));
        else
            MessageBox.Show("Please select an age.");

        if (newMale != null)
        {
            males.Add(newMale);
            maleLB.Items.Add(newMale.ToString());
        }
    }
    else
```

```
        MessageBox.Show("Please select a gender.");
}
```

I then wrote the code for the organic baby button, which is described with comments.

```
private void organicBabyButton_Click(object sender, EventArgs e)
{
    // Converts the lists to access the necessary element
    Female[] femaleArray = females.ToArray();
    Female mom = femaleArray[femaleLB.SelectedIndex];
    Male[] maleArray = males.ToArray();
    Male dad = maleArray[maleLB.SelectedIndex];

    // Makes sure we have parents who can have children
    if (mom == null || dad == null)
        MessageBox.Show("Please select a mother and a father.");
    else if (!mom.canBeAParent() || !dad.canBeAParent())
         MessageBox.Show("Please make sure both parents are old enough to have kids!");

    else
    {
        // Randomly decides if the baby will be a boy or girl
        if (rng.Next(2) == 0)
        {
            // Creates a new female and adds her to the lists
            Female newFemale = (Female)ff.createBaby(mom, dad);
            females.Add(newFemale);
            femaleLB.Items.Add(newFemale.ToString());
        }
        else
        {
            // Or creates a new male and adds him to the lists
            Male newMale = (Male)mf.createBaby(mom, dad);
            males.Add(newMale);
            maleLB.Items.Add(newMale.ToString());
        }
    }
}
```

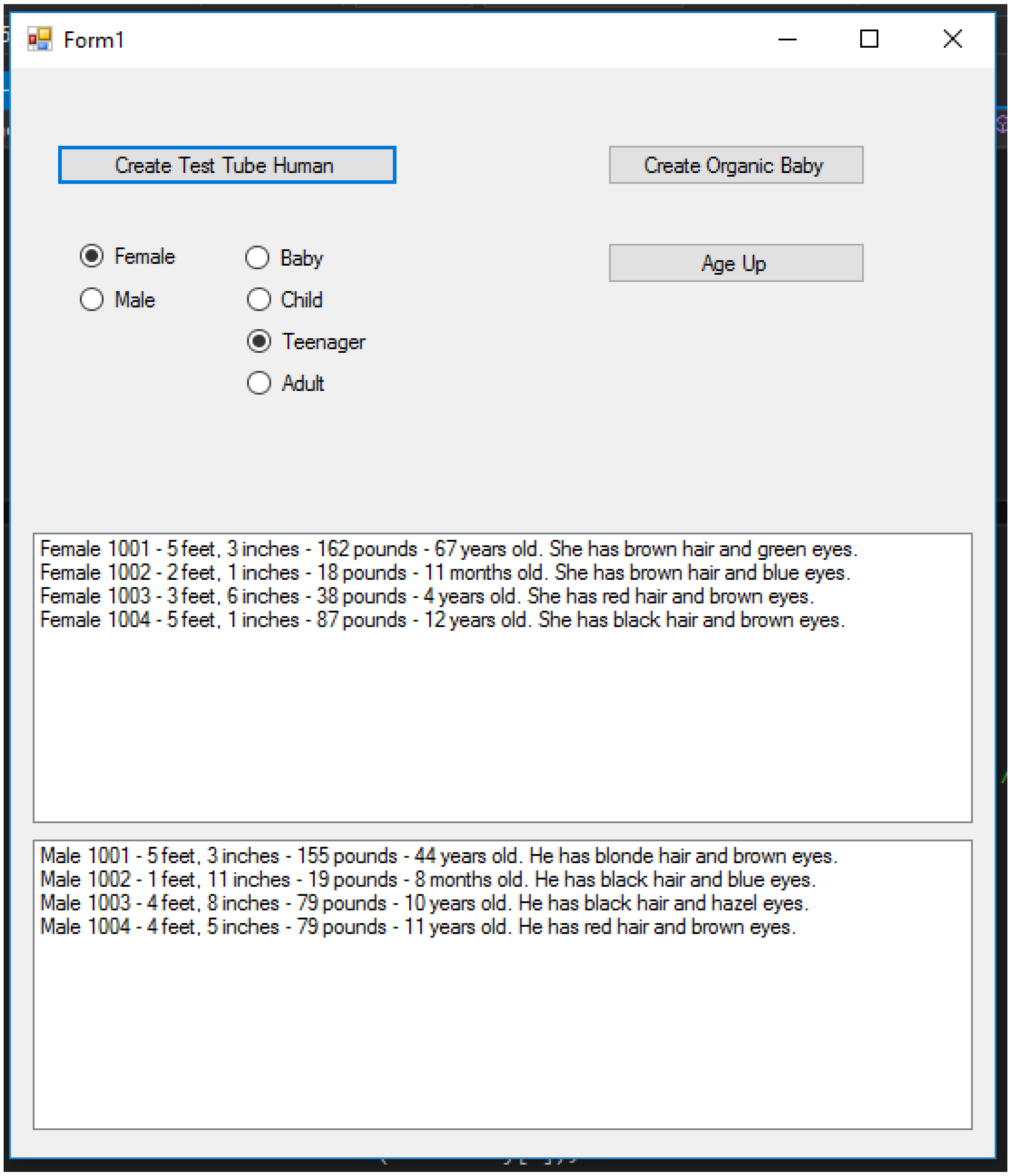
Finally, I implemented the age up button.

```
private void ageUpButton_Click(object sender, EventArgs e)
{
    // Convert lists to arrays for easy access
    Female[] femalesArray = females.ToArray();
    Male[] malesArray = males.ToArray();

    // We clear the list box and re-add each item as we go.
    femaleLB.Items.Clear();
    for (int i = 0; i < femalesArray.Length; i++)
    {
        if (!femalesArray[i].isDead() && femalesArray[i].ageUp() >= 960) // If the person
             is at least 80...
            if (rng.Next(5) == 0) // There is a chance...
                femalesArray[i].kill(); // That they might die :(
        femaleLB.Items.Add(femalesArray[i]); // Add her to the list anyways
    }

    // Repeat
    maleLB.Items.Clear();
    for (int i = 0; i < malesArray.Length; i++)
    {
        if (!malesArray[i].isDead() && malesArray[i].ageUp() >= 960)
            if (rng.Next(5) == 0)
                malesArray[i].kill();
        maleLB.Items.Add(malesArray[i]);
    }
}
```

With this, all of the code for my project was written and the program can be run.

The Deliverable

On the following pages are several screenshots of my program in action.

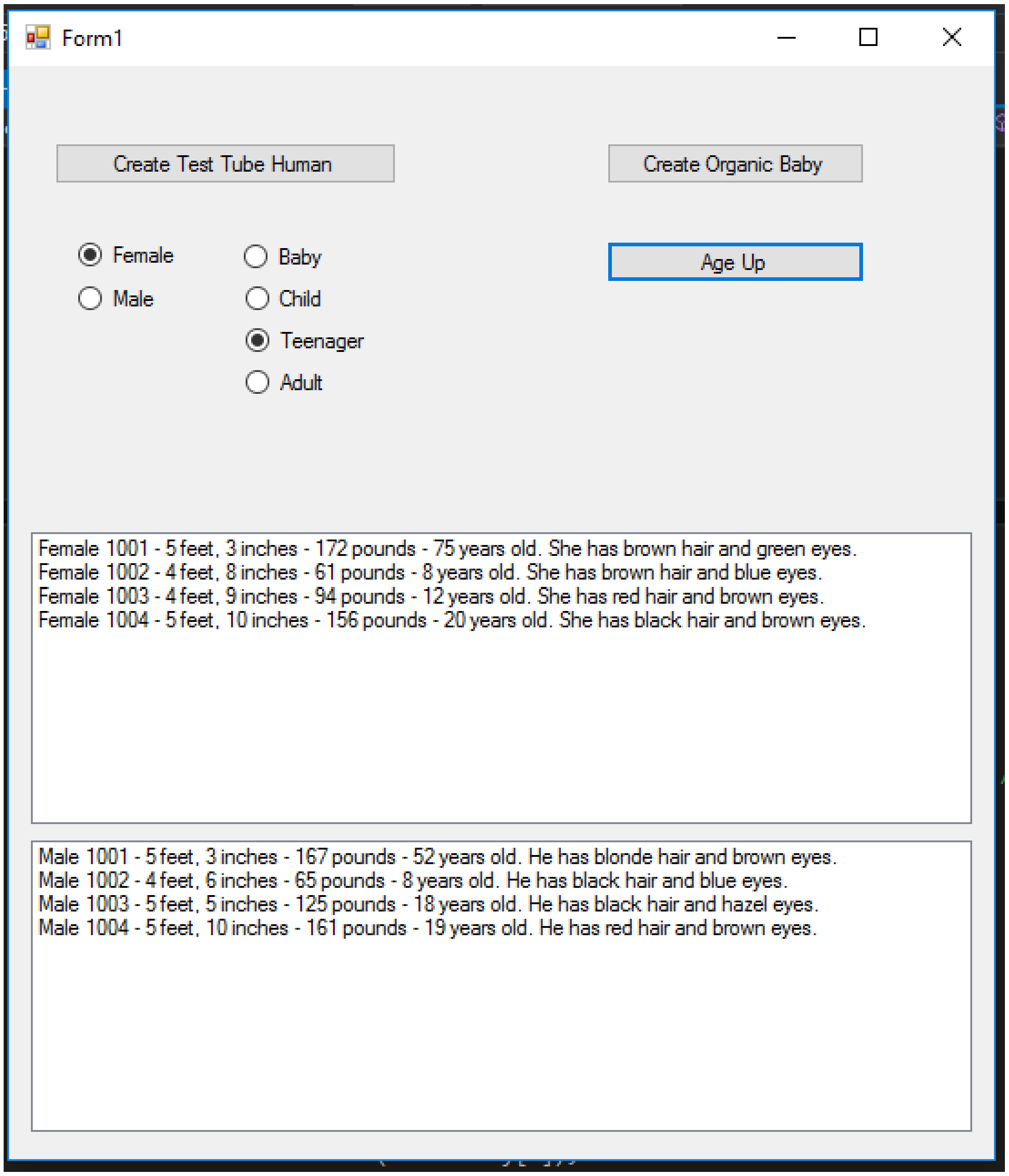


Female [✓] Male | Baby Child Teenager [✓] Adult

```
Female 1001 - 5 feet, 3 inches - 162 pounds - 67 years old. She has brown hair and green eyes.
Female 1002 - 2 feet, 1 inches - 18 pounds - 11 months old. She has brown hair and blue eyes.
Female 1003 - 3 feet, 6 inches - 38 pounds - 4 years old. She has red hair and brown eyes.
Female 1004 - 5 feet, 1 inches - 87 pounds - 12 years old. She has black hair and brown eyes.
```

```
Male 1001 - 5 feet, 3 inches - 155 pounds - 44 years old. He has blonde hair and brown eyes.
Male 1002 - 1 feet, 11 inches - 19 pounds - 8 months old. He has black hair and blue eyes.
Male 1003 - 4 feet, 8 inches - 79 pounds - 10 years old. He has black hair and hazel eyes.
Male 1004 - 4 feet, 5 inches - 79 pounds - 11 years old. He has red hair and brown eyes.
```

Here I opened the program and created one test tube human of each of the available combinations. We can see that their weights and heights are appropriate for their ages and their hair and eye colors are randomized.

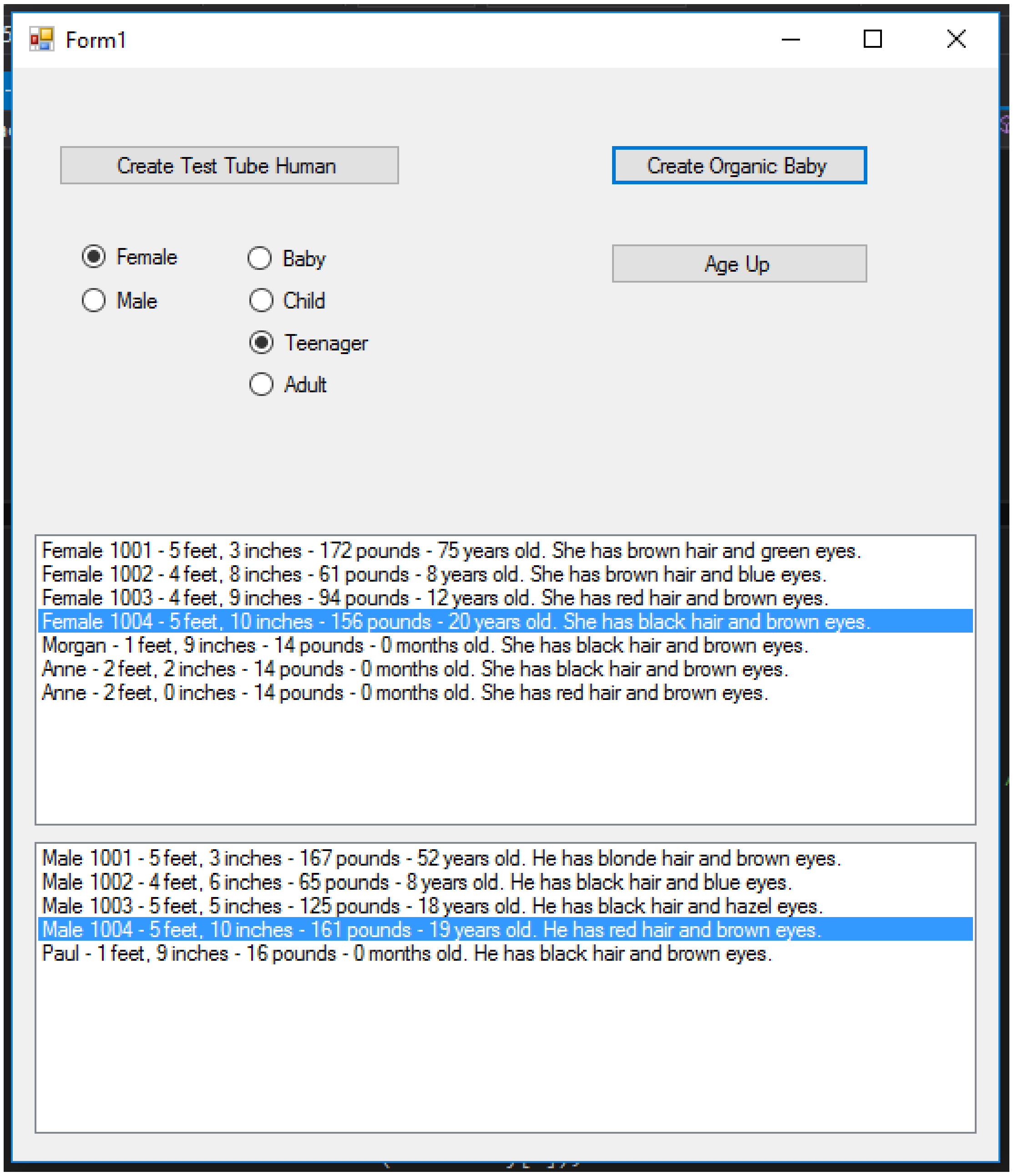


Female [✓] Male
Baby Child Teenager [✓] Adult

```
Female 1001 - 5 feet, 3 inches - 172 pounds - 75 years old. She has brown hair and green eyes.
Female 1002 - 4 feet, 8 inches - 61 pounds - 8 years old. She has brown hair and blue eyes.
Female 1003 - 4 feet, 9 inches - 94 pounds - 12 years old. She has red hair and brown eyes.
Female 1004 - 5 feet, 10 inches - 156 pounds - 20 years old. She has black hair and brown eyes.
```

```
Male 1001 - 5 feet, 3 inches - 167 pounds - 52 years old. He has blonde hair and brown eyes.
Male 1002 - 4 feet, 6 inches - 65 pounds - 8 years old. He has black hair and blue eyes.
Male 1003 - 5 feet, 5 inches - 125 pounds - 18 years old. He has black hair and hazel eyes.
Male 1004 - 5 feet, 10 inches - 161 pounds - 19 years old. He has red hair and brown eyes.
```

This screenshot is the result of me pressing the age up button eight times. We can see that each human's hair color, eye color, and name remained the same. Additionally, if the human was older than eighteen years, his or her height remained the same. However, weights are updated every year for each human and heights are updated each year for humans less than eighteen years old.



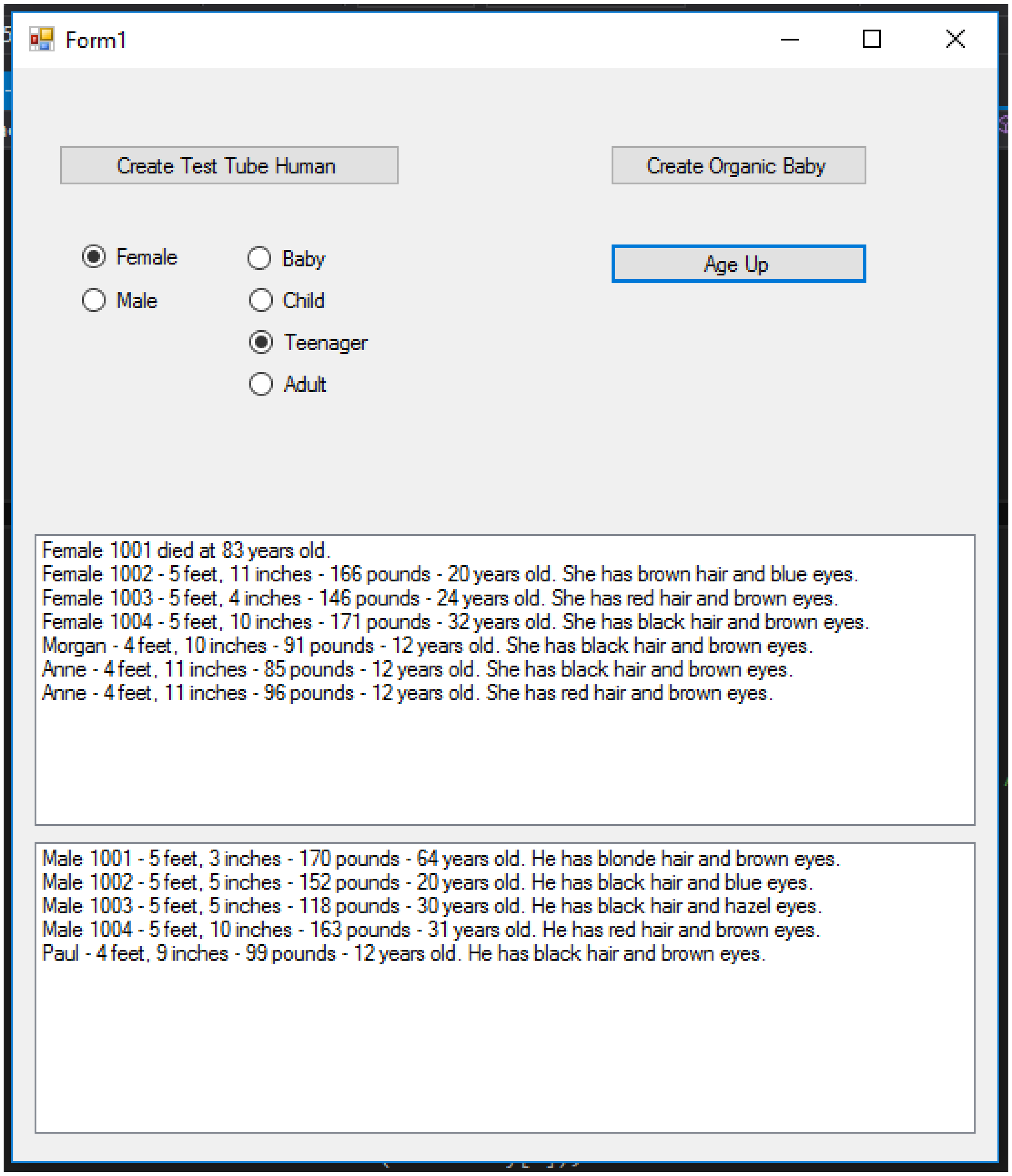
[✓ Female] Male

Baby Child [✓ Teenager] Adult

```
Female 1001 - 5 feet, 3 inches - 172 pounds - 75 years old. She has brown hair and green eyes.
Female 1002 - 4 feet, 8 inches - 61 pounds - 8 years old. She has brown hair and blue eyes.
Female 1003 - 4 feet, 9 inches - 94 pounds - 12 years old. She has red hair and brown eyes.
Female 1004 - 5 feet, 10 inches - 156 pounds - 20 years old. She has black hair and brown eyes.
Morgan - 1 feet, 9 inches - 14 pounds - 0 months old. She has black hair and brown eyes.
Anne - 2 feet, 2 inches - 14 pounds - 0 months old. She has black hair and brown eyes.
Anne - 2 feet, 0 inches - 14 pounds - 0 months old. She has red hair and brown eyes.
```

```
Male 1001 - 5 feet, 3 inches - 167 pounds - 52 years old. He has blonde hair and brown eyes.
Male 1002 - 4 feet, 6 inches - 65 pounds - 8 years old. He has black hair and blue eyes.
Male 1003 - 5 feet, 5 inches - 125 pounds - 18 years old. He has black hair and hazel eyes.
Male 1004 - 5 feet, 10 inches - 161 pounds - 19 years old. He has red hair and brown eyes.
Paul - 1 feet, 9 inches - 16 pounds - 0 months old. He has black hair and brown eyes.
```

Here, I had Female 1004 and Male 1004 have four children together. Their heights and weights are all similar, but not exactly the same, and their hair and eye colors come from their parents’ hair and eye colors.



[✓ Female] Male

Baby Child [✓ Teenager] Adult

```
Female 1001 died at 83 years old.
Female 1002 - 5 feet, 11 inches - 166 pounds - 20 years old. She has brown hair and blue eyes.
Female 1003 - 5 feet, 4 inches - 146 pounds - 24 years old. She has red hair and brown eyes.
Female 1004 - 5 feet, 10 inches - 171 pounds - 32 years old. She has black hair and brown eyes.
Morgan - 4 feet, 10 inches - 91 pounds - 12 years old. She has black hair and brown eyes.
Anne - 4 feet, 11 inches - 85 pounds - 12 years old. She has black hair and brown eyes.
Anne - 4 feet, 11 inches - 96 pounds - 12 years old. She has red hair and brown eyes.
```

```
Male 1001 - 5 feet, 3 inches - 170 pounds - 64 years old. He has blonde hair and brown eyes.
Male 1002 - 5 feet, 5 inches - 152 pounds - 20 years old. He has black hair and blue eyes.
Male 1003 - 5 feet, 5 inches - 118 pounds - 30 years old. He has black hair and hazel eyes.
Male 1004 - 5 feet, 10 inches - 163 pounds - 31 years old. He has red hair and brown eyes.
Paul - 4 feet, 9 inches - 99 pounds - 12 years old. He has black hair and brown eyes.
```

Finally, I pressed the age up button several times to demonstrate that humans can die in my app, but they will continue to be displayed.

Observations

This app was fun for me to write, and especially after writing an app that utilized the factory method pattern, the abstract factory pattern seemed very intuitive to me. I like that it provides different ways to create products so there can be some slight variations as needed. There were a lot of redundancies in my code, such as the shameless repetition of code in the four concrete product classes, that I couldn't figure out how to eliminate. I decided to focus more on the implementation of the pattern and temporarily ignore the repetition of code present throughout my program, but I would like to learn how to properly implement inheritable methods so that I don’t have the same problem in the future.