Table of Contents

Setup

Download **Greenfoot Lab 4 Project.zip** and extract it somewhere on your machine. Open the project within. When you're done, zip up your project and upload it to the DROPBOX.

If you're working on a team, put each teammate's name at the top of your **FlowerWorld.java** source file as a comment.

Review: Types of Variables

In Greenfoot and Java, we have 4 different types of variables:

1. A parameter to a method

```
public void pace(int numberSteps)
{
    walkLeft(numberSteps);
    walkRight(2 * numberSteps);
    walkLeft(numberSteps);
}
```

`numberSteps` is a parameter variable that is used in the program logic for the method **`pace()`**

When a caller method wishes to use the method **`pace()`** an actual parameter is needed.

Example: **`pace( 5 );`**

5 is copied into the parameter **`numberSteps`** and the code in **`pace()`** is executed.

A parameter is created when the method is called and the parameter goes away when the method completes.

2. A local variable within a method

```
public void jumpUpAndDown()
{
    int amount = getImage().getHeight() / 2;
    setLocation(getX(),getY() - amount);
    checkForPizza();
    wait(10);
    setLocation(getX(),getY() + amount);
}
```

`amount` is a local variable. The scope of amount is the { } that mark off the { } for the body of the method.

The programmer is free to reuse the name amount anywhere else in the Scenario and there will be confusion because this amount only makes sense inside the { } of the method.

A local variable is created when it comes into scope when the method is called and the local variable goes away when it goes out of scope, most likely when the method completes.

3. Instance variables

An instance variable is a **member variable** in a class, and every **instance** of the class (every variable created of that class type) gets its own version of that variable.

For example, if we had a **Student** class, and the class had **String name** declared within it, every **new Student()** we make would have its own name – the name wouldn't be shared.

4. Class (static) variables

A static variable is also a **member variable** in a class, but it has the keyword **static** added on. If a variable is static, it means that **there is only one variable, shared across all the instances of the class**.

So, perhaps all Students declared belong to the same school. If we declared **static String school**, then it would be shared across all Students.

```
class Student
{
    String name;
    static String school;
    // ...
}
```

It would have the same value across all instances, and if you replaced it through one Student, then all Students would have this new value.

```
Student a = new Student();
Student b = new Student();
Student c = new Student();
```

Three different students

```
a.name = "Harry";
b.name = "Hermione";
c.name = "Ron";
```

Each has their own unique name

```
a.school = "Hogwarts";
```

If we update the school through the **a** instance, we will see the same value for **b.school** and **c.school**.

Step 1: Fading Flowers

Copy and paste the following method into the Flower class and add a method call to the act method.

```
public void fade()
{
    GreenfootImage myImageAssistant = getImage();
    int previousTransparency = myImageAssistant.getTransparency();
    int newTransparency = (int)(((double)previousTransparency) * .8);
    myImageAssistant.setTransparency(newTransparency);
}
```

You probably have questions about the code listed about, in particular lines 1 and 3. We will discuss the code a little later in this lab.

Compile your Scenario, place a single Flower object in the world and click on >ACT numerous times, the flower should fade, i.e. have a smaller transparency, each time >Act is clicked

Reset the Scenario, place a single Flower object in the world and click on >Run
It might fade out really fast
We would like a way to slow things down so that we can see the progression of images as the flower fades. We need a way to only fade part of the time, say every 30 act cycles.

To accomplish this, we need an instance variable.

Creating an instance variable

1. Inside of the **Flower** class definition, we are going to declare an integer variable named **age**. This variable should be defined **within the class** and outside of all methods.
2. In the **Flower constructor**, initialize **age** to 0.
3. And in the **act()** method, increment **age** by 1 before each call to **fade()**.
4. See the code to the right for reference.

```
public class Flower extends Actor
{
    int age;

    public Flower()
    {
        age = 0;
    }

    public void act()
    {
        age += 1;
        fade();
    }
```

Now that we have an **age** variable, we want to slow down the rate that the flower fades. Since age is incremented every time that **act()** is called, we can use it as a sort of timer (though keep in mind that it isn't based on time since Greenfoot speed can be changed.)

Every 30th cycle, we will want to call **fade()**, instead of calling it every cycle. This means that, **fade()** will only be called when **age** is 30, 60, 90, etc.

But how do we tell if **age** is ***evenly divisible*** by 30? If it is evenly divisible, that means there's no remainder – we can get the remainder of a division with the % (modulus) operator. For example:

- Modulus 2 can be used to find if something is even or odd:
  - 10 % 2 is 0, so 10 is even.
  - 9 % 2 is 1, so it is odd.

- Let's see how Modulus 3 works:
  - 0 % 3 = 0 1 % 3 = 1 2 % 3 = 2
  - 3 % 3 = 0 4 % 3 = 1 5 % 3 = 2
  - 6 % 3 = 0 7 % 3 = 1 8 % 3 = 2

Notice that, for each number that is divisible by 3, *n % 3* is equal to 0. We can use the same thing to figure out every 30th cycle in the program:

```
public void act()
{
    age += 1;
    if ( age % 30 == 0 )
    {
        fade();
    }
}
```

Now you can click on **run** and any flowers on the screen will fade out more slowly.

- Will it fade faster or slower if you choose a number less than 30?
- Will it fade faster or slower if you choose a number greater than 30?

Step 2: Destroying Flowers

Because the transparency is being reduced, eventually the Flower object is not visible, but it remains in the World. If we were on a really slow computer, this would be a waste of processing, since the flower's **act()** method is still called, even though it is invisible.

Let's remove the flower from the world once it is completely transparent.

We need to make 2 decisions:

1. Will the Flower object make the decision to remove itself from the world, or will the FlowerWorld object make the decision to remove a Flower object from its world?
2. What criteria will be used in determining when to remove the Flower object:
    1. when the transparency fall below a certain value
    2. when the Flower object has been in world a certain length of time, based the value in the age instance variable (i.e. instance field)

The following code allows the Flower object to remove itself base on its age. The method code should be placed in the **Flower** class (inside the Flower class {}, but not inside any other method { })

```
public void checkToRemoveObject()
{
    if ( age == 200 )
    {
        World myWorldAssistant = getWorld();
        myWorldAssistant.removeObject(this);
    }
}
```

Make sure to call this method within the **Flower's act()** method as well. Call it at the very end of the method.

Step 3: Different Flowers

We can keep using the **Flower** class to make different types of flowers – different images, attributes, etc., but which all still function the same. This way, we don't have to make a dozen different **subclasses** of the Actor class, since the different flower types won't differ all that much.

When a flower is created, let's randomly assign it an image. Remember that when a **Flower** object is created, the **Flower()** constructor is called automatically. We will write a method called **selectFlowerImage()**, and call it from the Flower() constructor.

selectFlowerImage() method

Return type: void Parameters: none

1. Generate a random integer number between [0, 2] (inclusive) and store it in a variable, **flowerType**.
2. Use an **if / else if / else** statement to check the random value:
   If choice is 0, set the flower's image to **flower2.png**
   If choice is 1, set the flower's image to **sunflower.png**
   If choice is 2, set the flower's image to **tulip.png**
   Otherwise, by default set the flower's image to **ant3.png**.

Set the image of the flower using the **setImage** method:

```
setImage("flower2.png");
```

For the **else** statement, it sets the image to an ant. This is a... literal bug. Since the random number generator should give us 0, 1, or 2, we should not be able to get *anything else* in this if statement.

Check the next page to get the source for this method to check your work.

Remember to call **selectFlowerImage()** from the Flower() constructor!

Then create several new **Flower** instances and put them in the game world to make sure that you get different types of flowers!

selectFlowerImage solution

```
public void selectFlowerImage()
{
    int choice = Greenfoot.getRandomNumber(3);
    if ( choice == 0 )
    {
        setImage("flower2.png");
    }
    else if ( choice == 1 )
    {
        setImage("sunflower.png");
    }
    else if ( choice == 2 )
    {
        setImage("tulip.png");
    }
    else
    {
        setImage("ant3.png");
    }
}
```

Step 4: Flower Bed

Now that we have Flower objects it would be nice to a collection in the world that represents a flower bed. The first decision to be made is where to put the code to create a flower bed. If we put the code in the Flower class then each Flower object would create a flower bed and since we have no way of letting one Flower object tell the other Flower objects what to do, we would have lots and lots of flower beds.

There is only one world object and so that will be the best place to create our flower bed.

Within the **FlowerWorld** class, create a method called **populateFlowerBed()**. It will look like the following:

```
public void populateFlowerBed()
{
    for ( int i = 0; i < 8 ; i++)
    {
        addObject(new Flower(),
            Greenfoot.getRandomNumber(getWidth()),
            getHeight() - 30);
    }
}
```

This is a **for loop**. We will use loops more later on. Essentially, this loop creates the variable **i**, and **i** starts at 0, and ends at 7 (because < 8). It will run the contents inside the loop that many times – so 8 times total, and 8 flowers will be created.

Let's call **populateFlowerBed()** within the **FlowerWorld()** constructor. Now, when you hit the **reset** button, flowers will be added automatically. If you keep hitting **reset**, flowers will be at random x positions, but always at the same y position.

Let's also set each flower to a different **age** when we create the flower bed. To do that, we need to go back into the **Flower** class code, and we can add a **second constructor**...

```
public Flower()
{
    age = 0;
    selectFlowerImage();
}

public Flower(int startingAge)
{
    age = startingAge;
    selectFlowerImage();
}
```

It is completely valid to have two methods with the same name – whether or not they're constructors. This is known as **method overloading**. As long as they have different **parameter lists**, we can keep reusing the same method name!

Now with this new constructor, when we call **new Flower()** we can add in a starting age to initialize the flower instance to a different age...

Old Version

```
public void populateFlowerBed()
{
    for ( int i = 0; i < 8 ; i++)
    {
        addObject(new Flower(),
            Greenfoot.getRandomNumber(getWidth()),
            getHeight() - 30);
    }
}
```

New Version

```
public void populateFlowerBed()
{
    for ( int i = 0; i < 8 ; i++)
    {
        int randomAge = Greenfoot.getRandomNumber(200);
        addObject(new Flower(randomAge),
            Greenfoot.getRandomNumber(getWidth()),
            getHeight() - 30);
    }
}
```

Run and reset the program multiple times, the flowers will be destroyed at different times.

Step 5: Static Variables

Hard coding numbers and data in our programs are usually not a great practice – they can lead to duplicate code (harder to maintain, more places to fix), or just generally make it harder to go back and change the values later on.

Right now, all flowers expire at the age of 200. Instead of hard-coding this value, let's create a variable for it instead.

Since all flowers will expire at the same value, let's make it a **static** variable. And, since this variable shouldn't be changed anytime throughout the program, let's also make it **final**.

In other languages, **final** is called **const –** it is a variable whose value cannot change after its declaration.

In the **Flower** class, add:

```
public static final int FLOWER_LIFE = 200;
```

In programming, it is standard practice to name a const/final variable in ALL_CAPITAL_LETTERS like this.

Update the **checkToRemoveObject()** to use this new variable, instead of hard-coding the 200.

Within **FlowerWorld**, make sure to also update the usage of 200 in the **populateFlowerBed()** method. Since **FLOWER_LIFE** is a member of the **Flower** class, outside of the class we have to use it like:

```
int randomAge = Greenfoot.getRandomNumber(Flower.FLOWER_LIFE);
```

Now we can easily modify this value, to extend the flowers' lives, or shorten them – we just have one place to update.

Step 6: Adding More Flowers

Currently, the program creates flowers, runs, and then the flowers eventually all die. Let's add some functionality so that we keep adding new flowers throughout the program.

Before we write the code we need to make several decisions:

1. Which class should add flowers to the world as the Scenario executes
2. Should the flowers be added on a regular basis (like always on Monday) or on a random basis (once a week, but any day in the week)
3. Should we add one flower or multiple flowers at a time.

The answer to the first question is let the FlowerWorld class code add the flowers since there is only one FlowerWorld object.

For the 2nd question I have selected regular basis.

For the 3rd question I have 1 flower at a time.

In the **FlowerWorld** class, let's add some **member variables...**

- **ADD_FREQUENCY**, a static final int (private)
- **worldAge**, an int (private)

Initialize **worldAge** to 0 within the **FlowerWorld** constructor.

Final variables need to be assigned as soon as they're declared, so set **ADD_FREQUENCY** to any value sounds good to you (keep in mind that it is in game cycles).

Next, write a **checkToAddFlower()** method:

Return type: void Parameters: none

1. Check to see if the **worldAge** and **ADD_FREQUENCY** is evenly divisible via the modulus (%) operator. If it is...
    1. Use the **addObject** to add a **new Flower()** (using the constructor with no parameters) to create a new flower.
    2. X coordinate: Greenfoot.getRandomNumber(getWidth()),
    3. Y coordinate: getHeight() - 30

Your method should end up looking like:

```
public void checkToAddFlower()
{
    if ( worldAge % ADD_FREQUENCY == 0 )
    {
        addObject(new Flower(),
            Greenfoot.getRandomNumber( getWidth() ),
            getHeight() - 30 );
    }
}
```

This isn't called at the moment, and the worldAge isn't changing each cycle, either.

We now need to answer the following questions:

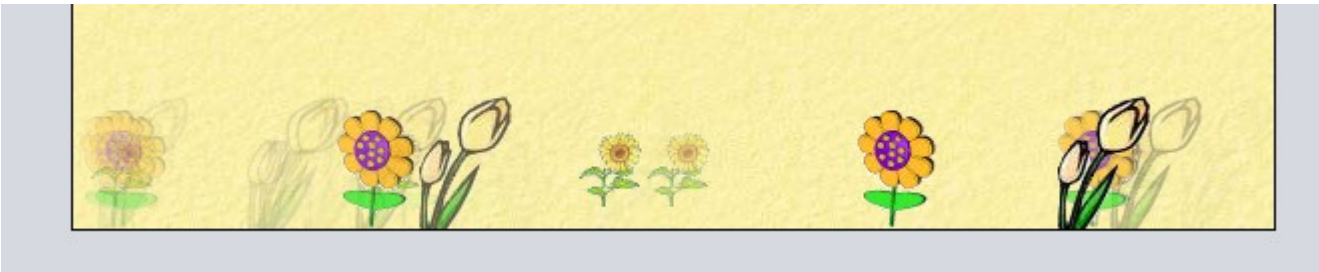
1. where to update **worldAge**
2. where to call **checkToAddFlower()**

Where should we add it?

Well, we can add an **act()** method to the **FlowerWorld**, and it will automatically be called when we hit **act** or **run**...

```
public void act()
{
    worldAge += 1;
    checkToAddFlower();
}
```

Now when we run the program, there will be a ***gradient of flowers wheeeee...***



Full code for reference

Flower.java

```
import greenfoot.*;

public class Flower extends Actor
{
    int age;
    public static final int FLOWER_LIFE = 200;

    public Flower()
    {
        age = 0;
        selectFlowerImage();
    }

    public Flower(int startingAge)
    {
        age = startingAge;
        selectFlowerImage();
    }

    public void act()
    {
        age += 1;
        if ( age % 30 == 0 )
        {
            fade();
        }
    }

    public void fade()
    {
        GreenfootImage myImageAssistant = getImage();
        int previousTransparency = myImageAssistant.getTransparency();
        int newTransparency = (int)(((double)previousTransparency) * .8);
        myImageAssistant.setTransparency(newTransparency);
    }

    public void checkToRemoveObject()
    {
        if ( age == 200 )
        {
            World myWorldAssistant = getWorld();
            myWorldAssistant.removeObject(this);
        }
    }

    public void selectFlowerImage()
    {
        int choice = Greenfoot.getRandomNumber(3);
        if ( choice == 0 )
        {
            setImage("flower2.png");
        }
        else if ( choice == 1 )
        {
            setImage("sunflower.png");
        }
```

```
        else if ( choice == 2 )
        {
            setImage("tulip.png");
        }
        else
        {
            setImage("ant3.png");
        }
    }
}
```

FlowerWorld.java

```
import greenfoot.*;

public class FlowerWorld extends World
{
    private static final int ADD_FREQUENCY = 20;
    int worldAge;

    public FlowerWorld()
    {
        super(600, 400, 1);
        populateFlowerBed();
        worldAge = 0;
    }

    public void act()
    {
        worldAge += 1;
        checkToAddFlower();
    }

    public void populateFlowerBed()
    {
        for ( int i = 0; i < 8 ; i++)
        {
            int randomAge = Greenfoot.getRandomNumber(Flower.FLOWER_LIFE);
            System.out.println( randomAge );
            addObject(new Flower(randomAge),
                Greenfoot.getRandomNumber(getWidth()),
                getHeight() - 30);
        }
    }

    public void checkToAddFlower()
    {
        if ( worldAge % ADD_FREQUENCY == 0 )
        {
            addObject(new Flower(),
                Greenfoot.getRandomNumber( getWidth() ),
                getHeight() - 30 );
        }
    }
}
```