Great Days in History: 28 September 1928

- 90 years ago…?
- Sir Alexander Fleming finds mould growing in his lab, leading to…?
- …the discovery of penicillin
- In honor of the occasion, the penicillium mold is rendered by computer:

Class #32:
More about Writing Methods

Software Design I (CS 120):  D. Mathias, 28 October 19

Simplifying the Penicillin Code

To do: Take the full code to draw the dish full of penicillium spores…
…and replace the code for the first "blob" with a private method that does the same thing

```java
private Window window;

public Penicillin()
{
    // Code to make window (omitted here)
    // declare the Oval variables to draw the image
    Oval dish1, dish2;
    Oval blob1, blob2, blob3, blob4, blob5, blob6;
    // draw the dish
    // (… code omitted here …)
    // create one penicillium blob
    blob1 = new Oval(140, 120, 120, 120);
    blob1.setBackground(Color.lightGray);
    blob2 = new Oval(150, 130, 100, 100);
    blob2.setBackground(Color.blue);
    window.add(blob1);
    window.add(blob2);
    // create second penicillium blob
    blob3 = new Oval(140, 280, 120, 120);
    blob3.setBackground(Color.lightGray);
    blob4 = new Oval(150, 290, 100, 100);
    blob4.setBackground(Color.blue);
    window.add(blob3);
    window.add(blob4);
    // create third penicillium blob
    // (… code omitted here …)
}
```

Simplifying the Penicillin Code

To do: Let's do the same thing for the dish

```java
private Window window;

public Penicillin()
{
    // Code to make window (omitted here)
    // declare the Oval variables to draw the image
    Oval dish1, dish2;
    dish1.setBackground(Color.green);
    dish2.setBackground(Color.yellow);
    window.add(dish1);
    window.add(dish2);
    // create one penicillium blob
    blob1 = new Oval(140, 120, 120, 120);
    blob1.setBackground(Color.lightGray);
    blob2 = new Oval(150, 130, 100, 100);
    blob2.setBackground(Color.blue);
    window.add(blob1);
    window.add(blob2);
    // create second penicillium blob
    // (… code omitted here …)
}
```
Control Flow with Methods

```java
public static void main(String[] args) {
    Penicillin p = new Penicillin();
p.drawDish();
p.drawBlob();
}

public Penicillin() {
    // code to draw window and dish
    // (code omitted here)
}

// draw Petri dish
private void drawDish() {
    // (code omitted here …)
}

// draw one penicillium blob
private void drawBlob() {
    // (code omitted here …)
}
```

Now, when `main()` runs:

1. Creates new `Penicillin()` object by calling constructor
2. Control passes to constructor, which makes window elements and basic dish image
3. Control returns to `main()`
4. `drawDish()` method is called
5. Control passes to that method, which also executes and then returns again to `main()`
6. `drawBlob()` method is called
7. Process repeats one last time, and then we come back to `main()` for the last time, and program is done

Adding Parameters to a Method

- To add input parameters:
  1. Figure out what you want to make different each time the method runs (here, we want to vary the \((x, y)\) location)
  2. Add parameters for each value you want to differ
  3. Find every occurrence of the things to change, and replace each fixed value with the corresponding parameter instead

- Now you have a much more powerful method!

```java
private void drawBlob(int x, int y) {
    Oval blob1, blob2;
    blob1 = new Oval(x, y, 120, 120);
    blob1.setBackground(Color.lightGray);
    blob2 = new Oval(x + 10, y + 10, 100, 100);
    blob2.setBackground(Color.blue);
    window.add(blob1);
    window.add(blob2);
}
```

Method Variables

- We place local variables (only used by this method) inside method
  - Not visible to or usable by any other part of the class
- **Problem**: this `drawBlob()` method is too specific!
  - To draw the other blobs in their different locations, we would need two more private methods, one for each blob
  - **To do**: Fix this problem. (That is, add input parameters to the method, so that you only need one method to draw all the blobs, with each at a different locations)

```java
private void drawBlob(int x, int y) {
    // Code goes here…
}
```

Method Parameters

- Method parameters are **variables** like any other
- Each must be **properly declared** when first introduced
- This requires that each one be given a type and an identifier, following all the normal rules for each
- Each parameter in the list is separated by commas

```java
private void drawBlob(int x, int y) {
    // Code goes here…
}
```
Parameters as Local Variables

- When we add parameters to a method, they also act like **local variables** during code execution.
- `x` and `y` are only visible/usable **inside** this method.
- They take different values, depending upon what is passed to them as input.

```java
private void drawBlob(int x, int y) {
    Oval blob1, blob2;
    blob1 = new Oval(x, y, 120, 120);
    blob2 = new Oval(x + 10, y + 10, 100, 100);
    blob1.setBackground(Color.lightGray);
    blob2.setBackground(Color.blue);
    window.add(blob1);
    window.add(blob2);
}
```

The return Statement

- We can mark the end of a method using simple code:
  ```java
  return;
  ```
- This will return control to the exact location in the code where the method was originally called.
- For simple `void` methods, we often do not bother.
- The method automatically returns control when it reaches the end of its code-block.
- Sometimes, however, we will want to explicitly return control, particularly when we want to end the method early in some situations.

```java
// draw one penicillium blob
private void drawBlob(int x, int y) {
    Oval blob1, blob2;
    blob1 = new Oval(x, y, 120, 120);
    blob2 = new Oval(x + 10, y + 10, 100, 100);
    blob1.setBackground(Color.lightGray);
    blob2.setBackground(Color.blue);
    window.add(blob1);
    window.add(blob2);
}
```

Local Values of Input Parameters

- Now, each time the method is called, the values of `x` and `y` change.
  1. 1st call sets our variables: `x == 140, y == 120`
  2. Runs, after which point `x, y` are forgotten (and they no longer exist!)
  3. 2nd call runs with new parameter values: `x == 140, y == 280`
  4. Runs, then forgets again
  5. etc...

```java
public static void main(String[] args) {
    // run constructor to set up window
    Penicillin pen = new Penicillin();
    // Draw the dish.
    pen.drawBlob();
    // create the penicillium blobs
    pen.drawBlob(140, 120);
    pen.drawBlob(140, 280);
    pen.drawBlob(280, 200);
    // draw one penicillium blob
    private void drawBlob(int x, int y) {
        Oval blob1, blob2;
        blob1 = new Oval(x, y, 120, 120);
        blob2 = new Oval(x + 10, y + 10, 100, 100);
        blob1.setBackground(Color.lightGray);
        blob2.setBackground(Color.blue);
        window.add(blob1);
        window.add(blob2);
    }
}
```

Review: public vs. private Method Access

- If we make a method in a class C **private**, it can only be used by object instances of the class C itself.
- If we make a method in C **public**, it can be used by:
  1. Any object instance from the class C itself
  2. Any other class that runs the C() class constructor and creates an instance of an object of type C to use
- When we write methods, we should decide if we want other coders to be able to use those methods or not.
  - **YES**: then make it public
  - **NO**: then make it private
Suppose we made the blob-drawing method public in Penicillin.

Then, if we give someone the Penicillin class code to use (like we have been using pre-written code for Window, Oval, etc.), they can write more code, like the following:

```java
public class Main
{
    public static void main(String[] args)
    {
        Penicillin pen = new Penicillin();
        pen.drawBlob(0, 0);
        pen.drawBlob(0, 400);
        pen.drawBlob(400, 0);
    }
}
```

Do you want this to be possible? You decide!

If you don’t make method private, in which case the compiler will not allow it.

Note: if we don’t want them to even be able to instantiate a Penicillin object at all, then we could also make the constructor private.

This Week & Next

- **Meetings this week:**
  - Monday/Wednesday: regular classroom
  - Tuesday/Friday: in the CS Lab (16 Wing)

- **Reading 05:** Ch. 6 due Thursday, Oct 31 at Noon

- **Program 05:** due 11:59 PM, Wednesday October 30

- **Midterm 2:** Monday November 11

- **Office Hours:** Wing 212
  - Monday/Friday: 2:15 PM–3:15 PM
  - Tuesday: 2:30 PM–1:30 PM
  - Wednesday: 12:05 PM–1:00 PM