CS514 – Fall 2025

Lab 1: Falling Penny

Instructor: Nancy Stevens

Due: Tuesday, 9/2/2025 at 11:59pm

INTRODUCTION

In this lab, we'll learn the basic syntax and structure of a Java program. Some content is derived from Modeling and Simulation in Python. We will be writing a program to calculate the speed of a falling penny.

DESCRIPTION

"I heard that, if you dropped a penny off of the Empire State Building, it would be falling so quickly that it could cut a person in half."

Let's figure out if we think that is true.

If we dig out our Physics text, we can learn that the formula for distance traveled is:

```
distance = acceleration * time^2 / 2
```

If we solve for time, we get:

```
time = sqrt(2 * distance / acceleration)
```

We know that acceleration (on earth, at sea level) is **`9.8 m/s^2.`**

The Empire State Building is **381 meters** tall. How long will it take to hit the ground?

PROCESS

Step 1. Start up IntellIJ. Add your name to the top of the file.

We'll be working with the **FallingPenny.java** file in the course repo under **labs**.

a) Add variables for

- `fallingTime`
- `acceleration`
- `ESBheight.`

What data types should they have?

b) Write an expression that computes **`fallingTime.`**

c) Add a print statement that displays your answer. It should print: *"It takes X seconds to reach the ground."*

d) Compile and run your code in IntellIJ.

So how fast is the penny moving at this point?

```
velocity = time * acceleration.
```

Step 2. Write an expression for this, and a print statement that displays the velocity.

Does this seem to match with reality? Why don't we hear of pocket-change-related disasters all the time?

We're leaving out air resistance. As the penny falls, the air pushes back up on the penny, slowing it down. It turns out that once the penny reaches *18 m/s,* these forces equal out, and the penny stops accelerating. How long will this take?

Step 3: Create a variable called **`timeToTerminalVelocity`**, and an expression that computes this. How far will the penny fall during this period?

Step 4: Create a variable called **`accelDistance`**, and an expression that computes it.

How long will it take to fall the remaining distance?

Step 5: Create a variable called **`timeAtTerminalVelocity`**, and an expression that computes it.

So far, this is really interesting, but what if we want to throw pennies off of other buildings, or go to other planets? We could redo all of this, but programmers are very lazy, and so we love to avoid doing things more than once.

Instead, we'll create a *static* method. This is a reusable piece of code that lets us provide different parameters for an expression.

Step 6: Make a method called **`fallingTime`**.

It should take one parameter, called **`fallingDistance`**, as input, and return the time needed to fall a given distance, according to the expression above.

What should the data types be?

Try it out with different distances.

What if we decide to visit other planets? 9.8m/s^2 is the acceleration due to gravity at sea level on Earth. Here's a few other fun places:

- on the moon: 1.6 m/s^2
- on mars: 3.69 m/s^2
- on jupiter: 24.79 m/s^2
- on saturn: 10.44 m/s^2
- on titan: 1.4 m/s^2
- on venus: 8.87 m/s^2

Step 7: Make a method called **`spaceFallingTime`** that takes two parameters: a *distance* and a *local acceleration,* and returns the *time elapsed*.

Remember how programmers are lazy? I don't want to remember all of those numbers! Let's make a helper function called **`getAcceleration`**.

Step 8: make a method called **`getAcceleration`**. It will take a string, representing our *location*, and return the *appropriate value*.

We'll do this using a *conditional*, or *if*, statement.

What if the user asks for a planet we don't know?

There's a few different ways to handle this - we could use a default value, print an error, or create an exception. For now, let's just print out an error message.

Step 9: Extend your conditional to have an *'else'* statement that prints an error message and returns a default value.

Now, let's add in some user input. Rather than hardcoding the location and the building height, let's get that from the user.

Step 10: Return to the main method and change it to prompt the user for a building height and a location. Please use the ***Scanner class*** to read this in. Call `spaceFallingTime` and print out the result. Enclose this in a *while loop*, so that the user can execute multiple queries.

REQUIREMENTS

Complete all the steps above. Please include your name in a comment at the top of `FallingPenny.java`.

GRADING RUBRIC

This homework will be graded using specifications grading. There are a sequence of tasks to complete. Grades will be assigned as follows:

Grade Level	Tasks to Complete Per Level
C	Program compiles; Steps 1-4
B	Complete all C-Level steps, plus steps 5-8 complete
A	Complete all the B-Level, plus steps 9 and 10

Submission Details

Congratulations! You are done with Lab 1. To turn it in, please do a git commit and git push to push your completed code up to GitHub. Please submit your code to your GitHub repo in a directory called **Lab1** by the due date. Late submissions are not accepted unless you are using a token.