Lunar Lander

Computer *simulation* is used by engineers to test out ideas before actually building expensive machines or putting people in dangerous situations. Simulation is a critical part of the the space program by NASA, for example.

For this program, you will create a simulation of a vehicle landing on the moon. (It turns out that this assignment is also a rather old computer game that has been around for decades.)

Here is how it works: You are in control of a lunar lander ship, descending to the surface of the moon for a landing. Gravity steadily accelerates your ship faster and faster toward the surface of the moon. You, the astronaut piloting the ship, have a single control: a button with the label "thrust" on it. Applying thrust slows your ship down. Your goal is to get your ship to land on the moon at a slow enough speed so that it doesn't crash on impact. What's the catch? You have only a limited amount of fuel. If you slow down your ship too much too early, you will run out of fuel and crash into the surface of the moon.

Your mission: Program this simulation in Java.

Details

You should create two classes: `LunarLander`, and `Simulation`. The `LunarLander` class represents the ship itself, and is the class that you will create an object from. The `Simulation` class is the one that has the `main` method in it, and controls how the simulation works. In your `LunarLander` class, you will need to keep the following information in instance variables:

- altitude: how far the ship is from the surface of the moon, in meters. Initial value: 1000 meters.
- velocity: how fast the ship is moving, in meters/second. A positive velocity means that that ship is moving toward the moon's surface. A negative velocity means that the ship is moving away. Initial value: 40 meters/second.
- fuel: how many units of fuel are left in the tank. Initial value: 25 units.

Your LunarLander object should have the following methods (you may have more if you wish):

- a constructor that initializes all your instance variables
- `public int getVelocity()` returns the velocity your lander
- `public int getAltitude()` returns the altitude of your lander
- `public int getFuel()` returns the units of fuel left in your lander
- `public void thrust()` applies one unit of fuel to give thrust to the ship, and decreases the velocity by 4 meters/sec
- `public void doOneSecond(int fuelUnits)` simulate one complete second of activity, where the astronaut has requested to burn fuelUnits units of fuel. Specifically, do the following:

    1. Expend `fuelUnits` of fuel from the fuel tank. Make sure you check to see if you actually have this much fuel. If not, burn all that you have.
    2. For each of unit of fuel that you burn, call the `thrust()` method to decrease your velocity.

3. Determine the new altitude for your lander. Your new altitude is your old altitude minus your velocity, minus an additional 2 meters/second for gravity. Gravity adds 2 meters/second to your velocity every second.

Your `Simulation` class should have one method: `public static void main(String[] args)`. This method should create a LunarLander object, provide the player with a welcome message, then repeatedly show the player the current information for the lander followed by a question as to how many units of thrust to burn. If the lander ends up hitting the surface of the moon with a velocity of 4 meters/second or less, the ship lands successfully! If the lander hits the surface with a velocity of 5 meters/second or more, the ship crashes. You'll find a sample game at the bottom of this assignment.

Bonus work

If you want to push yourself further, create an alternative simulation class called `SimulationRace` that is very similar to `Simulation` but has *two* `LunarLander` objects descending to the moon, each controlled by different users. The two players alternate back and forth entering thrust for their own landers, until one them lands or until they both crash. I'm leaving the specifications of this fairly vague: have fun with it.

Hints, Suggestions, Etc.

- Landing without crashing is tricky. To test your program, make the initial altitude smaller and the initial fuel amount bigger. This makes the game quicker to play and easier to win.
- Any numbers that you see scattered throughout this assignment that are not used for initializing the lander (gravity increases velocity by 2 meters/second, 4 meters/second or less to land successfully, etc.) should be stored as "final" variables (also known as constants) in your program.
- It is useful to be able to automatically exit the program, rather than having to wait until a successful land or crash. Set up your program so that if the player enters -1 for thrust, the game automatically ends. (The method `System.exit(0)` can be used to automatically exit your program.)

Here is a sample simulation:

```
Welcome to Lunar Lander!

Alt = 1000 Vel = 40 Fuel = 25
How much thrust this round? 0
Alt = 958 Vel = 42 Fuel = 25
How much thrust this round? 0
Alt = 914 Vel = 44 Fuel = 25
How much thrust this round? 0
Alt = 868 Vel = 46 Fuel = 25
How much thrust this round? 0
Alt = 820 Vel = 48 Fuel = 25
How much thrust this round? 0
Alt = 770 Vel = 50 Fuel = 25
How much thrust this round? 0
```

```
Alt = 718 Vel = 52 Fuel = 25
How much thrust this round? 0
Alt = 664 Vel = 54 Fuel = 25
How much thrust this round? 0
Alt = 608 Vel = 56 Fuel = 25
How much thrust this round? 0
Alt = 550 Vel = 58 Fuel = 25
How much thrust this round? 0
Alt = 490 Vel = 60 Fuel = 25
How much thrust this round? 0
Alt = 428 Vel = 62 Fuel = 25
How much thrust this round? 0
Alt = 364 Vel = 64 Fuel = 25
How much thrust this round? 0
Alt = 298 Vel = 66 Fuel = 25
How much thrust this round? 0
Alt = 230 Vel = 68 Fuel = 25
How much thrust this round? 0
Alt = 160 Vel = 70 Fuel = 25
How much thrust this round? 2
Alt = 96 Vel = 64 Fuel = 23
How much thrust this round? 0
Alt = 30 Vel = 66 Fuel = 23
How much thrust this round? 0
Alt = -38 Vel = 68 Fuel = 23
Oh no, you crashed!
```