Methods of the Vic class

Summary of what a Vic can do:

- `new Vic()` creates a Vic object.

- You can send **four** action messages to a Vic:
  1. `aVic.putCD()` – causes the mechanical arm to remove a CD from the top of the stack and put it in the slot at the current position. This does not change the arm's position in the sequence. If a CD is already in the slot, or if no CD is in the stack, the `putCD` operation will cause the program to fail.

  2. `aVic.takeCD()` - causes the mechanical arm to take a CD out of the slot at the current position and place it on top of the stack. This does not change the arm's position in the sequence. If there is no CD in the slot, the `takeCD` operation will cause the program to fail.

  3. `aVic.moveOn()` - moves the mechanical arm down from its current position in the sequence of slots to the next position.

  4. `aVic.backUp()` - moves the mechanical arm up from its current position in the sequence of slots to the position just before it.

- You can ask **two** questions of a Vic:
  1. `aVic.seesCD()` – is true if aVic's current slot has a CD in it and is false otherwise. Evaluation of this condition causes the program to fail if a Vic.seesSlot() is false.

  2. `aVic.seesSlot()` – is true if aVic is not past its last slot and is false otherwise. So it means aVic actually has a current slot.

- You have **four** Vic class methods:
  1. `Vic.stackHasCD()` – is true if there is at least on CD on the stack.

  2. `Vic.emptyStack()` – empties (or clears out) any CD's that happen to be on the stack.

  3. `Vic.reset(args)` – this method initializes where CD's are when the program starts. If this method is not called, the CD's are in random locations. (See "Testing Your Vic" for more info)

  4. `Vic.say("whatever")` – Prints out the expression to the drawing window (upper left)

- If `aVic.seesSlot()` is `false`, then `aVic.putCD()`, `aVic.takeCD()`, `aVic.moveOn()`, and `aVic.seesCD()` all cause the program to fail (i.e., gracefully terminate execution). Also, the `backUp` message causes the program to fail if the Vic object is positioned at its first slot. You should avoid letting this happen. If you let it happen, your program is not **robust**, since it does not handle unexpected input well.

Testing your Vic!!

(keep this for reference)

A good practice to get into in programming is to test your programs thoroughly. You would hate to be the person who was designing the software that goes into a Heart Pacemaker and find after three people die that your software had an error (called a software bug).

Currently, the Vic controller randomly generates CD collections to control. This is good once you have a program that you think works well, but while you are in the process of testing, it is nice to have consistent test data.

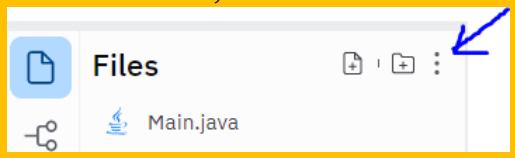
In Java programs, you have a way to do this. The parameter in the main method named `args` is the way.

For a Vic, it is fairly straightforward. You need to set the `args` parameter to a binary number, 1 meaning a CD exists, 0 means there is no CD in the slot.

So, for example, 011001 would mean that there are CD's in slots 2, 3, and 6 for only one collection. Another example: 11100 010 1111 would mean that there are 3 collections. The first collection has CD's in slots 1, 2, 3 (out of 5 slots); the second has a CD in slot 2 (out of 3 slots); the third has CD's in all four slots.

So….how do you set the parameter `args` to what you desire? Follow the steps below:

1. First of all, you should have a **Repl** with the files that you are working on.
2. In the files view, click on the three vertical dots on the upper right



3. Click on "Show hidden files". You will now see a file named **.replit**
4. On line 2 of **.replit**, make it look like this:

```
run = ["java", "-classpath", ".:target/dependency/*", "Main", "011001"]
```

5. add the following line to the beginning of your java program (the first line in **main**):

```
Vic.reset(args);
```

6. Run your program

This should make a CD collection with CDs in slots 2, 3, and 6

If you want to change where you want CDs, change that binary number to something else.

For example,

```
run = ["java", "-classpath", ".:target/dependency/*", "Main",
"11100", "010", "1111"]
```

would mean that there are 3 collections. The first collection has CD's in slots 1, 2, 3 (out of 5 slots); the second has a CD in slot 2 (out of 3 slots); the third has CD's in all four slots.