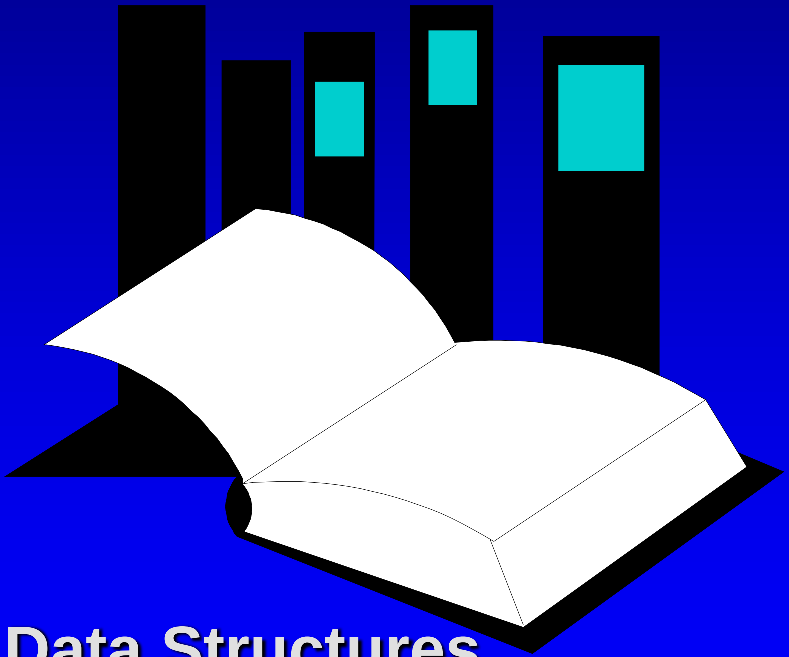
Object Oriented Programming

OOP

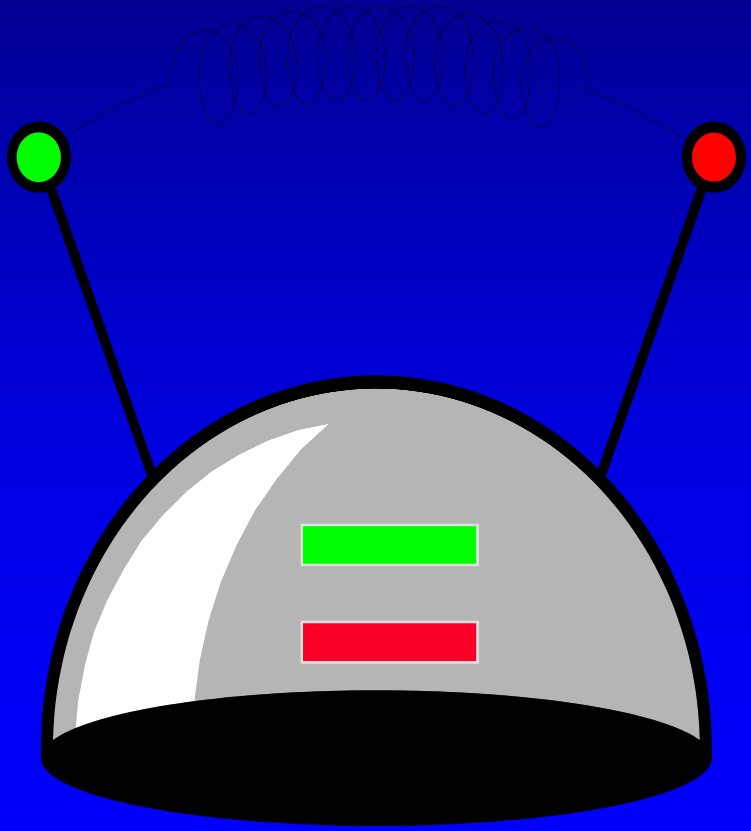


Data Structures and Other Objects Using Java

- Chapter 2 introduces Object Oriented Programming.
- OOP is a relatively new approach to programming which supports the creation of new data types and operations to manipulate those types.
- This presentation introduces OOP.

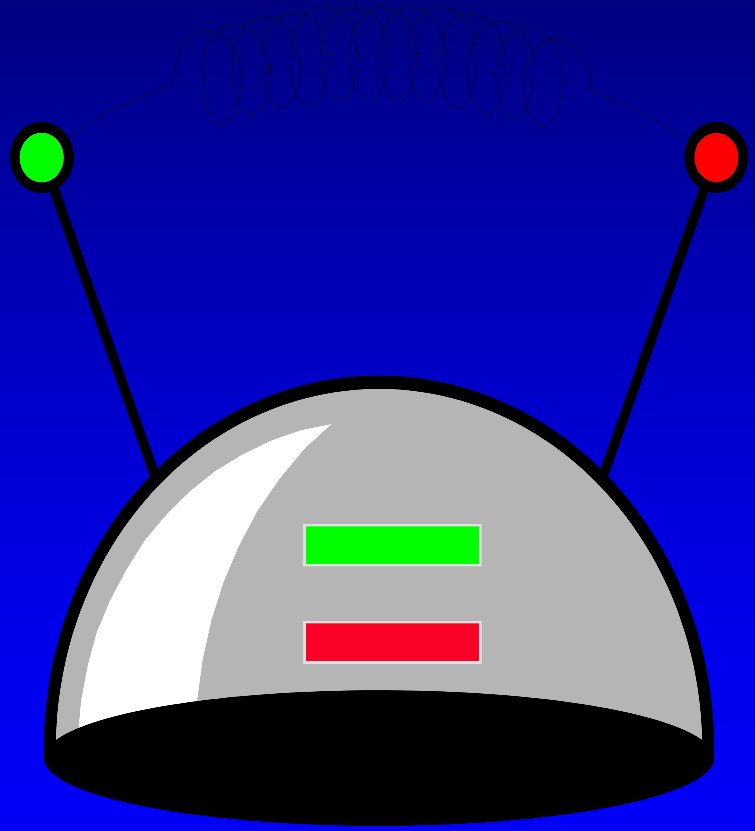
What is this Object ?

- There is no real answer to the question, but we’ll call it a “thinking cap”.
- The plan is to describe a thinking cap by telling you what actions can be done to it.

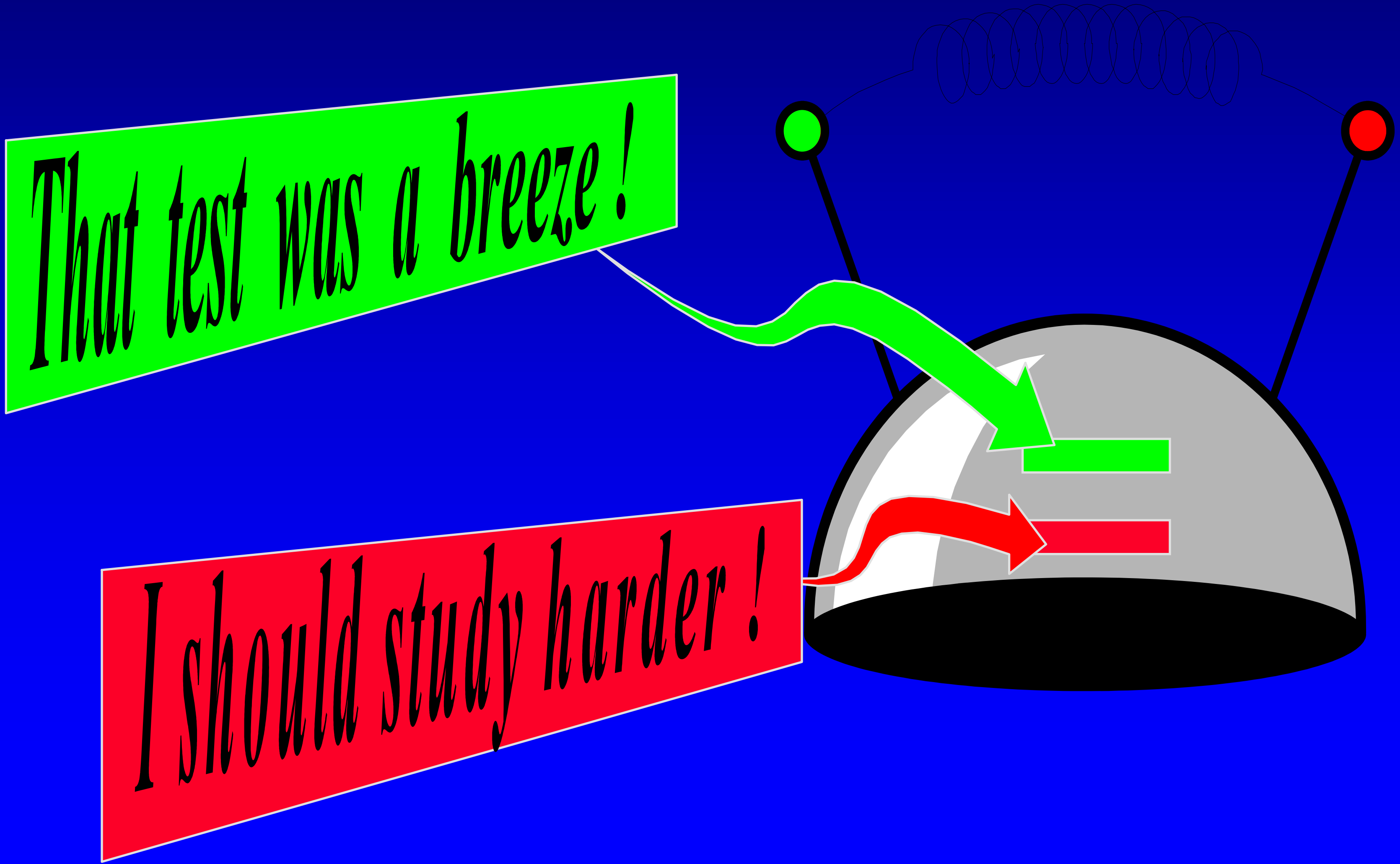


Using the Object's Slots

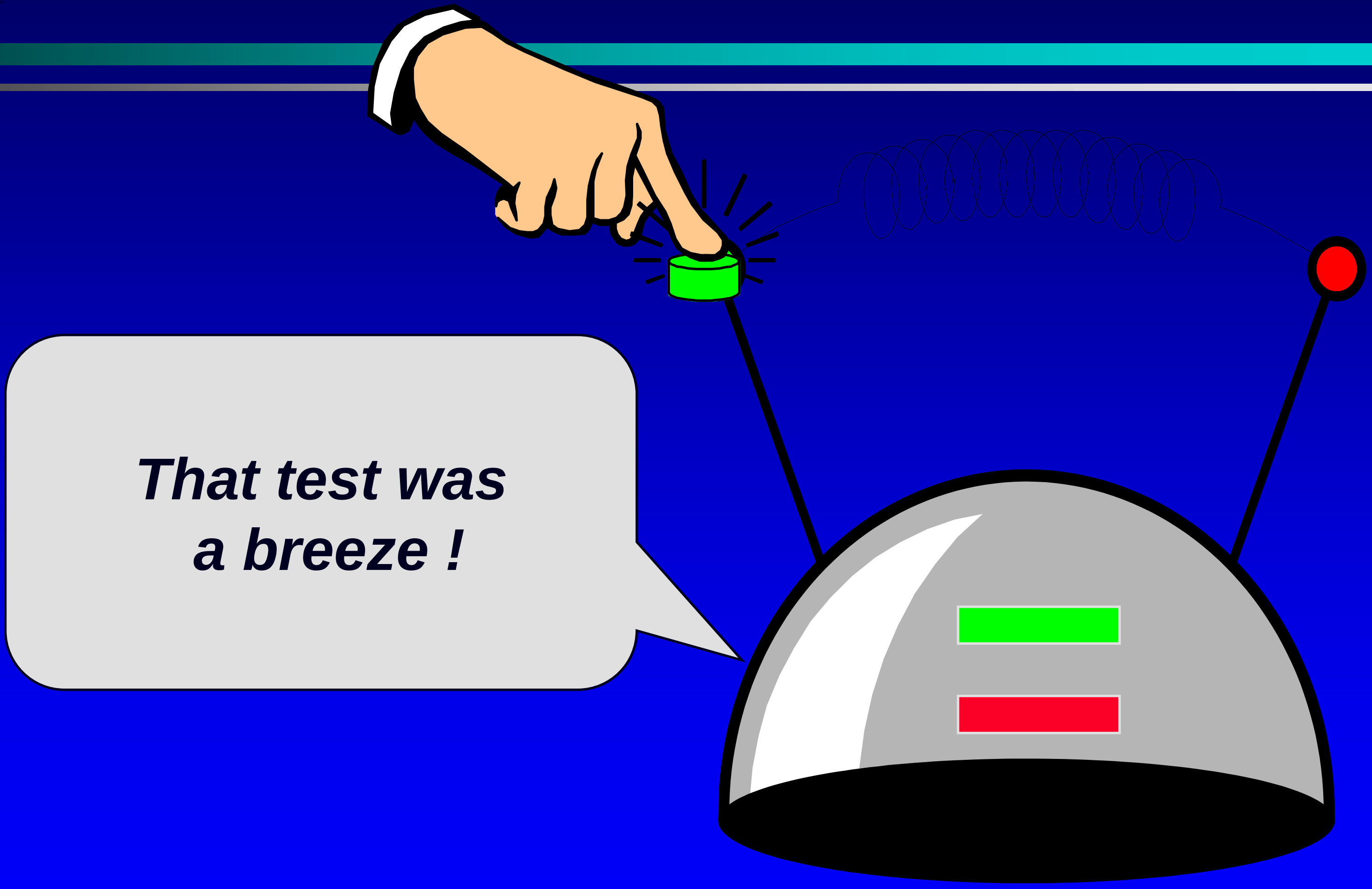
- You may put a piece of paper in each of the two slots (green and red), with a sentence written on each.
- You may push the green button and the thinking cap will speak the sentence from the green slot's paper.
- And same for the red button.



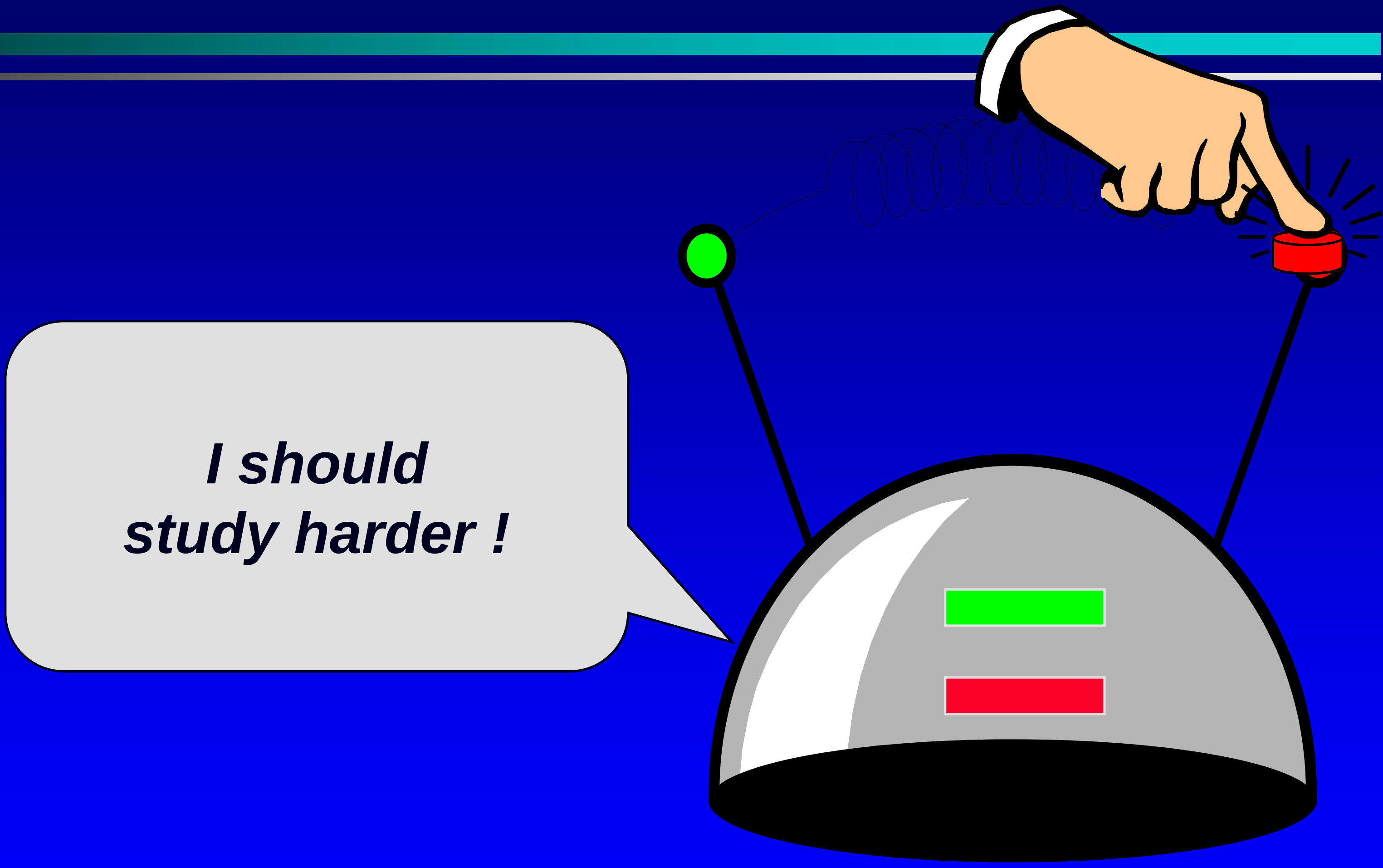
Example



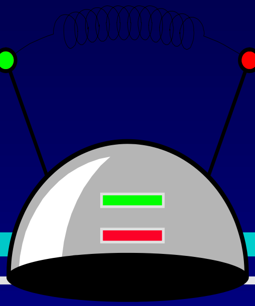
Example



Example



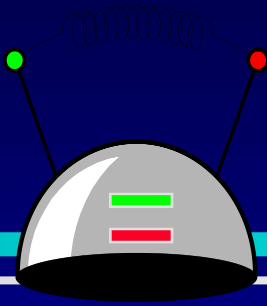
Thinking Cap Implementation



- We can implement the thinking cap using a data type called a class.

```
public class ThinkingCap
{

        . . .

 }
```

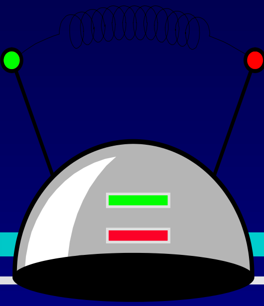
Thinking Cap Implementation



- The class will have two components called greenWords and redWords. These compnents are strings which hold the information that is placed in the two slots.
- Using a class permits two features . . .

```
public class ThinkingCap
{
    String greenWords;
    String redWords;
     . . .

}
```

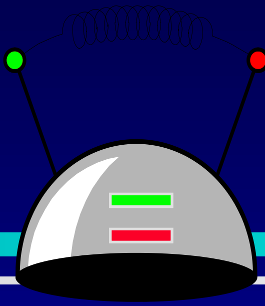
Thinking Cap Implementation



- The two components will be private instance variables. This ensures that nobody can directly access this information. The only access is through methods that we provide for the class.

```
public class ThinkingCap
{
   private char greenWords;
   private char redWords;
    . . .
}
```

Thinking Cap Implementation

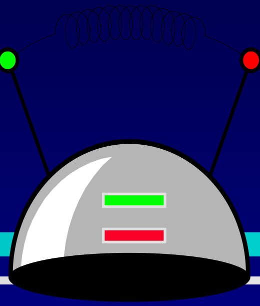


- In a class, the methods which manipulate the class are also listed.

```
class ThinkingCap
{
   private char greenWords;
   private char redWords;
    . . .
}
```

Implementations of the thinking cap methods go here.

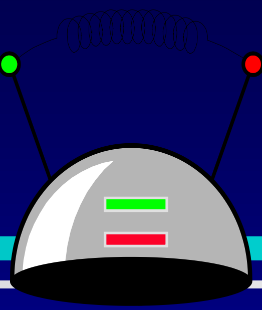
Thinking Cap Implementation



Our thinking cap has at least three methods:

```
public class ThinkingCap
{
   private char greenWords;
   private char redWords;

   public void slots(String newGreen, String newRed)...
   public void pushGreen( )...
   public void pushRed( )...

}
```

Thinking Cap Implementation

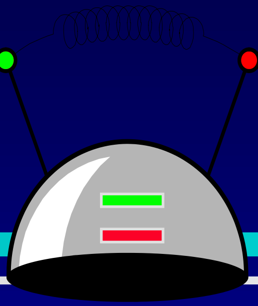


The code for a new class is generally put in a Java package, as shown here:

```
package edu.colorado.simulations;
public class ThinkingCap
{
   private char greenWords;
   private char redWords;

   public void slots(String newGreen,      ing
   public void pushGreen( )...
   public void pushRed( )...

}
```

This means that ThinkingCap.java and ThinkingCap.class files will be in a subdirectory edu/colorado/simulations

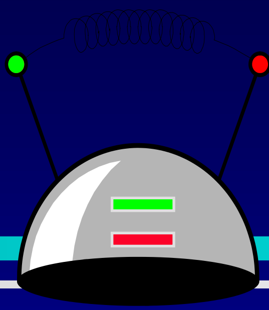
Using the Thinking Cap



- A program that wants to use the thinking cap can **import** the ThinkingCap class.

```
import
edu.colorado.simulations.ThinkingCap;

...
```

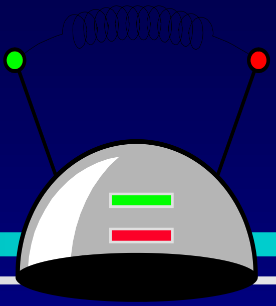
Using the Thinking Cap



- Just for fun, the example program will declare two ThinkingCap variables named student and fan.

```
import
edu.colorado.simulations.ThinkingCap;

public class Example
{
    public static void main( )
    {
        ThinkingCap student;
        ThinkingCap fan;
```

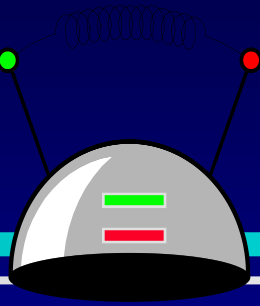
Using the Thinking Cap



- The variables are examples of reference variables, which means that they have the capability of refering to ThinkingCap objects that we create with the new operator.

```
import
edu.colorado.simulations.ThinkingCap
;

public class Example
{
    public static void main( )
    {
        ThinkingCap student;
        ThinkingCap fan;

         student = new ThinkingCap( );
         fan = new ThinkingCap( );
```

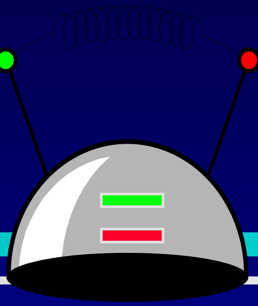
Using the Thinking Cap



- Once the ThinkingCaps are created, we can activate methods such as slot for the student thinking cap.

```
import
edu.colorado.simulations.ThinkingCap
;

public class Example
{
    public static void main( )
    {
        ThinkingCap student;
        ThinkingCap fan;

         student = new ThinkingCap( );
         fan = new ThinkingCap( );
         student.slots( "Hello",  "Bye");
```

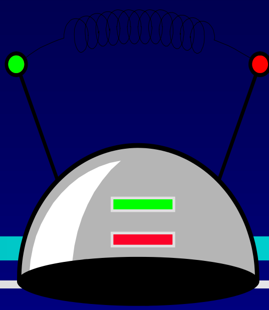
Using the Thinking Cap



- Once the ThinkingCaps are created, we can activate methods such as slot for the student thinking cap.

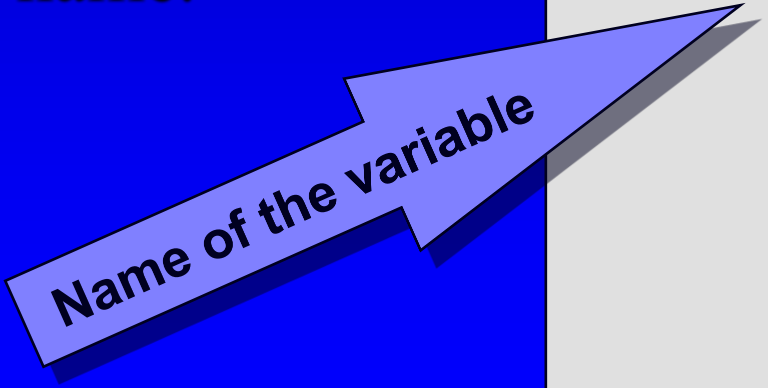
```
import
edu.colorado.simulations.ThinkingCap;

public class Example
{
    public static void main(String[ ] args)
    {
        ThinkingCap student;
        ThinkingCap fan;

         student = new ThinkingCap( );
         fan = new ThinkingCap( );
         student.slots( "Hello",  "Bye");
```

Using the Thinking Cap

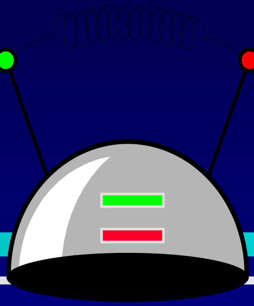


- The method activation consists of four parts, starting with the variable name.

```
student.slots( "Hello",  "Bye");
```

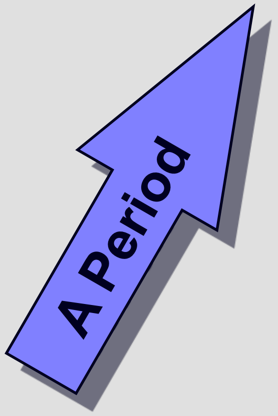


Using the Thinking Cap

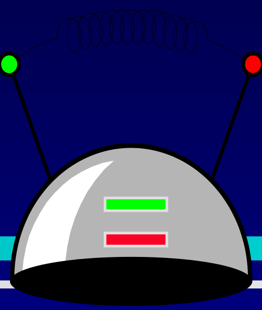


- The variable name is followed by a period.

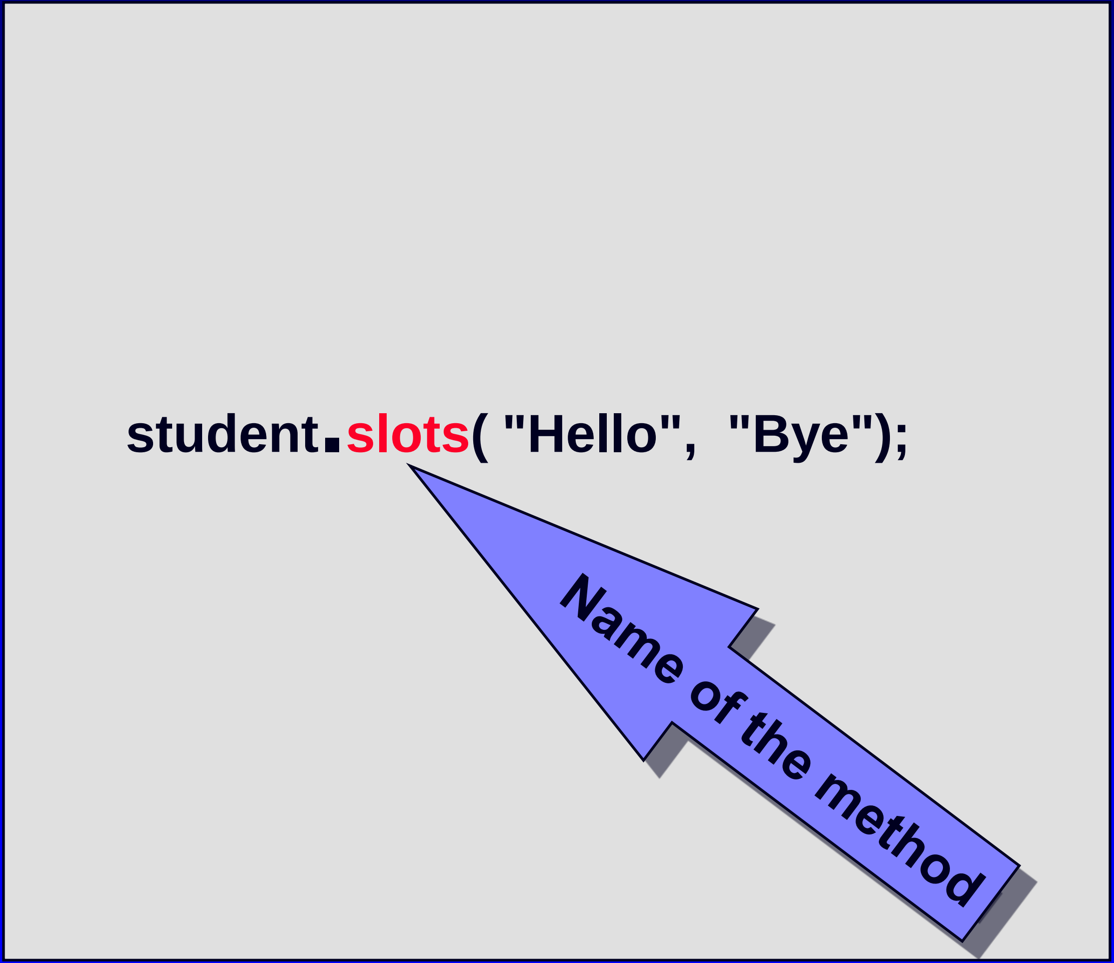
```
student.slots( "Hello",  "Bye");
```



Using the Thinking Cap



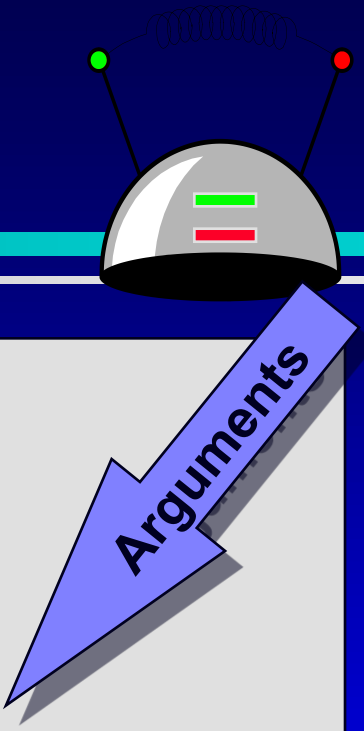
- After the period is the name of the method that you are activating.



```
student.slots( "Hello",  "Bye");
```

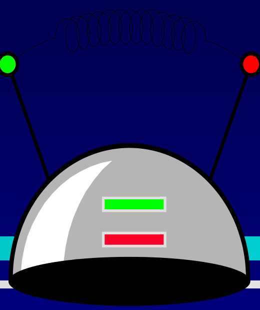
Using the Thinking Cap

- Finally, the arguments for the method. In this example the first argument (newGreen) is "Hello" and the second argument (newRed) is "Bye".



```
student.slots( "Hello",  "Bye");
```

A Quiz

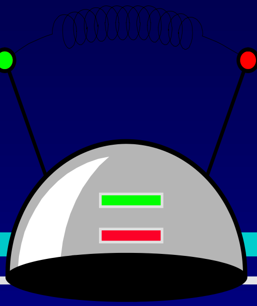


How would you activate student's pushGreen method ?

What would be the output of student's pushGreen method at this point in the program ?

```
public static void main(String[ ] args)
{
     ThinkingCap student;
     ThinkingCap fan;

      student = new ThinkingCap( );
      fan = new ThinkingCap( );

      student.slots( "Hello",  "Bye");
```

A Quiz

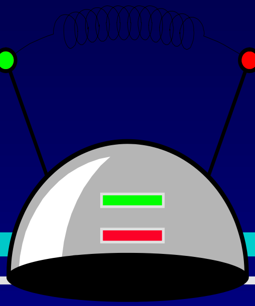


Notice that the **pushGreen** method has no arguments.

At this point, activating **student.pushGreen** will print the string **Hello**.

```
public static void main(String[ ] args)
{
     ThinkingCap student;
     ThinkingCap fan;

      student = new ThinkingCap( );
      fan = new ThinkingCap( );

      student.slots( "Hello",  "Bye");
      student.pushGreen( );
```

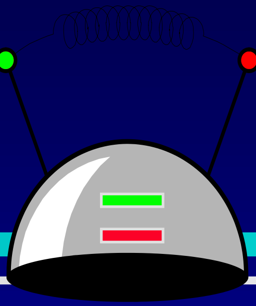
A Quiz



```
public static void main(String[ ] args)
{
    ThinkingCap student;
    ThinkingCap fan;
    student = new ThinkingCap( );
    fan = new ThinkingCap( );
    student.slots( "Hello",  "Bye");
    fan.slots( "Go Cougars!", "Boo!");
    student.pushGreen( );
    fan.pushGreen( );
    student.pushRed( );
    . . .
```

Trace through this program, and tell me the complete output.

A Quiz



```
public static void main(String[ ] args)
{
   ThinkingCap student;
   ThinkingCap fan;
   student = new ThinkingCap( );
   fan = new ThinkingCap( );
   student.slots( "Hello",  "Bye");
   fan.slots( "Go Cougars!", "Boo!");
   student.pushGreen( );
   fan.pushGreen( );
   student.pushRed( );
   . . .
```

Hello

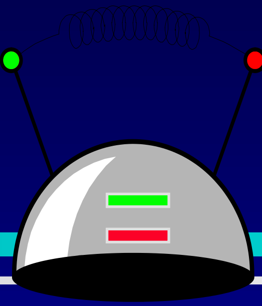
Go Cougars!

Bye

What you know about Objects

- ü Class = Data + Methods.
- ü You know how to write a new class type, and place the new class in a package.
- ü You know how to import the class into a program that uses class type.
- ü You know how to activate methods.
- û But you still need to learn how to write the implementations of a class's methods.

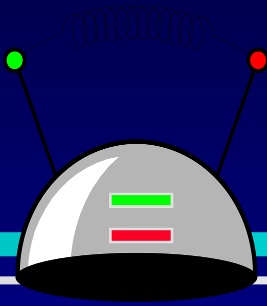
Thinking Cap Implementation



We will look at the body of slots, which must copy its two arguments to the two private instance variables.

```
public class ThinkingCap
{
    private String greenWords;
    private String redWords;

    public void slots(String newGreen, String newRed)...
    public void pushGreen( )...
    public void pushRed( )...

}
```

Thinking Cap Implementation

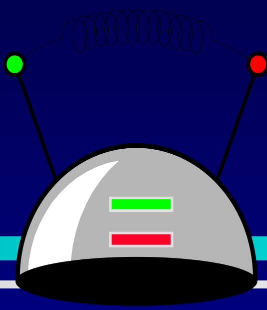


The method's implementation occurs after the parameter list

```
public void slots(String newGreen, String newRed)
{
    greenWords = newGreen;
    redWords = newRed;
}
```

There is one feature about a method's implementation . . .

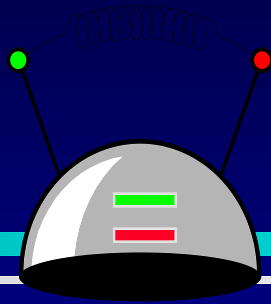
Thinking Cap Implementation



Within the body of the method, the class's instance variables and other methods may all be accessed.

```
public void slots(String newGreen, String newRed)
{
    greenWords = newGreen;
    redWords = newRed;
}
```

Thinking Cap Implementation



Within the body of the method, the class's instance variables and other methods may all be accessed.

```
public void slots(String newG
{
    greenWords = newGreen;
    redWords = newRed;
}
```

But, whose instance variables are these? Are they

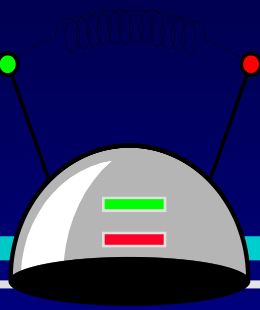
student.greenWords

student.redWords

fan.greenWords

fan.redWords

Thinking Cap Implementation



Within the body of the method, the class's instance variables and other methods may all be accessed.

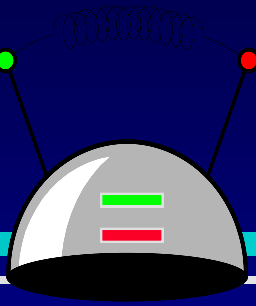
```
public void slots(String newG
{
    greenWords = newGreen;
    redWords = newRed;
}
```

If we activate student.slots:

student.greenWords

student.redWords

Thinking Cap Implementation



Within the body of the method, the class's instance variables and other methods may all be accessed.

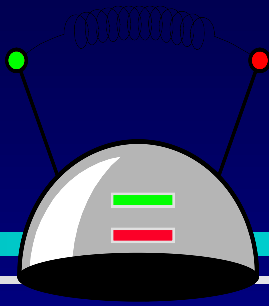
```
public void slots(String newG
{
    greenWords = newGreen;
    redWords = newRed;
}
```

If we activate fan.slots:

fan.greenWords

fan.redWords

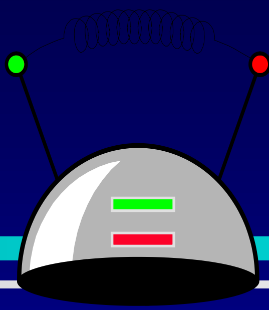
Thinking Cap Implementation



Here is the implementation of the pushGreen method, which prints the green Words:

```
public void pushGreen( )
{
    System.out.println(greenWords);
}
```

Thinking Cap Implementation



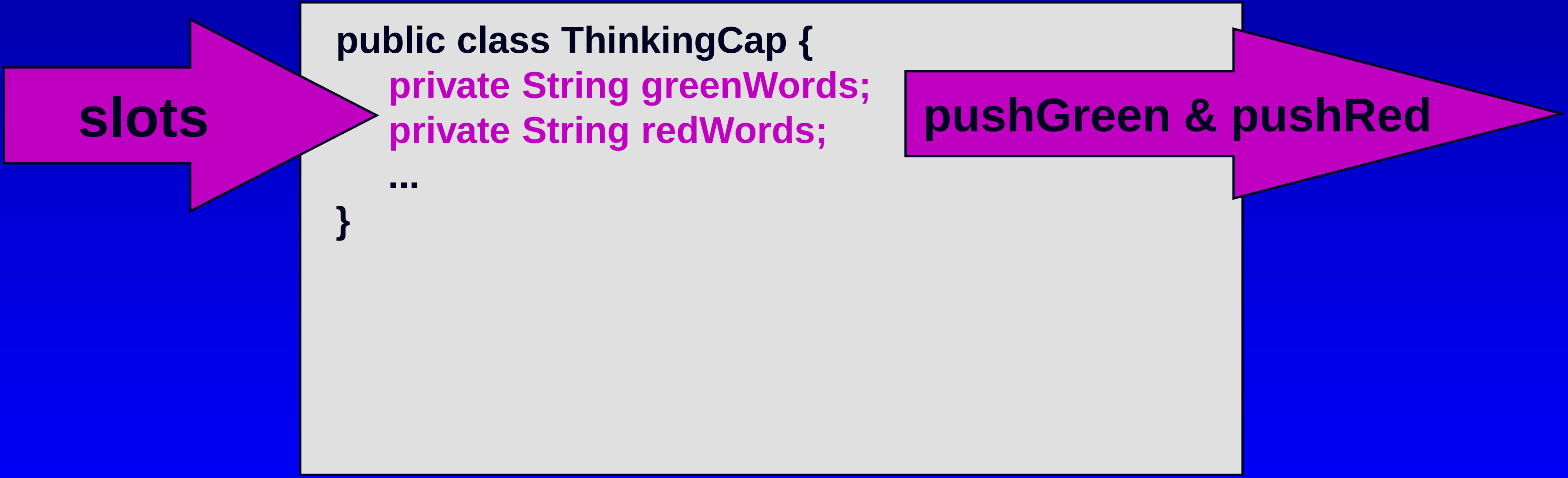
Here is the implementation of the pushGreen method, which prints the green Words:

```
public void pushGreen( )
{
    System.out.println(greenWords);
}
```

Notice how this method implementation uses the greenWords instance variable of the object.

A Common Pattern

- Often, one or more methods will place data in the instance variables...

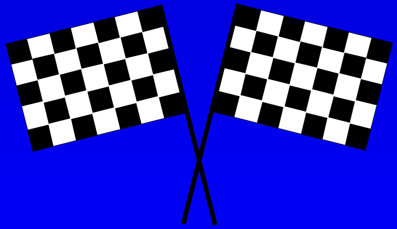


```
public class ThinkingCap {
    private String greenWords;
    private String redWords;
    ...
}
```

- ...so that other methods may use that data.

Summary

- Classes have instance variables and methods. An object is a variable where the data type is a class.
- You should know how to declare a new class type, how to implement its methods, how to use the class type.
- Frequently, the methods of an class type place information in the instance variables, or use information that's already in the instance variables.
- In the future we will see more features of OOP.



THE END