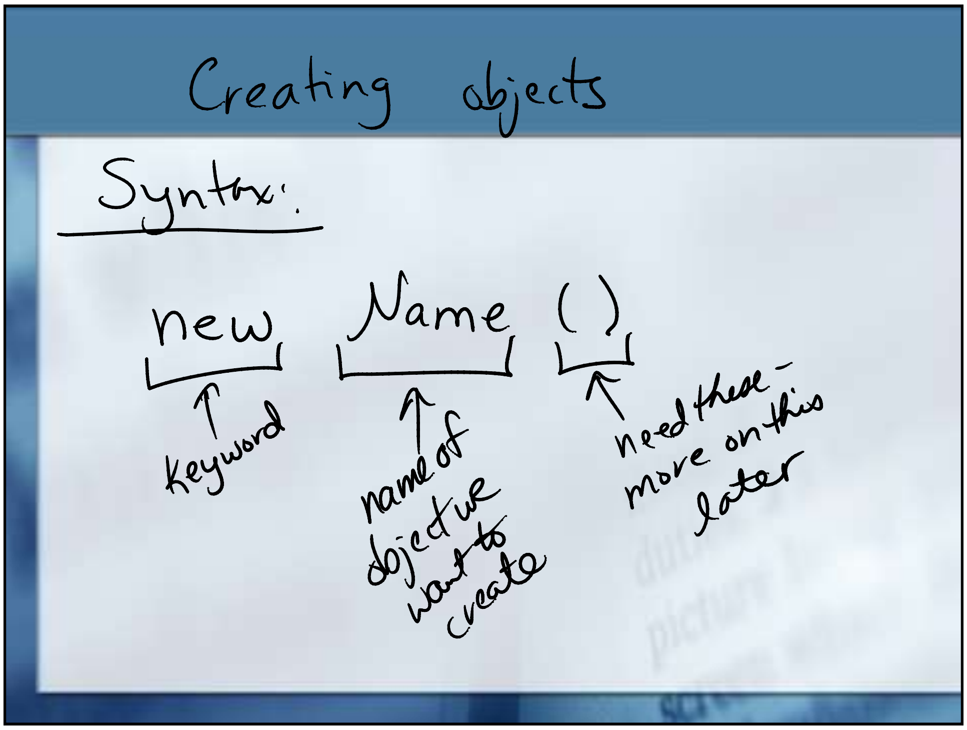
CSE 115/503

January 31 – February 4, 2011

Announcements

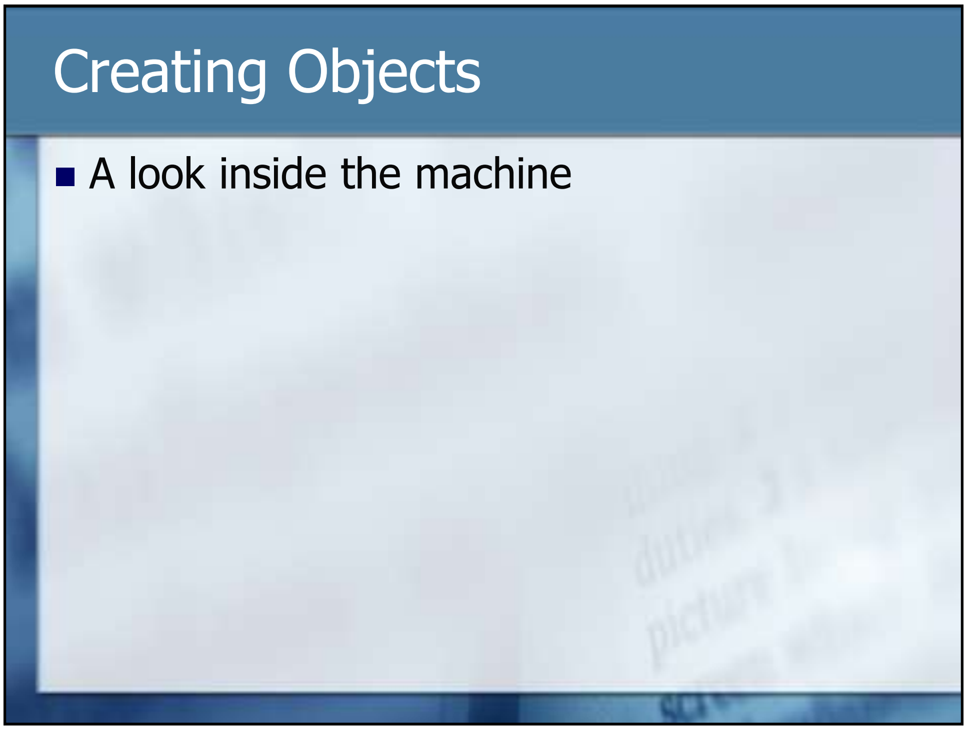
- Pick up (and READ) syllabus if you have not already done so.
- Syllabus Confirmation "test" on UBLearns needs to be completed by 1/31/11.
- Recitation Change forms have been processed and emails sent to those who turned one in to me.
- Lab 1 begins this week in recitation
- Exam 1 – ~~Monday, February 7~~ Wednesday, Feb 9th (first half of lecture – there will be class following the exam)



Creating objects

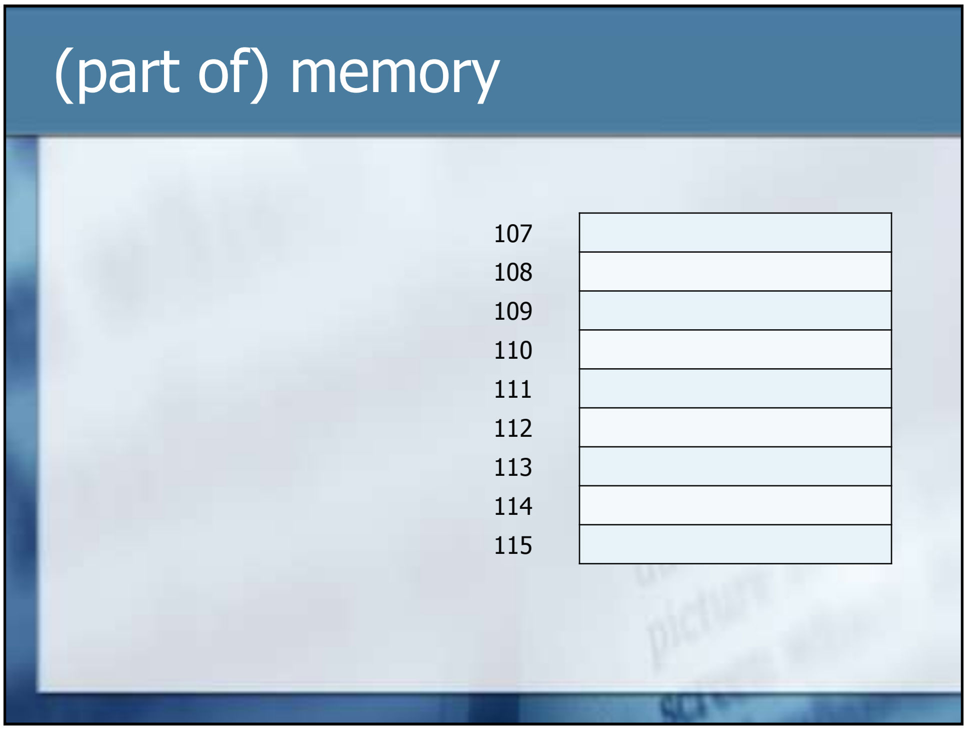
Syntax:

$$\underbrace{\text{new}}_{\uparrow\ \text{keyword}}\quad\underbrace{\text{Name}}_{\uparrow\ \text{name of object we want to create}}\quad\underbrace{(\ )}_{\uparrow\ \text{need these - more on this later}}$$



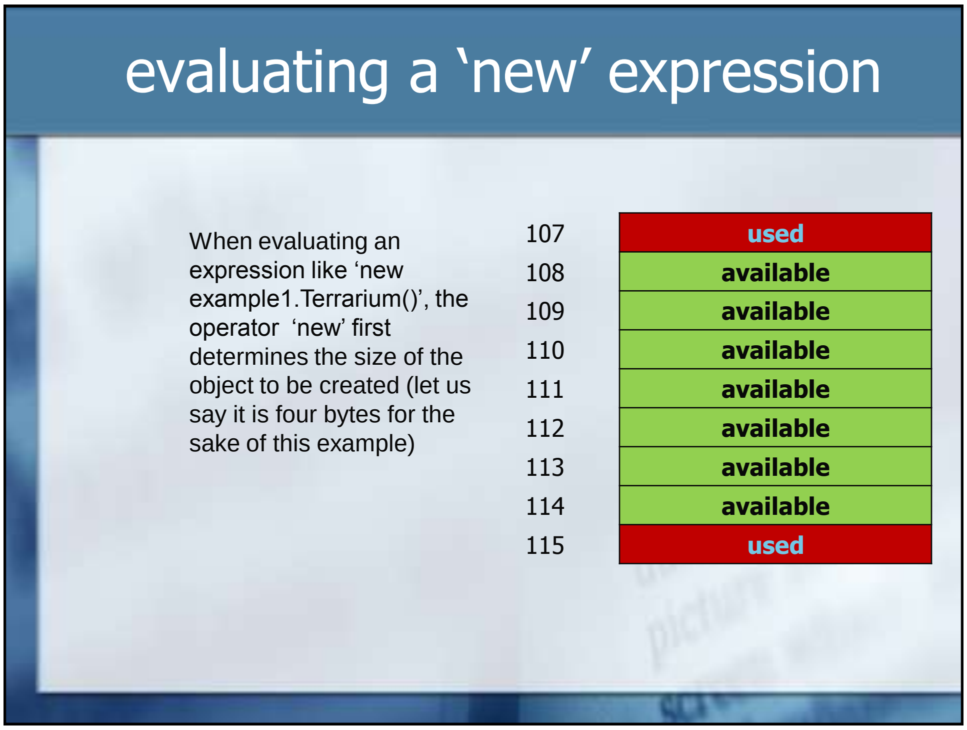
Creating Objects

- A look inside the machine



(part of) memory

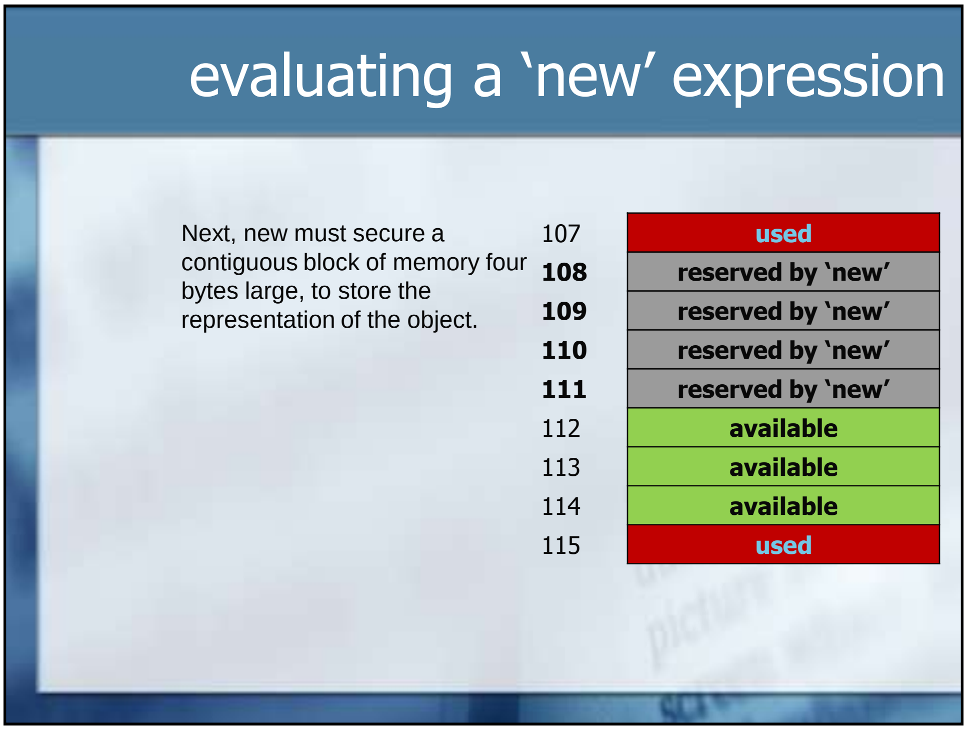
107	
108	
109	
110	
111	
112	
113	
114	
115	



evaluating a 'new' expression

When evaluating an expression like 'new example1.Terrarium()', the operator 'new' first determines the size of the object to be created (let us say it is four bytes for the sake of this example)

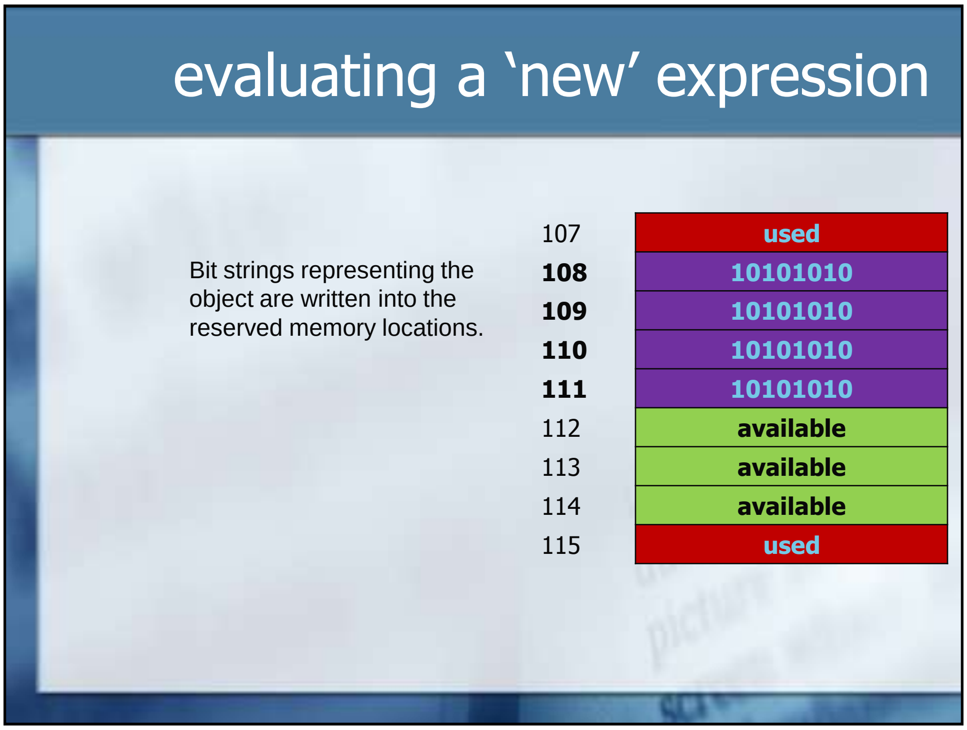
107	used
108	available
109	available
110	available
111	available
112	available
113	available
114	available
115	used



evaluating a ‘new’ expression

Next, new must secure a contiguous block of memory four bytes large, to store the representation of the object.

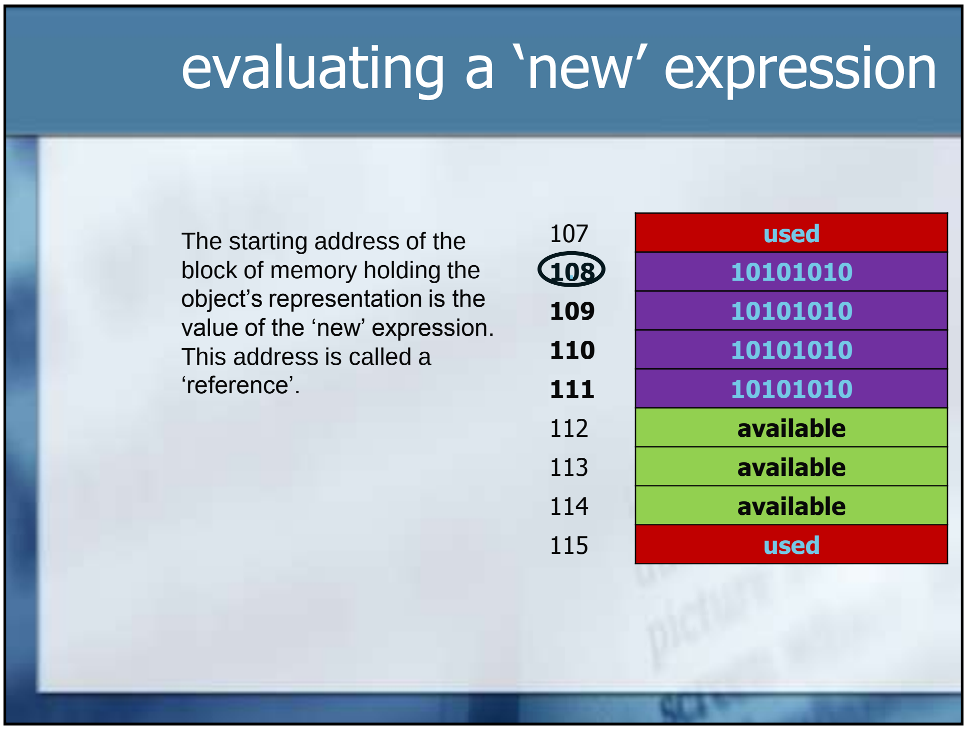
107	used
108	reserved by ‘new’
109	reserved by ‘new’
110	reserved by ‘new’
111	reserved by ‘new’
112	available
113	available
114	available
115	used



evaluating a ‘new’ expression

Bit strings representing the object are written into the reserved memory locations.

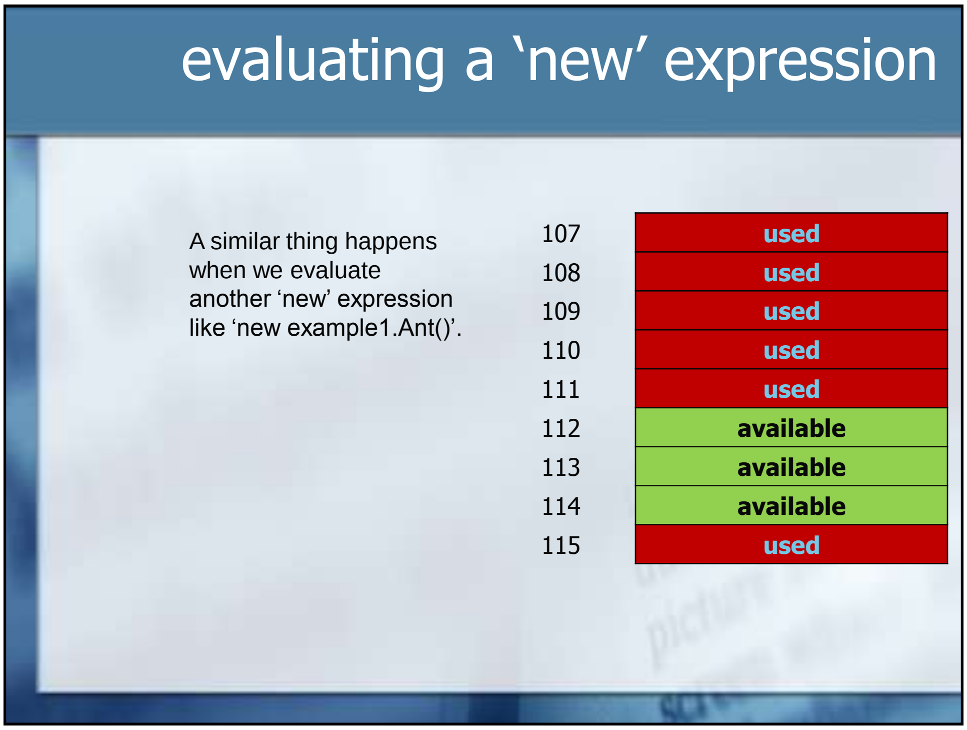
107	used
108	10101010
109	10101010
110	10101010
111	10101010
112	available
113	available
114	available
115	used



evaluating a 'new' expression

The starting address of the block of memory holding the object's representation is the value of the 'new' expression. This address is called a 'reference'.

107	used
108	10101010
109	10101010
110	10101010
111	10101010
112	available
113	available
114	available
115	used



evaluating a 'new' expression

A similar thing happens when we evaluate another 'new' expression like 'new example1.Ant()'.

107	used
108	used
109	used
110	used
111	used
112	available
113	available
114	available
115	used

evaluating a ‘new’ expression

Supposing that an example1.Ant object occupies two bytes of memory, new reserves a contiguous block of two bytes, writes bit strings representing the object to those memory locations, and the starting address of this block of memory is the value of the ‘new’ expression.

107	used
108	used
109	used
110	used
111	used
112	**11110000**
113	**11110000**
114	**available**
115	used

DrJava’s response

When we evaluate these ‘new’ expressions in DrJava, what is the response we get?

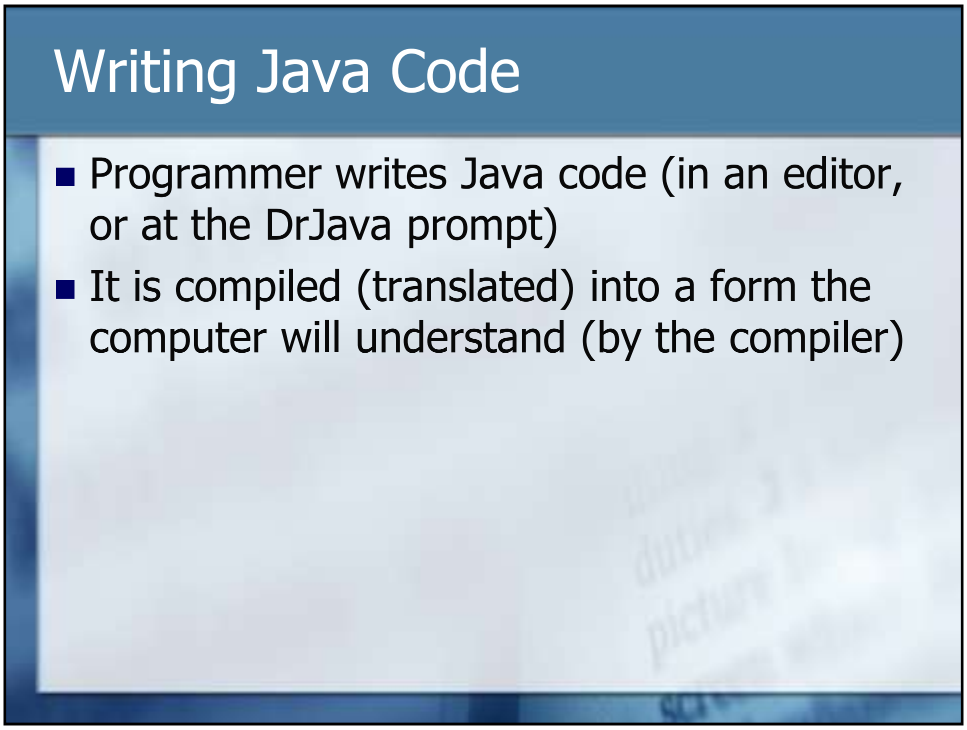
```
> new example1.Terrarium()
example1.Terrarium[frame0,0,0,608x434,layout=java.awt.Bord
erLayout,title=,resizable,normal,defaultCloseOperation=EXIT_
ON_CLOSE,rootPane=javax.swing.JRootPane[,4,30,600x400,la
yout=javax.swing.JRootPane$RootLayout,alignmentX=0.0,align
mentY=0.0,border=,flags=16777673,maximumSize=,minimum
Size=,preferredSize=],rootPaneCheckingEnabled=true]
```

DrJava's response

After DrJava evaluates the expression, it must print the value. The way Java works when a reference is printed is that a textual representation of the object it refers to is produced (as defined by the object itself)

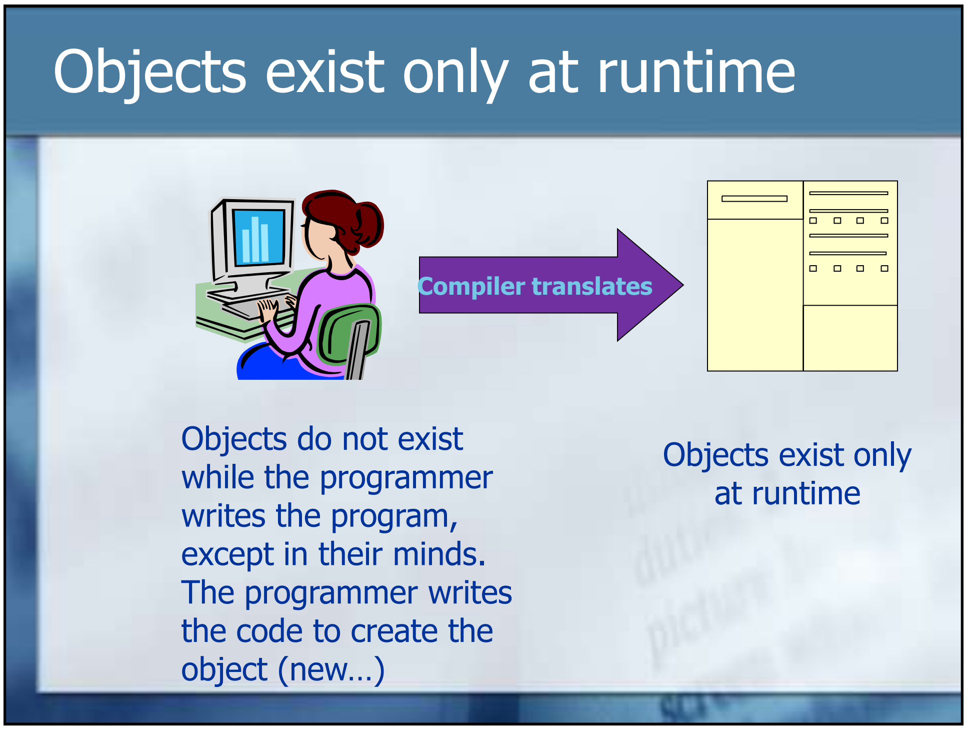
Where do objects come from? (The "birds and bees" lecture)

- We've seen how to create an object.
- But where does the object come from?
- How does DrJava know what an example1.Terrarium() object is?

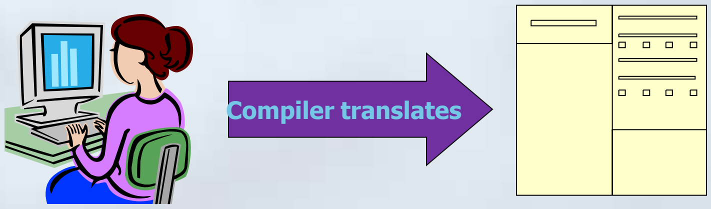


Writing Java Code

- Programmer writes Java code (in an editor, or at the DrJava prompt)
- It is compiled (translated) into a form the computer will understand (by the compiler)

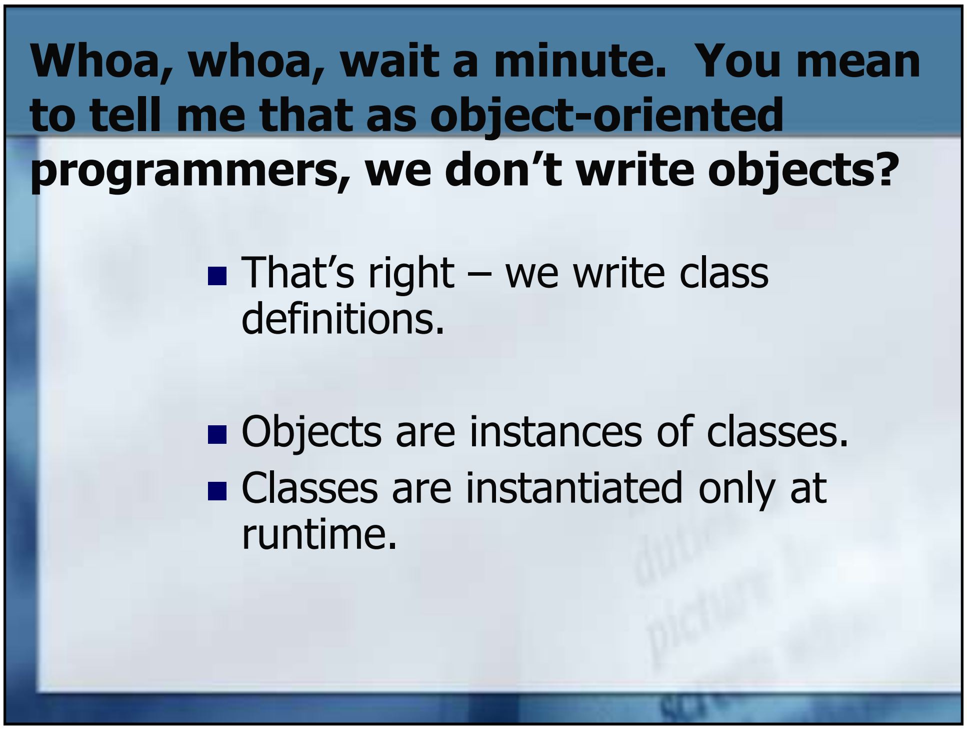


Objects exist only at runtime



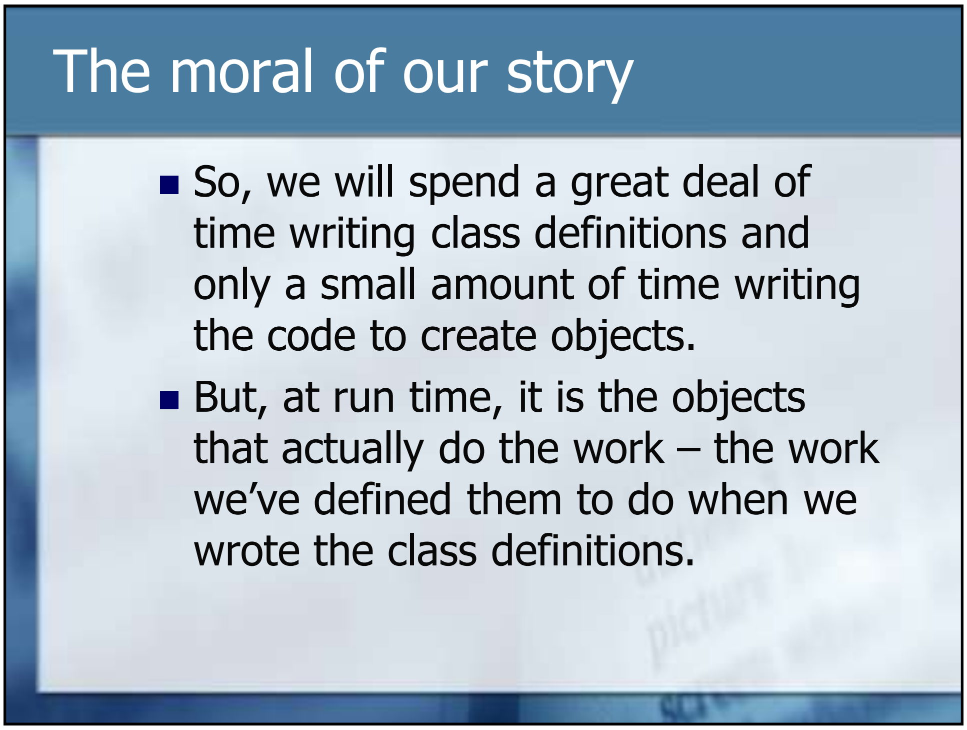
Objects do not exist while the programmer writes the program, except in their minds. The programmer writes the code to create the object (new…)

Objects exist only at runtime



Whoa, whoa, wait a minute. You mean to tell me that as object-oriented programmers, we don't write objects?

- That's right – we write class definitions.
- Objects are instances of classes.
- Classes are instantiated only at runtime.



The moral of our story

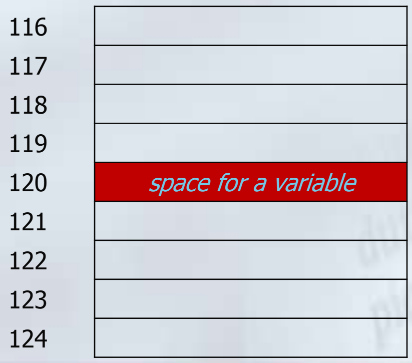
- So, we will spend a great deal of time writing class definitions and only a small amount of time writing the code to create objects.
- But, at run time, it is the objects that actually do the work – the work we've defined them to do when we wrote the class definitions.

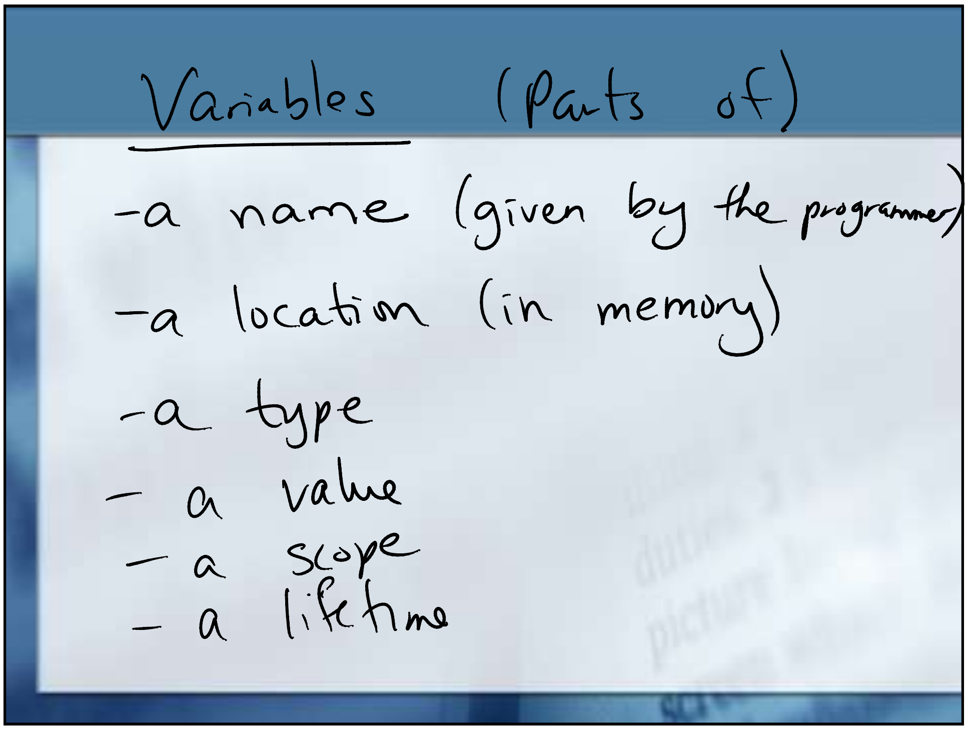
```
> new example1.Terrarium()

> new example1.Caterpillar()
>new example1.Terrarium().add(new example1.
                                   Caterpillar())
```

A variable is:
(at its most basic)

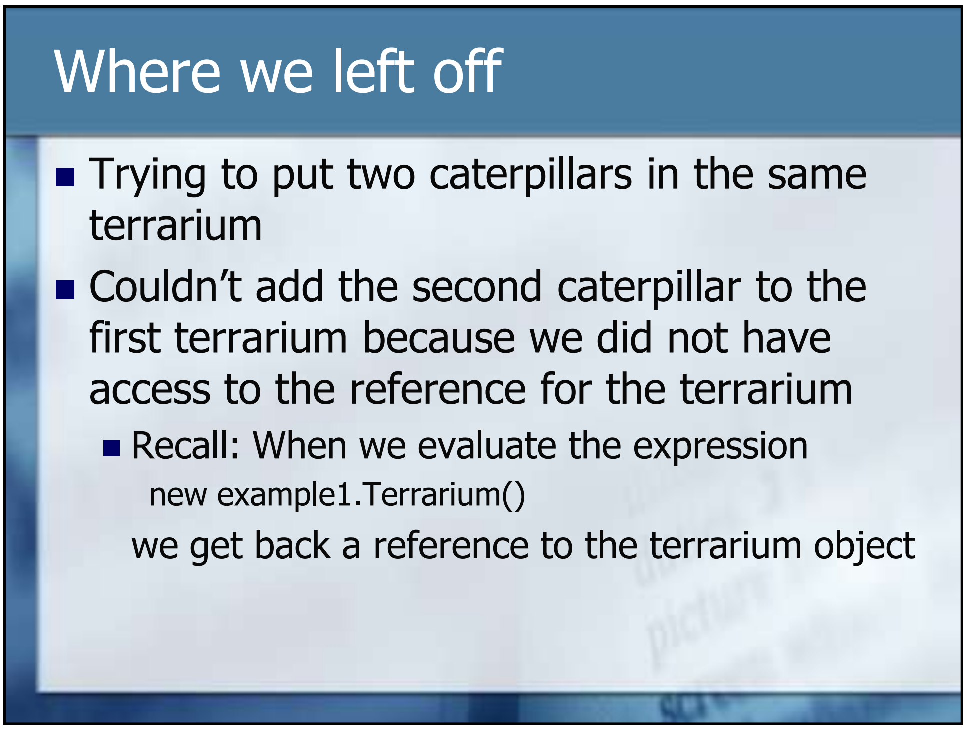
- a storage location in memory
- for example, location 120:





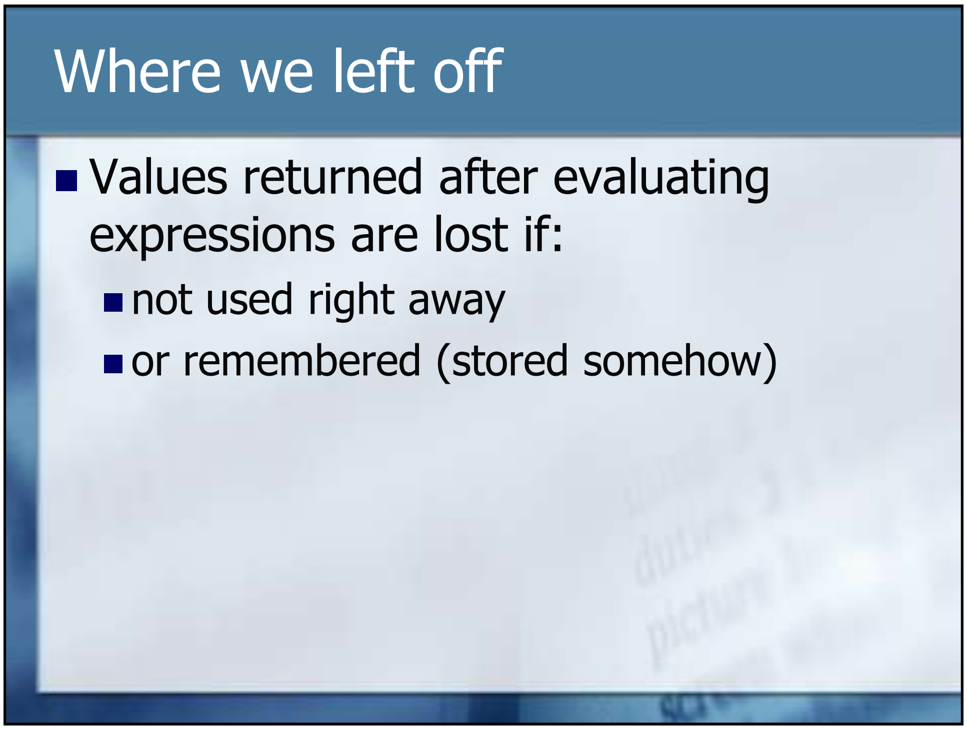
Variables (Parts of)

- a name (given by the programmer)
- a location (in memory)
- a type
- a value
- a scope
- a lifetime



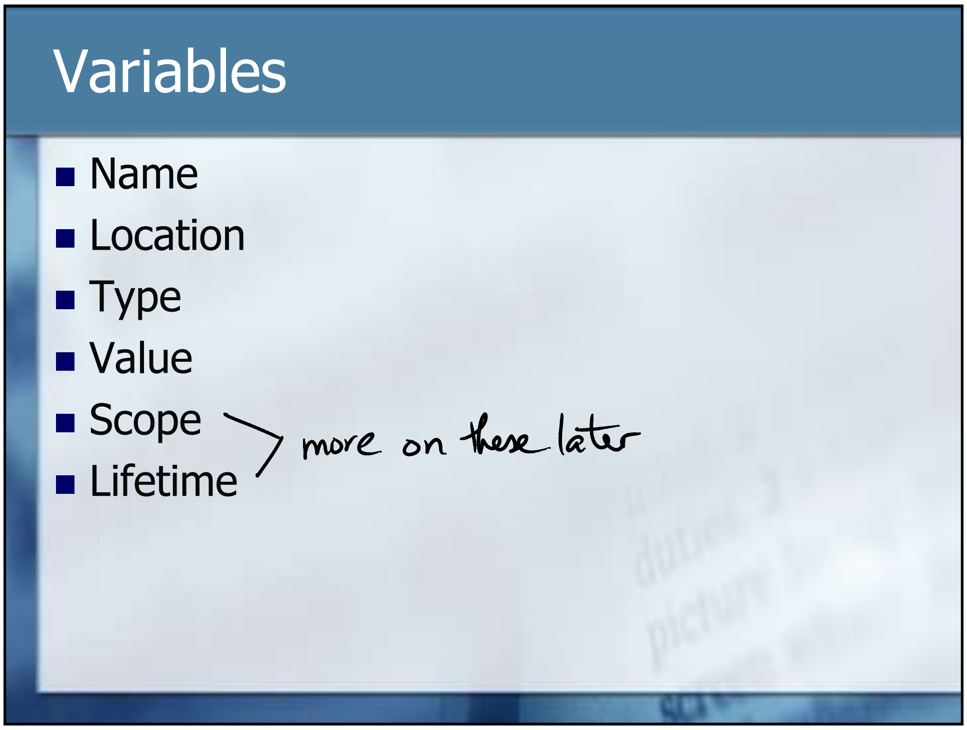
Where we left off

- Trying to put two caterpillars in the same terrarium
- Couldn’t add the second caterpillar to the first terrarium because we did not have access to the reference for the terrarium
  - Recall: When we evaluate the expression
    new example1.Terrarium()
    we get back a reference to the terrarium object



Where we left off

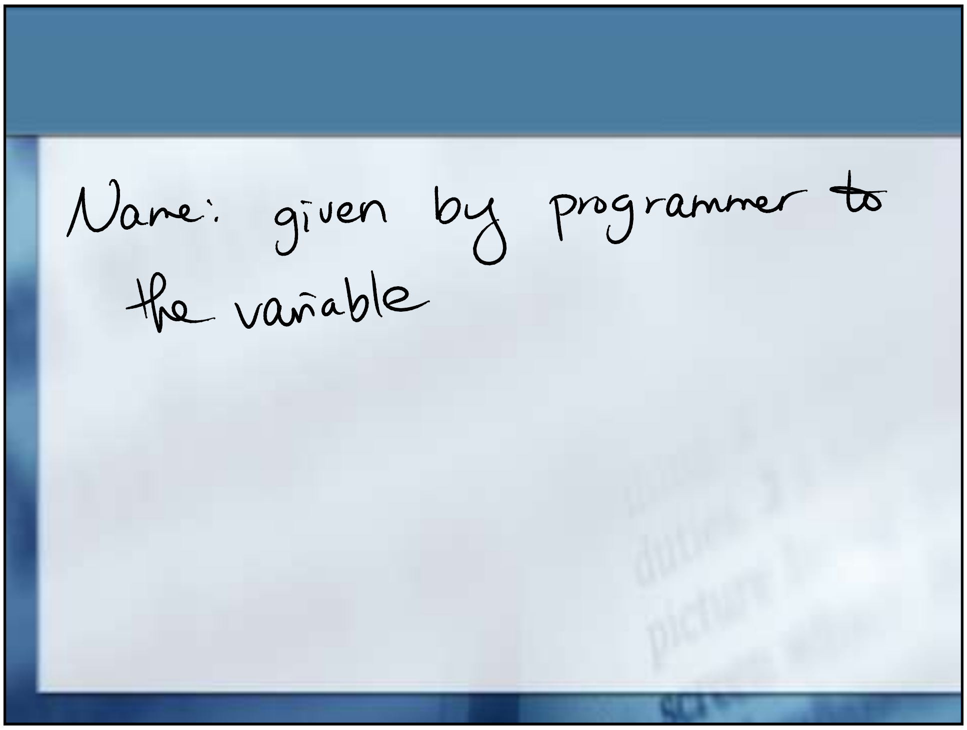
- Values returned after evaluating expressions are lost if:
  - not used right away
  - or remembered (stored somehow)



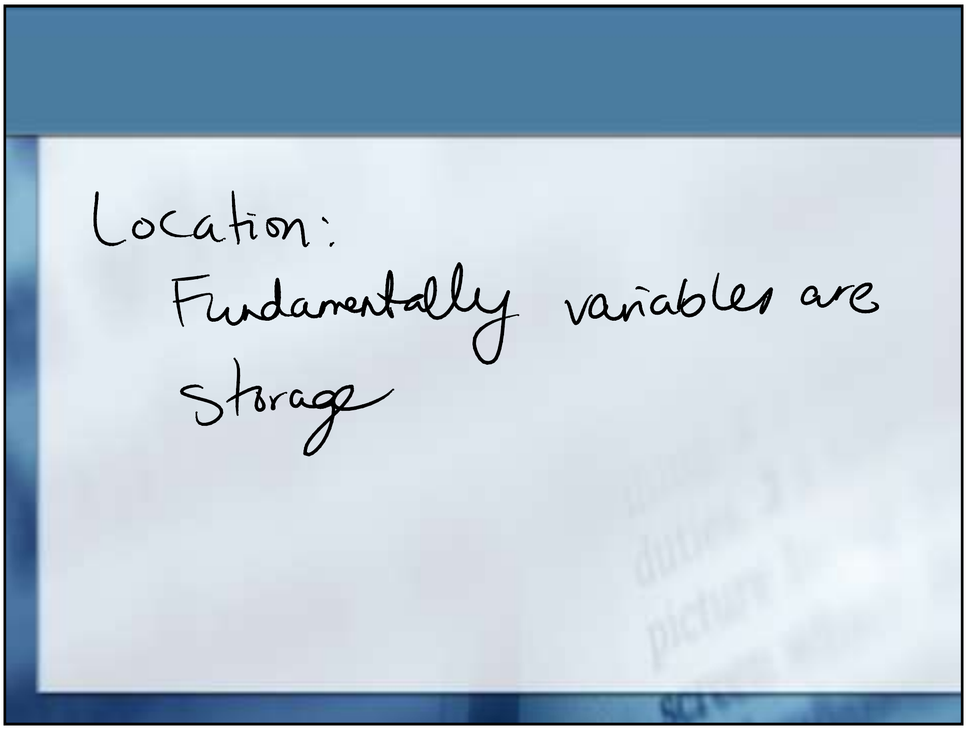
Variables

- Name
- Location
- Type
- Value
- Scope
- Lifetime

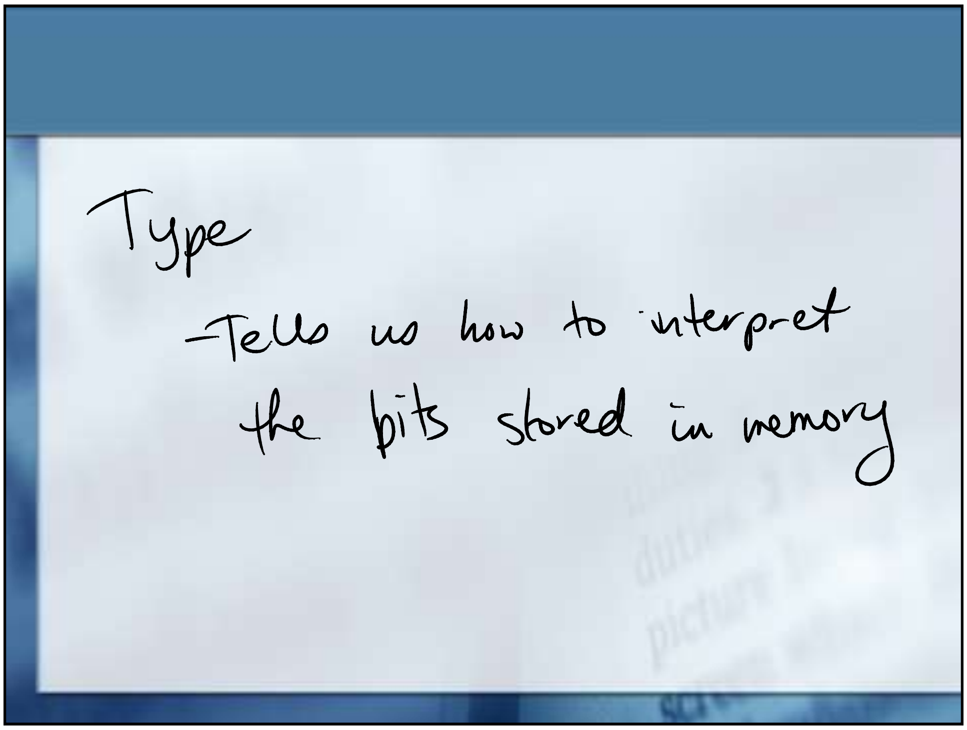
more on these later



Name: given by programmer to the variable

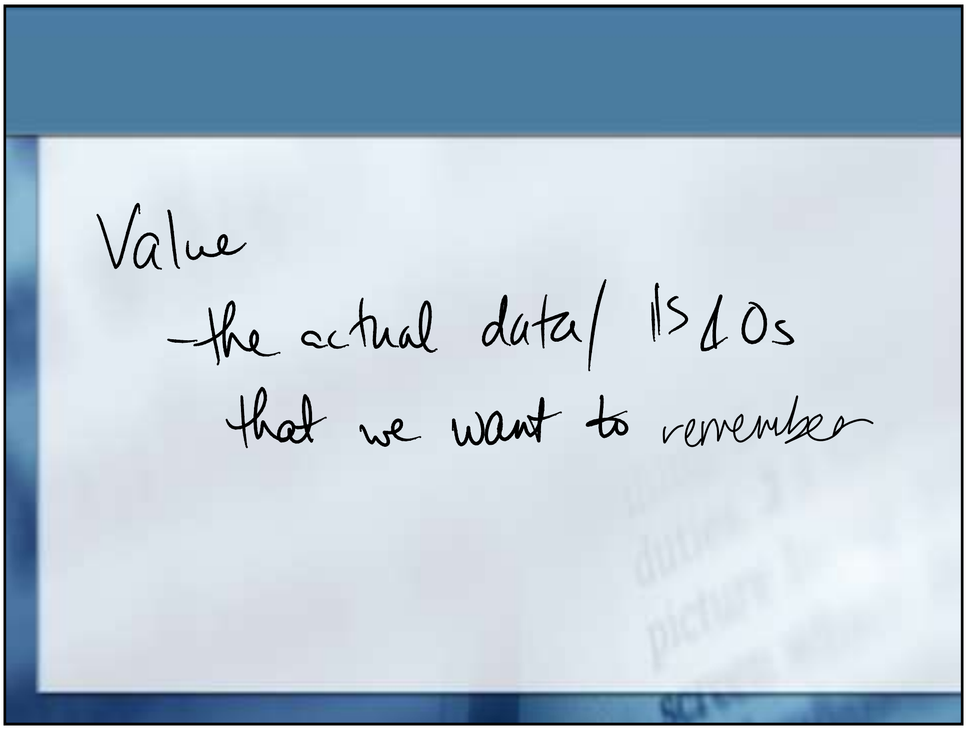


Location:
Fundamentally variables are storage



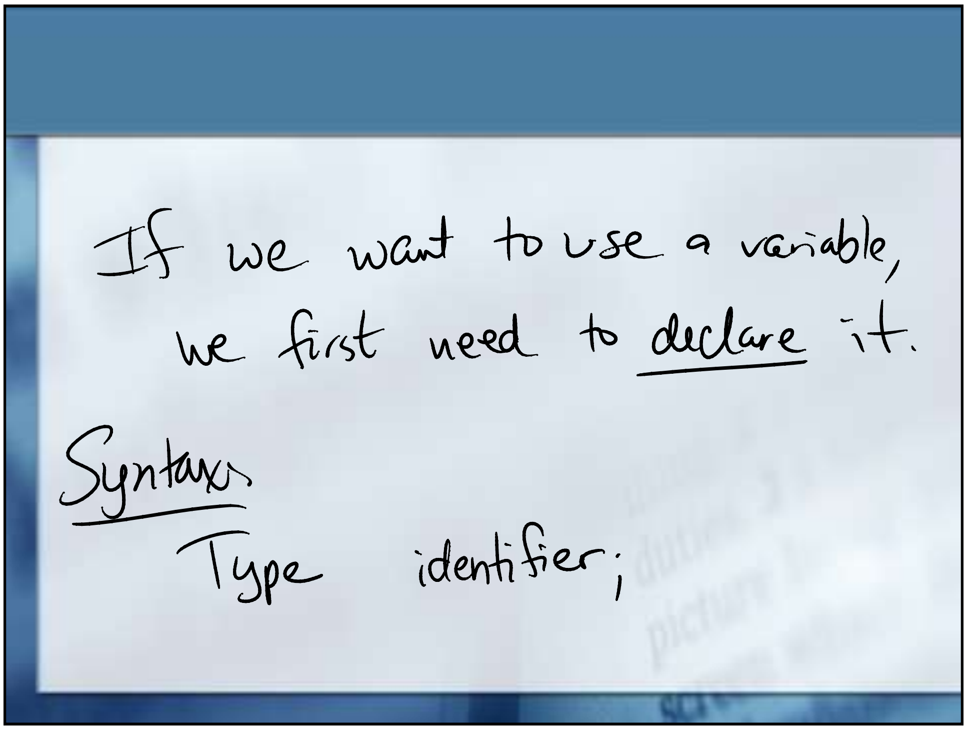
Type

- Tells us how to interpret the bits stored in memory



Value

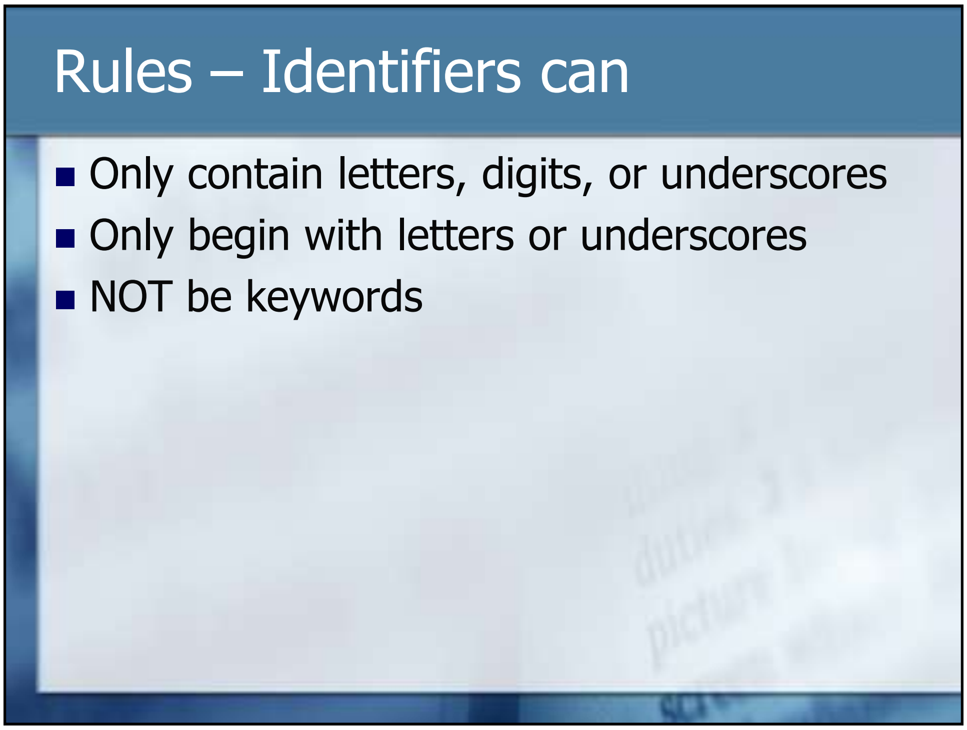
- the actual data/ 1s & 0s that we want to remember



If we want to use a variable, we first need to declare it.

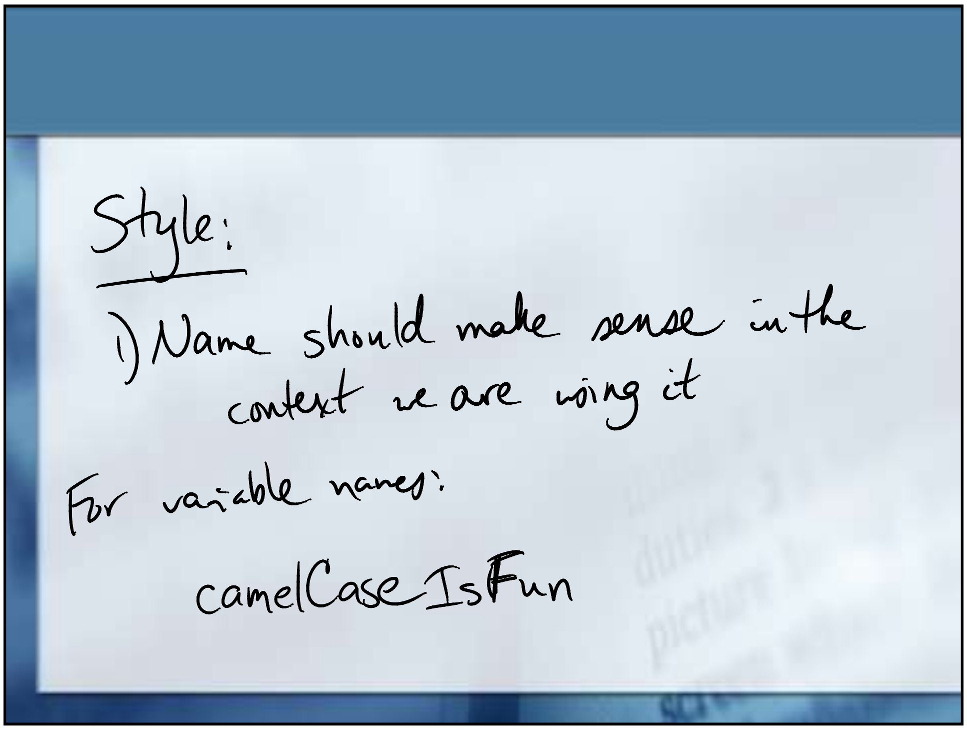
Syntax:

Type identifier;



Rules – Identifiers can

- Only contain letters, digits, or underscores
- Only begin with letters or underscores
- NOT be keywords

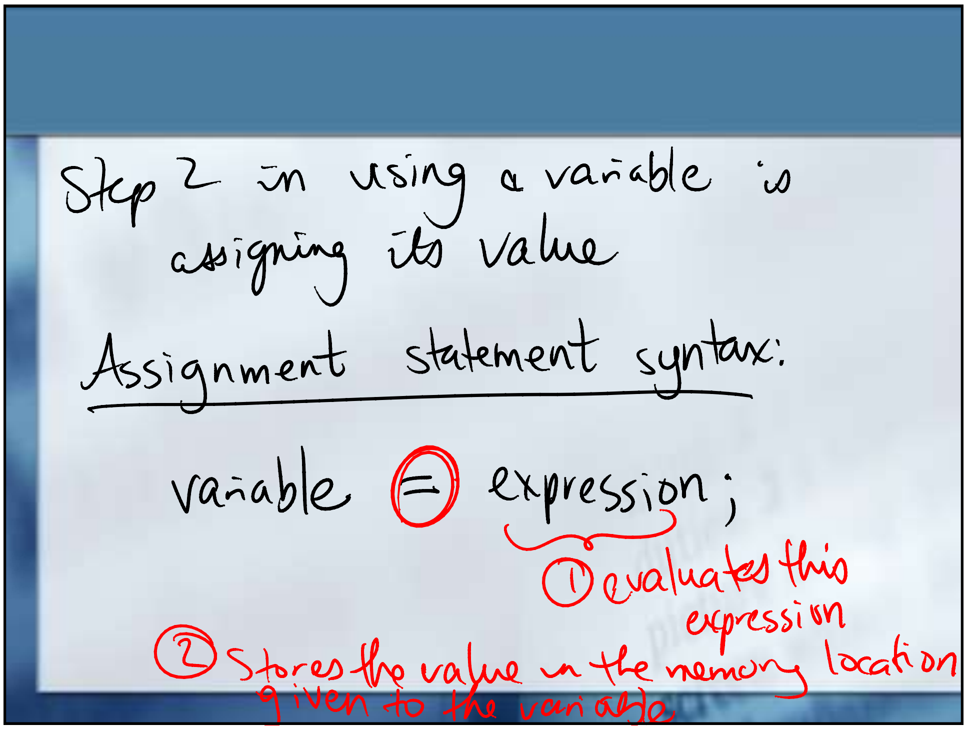


Style:

1) Name should make sense in the context we are using it

For variable names:

camelCaseIsFun



Step 2 in using a variable is assigning its value

Assignment statement syntax:

$\text{variable} \boxed{=} \underbrace{\text{expression}}$;

① evaluates this expression

② Stores the value in the memory location given to the variable