Variables

CSE 142, Summer 2002

Computer Programming 1

http://www.cs.washington.edu/education/courses/142/02su/

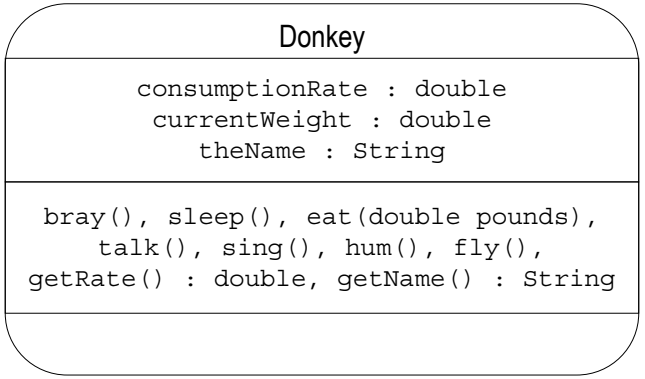
Readings and References

- Reading
  - Chapter 5, *An Introduction to Programming and Object Oriented Design using Java*, by Niño and Hosch
  - Chapter 6, *Introduction to Programming in Java*, Dugan
- Other References

Variables

- A *variable* is
  - a portion of memory reserved to hold a single value
- Our program uses little chunks of memory to store the values that it is working with
  - The program refers to each chunk by name, the name of the variable
  - When we declare a variable, we give it a name and a type
  - Java helps us make sure that we use the variable the way we intended by enforcing "type checking"

class diagram for Donkey



Variable declarations

```
public class Donkey {
   /**
    * Create a new Donkey.
    * @param name name of this Donkey
    * @param rate the rate at which this Donkey eats, specified in pounds/fortnight.
    * @param weight the initial weight of this Donkey
    */
    public Donkey(String name,double rate,double weight) {
        theName = name;
        consumptionRate = rate;
        […snip…] }
   /**
    * Eat some goodies.  There is some weight gain after eating.
    * @param pounds the number of pounds of food provided.
    */
    public void eat(double pounds){
        System.out.println(theName+" : Food!");
        double coverage = pounds/consumptionRate;
        […snip…] }
    //-------------------------------------------------------------------------
    /**
     * the consumption rate specified for this Donkey.  Given in
     * pounds per fortnight.  A fortnight is two weeks or 14 days.
     */
    double consumptionRate;
    […snip…] }
```

constructor parameter

method parameter

local variable

instance variable

Parameter variables

- Parameters are the means by which the caller provides information to the constructor or method it is calling
  - the caller knows that it wants some action to be performed
  - it knows that some other object knows how to do this action
  - the caller knows a little something about doing the action

    how much to eat, how long to sing, what color to make the tree, ...
  - The value provided by the caller is passed along in the form of a parameter

```
public void eat(double pounds) {
[…snip…]
}
```

called with

```
Donkey pet = new Donkey("Noble Steed");
pet.eat(1);
```

Donkey.java

Lifetime of a parameter variable

- When a constructor or method is called, parameter variables are created for its use during this pass through the code
  - The variables are initialized to the values provided by the caller
- The constructor or method can refer to the parameters by the names it used in the declaration, regardless of what the caller is using for names
- These *automatic variables* (copies of the provided value) are thrown away when the constructor or method returns control to the caller

Local variables

- Local variables are the way that a constructor or method creates little scratchpad notes to use as it does whatever its task is
- Local variables are declared within the body of the constructor or method
  - "within the body of the method" means "between the curly braces"

```
public void sleep {
   double nightLength = 8.5;
   this area is the body of the method
}
```

- Local variables can be used exactly as any other variable is used, but they have a limited lifetime

Lifetime of a local variable

- When a constructor or method executes a statement that declares a local variable, a little chunk of memory is made available that it can use for the variable
  - The variable *is not initialized* until the method does the initialization itself
- The constructor or method can refer to the variable throughout the remainder of the body
- These *local variables* are thrown away when body of the method is finished executing
  - Next time the method is called, the variables will be allocated fresh
  - There is no carryover of value from one execution to the next!

Instance variables

- Instance variables are the way an object keeps track of its state
  - There is one set of instance variables for *each* instance of the class
  - Each object of the class has its own set of instance variables
- Instance variables are declared outside the body of any constructor or method (but within the definition of the class)
  - "within the definition of the class" means "between the outside pair of curly braces"

```
public class Donkey {
    constructors and methods
    double consumptionRate;
}
```

- Instance variables retain their values as long as the object exists

Lifetime of an Instance Variable

- When a new object is created, the Java runtime libraries allocate a chunk of memory for the object
  - the chunk of memory has space in it for each instance variable
- Once the memory is allocated the appropriate constructor is called to initialize the instance variables
  - There is some initialization done by the system, but it is poor form to rely on that since you can easily overlook a variable that really should be initialized to some special value
  - The programmer is responsible for making sure that the initialization is done correctly and completely, usually in the constructor
- Instance variables exist from the time an object is created until the time it is destroyed

Compare Local Variable and Instance Variable

- Local Variable
  - Defined inside a method
  - Exists only while the method is being executed
  - Can be accessed only from the method
  - Is only meaningful during execution of the method
  - Contains some intermediate value needed only during execution of the method; its value is not part of the object's state
- Instance Variable
  - Defined outside any method
  - Exists as long as the object exists
  - Can be accessed from any method in the class
  - Has a meaningful value at any time during the life of the object, whether the object is actively doing something or not
  - Represents a property of the object; its value is part of the object's state

table from Niño & Hosch

Type checking

- Java helps as much as it can to make sure you use variables the way you said you were going to when you declared them
- If you said that **`currentWeight`** is an **`int`**, then Java will make sure you don't unintentionally put a **`double`** value in it and lose the fractional part

```
int currentWeight;
currentWeight = 2;
currentWeight = currentWeight+0.5;
```

- What should the value of currentWeight be at this point?
  - you said it was an integer, why are you adding 0.5 to it?
  - the Java compiler decides that this must be a mistake

    error: "possible loss of precision"

Type casting

- If the compiler thinks you are making a mistake, it will tell you so

```
currentWeight = currentWeight + (currentWeight*percentGain);
```

  "possible loss of precision. found double, required int"
- If you are really sure that you know it's okay, you can tell the compiler not to worry about it
  - "I know there's a possible loss of precision, don't fret about it."
- The mechanism for doing this is called casting
- The type you want the value converted to is placed in parentheses in front of the value or expression to convert

```
currentWeight = currentWeight+(int)(currentWeight*percentGain);
```

- The compiler will convert the value to **`int`** for you
  - beware: loss of precision may be a real problem!

SlideE.java

void

- Ordinarily we specify the type of object returned by a method

```
public String getName() {
    return theName;
}
```

- Sometimes we need to specify "nothing is here"
- the keyword **`void`** is used when we want to say that nothing is returned from a method

```
public void bray() {
    System.out.println(theName+" : HeeHaw!");
}
```