Classes and Arrays

ISE 108: Introduction to Programming
Stony Brook University

The Story So Far...

- So far, we've learned to write small Processing programs
  - We can use conditionals, loops, and functions
  - We also need to use (a lot of) variables
- As our programs get larger, we need more and more variables
- How do we keep track of everything?

Information Overload

- We have too much data to keep track of
  - e.g., ten different shapes need ten different x coordinates
- We need some way to:
  - Bundle related data together
  - Avoid having to remember a gazillion* variable names

* gazillion: n. Technical term meaning "far too many"

The Solution

- Classes
  - let us treat related pieces of data as a single entity
  - let us create more sophisticated program "variables"
- Arrays
  - let us store multiple values of the same type as a single large block

Classes

Objects

- An object is a “bundle” of data (variables) and code (functions)
  - An object represents some “thing” in our program
- Objects are essentially variables that can:
  - store multiple pieces of information
  - do things with that information

Objects Are Self-Managing

- An object contains all of the functions (methods, representing “behavior”) that need to operate on its data (“attributes”)
  - This is called ***encapsulation***, meaning that an object is responsible for managing its own data
- An object may hide its contents from other parts of the program
  - This protects our data from random changes

Basic Terminology

- **Class**: a description of the structure of an object
  - Think of a class as the blueprint for a type of object
  - A class defines a new type of variable for your program
- **Instance**: a specific example from a class
  - e.g. “a car” (class) vs. “*that* car over there” (instance)

Classes vs. Instances

- Every instance of a given class has the same kind of information, but different actual values
  - e.g. every student has an ID number, but each student's ID number is unique

The Anatomy of a Class

- A class contains two types of elements:
  - Instance data — variables representing an object's attributes
  - Instance methods — methods (code) that define an object's behaviors
- Each class is defined using the keyword "class"
- Curly braces enclose the data and method declarations that make up the class

Class Definition Example

```
class Example // defines a new "Example" data type
{
  instance variable declarations go here...

  instance method definitions go here...
}
```

- After you define a class, you can use it as a new variable type in your programs, just like `int`, `double`, etc.
- In this case, we can now create new "Example" variables

Creating An Object For Use

- To create (instantiate) an object, use the keyword `new` :

  ```
  MyClass foo = new MyClass( );
  ```

- "new" invokes an object's *constructor*
- `foo` is a *reference variable*
  - It holds the memory address of a `MyClass` object
  - Classes are stored differently in RAM than "primitive" (built-in) types like `int` and `double`

Constructors

- A constructor is a special method (function) that is called to instantiate ("construct") a new object
- Constructors set the initial values of the object's data
  - These values may be defined in the constructor or passed in as arguments when the object is instantiated

More on Constructors

- A constructor has the same name as the class
- A constructor has NO return type
- If (and *only* if) no constructors are defined, Processing provides a class with a *default constructor*
  - This constructor takes no parameters and does not assign any specific values to the object's variables

Constructor Example

```
class Car
{
    // Variables
    int currentSpeed;
    int maxSpeed;
    int mileage;

    // Constructor
    Car (int max)
    {
        maxSpeed = max;
        currentSpeed = 0;
        mileage = 0;
    }
}
```

- When we create a new `Car` object, we need to supply a value for the constructor:

```
Car c = new Car(140);
```

Elements of a Class

- A class holds *instance variables* and *instance methods*
  - "instance": each object instance has its own copy
- Instance variables are declared like normal variables
- Instance methods are functions that belong to the object
  - They are used to operate on instance variables

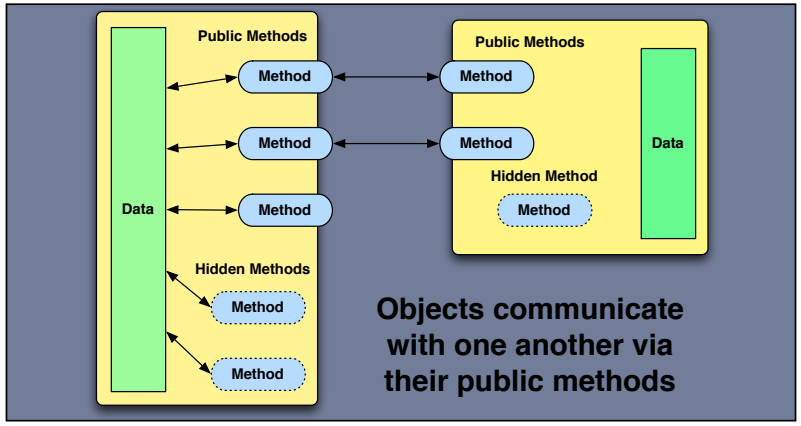
Access Modifiers

- A class may restrict access to its methods and variables
- The keyword `public` means that something is visible and accessible inside and outside the class
- The keyword `private` means that a variable or method can only be seen or used by methods inside the class
- If we omit the access modifier, Processing lets us use class elements from the rest of our program
  - Technically, this is “package” (not public) access

General Access Guidelines

- Programming tip: Always make the variables in your classes `private` whenever possible
  - This prevents unexpected changes from outside
- Methods should *only* be `public` if other parts of the program may need to call them; otherwise, they should be `private` and kept as internal helper methods
  - Private methods may only be used by other methods in the same class

Object Interactions



Arrays

Arrays

- Programs often use large quantities of similar data
- Assigning a unique variable (and name) to each piece of data is tedious
  - Ex. var1, var2, var3, ...
- An array is a collection of many variables of the same type, all under one name
  - Arrays can be of any type

Array Storage

```
int[] a = new int[6];
```

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	

Declaring An Array

- To *declare* an array, follow the type with square brackets:

```
double[] foo;
```

- To *create* an array, use `new` and supply a size:

```
foo = new double[5];
```

  - If this looks like what you do for a class, you're right; arrays are actually (specialized) objects
  - The array size must be a positive integer

Array Size

- The size of an array is fixed upon creation
- The `length` field returns an array's size

```
int[] a = new int[5];
int size = a.length; // size is 5
```

- Note that `length` has no parentheses!
  - It's a field (variable), not a function/method

Examples

- Definitions:

```
char[] c;
int [] value = new int[10];
```

- End Result:
  - Array object variable `c` is un-initialized
  - Array object variable `v` references a new ten-element list of integers
    - Each of the integers is initialized to 0 by default

Array Examples

```
char[] alphabet = new char[26];
```

```
int numPlayers = 5;
int[] scores = new int[numPlayers];
```

```
String[] phrases = new String[15];
```

Creating Arrays

- Arrays can be initialized at declaration:

```
int [] bar = {5, 4, 3, 2, 1};
```

- The number of items in the initialization list sets the size of the array
  - In this example, `bar` has five elements
- The array is filled with the values in the initialization list

Array Elements

- Individual elements of an array are accessed by using the array name, followed by an (integer) index value, enclosed in brackets
  - Ex. `myArray[1]`
- Indices are numbered starting with 0
  - e.g., `myArray[1]` refers to the second element in `myArray`

Arrays and Loops

- Loops (especially `for` loops) are the perfect way to manipulate arrays:

```
int [] a = new int[5];
for (int i = 0; i < a.length; i = i + 1)
{
    a[i] = i * 2;
}
```