TCSS 142 – Introduction to Programming

Autumn 2014
Day 18

Day 18 Overview

- Abstract data types
- Classes

Data Types

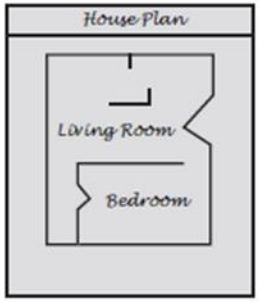
- All data types have the following components in common:
  - The set of values valid for that type
  - The set of operations that can be performed on that type

- Each programming languages defines its own built-in data types
  - the set is finite
  - what do we do when we want to represent entities such as a student, a planet, a house, etc?

Class

- A class – programming construct that allows a programmer to create his/her own data type
  - a template for creating variables (objects) of that new type
  - we call such data types ADTs (abstract data types)

Blueprint that describes a house



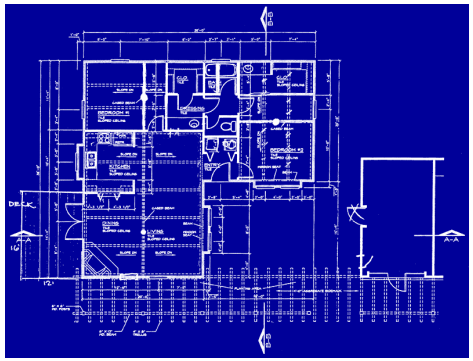
Instances of the house described by the blueprint

Blueprint analogy

iPod blueprint

state:
- current song
- volume
- battery life

behavior:
- power on/off
- change station/song
- change volume
- choose random song



creates

iPod #1

state:
- song = "1,000,000 Miles"
- volume = 17
- battery life = 2.5 hrs

behavior:
- power on/off
- change station/song
- change volume
- choose random song

iPod #2

state:
- song = "Letting You"
- volume = 9
- battery life = 3.41 hrs

behavior:
- power on/off
- change station/song
- change volume
- choose random song

iPod #3

state:
- song = "Discipline"
- volume = 24
- battery life = 1.8 hrs

behavior:
- power on/off
- change station/song
- change volume
- choose random song

ADT

- An ADT specifies the set of possible of values and the operations it will support
  - but it hides implementation details, e.g. string library description
  - when you build your own ADT, you build it out of the existing components

ADT Example

TYPE

TimeType

DOMAIN

Each TimeType value is a time in hours, minutes, and seconds.

OPERATIONS

Set the time

Print the time

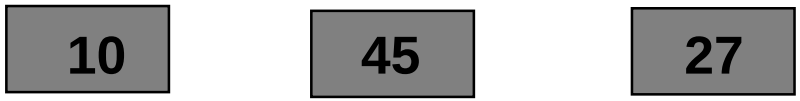
Increment by one second

Compare 2 times for equality

Determine if one time is "less than" another

Possible Implementations

3 int variables

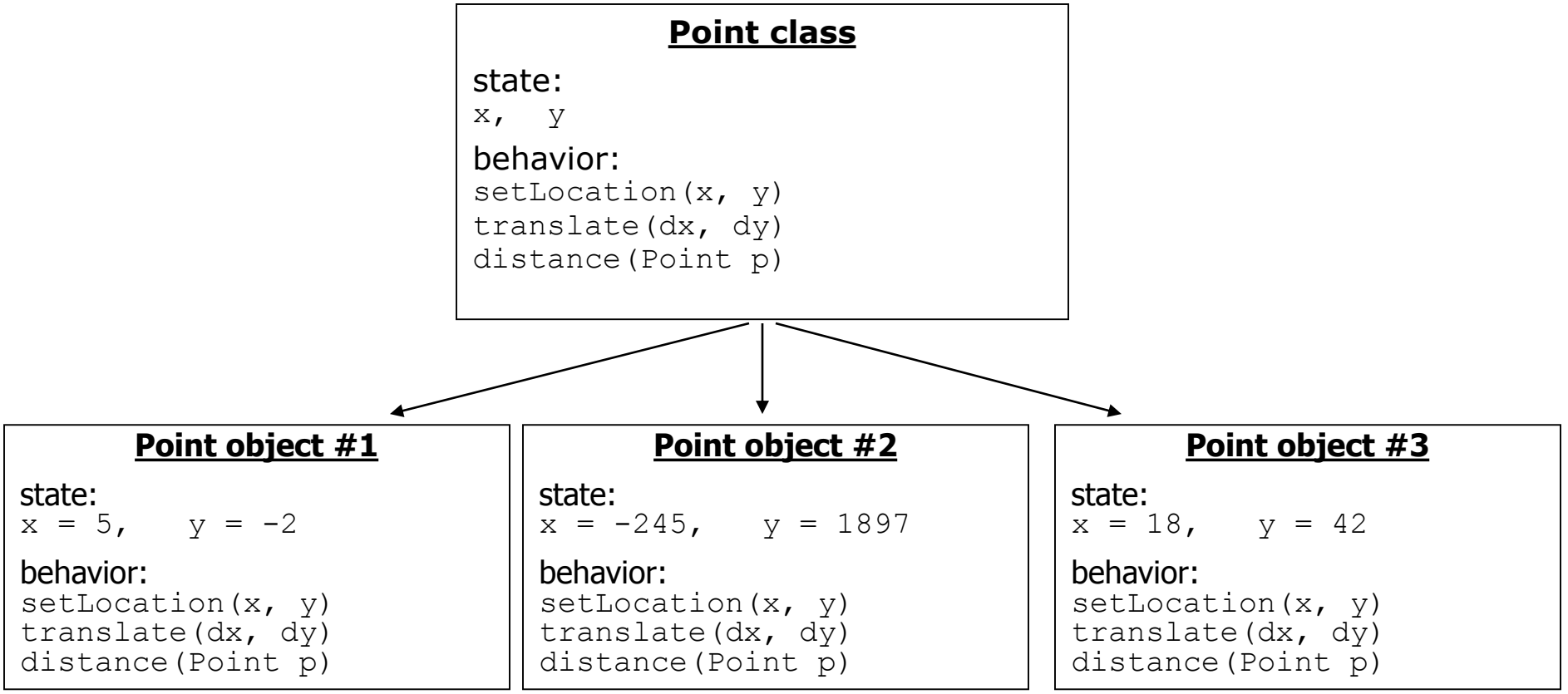


3 strings

3-element int list

- actual choice of representation depends on time, space, and algorithms needed to implement operations

`Point` class as blueprint



- The class (blueprint) will describe how to create objects.
- Each object will contain its own data and methods (functions specific to the class).

Point ADT Example

- Let's create a class that represents a point on a 2D plane
  - Put it in a file called `point.py`
  - Each class should provide a special method called a constructor
    - One special parameter in all methods: `self`
    - Attributes are defined in a constructor
    - We will be making all attributes private `_ _`
    - A constructor is never called by a programmer
    - A constructor has to be named `_ _ init _ _`
  - Let's add function main below our class and create a couple of point objects

Point ADT Example

- Extend the `Point` class with the following:
  - Add `x` and y getters and test
  - Add `setLocation` method and test
  - Add `translate` method and test
  - Add _ _ str _ _ method to make printing easier
  - Add a `flip` method that takes no arguments and swaps `x` and `y` coordinates

Client Code

- Code that uses an ADT is called a client.
- Client code uses public methods to handle objects.
- Client code typically is written in a separate file.
  - Move all the tests into client code called `pointDriver.py`
  - Put both files in the same directory

Exercise

Create a class called `Name` that represents a person's name. It will have the fields that represent the person's first name, last name, and middle initial. (Your class should contain only fields for now.)

Add the following methods to the `Name` class:

- Add a constructor to the Name class that accepts a first name, middle initial, and last name as parameters and initializes the Name object's state with those values.
- Add a method `getReverseOrder()` that returns the person's name in reverse order, with the last name preceding the first name and middle initial. For example, if the first name is "John", the middle initial is "Q", and the last name is "Public", returns "Public, John Q.".

Test your methods in client code.

Exercise Contd

Add `__str__` method for the `Name` class that returns a string such as "John Q. Public".

Add appropriate accessor/getter methods to the class (one for each field).

Add setter methods called `setFirstName`, `setMiddleInitial`, and `setLastName` to your `Name` class.

Passing Objects as Arguments

- Methods and functions often need to accept objects as arguments or return objects

- When you pass an object as an argument or return it, you are actually passing a reference to the object
  - The receiving method or function has access to the actual object
  - Using any mutator method will modify the object

- Let's add `distance` method to our code (receives another Point)

Mutable vs Immutable

- Some classes are designed in Python as "blueprints" for immutable objects
  - value cannot be changed after the object is constructed
  - example: strings

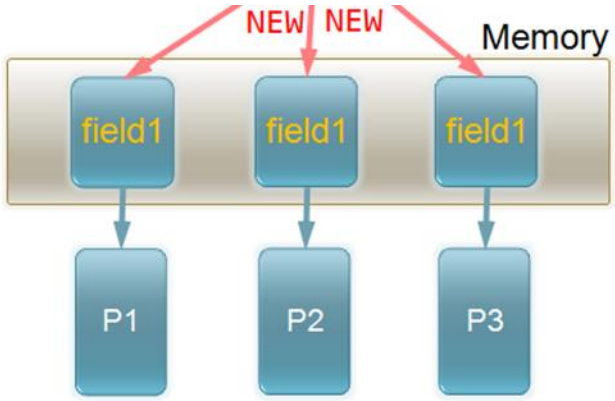
- Some classes are designed in Python as "blueprints" for mutable objects
  - value can be changed after the object is constructed
  - example: list
  - example: Point class
    - If you want to create an immutable class, you start out with not providing any mutator methods

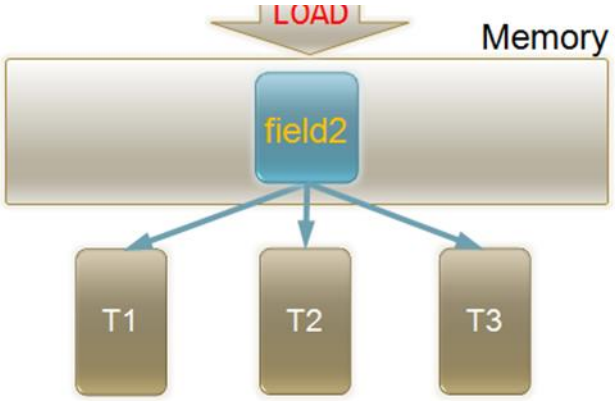
Class and Instance Fields

- A class field is shared among all instances of a class.

```
class A:
    def __init__(self, param):
        self.__field1 = param
```

```
class B:
    field2 = 3.5
    # methods
```





- To access class instance fields, one should use `className.fieldname`

Book ADT Exercise

- Create a class to represent a book in a library and test it with client code

Fields:

A counter that keeps track of the number of books in the library

List of strings of size 10 for authors

String for a title

integer for edition

library id – each new book gets a new unique id, starting with number 1

Behaviors:

Parameterized constructor that takes authors list, title, and edition number

Getters and setters for each instance field

String method

Getter for the number of books in the library

Make sure no encapsulation violations occur (authors list)

Procedural vs Object-Oriented

- Until chapter 10 we focused on procedural programming:
  - The steps to be followed to achieve the solution
  - Typically when looking at the problem description, we look at actions/verbs and figure out what to do

- The other type of programming paradigm is called object-oriented paradigm:
  - In the OOD, we look for objects we need to create in our program first and then we implement actions they need to perform
  - When looking at the problem description, we look at nouns and figure out how to simulate such data first (ADT – abstract data type)

Final Exam

Last Slide ☺

- Class ends at 17:10