&
Object-Oriented Programming

(POOP)

Modified from CS61A Summer 2012 Lecture at UC Berkeley

Where We Were: Functional Programming

Style of programming

One of many *programming paradigms* where **functions** are the central players.

Functions can work on data, which can sometimes be other functions.

Where We Were: Functional Programming

Key Feature: Composition

Functions can receive input from, and provide output to, other functions.

Many problems can be solved by “chaining” the outputs of one function to the inputs of the next.

Where We Were: Functional Programming

Key Feature: Composition

Example: Sum of all prime numbers from 2 to 100.

3. Sum the prime numbers up.

2. Filter out the prime numbers.

```
sum(picksPrimes(generateNums(2, 100)));
```

1. Generate all the prime numbers from 2 to 100.

Where We Were: Functional Programming

Key Feature: Statelessness

Functions always produce the *same outputs* for the *same inputs*.

This makes them incredibly useful for, for example, processing *a lot* of data *in parallel*.

Where We Were: Functional Programming

Key Feature: Statelessness

Because of statelessness, if we need to *update* a piece of data, we need to create a whole new piece of data.

```
String x = “Hello”;
words = new String[16];
words[15] = x;
```

Where We Were: Functional Programming

Key Feature: Statelessness

Functional programming is clunky for data that *changes over time*, or that has *state*.

We need an easier way to work with stateful data.

Object-Oriented Programming

A new programming paradigm where **objects** are the central players.

Hence the term "object-oriented".

Objects are data structures that are combined with associated behaviors.

They are "smart bags" of data that have state and can interact.

Functions can do one thing; objects can do many related things.

Terminology

OBJECT

CLASS

Any **person** is a **human**.

A **single person** has a **name** and **age**.

INSTANCE of the Human class

INSTANCE VARIABLES

Terminology

An **object** is an **instance** of a **class**.

For example, a person is an instance of a human.

The class describes its objects: it is a *template*.

Terminology

Objects and instance variables have a "has-a" relationship.

An **instance variable** is an ***attribute*** specific to an instance.

An **object** ***has an*** **instance variable**.

Terminology

METHODS

A single person can **eat** and **sleep**.

The **population** of the Earth is 7 billion.

CLASS VARIABLES

Terminology

Objects have certain behaviors, known as **methods**.

There are attributes for the class as a whole, not for specific instances: these are **class variables**.

Pokémon OOP

```
public class Pokemon
{
    static int totalPokemon = 0;
    String name;
    String owner;
    int hp;

    public Pokemon(String name, String owner, int hit_pts)
    {
        this.name = name;
        this.owner = owner;
        this.hp = hit_pts;
        Pokemon.totalPokemon += 1;
    }
}
```

CLASS

CLASS VARIABLE

INSTANCE VARIABLES

http://pkmnhackersonline.com/wp-content/uploads/2012/06/pokemon-logo.jpg

Pokémon OOP

```
public class Pokemon
{
    static int totalPokemon = 0;
    String name;
    String owner;
    int hp;

    public Pokemon(String name, String owner, int hit_pts)
    {
        this.name = name;
        this.owner = owner;
        this.hp = hit_pts;
        Pokemon.totalPokemon += 1;
    }
}
```

CONSTRUCTOR

this refers to the INSTANCE.

Class variables are referenced using the *name of the class*, since they do not belong to a specific instance.

Pokémon OOP

```
public class Pokemon
{
    static int totalPokemon = 0;
    String name;
    String owner;
    int hp;

    public Pokemon(String name, String owner, int hit_pts)
    {
        this.name = name;
        this.owner = owner;
        this.hp = hit_pts;
        Pokemon.totalPokemon += 1;
    }
}
```

Name *of* the instance (this)

"of"

Total number *of* Pokémon

Pokémon OOP

```
public class Pokemon
{
    ...
    public void increase_hp(int amount)
    {
        this.hp += amount;
    }
    public void decrease_hp(int amount)
    {
        this.hp -= amount;
    }
}
```

METHODS

Pokémon OOP

```
public class Pokemon
{
    ...
    public String getName() {
        return this.name;
    }
    public String getOwner() {
        return this.owner;
    }
    public int getHitPts() {
        return this.hp;
    }
}
```

SELECTORS

Pokémon OOP

```
Pokemon ashs_pikachu = new Pokemon("Pikachu", "Ash", 300);
Pokemon mistys_togepi = new Pokemon("Togepi", "Misty", 245);
System.out.println(mistys_togepi.getOwner());
```

Misty

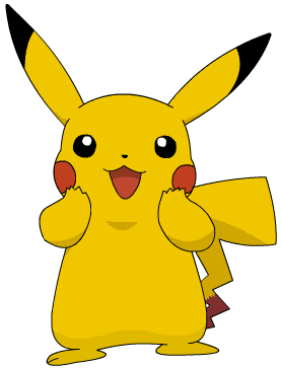
```
System.out.println(ashs_pikachu.getHitPts());
```

300

```
ashs_pikachu.increase_hp(150);
System.out.println(ashs_pikachu.getHitPts());
```

450

We now have state!
The same expression
evaluates to different values.



http://media.photobucket.com/image/pikachu%20surprised/Romantic_Princess/l_7b0cb7ca3a96b9e3350eef6a2c99f981.gif?o=2

Pokémon OOP

The statement

```
Pokemon ashs_pikachu = new Pokemon(“Pikachu”, “Ash”, 300);
```

instantiates a new object.

The Pokemon method (the constructor) is called by this statement.

Objects can *only* be created by the constructor.

Smart Bags of Data

```
Pokemon ashs_pikachu = new Pokemon("Pikachu", "Ash", 300);
Pokemon mistys_togepi = new Pokemon("Togepi", "Misty", 245);
```

The statements above create *two* new objects:

Instance variables:
name
owner
hp

Methods:
increase_hp
decrease_hp
getName
getOwner
getHitPts

Instance variables:
name
owner
hp

Methods:
increase_hp
decrease_hp
getName
getOwner
getHitPts

http://www.contrib.andrew.cmu.edu/~nguesto/images/pikachu.jpg
http://media.photobucket.com/image/recent/BrazilianYeti/Togepi.png

Smart Bags of Data

Each object gets its own set of instance variables.

Each object is a “smart bag” of data: it has data and it can also manipulate the data.

Instance variables:
`name`
`owner`
`hp`

Methods:
`increase_hp`
`decrease_hp`
`getName`
`getOwner`
`getHitPts`

Instance variables:
`name`
`owner`
`hp`

Methods:
`increase_hp`
`decrease_hp`
`getName`
`getOwner`
`getHitPts`

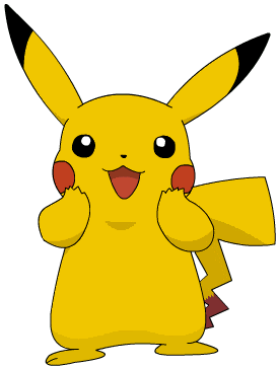
http://www.contrib.andrew.cmu.edu/~nguesto/images/pikachu.jpg
http://media.photobucket.com/image/recent/BrazilianYeti/Togepi.png

Calling Instance Methods

Each method belongs to an instance.

You call it using the name of that instance.

```
ashs_pikachu.increase_hp(150);
```



http://media.photobucket.com/image/pikachu%20surprised/Romantic_Princess/l_7b0cb7ca3a96b9e3350eef6a2c99f981.gif?o=2

Object Identity

Every object has its own set of independent instance variables and methods.

Assigning the same object to two different variables.

```
Pokemon ashs_pikachu = new Pokemon("Pikachu", "Ash", 300);
Pokemon brocks_pikachu = ashs_pikachu;
System.out.println(brocks_pikachu == ashs_pikachu);
true
```

The == operator checks if the two variables evaluate to the same object.

```
Pokemon brocks_pikachu = new Pokemon("Pikachu", "Brock",
300);
System.out.println(brocks_pikachu == ashs_pikachu);
false
```

OOP: Practice

Which methods in the Pokemon class should be modified to ensure that the HP never goes down below zero?

How should it be modified?

We modify the `decrease_hp` method:

```
public void decrease_hp(int amount)
{
    this.hp -= amount;
    if (this.hp < 0)
    {
        this.hp = 0;
    }
}
```

OOP: Practice

Write the method `attack` that takes another Pokemon object as an argument. When this method is called on a Pokemon object, the object screams (i.e. prints) its name and reduces the HP of the opposing Pokémon by 50.

```
System.out.println(mistys_togepi.getHitPts());
245
ashs_pikachu.attack(mistys_togepi);
Pikachu!
System.out.println(mistys_togepi.getHitPts());
195
```

OOP: Practice

Write the method `attack` that takes another Pokemon object as an argument. When this method is called on a Pokemon object, the object screams (i.e. prints) its name and reduces the HP of the opposing Pokémon by 50.

```
…
public void attack(Pokemon other)
{
    System.out.println(this.getName() + “!”);
    other.decrease_hp(50);
}
```

A Note About Variables

All instance variables so far have not had their visibility set.

We will want to change this later!

We will see why when we talk about public vs. private visibility.

The private visibility tells Java, and other Java programmers, to not use the variable outside the class.

http://images.wikia.com/poohadventures/images/5/5d/Brock.gif

Properties

We can create attributes that are computed from other attributes, but need not necessarily be instance variables.

Say we want each Pokemon object to say its complete name, constructed from its owner's name and its own name.

Properties

One way is to define a new *method*.

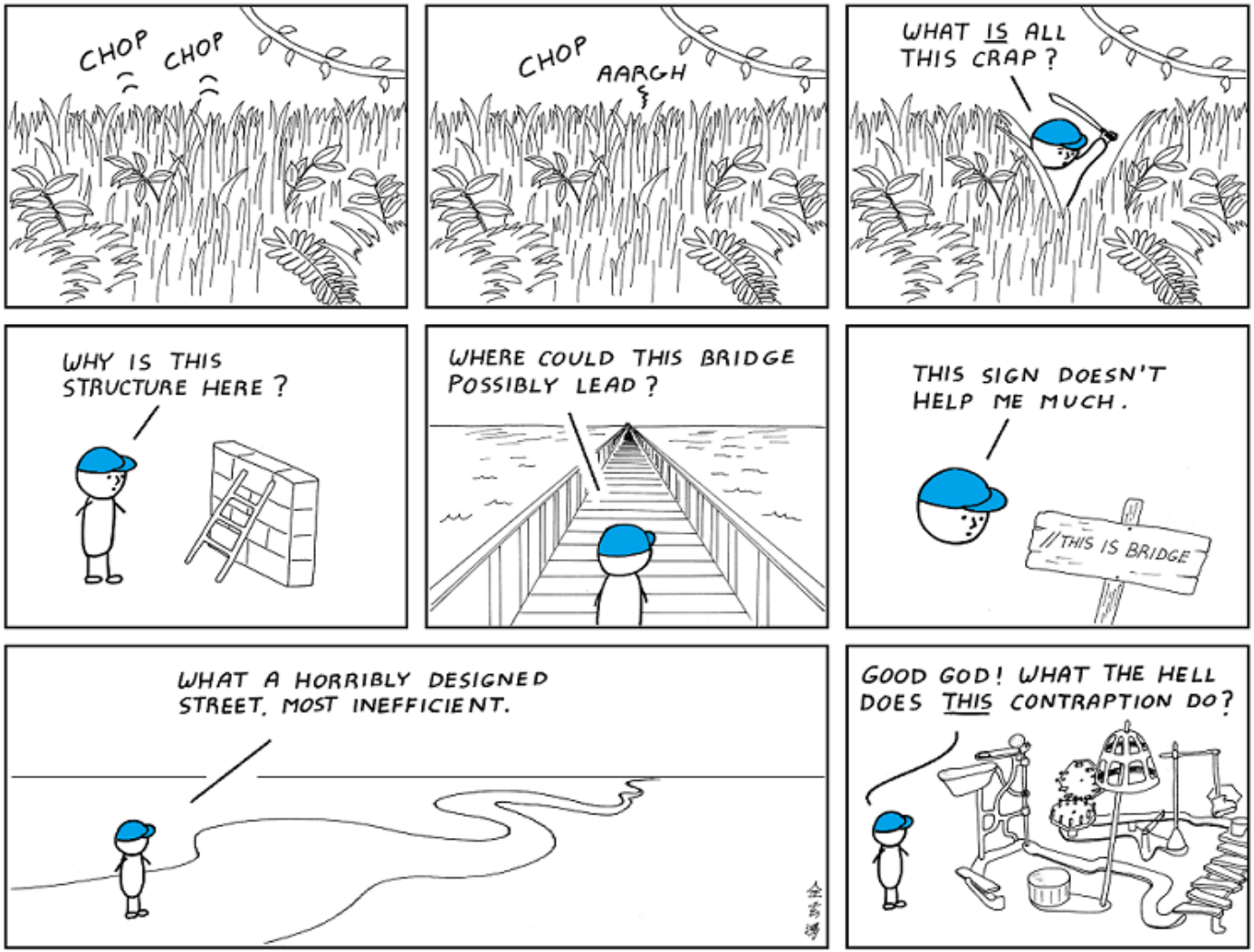
```
public class Pokemon
{
    ...
    public String completeName()
    {
        return this.owner + “’s ” + this.name;
    }
}

System.out.println(ashs_pikachu.completeName());
Ash’s Pikachu
```

However, this seems like it should be an attribute (something the data *is*), instead of a method (something the data can *do*).

Today's Coding Tip:

Make sure you (and others) can understand your code



I hate reading other people's code.

http://abstrusegoose.com/432