# Introduction to Objects & Classes

James Brucker

## What is ...

Bicycle

?

## Bicycle is a Kind of Thing

Bicycle is something that has:

- 2 wheels
- frame
- seat
- peddles
- gears
- color

## and it can ...

Bicycle can:

- move forward
- steer (change direction)
- apply power using your legs by pushing on peddles
- stop

## How to Describe "Bicycle"

| | |
|---|---|
| What it has or knows | - attributes |
| What it can do | - behavior |

## "Bicycle has Wheels, Gears, and Moves"

So, *what* ***size*** *are the wheels?*

*Is the bicycle* ***moving****?*

*How* ***many*** *gears?*

*What* ***<u>color</u>*** *is the bicycle?*

# How to answer?

So, *what size are the wheels?*

*... it depends on a particular bicycle*

*Is the bicycle moving?*

*... it depends on a bicycle and its state*

*How many gears?*

*... it depends on a particular bicycle*

*What color is the bicycle?*

*... it depends on a particular bicycle*

## Summary

"Bicycle" describes a class of objects (things).

Definition of "Bicycle" includes:

- attributes (what a bicycle has)
- behavior (what a bicycle can do)
- possible states (moving, parked, ...)

## What is an Object?

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

An object *encapsulates* both data and behavior.

An object contains both data and methods that operate on the data.

## Class

A Class is the definition (or blue print) for a kind of object.

A class defines:

- attributes - properties of object of this class
- behavior - what it can do
- states - how behavior depends on values of attributes

## Objects - Conceptual meaning

Objects represent "things" in the problem domain.

Examples:

| | |
|---|---|
| Banking app: | **money** |
| | **bank account** |
| | **customer** |
| Board game: | **game board** |
| (chess) | **game piece** |
| | **player** |

## Objects - your turn

Suppose you are writing an e-commerce application.

What are some *kinds of objects* you would need to model an e-commerse application?

______________________

______________________

______________________

______________________

______________________

______________________

## 3 Characteristics of Objects

Objects have

**Behavior** - what an object can do

**Attributes** or **Data** - what an object knows,

  or other objects it knows about (references)

**Identity** - two objects are unique, even if they have the same type and state

# Name some Classes in Python

| | |
|---|---|
| __________ | - class for strings |
| datetime.date | - dates on a calendar |
| __________ | - numbers like 1.25 |

## Creating Objects from Classes

```
s = str("hi there")       # create a string

d = datetime.date(2020, 12, 25)

f = 1.25
```

# String Class & Object in Java

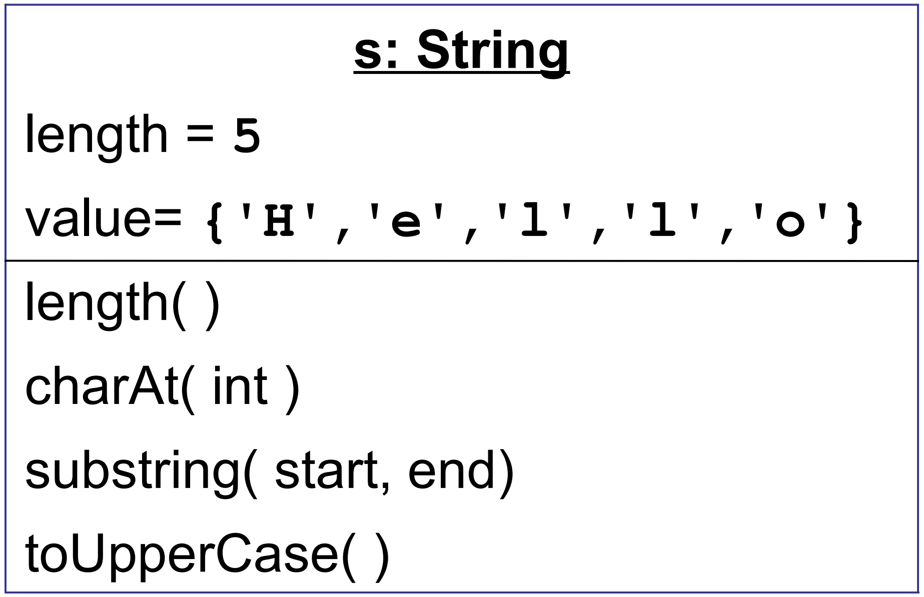
Consider a String object:

```
String s = "Hello";
```

What are the...

**attributes** - what the object ***knows*** (also called ***fields***)

**behavior** - what the object can ***do*** (its ***mehods***)



**<u>s: String</u>**

length = `5`

value= `{'H','e','l','l','o'}`

---

length( )

charAt( int )

substring( start, end)

toUpperCase( )

attributes are information an object remembers or stores
*Also called*: fields

behavior is what the object can do.
*Also called:* methods

## Objects have Behavior

To invoke an object's behavior, write:

**`object.method( )`**

A variable that *refers* to the object

A method that belongs to the object

```
>>> import datetime
>>> xmas = datetime.date(2020, 12, 25)
# What day of week is Christmas?
>>> xmas.ctime()
"Fri Dec 25 00:00:00 2020"
```

## Where does Behavior Come From?

An object's behavior is determined by …

1. **methods** defined in the object's class.

and

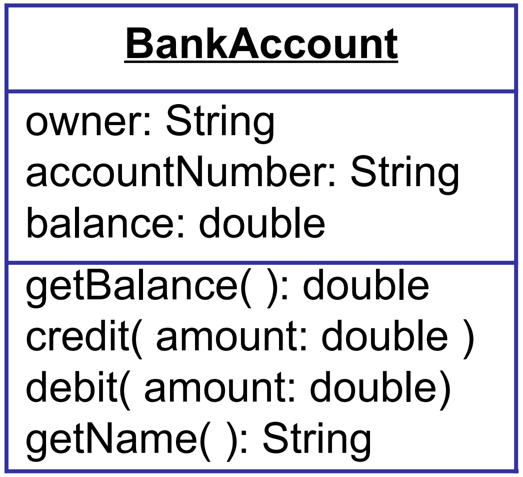
2. methods the class **inherits** from superclass, or super-superclass, etc.

## Attributes for Knowing stuff

Attributes store what an object knows.

Attributes are also called *fields*.

Example*: a Bank Account knows its account number, owner, and balance.*



## Objects have Identity

Two dates are distinct even if they have same values:

```
>>> x = datetime.date(2020, 1, 1)
>>> y = datetime.date(2020, 1, 1)
>>> x == y
True
>>> x is y
False
>>> id(x)              # every Python object has an id
139932742733136
>>> id(y)
139932742747800
```

# strings are tricky

Python and Java consolidate ("pool") string constants.

```
>>> x = "dog"
>>> y = str("dog")     # should be a new string
>>> x is y
True
>>> y = "DOG".lower()
>>> y
'dog'
>>> x is y
False
>>> x == y             # this invokes x.__eq__(y)
True
```

# Object Identity Example

- Two new Honda Civic cars made at the same factory on the same day with the same features ... can be distinguished.

!=

# Identity and == in Java

In Java, **`x == y`** always tests if x and y refer to the same object.

Objects are unique, even if their states are the same

```
Integer a = new Integer(10);
Integer b = new Integer(10);
a == b    // false - a and b refer to unique objects
```

Java primitive types only have a value, they are not objects.

```
int n = 10;
int m = 10;
n == m    // true - they are the same value
```

## Class defines a kind of object

Memorize this.

Definition:

"A **class** is a **blueprint** or **definition** for a ***kind*** of object**."**

`Sale` class defines the attributes of a sale.

`Sale` class defines the behavior (methods) of a sale.

`Sale` class defines how to create a sale.

# Two Ways to Create Objects

1. Invoke the constructor

```
>>> date = datetime.date(2020,5,1)
```

2. Some classes have a *factory method* to create objects

```
>>> now = datetime.date.today( )
# today() is a class method of date class
```

## Creating Objects in Java

1. Use "new" to create an object from a Class.

```
Date xmas = new Date(2020,11,25);
```

2. Some classes have a *factory method* to create objects.

```
LocalDate xmas = LocalDate.of(2020,12,25);
LocalDate today = LocalDate.now();
```

# A Variable is NOT an Object

A variable is only a ***reference*** to an object, not the actual object.

| | |
|---|---|
| `>>> s = "hi"` | s is NOT a string object |
| `>>> x = [1,2,3]` | x is NOT a List object |

## Other Use for Classes

Some classes don't represent "kinds of things".

Other uses are:

1. provide services

2. programming artifice - helps our code, but class has no meaning in the problem domain

## Class as Services

**`Math`** (Python **`math`**) provides services for doing math:

Math.sqrt( x )

Math.hypot( x, y )

Math.ceil( 1.00001 )

**`System`** provides access to operating system services

System.out - object connected to console output

System.in - object connected to console input

System.getenv(("USER") - get environment variable

## Class as Artifice: "application class"

We usually write a **Main** or **Application** class that does:

a) create initial objects

b) connect objects together (set references)

c) start or "run" the app

This class is useful, but doesn't represent a real thing.

```
public class GuessingGameApp {
    public static void main(String [] args) {
        Game game = new Game(100 /* max secret */);
        GameUI ui = new GameUI( game );
        ui.run();
    }
```

## Review

1. What is the definition of a **class** in OOP?

2. What are the **3 characteristics of objects**?

3. How do you create a Date object for the date Feb 15, 2000?

4. Is this true or false? Why?

```
Double x = new Double(1.0);
Double y = new Double(1.0);
(x == y)
```