Introduction to Objects

James Brucker

What is an Object?

An object is a program element that *encapsulates* both data and behavior.

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

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-commerse 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

State or **Knowledge** 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

String Object

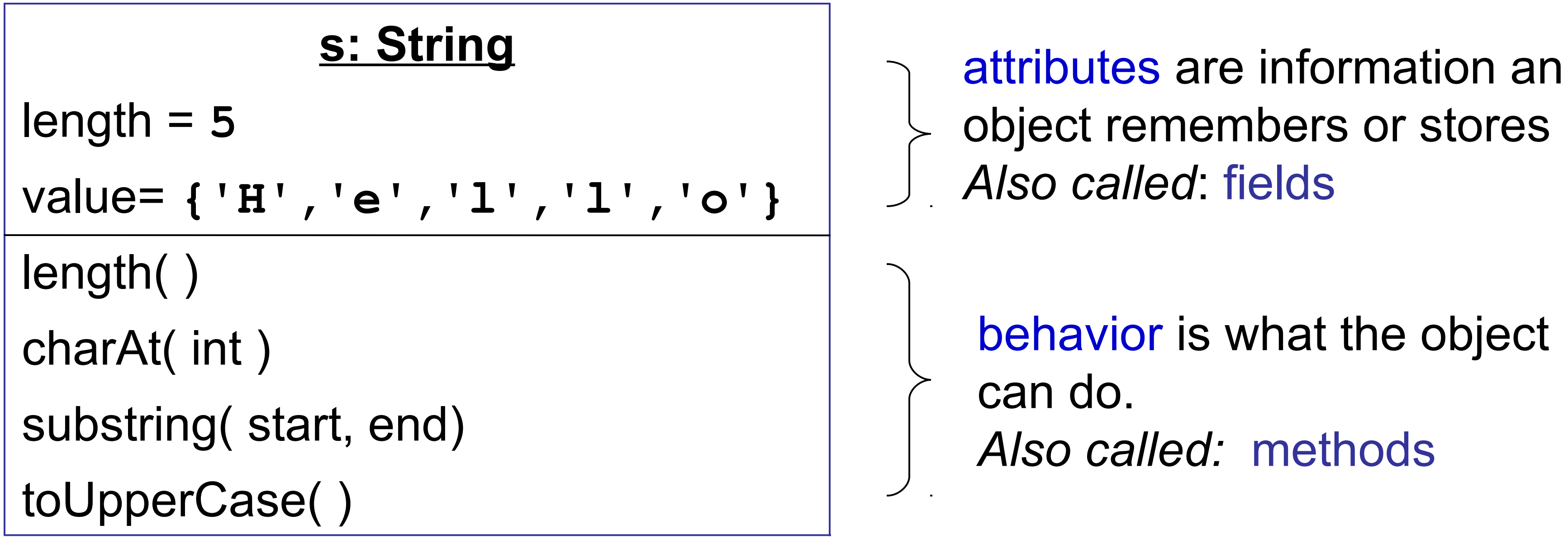
Consider a String object:

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

What are the...

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

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



s: String

length = 5

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

length( )

charAt(int)

substring(start, end)

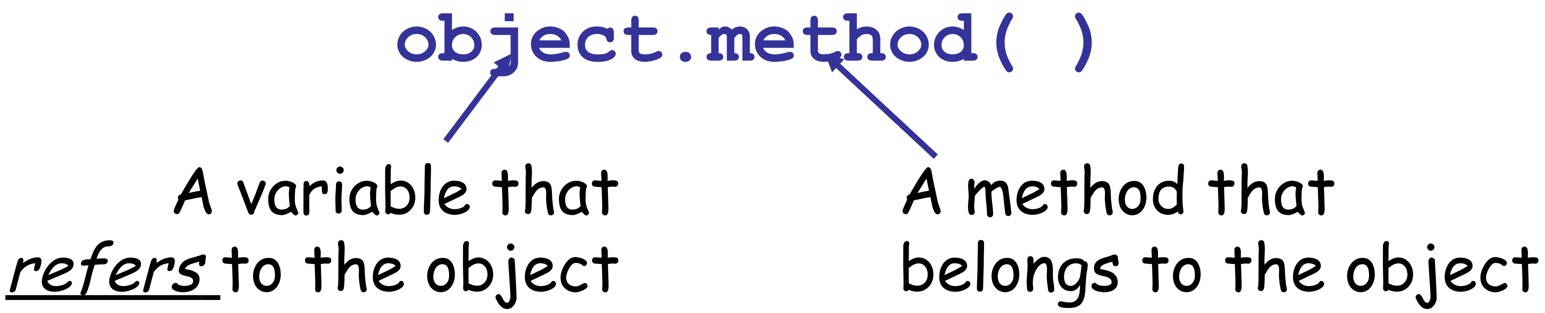
toUpperCase( )

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

```
> String s = "Hello Dog";
> s.length()
9
> s.toUpperCase()
"HELLO DOG"
> s.substring(0,5)     // method with a parameter
"Hello"
```

Where does Behavior Come From?

An object's behavior is determined by …

1. methods defined in object's class.

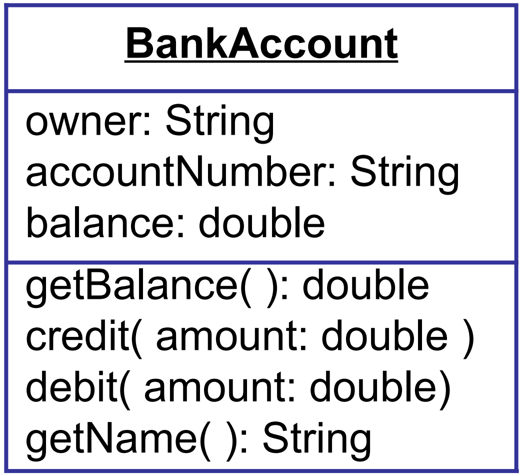
2. methods the class inherits from superclass, or super-superclass, etc.

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.*



BankAccount

owner: String
accountNumber: String
balance: double

getBalance( ): double
credit(amount: double)
debit(amount: double)
getName( ): String

Objects know about other Objects

An object can store references to other objects as attributes.

Example: a Quiz class contains references to Questions in the quiz.

```
class Quiz {
    private Question[10] questions;
    private int numQuestions;
```

Objects have Identity

These two strings are *distinct* even if have same values:

```
String s = "Dog";
String t = new String("Dog");
# == tests if two variables refer to same object
> s == t
false
> s.equals(t)
true
```

Be careful when comparing **string constants** ("dog"). Due to Java's *String pooling,* sometimes == works for Strings, sometimes not! Always use **`s.equals()`** to compare **values**.

More about identity...

Primitive types don't have identity. Only have a value.

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

But 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
```

Objects are *distinct*, even if identical

Objects are distinct!

Each "new" object is different, even if attributes are same.

```
                /* Date(year-1900, month, day) */
Date now1 = new Date(100, 0, 1 );
Date now2 = new Date(100, 0, 1 ); //same!
if (now1 == now2) /* same object? */;
```

FALSE

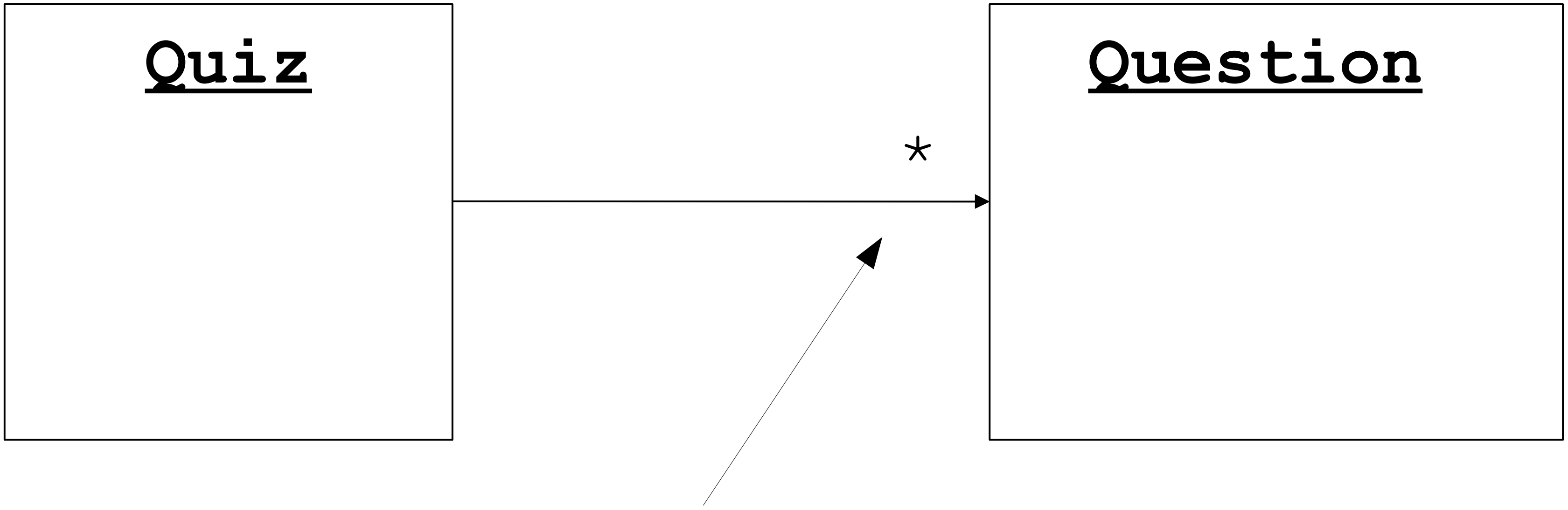
Object Identity Example

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

!=

Showing Relationships in UML

A Quiz has zero or more questions



* means "*zero or more*"

Where Do Objects Come From?

How to we define a *kind* of objects?

How do we define the attributes and behavior for a *kind of object*?

Examples:

- What is a `BankAccount?`
- What can a `BankAccount` do?
- How do we create a `BankAccount?`

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.

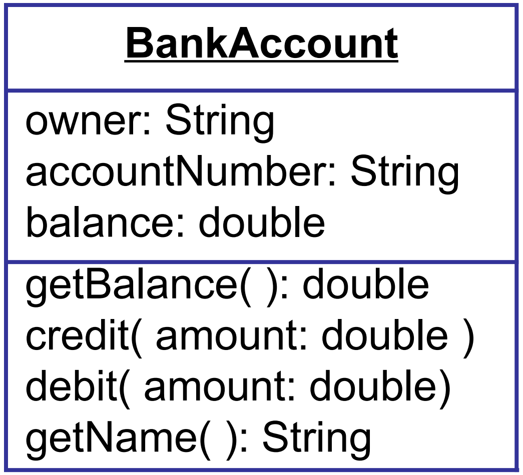
Sale class defines how to create a sale.

Class

A class defines the *attributes* and *behavior* that it will support.

Example:

class name:	**BankAccount**
attributes:	accountNumber, owner, balance
behavior:	getBalance( ), credit(amount), debit(amount), getOwner( )



BankAccount

owner: String
accountNumber: String
balance: double

getBalance( ): double
credit(amount: double)
debit(amount: double)
getName( ): String

A UML class diagram - Chapter 3 of UML Distilled

Object is an "instance" of a class

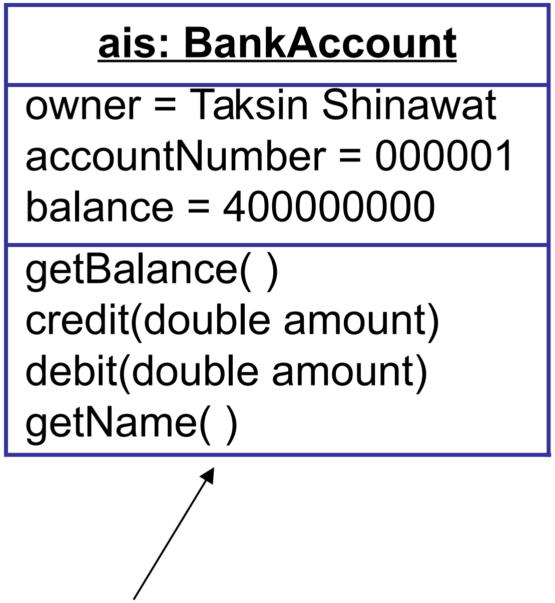
An object is an actual *instance* (instantiation) of the class.

Each object has its own set of attribute values, whose value may (and will) differ from other objects.

All *instances* share the same behavior.

Example:

```
BankAccount ais =
        new BankAccount("Taksin Shinawat" );
ais.credit( 40000000000 );
ais.credit( 32000000000 );
ais.debit( 50000000000 ); // seize his assets
ais.getBalance( ) ;  // = 22,000,000,000
```



ais: BankAccount

owner = Taksin Shinawat
accountNumber = 000001
balance = 400000000

getBalance( )
credit(double amount)
debit(double amount)
getName( )

A UML object diagram

Changing an Object's State

Some methods **change** an object's state (attributes).

These are *usually* "set" methods.

Change a date:

```
Date now =  new Date( );       // today
System.out.println( now );     // maybe 16 Jan 2013
now.setMonth( 11 );            // change to Dec.
now.setDate( 1 );              // 1st day of month
now.setHour( 12 );             // 12:00 noon
System.out.println( now );     // value has changed
```

More about Creating Objects

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

```
Coin fivebaht = new Coin(5);
```

fivebaht *is a* **reference** to a Coin object (like a pointer)

2. Some classes have a *factory method* for creating objects.

```
Calendar cal = Calendar.getInstance( );
```

getInstance( ) *is a static method that creates a new Calendar object*

Factory Method

To create a Calendar use getInstance( )

```
Calendar cal = Calendar.getInstance( );
```

java.util.Calendar does not have a public constructor.

- You can not write "new Calendar( )". (constructor is private)
- Instead you use the ***static method*** **`getInstance`**( ).

Reasons of this:

- Enable data validation before the object is created.
- Enable polymorphism: a factory method can return any compatible type, not just the declared type (Calendar).
- Hide complexity (Calendar depends on region).

Variables *refer* to Objects

```
String s;
```

Defines a variable named "s" of type String. It doesn't *refer* to any String object yet, so its value is null.

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

Make s refer to a String object "Hello"

```
s = new
String("Bye");
```

Make s refer to a String object "Bye".

```
String s2 = s;
```

Define another String variable s2, and make it *refer* to the object ("Bye") that s refers to.

This does not copy the object!

A Variable is NOT an Object

`String s;`	s is NOT a String object
`Date now;`	now is NOT a Date object.
`s = "hello";`	s is *still* NOT a String. s *refers to a String* object.
`now = new Date( );`	now is *still* NOT a Date. now *refers to a Date 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` 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.currentTimeMillis( ) - current time (millisec)
- 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 for coding, 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();
    }
```

Define Your Own Class

- This is covered in my "Java Basics" slides named Introduction-to-Java-2

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)
```