Lecture 10: Classes and Objects

Sierra College
CSCI-12
Spring 2015
Mon 03/02/15

Announcements

- **General**
  - I am behind on my grading, owing to heavy efforts on getting an online CS-12 thru distance learning review (by 3/2).
  - This is NOT typical for me in this course, and I will be getting back grading cranked out this week. ***I apologize for this slowdown in giving you assignment feedback!***
- **Schedule**
- **Past due assignments**
  - HW07: Variables, accepted thru Tues 3/3 @ 11pm
  - HW08: Operators, accepted thru Fri 3/6 @ 11pm
- **Current assignments**
  - HW09: External Input, due Tues 3/3 @ 11pm (*lab time today*)
- **New assignments**
  - HW10: Methods, due **Monday 3/9 @ 11pm** (lab time Weds)
    - Write a couple of simple nested Java methods, call them from main()
    - An extra weekend on this one, this assignment seems to “grind people’s gears” (but don’t wait until lab to get started on it)

Lecture Topics

- **Last time**:
  - Methods in Java
- **Today**:
  - Tail end of last lecture:
    - Calling methods
    - Procedure for writing a method (similar to HW10)
  - Classes and objects introduction
    - An overview of classes vs objects
    - The structure of classes

What’s A Class?

- A **class** is THE fundamental building block of all Java (and all other object-oriented, or O-O) programming
  - ***“EVERYTHING written in Java is a CLASS”***
- A **class** is the {***pick your term from below***} for some type of real-world or abstract entity: it is **a software “thing”**
  - Template
  - Blueprint
  - Generic description
- Classes couple together in one file:
  - What the thing **“is”** (how it is described): **variables**
  - What the thing **“does”,** or what can be done TO it **methods**

2 Characteristics of a Class

What is “is”

- Fields, or instance variables

- Attributes, data
- Descriptions
- “Nouns”

What it “does”

- Methods

- Behaviors
- Actions
- “Verbs”

What’s An Object?

- An **object** is one specific instance of a class
  - It is just like a new variable
  - Its datatype is that of its specifying class
  - Its data values are specific to that particular instance of the class
- Example:
  - *Cat* (class): name, breed, age, birthdate, owner
  - *myCat* (object): “Mr. Bigglesworth”, “Persian”, 7, 4/1/2008, “Dr. Evil”

Classes vs. Objects

- Classes and objects are central, dual concepts in O-O
  - **Classes** are the **general**
  - **Objects** are the **specific**
  - Class : Object ←→ Blueprint: House
  - Class : Object ←→ Cookie Cutter : Cookie
- The same class can be used repeatedly to create endless object instances
  - The same house blueprint can be used all over a development, but each house has a distinct address, paint scheme, and landscaping
  - The same cookie cutter can make endless dozens of cookies, but each cookie has its own flavor, frosting, and design

Examples of a Class

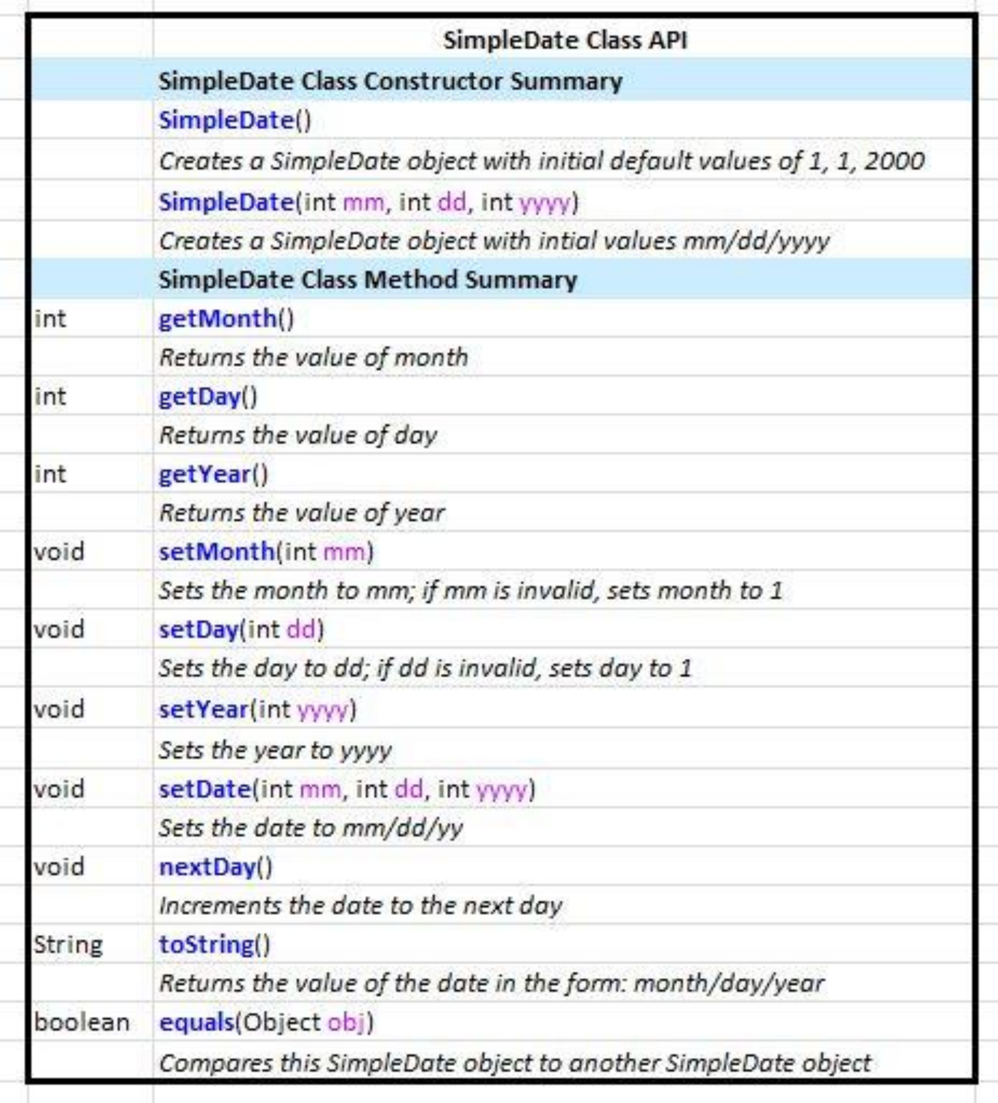
Class	Sample Attributes	Sample Behaviors
Person	firstName, lastName, birthdate, height, weight, …	eat(), sleep(), move(), breathe(), speak(), …
Dog	name, age, breed, weight, …	eat(), sleep(), run(), bark(), wagTail(), fetch(), giveBath(), …
Car	make, model, licensePlate, vin, color, odometer, …	start(), stop(), accelerate(), brake(), wash(), …
Course	courseId, department, subject, units, schedule, instructor, students, …	giveExam(), gradeAssignments(), addStudent(), evaluateCourse(), …
Computer	osName, osLevel, type, manufacturer, memory, screenSize, cpu, …	startUp(), shutDown(), changeSettings(), upgradeOs(), installApp(), connectToNetwork(), …

- Same examples as in an earlier lecture, except now we’ve updated everything to reflect good variable and method naming
- Also, we use ( ) to clearly distinguish methods

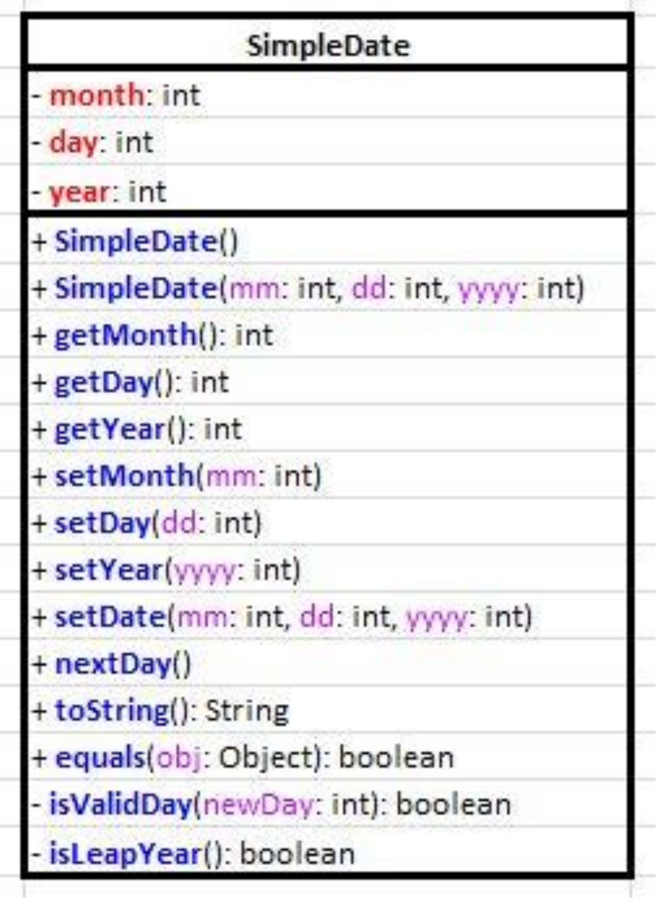
Advantages of Using Classes

- Efficiency
  - Code reuse → “intelligent laziness”
  - Well-written classes can be reused in many applications
  - Saves development time and cost
  - “Don’t reinvent the wheel” or “boil the ocean”
- Encapsulation
  - Data/operations cleanly packaged together, just use its API
  - Operations on data are better isolated
  - Easier to debug problems and maintain code
- Reliability
  - Use code which has been well-used and well shaken out
  - Proven, well-tested, modular software components

Class Example: *SimpleDate*



	SimpleDate Class API
	SimpleDate Class Constructor Summary
	SimpleDate()
	Creates a SimpleDate object with initial default values of 1, 1, 2000
	SimpleDate(int mm, int dd, int yyyy)
	Creates a SimpleDate object with intial values mm/dd/yyyy
	SimpleDate Class Method Summary
int	getMonth()
	Returns the value of month
int	getDay()
	Returns the value of day
int	getYear()
	Returns the value of year
void	setMonth(int mm)
	Sets the month to mm; if mm is invalid, sets month to 1
void	setDay(int dd)
	Sets the day to dd; if dd is invalid, sets day to 1
void	setYear(int yyyy)
	Sets the year to yyyy
void	setDate(int mm, int dd, int yyyy)
	Sets the date to mm/dd/yy
void	nextDay()
	Increments the date to the next day
String	toString()
	Returns the value of the date in the form: month/day/year
boolean	equals(Object obj)
	Compares this SimpleDate object to another SimpleDate object



SimpleDate
- month: int
- day: int
- year: int
+ SimpleDate()
+ SimpleDate(mm: int, dd: int, yyyy: int)
+ getMonth(): int
+ getDay(): int
+ getYear(): int
+ setMonth(mm: int)
+ setDay(dd: int)
+ setYear(yyyy: int)
+ setDate(mm: int, dd: int, yyyy: int)
+ nextDay()
+ toString(): String
+ equals(obj: Object): boolean
- isValidDay(newDay: int): boolean
- isLeapYear(): boolean

Class Depictions: Two Common Ways

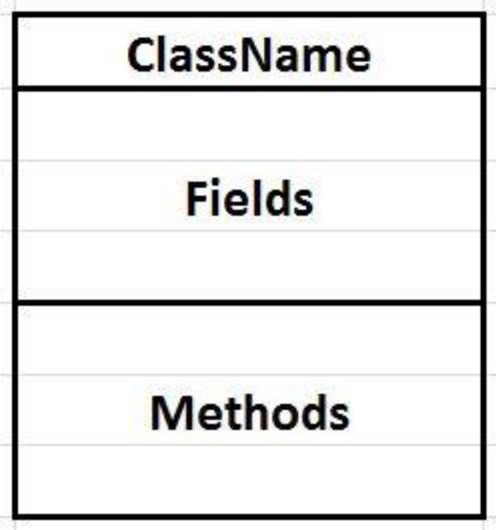
Class API

- **API** = Application Programming Interface
- What we'll usually see in our text
- Intended more for application programmers, users of the class
- Just the details needed to use the class
- Analogy: the owner's manual for your car

UML

- **UML** = Unified Modeling Language
- Intended more for s/w designers or architects, creators of the class
- All the details needed to code the class
- Analogy: the engineering drawings for your car

UML Class Diagram



- Classes are frequently depicted using a **UML class diagram**
- UML = Unified Modeling Language
  - Formal treatment of UML is beyond the scope of this course
- This lets us consider/design the needs of a class, without getting lost in its details quite yet
- We specify each method’s **calling interface**
- This is sometimes referred to as “Object Modeling”

What's In a Class?

- Classes are fully self-contained within their class source files (.java)
  - One class, one .java file
  - Class name and filename (.java) must match <u>exactly</u>
  - Class ***MyClass*** must reside within ***MyClass.java***
- Classes contain only two things:
  - **Fields/instance variables** (the data)
  - **Methods** (the operations)
- Collectively, these are called the class **members**

members = fields + methods

Class Naming Conventions

- Java classes should adhere to good naming conventions, just like any variables or methods
  - Should be “descriptive” and meaningful
  - Characters [A-Z, a-z, 0-9, a few others], no leading number, etc.
- Class names
  - Should be noun-based, since classes are “things”
  - Start with a **capital letter** (to differentiate them from variables/objects)
  - Internal words are capitalized (camelCase)
- Object names
  - Start with a **lower case letter** (just as for any other variable)
  - Internal words are capitalized
- Examples:

– Classes:	Person	Cat	School
– Objects:	joeSmith	fluffy	sierraCollege

Class Data

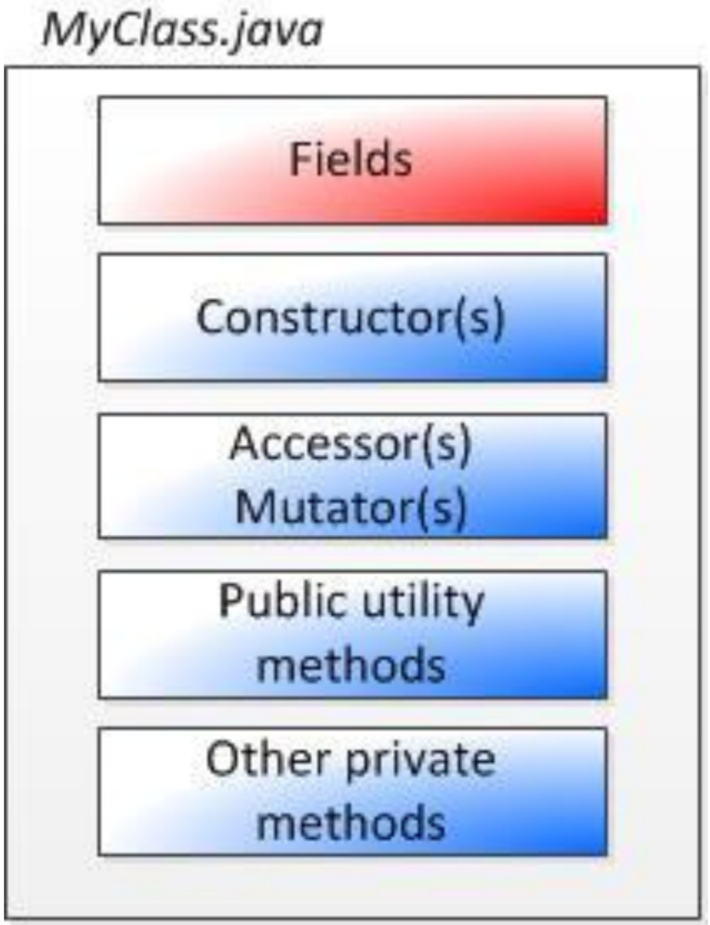
- Contained in **fields**, or **instance variables**
  - Simply the set of **variables** which describe a class
  - Their values are specific to each object instance of the class
  - Can be of any of the 8 primitive datatypes, or of some other class type
  - All class data should be declared **private**
- Taken together, they describe the full state of any object at any point in time
  - Class: *Student*
  - Class fields: *String* name, *SimpleDate* birthDate, *long* id, *double* gpa, *int* unitsCompleted
  - Object: joeJones
  - Object data: "Joe Jones", 7/18/1995, 900068312, 3.65, 46

Class Methods

- **Methods** provide interaction with the outside world
  - Specify what a class can do
  - Specify what the outside world can do to a class
- Methods are made known via the class API
  - Most methods are **public**, available for outside use
  - Some methods are **private**, for internal class use only
- Example method actions:
  - Initialize, query, and update fields
  - Perform computations, display data
  - Exchange data with other objects
- There are several general types of methods
  - **Constructors**
  - **Accessors/mutators**
  - Utility methods (public)
  - Other private methods

Anatomy of a Class

- This is a general layout of a .java class file
- Order is not set in stone, however:
  - Fields appear first
  - Constructors are the first methods



Class Constructors

- **Constructors** are special types of methods
  - They have the same name as the class itself
  - They are used to **instantiate** (create) new objects
  - They are used to initialize the data of a object, or perform any other startup computations
- Multiple constructors are permitted (this is common)
  - Alternate ways of instantiating a new object
  - Each one must be distinct in terms of argument number, datatypes, and/or ordering
  - This is an O-O concept called **overloading**

Constructor Types

- **Default constructor**
  - Has an empty method argument list
  - Up to the designer to specify "sensible" default field values
  - Example: SimpleDate() → defaults to 1/1/2000
- **Full constructor**
  - Each class field appears in the method argument list
  - Allows the application developer full control over new objects
  - Example: SimpleDate(9, 29, 2014) → 9/29/2014
- Other overloaded constructor forms
  - Other potentially useful forms are at class designer's discretion
  - Example: SimpleDate(12, 25) → 12/25/2014
    SimpleDate(2014) → 1/1/2014

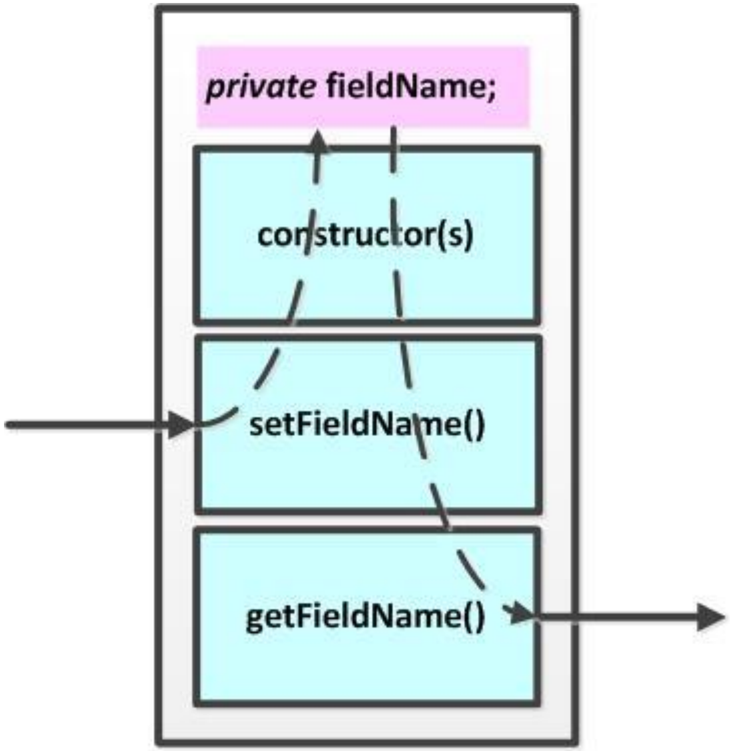
* Note: these last 2 forms are not actually provided by the *SimpleDate* API

Class Accessors and Mutators

- Class accessors and mutators are special purpose methods
  - **Accessors** are data-returning methods which get one field value (“getters”)
  - **Mutators** are *void* methods which set one field value (“setters”)
- Accessors and mutators typically follow a **get/set** naming convention:
  - Accessors: getFieldName()
  - Mutators: setFieldName()

Class Accessors and Mutators

- In a well-designed class, accessors and mutators provide the only outside access to instance variables
  - The instance variables are **private**
  - The accessors and mutators are **public**
- They are like an internal pipeline to/from the data, for the outside world
- ***"A class manages all of its own data"***



SimpleDate Accessor Methods

Return value	Method name and argument list
`int`	`getMonth( )` returns the value of *month*
`int`	`getDay( )` returns the value of *day*
`int`	`getYear( )` returns the value of *year*

SimpleDate Mutator Methods

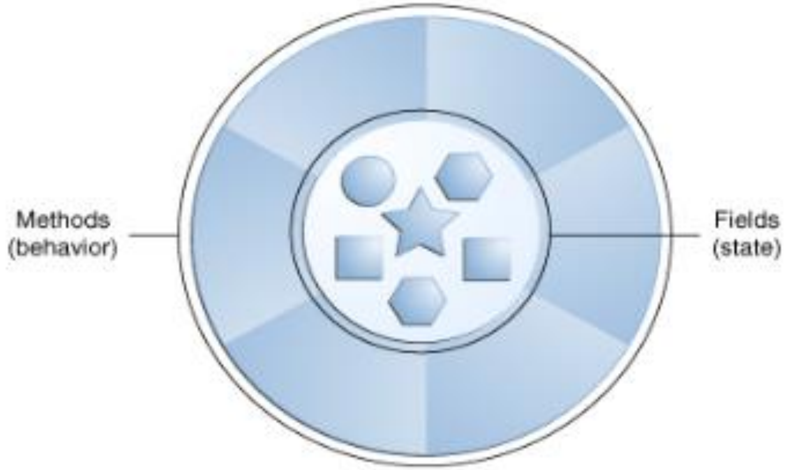
Return value	Method name and argument list
`void`	`setMonth( int mm )` sets the value of *month* to *mm*. If *mm* is not a valid month, sets *month* to 1.
`void`	`setDay( int dd )` sets the value of *day* to *dd*. If *dd* is not a valid day, sets *day* to 1.
`void`	`setYear( int yyyy )` sets the value of *year* to yyyy

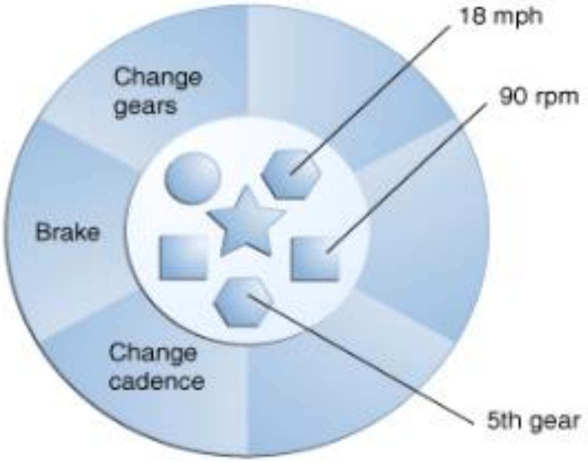
Encapsulation

- **Encapsulation** is an O-O concept which says that, in a well-designed class:
  - All class fields should be **private**
  - **Public** class methods (accessors/mutators) provide the ONLY interface to the data
- This type of **data hiding** has benefits:
  - Restricted access limits from-any-direction changes
  - **Data validity** can be enforced (only good values assigned)
- Simple analogy: a restaurant
  - We don’t just walk in back and randomly cook food
  - We are restricted to ordering off a menu, and via a server
  - This controls inventory, prevents kitchen chaos

Encapsulation Example

- Bicycle example, from the Oracle Java Trail
- A bicycle may have fields such as:
  - Cadence
  - Speed
  - Gear
- Class methods provide a “hard, protective shell” around the class data





For Next Time

- **Lecture Prep**
  - Text readings and lecture notes
- **Assignments**
  - See slide 2 for new/current/past due assignments