Lecture 11:
Using Classes

Sierra College
CSCI-12
Spring 2015
Weds 03/04/15

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**
  - Midterm exam: 3 wks from today (Weds 3/25), before spring break
- **Past due assignments**
  - HW08: Operators, accepted thru Fri 3/6 @ 11pm
  - HW09: External Input, accepted thru Tues 3/10 @ 11pm
- **Current assignments**
  - HW10: Methods, due Monday 3/9 @ 11pm (*lab time today*)
- **New assignments**
  - HW11: SimpleDate, due Tuesday 3/10 @ 11pm (lab time today)
    - Some simple manipulations of the *SimpleDate* class
    - We will go over examples of using *SimpleDate* in lecture today

Lecture Topics

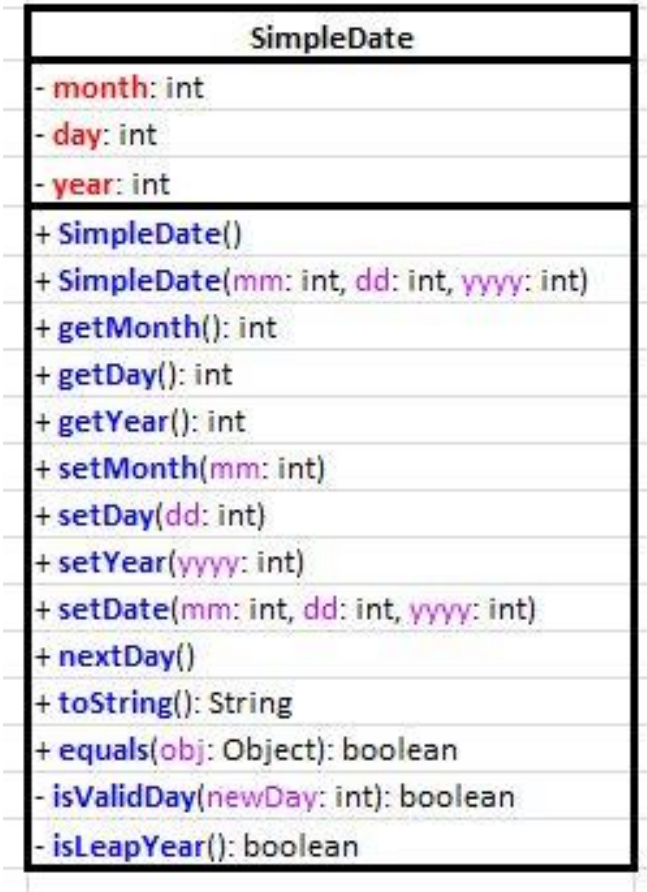
- **Last time**:
  - Calling methods
  - Creating methods
  - Classes and objects (cookie cutters and ~~Oreos~~ cookies)
- **Today**:
  - More on creating and using classes/objects
  - The *SimpleDate* class

Using Classes

- To **use** a class:
  - We DON'T need to know its internal details
  - We DO need to know its API, however
  - **API = Application Programming Interface**, its "documentation"
- The API for a class tells an application developer:
  - How to create new objects of that class type
  - What methods are available for use
  - How to call those methods
- If *Class1* uses *Class2*, we say that *Class1* is a **client** of *Class2*
  - The ***SimpleDateUsage*** class is a **client** of your ***SimpleDate*** class

Class API: *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



Using a Class Within Another Class

- First, both classes need to be mutually visible
  - For CS-12, we will simply make sure both .java files are in the same OS directory
  - For larger, or more general applications, we would use the **PATH** and **CLASSPATH** settings in our programming environment
- **Declare** an object
  - Just like declaring any other variable
  - Results in an **object reference**
- **Instantiate** the object
  - Call one of the constructor method forms, using the ***new*** keyword
  - Creates an actual new object in memory
- **Use** the object in the application program
  - Access the various methods using **dot notation**

SimpleDate Class Constructors

SimpleDate Class Constructors
`SimpleDate( )` creates a *SimpleDate* object with initial month, day, and year values of 1, 1, 2000
`SimpleDate( int mm, int dd, int yyyy )` creates a *SimpleDate* object with initial month, day, and year values of *mm*, *dd*, and *yyyy*

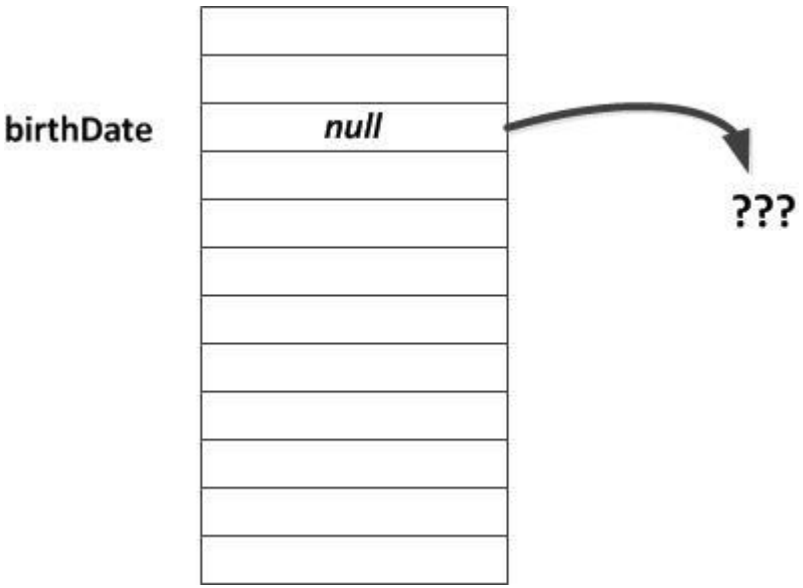
Declaring Objects

- Objects, like any other type of variable, must first be **declared**

- Syntax:
  ClassName object;
  ClassName obj1, obj2, … ;

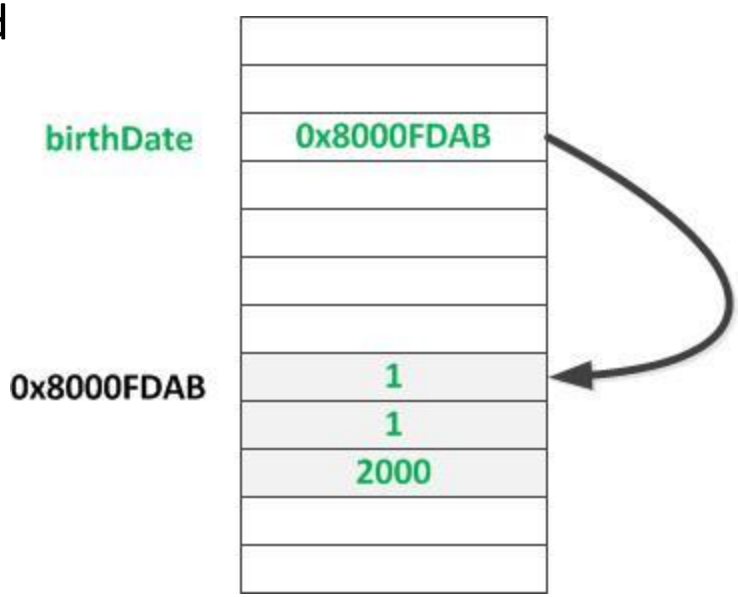
- Examples:
  SimpleDate birthDate;
  SimpleDate gradDate, holiday, nextWeekend;

- Declaration of an object results in an **object reference**
  - **Object reference** is an identifer which stores the **memory address** of the object
  - At this point, the object reference is ***null***, because it doesn't point anywhere… yet
  - The object cannot be used yet
  - Similar to an uninitialized pointer in C/C++;



Instantiating Objects

- After an object is declared, it must be **instantiated** before it can be used
  - Instantiation sets aside space in memory
  - One of the constructor methods is called using the **new** keyword
  - As with variables, declaration and instantiation may be combined into one statement



- General syntax:

  object = new ClassName(argList);

- Examples:

  birthDate = new SimpleDate();

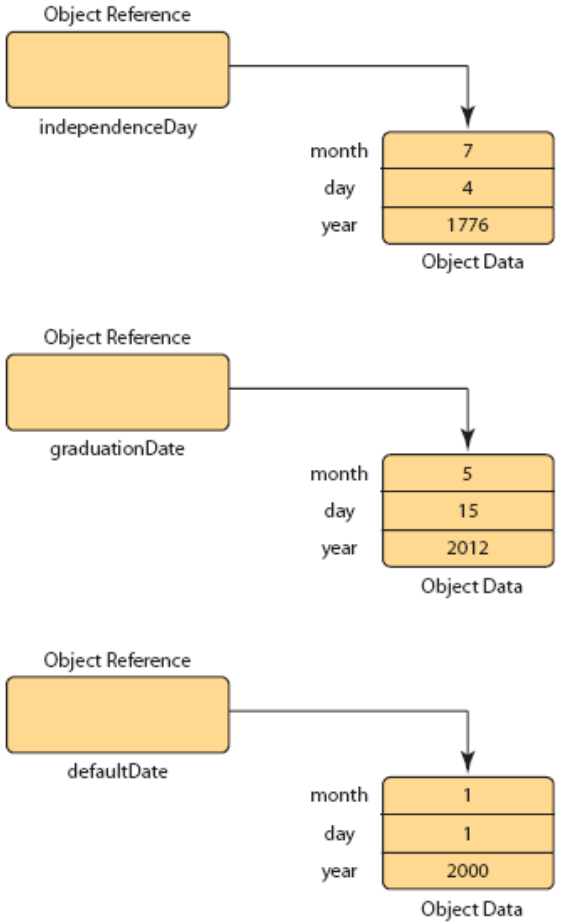
  SimpleDate today = new SimpleDate(3, 4, 2015);

Objects After Instantiation

```
SimpleDate independenceDay;
independenceDay =
    new SimpleDate( 7, 4, 1776 );

SimpleDate graduationDate =
    new SimpleDate( 5, 15, 2012 );

SimpleDate defaultDate =
    new SimpleDate( );
```



Calling Methods

- When we want to use methods in a client program:
  - We need to specify which object's data should be used
  - We use them in the context of that specific object
  - We call them using **dot notation**
- General form:

  object.method(arg1, arg2, arg3, …)
- Example:

  SimpleDate halloween = new SimpleDate(); // now 1/1/2000

  halloween.setDate(10, 31, 2015);

- See ***SimpleDateUsage.java*** in **Example Source Code**

Learning Your Way Around a New Class

- Inspect its API
  - Constructors (ways to create a new object)
  - Other methods (what to do with that new object)
- Create a new object
  - ***new*** keyword
  - Use any/all constructor forms
  - Print its starting state
- Manipulate the object
  - Make some changes using its methods
  - Print the state after
  - Try extracting some data with its accessors

- ***Use this paradigm in completing your next few homework assignments, using some new classes***

```
Widget  myObj = new Widget(…);
int field1;

// starting object state
System.out.println(“starting out: “ +
                              myObj);

// make some change
myObj.someMethod1(…);
System.out.println(“after step 1: “ +
                              myObj);

// make another change
myObj.someMethod2(…);
System.out.println(“after step 2: “ +
                              myObj);

// extract some of its data
field1= myObj.getField1();
System.out.println(“field1 is: “ + field1);
```

Object References

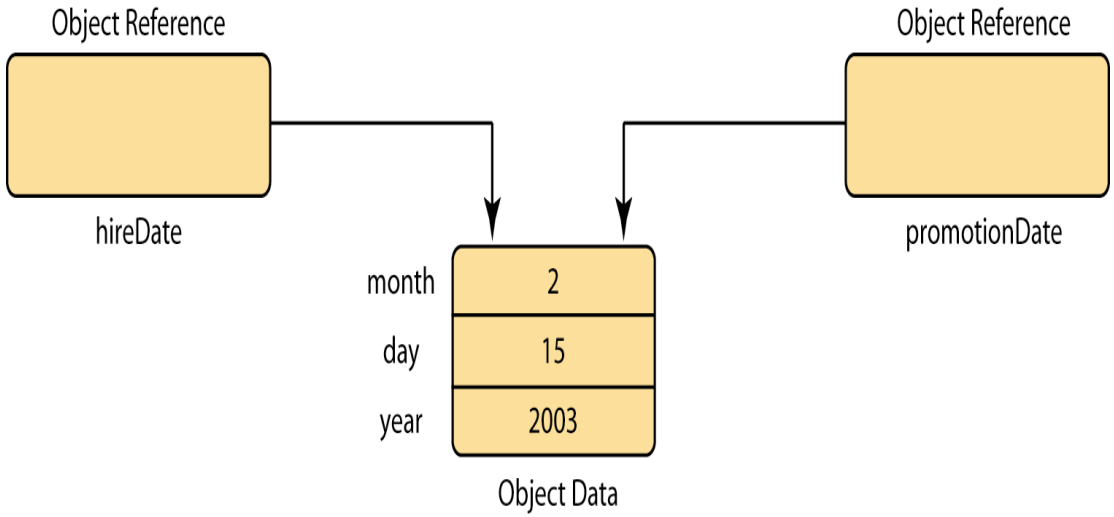
- **Object references** point to the memory storage location of the actual object data
- One object's data:
  - Can have multiple object references pointing to it
    - For example, object references can be assigned (=) to one another
  - Can have NO object references pointing to it
    - If so, that piece of memory is marked for **garbage collection** by the JVM
    - JVM will free up the memory if no object references point to specific object data
    - This can be forced by setting a particular object reference to ***null***

- See ***ObjectReferences.java*** in **Example Source Code**

Multiple Object References to Data

- Two object references can point to the SAME data

```
SimpleDate hireDate, promotionDate;
hireDate = new SimpleDate(2, 15, 2003);
promotionDate = hireDate;
```



null Object References

- If an object reference points to no object data, it has a ***null*** value
  - Example: an object has been declared, but not yet instantiated
  - Example: we want to delete an object by marking it for **garbage collection** by the JVM
- Trying to use or access a null object reference will cause a run-time ***NullPointerException***

- See ***ObjectReferences.java*** in **Example Source Code**

A Recap of Classes (1)

- **Classes** are the general blueprints, **objects** are their specific instances (cookie cutters and cookies)
- Classes encapsulate:
  - What a class IS (**fields**) – (almost) always **private**
  - What a class DOES (**methods**) – many will be **public**
- A class is fully described by its **API**
  - How to create one (**constructors**)
  - What can be done to or with it (**methods**)
  - Only what we need to know, no internal details

A Recap of Classes (2)

- By conventions:
  - Class names begin with an upper-case letter
  - Object names begin with a lower-case letter (like for any other variables)
- For application purposes:
  - Class → a new datatype
  - Object → a new variable of that datatype
- Objects are instantiated by using class **constructors**
  - Special methods with same name as the class itself
  - Multiple forms commonly exist, this is called **overloading**
- New objects must be declared and instantiated, just like for any variable

```
SimpleDate  defaultDate;
defaultDate = new SimpleDate();                    //  1/1/2000, using default constructor
Simple Date specificDate = new SimpleDate(9, 11, 2001);    //  using full constructor
```

A Recap of Classes (3)

- Once an object has been instantiated, you have full access to all of its methods using dot notation:

```
defaultDate.setDate(12, 25, 2014);          // now 12/25/2014
defaultDate.nextDay();                      // now 12/26/2014
mon = defaultDate.getMonth();               // assumes mon is an int (12)
defaultDate.setMonth( mon-2 );              // now 10/26/2014
stat = defaultDate.equals(specificDate);    // assumes stat is a boolean
System.out.println(defaultDate.toString()); // prints 10/26/2014
System.out.println(stat);                   // prints false
```

- **Important: API methods are usually called in the context of an existing object, not alone**
  - We'll see the exception to this when we consider *static* classes

A Recap of Classes (4)

- When an object variable is **declared**, it really results in an **object reference** (a “pointer” to some TBD memory location)
- Once the object is **instantiated**, the object reference finally points to some allocated memory
- Multiple objects (object references) can point to the same allocated memory
- Object variables can be “deleted” by setting them to *null*
  - If nothing points to certain object memory, it is marked for “**garbage collection**” by the JVM

For Next Time

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