Objects and Classes

Lecture 3

Object-Oriented Programming

Agenda

- Objects Introduction
- Capabilities of an Object
- Properties of an Object
- Object State
- Classes
- Object Classes
- Object Instances
- Objects in Memory
- Existing Classes and Objects
- String Object
- Reusing Existing Classes
- Constructor
- Third Party Classes
- Turtle World Examples

What are Objects?

- Building blocks of a software
  - A set of cooperating objects that work together by sending messages to each other
- Object model *tangible* things
  - school
  - Car
- Objects model *conceptual things*
  - meeting
  - date
- Objects model *processes*
  - finding a path through a maze
  - sorting a deck of cards

What does an Object have?

- Objects have
  - *capabilities*: what they can do, how they behave
  - *properties*: features that describe them

Capabilities of an Object

- Objects have *capabilities* that allow them to perform specific actions
  - objects are smart—they "know" how to do things
  - an object gets something done only if some other object tells it to use one of its capabilities
- Also called *behaviours*

Flavors of Capabilities

Capabilities can be:

- *constructors*: establish initial state of object's properties
- commands: change object's properties
- queries: provide answers based on object's properties

Example of Capabilities

- Example: *trash cans* are capable of performing specific actions
  - constructor: be created
  - commands: add trash, empty yourself
  - queries: reply whether lid is open or closed, or whether can is full or empty

Properties of an Object

- *Properties* determine how an object acts
  - some properties may be constant, others variable
  - properties themselves are objects — they also can receive messages
  - the *jug's* lid and its contents are also objects

Flavors of Properties

Properties can be:

- attributes: things that help describe an object
- components: things that are "part of" an object
- associations: things an object knows about, but are not parts of that object

Object State

- *State*: collection of all of an object's properties; changes if any property changes
  - some don't change, e.g., steering wheel of car
  - others do, e.g., car's color

Example of Properties

- Example: properties of *jug*

- attributes: color, material, smell
- components: lid, container
- associations: a *jug* can be associated with the room it's in

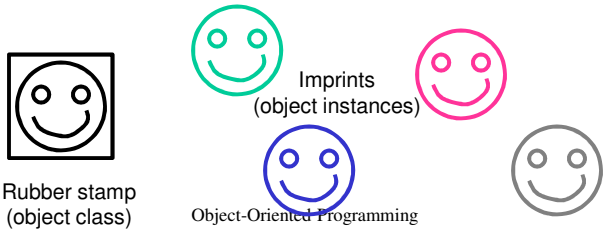
Classes

- Our current conception: each object corresponds directly to a *particular* real-life object, e.g., a specific atom or automobile

- Disadvantage: it's much too impractical to work with objects this way
  - there may be infinitely many objects (i.e., modeling all atoms in the universe)
  - may not want to describe each individual separately; they may have much in common

Classes (cont'd)

- Classifying objects factors out commonality among sets of similar objects
  - describe what is common just once
  - then "stamp out" any number of copies later



Object Classes

- *Object class*
  - a class is a category of object
  - defines capabilities and properties common among a set of individual objects
    - all *trash can*s can open, close, empty their trash
  - defines template for making *object instances*
    - particular *trash can*s may have a metal casing, be blue, be a certain size, etc.

Object Classes

- Classes implement capabilities as *methods*
  - a method is a sequence of statements in Java
  - objects cooperate by sending messages to others
  - each message "invokes a method"
    - i.e., Java executes the sequence of statements in the method in response to a message
- Classes implement properties as *instance variables*
  - slot of memory allocated to the object that can hold a potentially changeable value

Object Instances

- *Object instances* are individual objects
  - made from class template
  - one class may represent any number of object instances
  - creating an object instance is called *instantiating* that object
- Shorthand:
  - class: object class
  - instance: object instance (not to be confused with instance variable)
- Different instances of, say, `TrashCan` class may have:
  - different color and position
  - different types of trash inside
- So their *instance variables* have different values

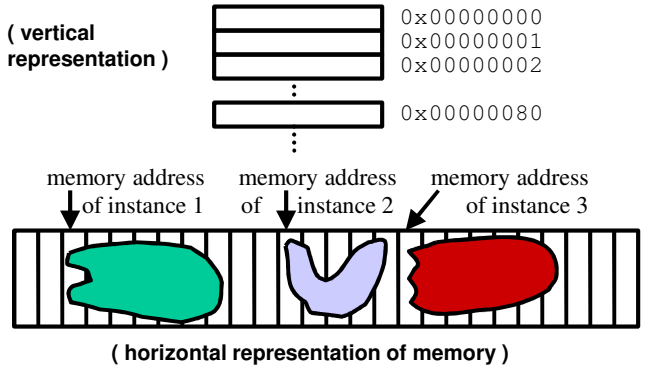
Object Classes

- Individual instances have individual identities
  - this allows other objects to send messages to given object
  - each instance is unique, even though they all have the same capabilities
  - think of class of 07CP students

Objects in Memory

- Every instance is stored in computer's memory
  - memory is a set of consecutively numbered storage locations, each containing a byte
  - each instance is stored in a series of contiguous bytes starting at a given location
- An instance is identified and referenced by unique *address* that refers to its starting location
  - address looks like `0xeff8a9f4` (hexadecimal notation, base 16)
  - just like postal address represents actual home
- But we know from our previous knowledge that the memory contents are written in binary format.
  - Hexadecimal numbers are commonly used for specifying memory addresses because the are easily converted to binary (base 2) numbers and they can be expressed in fewer digits than their decimal (base 10) counterparts.

Objects in Memory



Using Existing Classes and Objects

- Java allows code reuse. In fact this is one of the fundamental strength of Java
- Existing classes could be used by instantiating them into objects instances
- For using existing objects the compiler must know where they are located.
  - In Java existing classes are packed into *package*
  - Java compiler must know the location of the package to use the existing classes

String Object

- Remember the following Java syntax from last time

```
String city = "Lahore";
```

  - Why the "S" in String is not in small caps like int, double, etc.
  - String is an existing Java class and it is used to store values specified as strings.
  - As a rule of thumb in Java anything specified in double quotes " " is a String.

Reusing Existing Classes

- To use existing classes
  - We must know which package to use.
  - e.g. `java.lang.String`
  - java.lang is the package and String is the class being used from that package.
- Two types of reusable packages
  - Packages that are available with the Java compiler.
  - Packages developed by third parties.

Reusing Existing Classes (cont'd)

- Packages developed by third parties must be included in the *classpath* of the Java compiler.
- A classpath specifies the area on the computer where the compiler should search for the definition of the existing classes.

Syntax for Using Existing Classes

- Existing classes are instantiated into objects by this syntax.

```
String city = new String()
```

① ② ③ ④

① **Tells the compiler to search for the definition of a class named "String"**
② **A unique name for the object in the memory**
③ **"new" keyword to reserve the space for the type String in the memory**
④ **String() is a constructor of the class "String"**

Constructor

- *Constructor* is a special method that is called whenever a class is instantiated (created)
  - another object sends a message that calls a constructor
  - A constructor is the first message an object receives and cannot be called subsequently
  - *establishes initial state of properties* for instance
- Constructors have special syntax:
  - must always have *same name as class name*

Constructors (cont'd)

- A class could have many constructors e.g.

```
String city = new String()
&
String city = new String("Lahore")
```

Both construct a String object but in the first no initial value is given to the String and in the second a value "Lahore" is given.

Declaring a String

- Is there a difference between

```
String city = "Lahore"
```

&

```
String city = new String("Lahore")
```

- No - the first one is just a shorthand for the second one.
- This shorthand notation is only good for String objects, no other objects could be instantiated like this. For all the other objects one needs the second notation

Using Object Instances

- String is a class with capabilities and properties.
  - Properties are usually hidden from the outside world.
  - (Some) capabilities are exposed to the outside world.
- Which String capability we have seen so far?

Using Third Party Classes

- Library of turtles that move around inside a virtual world.
- The very first step to use this library/package is to create a world

```
World worldObj = new World();
```

Creates a world object and allocate it a space in memory.

On Computer Demo

Turtle World

- In this world i.e. worldObj we would like to put some turtles

```
Turtle turtle2 = new Turtle(30,50,worldObj);
```

`Turtle(30,50,worldObj)` is a constructor that specifies three propeties for the turtle2.

30, 50 specifies the location of turtle2 in the worldObj.

Demo

Using Turtle Capabilities

- The previous code performs the creation of the world and places a turtle in it.
- Now we could use some capabilities of turtle2 to perform some useful tasks.
- We will use the dot notation to invoke turtle2's capabilities.
- The dot notation will send a message to turtle to perform a specific task.
  - ***objectInstance . Message (paramterList)***

Using Turtle Capabilities

- We would like turtle2 to move forward 20 units.

```
turtle2.forward(20);
```

Using Turtle Capabilities

- We would like turtle2 to turn left.
  - `turtle2.turnLeft();`

Readings

Book Name: Introduction to Computing and Programming in Java A Multimedia Approach

Author: Mark Guzdial and Barbara Ericson

Content: Chapter 3

Book Name: Object Oriented Programming in Java – A Graphical Approach

Author: Kathryn E. Sanders & Andries van Dam

Content: Pages 17-39

Acknowledgements

- While preparing this course I have greatly benefited from the material developed by the following people:
  - Andy Van Dam (Brown University)
  - Mark Sheldon (Wellesley College)
  - Robert Sedgewick and Kevin Wayne (Princeton University)
  - Mark Guzdial and Barbara Ericsson (Georgia Tech)
  - Richard Halterman (Southern Adventist University)

Class Participation

- Q & A