Object Oriented Analysis and Design (OOAD)

Instead of functional AD
Instead of spaghetti coding
Like software version of integrated circuits. (Cox, 1986)

**object** *n.*
Something perceptible by one or more of the senses, especially by vision or touch; a material thing.
A focus of attention, feeling, thought, or action: *an object of contempt.*
The purpose, aim, or goal of a specific action or effort: *the object of the game.*

**Grammar.**
A noun, pronoun, or noun phrase that receives or is affected by the action of a verb within a sentence.
A noun or substantive governed by a preposition.

**Philosophy.** Something intelligible or perceptible by the mind.

**Computer Science.** A discrete item that can be selected and maneuvered, such as an onscreen graphic. In object-oriented programming, objects include data and the procedures necessary to operate on that data.
**object**  *n.*

Something perceptible by one or more of the senses, especially by vision or touch; a material thing.

A focus of attention, feeling, thought, or action: *an object of contempt.*

The purpose, aim, or goal of a specific action or effort: *the object of the game.*

**Grammar.**

A noun, pronoun, or noun phrase that receives or is affected by the action of a verb within a sentence.

A noun or substantive governed by a preposition.

**Philosophy.** Something intelligible or perceptible by the mind.

**Computer Science.** A discrete item that can be selected and maneuvered, such as an onscreen graphic. In object-oriented programming, objects include data and the procedures necessary to operate on that data.

**orient·ed, or·i·ent·ing, or·i·ents**  *v. tr.*

To locate or place in a particular relation to the points of the compass: *orient the swimming pool north and south.*

To locate or position so as to face the east.

To build (a church) with the nave laid out in an east-west direction and the main altar usually at the eastern end.

To align or position with respect to a point or system of reference: *oriented the telescope toward the moon; oriented her interests toward health care.*

To determine the bearings of.

To make familiar with or adjusted to facts, principles, or a situation.

To focus (the content of a story or film, for example) toward the concerns and interests of a specific group.

**v. intr.**

To turn toward the east.

To become adjusted or aligned.
OO – Object Oriented

• Data + Behavior = Object
  – Attributes / Variables
  – Methods

• Encapsulation / Data Hiding
  – Access to data and behavior can be restricted
    • Public, Private and Protected

• Inheritance
  – Children inherit the “characteristics” of Parents
  – Superclass <- Subclass

• Overriding behavior
  – Child can change the behavior of the parent by overriding the method of a the parent

• Overloading / Polymorphism
  – Same method name but different parameters means compiler selects method that matches the parameters in the invocation.
OO

• Building things that have behavior, instead of designing behavior and then passing around things (function oriented languages).

• Objects – Nouns – Person, Place or Thing (Concepts, too.)
  – Have *Attributes* – Size, Color,
  – Have Individual *Identity*
  – Have *Behavior* – do things - Verbs
OO Concepts

- Encapsulation
- “Shielding” – Information & Code Hiding
- Maintain State
- Send messages (invoke methods)
- Classes
- Inheritance
- Polymorphism
- Genericity
- Objects Have an unique Identity in runtime
Classes vs. Objects

- **Class definition** is a “blueprint”
- **Class** is group of related “things”
- Class definition and class are used semi-interchangably.
- **Object** is the “building” built from the “blueprint”
Objects

• Object is an **instantiation** of a class
• Many objects can be instantiated from a single class definition
  – Button is a class but a GUI might have many of the button object instances of the Button class
• Class also refers to the group that an object instance type belongs to
  – Objects are of a certain Class type
Inheritance

- A car is a type of Vehicle
- A PickupTruck is a type of Vehicle
- Vehicle is the **superclass** of Car & PickupTruck
- Vehicle is the **parent** class
public class Car extends Vehicle
{
    Trunk trunk;
    Color paintColor;
    Color interiorColor;
    Boolean convertible = false;
    static int TotalCars=0;

    public Color getPaintColor()
    {
        return paintColor;
    }

    public boolean setPaintColor(Color newColor)
    {
        paintColor = newColor;
    }
}
OO Concepts (2)

- **Encapsulation** – Attributes and Behavior are contained in the same structures in code (a class definition) and in memory as an object instance. Easy development, reuse, source management and management of code in runtime. Think “building blocks”

- **“Shielding”** - behavior and attributes are able to be made non-visible through visibility stereotype. Limits behavioral side effects caused by code somewhere else changing your “variables”. Think “integrated circuit” – Public, protected or private visibility.

- **Maintain State** – not just functions but the state and changes to that state are part of the object. “If you change the shoe color it stays changed in memory so the next time you reference an object the state change is preserved. The shoe color is the new color.”

*Objects are generally allocated on the heap not the stack and so can live longer than stack based memory “thingies”.*
**OO Concepts (3)**

- **Send messages (invoke methods)** – objects communicate with each other by sending and receiving messages (objects). In Java these are parameters to methods. Messages are interrogative, imperative or informative.
- **Classes** – objects are defined by a class definition which is used to define their initial state and attributes upon instantiation (creation in memory). Like a “blueprint” for an object instance.
- **Inheritance** – Class can be built on other classes. The inherit both the attributes of the parent class and the behavior of the parent class.
OO Concepts (4)

- **Polymorphism** – a method with the same name can be created as another, as long as they have different methods signatures. The method that matches the signature is the one that gets invoked.
  
  Ex:
  
  ```java
  add(int a, int b)  // adds the two parameters together
  add(int a, string b)  // converts the string to an int
  ```

  Also, operators (+,-,/,*) can be overloaded to provide polymorphism e.g. ‘+’ can mean addition for integers and concatenation for strings.

- **Genericity** – a class definition that has the types of objects manipulated defined in runtime. Parameterized class definitions or use of a base object to define the class and using object introspection in runtime to determine the type of the object. Think “an array of objects” and Collection classes.

- **Object ID** – each object is unique in runtime, even if I have multiple instances of the same class.
OO Visibility Modifiers

• **Public** – visible to all other classes
• **Protected** – visible to class and children of this class
• **Private** – visible only within the class not visible to its children
• **Package** – visible to all the classes in the package and nobody else
UML – symbology for class diagrams

- Public
- Private
- Protected
<<abstract>>
Vehicle
-------------
Wheel theSteeringWheel;
Tire tires[4];
Door doors[4];
Engine engine;
-------------
int pressGasPedal(int force)
int turnWheel(int degree)
int pressBrake();
int getCurrentVelocity();

F100Pickup
----------
Bed bed;
Color externalColor;
Color interiorColor;
Seat seats[2];
----------
boolean openDoor(int doorNumber);
boolean isDoorOpen(int doorNum);

Camaro
----------
AlpineStereo stereo;
Color externalColor;
Color interiorColor;
Seat seats[4];
----------
boolean openDoor(int doorNumber);
boolean isDoorOpen(int doorNum);
UML – modifiers & stereotypes

<<abstract>>
Vehicle
-----------
-Wheel theSteeringWheel;
-Tire tires[4];
-Door doors[4];
-Engine engine;
-----------
int pressGasPedal(int force)
int turnWheel(int degree)
int pressBrake();
int getCurrentVelocity();

F100Pickup
-----------
Bed bed;
Color externalColor;
Color tnteriorColor;
Seat seats[2];
-----------
boolean openDoor(int doorNumber);
boolean isDoorOpen(int doorNum);

Camaro
-----------
AlpineStereo stereo;
Color externalColor;
Color tnteriorColor;
Seat seats[4];
-----------
boolean openDoor(int doorNumber);
boolean isDoorOpen(int doorNum);
UML <<Stereotypes>>

- <<singleton>> – only one instance of a class is allowed to be instantiated. This single instance is shared by multiple client objects.
- <<abstract>> – a non-instanciable version of a class or method.
- <<interface>> – a specification of methods that must be supported by a class that implements the interface.
- <<utility>> – a collection of utility classes.
OO Example

- A Telephone is a Class (<<abstract>>)
- It has attributes
- It has several interfaces
- It has behavior

*abstract means it isn’t instanciable as an object. It must be extended to become a concrete class that is instanciable into an object instance.
OO Example

• **A Telephone is a Class** (<<abstract>>)  
• attributes  
  - color, size, brand, ...
• It has several interfaces  
  - a `voiceInputDevice`  
  - a `voiceOutputDevice`  
  - a `dialerDevice`
• It has behavior, you can:  
  - `getAConnection()`  
  - `makeACall()`  
  - `talk()`  
  - `listen()`
Telephone class

<<abstract>>
Telephone
--------------------------
-Color color;
-String phoneNumOfTelephone;
-String phoneType;
-boolean inAPhoneCall=false;
--------------------------
boolean getAConnection();
boolean makeACall(String num);
String listen();
boolean hangUp();

<<interface>>
IDialerDevice
--------------------------
boolean dialANumber(String num);

<<interface>>
IVoiceInputDevice
--------------------------
boolean talk(String);

<<interface>>
IVoiceOutputDevice
--------------------------
String listen();
Class vs. Object

Class

Objects (instances of a Class)
Class vs. Object

Objects (instances of a Class)

Class
Class vs. Object

Class

Objects (instances of a Class)
Subclasses of Telephone

• and instances of those subclasses
  • OldPhone, DeskPhone, DisplayPhone, CordlessPhone and DialPhone
A subclass that adds attributes

- A PayPhone is a subclass of a Telephone
- It also has a CoinSlot and CoinReturn
PayPhone class

<<abstract>>
Telephone
-------------
-------------

<<interface>>
IDialerDevice

<<interface>>
IVoiceInputDevice

<<interface>>
IVoiceOutputDevice

<<abstract>>
PayPhone
-------------
- float amountDeposited;
- CoinSlot myCoinSlot;
- CoinReturn myCoinReturn;
-------------
boolean insertCoin(float coin);
float pressCoinReturn();
What about these?

- All are types of Telephone
- They may have additional attributes
- A SpeakerPhone is an <<abstract>> class that is a subclass of Telephone
Derivation – Subclass is derived from superclass

- CellPhone is derived from Telephone
- SpeakerPhone is derived from Telephone
- FaxMachine is derived from Telephone

- All are <<abstract>> classes

- MotorolaV100 is a <<concrete>> subclass of a CellPhone
Subclasses – abstract and concrete

<<abstract>> Telephone

<<abstract>> CellPhone

<<abstract>> Fax<Machine>

<<abstract>> SpeakerPhone

<<interface>> IDialerDevice

<<interface>> IVoiceInputDevice

<<interface>> IVoiceOutputDevice

MotorolaV100

myWifesCellPhone: MotorolaV100
OO Naming Conventions & Syntax

- **Class TitleCase with first letter uppercase**
  - Shoe, AnalysisEngine, ContextFactory
- **Objects titleCase with first letter lower**
  - myShoe, analysisEngine, myAnalysisEngine
- **Class Methods TitleCase with first letter upper**
  - TotalNumShoes(), GetTotalObjectInstances()
- **Instance Methods titleCase with first letter lower**
  - getShoeSize(), getContext(), isShoeBeingWorn()
- **Class Attributes …**
  - long TotalShoesInMemory;
  - Url HostIDOfSingletonFactory;
- **Instance Attributes …**
  - int shoeSize, Color shoeColor, long contextID
Java is …

• An OO language
• Created in 1991-95! by James Gosling and others at SUN
• Originally called Green then Oak then Java
• Originally, Developed for mobile devices and set top TV boxes because C++ was too hard.
• tutorials - http://java.sun.com/docs/books/tutorial/index.html
Java is … (2)

- Platform Independent
- Object Oriented
- C++ like syntax
- Simple language structure
- Single inheritance
- Virtual Machine Based - runtime interpreter / compiler to pseudo machine code
Java is ... (3)

- Automated **garbage collection** – objects in memory will get deleted when they are no longer utilized (as opposed to C/C++ which was manual memory management and the major source of bugs)
- **References** instead of pointers – power of C++ pointers without any of the confusion, unsafety, complexity
Java is … (4)

• Everything is an **object** (almost – `int`, `long`, `byte`, `char`, `float`, `double` are *primitive* types). All classes are subclasses of `Object`.

• Method calls are **by reference** for all non-primitive types (they are pass by value). – by reference means that the method gets access to the callers copy and can change its state permanently.
Java is … (5)

• **Polymorphic** methods – an class can define multiple versions of a method, say:
  
  - `add(int a, int b)`
  - `add(int a, int b, int c)`
  - `add(String A, String B)`

  the correct method to invoke is determined by matching the parameters in the invocation (the method signature).
Java is … (6)

• No preprocessors. No macros.
• No constants.
• No operator overloading.
• No global variables.
• De facto standard (not an official open standard – SUN controls it).
• Getting really fast – tests with profiling JIT JVMs have found it to be up to 2x as fast as C++ code! (and getting faster.)
• Supported by all major vendors (except Microsoft) – IBM, Oracle, HP, Borland, …
Java History

• 1991: Oak is created for consumer electronics
• 1993: Oak used in interactive TV system dev
• 1994: Mosaic browser released
• 1995: Becomes Java – oriented to the Internet
• 1996: Netscape releases Navigator 2.0 (Java-enabled browser). Sun makes JDK freely available.
• 1997: EJB specification
• see http://java.sun.com/features/1998/05/birthday.html

Why did they pick Java as the name?
Java Technology Releases

- J2SE – Standard Edition
- J2EE – Enterprise Edition (Servlets/EJB)
- J2ME – Micro Edition (PDAs, CellPhone)
- other releases like Jini, Jiro,
Java J2SE – Standard Edition

- I/O – Files, Streams, Pipes
- Swing (GUI)
- Applets – run in a web page on the client
- RMI (Remote Method Invocation) – remotely calling other object’s methods
- Math
- JavaBeans – client side component model
- Security
- 2D Graphics
- Internationalization / Localization
J2EE – Enterprise Edition

- EJB (Enterprise Java Beans) – server side component model. Business logic and data logic components.
- Servlets – server side active web pages written in java. Runs on web server.
- JSP (Java Server Pages) – HTML + java mingled together that gets compiled into servlet automatically by the web server.
- JDBC – database access API
- JNDI (Java Naming and Directory Interface) – finding other objects and database tables, etc. Looking up distributed things in a central repository.
- JavaMail – email API
- JMS (Java Message Service) – message oriented middleware
- JCA (Java Connector Architecture) – standard adapters to legacy systems like mainframes or SAP r/3
Java Technologies
J2ME – Micro Edition

• Targeted at consumer electronics and embedded devices.
• Virtual machine and a minimal set of core libraries
• Extending the capabilities of the minimal configuration by adding additional libraries
Java API Relationships
JVM – Java Virtual Machine

• A piece of software that runs on top of a physical processor providing a layer of abstraction between Java programs and the processor.

• Standard instruction set (bytecodes) that get translated in runtime to architecture/processor specific operands (machine code).

• JVMs are written to match the operand of the specific platform (Win/x86, Solaris/SPARC, Linux/x86, IBM mainframe, cellphone, PDA).

• On windows invoked by running java.exe.
JVM

• **Bytecodes** are **verified**- checked to insure no illegal operations, pointer based bugs, etc. – bytecode verifier

• Interpreted in runtime – converted to machine specific instructions that are written to memory and executed by the processor

• Or the whole program can be **JIT** (just in time) Compiled to machine specific instructions. Faster execution.

• Or in runtime, code can be **profiled** and the most frequently executed code can be precompiled and cached in memory to improve performance with minimum memory utilization. (HotSpot from SUN)
Simple Java Program

```java
public static void main(String[] args) {
    System.out.println("Hello World");
}
```
the Java compiler
`javac.exe`

- Converts java code into a compiled class file (`java` byte codes in a `.class` file)
- Java byte codes are instructions for the Java Virtual Machine (JVM) to execute.
- Byte codes are for a “pseudo” or “virtual” microprocessor. They get translated into the machine specific/architecture specific instructions on the fly by the JVM.
The java JVM

java.exe

• To run a program in the JVM
• Use java.exe <classname>
• That class must have a method named “main()”
• Ex:
  
  ```java
  public static void main(String[] args)
  {
      System.out.println("Hello World");
  }
  ```

• Has to be a public static method.
• Parameters accepted are commandline arguments (if any)
• JVM jumps to that “main” method and executes from there.
• Example execution:
Using an IDE to create

Create a new NetBeans object, for example:

- A Swing Form
- An Applet
- A JavaBean
- A Java Server Page

Or, select from the complete set of object templates.

Tip of the Day

You can use VCS Groups to keep track of which version-controlled files have changed in your working directory by automatically adding all modified files to the default group.

Choose VCS Groups from the Versioning menu in the main window. Right-click the VCS Groups node and choose Properties. Then set...
Select “main”
name it "MyFirstClass"
click next, next, next, ...
Generated code
Add line and then press execute

class MyFirstClass

public static void main(String[] args) {
    System.out.println("Hello Class!");
}

add this line
Java Packages

• A hierarchical way of storing related classes.

• `package` statement tells compiler where to associate the class with.

• `import` statement tells compiler where to look for class definitions. Avoids having to fully qualify every thing e.g. use `String` instead of `java.lang.String`
Looking up java classes at
http://java.sun.com/j2se/1.3/docs/api/

Java™ 2 Platform
Std. Ed. v1.3.1
All Classes

Packages
java.applet
java.awt

java.util
java.lang

DataTruncation
Date
DateFormat
DataFormatSymbols
DateGraphics
DecimalFormat
DecimalFormatSymbols
DefaultFormatterRangeModel
DefaultButtonModel
DefaultCaret
DefaultCellEditor
DefaultChoiceModel
DefaultComboBoxModel
DefaultDesktopManager
DefaultEditorKit
DefaultEditorKit.BeepAction
DefaultEditorKit.CopyAction
DefaultEditorKit.CutAction
DefaultEditorKit.DefaultAction
DefaultEditorKit.DefaultAction
DefaultEditorKit.InsertBefore
DefaultEditorKit.InsertAfter

Overview Package Class Use Tree Deprecated Index Help
PREV CLASS NEXT CLASS
SUMMARY: INNER | FIELD | CONSTR | METHOD
PREV FRAME NO FRAMES
DETAIL: FIELD | CONSTR | METHOD

java.util
Class Date

java.lang.object

+------------------+
| java.util.Date   |
+------------------+

All Implemented Interfaces:

Cloneable, Comparable, Serializable

Direct Known Subclasses:

Date, Time, Timestamp

public class Date

extends Object

implements Serializable, Cloneable, Comparable

The class Date represents a specific instant in time, with millisecond precision.

Prior to JDK 1.1, the class Date had two additional functions. It allowed the interpretation of dates as year, month, day, hour, minute, and second values. It also allowed the formatting and parsing of date strings. Unfortunately, the API for these functions was not amenable to internationalization. As of JDK 1.1, the Calendar class should be used to convert between dates and time fields and the Date/Format class should be used to format and parse date strings. The corresponding methods in Date are deprecated.
Java Coding Conventions

• Class name Title Case with first letter Caps.
  - AutoPilotSystem
  - Order
  - Book
  - BookShelf

• Method and Attribute names in title case with first letter lower case.
  - String getPilotName(); //method
  - String pilotName; //attribute
  - String socialSecurityNumber;
  - Int age;
  - Book myBook;
package GVOnlineCommandline;

public class Book {
    int numOfPages = 0;
    String title;
    String authorName;
    String pages;
    public void book() {
        // constructor
    }

    public void setTitle(String newTitle) {
        title = newTitle;
    }
    public String getTitle() {
        return (title);
    }
}
running the javadoc processor
 GVBooksOnline.GVBooksOnline

Class MainAppFrame

java.lang.Object
 | 
 | ---java.awt.Component
 |     | 
 |     | ---java.awt.Container
 |     |     | 
 |     |     | ---java.awt.Window
 |     |     |     | 
 |     |     |     | ---java.awt.Frame
 |     |     |     |      | 
 |     |     |     |      | ---java.swing.JFrame
 |     |     |     |      |     | 
 |     |     |     |      |     | ---GVBooksOnline.GVBooksOnline.MainAppFrame

All Implemented Interfaces:

Deprecated. (see How and When To Deprecate APIs)

public class MainAppFrame
   extends javax.swing.JFrame

Copyright Poelman & Associates,
Inc. 2002 - All Rights Reserved
javadoc syntax

* @author (classes and interfaces only, required)
* @version (classes and interfaces only, required)
  (see footnote 1) *
* @param (methods and constructors only)
* @return (methods only)
  * @exception
  * @throws (is a synonym added in Javadoc 1.2)
* @see
* @since
* @serial (or @serialField or @serialData)
* @deprecated (see How and When To Deprecate APIs)
Summary

• OO
  – Encapsulation
  – Polymorphism
  – Inheritance
  – Encapsulation
  – Classes
  – "Shielding"
  – Maintain State
  – Send messages
  – Objects Identity
  – Genericity

• Java
  – OO
  – Platform Independent
  – Compiled and Interpreted
  – JVM
  – Garbage Collection
  – J2SE, J2EE, J2ME
  – NetBeans IDE