2190102 Advanced Computer Programming

Course Introduction

2190102 Advanced Computer Programming : *Atiwong Suchato*
Faculty of Engineering, Chulalongkorn University

Moving Beyond 2190101

- Learning *new programming concepts*.
- Developing programs in *standard application frameworks*.
- Applying the concepts to *different frameworks and programming languages*.
- Develop a "*Ready-to-ship*" applications (through class project).

2190102 Advanced Computer Programming : *Atiwong Suchato*
Faculty of Engineering, Chulalongkorn University

Corequisite Courses

2190102   Advanced Computer Programming
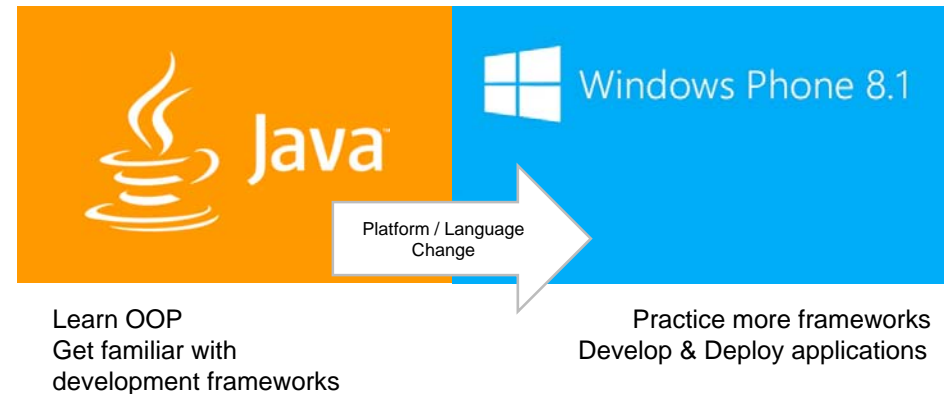2190152   Advanced Computer Programming Lab

Register both / Withdraw both

2190102 Advanced Computer Programming : *Atiwong Suchato*
Faculty of Engineering, Chulalongkorn University
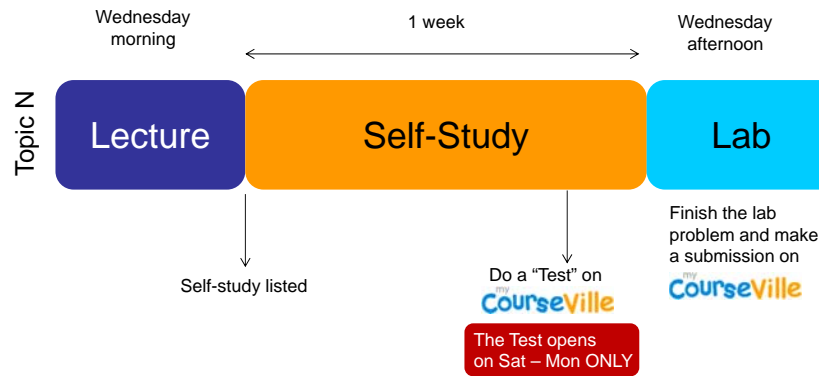

Chosen Technology

Java   Windows Phone 8.1

Platform / Language Change

Learn OOP
Get familiar with
development frameworks

Practice more frameworks
Develop & Deploy applications

2190102 Advanced Computer Programming : *Atiwong Suchato*
Faculty of Engineering, Chulalongkorn University

Weekly Learning Format

Topic N

Wednesday morning

1 week

Wednesday afternoon

Lecture — **Self-Study** — **Lab**

Self-study listed

Do a "Test" on
my CourseVille

The Test opens on Sat – Mon ONLY

Finish the lab problem and make a submission on
my CourseVille

2190102 Advanced Computer Programming : *Atiwong Suchato*
Faculty of Engineering, Chulalongkorn University

Schedule

Week#	Date	Topics in Lecture & Assigned Self-study	Lab Problem
1	13-Aug-14	Course Orientation Topic 1: OOP Concepts / Debugging in Eclipse / Java Applet	
2	20-Aug-14	Topic 2: Interface / Thread / Animation Applet	Topic 1
3	27-Aug-14	Topic 3: GUI Framework / Swing Application	Topic 2
4	3-Sep-14	Topic 4: Swing Application (Continued)	Topic 3
5	10-Sep-14	Topic 5: Collection Framework / I/O	Topic 4
6	17-Sep-14	Topic 6: .NET Framework / OOP with C#	Topic 5
7	24-Sep-14	Reviews	Topic 6
8		Midterm Exam (Topic 1 – 6)	
9	8-Oct-14	Topic 7: OOP with C# (Continued)	
10	15-Oct-14	Topic 8: Windows Phone 8.1 Programming / XML	Topic 7
11	22-Oct-14	Topic 9: Making Animations	Topic 8
12	29-Oct-14	Topic 10: Camera, Media, and Audio	Topic 9
13	5-Nov-14	Topic 11: Network Communication	Topic 10
14	12-Nov-14	Topic 12: Serialized Data / Asynchronous Tasks	Topic 11
15	19-Nov-14	Extra Topics	Topic 12
16	26-Nov-14	Project Presentation	
		Final Exam (Topic 7 – 12)	

2190102 Advanced Computer Programming : *Atiwong Suchato*
Faculty of Engineering, Chulalongkorn University

Grading

Self-study	Final Project
10	**30**
Midterm	Final
30	**30**

2190102 Advanced Computer Programming : *Atiwong Suchato*
Faculty of Engineering, Chulalongkorn University

Grade Letters

[80,100] → **A**

[75,80) → **B+**

[70,75) → **B**

[65,70) → **C+**

[60,65) → **C**

[55,60) → **D+**

[50,55) → **D**

[0,50) → **F**

2190102 Advanced Computer Programming : *Atiwong Suchato*
Faculty of Engineering, Chulalongkorn University

Final Project

2011/2

ICE IMP
(Photoshop)

2012/2

APP PRO

ICE World
(The Sims)

2013/2

ICE Card War
(Yu-Gi-Oh)

Topic 1

- Object-Oriented Programming Concepts
  - OOP in Java
  - UML Class Diagram
- Using Eclipse IDE
  - Debugging in Eclipse
- Java Applet

Object-Oriented Programming (OOP)

and additional issues on Java

OOP

- OOP is a programming paradigm that uses *objects* and their interactions to design computer programs.
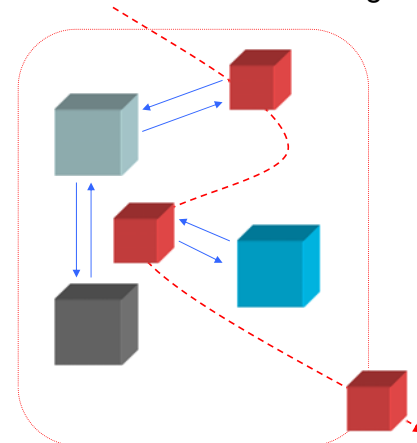
A program is a result of cooperating *objects*.

Each object is capable of:
- Receiving message
- Processing data
- Sending message

A List of Languages with OOP Features

ABAP	FPr	Oberon (Oberon-1)	Lua	Snit
Ada 95	FreeBASIC	Oberon-2	Lisaac	incr Tcl
AmigaE	F-Script	Object Pascal	MOO	Ubercode
BETA	F#	Delphi	NewtonScript	Vala
Blue	Gambas	Free Pascal	Obliq	Visual Basic
Boo	Graphtalk	Turbo Pascal	REBOL	VB.NET
C++	IDLscript	Object REXX	Self	VBScript
C#	J	Objective-C	Python	VBA
Chapel	J#	OCaml	REALbasic	Visual FoxPro
Clarion	JADE	Omnis Studio	Revolution	Visual Prolog
CLU	Java	Oz	Ruby	XBase++
COBOL	Groovy	Mozart Programming System	S	ZZT-oop
Cobra	Join Java	Perl since v5	R	
ColdFusion	X10	PHP5	Scala	
Common Lisp	Lasso	Power Builder	Seed7	
COOL	Lava	Prototype-based languages	SenseTalk	
CorbaScript	Lexico	ABCL/1, ABCL/R, ABCL/R2, ABCL/c+	Simula	
Curl	Lingo	Agora	Smalltalk	
D	LISP	Cecil	Self	
Dylan	Logtalk	Cel	Bistro	
E	MATLAB	ECMAScript	Squeak	
Eiffel	Modula-2	ActionScript	Squirrel	
Sather	Modula-3	JavaScript	Superx++	
Falcon	Nemerle	JScript	TADS	
Fancy	NetRexx	Etoys (in Squeak)	Tcl	
Fortran 2003	Noop	Io	Xotcl (similar to CLOS)	

http://en.wikipedia.org/wiki/List_of_object-oriented_programming_languages

---

Objects and Classes

	Interpretation in the real world	Representation in computer programs
Object	An *Object* represents anything in the real world that can be distinctly identified.	An *Object* has an identity, a state, and a behavior. **Actual Data**
Class	A *Class* represents a set of objects with similar characteristics and behavior. These objects are called *instances* of the class.	A Class characterizes the structure of states and behaviors that are shared by all its instances. **Data Type**

Objects and Classes

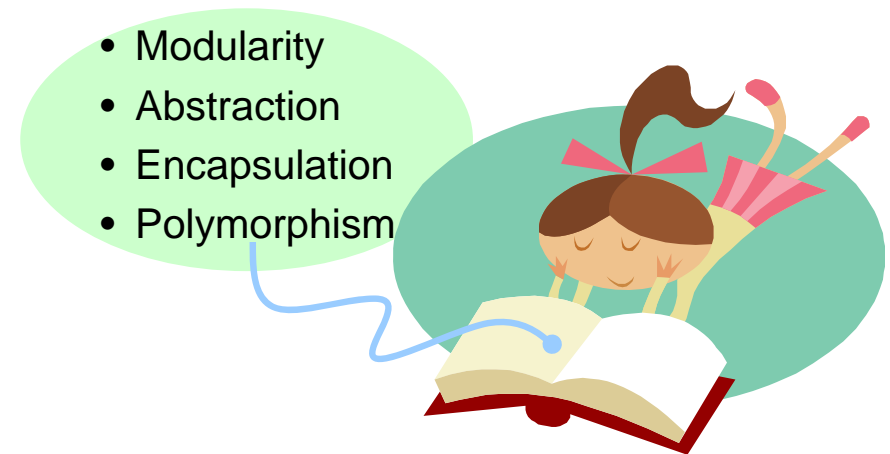|	Representation in computer programs
Object	An *Object* has an identity, a state, and a behavior.
Class	A Class characterizes the structure of states and behaviors that are shared by all its instances.

method

field or *attribute*

Important Concepts

- Modularity
- Abstraction
- Encapsulation
- Polymorphism

Modularity

A complex software system should be decomposed into a set of *highly cohesive* but *loosely coupled* modules.

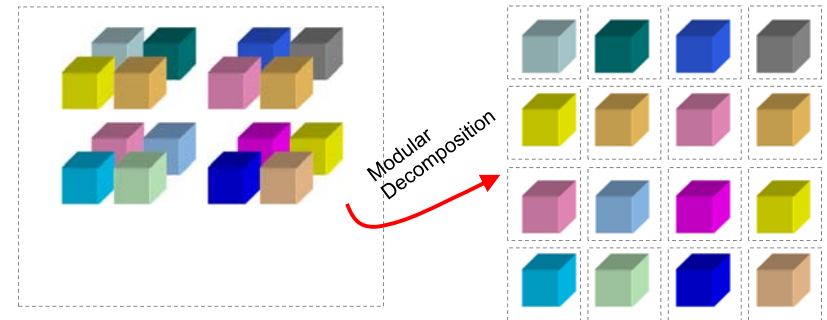Cohesion → Functional relatedness of the entities within a module

Coupling → Interdependency among different modules

Modularity

Complex system consisting of a large number of functionalities

→ Relatively small & simple modules (Highly cohesive)

→ Interactions among modules are simple (loosely coupled)

Modularity

packages

class
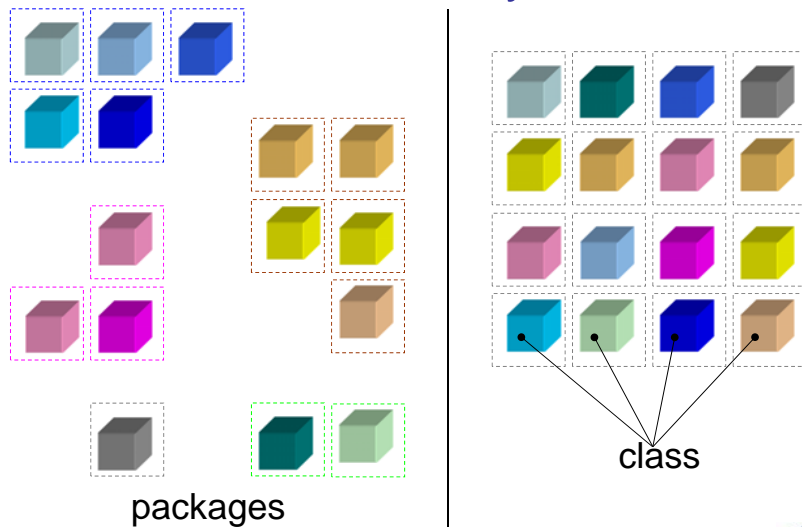
Abstraction & Encapsulation

Separating the essential from the nonessential characteristics
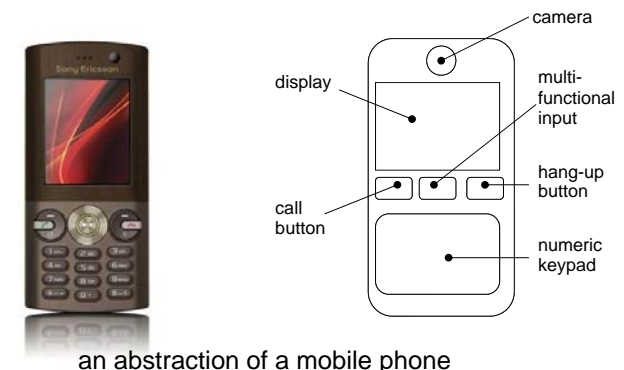
camera

display

multi-functional input

hang-up button

call button

numeric keypad

an abstraction of a mobile phone

Abstraction & Encapsulation

The implementation of a module should be separated from its *contractual interface* and hidden from the clients of the module
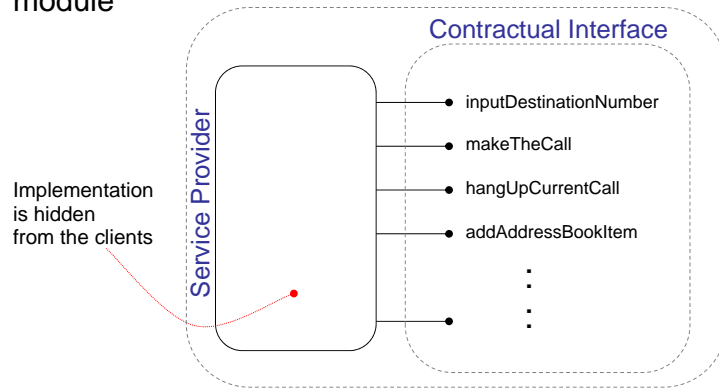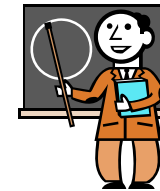
Contractual Interface

Service Provider

Implementation is hidden from the clients

- inputDestinationNumber
- makeTheCall
- hangUpCurrentCall
- addAddressBookItem

Polymorphism

- Several service providers can honor the same contractual interface.
- These service providers can be interchanged without effecting the clients.

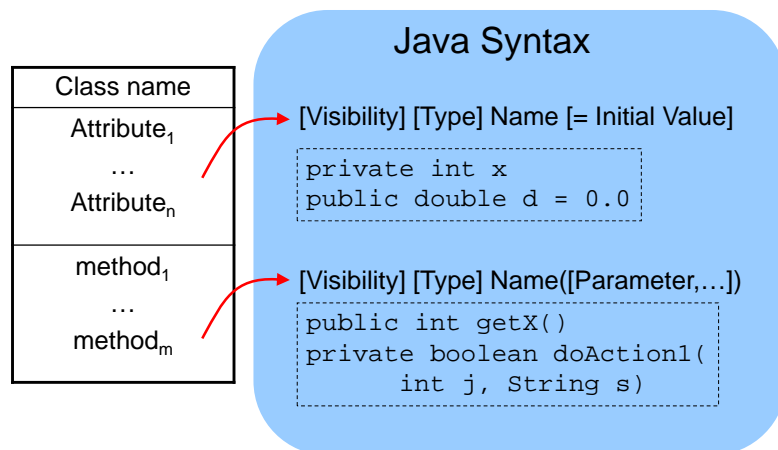The ability to do this dynamically is called
"Polymorphism".

UML Notation for Classes

Class name
Attribute$_1$ …   Attribute$_n$
method$_1$ …   method$_m$

Java Syntax

[Visibility] [Type] Name [= Initial Value]

```
private int x
public double d = 0.0
```

[Visibility] [Type] Name([Parameter,…])

```
public int getX()
private boolean doAction1(
        int j, String s)
```

UML stands for **U**nified **M**odeling **L**anguage.

UML Notation for Classes

Class name
Attribute$_1$ …   Attribute$_n$
method$_1$ …   method$_m$

UML Syntax

[Visibility] Name [:Type] [= Initial Value]

```
-x :int
+d :double = 0.0
```

[Visibility] Name([Parameter,…]) [:Type]

```
+getX():int
-doAction1(j:int,
          s:String):boolean
```

Visibility	Java	UML
public	public	+
protected	protected	#
package		~
private	private	-
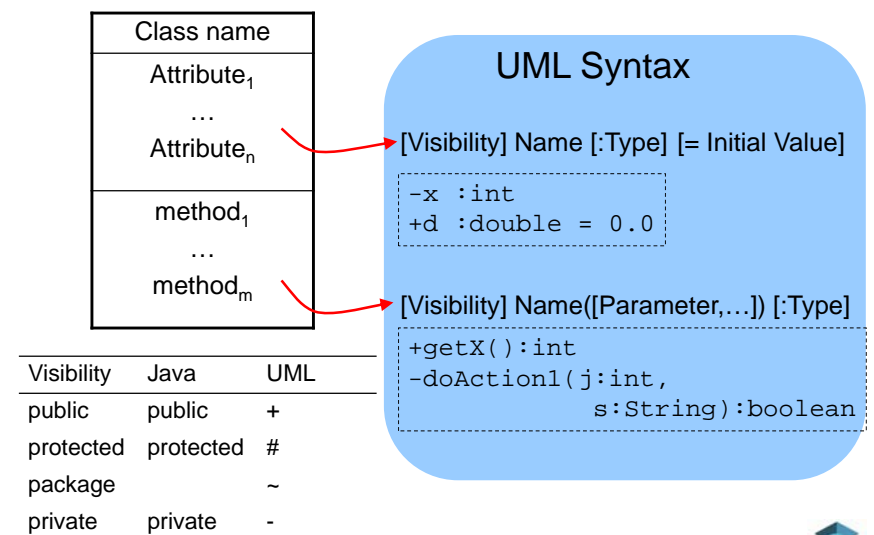
Java Class Definition

```
public class Point
{
    private int x;            ← Attributes
    private int y;

    public Point(){           ← Methods
        x = 0; y = 0;
    }
    public int getX(){
        return x;
    }                         ← Accessors
    public int getY(){
        return y;
    }
    public void setX(int x){
        this.x = x;
    }                         ← Mutators
    public void setY(int y){
        this.y = y;
    }
}
```

UML Notation for Objects
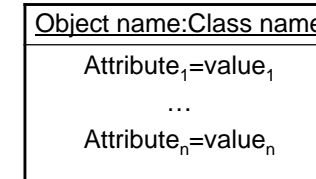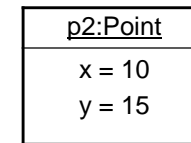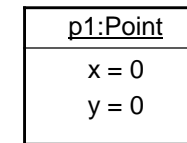
Object name:Class name
$Attribute_1 = value_1$
…
$Attribute_n = value_n$

Example:

p1:Point		p2:Point
x = 0		x = 10
y = 0		y = 15

UML stands for **U**nified **M**odeling **L**anguage.

Convention for Class and Object Names

Style Convention:

- Class names should begin with uppercase letters.
- Attribute and method names should begin with lowercase letters.
- The first word in method names should be verbs.
- If a name consists of multiple words, it is formed by concatenating the words and capitalizing each word except the first.
- Attributes whose values are constant should be name using all uppercase letters. If there are multiple words, use underscores (_) to separate words.
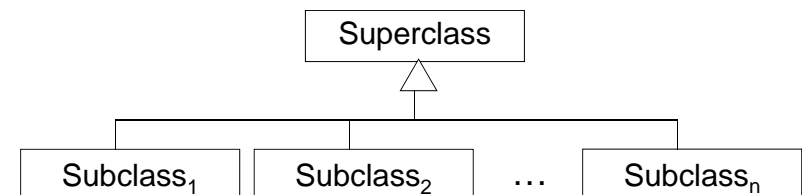

Inheritance

- The *extension* relation between two classes.
- When class *C2* extends class *C1*, class *C2* is known as a *subclass* of class *C1*. Class *C1* is known as a *superclass* of class *C2*.
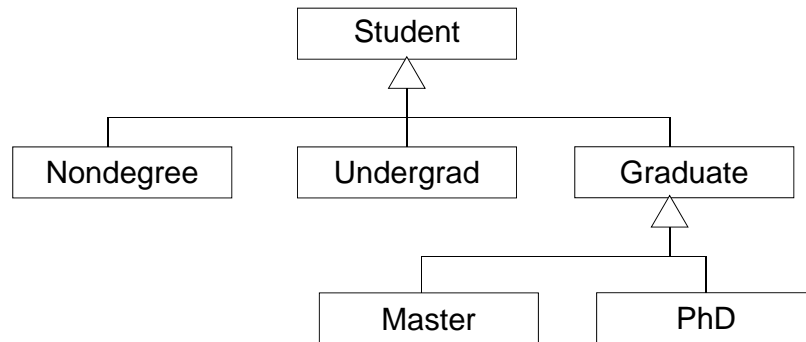
UML Notation: extension of a class

Superclass

Subclass$_1$	Subclass$_2$	…	Subclass$_n$

Inheritance

```
        Student
       /   |   \
Nondegree Undergrad Graduate
                   /    \
              Master    PhD
```

Inheritance model is the *is-a* (or *is-an*) relationship in the real world.

The Tools

http://www.eclipse.org/downloads/

Version Name ⬍	Date ⬍	Platform Version ⬍	Projects ⬍
Austin	21 June 2004	3.0[14]	
N/A	28 June 2005	3.1	
Callisto	30 June 2006	3.2	Callisto projects[15]
Europa	29 June 2007	3.3	Europa projects[16]
Ganymede	25 June 2008	3.4	Ganymede projects[17]
Galileo	24 June 2009	3.5	Galileo projects[18]
Helios	23 June 2010	3.6	Helios projects[19]
Indigo	22 June 2011	3.7	Indigo projects[20]
Juno	27 June 2012	3.8 and 4.2[21] [Notes 1]	Juno projects[24]
Kepler	26 June 2013	4.3	Kepler projects[25]
Luna	25 June 2014	**4.4**	Luna projects[26]
Mars	24 June 2015 (planned)	4.5	Mars projects[27]

🟥 Old version   🟨 Older version, still supported   🟩 **Latest version**   🟦 Future release

---

Running / Debugging

Run as Java Application
Debug as Java Application
Perspective
Breakpoints
Variable watch
Step into / Step over

HelloMoon.java

Java Applets

a brief look

Java Applets

- An *application* is a stand-alone program.
- An *applet* is a program that can be embedded in an HTML page.
- A Java program can be designed to function as:
  - an application
  - an applet
  - both an application and an applet

Java Applets

- is not a stand-alone program so it does not need *main()*.
- must extend the *Applet* class.
- An applet is invoked in an *applet context*. (i.e. a *browser* or an *applet viewer*)

Are Java Applets wrong choices for today's development ?

?

Java Applets are convenient means to start developing software based on "Framework"

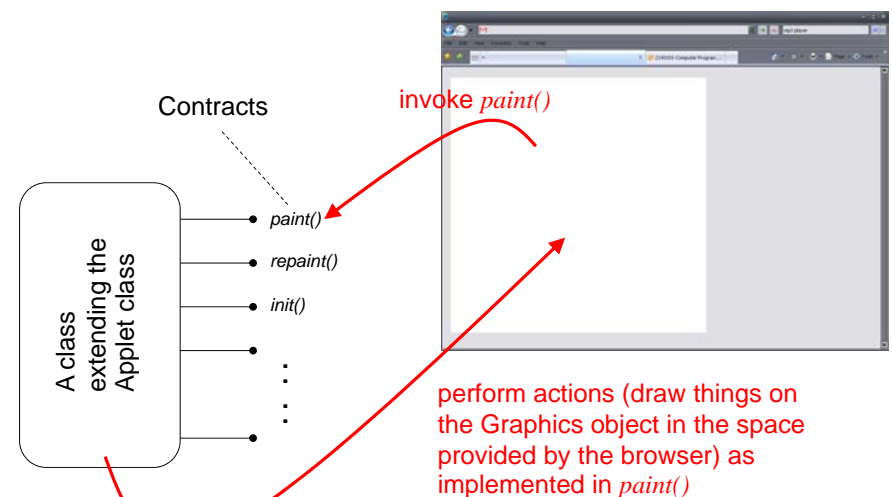# Java Applets

Contracts

invoke *paint()*

A class extending the Applet class

- *paint()*
- *repaint()*
- *init()*
- •
  .
  .
- •

perform actions (draw things on the Graphics object in the space provided by the browser) as implemented in *paint()*

Anatomy of an Applet

HelloWorldApplet.java

```
import java.awt.Graphics;
import java.applet.Applet;
public class HelloWorldApplet extends Applet
{
 public void paint(Graphics g){
  g.drawString("Hello World",100,100);
 }
}
```

The class must extends *Applet*.
Also, *Applet* has to be imported.

HelloWorldApplet.java

Anatomy of an Applet

HelloWorldApplet.java

```
import java.awt.Graphics;
import java.applet.Applet;
public class HelloWorldApplet extends Applet
{
 public void paint(Graphics g){
  g.drawString("Hello World",100,100);
 }
}
```

The class must implement *paint()*

HelloWorldApplet.java

Anatomy of an Applet

HelloWorldApplet.java

```
import java.awt.Graphics;
import java.applet.Applet;
public class HelloWorldApplet extends Applet
{
 public void paint(Graphics g){
  g.drawString("Hello World",100,100);
 }
}
```

A *Graphics* object is passed to *paint()*.
This object can be thought as the canvas
to be painted on.

HelloWorldApplet.java

Anatomy of an Applet

HelloWorldApplet.java

```
import java.awt.Graphics;
import java.applet.Applet;
public class HelloWorldApplet extends Applet
{
 public void paint(Graphics g){
  g.drawString("Hello World",100,100);
 }
}
```
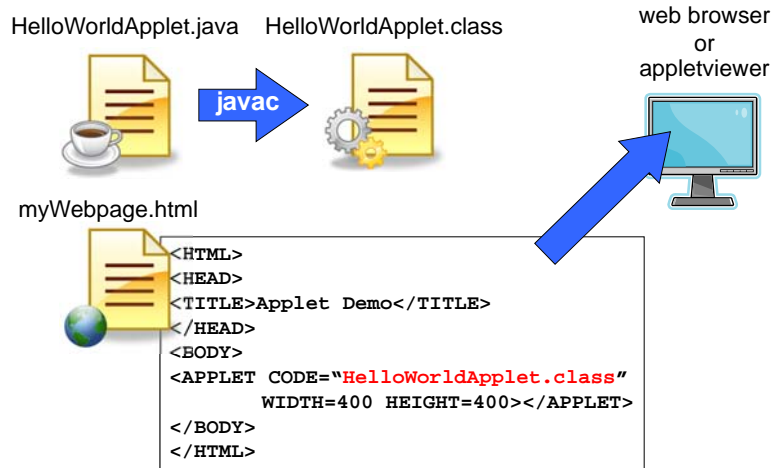
drawString() is an object method defined in *Graphics*.
It put the specified *String* on to the *Graphics* object that the
method is invoked from.

Embedding an Applet

HelloWorldApplet.java → HelloWorldApplet.class → web browser or appletviewer

javac

myWebpage.html

```
<HTML>
<HEAD>
<TITLE>Applet Demo</TITLE>
</HEAD>
<BODY>
<APPLET CODE="HelloWorldApplet.class"
        WIDTH=400 HEIGHT=400></APPLET>
</BODY>
</HTML>
```

Applet Viewer: HelloWorldApplet.class
Applet

Hello World

Applet started.

Drawing Example

(10,10) (20,20)

40 60

40

60

```
import java.awt.*;
import java.applet.Applet;
public class DrawingDemo1 extends Applet
{       public void paint(Graphics g)
        {       g.setColor(Color.MAGENTA);
                g.fillRect(10,10,60,60);
                g.setColor(Color.BLUE);
                g.fillRect(20,20,40,40);
                g.setColor(Color.YELLOW);
                g.fillOval(20,20,40,40);

        }
}
```

Applet
Applet started.

DrawingDemo.java

Drawing Example

(0,0) always centered

40

40

width of the applet

height of the applet

```
import java.awt.*;
import java.applet.Applet;
public class DrawingDemo2 extends Applet
{   public void paint(Graphics g){
        Dimension d = this.getSize();
        g.setColor(Color.MAGENTA);
        g.fillRect(0,0,d.width,d.height);
        g.setColor(Color.BLUE);
        g.fillRect(d.width/2-20,d.height/2-20,
                   40,40);
        g.setColor(Color.YELLOW);
        g.fillOval(d.width/2-20,d.height/2-20,
                   40,40);

    }
}
```

Applet Viewer: DrawingDem...
Applet
Applet started.

Applet View...
Applet
Applet started.

DrawingDemo.java
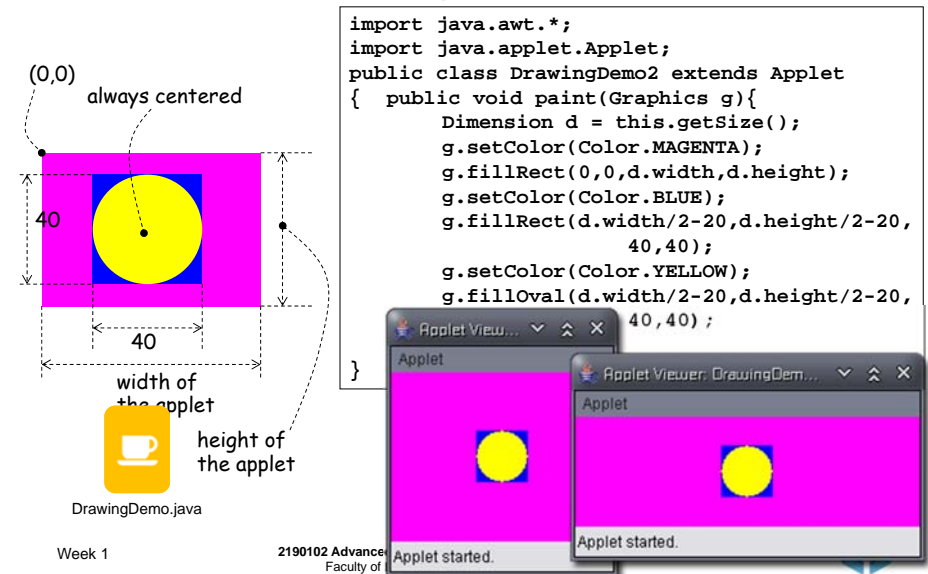
The *paint( )* Method

> Try minimize the applet, and restore again. Observe the change.

```java
import java.awt.*;
import java.applet.Applet;
public class PaintInvokeDemo extends Applet
{
    public void paint(Graphics g){
        Dimension d =this.getSize();
        g.setColor(pickRandomColor());
        g.fillRect(0,0,d.width,d.height);
    }
    private Color pickRandomColor(){
        float r =(float)Math.random();
        float g =(float)Math.random();
        float b =(float)Math.random();
        return new Color(r,g,b);
    }
}
```

PaintInvokeDemo.java
Week 1

Self-Study for Topic 1

A: **Trail: Learning the Java Language:**

http://docs.oracle.com/javase/tutorial/java/TOC.html

Read all pages under "Object-Oriented Programming Concepts".

Read all pages under "Packages".

Read the page titled "Enum Types".

Find a section on how to define and use Java methods when the number of input arguments is unknown at compile-time and read that section.

Week 1

Self-Study for Topic 1

B: **Java – Applet Basics:**

http://www.tutorialspoint.com/java/java_applet_basics.htm

Read the page from the beginning up to the section titled "Specifying Applet Parameters:"

Week 1

Self-Study Test: Topic 1

The test must be done during:

Saturday 16 August
to Monday   18 August

in the "Assessment" section of

my CourseVille

Week 1