

# Structural Programming and Data Structures

Winter 2000

## CMPUT 102: Introduction

Dr. Osmar R. Zaiane



University of Alberta

## Class and Office Hours

### Class:

Mondays, Wednesdays and Fridays from 14:00 to 14:50

### Office Hours:

Tuesdays and Thursdays from 11:00 to 11:45

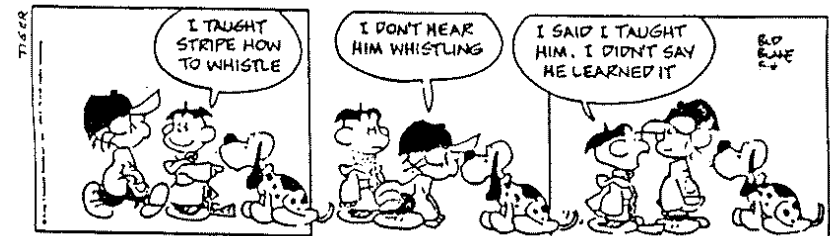
By appointment:

E-mail [zaiane@cs.ualberta.ca](mailto:zaiane@cs.ualberta.ca)

Tel: 492 7569

## Course Requirements

- Pure Math 30 and CS 30 or equivalent. A basic knowledge of computer programming is required. Students should understand variables, assignment, arithmetic expressions, if statements and loops. Students who do not have a basic knowledge of computer programming should enrol in CMPUT 101.
- There are two routes that can be followed to take Computing Science courses, even for students who want to specialize in Computing Science:



## Course Objectives

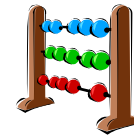
The objects of the course are to introduce the students to the issues of Computer Science problem solving by writing computer programs in a high-level programming language called Java. Students are introduced to concepts and simple algorithms and techniques for constructing elegant and robust solutions to problems.



After completing the course, students should understand and use the concepts: object, primitive value, message, method, selection control structure (if, switch), repetition control structures (while, for), variable, object reference, method parameter, container (Arrays, Vectors, Stacks etc.), searching, sorting, recursion and inheritance.

## Evaluation and Grading

Your final grade will depend on the entire profile of the grades in your lecture section and a particular composite score does not guarantee a particular final grade. However, your composite score will be computed using the following weights:



• Lab Exercises	10%
• Assignment 1	5%
• Assignment 2	5%
• Lab Examination	10%
• Term Examination 1	15%
• Term Examination 2	20%
• Final Examination	35%

## More About Evaluation

### Re-examination.

None, except as per regulation.

### Collaboration.

Collaborate on assignments; do not merely copy.



### Plagiarism.

Work submitted by a student that is the work of another student or a tutor is considered plagiarism. Read **Sections 26.1.4 and 26.1.5** of the University of Alberta calendar. Cases of plagiarism are immediately referred to the Dean of Science, who determines what course of action is appropriate.

## Notes and Textbook

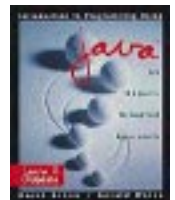


### Course home page:

<http://www.cs.ualberta.ca/~zaiane/courses/cmput102/>

### Textbook:

Introduction to Programming Using Java:  
An Object-Oriented Approach, Java 2 Update  
by David Arnow and Gerald Weiss.  
Addison Wesley, 2000





## Legend of Link Icons

- Link to an HTML page (default)
- Link to a text file
- Link to a page to be displayed in a new browser window
- Link to a Portable Document Format (PDF) file
- Link to a Postscript (PS) file
- Link to slides presentation
- Link to a compressed (gz or zip) file
- Link to an image
- Link to a video
- Link to a C/C++ program listing
- Link to a Java class file
- Link to a data file
- Link to password protected html page
- Link to a page under construction

## On-line Resources



- CMPUT 114/102 web page
- Course slides
- Web links
- Glossary
- Student submitted resources
- Student spaces
- U-Chat
- Frequently asked questions



## Course Schedule

(Tentative, subject to changes)

There are 14 weeks from January 10<sup>th</sup> to April 12<sup>th</sup>.

Week 7 is reading week (Feb. 21-25)

There are 9 lab exercises (**one each week**)

One lab exam **week 11**

Assignment 1 distribution **week 5**

Assignment 1 due **week 9**

Assignment 2 distribution **week 8**

Assignment 2 due **week 12**

Midterm 1 **week 6**

Midterm 2 **week 10**

**Final Exam Week 15 (April 19)**

## Course Content

- |  |   |
|--|---|
| <ul style="list-style-type: none"> <li>• Introduction</li> <li>• Objects</li> <li>• Methods</li> <li>• Tracing Programs</li> <li>• Object State</li> <li>• Sharing resources</li> <li>• Selection</li> <li>• Repetition</li> </ul> | <ul style="list-style-type: none"> <li>• Vectors</li> <li>• Testing/Debugging</li> <li>• Arrays</li> <li>• Searching</li> <li>• Files I/O</li> <li>• Sorting</li> <li>• Inheritance</li> <li>• Recursion</li> </ul> |
|--|---|



## Quick Tour of the Course Web Site



<http://www.cs.ualberta.ca/~zaiane/courses/cmp102/>

## Course Content



- |   |   |
|---|---|
| <ul style="list-style-type: none"> <li>• <b>Introduction</b></li> <li>• Objects</li> <li>• Methods</li> <li>• Tracing Programs</li> <li>• Object State</li> <li>• Sharing resources</li> <li>• Selection</li> <li>• Repetition</li> </ul> | <ul style="list-style-type: none"> <li>• Vectors</li> <li>• Testing/Debugging</li> <li>• Arrays</li> <li>• Searching</li> <li>• Files I/O</li> <li>• Sorting</li> <li>• Inheritance</li> <li>• Recursion</li> </ul> |
|---|---|



**Lecture 1 – Lecture 2 – Lecture 3**

# Objectives of Lecture 1

## A Simple Java Application

- Get a rough initial idea about the programming process by:
  - Creating and running a simple Java application.
  - Using a programming environment to create and run the application.
- There is no need to try to understand any of the code.

# Outline of Lecture 1



- Kinds of Java programs
- Run Adventure Version 8
- Code Warrior application demonstration
- Review - (Edit, Compile)+, Load, Run
- Application template
- Write Adventure Version 0

# Kinds of Java Programs

- There are three kinds of Java programs:
  - Applications
  - Applets
  - Libraries
- An **application** is a Java program that is run by using a Java interpreter program.
- An **applet** is a Java program that is run by a Java-enabled web browser.
- A **library** is a set of Java classes that can be used by another Java program.

# Outline of Lecture 1



- Kinds of Java programs
- Run Adventure Version 8
- Code Warrior application demonstration
- Review - (Edit, Compile)+, Load, Run
- Application template
- Write Adventure Version 0

# The Adventure Application

- We will write an application called the Arithmetic Adventure game in these lectures.
- We will use it as a running example throughout the course, slowly adding functionality to it.



# Try Adventure Version 8

- Start Code Warrior.
- Open an existing project called Adventure8.
- Run.

# Outline of Lecture 1

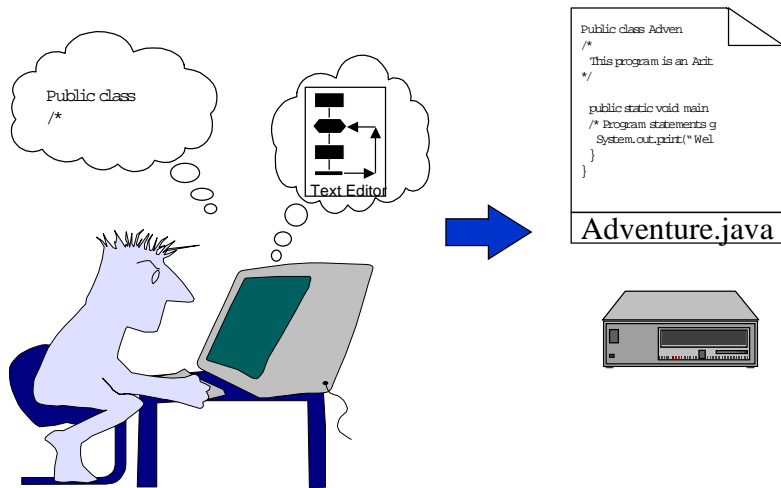


- Kinds of Java programs
- Run Adventure Version 8
- Code Warrior application demonstration
- Review - (Edit, Compile)+, Load, Run
- Application template
- Write Adventure Version 0

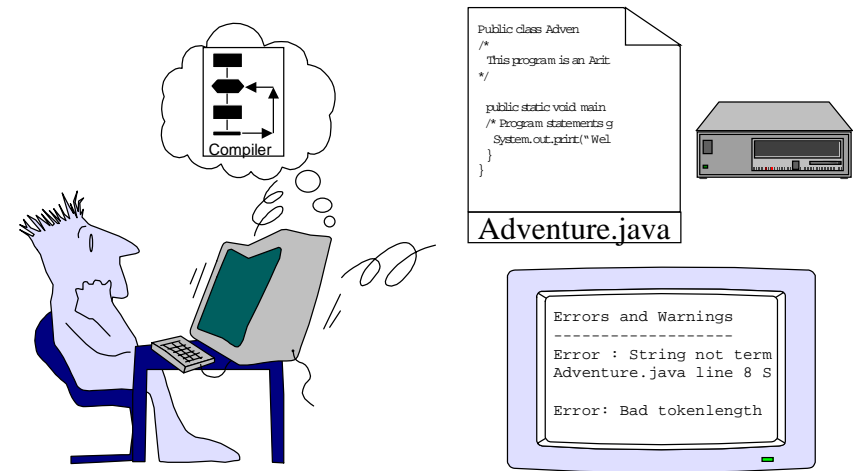
# Demonstration Application

- Start Code Warrior.
- Create a new project called Adventure0.
- Open the java source file.
- Edit the code.
- Save as... Adventure.java
- Java Application Settings - Java Target.
- Make.
- Run.
- Demonstrate a compilation error.

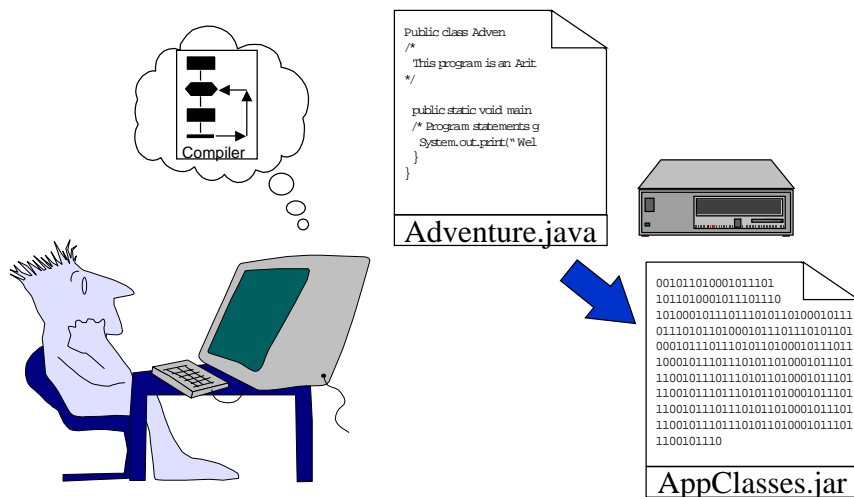
## Review - Editing



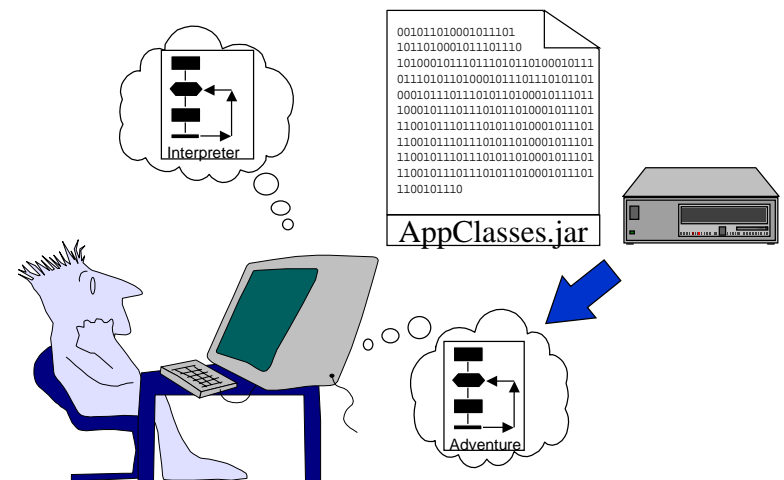
## Review - Compiling (Error)



## Review - Compiling (Success)

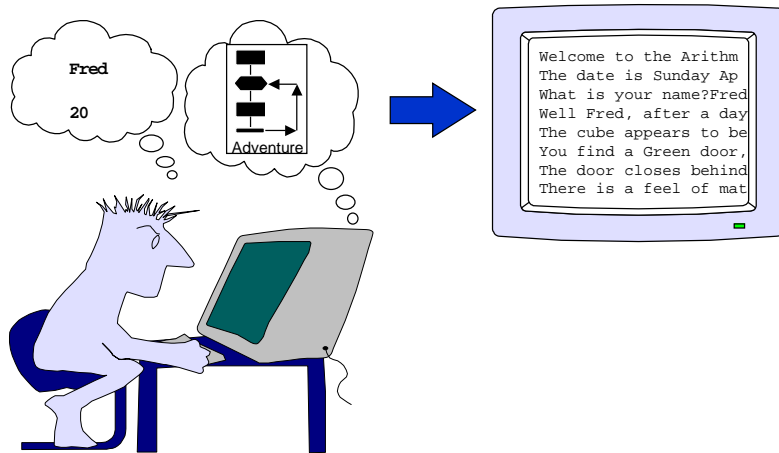


## Review - Loading





## Review - Run



## Outline of Lecture 1



- Kinds of Java programs
- Run Adventure Version 8
- Code Warrior application demonstration
- Review - (Edit, Compile)+, Load, Run
- Application template
- Write Adventure Version 0

## Java Application Template

```
public class put program name here {  
    /*  
        Program description.  
    */  
    public static void main(String args[]) {  
        /* Program statements go here. */  
        put statements here  
    }  
}
```

## Outline of Lecture 1



- Kinds of Java programs
- Run Adventure Version 8
- Code Warrior application demonstration
- Review - (Edit, Compile)+, Load, Run
- Application template
- Write Adventure Version 0



## The Adventure Application

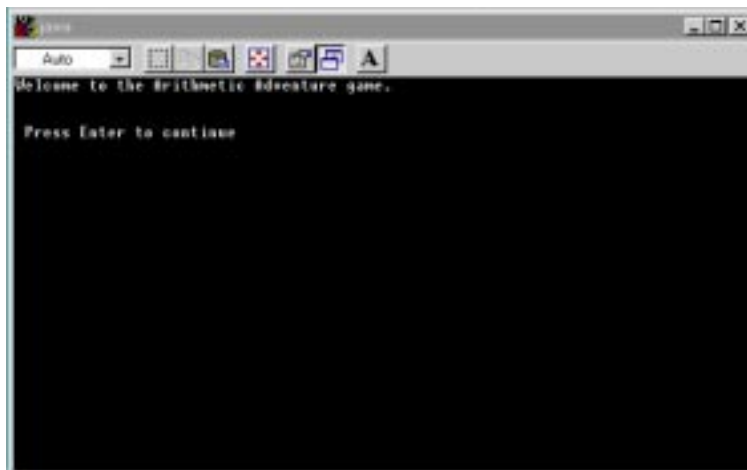
- In version zero, it just displays a greeting on the screen, just as our demonstration application did.



## Program - Adventure V0

```
public class Adventure {  
    /* Version 0  
    This program is an arithmetic adventure game  
    where an adventurer navigates rooms that  
    contain treasure chests that are opened by  
    correctly answering arithmetic problems.  
    */  
    public static void main(String args[]) {  
        /* Program statements go here. */  
        System.out.print("Welcome to the Arithmetic Adventure  
game.");  
    }  
}
```

## Adventure V0 Output



## Initial Programming Strategy

- For now we will use a fixed application template without trying to understand the details.
- We can create a new application by changing the program name and the statements that the template contains.

# Course Content



## • Introduction

- Objects
- Methods
- Tracing Programs
- Object State
- Sharing resources
- Selection
- Repetition
- Vectors
- Testing/Debugging
- Arrays
- Searching
- Files I/O
- Sorting
- Inheritance
- Recursion



Lecture 1 – **Lecture 2** – Lecture 3

## Objectives of Lecture 2

### A Simple Java Applet

- Get a rough initial idea about the programming process by:
  - Creating and running a simple Java applet.
  - Using a programming environment to create and run the applet.
- There is no need to try to understand any of the code.

## Outline of Lecture 2



- Kinds of Java programs
- Code Warrior applet demonstration
- Applet template
- Applet0
- HTML

## Kinds of Java Programs

- There are three kinds of Java programs:
  - Applications
  - Applets
  - Libraries
- An **application** is a Java program that is run by using a Java interpreter program.
- An **applet** is a Java program that is run by a Java-enabled web browser.
- A **library** is a set of Java classes that can be used by another Java program.

## Outline of Lecture 2



- Kinds of Java programs
- Code Warrior applet demonstration
- Applet template
- Applet0
- HTML

## Demonstration Applet

- Start Code Warrior.
- Create a new project called Applet0.
- Open the java source file.
- Edit the code.
- Save as... Applet0.java
- Java Applet Settings - Java Target.
- Edit the html.
- Make.
- Run from environment and browser.

## Outline of Lecture 2



- Kinds of Java programs
- Code Warrior applet demonstration
- Applet template
- Applet0
- HTML

## Java Applet Template

```
import java.awt.*;
import java.applet.*;
public class      put applet subclass name here
/*
   Applet description.
*/

public void paint(Graphics graphics) {
    /* applet display statements go here. */

    put display statements here
}
```

## Outline of Lecture 2



- Kinds of Java programs
- Code Warrior applet demonstration
- Applet template
- Applet0
- HTML

## Applet Applet0

```
import java.awt.*;
import java.applet.*;
public class Applet0 extends Applet {
/*
    This is our first applet.
*/

    public void paint(Graphics graphics) {
        /* Display myself on the given Graphics */
        graphics.drawString("This is an applet.", 10, 50);
    }
}
```

x and y coordinates of upper left corner of the String

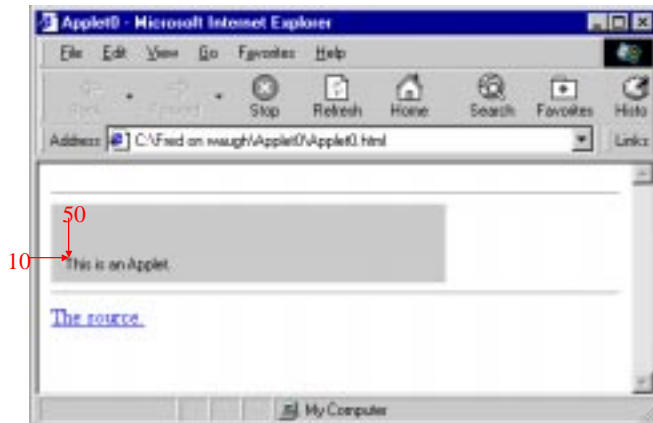
## HTML Documents

- A web browser runs an Applet by first loading a document written in a language called HTML (HyperText Markup Language).
- HTML defines the meanings of tags that are added to document text, which specify the format of the document and links to other documents.
- One of the tags in the HTML file tells the browser to load an applet.

## Applet0.html

```
<HTML>
<HEAD>
    <TITLE>Applet0</TITLE>
</HEAD>
<BODY>
    <HR>
    <APPLET
        ARCHIVE="JavaClasses.jar"
        CODE="Applet0.class"
        WIDTH=300 HEIGHT=60>
    </APPLET>
    <HR>
    <A HREF="Applet0.java">The source.</A>
</BODY>
</HTML>
```

## Applet0 Output



## Outline of Lecture 2



- Kinds of Java programs
- Code Warrior applet demonstration
- Applet template
- Applet0
- HTML

## Some HTML

```
<HTML>
<HEAD>
  <TITLE>title text here</TITLE>
</HEAD>

<BODY BGCOLOR="#00FF00">

  Text and tags come here

</BODY>
</HTML>
```



## Some HTML – con't

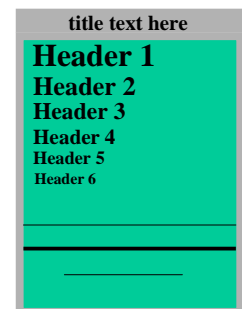
Some useful tags:

### Headers:

```
<H1>Header 1</H1>
<H2>Header 2</H2>
<H3>Header 3</H3>
<H4>Header 4</H4>
<H5>Header 5</H5>
<H6>Header 6</H6>
```

### Horizontal line:

```
<HR>
<HR SIZE=4>
<HR WIDTH=50%>
```



## Some HTML – con't

Some useful tags:

### Line breaks and Paragraphs:

Line 1 and <BR>Line 2

<P>Paragraph</P>

### Images:

<IMG SRC=MyFile.gif WIDTH=100 HEIGHT=200>

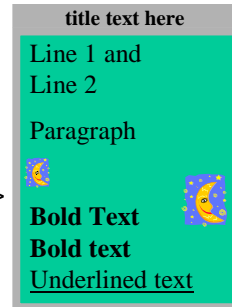
<IMG SRC=MyFile.gif ALIGN=right>

### Bold Text and Underlined Text:

<B>Bold Text</B>

<STRONG>Bold text</STRONG>

<U>Underlined text</U>



## Some HTML – con't

Some useful tags:

### Hyperlinks:

<A HREF=http://www.ualberta.ca>This a link to UofA</a>

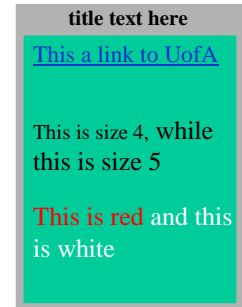
### Font sizes and Colours:

<FONT SIZE=4>this is size 4,</FONT>

<FONT SIZE=5>while this is size 5</FONT>

<FONT COLOR=red>This is red</FONT>

<FONT COLOR=white>and this is white</FONT>



## Some HTML – con't

Some useful tags:

### Ordered lists:

<OL>

<LI>element 1

<LI>element 2

<LI>element 3

</OL>

### Unordered lists:

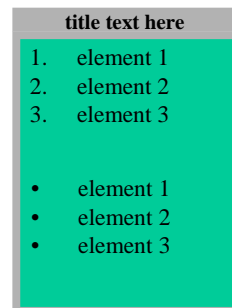
<UL>

<LI>element 1

<LI>element 2

<LI>element 3

</UL>



## Some HTML – con't

Some useful tags:

### Tables:

<TABLE BORDER=1 WIDTH=100%>

<TR>

<TH>Title1</TH><TH>Title 2</TH>

</TR>

<TR>

<TD>data 1</TD><TD>data 2</TD>

</TR>

<TR>

<TD>data 3</TD><TD>data 4</TD>

</TR>

</TABLE>

title text here	
Title 1	Title 2
data 1	data 2
data 3	data 4

# Course Content



## • Introduction

- Objects
- Methods
- Tracing Programs
- Object State
- Sharing resources
- Selection
- Repetition
- Vectors
- Testing/Debugging
- Arrays
- Searching
- Files I/O
- Sorting
- Inheritance
- Recursion



Lecture 1 – Lecture 2 – **Lecture 3**

## Objectives of Lecture 3

### Computing Using Simple Messages

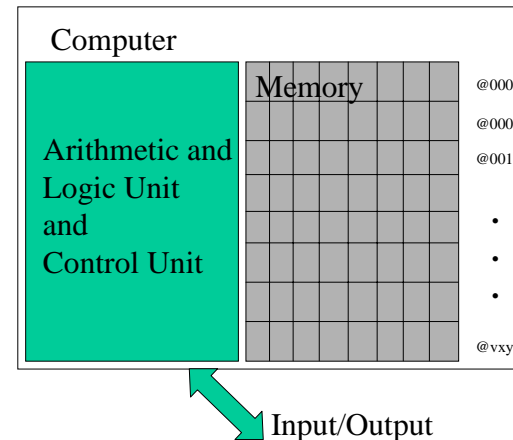
- Get an initial impression about the concepts of objects and message passing between objects.
- The Object-Oriented paradigm for solving computation problems will be briefly introduced.
- A model for representing and expressing computation will be put forward.

## Outline of Lecture 3



- Objects , messages, classes, instances and state.
- Message expressions that compute results - String examples
- State changes - Stack examples
- Message arguments - Stack and PrintStream examples
- Composed Messages - String > PrintStream examples

## Traditional Programs

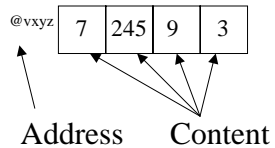


- Programs are usually sets of operations to load, swap, change, etc. content of memory units.
- The result of a program is typically the final content of some memory units.

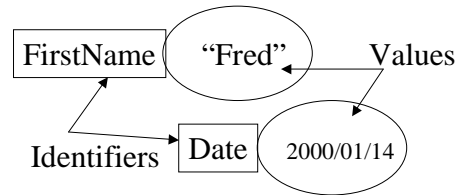


# Memory Units vs. Variables

## Memory Units



## Variables



- Variables are used to represent data items (i.e. store and refer to values) and are mapped into memory units.
- Primitive types are used to define variables: int, char, float, boolean...

# Procedural programming

Basic, Pascal, C, ...

- A program is a sequence of computer instructions.
  - Operations on values of variables, conditional statements, etc.
  - Problem is subdivided into sub-problems
    - ➔ Procedure to solve sub-problems.
- Procedures may be called with some parameters

We will see a new programming paradigm: Object-Oriented approach.  
Problems are solved by sending messages.

Java, C++, Smalltalk, Eiphel, ...

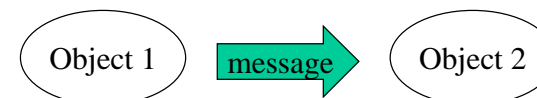
# Objects



- An **object** has a private state and a public message protocol.
- The **message protocol** of an object is the fixed set of all messages that can be sent to the object.
- The object that receives a message is called the **receiver** object.

# Messages

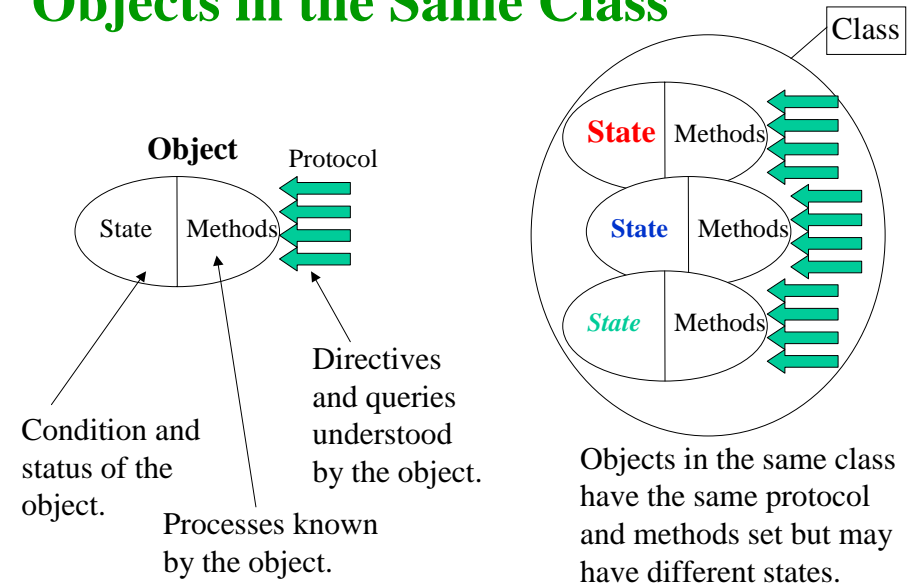
- A **message** is a request for the receiver object to do one or more of the following:
  - compute and return a result
  - change its state
  - send messages to other objects.



## Classes and Instances

- Objects that share the same message protocol are organized into a group called a **class** and each object in the class is called an **instance** of the class.
- The **state** of an object is what makes it different from other objects in the same class.
- For example, we might have two different instances of the class String, where one is “Fred” and the other is “Barney”.

## Objects in the Same Class



## Partial Protocol - String

- Some messages compute and return result objects.

toUpperCase

- Return a String that is the same as the receiver except all upper case.

trim

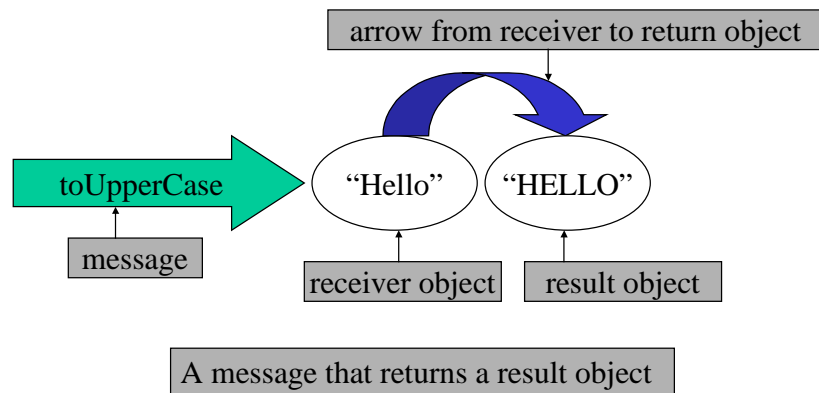
- Return a String that is the same as the receiver except with leading and trailing blanks removed.

## Outline of Lecture 3

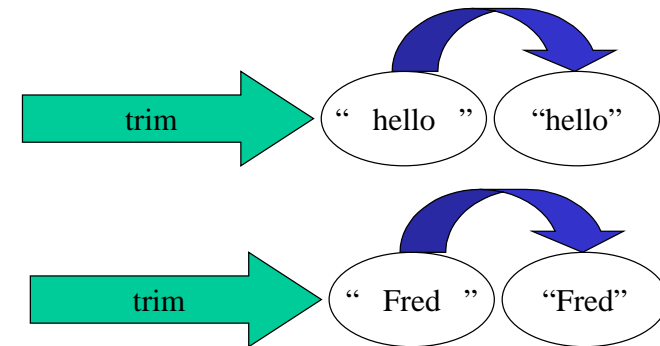


- Objects , messages, classes, instances and state.
- Message expressions that compute results - String examples
- State changes - Stack examples
- Message arguments - Stack and PrintStream examples
- Composed Messages - String > PrintStream examples

## String Example - toUpperCase



## String Example - trim



Different instances of the same class can respond differently because they have different state.

## Outline of Lecture 3



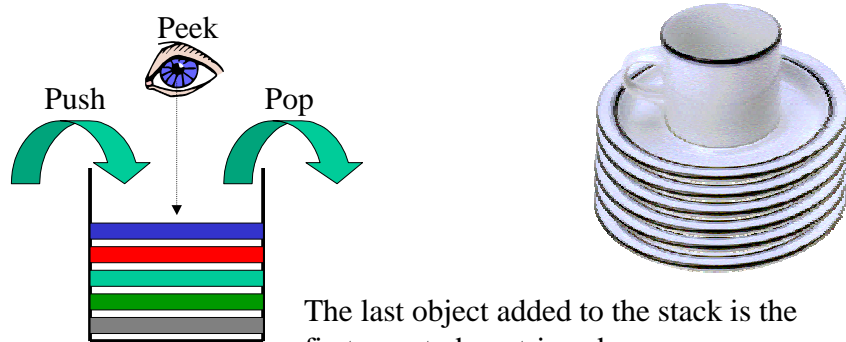
- Objects , messages, classes, instances and state.
- Message expressions that compute results - String examples
- State changes - Stack examples
- Message arguments - Stack and PrintStream examples
- Composed Messages - String > PrintStream examples

## State Changes

- An object that can change its state is called **mutable**.
- An object that cannot change its state is called **immutable**.
- If a message changes the state of an object, the change is called a **side-effect** of the message.
- A message to a mutable object may produce different results at different times.

# Stack

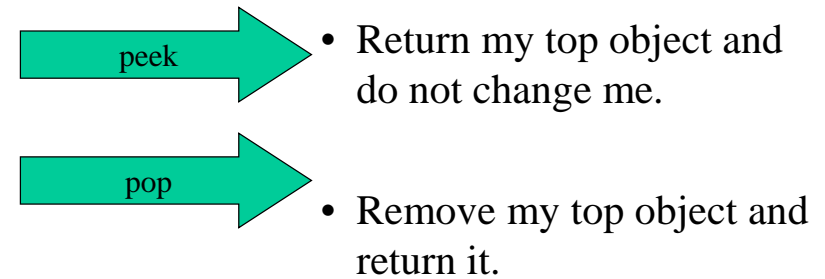
A stack is an object consisting of a pile of objects.



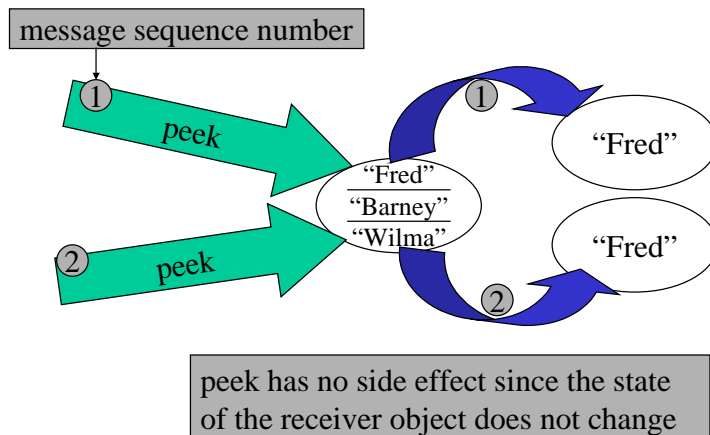
The last object added to the stack is the first one to be retrieved.  
LIFO: Last In First Out

# Partial Protocol for Stack

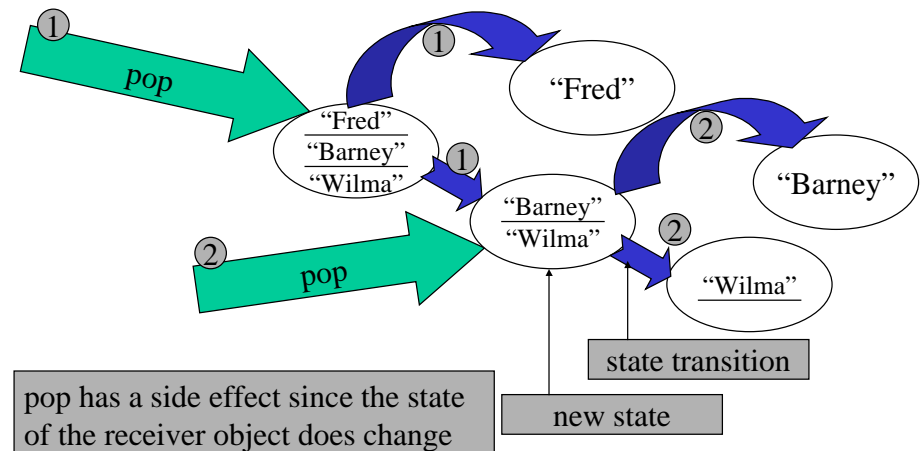
- Some messages change the state of the receiver object.



# Stack Example - peek



# Stack Example - pop



## Outline of Lecture 3



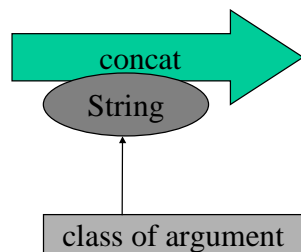
- Objects , messages, classes, instances and state.
- Message expressions that compute results - String examples
- State changes - Stack examples
- Message arguments - Stack and PrintStream examples
- Composed Messages - String > PrintStream examples

## Message Arguments

- Sometimes more than one object is needed to perform a computation.
- In these cases, the message to the receiver object contains one or more additional objects called **argument** objects.
- In some languages the class of the argument object must be specified as part of the message protocol.

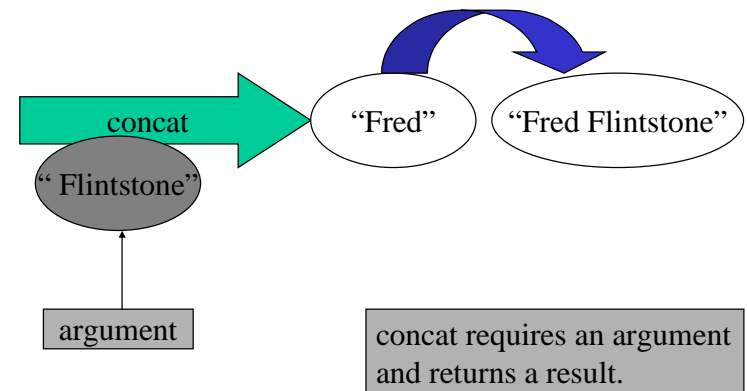
## More Protocol for String

- Some messages require arguments.



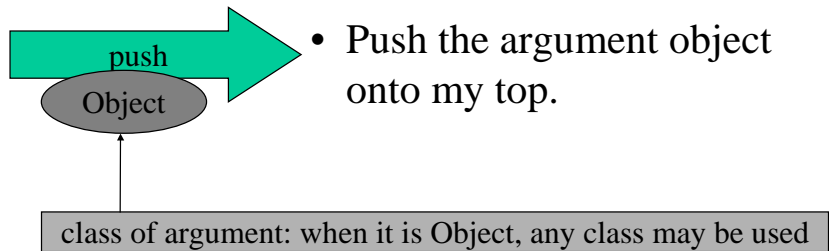
- Return a new String that is the result of concatenating the argument String to the end of me.

## String Example - concat



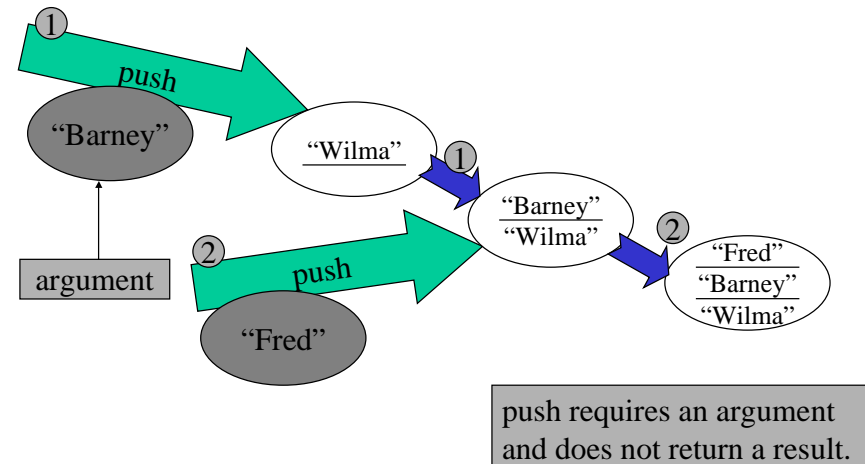
## More Protocol for Stack

- Some messages require arguments.
- Some messages do not return a result.



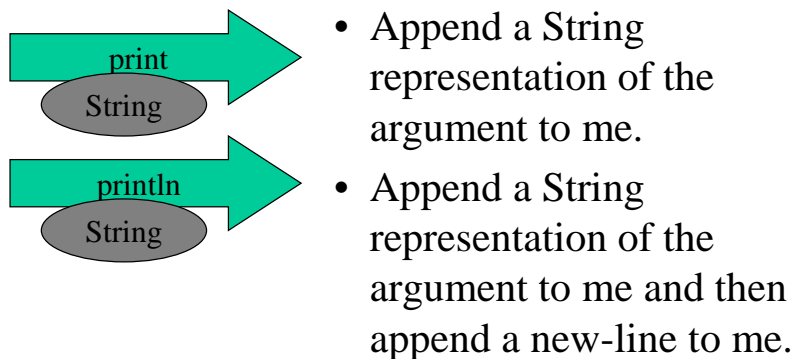
- Push the argument object onto my top.

## Stack Example - push

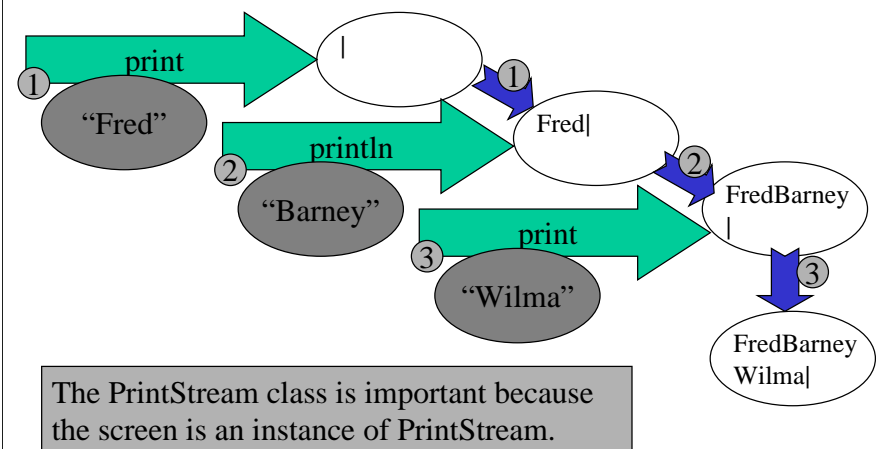


## Partial Protocol - PrintStream

- Some messages require arguments.
- Some messages do not return a result.



## PrintStream Example - print/println



## Outline of Lecture 3

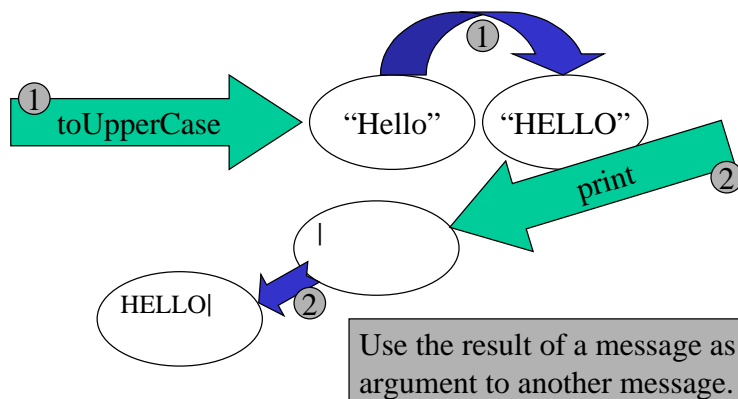


- Objects , messages, classes, instances and state.
- Message expressions that compute results - String examples
- State changes - Stack examples
- Message arguments - Stack and PrintStream examples
- **Composed Messages - String > PrintStream examples**

## Composed Messages

- Since a message can return an object, we can use the returned object as an argument in another message.
- Since a message can return an object, we can send a message to the returned object.

## Example - toUpperCase > print



## Example - toUpperCase > trim

