Structural Programming and Data Structures

Winter 2000

CMPUT 102: Object State

Dr. Osmar R. Zaïane

University of Alberta

Course Content

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

Objectives of Lecture 12

Implementing Classes – Object State

- Understand the state of an object.
- Implement classes with objects that have a state.
- Re-write the Adventure program such that we have many classes and objects with states.

Outline of Lecture 12

- Re-visit the Tune program.
- Using a class
- Instance variables for object state
- Syntax for instance variable references
- Accessing the state of the current object - *this*
- Example class implementation - Adventurer in Adventure Version 3
- Managing multiple classes in Code Warrior

```
public class Tunes {
/*
  Creates a collection of CDs. Adds CDs to the collection
  and displays a summary of the collection value.
*/

   public static void main(String args[]) {

  /* Program statements go here */

    CD_Collection music;

      music = new CD_Collection(5, 50.00f);

      music.addCDs(1, 10.99f);
      music.addCDs(3, 20.99f);
      music.displayCDs();

  }
}
class CD_Collection {
/*  Monitors the value of a collection of musical CDs. */

/* Private instance variables */

   private int numCDs;
   private float valueCDs;

   public CD_Collection (int initialNum, float initialVal) {
   /*
      Initializes the collection with the given number of CDs
      and the given value of the CD collection.
   */

      this.numCDs = initialNum;
      this.valueCDs = initialVal;
   }

   public void add_cds(int number, float value) {
   /*
      Adds CDs to the collection and adjusts the total value.
   */
      this.numCDs = this.numCDs + number;
      this.valueCDs = this.valueCDs + value;
   }

   public void displayCDs() {
   /*
      Displays the number of CDs in the collection and the total
      value of the collection.
   */

      System.out.println("==================================");
      System.out.print  ("Total Number of CDs: ");
      System.out.println(this.numCDs);
      System.out.print  ("Total Value of Collection: ");
      System.out.println(this.valueCDs);
      System.out.print  ("Average cost per CD: $");
      System.out.println(this.averageCost());
      System.out.println("==================================");
   }

   private float averageCost() {
    /*
      Determines the average cost of a CD in the collection.
   */
     float average;

      average = this.valueCDs/this.numCDs;

      return average;
   }

}
```

numCDs
valueCDs

Outline of Lecture 12

- Re-visit the Tune program.
- Using a class
- Instance variables for object state
- Syntax for instance variable references
- Accessing the state of the current object - *this*
- Example class implementation - Adventurer in Adventure Version 3
- Managing multiple classes in Code Warrior

Using a Class

- To use a class we only need to know the class protocol:
  - – a list of public variables
  - – a list of constructors
  - – a list of instance messages

Using Class Adventurer

- Consider the protocol for an Adventurer class that has no public variables, has instance messages:

```
public String name()
public Integer tokens()
public void gainTokens(int gain)
public void loseTokens(int loss)
public void reportTokens()
```

and has a constructor:

```
public Adventurer(String name)
```

Request Messages

- Recall that an object must be able to return an object or value when a message is sent to it.
- For example, an Adventurer object must return a String in response to the name message.
- How do we implement such messages?
  - – let an object "remember" all "requestable" objects
  - – ask another object for the requested object
  - – compute a new object built from other objects

Outline of Lecture 12

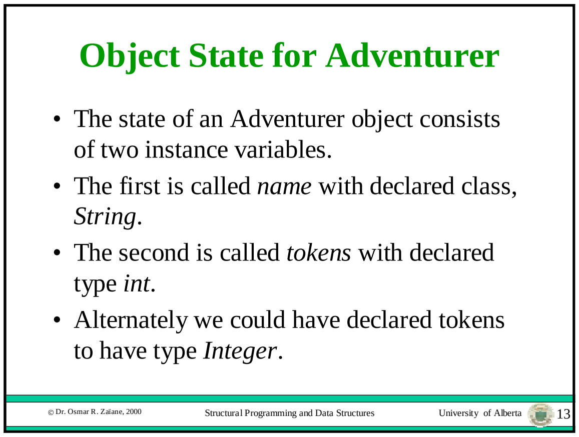
- Re-visit the Tune program.
- Using a class
- Instance variables for object state
- Syntax for instance variable references
- Accessing the state of the current object - *this*
- Example class implementation - Adventurer in Adventure Version 3
- Managing multiple classes in Code Warrior

Remembering versus Computing

- An object only needs to remember enough information to satisfy its protocol.
- For example, if a Person object must respond to the messages age() and birthDate(), it is sufficient to remember a birth-date object.
- An age object can then be computed from the birth-date object and a current date object.

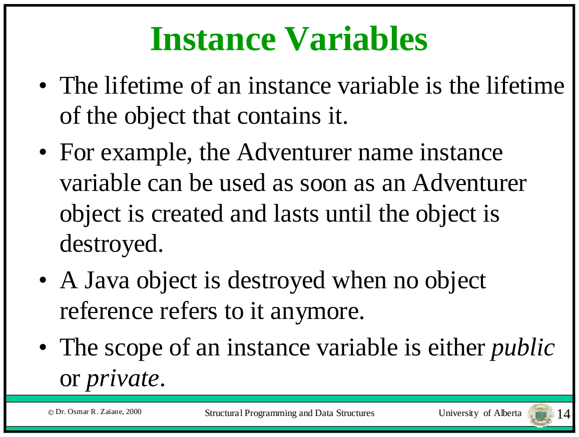
Object State

- The **state** of an object is the set of objects and values that an object "remembers".
- We use variables to remember this information.
- When a class is implemented we declare one **instance variable** for each object or value that an instance of that class must remember.
- Like other variables, each instance variable has a name and declared type.



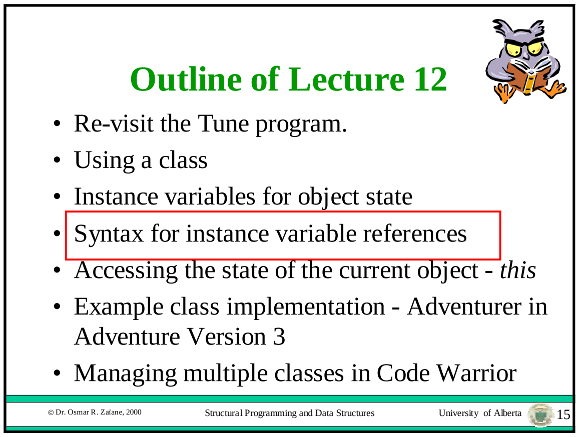
Object State for Adventurer

- The state of an Adventurer object consists of two instance variables.
- The first is called *name* with declared class, *String*.
- The second is called *tokens* with declared type *int*.
- Alternately we could have declared tokens to have type *Integer*.



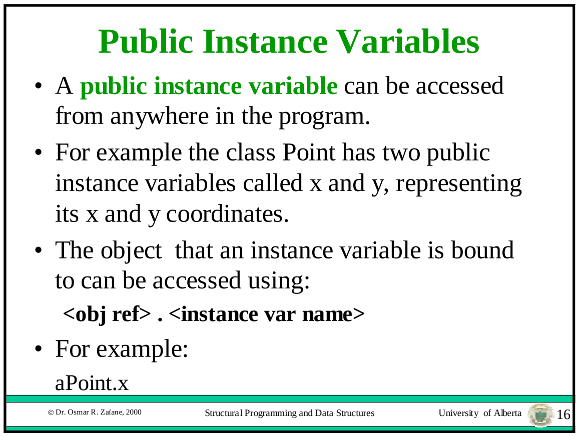
Instance Variables

- The lifetime of an instance variable is the lifetime of the object that contains it.
- For example, the Adventurer name instance variable can be used as soon as an Adventurer object is created and lasts until the object is destroyed.
- A Java object is destroyed when no object reference refers to it anymore.
- The scope of an instance variable is either *public* or *private*.



Outline of Lecture 12

- Re-visit the Tune program.
- Using a class
- Instance variables for object state
- Syntax for instance variable references
- Accessing the state of the current object - *this*
- Example class implementation - Adventurer in Adventure Version 3
- Managing multiple classes in Code Warrior

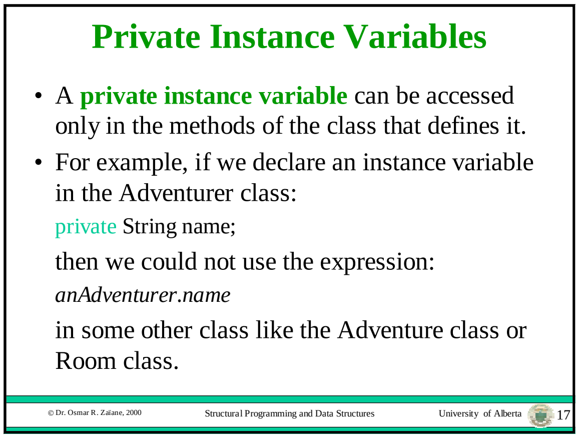


Public Instance Variables

- A **public instance variable** can be accessed from anywhere in the program.
- For example the class Point has two public instance variables called x and y, representing its x and y coordinates.
- The object that an instance variable is bound to can be accessed using:

  <obj ref> . <instance var name>
- For example:

  aPoint.x



Private Instance Variables

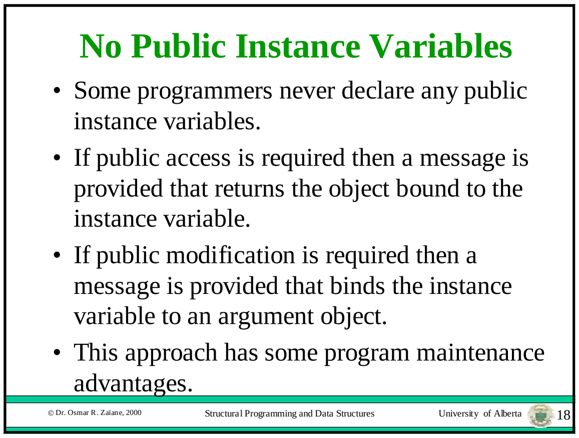
- A **private instance variable** can be accessed only in the methods of the class that defines it.
- For example, if we declare an instance variable in the Adventurer class:

  private String name;

  then we could not use the expression:

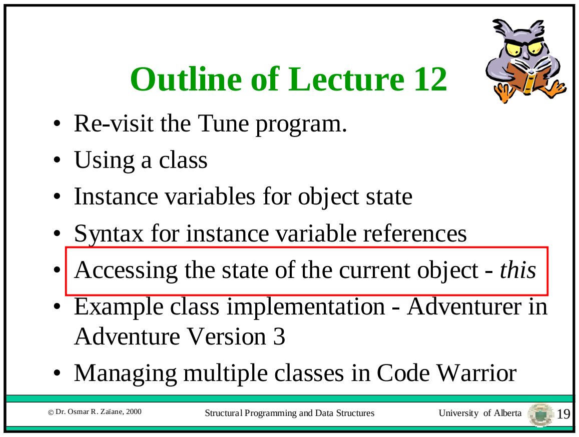
  anAdventurer.name

  in some other class like the Adventure class or Room class.



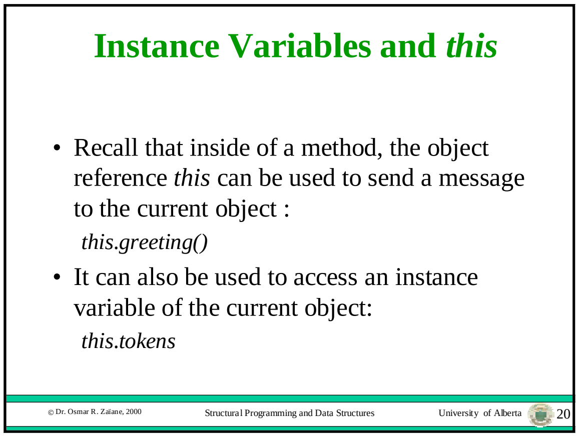
No Public Instance Variables

- Some programmers never declare any public instance variables.
- If public access is required then a message is provided that returns the object bound to the instance variable.
- If public modification is required then a message is provided that binds the instance variable to an argument object.
- This approach has some program maintenance advantages.



Outline of Lecture 12

- Re-visit the Tune program.
- Using a class
- Instance variables for object state
- Syntax for instance variable references
- Accessing the state of the current object - *this*
- Example class implementation - Adventurer in Adventure Version 3
- Managing multiple classes in Code Warrior

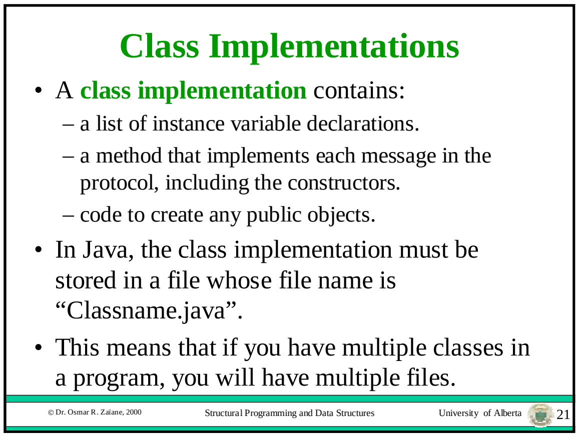


Instance Variables and *this*

- Recall that inside of a method, the object reference *this* can be used to send a message to the current object :

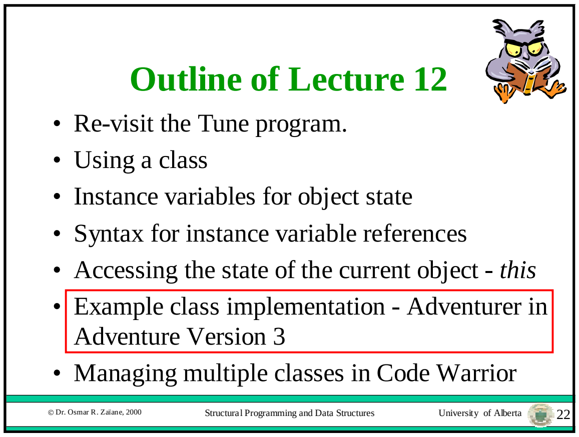
  this.greeting()
- It can also be used to access an instance variable of the current object:

  this.tokens



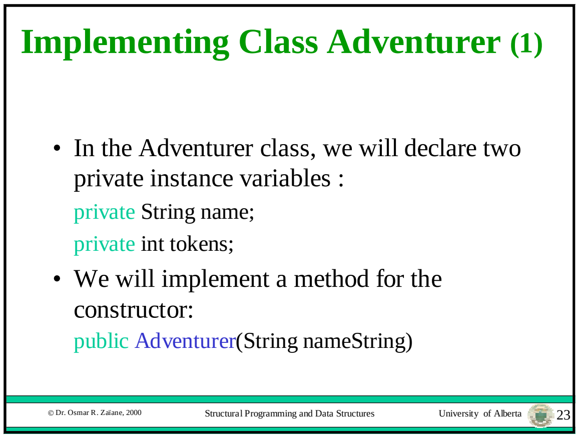
Class Implementations

- A **class implementation** contains:
  - a list of instance variable declarations.
  - a method that implements each message in the protocol, including the constructors.
  - code to create any public objects.
- In Java, the class implementation must be stored in a file whose file name is “Classname.java”.
- This means that if you have multiple classes in a program, you will have multiple files.



Outline of Lecture 12

- Re-visit the Tune program.
- Using a class
- Instance variables for object state
- Syntax for instance variable references
- Accessing the state of the current object - *this*
- Example class implementation - Adventurer in Adventure Version 3
- Managing multiple classes in Code Warrior



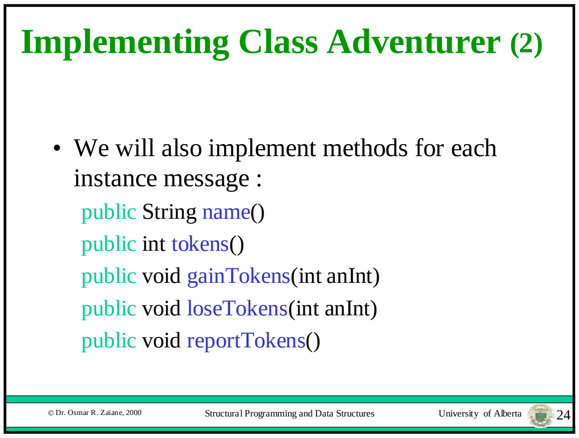
Implementing Class Adventurer (1)

- In the Adventurer class, we will declare two private instance variables :

  private String name;

  private int tokens;
- We will implement a method for the constructor:

  public Adventurer(String nameString)



Implementing Class Adventurer (2)

- We will also implement methods for each instance message :

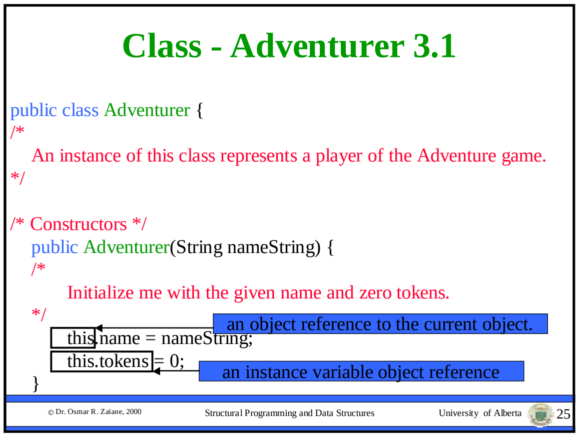
  public String name()

  public int tokens()

  public void gainTokens(int anInt)

  public void loseTokens(int anInt)

  public void reportTokens()

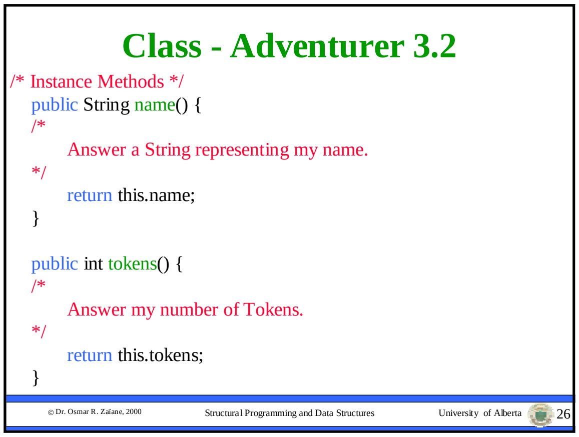


Class - Adventurer 3.1

```
public class Adventurer {
/*
   An instance of this class represents a player of the Adventure game.
*/

/* Constructors */
   public Adventurer(String nameString) {
   /*
        Initialize me with the given name and zero tokens.
   */
        this.name = nameString;
        this.tokens = 0;
   }
```

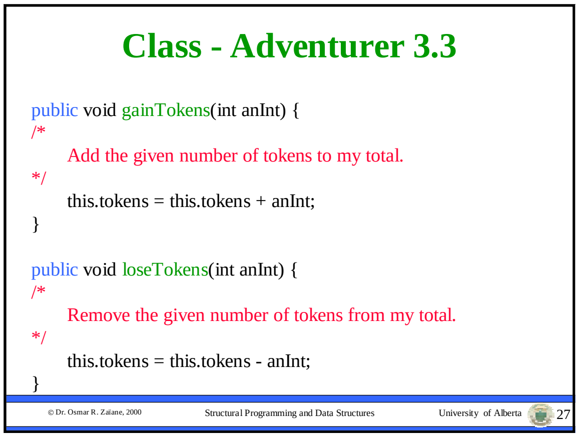
an object reference to the current object.

an instance variable object reference



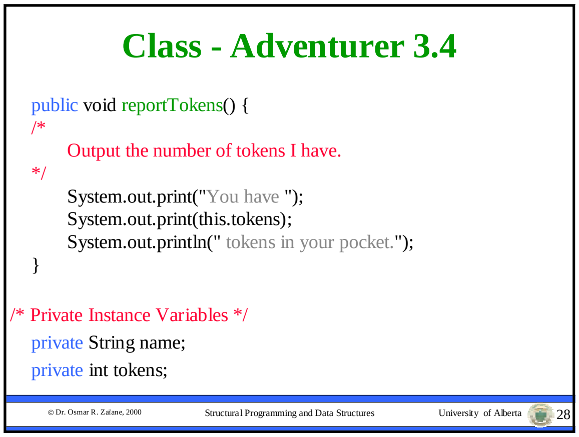
Class - Adventurer 3.2

```
/* Instance Methods */
   public String name() {
   /*
        Answer a String representing my name.
   */
        return this.name;
   }

   public int tokens() {
   /*
        Answer my number of Tokens.
   */
        return this.tokens;
   }
```



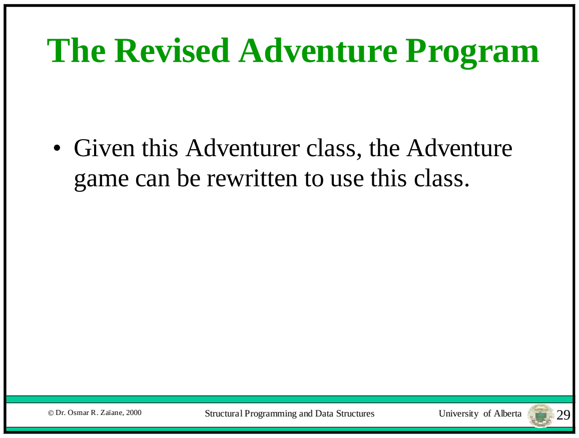
Class - Adventurer 3.3

```
   public void gainTokens(int anInt) {
   /*
        Add the given number of tokens to my total.
   */
        this.tokens = this.tokens + anInt;
   }

   public void loseTokens(int anInt) {
   /*
        Remove the given number of tokens from my total.
   */
        this.tokens = this.tokens - anInt;
   }
```



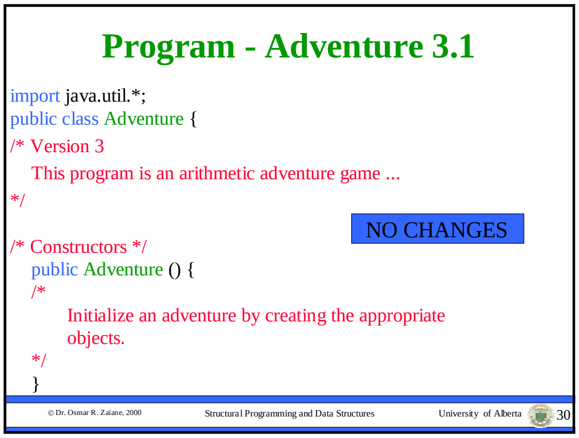
Class - Adventurer 3.4

```
   public void reportTokens() {
   /*
        Output the number of tokens I have.
   */
        System.out.print("You have ");
        System.out.print(this.tokens);
        System.out.println(" tokens in your pocket.");
   }

/* Private Instance Variables */
   private String name;
   private int tokens;
```



The Revised Adventure Program

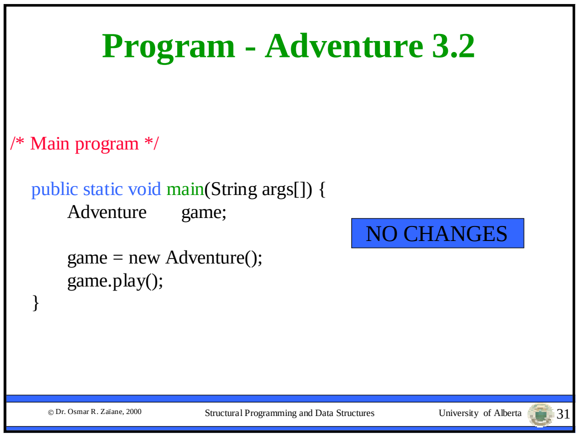
- Given this Adventurer class, the Adventure game can be rewritten to use this class.



Program - Adventure 3.1

```
import java.util.*;
public class Adventure {
/* Version 3
   This program is an arithmetic adventure game ...
*/

/* Constructors */
   public Adventure () {
   /*
        Initialize an adventure by creating the appropriate
        objects.
   */
   }
```

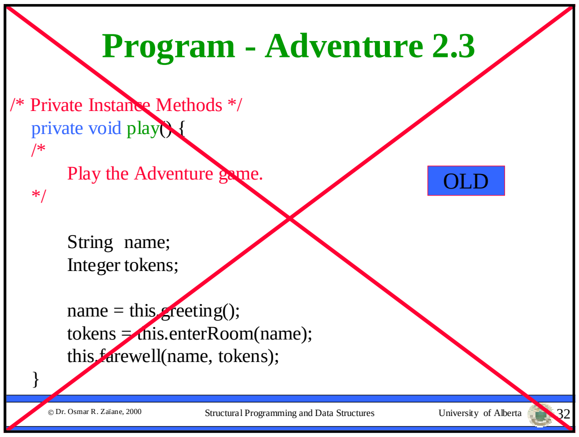
NO CHANGES



Program - Adventure 3.2

```
/* Main program */

    public static void main(String args[]) {
        Adventure      game;

        game = new Adventure();
        game.play();
    }
```

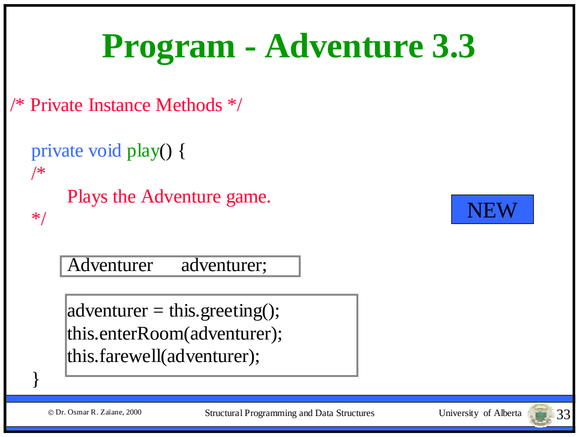
NO CHANGES



Program - Adventure 2.3

```
/* Private Instance Methods */
    private void play() {
    /*
        Play the Adventure game.
    */

        String  name;
        Integer tokens;

        name = this.greeting();
        tokens = this.enterRoom(name);
        this.farewell(name, tokens);
    }
```

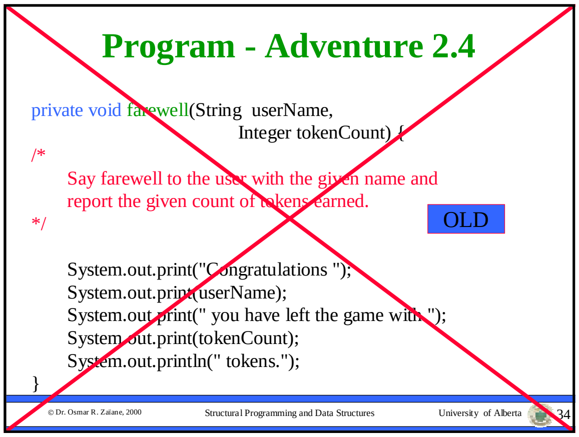
OLD



Program - Adventure 3.3

```
/* Private Instance Methods */

    private void play() {
    /*
        Plays the Adventure game.
    */

        Adventurer      adventurer;

        adventurer = this.greeting();
        this.enterRoom(adventurer);
        this.farewell(adventurer);
    }
```

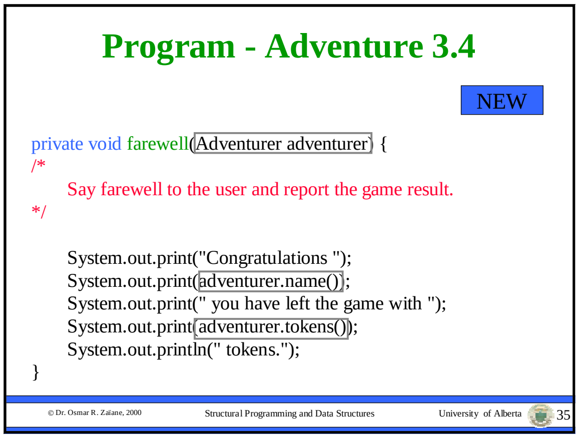
NEW



Program - Adventure 2.4

```
    private void farewell(String  userName,
                             Integer tokenCount) {
    /*
        Say farewell to the user with the given name and
        report the given count of tokens earned.
    */

        System.out.print("Congratulations ");
        System.out.print(userName);
        System.out.print(" you have left the game with ");
        System.out.print(tokenCount);
        System.out.println(" tokens.");
    }
```

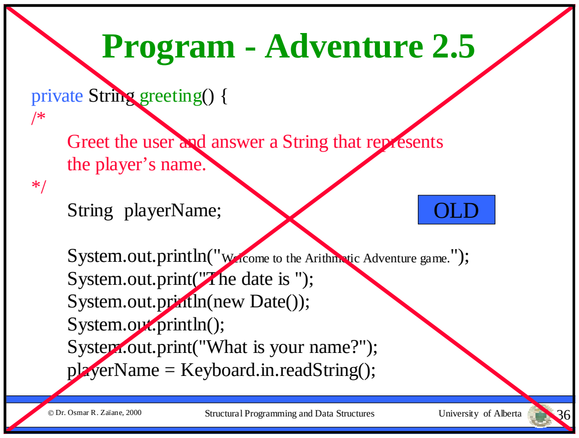
OLD



Program - Adventure 3.4

NEW

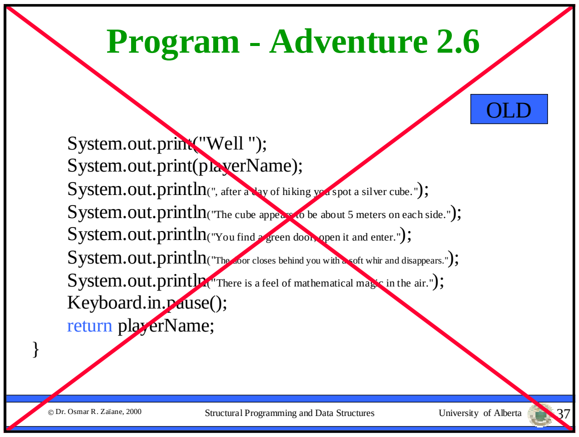
```
    private void farewell(Adventurer adventurer) {
    /*
        Say farewell to the user and report the game result.
    */

        System.out.print("Congratulations ");
        System.out.print(adventurer.name());
        System.out.print(" you have left the game with ");
        System.out.print(adventurer.tokens());
        System.out.println(" tokens.");
    }
```



Program - Adventure 2.5

```
    private String greeting() {
    /*
        Greet the user and answer a String that represents
        the player's name.
    */
        String  playerName;

        System.out.println("Welcome to the Arithmetic Adventure game.");
        System.out.print("The date is ");
        System.out.println(new Date());
        System.out.println();
        System.out.print("What is your name?");
        playerName = Keyboard.in.readString();
```

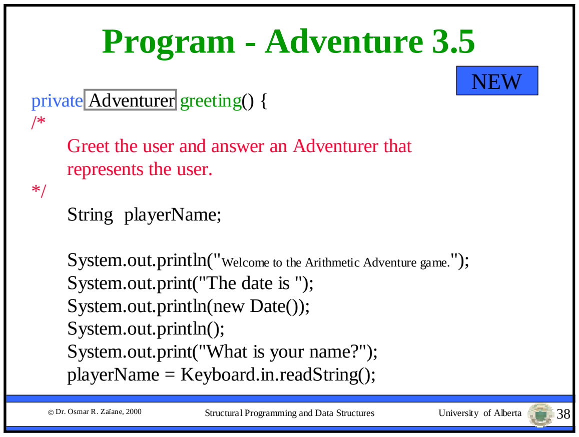
OLD



Program - Adventure 2.6

OLD

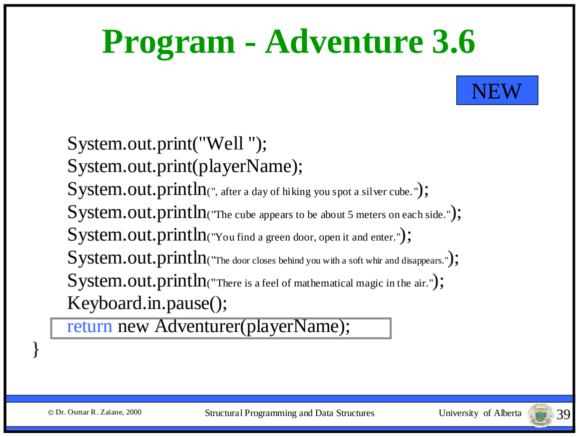
```
        System.out.print("Well ");
        System.out.print(playerName);
        System.out.println(", after a day of hiking you spot a silver cube.");
        System.out.println("The cube appears to be about 5 meters on each side.");
        System.out.println("You find a green door, open it and enter.");
        System.out.println("The door closes behind you with a soft whir and disappears.");
        System.out.println("There is a feel of mathematical magic in the air.");
        Keyboard.in.pause();
        return playerName;
}
```



Program - Adventure 3.5

NEW

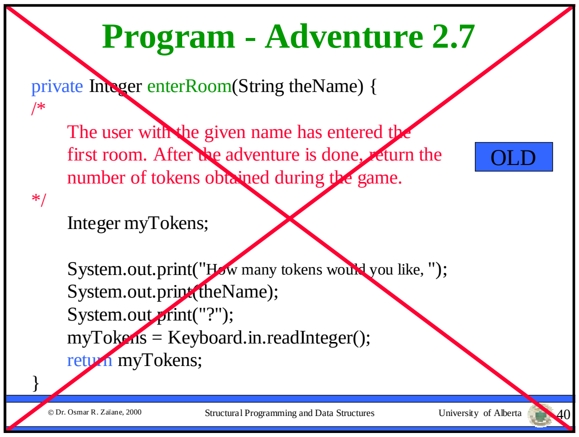
```
private Adventurer greeting() {
/*
    Greet the user and answer an Adventurer that
    represents the user.
*/
    String  playerName;

    System.out.println("Welcome to the Arithmetic Adventure game.");
    System.out.print("The date is ");
    System.out.println(new Date());
    System.out.println();
    System.out.print("What is your name?");
    playerName = Keyboard.in.readString();
```



Program - Adventure 3.6

NEW

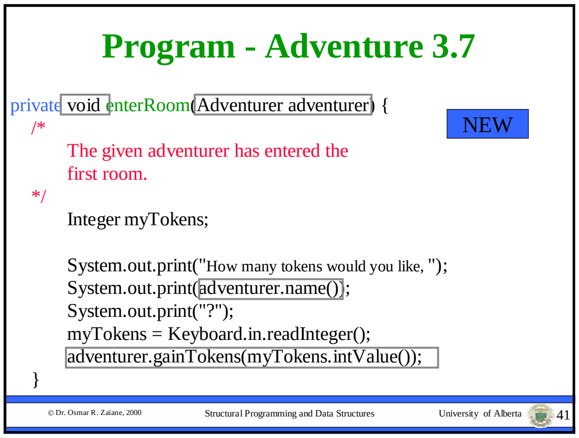
```
        System.out.print("Well ");
        System.out.print(playerName);
        System.out.println(", after a day of hiking you spot a silver cube.");
        System.out.println("The cube appears to be about 5 meters on each side.");
        System.out.println("You find a green door, open it and enter.");
        System.out.println("The door closes behind you with a soft whir and disappears.");
        System.out.println("There is a feel of mathematical magic in the air.");
        Keyboard.in.pause();
        return new Adventurer(playerName);
}
```



Program - Adventure 2.7

OLD

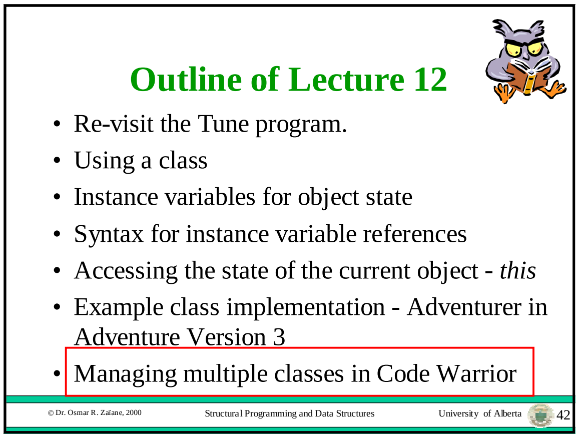
```
private Integer enterRoom(String theName) {
/*
    The user with the given name has entered the
    first room.  After the adventure is done, return the
    number of tokens obtained during the game.
*/
    Integer myTokens;

    System.out.print("How many tokens would you like, ");
    System.out.print(theName);
    System.out.print("?");
    myTokens = Keyboard.in.readInteger();
    return myTokens;
}
```



Program - Adventure 3.7

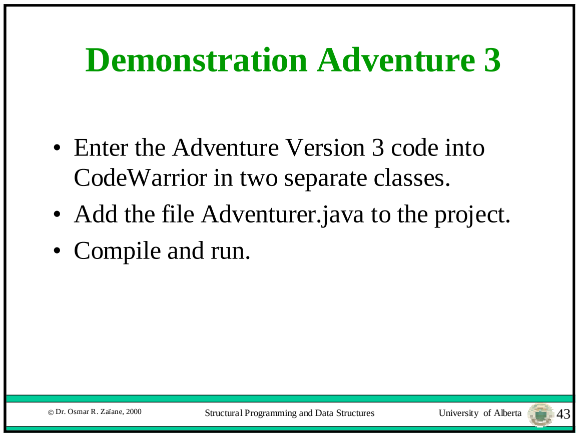
NEW

```
private void enterRoom(Adventurer adventurer) {
   /*
        The given adventurer has entered the
        first room.
   */
        Integer myTokens;

        System.out.print("How many tokens would you like, ");
        System.out.print(adventurer.name());
        System.out.print("?");
        myTokens = Keyboard.in.readInteger();
        adventurer.gainTokens(myTokens.intValue());
}
```



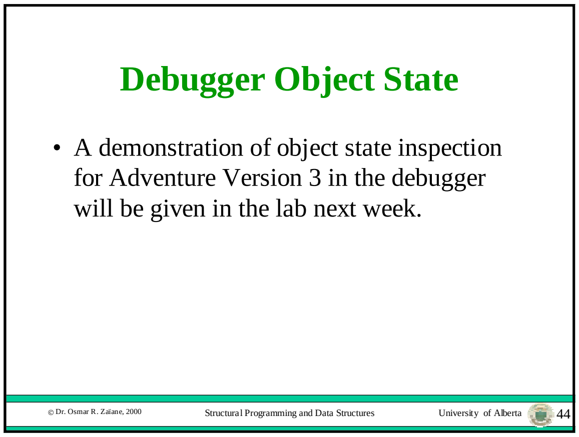
Outline of Lecture 12

- Re-visit the Tune program.
- Using a class
- Instance variables for object state
- Syntax for instance variable references
- Accessing the state of the current object - *this*
- Example class implementation - Adventurer in Adventure Version 3
- Managing multiple classes in Code Warrior



Demonstration Adventure 3

- Enter the Adventure Version 3 code into CodeWarrior in two separate classes.
- Add the file Adventurer.java to the project.
- Compile and run.



Debugger Object State

- A demonstration of object state inspection for Adventure Version 3 in the debugger will be given in the lab next week.