

Structural Programming and Data Structures

Winter 2000

CMPUT 102: Tracing Programs

Dr. Osmar R. Zaiane



University of Alberta

© Dr. Osmar R. Zaiane, 2000

Structural Programming and Data Structures

University of Alberta

1

Course Content

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



© Dr. Osmar R. Zaiane, 2000

Structural Programming and Data Structures

University of Alberta

2

Objectives of Lecture 11 Tracing Programs and the Debugger

- Learn how to trace the execution of a Java program.
- Understand what is happening during the execution of a program.
- Use program tracing:
 - to find errors in programs;
 - to understand what a program is supposed to do.
- Introduce the debugging facilities.

© Dr. Osmar R. Zaiane, 2000

Structural Programming and Data Structures

University of Alberta

3

Outline of Lecture 11



- Example of a new program
- Notation for hand tracing
- Hand tracing Adventure
- The Code Warrior Debugger
- Tracing the example program again

© Dr. Osmar R. Zaiane, 2000

Structural Programming and Data Structures

University of Alberta

4

```
public class Times {
    /*
     * 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.addCD(1, 10.99f);
        music.addCD(2, 20.99f);
        music.displayCDs();
    }

    class CD_Collection {
        /*
         * Mention 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) {
            /*
             * Add 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.println("Total Number of CDs: " + this.numCDs);
            System.out.println("Total Value of Collection: " + this.valueCDs);
            System.out.println("Average cost per CD: " + 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;
        }
    }
}
```

© Dr. Osmar R. Zaiane, 2000

Structural Programming and Data Structures

University of Alberta

5

Outline of Lecture 11



- Example of a new program
- Notation for hand tracing
- Hand tracing Adventure
- The Code Warrior Debugger
- Tracing the example program again

© Dr. Osmar R. Zaiane, 2000

Structural Programming and Data Structures

University of Alberta

6

Tracing

- Tracing is a technique that follows the execution of program in detail.
- Tracing can be used to understand how a Java program works.
- Tracing can also be used to find semantic errors in a program.
- A program can be hand traced by drawing diagrams.
- A program can also be traced using a tool called a debugger.

© Dr. Omar R. Zaidan, 2000

Structural Programming and Data Structures

University of Alberta

7

Notation for Hand Tracing

- Every method is represented by a rectangle.
- Every object is represented by an oval labeled by its class or its contents.
- Every reference is represented by a rectangle in the method that declares it.
- However, you can ignore public imported variables.
- Every reference has an arc connecting it to the object that it references.

© Dr. Omar R. Zaidan, 2000

Structural Programming and Data Structures

University of Alberta

8

Outline of Lecture 11



- Example of a new program
- Notation for hand tracing
- Hand tracing Adventure
- The Code Warrior Debugger
- Tracing the example program again

© Dr. Omar R. Zaidan, 2000

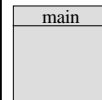
Structural Programming and Data Structures

University of Alberta

9

Adventure Trace - call main

- Since this is an application, the interpreter invokes the static method called *main*.
- Since *main* is static, there is no *this*.



© Dr. Omar R. Zaidan, 2000

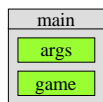
Structural Programming and Data Structures


University of Alberta

10

Adventure Trace - main

- The parameter *args* is a reference
- The variable *game* is a reference



```
public static void main(String args[]) {  
    Adventure game;   
  
    game = new Adventure();  
    game.play();  
}
```

© Dr. Omar R. Zaidan, 2000

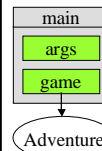
Structural Programming and Data Structures


University of Alberta

11

Adventure Trace - main - game

- When the new Adventure object is created we draw it and when the *game* reference is bound to the new object we connect it.



```
public static void main(String args[]) {  
    Adventure game;  
  
    game = new Adventure();   
    game.play();  
}
```

© Dr. Omar R. Zaidan, 2000

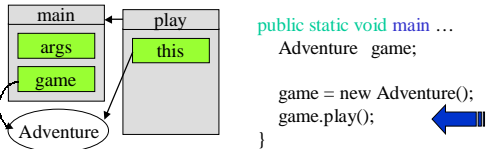
Structural Programming and Data Structures

University of Alberta

12

Adventure Trace - call play

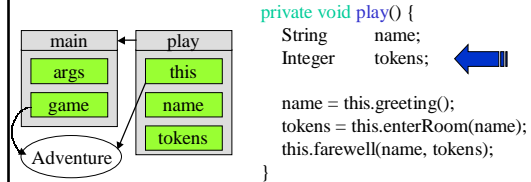
- When the play() message is sent to the *game* object, we draw a rectangle for the play() method that contains the reference *this*, connect the methods and bind the *this* reference to the receiver object.



© Dr. Omar R. Zaiane, 2000 Structural Programming and Data Structures University of Alberta 13

Adventure Trace - play

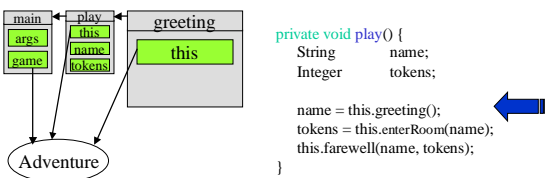
- There are no method parameters, there are two variables, *name* and *tokens*.



© Dr. Omar R. Zaiane, 2000 Structural Programming and Data Structures University of Alberta 14

Adventure Trace - call greeting

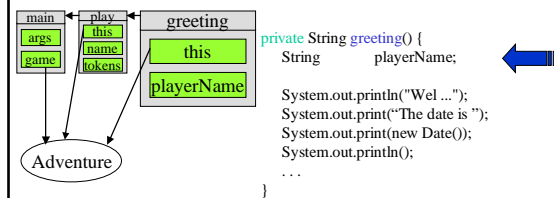
- When greeting() is sent to the *this* object, we draw a greeting() method with a new *this* reference, connect the methods and bind the new *this* to the receiver object.



© Dr. Omar R. Zaiane, 2000 Structural Programming and Data Structures University of Alberta 15

Adventure Trace - greeting

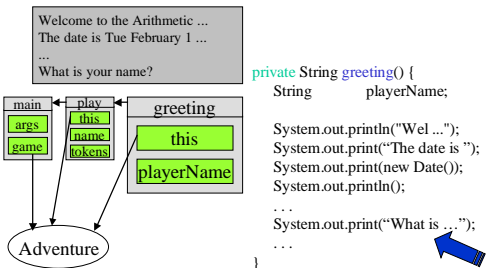
- There are no method parameters, there is one variable, *playerName*.



© Dr. Omar R. Zaiane, 2000 Structural Programming and Data Structures University of Alberta 16

Adventure Trace - greeting output

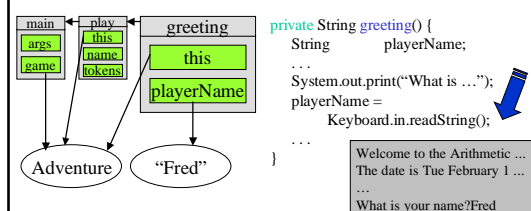
- Output some information.



© Dr. Omar R. Zaiane, 2000 Structural Programming and Data Structures University of Alberta 17

Adventure Trace - greeting input

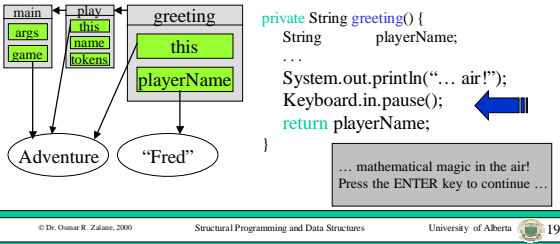
- Send readString() to the keyboard, get back a String object that represents what the user typed and bind *playerName* to it.



© Dr. Omar R. Zaiane, 2000 Structural Programming and Data Structures University of Alberta 18

Adventure Trace - greeting pause

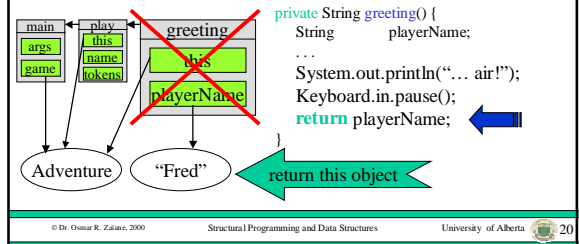
- Output some more information and ask the keyboard to pause.
- Wait until the user presses the ENTER key.



© Dr. Omar R. Zaiane, 2000 Structural Programming and Data Structures University of Alberta 19

Adventure Trace - greeting return

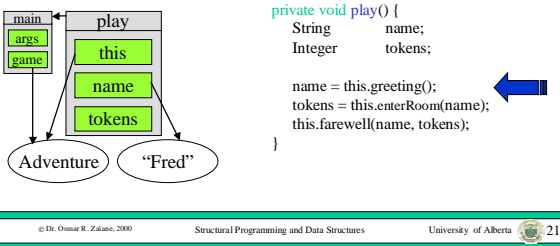
- Return the object bound to the variable *playerName* as the result of the message and discard the method.



© Dr. Omar R. Zaiane, 2000 Structural Programming and Data Structures University of Alberta 20

Adventure Trace - play name

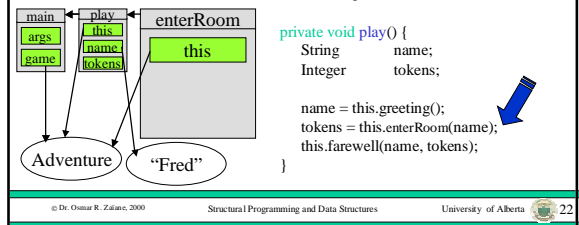
- Bind the variable *name* to the object that was returned from the `greeting()` message.



© Dr. Omar R. Zaiane, 2000 Structural Programming and Data Structures University of Alberta 21

Adventure Trace - call enterRoom

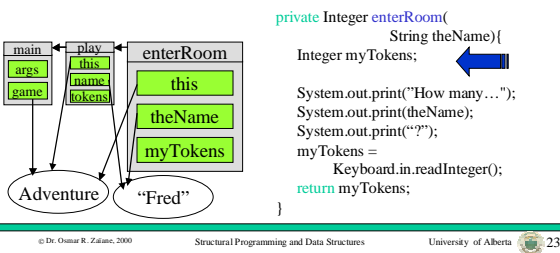
- When `enterRoom()` is sent to *this*, we draw an `enterRoom()` method with a new *this* reference, connect the methods and bind the new *this* to the receiver object.



© Dr. Omar R. Zaiane, 2000 Structural Programming and Data Structures University of Alberta 22

Adventure Trace - enterRoom

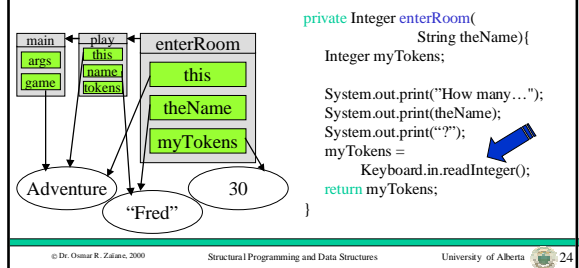
- There is a method parameter called *theName* that is bound to the argument object and a variable, *myTokens*.



© Dr. Omar R. Zaiane, 2000 Structural Programming and Data Structures University of Alberta 23

Adventure Trace - enterRoom input

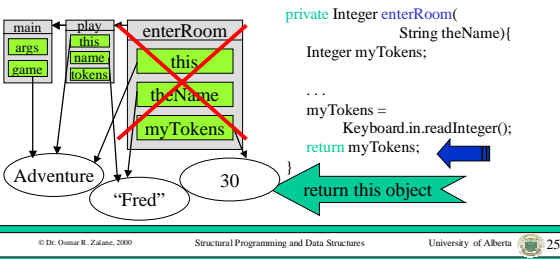
- Output some information, input an Integer from the keyboard and bind *myTokens* to it.



© Dr. Omar R. Zaiane, 2000 Structural Programming and Data Structures University of Alberta 24

Adventure Trace - enterRoom return

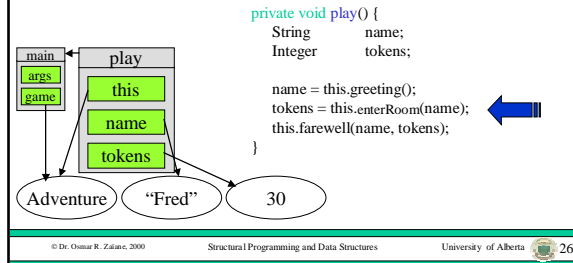
- Return the object bound to the variable *myTokens* as the result of the message and discard the method.



© Dr. Omar R. Zaiane, 2000 Structural Programming and Data Structures University of Alberta 25

Adventure Trace - play tokens

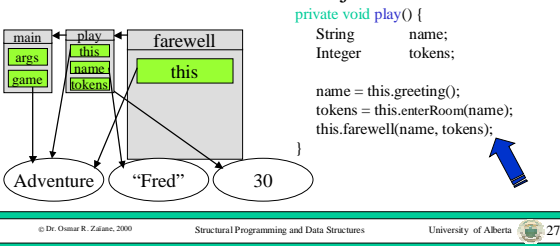
- Bind the variable *tokens* to the object that was returned from the enterRoom() message.



© Dr. Omar R. Zaiane, 2000 Structural Programming and Data Structures University of Alberta 26

Adventure Trace - call farewell

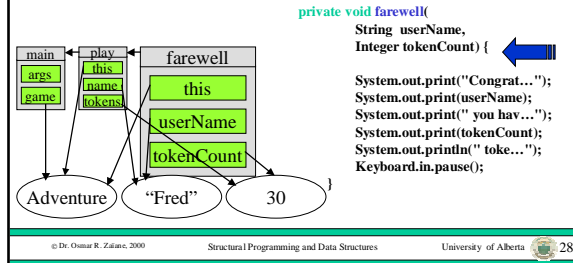
- When farewell() is sent to *this*, we draw a farewell() method with a new *this* reference, connect the methods and bind the new *this* to the receiver object.



© Dr. Omar R. Zaiane, 2000 Structural Programming and Data Structures University of Alberta 27

Adventure Trace - farewell

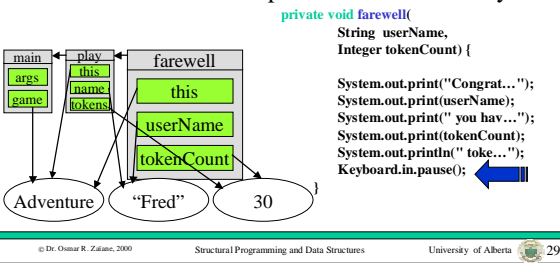
- There are method parameters called *userName* and *tokenCount* that are bound to the argument objects and no variables.



© Dr. Omar R. Zaiane, 2000 Structural Programming and Data Structures University of Alberta 28

Adventure Trace - farewell output

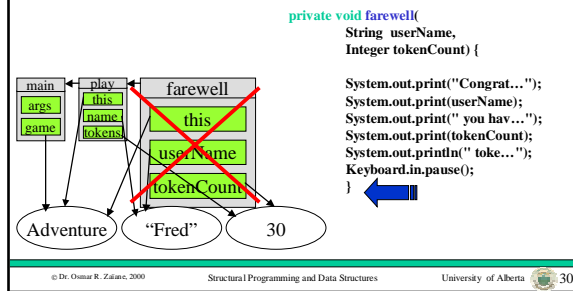
- Output some information and ask the keyboard to pause.
- Wait until the user presses the ENTER key.



© Dr. Omar R. Zaiane, 2000 Structural Programming and Data Structures University of Alberta 29

Adventure Trace - farewell return

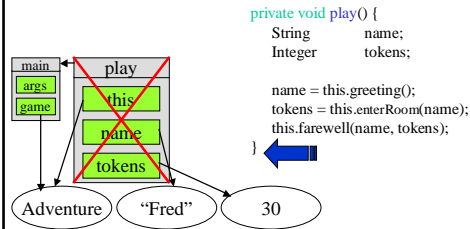
- This method does not return a result so just discard the method.



© Dr. Omar R. Zaiane, 2000 Structural Programming and Data Structures University of Alberta 30

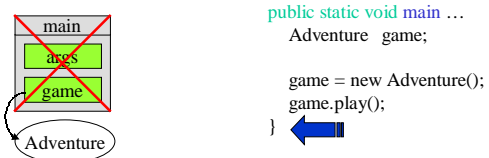
Adventure Trace - play return

- This method does not return a result so just discard the method.



Adventure Trace - main return

- The static main method does not return a result so just discard the method.
- The program is now done.



Outline of Lecture 11



- Example of a new program
- Notation for hand tracing
- Hand tracing Adventure
- The Code Warrior Debugger
- Tracing the example program again

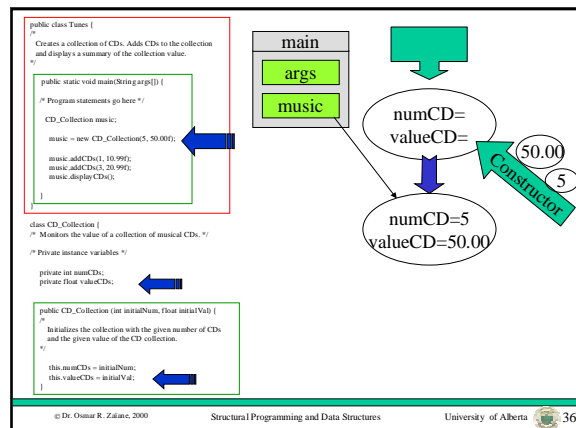
Demonstration Debugger

- Trace Adventure Version 2 in CodeWarrior using the debugger.
- A demo of the debugger will be given in the lab.
- The Debugger will allow you to execute you Java program statement by statement, and visualize your objects and variables during runtime.

Outline of Lecture 11



- Example of a new program
- Notation for hand tracing
- Hand tracing Adventure
- The Code Warrior Debugger
- Tracing the example program again



```

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, 50.00);
        music.addCDs(1, 10.99);
        music.addCDs(3, 20.99);
        music.displayCDs();
    }
}

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

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

```

© Dr. Osnar R. Zaiac, 2000 Structural Programming and Data Structures University of Alberta 37

```

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, 50.00);
        music.addCDs(1, 10.99);
        music.addCDs(3, 20.99);
        music.displayCDs();
    }
}

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

    public void displayCDs() {
        /*
        Displays the number of CDs in the collection and the total
        value of the collection.
        */
        System.out.println("-----");
        System.out.println("Total Number of CDs: ");
        System.out.println(this.numCDs);
        System.out.println("Total Value of Collection: ");
        System.out.println(this.valueCDs);
        System.out.println("Average cost per CD: $");
        System.out.println(this.averageCost());
        System.out.println("-----");
    }
}

```

© Dr. Osnar R. Zaiac, 2000 Structural Programming and Data Structures University of Alberta 38

```

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, 50.00);
        music.addCDs(1, 10.99);
        music.addCDs(3, 20.99);
        music.displayCDs();
    }
}

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

    private float averageCost() {
        /*
        Determines the average cost of a CD in the collection.
        */
        float average;
        average = this.valueCDs / this.numCDs;
        return average;
    }
}

```

© Dr. Osnar R. Zaiac, 2000 Structural Programming and Data Structures University of Alberta 39

```

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, 50.00);
        music.addCDs(1, 10.99);
        music.addCDs(3, 20.99);
        music.displayCDs();
    }
}

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

    public void displayCDs() {
        /*
        Displays the number of CDs in the collection and the total
        value of the collection.
        */
        System.out.println("-----");
        System.out.println("Total Number of CDs: ");
        System.out.println(this.numCDs);
        System.out.println("Total Value of Collection: ");
        System.out.println(this.valueCDs);
        System.out.println("Average cost per CD: $");
        System.out.println(this.averageCost());
        System.out.println("-----");
    }
}

```

© Dr. Osnar R. Zaiac, 2000 Structural Programming and Data Structures University of Alberta 40