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.

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, 50.00f);
        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.println("Total Number of CDs: ");System.out.println(this.numCDs);
        System.out.println("Total Value of CDs: ");System.out.println(this.valueCDs);
        System.out.println("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;
    }
}

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

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.

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.
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

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.

Adventure Trace - main

• The parameter args is a reference
• The variable game is a reference

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

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.

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

```
public static void main ... 
    Adventure game; 
    game = new Adventure(); 
    game.play(); 
}
```

**Adventure Trace - play**

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

```
private void play() { 
    String name; 
    Integer tokens; 
    name = this.greeting(); 
    tokens = this.enterRoom(name); 
    this.farewell(name, tokens); 
}
```

**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.

```
private String greeting() { 
    String playerName; 
    System.out.println("Wel ... "); 
    System.out.print("The date is "); 
    System.out.print(new Date()); 
    System.out.println(); 
    name = this.greeting(); 
    tokens = this.enterRoom(name); 
    this.farewell(name, tokens); 
}
```

**Adventure Trace - greeting**

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

```
private String greeting() { 
    String playerName; 
    System.out.println("Wel ..."); 
    System.out.print("The date is "); 
    System.out.print(new Date()); 
    System.out.println(); 
}
```
### Adventure Trace - greeting output

- Output some information.

```java
private String greeting() {
    String playerName;
    System.out.println("Wel ...");
    System.out.print("The date is ");System.out.print(new Date());System.out.println();
    System.out.print("What is ... ");
    playerName = Keyboard.in.readString();
    return playerName;
}
```

---

### Adventure Trace - greeting return

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

```java
private String greeting() {
    String playerName;
    System.out.println("… air!");
    Keyboard.in.pause();
    return playerName;
}
```
**Adventure Trace - play name**

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

```java
private void play() {
    String name;
    Integer tokens;

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

**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.

```java
private void play() {
    String name;
    Integer tokens;

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

**Adventure Trace - enterRoom**

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

```java
private Integer enterRoom(String theName) {
    Integer myTokens;
    System.out.print("How many…");
    System.out.print(theName); System.out.print("?");
    myTokens = Keyboard.in.readInteger();
    return myTokens;
}
```

**Adventure Trace - enterRoom input**

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

```java
private Integer enterRoom(String theName) {
    Integer myTokens;

    System.out.print("How many…");
    System.out.print(theName); System.out.print("?");
    myTokens = Keyboard.in.readInteger();
    return myTokens;
}
```
**Adventure Trace - enterRoom**

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

```java
private Integer enterRoom(String theName) {
    Integer myTokens = Keyboard.in.readInteger();
    return myTokens;
}
```

**Adventure Trace - play tokens**

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

```java
private void play() {
    String name;
    Integer tokens;

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

**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.

```java
private void farewell(String userName, Integer tokenCount) {
    System.out.print("Congrat…");
    System.out.print(userName);
    System.out.print(" you hav…");
    System.out.println(" toke…");
    Keyboard.in.pause();
}
```

**Adventure Trace - farewell**

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

```java
private void farewell(String userName,
                      Integer tokenCount) {
    System.out.print("Congrat…");
    System.out.print(userName);
    System.out.print(" you hav…");
    System.out.println(" toke…");
    Keyboard.in.pause();
}
```
Adventure Trace - farewell output

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

```java
private void farewell(String userName, Integer tokenCount) {
    System.out.print("Congrat…");
    System.out.print(userName);
    System.out.print(" you hav…");
    System.out.print(tokenCount);
    System.out.println(" toke…");
    Keyboard.in.pause();
}
```

Adventure Trace - farewell return

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

```java
private void farewell(String userName, Integer tokenCount) {
    System.out.print("Congrat…");
    System.out.print(userName);
    System.out.print(" you hav…");
    System.out.print(tokenCount);
    System.out.println(" toke…");
    Keyboard.in.pause();
}
```

Adventure Trace - play return

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

```java
private void play() {
    String name;
    Integer tokens;
    name = this.greeting();
    tokens = this.enterRoom(name);
    this.farewell(name, tokens);
}
```

Adventure Trace - main return

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

```java
public static void main(String[] args) {
    Adventure game;
    game = new Adventure();
    game.play();
}
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 your Java program statement by statement, and visualize your objects and variables during runtime.

```
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 CD_Collection (int initialNum) {
        this(initialNum, initialVal);
    }
    public CD_Collection (int) {
        this(initialVal);
    }
    public CD_Collection () {
        this(1, 1.00f);
    }
}
```
public class Tunes {
    /* Creates a collection of CDs. Adds CDs to the collection and displays a summary of the collection value. */
    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 float averageCost() {
        /* Determines the average cost of a CD in the collection. */
        float average;
        average = this.valueCDs / this.numCDs;
        return average;
    }
}

public class Tunes {
    /* Creates a collection of CDs. Adds CDs to the collection and displays a summary of the collection value. */
    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 float averageCost() {
        /* Determines the average cost of a CD in the collection. */
        float average;
        average = this.valueCDs / this.numCDs;
        return average;
    }
}