Repetition Control Structures - while

Cmput 114 - Lecture 16
Department of Computing Science
University of Alberta

About This Lecture

- So far our programs execute each statement exactly once or zero times, if a selection control structure is used.
- In this lecture we will learn how to write programs in which statements can be executed many times using an indefinite repetition control structure called the while statement.

Outline

- The while statement
- Input validation
- Adventure Version 7

Repetition

- So far we have seen two control structures:
  - a sequence of statements that executes each statement in the sequence
  - a selection control structure that selectively executes statements.
- Sometimes it is useful to execute a block of several statements more than once.
- A control structure that supports this is called a repetition control structure.

Java Syntax: while Statement

- The syntax for a while statement in Java is:

```
<while statement> ::=
    while (<condition>) <statement>
```

- For example:

```
i = 0;
while (i <= 4) {
    System.out.println(i);
    i = i + 1;
}
System.out.println("That's all folks");
```


Semantics - while

- If the condition evaluates to true then the statement is executed.
- The condition is then evaluated again.
- If the condition is still true then the statement is executed again.
- This continues until the condition evaluates to false.
- Control then goes to the statement after the while statement.

7

While Semantics - Example

```
i = 0;
while (i <= 4) {
      System.out.println(i);
      i = i + 1;
}
System.out.println("That's all folks!");
```

```
0
1
2
3
4
That's all folks!
```

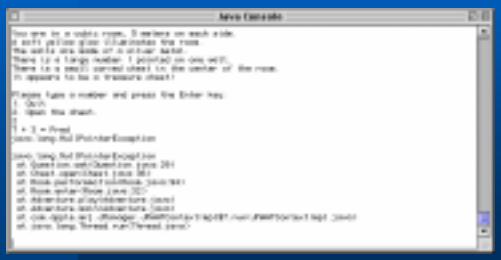

8

Input Validation - Question

- What happens in the Adventure game if a Question asks the user for an answer and the user types a String that does not represent an integer?

9

Question Input Validation Error



10

Question - No Input Validation

```
/* Instance Methods */
  public boolean ask() {
  /*
      Pose myself and answer true or false depending
      on whether the user answers correctly or not.
  */
      Integer      answer;

      System.out.print(this.leftOperand);
      System.out.print(" + ");
      System.out.print(this.rightOperand);
      System.out.print(" = ");
      answer = Keyboard.in.readInteger();
      return (answer.intValue() == this.answer);
  }
```

answer may be bound to null

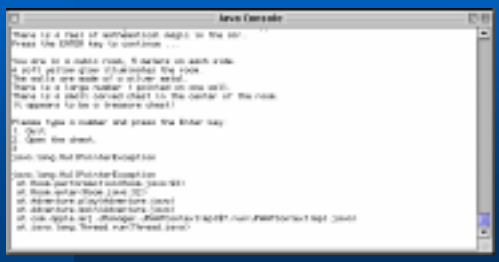
11

Input Validation - TextMenu

- What happens in the Adventure game if a TextMenu queries the user for an answer and the user makes an invalid choice?

12

TextMenu Input Validation Error



13

TextMenu - Input Validation 1

```
private Room performAction(String action, Adventurer adventurer) {
/*Perform the action described by the given String for
      the given Adventurer. Return the room the user
      selected, null if the user selected quit and this
      room if the user selected to open the chest.
*/
  if (action.equals("Open the chest.")) {
      this.chest.open(adventurer);
      this.chest = null;
      return this;
  }
  if (action.equals("Quit"))
      return null;
  return null;
}
```

action is bound to null

14

TextMenu - Input Validation 2

```
public Room enter(Adventurer adventurer) {
/*
    Describe myself, display a list of options, and
    perform the selected option. If the user selected
    quit then return null. If the user selected to go
    to another Room then return that Room. Otherwise
    return this Room.
*/
    TextMenu    menu;
    String      action;
    this.display();
    menu = this.buildMenu();
    action = menu.launch();
    return this.performAction(action, adventurer);
}
```

action is bound to null

15

TextMenu - Input Validation 3

```
  public String launch() {
    /*
        Display myself and answer the String entry
        selected by the user.
    */
        Integer     choice;
        int         index;

        this.display();
        choice = Keyboard.in.readInteger();
        if (choice == null)
            return this.entry1;
        index = choice.intValue();
```

choice is bound to Integer(3)
So index is bound to 3

16

TextMenu - Input Validation 4

```
        switch (index) {
            case 1: return this.entry1;
            case 2: return this.entry2;
            case 3: return this.entry3;
            case 4: return this.entry4;
            case 5: return this.entry5;
            default: return this.entry1;
        }
    }
```

entry3 is bound to null

17

Validate User Input Using while

- A while statement can be used to check user input and to re-query until valid input is entered.
- The general format of this approach is:

```
answer = null;
while (answer == null) {
  //display prompt
  answer = //valid answer or null
}
```


18

while in Class Question

```
answer = null;
while (answer == null) {
  this.display();
  answer = keyboard.readInteger();
}
return (answer.intValue() == this.answer());
```

- Read strings typed by the user until the user enters a string that can be converted to an Integer.
- Exit the while statement and then return whether it is the correct answer or not.

19

while in class TextMenu

```
index = 0;
while ((index < 1)|(index > this.size)) {
  this.display();
  choice = Keyboard.in.readInteger();
  if (choice == null)
     index = 0;
  else
     index = choice.intValue();
}
```

- Read strings typed by the user until the user enters a string that can be converted to an Integer between 1 and some size.

20

Adventure 7

- Modify the Arithmetic Adventure game to do input validation.

21

Adventure - Changes Summary 1

- In the TextMenu class we will:
  - Add an instance variable called size which indicates how many legal entries I have.
  - Modify the constructor TextMenu().
  - Modify the instance method add().
  - Replace instance method launch().
  - Add instance method getUserSelection().

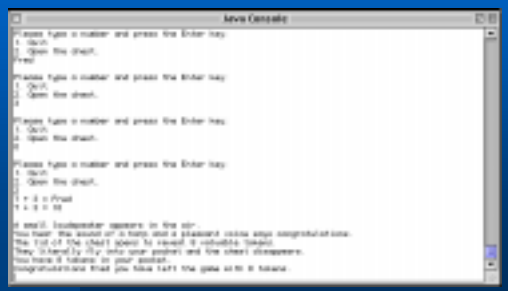
22

Adventure - Changes Summary 2

- In the Question class we will:
  - Replace the ask() method.
  - Add a display() method.
- Leave the classes: Adventure, Adventurer, RandomInt, Chest and Room unchanged.

23

Running Adventure 7



NO CHANGES

24

Class - Question 7.1

```
import java.util.*;
public class Question {
/*
  An instance of this class represents an arithmetic
  problem in the Arithmetic Adventure game.
*/

/* Constructor */
  public Question() {
  /*
     Initialize me so that I have two operands.
  */
     this.leftOperand = Question.generator.next(Question.maxOperand);
     this.rightOperand = Question.generator.next(Question.maxOperand);
  }
```

OLD

25

Class - Question 6.2

```
  /* Instance Methods */
  public boolean ask() {
  /*
     Pose myself. Return true if the user's response
     was correct and false otherwise.
  */
     Integer     answer;

     System.out.print(this.leftOperand);
     System.out.print(" + ");
     System.out.print(this.rightOperand);
     System.out.print(" = ");
     answer = Keyboard.in.readInteger();
     return answer.intValue() == this.answer();
}
```


NEW

26

Class - Question 7.2

```
  /* Instance Methods */
  public boolean ask() {
  /*
     Pose myself. Return true if the user's response
     was correct and false otherwise.
  */
     Integer     answer;

     answer = null;
     while (answer == null) {
           this.display();
           answer = Keyboard.in.readInteger();
     }
     return answer.intValue() == this.answer();
}
```


NO CHANGES

27

Class - Question 7.3

```
  public int answer() {
  /*
     Answer my correct answer.
  */
     return this.leftOperand + this.rightOperand;
  }

/* Private Static Variables */
  private static final int maxOperand = 9;
  private static final RandomInt
     generator = new RandomInt(2);
/* Private Instance Variables */
  private int leftOperand;
  private int rightOperand;
```


NEW

28

Class - Question 7.4

```
/* Private Instance Methods */

  public void display() {
  /*
     Display myself.
  */
     System.out.print(this.leftOperand);
     System.out.print(" + ");
     System.out.print(this.rightOperand);
     System.out.print(" = ");
  }
```


OLD

29

Class - TextMenu 6.1

```
import java.io.*;
import java.util.*;
public class TextMenu {
/*
  An instance of this class displays a list of strings for
  the user and allows the user to pick one. For now, up to
  five entries are supported.
*/
/* Contructor */

  public TextMenu() {
  /*
     Initialize me with no entries.
  */
  }
```


NEW

30

Class - TextMenu 7.1

```
import java.io.*;
import java.util.*;
public class TextMenu {
/*
  An instance of this class displays a list of strings for
  the user and allows the user to pick one. For now, up to
  five entries are supported.
*/
/* Contructor */

  public TextMenu() {
  /*
     Initialize me with no entries.
  */
     this.size = 0;
  }
```

OLD

31

Class - TextMenu 6.2

```
/* Instance Methods */
  public void add(String entry) {
  /*
      Add the given String to me as my next choice.
  */
      if (entry1 == null) {
            this.entry1 = entry;
            return;
      }
      if (entry2 == null) {
            this.entry2 = entry;
            return;
      }
  //more of the same for entries 3, 4 and 5.
  }
```


NEW

32

Class - TextMenu 7.2

```
/* Instance Methods */
  public void add(String entry) {
  /*
      Add the given String to me as my next choice.
  */
      this.size = this.size + 1;
      if (entry1 == null) {
            this.entry1 = entry;
            return;
      }
      if (entry2 == null) {
            this.entry2 = entry;
            return;
      }
  //more of the same for entries 3, 4 and 5.
  }
```


OLD

33

Class - TextMenu 6.3

```
  public String launch() {
  /*
      Display myself and answer the String entry selected
      by the user.
  */
      Integer     choice;
      int         index;

      this.display();
      choice = Keyboard.in.readInteger();
      if (choice == null)
            return this.entry1;
      index = choice.intValue();
```


OLD

34

Class - TextMenu 6.4

```
      switch (index) {
            case 1: return this.entry1;
            case 2: return this.entry2;
            case 3: return this.entry3;
            case 4: return this.entry4;
            case 5: return this.entry5;
            default: return this.entry1;
      }
  }
```


NEW

35

Class - TextMenu 7.3

```
  public String launch() {
  /*
      Display myself and answer the String entry selected
      by the user.
  */
      String      action;
      int         index;

      index = this.getUserSelection();
```


NEW

36

Class - TextMenu 7.4

```
      switch (index) {
            case 1: action = this.entry1; break;
            case 2: action = this.entry2; break;
            case 3: action = this.entry3; break;
            case 4: action = this.entry4; break;
            case 5: action = this.entry5; break;
            default: action = "";
      }
      return action;
  }
```

OLD

37

Class - TextMenu 6.5

```
/* Private Instance Variables */

  private String entry1;
  private String entry2;
  private String entry3;
  private String entry4;
  private String entry5;

/* Private Instance Methods */
```


NEW

38

Class - TextMenu 7.5

```
/* Private Instance Variables */

  private String entry1;
  private String entry2;
  private String entry3;
  private String entry4;
  private String entry5;
  private int size;

/* Private Instance Methods */
```


NO CHANGE

39

Class - TextMenu 7.6

```
  private void display() {
  /*
      Display myself on the screen.
  */
      String      entry;
      int         index;
      System.out.println();
      System.out.println("Please type a number and press the Enter key:");
      if (this.entry1 != null) {
            System.out.print("1. ");
            System.out.println(this.entry1);
      }
      // same code for entry2, entry3, entry4 and entry5
  }
```


NEW

40

Class - TextMenu 7.7

```
   private int getUserSelection() {
  /*
     Query the user for an action and answer the index of
     the choice. If the user does not answer with a valid
     action, query again.
  */

     Integer     choice;
     int         index;

     index = 0;
```


NEW

41

Class - TextMenu 7.8

```
  while ((index < 1) || (index > this.size)) {
     this.display();
     choice = Keyboard.in.readInteger();
     if (choice == null)
           index = 0;
     else
           index = choice.intValue();
  }
  return index;
```


42

The Rest of the Classes are Omitted

- The rest of the classes are omitted to save space.
- See Lecture 15 for a listing.