Lambda Expressions

- Example of a lambda expression: A function that gets the area of a triangle object

```
(Object obj) -> {
                CSquare sqareObjct = (CSquare) obj;
                return sqareObjct.getWidth() * sqareObjct.getWidth();
        };
```

Lambda Expressions (cont)

- In Java, a lambda expression cannot stand alone.
- It must be assigned to a variable whose type is a functional interface:

```
MeasureInterface squareMeasurer = (Object obj) -> {
                CSquare sqareObjct = (CSquare) obj;
                return sqareObjct.getWidth() * sqareObjct.getWidth();
          };
```

- Now the following actions occur:
  - A class is defined that implements the functional interface.
  - The single abstract method is defined by the lambda expression.
  - An object of that class is constructed.
  - The variable is assigned a reference to that object.

Lambda Expressions (cont)

- Then the parameter variable to the calcAverage function is initialized by using the object:

```
MeasureInterface squareMeasurer = (Object obj) -> {
                CSquare sqareObjct = (CSquare) obj;
                return sqareObjct.getWidth() * sqareObjct.getWidth();
        };

CSquare[] squares = { new CSquare(1), new CSquare(2),
                      new CSquare(3), new CSquare(4) };

double average = calcAverage(squares, squareMeasurer);
System.out.printf("The average is: %.2f", average);
```

Inner Classes

- Trivial class can be declared inside a method:

```
public class MeasurerTester
{
      public static void main(String[] args)
      {
              public class MeasureSquare implements MeasureInterface
              {
                      . . .
              }
              . . .
              MeasureInterface sqrMeasurer = new MeasureSquare();

              CSquare[] squares = { new CSquare(4), new CSquare(3),
                                    new CSquare(6) };

              double average = calcAverage(squares, sqrMeasurer);
              System.out.printf("The average is: %.2f", average);

              . . .
      }
}
```

- An inner class is a class that is declared inside another class.

Inner Classes

- We can also declare inner class inside an enclosing class, but outside its methods.
  - It is available to all methods of enclosing class:

- Compiler turns an inner class into a regular class file with a strange name:
  MeasurerTester$1AreaMeasurer.class

- Inner classes are commonly used for utility classes that should not be visible elsewhere in a program.

Inner Classes

```
public class MeasurerTester
{
      public class MeasureSquare implements MeasureInterface
      {
                  . . .
      }

      public static void main(String[] args)
      {
            MeasureInterface sqrMeasurer = new MeasureSquare();

            CSquare[] squares = { new CSquare(4), new CSquare(3),
                                  new CSquare(6) };

            double average = calcAverage(squares, sqrMeasurer);
            System.out.printf("The average is: %.2f", average);
            . . .
      }
}
```

Self Check 10.21

- When would we place an inner class inside a class but outside any methods?
- Answer: When the inner class is needed by more than one method of the classes.

Mock Objects

- Problem: Want to test a class before the entire program has been completed.
- A mock object provides the same services as another object, but in a simplified manner.
- If you just want to practice arranging the Christmas decorations, you don't need a real tree
  - Similarly, when you develop a computer program, you can use mock objects to test parts of your program.

Mock Objects

- Want to test GradingProgram without having a fully functional GradeBook class.
  - Declare an interface type with the same methods that the GradeBook class provides
- Convention: use the letter I as a prefix for the interface name
- Example: a grade book application, GradingProgram, manages quiz scores using class GradeBook with methods:
  - public void addScore(int studentId, double score)
  - public double getAverageScore(int studentId)
  - public void save(String filename)

```
public interface IGradeBook
{
      void addScore(int studentId, double score);
      double getAverageScore(int studentId);
      void save(String filename);
      . . .
}
```

Mock Objects

- Both the mock class and the actual class implement the same interface.
  - The GradingProgram class should only use this interface as a parameter
    - never the GradeBook class which implements this interface.

- Meanwhile, provide a simplified mock implementation, restricted to the case of one student and without saving functionality:

Mock Objects

```
public class MockGradeBook implements IGradeBook
{
      private ArrayList<Double> scores;
      public MockGradeBook() { scores = new ArrayList<Double>(); }

      public void addScore(int studentId, double score)
      {  // Ignore studentId
          scores.add(score);
      }

      public double getAverageScore(int studentId)
      {    double total = 0;
            for (double x : scores) { total = total + x; }
            return total / scores.size();
      }

      public void save(String filename)
      {
            // Do nothing: it is a mock class
      }
      . . .
}
```

Mock Objects

- Now construct an instance of MockGradeBook and use it immediately to test the GradingProgram class.
- When you are ready to test the actual class, simply use a GradeBook instance instead.

Self Check 10.24

- Why is it necessary that the real class and the mock class implement the same interface type?

- Answer: You want to implement the GradingProgram class in terms of that interface so that it doesn't have to change when you switch between the mock class and the actual class.

Thank you

Please let me know if you have any questions.

Chapter 11 – Input/Output and Exception Handling

Dr Kafi Rahman
Assistant Professor @CS
Truman State University

Reading and Writing Text Files - Reading

- Use Scanner class for reading text files To read from a disk file:
- Construct a File object representing the input file
  - File inputFile = new File("input.txt");
- Use this File object to construct a Scanner object:
  - Scanner in = new Scanner(reader);
- Use the Scanner methods to read data from file
  - next, nextInt, and nextDouble

Reading and Writing Text Files - Reading

- A loop to process numbers in the input file:

```
while (in.hasNextDouble())
{
   double value = in.nextDouble();
   // Process value.
}
```

Reading and Writing Text Files - Writing

- To write to a file, construct a PrintWriter object:
  - PrintWriter out = new PrintWriter("output.txt");
  - out.println("Hello, World!");
  - out.printf("Total: %8.2f\n", total);

- If file already exists, it is emptied before the new data are written into it.
- If file doesn't exist, an empty file is created.
- Use print and println to write into a PrintWriter:
- You must close a file when you are done processing it:
  - in.close();
  - out.close();
- Otherwise, not all of the output may be written to the disk file.
- Should specify "UTF-8" as the second parameter when constructing a Scanner or a PrintWriter.

FileNotFoundException

- When the input or output file doesn't exist, a FileNotFoundException can occur.
- To handle the exception, label the main method like this:
  - public static void main(String[] args) throws FileNotFoundException

Reading and Writing Text Files - Example

- Read a file containing numbers
- Sample input
  - 32 54 67.5 29 35 80 115 44.5 100 65
- Write the numbers in a column followed by their total Output from sample input

```
Sample input:
32.00
54.00
67.50
29.00
35.00
80.00
115.00
44.50
100.00
65.00

Sample output:
Total: 622.00
```

Section 1: Total.java

```java
public static void main(String[] args) throws FileNotFoundException
{ // Prompt for the input and output file names
      Scanner console = new Scanner(System.in);
      System.out.print("Input file: ");
      String inputFileName = console.next();
      System.out.print("Output file: ");
      String outputFileName = console.next();

      // Construct the Scanner and PrintWriter objects for reading and writing
      File inputFile = new File(inputFileName);
      Scanner in = new Scanner(inputFile);
      PrintWriter out = new PrintWriter(outputFileName);

      // Read the input and write the output
      double total = 0;
      while (in.hasNextDouble()) {
            double value = in.nextDouble();
            out.printf("%15.2f%n", value);
            total = total + value;
      }

      out.printf("Total: %8.2f%n", total);

      in.close();
      out.close();
}
```

Self Check 11.1

- What happens when you supply the same name for the input and output files to the Total program?
  - Let us review the program

Self Check 11.1

- What happens when you supply the same name for the input and output files to the Total program?
  - Let us review the program

- Answer: When the PrintWriter object is created, the output file is emptied.
  - Sadly, that is the same file as the input file.
  - The input file is now empty and the while loop exits immediately.

Self Check 11.2

- What happens when you supply the name of a nonexistent input file to the Total program?
  - Let us review the program

Self Check 11.2

- What happens when you supply the name of a nonexistent input file to the Total program?
  - Let us review the program

- Answer: The program throws a FileNotFoundException and terminates.

Self Check 11.3

- Suppose you wanted to add the total to an existing file instead of writing a new file. Self Check 1 indicates that you cannot simply do this by specifying the same file for input and output. How can you achieve this task? Provide the pseudocode for the solution.

- Answer:

```
Open a scanner for the file.
For each number in the scanner
    Add the number to an array.
Close the scanner.
Set total to 0.
Open a print writer for the file.
For each number in the array
    Write the number to the print writer.
    Add the number to total.
Write total to the print writer.
Close the print writer.
```

Self Check 11.4

- How do you modify the program so that it shows the average, not the total, of the inputs?

- Answer: Add a variable count that is incremented whenever a number is read.
- At the end, print the average, not the total, as the following
    - `out.printf("Average: %8.2f\n", total / count);`

Text Input and Output

- The next method of the Scanner class reads a string that is delimited by white space.
- A loop for processing a file:

```
while (in.hasNext())
{
    String input = in.next();
    System.out.println(input);
}
```

- If the input is "Mary had a little lamb", the loop prints each word on a separate line
  - Mary
  - had
  - a
  - little
  - lamb

Text Input and Output

- The next method returns any sequence of characters that is not white space.
- White space includes: spaces, tab characters, and the newline characters that separate lines.
- These strings are considered "words" by the next method
    - snow.
    - 1729
    - C++
- When next is called:
    - Input characters that are white space are consumed - removed from the input
    - They do not become part of the word
    - The first character that is not white space becomes the first character of the word
    - More characters are added until
        - Either another white space character occurs
        - Or the end of the input file has been reached
- If the end of the input file is reached before any character was added to the word a "no such element exception" occurs.

Thank you

Please let me know if you have any questions.

Questions?