420-B10

Test 3 Review

Topics:

* formatting output
* the while statement
* constructors
* reading from a data file
* GUI programming

Terminology:

* formal parameter
* actual argument
* constructor
* sentinel
* priming read

# Formatting Output

## Write a program that reads two double precision numbers, x and y, divides x by y and uses the NumberFormat class to print the resulting quotient in the following formats:

* a decimal number with 4 significant digits
* a percentage with 2 significant digits
* a dollar amount.

For example, a sample run might look like:

Enter two double precision numbers: 34 79

Quotient as a decimal with 4 significant digits: 0.4304

Quotient as a percent with 2 significant digits: 43.04%

Quotient as a dollar amount: $0.43

## Inport java.util.scanner;

Public class quotientCalculator {

## Public static void main (String [] args) {

## Scanner blarg = new Scanner(System.in);

## double x, y;

## System.out.println(“Enter two double precision numbers: “);

x = blarg.nextDouble();

y = blarg.nextDouble();

System.out.println(“Quotient as a decimal with 4 significant digits: “

+ x / y );

System.out.println(“Quotient as a percent with 2 signigicant digits: “

+ (x / y) \* 100 );

System.out.println(“Quotient as a dollar amount: “

+ (x / y) );

} //main

}

## Use a **printf** statement to print a line of a report containing **studentName**, **courseNumber**, **mark** and **classAverage** given the following declarations:

**String studentName;**

**String courseNumber;**

**int mark;**

**double classAverage;**

import java.util.Scanner;

public class something {

public static void main(Strings [] args) {

Scanner iHateThis = new Scanner();

String studentName, courseNumber;

double classAverage;

int mark;

System.out.printf(“%20s” , studentNumber);

System.out.printf(“%10s” , courseNumebr);

System.out.printf(“%20i” , mark);

System.out.printf(“%10d” , classAverage);

}

}

Space the data out as follows:

**studentName** – 20 spaces left aligned

**courseNumber** – 10 spaces left aligned

**mark** – 10 spaces right-aligned

**classAverage** – 10 spaces right-aligned; with 2 significant digits

# The while statement

## Correct the errors in the following **while** statements:

int num = 0;

while (num < 5){

System.out.println("num is " + num);

++num;

}

int num = 50;

while (num > 0)

num += 5;

System.out.println("The final value of num is " + num);

## Write a **while** statement that adds up all the integers between 0 and 1000.

int num, currentSum;

currentSum = 0;

while (num <= 1000){

System.out.println(num + currentSum);

currentSum += num;

num++;

}

## Write a **while** statement that reads a list of integers from the user and finds the product of the numbers read. Stop reading when 0 is entered. Assume that a Scanner called keyboard has been defined.

Import java.util.Scanner;

Public class thisDoesSomething {

Public static void main(Strings [] args) {

Scanner reader = new Scanner(System.in)

int num, sum;

sum = 0;

while (num != 0) {

System.out.println(“please enter a number to add. Press 0 to exit. ”);

num = reader.nextInt();

sum += num;

}

}

}

## Write a code segment for the following loops:

### a **while** loop to display the odd integers in the range 1 to 29.

### a **while** loop to display the squares of even integers in the range 2 to 20

## Write a loop to simulate 600 throws of a dice. Count the number of times each of the sides is rolled. The output from the program might look like:

1 was rolled 96 times.

2 was rolled 91 times.

3 was rolled 90 times.

4 was rolled 120 times.

5 was rolled 110 times.

6 was rolled 94 times.

## Use a **while** loop to find the smallest integer **n** such that **n2** is greater than 12000.

## Use nested loops to print the following patterns:

1

1 2

1 2 3

1 2 3 4

1 2 3 4 5

1 2 3 4 5 6

1 2 3 4 5 6

1 2 3 4 5

1 2 3 4

1 2 3

1 2

1

## Write a **while** loop to validate an input value, **station**, such that it is between 87.8 to 108.0.

## Write a program segment that sets a Boolean variable **dangerous** to **true** and stops reading data if **pressure** (a **double** variable being read in) exceeds 510.0. Use **dangerous** as a flag to control the loop.

# Constructors

## Code two constructors for the **Movie** class shown here. The first constructor should be a default constructor and initialize **movieTitle** and **genre** to “*Unknown*” and **yearOscarWon** to *0*. The second constructor should have 3 parameters – one for each instance variable. The instance variables should be initialized to the corresponding constructor parameters.

**public** **class** Movie

{

**private** String movieTitle;

**private** **int** yearOscarWon;

**private** String genre;

**public** String getMovieTitle()

{

**return** movieTitle;

} // getMovieTitle()

**public** **void** setMovieTitle(String title)

{

movieTitle = title;

} // setMovieTitle(String)

**public** **int** getYearOscarWon()

{

**return** yearOscarWon;

} // getYearOscarWon()

**public** **void** setYearOscarWon(**int** year)

{

yearOscarWon = year;

} // setYearOscarWon(**int)**

**public** String getGenre()

{

**return** genre;

} // getGenre()

**public** **void** setGenre(String type)

{

genre = type;

} // setGenre(String)

} // Movie class

# Reading from a Data File

## The "bestPictures.txt" file contains a list of all the movies that have won best picture at the Academy Awards. For each movie, it lists the year it won, the movie title and the genre of the movie. Each field is separated by a "~". For example the first two lines of the file are:

**1928~Wings~war**

**1929~Broadway Melody~musical**

## Write a method called **readBestMovies()**. It should have a String parameter representing the genre and does not return anything.

### open a Scanner object to read "bestPictures.txt"

### read each record in the file.

### if the genre is equal to the parameter, display the year made and the movie title.

# GUI Programming

## Write a frame called **BestMovieFrame**. It should have a label, a text field, two buttons and a display area. The display area should be in a scroll pane. The text field should be size 10, the display area should be 13 rows by 30 columns and the frame should be 300 by 325. The frame should look like the sample below. If the user clicks the “All Genres” button, the **displayAllMovies()** method should be called. If the user clicks the “One Genre” button, the **displayMovies(String** *genre***)** method should be called. The text in the text field should be used as a parameter to the method. If there is no data in the text field when the “One Genre” button is clicked display an error message in the text area. Assume that the **displayAllMovies()** and **displayMovies(String** *genre***)** exist. You do not have to write them.

