Introduction to Programming using Java

Dr. Jey Veerasamy
jeyv@utdallas.edu

July 31st – August 23rd
9:30 am to 12 noon

Logistics

- Laptops – work with your neighbor if you did not bring a laptop
- Restrooms – go right when you go out of TI auditorium
- Break : 10:45am to 11am – I will use this time to provide extra help too.
- Cell-phones – poor signal within the classrooms - switch off to avoid distractions and battery drain.
- Signup sheet will be there for every class – please sign-in. You are welcome to bring your friends since we have plenty of additional seats!

Instructor: Dr. Jey Veerasamy

- Dad was a school teacher
- Completed M.S. and Ph.D. in UT Dallas in 1999
- 16 years of telecom software industry experience in Nortel and Samsung
- Taught as adjunct and online faculty in several colleges along with full-time work.
- Returned back to UT Dallas as full-time teaching faculty in Fall 2010.

UTD

CS Department: Highlights

- The UTD CS dept started as a small program within the Mathematical Sciences in the 70s
- One of the largest CS dept's in the US today
- 55 faculty members
- 120+ Research and Teaching Assistants
- 15 Staff members including 4 Tech. Support
- 1500+ Students (130 Ph.D. +700 MS +720 BS)
- Full range of programs in CS, SE and TE:

  -- BS, MS and Ph.D.

UTD

CS Department: Accomplishments

- Ranked 29th in UC Irvine's publications ranking of CS graduate programs
- Ranked 24th worldwide in UC Irvine's publications ranking of SE graduate programs
- 8 of our faculty hold Young Investigator awards
- Top 5 producer of CS degrees
- Placed 14th worldwide in ACM Programming Competition (just behind MIT & CalTech in US)

UTD

CS Department: Distinguished Faculty

- Over 55 memberships on editorial boards of computer science journals
- Research expenditure over $16 million in last two years
- Published 250+ papers last year
- Involved in numerous leading technical conferences as conference chairs or program committee chairs/members

What is programming?

- Developing software applications & games
- Software is not limited to PC
  - most complex systems run software
  - smart phones, game devices, even DVD players

Programming …

- is NOT a boring or repetitive activity
- does NOT require you to sit in dark room and type in computer all day!
- does NOT involve complex Math

- requires logical thinking – technical common sense
- write minimal code & combine with existing components to build new applications
- Solve customers' problems & improves quality of life for every one.

Why learn programming?

- Software Engineers get great pay!
- Less stressful compared to several other high paying jobs – room for trial & error
- Automation continues…
- Computer touches our lives more & more every day…
- More component based programming → always room for simple programs to do large tasks!

Analogy for learning to program: Learning to ride bicycle

- Difficulties for beginners:
  - Learning to balance & go forward together

- Difficulties for experienced folks:
  - None.

Learning to program: Difficulties for beginners

1. Syntax errors

- struggle for hours to fix syntax errors
- Loose confidence
- Frustrating experience
- Run away & never come back if possible!

2. Logic errors

Logic is simple for small programs. It can be an issue if student has mental block against math.

Difficulties for experienced programmer?

NOT syntax errors – it is just a nuisance!

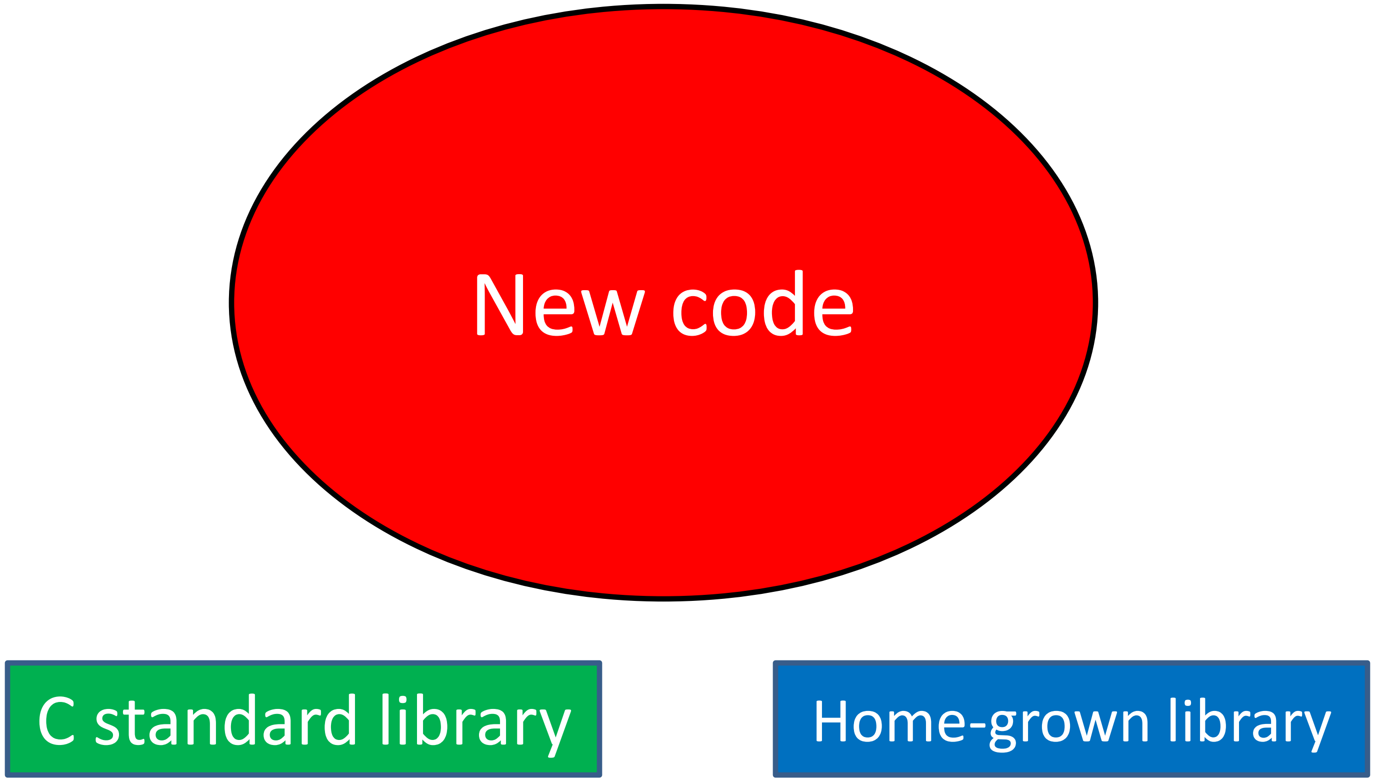
More worried about logic errors (aka SW bugs) that are hard to reproduce.

Continuous learning

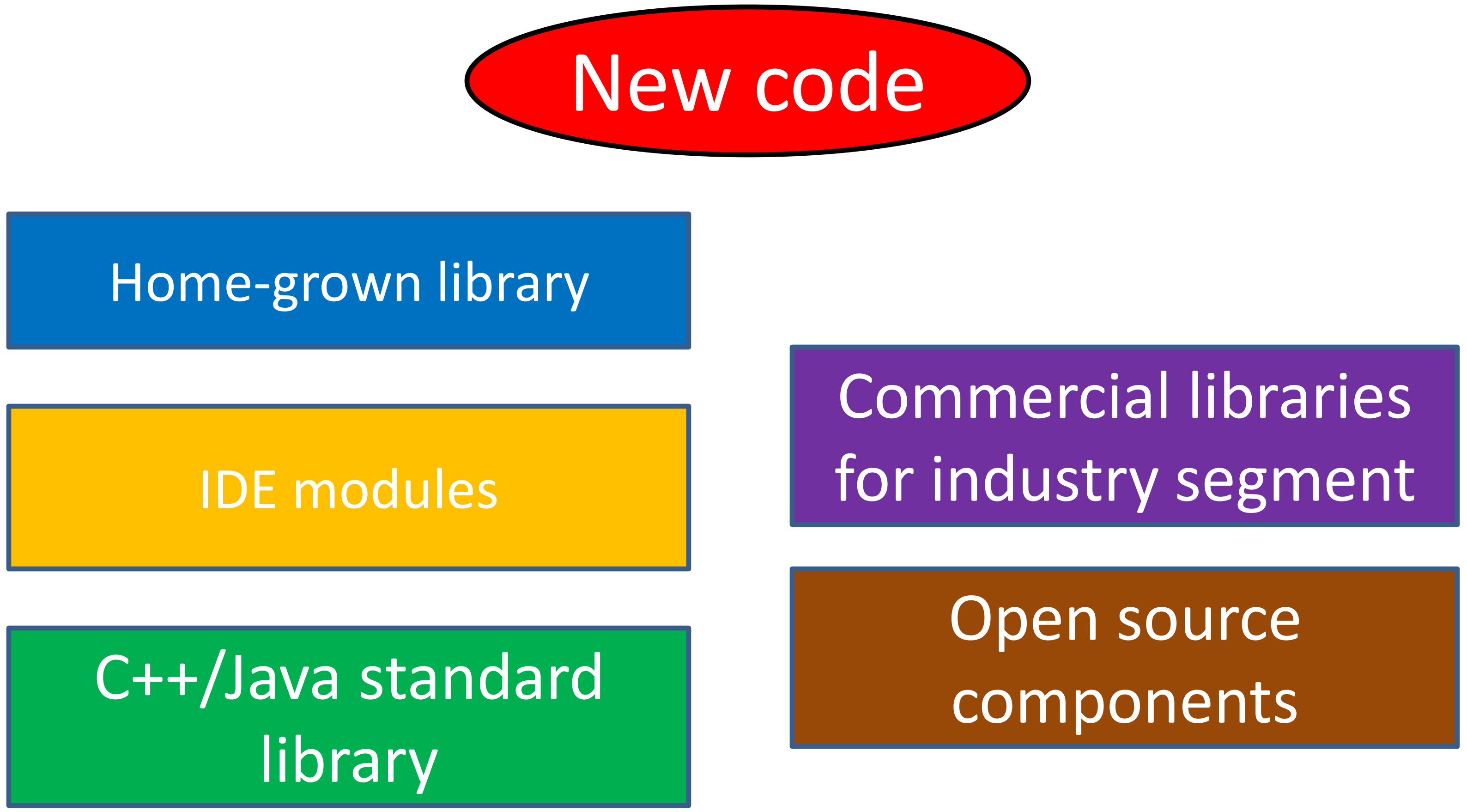
How to reduce difficulties for beginners?

- Use the “start of the art” tools like Netbeans IDE (Integrated Development Environment) to help us!
- Few other IDEs are Eclipse, JGRASP, … (Search for “Java IDE” in the web to learn more)
- IDEs take care of mundane steps so that we can focus on learning and programming.
- Also, take advantage of expanded libraries provided by new languages and use them as building blocks.

A typical software project development in 1990



Same project NOW


New code
Home-grown library
IDE modules
C++/Java standard library
Commercial libraries for industry segment
Open source components

A few examples

- Recipe to make your favorite food
- Assembly instructions for a toy
- Coming to college from home

What is common about these activities?

A few examples

- Recipe to make your favorite food
- Assembly instructions for a toy
- Coming to college from home

What is common about these activities?

Sequence

Programming concepts: Sequence structure

```
instruction 1;
instruction 2;
instruction 3;
…
```

NetBeans IDE – getting started

- Start the tool
- Click on new Project icon in top toolbar
- Java category and Java Application have been pre-selected. Click on Next
- Use a meaningful project name for each project/program. Click on Finish.
- It will add a Java source file automatically with a skeleton code.

Sample skeleton code

```
package hello;

/**
 *
 * @author veerasam
 */
public class Hello {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
    }
}
```

Program to print Hello!

```
package hello;

import java.util.*;

public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello to Java!");
    }
}
```

Comments have been removed to conserve space. Assumes project name “hello”

Few notes

- Compiler translates the program to binary executable.
- NetBeans features automatic incremental compilation – syntax errors appear as you type in.
- It is good to keep the code formatted properly (indentation). Right-click within the editor any time and select Format.
- Comments are ignored by the compiler. Comments are used for recording ideas/thoughts in plain English so that we can make sense of the code later.
- // is used for one line comment, /* …. */ is used multi-line comments.
- For initial sessions, almost all our code will go into main() method. Do not change anything else.
- Java is case-sensitive. Example: int and Int are treated differently.

Structure for simple programs

- Input – get the necessary user input
- Processing – do some computation
- Output – show the results to the user

Problem:
get 5 numbers and output average

```
Enter 5 numbers:
11
12
12
14
15
Average is 12.2
```

Program output in GREEN, user input in BLUE

Idea/pseudocode: get 5 numbers and output average

```
Prompt & get the score for number1
Prompt & get the score for number2
Prompt & get the score for number3
Prompt & get the score for number4
Prompt & get the score for number5
average = (number1 + number2 + number3 + number4 + number5) / 5
output average
```

Idea/pseudocode - why?

- As the problems become bigger, it is harder to code directly from the problem description.
- It is better to capture the logic first, build confidence, then convert it to actual code.
- Pseudocode is for human understanding, so plain English is preferred. It can use indentation and language constructs like IF, WHILE, FOR, … but no need to follow any language syntax specifics.
- Can contain just high level ideas or detailed instructions that is equivalent to actual code.
- Another option is to use Flowcharts, but it occupies too much space & it cannot be stored as comments within the source files.

Java program

```
package add5;

import java.util.*;

public class Add5 {

    public static void main(String[] args) {

        Scanner keyboard = new Scanner(System.in);

        System.out.print("Enter 5 numbers: ");
        int number1 = keyboard.nextInt();
        int number2 = keyboard.nextInt();
        int number3 = keyboard.nextInt();
        int number4 = keyboard.nextInt();
        int number5 = keyboard.nextInt();

        double average = (number1 + number2 + number3 + number4 + number5) / 
5.0;

        System.out.println("Average is " + average);
    }
}
```

Comments have been removed to conserve space. Assumes project name “add5”

Variables

- Placeholders to store values, similar to variables we use in math equations. Names should start with a letter, then they can contain numbers.
- Popular variable types in Java are
  - int to store integer values
  - double to store real numbers (contains fractions, also too huge or too small values)
  - String to store strings typically used for messages
  - Other data types: byte, char, boolean, float so on.

http://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html

Basic/Primitive Data Types

- Primitive data types are built into the Java language and are not derived from classes.
- There are 8 Java primitive data types.
  - `byte`
  - `short`
  - `int`
  - `long`
  - `float`
  - `double`
  - `boolean`
  - `char`

Numeric Data Types

byte	1 byte	Integers in the range -128 to +127 (-2^7 to 2^7-1)
short	2 bytes	Integers in the range of -32,768 to +32,767 (-2^{15} to 2^{15}-1)
int	4 bytes	Integers in the range of -2,147,483,648 to +2,147,483,647 (0xFFFFFFFF to 0x7FFFFFFF) (Two's complement form to handle negative numbers) -2^{31} to 2^{31}-1
long	8 bytes	Integers in the range of -9,223,372,036,854,775,808 to +9,223,372,036,854,775,807 -2^{63} to 2^{63}-1
float	4 bytes	Floating-point numbers in the range of ±3.410E-38 to ±3.410E38, with 7 digits of accuracy
double	8 bytes	Floating-point numbers in the range of ±1.710E-308 to ±1.710E308, with 15 digits of accuracy

Java program: add 5 numbers and output average - notes

- Need to use double or float to store average. int data type cannot handle fractional part.
- int / int results in integer division - returns the quotient and throws away the remainder. For example, 5 / 2 results in 2, NOT 2.5.
- To avoid integer division, at least one operand has to be real number. Easiest way is to divide the sum by 5.0 instead of 5 (as shown in the code). Another option is to use “double” for all variables.

Problem: compute weighted average

- Compute the weighted score based on individual assignments' scores. Let us say there are only 3 assignments & 2 exams, each with max score of 100. Respective weights are (10%, 10%, 10%, 35% and 35%)

Sample input & output

```
Enter score for assignment #1:
100
Enter score for assignment #2:
100
Enter score for assignment #3:
100
Enter score for exam #1:
95
Enter score for exam #2:
95
Weighted sum is 96.5%
```

Idea/Pseudocode

```
Prompt & get the score for assignment1
Prompt & get the score for assignment2
Prompt & get the score for assignment3
Prompt & get the score for exam1
Prompt & get the score for exam2
weightedScore = (assignment1 + assignment2 + assignment3) * 0.1 + (exam1 + exam2) * .35
output weightedScore
```

Java program

```
package weightedsum;

import java.util.*;

public class WeightedSum {

    public static void main(String[] args) {
        Scanner keyboard = new Scanner(System.in);

        System.out.print("Enter score for assignment #1: ");
        int assign1 = keyboard.nextInt();
        System.out.print("Enter score for assignment #2: ");
        int assign2 = keyboard.nextInt();
        System.out.print("Enter score for assignment #3: ");
        int assign3 = keyboard.nextInt();

        System.out.print("Enter score for exam 1: ");
        int exam1 = keyboard.nextInt();
        System.out.print("Enter score for exam 2: ");
        int exam2 = keyboard.nextInt();

        double sum = assign1 * 0.1 + assign2 * 0.1 + assign3 * 0.1
                        + exam1 * 0.35 + exam2 * 0.35;

        System.out.println("Weighted sum is " + sum + "%" );
    }
}
```

Comments have been removed to conserve space. Assumes project name “add5”

Java program : several ways to do same computation

```
double sum = assign1 * 0.1 + assign2 * 0.1 + assign3 * 0.1
                        + exam1 * 0.35 + exam2 * 0.35;
```

can also be written as

```
double sum = 0.1 * (assign1 + assign2 + assign3)
              + 0.35 * (exam1 + exam2);
```

(or)

```
double sum = 0.1 * (assign1 + assign2 + assign3);
sum += 0.35 * (exam1 + exam2);
```

(or)

```
double sum = 0;
sum += 0.1 * (assign1 + assign2 + assign3);
sum += 0.35 * (exam1 + exam2);
```

Java program : several ways to do same computation …

(or)

```
double sum = assign1 * 0.1;
sum += assign2 * 0.1;
sum += assign3 * 0.1;
sum += exam1 * 0.35;
sum += exam2 * 0.35;
```

(or)

```
double assignWeight = 0.1; double examWeight = 0.35;

double sum = assignWeight * (assign1 + assign2 + assign3)
              + examWeight * (exam1 + exam2);
```

(or several more ways!)

Note: When variable names contain multiple words, Java convention is to camel casing – use uppercase for first letter each additional word. That is why we used variable names like examWeight.

Problem: Country Store

Let us say we have a simple store that sells only the following 5 items. Write a program to do the check-out. That is, ask the user to input the weights for each product and output the total price.

Product	**Price per pound**
Bananas	$ 0.44
Apples	$ 0.99
Cucumbers	$ 1.19
Carrots	$ 0.89
Oranges	$ 0.79

Sample input & output

```
Enter weight for Bananas:
2.5
Enter weight for Apples:
3.4
Enter weight for Cucumbers:
2.3
Enter weight for Carrots:
4.5
Enter weight for Oranges:
3.7
Total price is $ 14.13
```

Product	**Price per pound**
Bananas	$ 0.44
Apples	$ 0.99
Cucumbers	$ 1.19
Carrots	$ 0.89
Oranges	$ 0.79

Pseudocode #1

```
Prompt & get the weight for Bananas
Prompt & get the weight for Apples
Prompt & get the weight for Cucumbers
Prompt & get the weight for Carrots
Prompt & get the weight for Oranges
total = bananaWeight * 0.44 + appleWeight * 0.99 + cucumberWeight * 1.19 + carrotWeight * 0.89 + orangeWeight * 0.79
output total
```

Pseudocode #2

```
Initialize total to 0
Prompt & get the weight for Bananas
total += weight * 0.44
Prompt & get the weight for Apples
total += weight * 0.99
Prompt & get the weight for Cucumbers
total += weight * 1.19
Prompt & get the weight for Carrots
total += weight * 0.89
Prompt & get the weight for Oranges
total += weight * 0.79
output total
```

See store.java for the code.

Pseudocode #1 vs #2

- 2nd version uses minimal # of variables – reuses weight for all 5 products since individual weights are not needed after computing sub-totals.
- Both are acceptable mechanisms!

Activities

- Drive car or take DART bus?
- Party or study?
- Fly or drive?

What is the common idea for all these activities?

Activities

- Drive car or take DART bus?
- Party or study?
- Fly or drive?

What is the common idea for all these activities?

Decision or Selection

Selection structure

```
IF condition is true THEN
    do this;
ELSE
    do that;
ENDIF
```

Note: ELSE portion is optional.

Selection structure in Java

```
if (condition)
    statement;

if (condition)
    statement1;
else
    statement2;
```

```
if (condition) {
    statement1;
    …
} else {
    statement2;
    …
}
```

if statement – be careful!

```
if (condition)
     statement1;
     statement2;
```

is treated by compiler as

```
if (condition)
     statement1;
statement2;
```

Important to use { } when there are multiple statements in the body!

Problem:
compute weekly pay with a restriction

Get hourly pay rate & # of hours, compute the weekly pay, but do not pay for hours beyond 50.

Sample input/output

```
Enter hourly pay rate: 100
Enter hours: 30
Weekly pay is $ 3000

Enter hourly pay rate: 100
Enter hours: 60
Weekly pay is $ 5000
```

Pseudocode

```
Prompt & get hourly pay rate & # of hours
IF hours <= 50
    pay = hours * payRate;
ELSE
    pay = 50 * payRate;
output pay
```

Java code

```
System.out.print("Enter hourly pay rate: ");
double payRate = keyboard.nextDouble();

System.out.print("Enter # of hours: ");
double hours = keyboard.nextDouble();

double pay;

if (hours <= 50) {
    pay = payRate * hours;
} else {
    pay = payRate * 50;
}

System.out.println("Weekly pay is " + pay);
```

Note: only the relevant code is shown.

Several other ways to do same computation

```
 if (hours > 50) {
      pay = payRate * 50;
 } else {
      pay = payRate * hours;
}

(or)

if (hours > 50) {
      hours = 50;
}
pay = payRate * hours;
```

Note: { } is not required when IF statement contains only one line. It is a good habit though.

Problem: Weekly Pay Version 2

Get hourly pay rate & # of hours, compute the weekly pay as per the following table:

Hour	Actual pay rate
0 to 40	Hourly Rate
41 to 50	Hourly Rate * 1.5
Hours > 50	0

Basically, workers get paid 50% more for each hour beyond 40, but they will not be paid for hours beyond 50.

Problem: Weekly Pay Version 2

- How many tests we need to run to validate the program?

3, one for each case.

Sample input/output

```
Enter hourly pay rate: 100
Enter hours: 30
Weekly pay is $ 3000
```

```
Enter hourly pay rate: 100
Enter hours: 45
Weekly pay is $ 4750
```

```
Enter hourly pay rate: 100
Enter hours: 60
Weekly pay is $ 5500
```

Pseudocode #1

```
IF hours <= 40
        pay = hours * payRate;
ELSE IF hours <= 50
        pay = 40 * payRate + (hours – 40) *payRate * 1.5;
ELSE
        pay = 40 * payRate + 10 * payRate * 1.5;
```

Java code – chained IF statement

```
if (hours <= 40)
        pay = hours * payRate;
else if (hours <= 50)
        pay = 40 * payRate + (hours – 40) *payRate * 1.5;
else
        pay = 40 * payRate + 10 * payRate * 1.5;
```

Java code – nested if statement

```
if (hours <= 40)
        pay = hours * payRate;
else
        if (hours <= 50)
                pay = 40 * payRate + (hours – 40) *payRate * 1.5;
        else
                pay = 40 * payRate + 10 * payRate * 1.5;
```

Chained IF statement is preferred since it involves less indentation.

Pseudocode #2 – 3 IF statements

```
IF hours <= 40
        pay = hours * payRate;

IF (hours > 40) && (hours <= 50)
        pay = 40 * payRate + (hours – 40) *payRate * 1.5;

IF (hours > 50)
        pay = 40 * payRate + 10 * payRate * 1.5;
```

Pseudocode #3 – simplify equations

```
IF hours <= 40
        pay = hours * payRate;
ELSE
        basePay = 40 * payRate;
        overtimeRate = payRate * 1.5;
        IF hours <= 50
                pay = basePay + (hours – 40) *overtimeRate;
        ELSE
                pay = basePay + 10 * overtimeRate;
```

Java code #3

```
if (hours <= 40)
        pay = hours * payRate;
else {
        basePay = 40 * payRate;
        overtimeRate = payRate * 1.5;
        if (hours <= 50)
                pay = basePay + (hours – 40) *overtimeRate;
        else
                pay = basePay + 10 * overtimeRate;
}
```

Pseudocode #4

```
IF hours > 50
        hours= 50;


IF hours <= 40
    pay = payRate * hours;
ELSE
    pay = payRate * 40 + payRate * 1.5 * (hours – 40);
```

These are just a handful of ways. Several more ways are possible!

Problem: Country Store Version 2

Enhance the checkout program to apply the following discount based on final total price.

Total price	Discount
$50 and above	10%
$75 and above	15%
$100 and above	20%

Pseudocode/idea

After computing the total:

```
if (total > 100)
        apply 20%
else if (total > 75)
        apply 15%
else if (total > 50)
        apply 10%
```

Java : switch structure

```
switch (num) {
        case 0: ….
              break;
        case 1: ….
              break;
        case 2: …
                break;
        case 3: …
                break;
        default:
                …
}
```

```
if (num == 0)
        …
else if (num == 1)
        …
else if (num == 2)
        …
else if (num == 3)
        …
else
        …
```

Note: int or char is commonly used ones with switch(). Real numbers cannot be used with switch().

series of if statements vs. switch()

- case statements within switch() look bit cleaner, compared to so many IF conditions.

Problem: Math practice

Program should come up with 2 random integers (first one between 1 and 100 and second one between 1 and 20) and randomly select an operator (+, -, * or /) and post the question to the user. Get the answer and validate and output a message.

- Sample input & output:

```
45 * 15 ? 675
Very good.
```

Ideas

- Use Java’s random number generator to get numbers.
- For operator, generate random number 0 to 3, then map it to operator (+, -, *, /) using switch statement.
- See MathPractice.java for full Java code.

Activities

- Bring in tons of purchased items from car to house
- Load up truck when moving from a home
- Eat cookies from a box
- Taking an exam that has several questions

What is the common idea for all these activities?

Activities

- Bring in tons of purchased items from car to house
- Load up truck when moving from a home
- Eat cookies from a box
- Taking an exam that has several questions

What is the common idea for all these activities?

Repetition/Loop

Repetition structure (pseudocode)

```
WHILE (more items to process)
        process the next item;
ENDWHILE

FOR  month = 1 to 12
      do monthly processing
ENDFOR
```

Repetition structures in Java

```
while (condition)
        statement;

while (condition) {
        statement1;
        statement2;
        …
}

do {
        statement1;
        …
} while (condition);
```

```
for( int i=0 ; i<n ; i++ )
        statement;

for( int i=0 ; i<n ; i++ ) {
        statement1;
        statement2;
        …
}
```

while vs. do … while vs. for

- body of while loop may not execute at all!
- body of do…while loop is guaranteed to execute at least once.
- for loop is a simpler version of while loop & it is used when we know exact # of times loop needs to be executed.

Problem:
average of 5 numbers

Re-do the problem to compute the average of 5 numbers using a loop. Use minimal # of variables.

```
Enter the numbers:
91
92
92
93
94
Average is: 92.4
```

Idea

- Use a loop to get 5 numbers and add them up.
- Since we know the count upfront, for loop is preferred.
- See add5while.java and add5for.java for the code.

Problem:
compute average for any input list

Let us say you want to compute the average score of a class, but you do not know # of students in the class! What can you do?

Problem:
compute average of any input list

Let us say you want to compute the average score of a class, but you do not know # of students in the class! What you will do?

Use out-of-range value like -1 to indicate the end of input.

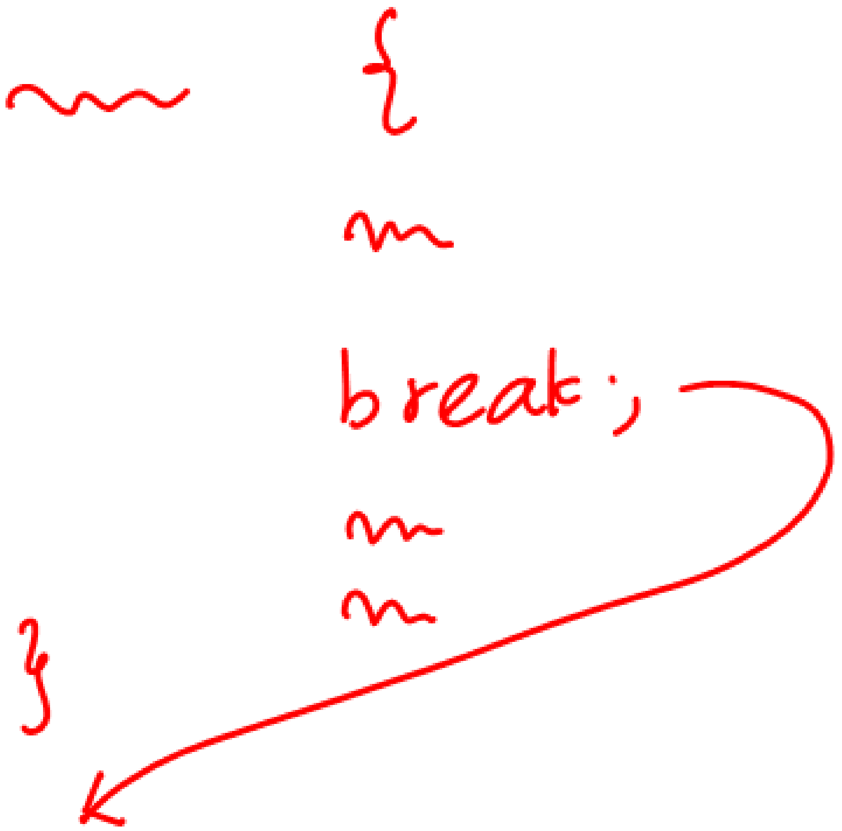
```
Enter the numbers:
91
92
93
94
-1
Average is: 92.5
```

Idea

- Repeat the loop until -1 is seen as input.
- Keep track of # of input items
- Compute the average as total / count

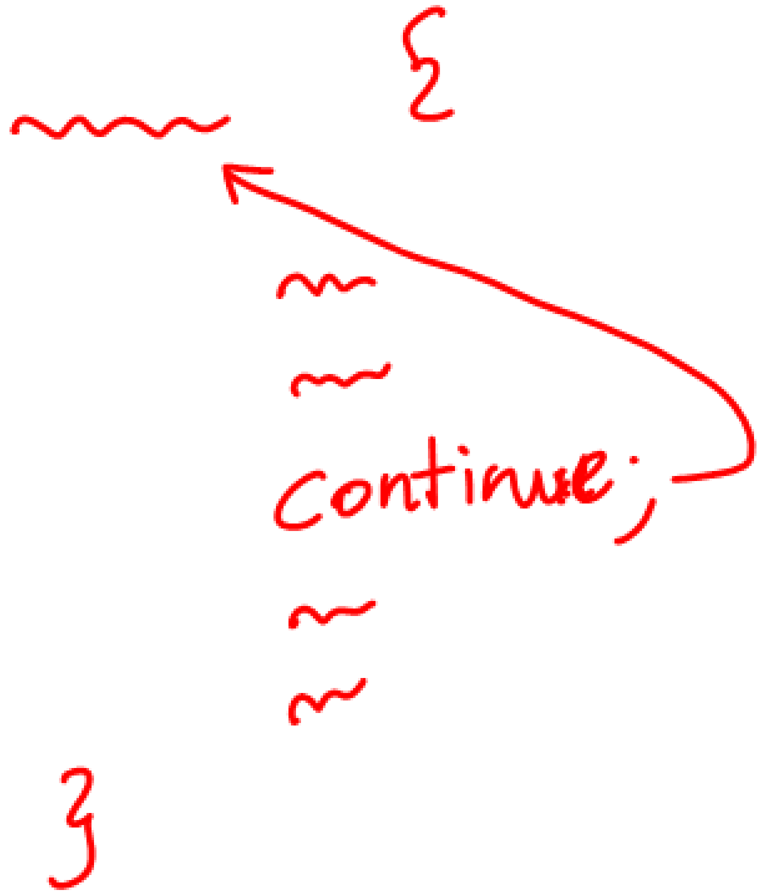
break statement

- breaks the loop and continues to the statement after the loop body:



continue statement

- Ignores the lines below that statement and continues with the loop.



Problem: Math Practice - Version 2

Make the user answer 10 questons and keep track of user’s performance. Output the final score.

Here is a sample message after answering 10 questions:

```
You got 7 correct and 3 wrong. Play again soon!
```

Ideas

- use for loop to repeat 10 times
- use loop variable as question #
- use 2 variables to keep track of correct/incorrect – increment as needed
- print final stats (# correct, # incorrect)

Problem: Math Practice - Version 3

Same as Version 2, but uses additional method for playing the game.

See the code for details.

For advanced level students only

- Let us say we want to control the distribution of questions per operator. For example, let us say we want addition problems for ~35% of the time, subtraction problems for another ~35% of the time, multiplication problems for ~20% of the time, and integer division problems for remaining ~10%.
- We can even make it more generic: We can prompt & get those % values from the user, then we can try to setup the distribution of questions accordingly.
- I will be happy to discuss your ideas in the class after each session is over (after 12 noon).

Problem: Country Store Version 3

Change the input mechanism for the store – list all 5 products every time, let the user select a product, then enter the weight. Keep adding the purchase to total, repeat the prompt until the user is done.

Country Store Version 3 : Prompt

```
Available products:
1.  Bananas ($ 0.44 / lb)
2.  Apples ($ 0.99 / lb)
3.  Cucumbers ($ 1.19 / lb)
4.  Carrots ($ 0. 89 / lb)
5.  Oranges ($ 0.79 / lb)
Enter selection (0 to finish check-out) : 2
Enter weight: 2.45
```

Guessing game

Pair up with your neighbor and play this game:

Think of a number between 1 and 100. Ask your neighbor to guess that number.

Repeat the following steps as many times as needed:

- Neighbor asks, “Is it NN?”
- You respond with “yes!” or “go lower” or “go higher”

Goal is to ask minimum # of questions.

Guessing game – ideas?

- Ask about the middle value
- Based on the response, we can focus on one-half of the range.
- Repeat the top 2 steps until you say “yes!”

Let the computer find your number: Guessing game

Think of a number between 1 and 100. Write a program so that the computer will ask you a series of questions and determine that number based on your answers.

Repeat the following steps as many times as needed:

- Computer asks, “Is it NN?”
- User responds with <, =, or >

Guessing game : Sample runs

```
Is it 50?
<
Is it 25?
>
Is it 37?
>
Is it 43?
=
Good game!
```

```
Is it 50?
<
Is it 25?
<
Is it 12?
>
Is it 18?
>
Is it 21?
<
Is it 19?
>
Your number is 20. Good game!
```

Pseudocode

- Initialize range (low = 1, high = 100)
- while (true)
  - compute mid = (low + high) / 2
  - ask the user
  - user responds with <, >, =
    - String input = keyboard.next();
  - = → we are done!
    - if (input.equals("<"))
  - < → high = mid-1   // go into first half.
  - > →low = mid+1 // go into second half.

Ideas for coding

- Get the user input as a String.

String input = keyboard.next();

- Since String is a complex data type, it needs to be compared like

 (input.equals("<"))

- You can also check the first character of the string alone:

(input.charAt(0) == '<‘)

Reverse Guessing game

Let the computer think of a number between 1 and 100 (In other words, generate a random number from 1 to 100 range). Write a program so that the computer will respond to your guesses until the number is guessed.

Repeat the following steps as many times as needed:

- You say, “NN”
- Computer responds with “Yes! Good job!!”, “go lower!” or “go higher!”

Reverse Guessing game : Sample runs

```
Enter your guess: 80
go higher!
Enter your guess: 95
go lower!
Enter your guess: 90
Yes! Good job!!
```

```
Enter your guess: 20
go higher!
Enter your guess: 60
go lower!
Enter your guess: 40
go higher!
Enter your guess: 45
go higher!
Enter your guess: 50
Yes! Good job!!
```

Pseudocode

- Computer thinks of a number – uses random number generator
  - Random generator = new Random();
  - int number = generator.nextInt(100) + 1
- while (user has not guessed it correctly yet)
  - get user’s guess
  - compare and output appropriate message
    - if (guess == number)
    - if (guess < number)
    - if (guess > number)

Reverse Guessing game Version 2

What is the point of playing a game if it does not output points? ☺ Let us enhance the reverse guessing game to output the # of points based on your performance.

# of guesses	Points
1	100
2	50
3	35
4	25
5	20
6 and above	16 - # of guesses, but do not go negative.

Ideas

- have a variable count to keep track # of guesses
- use switch() statement in the bottom to convert # of guesses to actual points.

For more details

- Java language basics : official tutorial

http://docs.oracle.com/javase/tutorial/java/nutsandbolts/index.html