Java Programming, Comprehensive

Lecture 2

Bineet Sharma

Agenda

- Control statements
  - Relational/Logical operators
  - if/else, nested ifs, switch
  - Loops
- Static method
- Data validation
  - Syntax, Logical, and Run-time error
- String
  - String, StringBuffer, and StringBuilder
- Rounding issues
  - BigDecimal

Java Language Overview: **Control Stmts**

- Java Relational Operators:

Operator	**Name**
`==`	Equality
`!=`	Inequality
`>`	Greater Than
`<`	Less Than
`>=`	Greater Than Or Equal
`<=`	Less Than Or Equal

Java Language Overview: Control Stmts

- Relational Operators produce Boolean output (true/ false). Examples of Boolean Expressions :

```
discountPercent == 2.3  // equal to a numeric literal
letter == 'y'           // equal to a char literal
isValid == false        // equal to the false value

subtotal != 0           // not equal to a numeric literal

years > 0               // greater than a numeric literal
i < months              // less than a variable

subtotal >= 500           // greater than or equal to a
                          // numeric literal
quantity <= reorderPoint  // less than or equal to a
                          // variable

isValid                 // isValid is equal to true
!isValid                // isValid is equal to false
```

Java Language Overview: Control Stmts

- Equality operators (==, !=) won't give you desired results for Strings, why? Let us look at an example:

```
Scanner sc = new Scanner(System.in);
System.out.print("Enter string1: ");
String string1 = sc.next();
System.out.print("Enter string2: ");
String string2 = sc.next();

if (string1 == string2)     // this will always be false
    System.out.println("string1 = string2");
else
    System.out.println("string1 not = string2");
```

- Because Java stores string literals in pools to reduce duplication, the equality and inequality tests for strings may not work as shown above when two String objects are assigned the same literal value
- So, this means though you are allowed to use equality operators with strings, but, they may not give you the output you desired
- Other relational operators (>, >= etc) are not even allowed

Java Language Overview: Control Stmts

- Logical operators used to combine relational operators

Operator	Name	
`!`	Not	
`&&`	And	//short circuit operators
`||`	Or	
`&`	And	//bitwise AND
`|`	Or	

- Examples of logical operators:

```
subtotal >= 250 && subtotal < 500
timeInService <= 4 || timeInService >= 12

isValid == true & counter++ < years
isValid == true | counter++ < years

(subtotal >= 250 && subtotal < 500) || isValid == true

!(counter++ >= years)
```

- What is the precedence of evaluation?

Java Language Overview: Control Stmts

- The syntax of the if/else statement:

```
if (booleanExpression) {statements}
[else if (booleanExpression) {statements}] ...
[else {statements}]
```

- Example of if/else clause:

```
double discountPercent = 0.0;
if (subtotal >= 100 && subtotal < 200)
    discountPercent = .1;
else if (subtotal >= 200 && subtotal < 300)
    discountPercent = .2;
else if (subtotal >= 300)
    discountPercent = .3;
else
    discountPercent = 0.05;
```

Java Language Overview: Control Stmts

- Example of if/else clause with blocks of code:

```
double discountPercent = 0.0;
String shippingMethod = "";
if (customerType.equals("R"))
{                                          // start block
    discountPercent = .1;
    shippingMethod = "UPS";
}                                          // end block
else if (customerType.equals("C")
{                                          // start block
    discountPercent = .2;
    shippingMethod = "Bulk";
}                                          // end block
else
    shippingMethod = "USPS";
```

Java Language Overview: Control Stmts

- Example of nested if statements:

```
if (customerType.equals("R"))
{
    if (subtotal >= 100)                    // begin nested if
        discountPercent = .2;
    else
        discountPercent =.1;                // end nested if
}
else
    discountPercent = .4;
```

Java Language Overview: Control Stmts

- The ternary operator "?:" (conditional operator):

```
(booleanExpression)? true_replacement: false_replacement
```

- Example of ?: clause:

```
//instead of this
if (a > b)
        max = a;
else
        max = b;
//use this
max = (a > b) ? a : b;
//you can use in other statement like this
System.out.println((a > b) ? "a is max" : "b is max");

??Can you do this for three numbers?
```

Java Language Overview: Control Stmts

- The syntax of switch statement:

```
switch (switchExpression)
{
    case label1:
        statements
        break;
   [case label2:
        statements
        break;] ...
   [default:
        statements
        break;]
}
```

- Example of a switch statement using integer:

```
switch (productID)
{
    case 1:
        productDescription = "Hammer";
        break;
    case 2:
        productDescription = "Box of Nails";
        break;
    default:
        productDescription = "Product not found";
        break;
}
```

Java Language Overview: Control Stmts

- Example of a switch statement using **string** (JDK 1.7+):

```
switch (productCode)
{
    case "hm01":
        productDescription = "Hammer";
        break;
    case "bn03":
        productDescription = "Box of Nails";
        break;
    default:
        productDescription = "Product not found";
        break;
}
```

- Example of a switch statement that falls through:

```
switch (dayOfWeek)
{
    case 2:
    case 3:
    case 4: case 5: case 6:
      day = "weekday"; break;
    case 1:
    case 7:
        day = "weekend"; break;
}
```

Java Language Overview: Control Stmts

- Which conditional statements to use?
  - if
  - if..else
  - Nested if .. else
  - ?:
  - switch

Java Language Overview: Control Stmts

- Working with loops. The syntax of the while loop:

```
while (booleanExpression)
{
    statements
}
```

An example of a while loop that calculates a future value

```
int i = 1;
int months = 36;
while (i <= months)
{
     futureValue = (futureValue + monthlyPayment) *
          (1 + monthlyInterestRate);
     i++;
}
```

Java Language Overview: Control Stmts

- Working with loops. The syntax of the do-while loop:

```
do
{
    statements
}
while (booleanExpression);
```

An example of a do-while loop that calculates a future value

```
int i = 1;
int months = 36;
do
{
     futureValue = (futureValue + monthlyPayment) *
          (1 + monthlyInterestRate);
     i++;
}
while (i <= months);
```

- What is the difference between a while and a do while loop?

Java Language Overview: Control Stmts

- Working with loops. The syntax of for loop:

```
for (initializationExpression; booleanExpression;
     incrementExpression)
{
    statements
}
```

- Example: A for loop that stores numbers 0-4 in a string

With a single statement

```
String numbers = "";
for (int i = 0; i < 5; i++)
    numbers += i + " ";
```

With a block of statements

```
String numbers = "";
for (int i = 0; i < 5; i++)
{
    numbers += i;
    numbers += " ";
}
```

Java Language Overview: Control Stmts

- More for loop example: A for loop that adds 8, 6, 4, 2

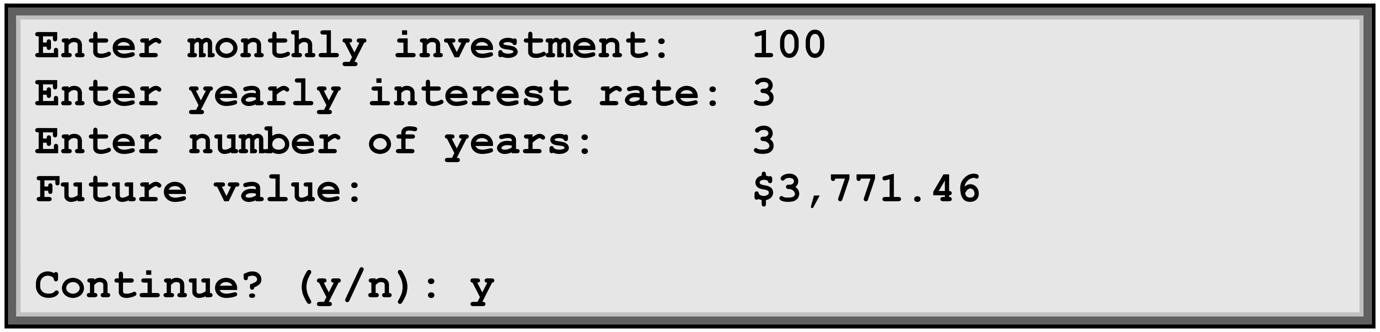
```
int sum = 0;
for (int j = 8; j > 0; j -= 2)
{
     sum += j;
}
```

- A for loop example that calculates a future value

```
for (int i = 1; i <= months; i++)
{
     futureValue = (futureValue + monthlyPayment) *
          (1 + monthlyInterestRate);
}
```

Java Language Overview: Control Stmts

- Output of the Future Value application using for loop



```
Enter monthly investment:    100
Enter yearly interest rate: 3
Enter number of years:      3
Future value:               $3,771.46

Continue? (y/n): y
```

Java Language Overview: Control Stmts

- Code of the Future Value application using for loop

```
import java.util.Scanner;
import java.text.NumberFormat;

public class FutureValueApp
{
    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);
        String choice = "y";
        while (!choice.equalsIgnoreCase("n"))
        {
            // get the input from the user
            System.out.print(
                "Enter monthly investment:   ");
            double monthlyInvestment = sc.nextDouble();
            System.out.print(
                "Enter yearly interest rate: ");
            double interestRate = sc.nextDouble();
            System.out.print(
                "Enter number of years:      ");
            int years = sc.nextInt();
```

Java Language Overview: Control Stmts

- Code of the Future Value application using for loop (cont.)

```
// convert yearly to monthly values
// and initialize future value
double monthlyInterestRate =
    interestRate/12/100;
int months = years * 12;
double futureValue = 0.0;

// use a for loop to calculate the future value
for (int i = 1; i <= months; i++)
{
    futureValue =
        (futureValue + monthlyInvestment) *
        (1 + monthlyInterestRate);
}
```

Java Language Overview: Control Stmts

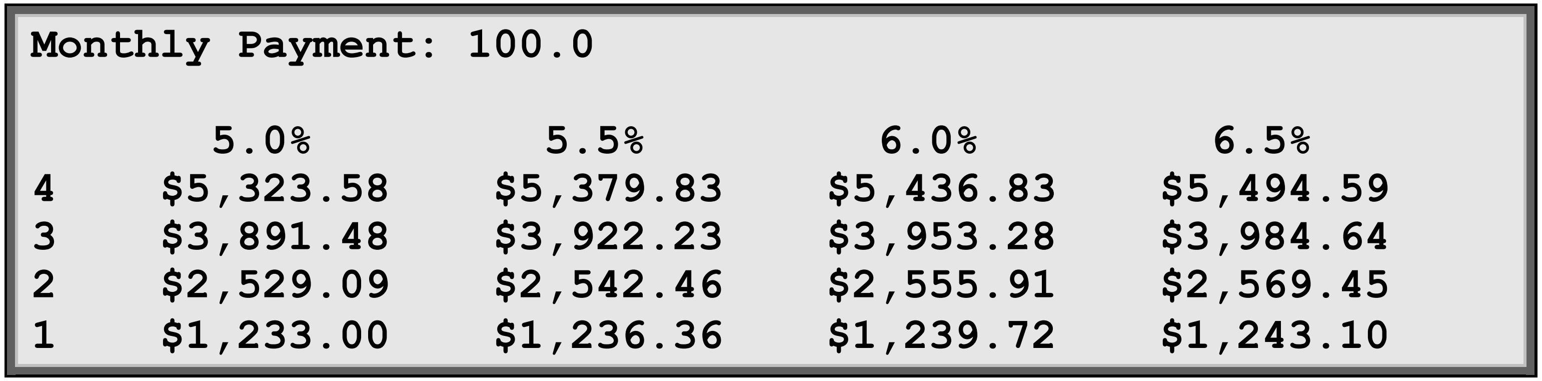
- Code of the Future Value application using for loop (cont.)

```
                // format and display the result
                NumberFormat currency =
                    NumberFormat.getCurrencyInstance();
                System.out.println(
                    "Future value:               " +
                    currency.format(futureValue));
                System.out.println();

                // see if the user wants to continue
                System.out.print("Continue? (y/n): ");
                choice = sc.next();
                System.out.println();
            }
        }
    }
```

Java Language Overview: Control Stmts

- Output of an application which uses nested for loops



```
Monthly Payment: 100.0

       5.0%         5.5%         6.0%         6.5%
4    $5,323.58    $5,379.83    $5,436.83    $5,494.59
3    $3,891.48    $3,922.23    $3,953.28    $3,984.64
2    $2,529.09    $2,542.46    $2,555.91    $2,569.45
1    $1,233.00    $1,236.36    $1,239.72    $1,243.10
```

Java Language Overview: Control Stmts

- Nested loops that print a table of future values

```
// get the currency and percent formatters
NumberFormat currency = NumberFormat.getCurrencyInstance();
NumberFormat percent = NumberFormat.getPercentInstance();
percent.setMinimumFractionDigits(1);

// set the monthly payment to 100
// and display it to the user
double monthlyPayment = 100.0;
System.out.println("Monthly Payment: " + monthlyPayment);
System.out.println();

// declare a variable to store the table
String table  = "       ";

// fill the first row of the table
for (double rate = 5.0; rate < 7.0; rate += .5)
{
    table += percent.format(rate/100) + "         ";
}
table += "\n";
```

Java Language Overview: Control Stmts

- Nested loops that print a table of future values (cont.)

```
// loop through each row
for (int years = 4; years > 0; years--)
{
    // append the years variable to the start of the row
    String row = years + "    ";
    // loop through each column
    for (double rate = 5.0; rate < 7.0; rate += .5)
    {
        // calculate the future value for each rate
        int months = years * 12;
        double monthlyInterestRate = rate/12/100;
        double futureValue = 0.0;
        for (int i = 1; i <= months; i++)
        {
            futureValue =
                (futureValue + monthlyPayment) *
                (1 + monthlyInterestRate);
        }
        // add the calculation to the row
        row += currency.format(futureValue) + "    ";
    }
    table += row + "\n";
    row = "";
}

System.out.println(table);
```

Java Language Overview: Control Stmts

- "break" and "continue" statements
- A break statement that exits the inner loop (**loop and a half**)

```
for (int i = 1; i < 4; i++)
{
    System.out.println("Outer " + i);
    while (true)
    {
        int number = (int) (Math.random() * 10);
        System.out.println("   Inner " + number);
        if (number > 7)
            break;
    }
}
```

Java Language Overview: Control Stmts

- The structure of the labeled break statement

```
labelName:
loop declaration
{
    statements
    another loop declaration
    {
        statements
        if (conditionalExpression)
        {
            statements
            break labelName;
        }
    }
}
```

Java Language Overview: Control Stmts

- A labeled break statement that exits the outer loop (cont.)

```
outerLoop:
for (int i = 1; i < 4; i++)
{
    System.out.println("Outer " + i);
    while (true)
    {
        int number = (int) (Math.random() * 10);
        System.out.println("   Inner " + number);
        if (number > 7)
            break outerLoop;
    }   // break takes execution out of the for loop
        // instead of immediate while loop

}
```

Java Language Overview: Control Stmts

- A continue statement that jumps to the beginning of the loop

```
for (int j = 1; j < 10; j++)
{
    int number = (int) (Math.random() * 10);
    System.out.println(number);
    if (number <= 7)
        continue;
    System.out.println("This number is greater than 7");

}
//continue takes it to end of the loop, so that side
effects takes place, e.g. j will be incremented
```

Java Language Overview: Control Stmts

- The structure of the labeled continue statement

```
labelName:
loop declaration
{
    statements
    another loop declaration
    {
        statements
        if (conditionalExpression)
        {
            statements
            continue labelName;
        }
    }
}
```

Java Language Overview: Control Stmts

- A labeled continue statement that jumps to the beginning of the outer loop

```
outerLoop:
for (int i = 2; i < 20; i++)
{
    for (int j = 2; j < i-1; j++)
    {
        int remainder = i % j;
        if (remainder == 0)
            continue outerLoop;
    }
    System.out.println(i);

}
```

- What does this code do? Can we improve the code?
- Be careful with the uses of label. Why?

Java Language Overview: Control Stmts

- For loop is versatile. You can ignore one, two or all three expressions

A for loop, which stores 0-4 numbers in a string

```
String numbers = "";
for (int i = 0; i < 5; i++)
    numbers += i + " ";
```

You can rewrite-same logic with missing first expression

```
String numbers = "";
int i = 0;
for (; i < 5; i++)
    numbers += i + " ";
```

With missing first and second expressions

```
String numbers = "";
int i = 0;
for (;i<5;) {
    numbers += i++ + " ";
}
```

Java Language Overview: Control Stmts

- For loop is versatile. You can ignore one, two or all three expressions

A for loop, which stores 0-4 numbers in a string

```
String numbers = "";
for (int i = 0; i < 5; i++) //does ++i matter?
    numbers += i + " ";
...
```

With missing all three expressions

```
String numbers = "";
int i = 0;
for (;;) {
    numbers += i++ + " ";
    if (i > 4) break;
}
```

- Do you suffer from **OBO(E)** syndrome?

Java Language Overview: Control Stmts

- Which Loop?
  - while
  - do-while
  - for

Java Language Overview: Structure Programming

- Structure Programming helps write code which:
  - Simple
  - Avoids indiscriminate use of GOTO statements (which creates spaghetti code)
  - Is easy to scale, generalize, read, & maintain
- There are only 7 different ways to use three Java structure:
  - 1 sequential structure
  - 3 different selection structures (if, if..else, switch)
  - 3 different loops (for, while, do..while)
- And these three different structures can be only used in two different ways
  - Stacked, or
  - Nested

Java Language Overview: Static Method

- Static method: basic syntax

```
public|private static returnType
methodName([parameterList])
{
   statements
}
```

- A static method with no parameters and no return type

```
private static void printWelcomeMessage()
{
    System.out.println("Hello New User");
}
```

Java Language Overview: Static Method

- A static method with three parameters that returns a double value

```
public static double calculateFutureValue(
        double monthlyInvestment,
        double monthlyInterestRate, int months)
{
    double futureValue = 0.0;
    for (int i = 1; i <= months; i++)
    {
        futureValue = (futureValue + monthlyInvestment)
            * (1 + monthlyInterestRate);
    }
    return futureValue;
}
```

Java Language Overview: Static Method

The syntax for calling a static method that's in the same class

```
methodName([argumentList])
```

A call statement with no arguments

```
printWelcomeMessage();
```

A call statement that passes three arguments

```
double futureValue = calculateFutureValue(
    investment, rate, months);
```

The syntax for calling a static method from different class

```
ClassName.methodName([argumentList])
```

A call statement that passes one argument

```
String strValue = Double.toString(value);
```

Java Language Overview: Static Method

- The code for the Future Value application with a static method

```
import java.util.Scanner;
import java.text.NumberFormat;

public class FutureValueApp {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String choice = "y";
        while (!choice.equalsIgnoreCase("n")) {
            // get the input from the user
            System.out.print(
                "Enter monthly investment:   ");
            double monthlyInvestment = sc.nextDouble();
            System.out.print(
                "Enter yearly interest rate: ");
            double interestRate = sc.nextDouble();
            System.out.print(
                "Enter number of years:      ");
            int years = sc.nextInt();
```

Java Language Overview: Static Method

- The code for the Future Value application with a static method (cont.)

```
// convert yearly values to monthly values
double monthlyInterestRate =
    interestRate/12/100;
int months = years * 12;

// call the future value method
double futureValue =
    calculateFutureValue(monthlyInvestment,
                         monthlyInterestRate,
                         months);

// format and display the result
NumberFormat currency =
    NumberFormat.getCurrencyInstance();
System.out.println(
    "Future value:                " +
    currency.format(futureValue));
System.out.println();
```

Java Language Overview: Static Method

- The code for the Future Value application with a static method (cont.)

```
                // see if the user wants to continue
                 System.out.print("Continue? (y/n): ");
                 choice = sc.next();
                 System.out.println();
             }
         }
         // a static method that requires three arguments
         // and returns a double
         private static double calculateFutureValue(
                       double monthlyInvestment,
                       double monthlyInterestRate, int months) {

             double futureValue = 0.0;
             for (int i = 1; i <= months; i++) {
                 futureValue =(futureValue+monthlyInvestment) *
                              (1 + monthlyInterestRate);
             }
             return futureValue;
         }
     }
```

Java Language Overview: Static Method

- Static Methods:
  - Can only use static (class) variable, local variables and parameters and other static methods
  - Cannot use non static variables and methods
  - Can be used using class names or object references
- Why Static Methods?:
  - Usually created for utility type of routines, which does not depend on state of the object
  - Most of the static methods are grouped together in a class
  - Over use of static methods in a regular class is not good OOP as objects have state and behaviors and static methods don't work on state

Java Language Overview: Data Validation

- While developing an application you will run into multiple issues (errors):
  - Syntax or compile-time error: which occurs when a statement can't be compiled. You have no other choice but to fix it before you can run your application
  - Logical error: Even if your application runs, it may not give correct result as desired. They are called bugs. These bugs must be fixed before you ship the code
  - Runtime error/exception: These exception causes your program to end prematurely. These happen for multiple reasons and must be fixed to continue testing/using your code

Java Language Overview: Data Validation

- How do you fix these issues?
  - Syntax or compile-time error: with the help of compiler and documentation
  - Logical error: by running the code in the debug mode at or near the offending line and/or going through error logs
  - Runtime error/exception: by using exception handling mechanism provided by Java API, or defensive programming

Java Language Overview: Data Validation

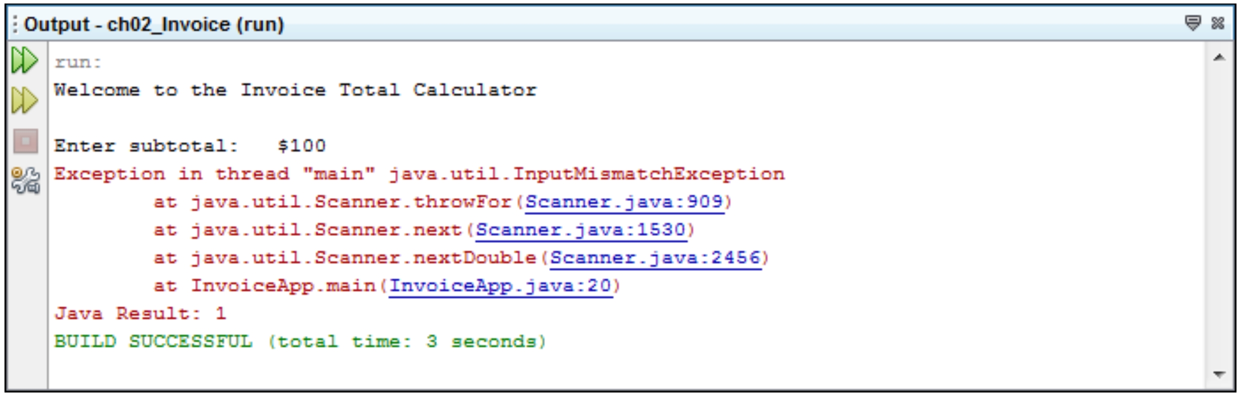
- How to fix logical error? Art of debugging
  - Application development involves
    - Design – architect
    - Coding/development – engineer
    - Testing – vandal
    - Debugging – CSI Miami
  - Cost of development increases 10 folds in each of these phases
    - Design, Development (10), Debugging (100), Deployment (1000)
  - NASA lost Rover for a simple mistake in the code, costing $500M ($1000/line of code)

Java Language Overview: Data Validation

- What are bugs?
  - A bad variable values
  - You wrote a faulty logic
  - What you assumed (during development vs during run-time)
- What should be your approach?
  - Most problems are simple
  - Be systematic in your approach
    - Using println(s), in each method, test parameters, unit testing
  - Be thorough in your assumptions
  - Follow facts vs. intuition
  - Be critical of codes
  - Look at the details
  - **Above all don't panic!**

Java Language Overview: Data Validation

- A runtime error situation with Invoice application

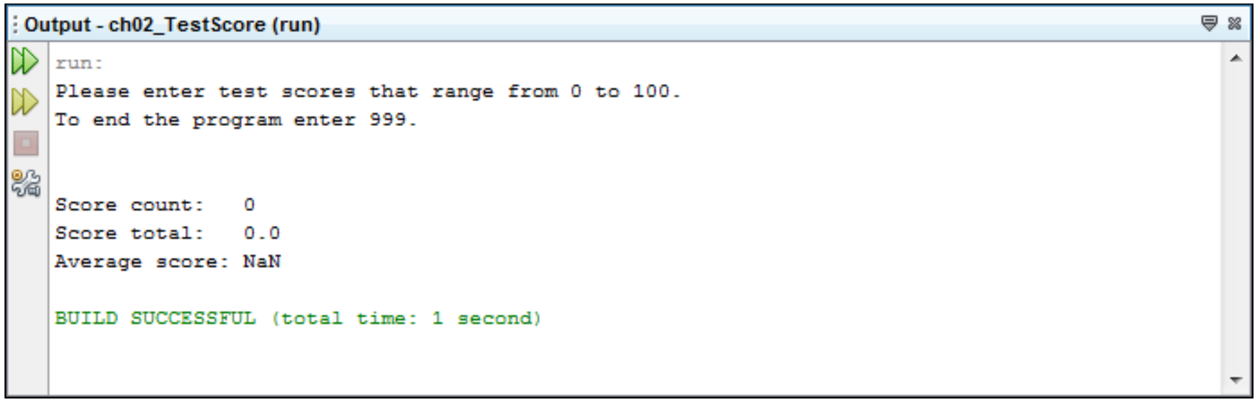


Output - ch02_Invoice (run)

```
run:
Welcome to the Invoice Total Calculator

Enter subtotal:   $100
Exception in thread "main" java.util.InputMismatchException
        at java.util.Scanner.throwFor(Scanner.java:909)
        at java.util.Scanner.next(Scanner.java:1530)
        at java.util.Scanner.nextDouble(Scanner.java:2456)
        at InvoiceApp.main(InvoiceApp.java:20)
Java Result: 1
BUILD SUCCESSFUL (total time: 3 seconds)
```

- A logic error situation with the Test Score application



Output - ch02_TestScore (run)

```
run:
Please enter test scores that range from 0 to 100.
To end the program enter 999.


Score count:   0
Score total:   0.0
Average score: NaN

BUILD SUCCESSFUL (total time: 1 second)
```

Java Language Overview: Data Validation

- Java run-time (JVM) generates such exception when abnormal thing happens
- Java bundles the information regarding that exception in a class which are derived from **Throwable**. It then creates that object and **throws** to the offending methods which caused it
- Java allows ways to **catch** such exception object in your code to handle in a graceful way or throw back to the method which called your method
- You can generate your own exception as well

Java Language Overview: Data Validation

- Some of the classes in the Exception hierarchy:

```
Exception
    RuntimeException
        NoSuchElementException
            InputMismatchException
        IllegalArgumentException
            NumberFormatException
        ArithmeticException
        NullPointerException
```

- Output (stack trace) after an InputMismatchException has been thrown

```
Enter subtotal:   $100
Exception in thread "main"
java.util.InputMismatchException
    at java.util.Scanner.throwFor(Scanner.java:909)
    at java.util.Scanner.next(Scanner.java:1530)
    at java.util.Scanner.nextDouble(Scanner.java:2456)
    at InvoiceApp.main(InvoiceApp.java:20)
```

Java Language Overview: Data Validation

- Methods that can throw an exception

Class	Method	Throws
`Scanner`	`nextInt()`	`InputMismatchException`
`Scanner`	`nextDouble()`	`InputMismatchException`
`Integer`	`parseInt(String)`	`NumberFormatException`
`Double`	`parseDouble(String)`	`NumberFormatException`

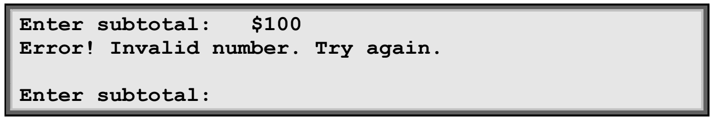
Java Language Overview: Data Validation

- Handling InputMismatchException
- Syntax for the try .. catch block

```
try { statements }
catch (ExceptionClass exceptionName) { statements }
```

- Example of handling InputMismatchException

```
 import.java.util.InputMismatchException;
//… more code
 double subtotal = 0.0;
 try
 {
     System.out.print("Enter subtotal:   ");
     subtotal = sc.nextDouble();
 }
 catch (InputMismatchException e)
 {
     sc.next();  // discard the incorrectly entered double
     System.out.println(
         "Error! Invalid number. Try again.\n");
     continue;   // jump to the top of the loop
 }
 //… more code
```



```
Enter subtotal:   $100
Error! Invalid number. Try again.

Enter subtotal:
```

Java Language Overview: Data Validation

- Exception handling example: Future Value Application

```
import java.util.*;
import java.text.NumberFormat;

public class FutureValueExceptionApp
{
    public static void main(String[] args)
    {
        System.out.println(
            "Welcome to the Future Value Calculator\n");
        Scanner sc = new Scanner(System.in);
        String choice = "y";
        while (choice.equalsIgnoreCase("y"))
        {
            double monthlyInvestment = 0.0;
            double interestRate = 0.0;
            int years = 0;
            try
            {
                System.out.print(
                    "Enter monthly investment:    ");
```

Java Language Overview: Data Validation

- Exception handling example: Future Value Application (cont.)

```
            monthlyInvestment = sc.nextDouble();
            System.out.print(
                "Enter yearly interest rate: ");
            interestRate = sc.nextDouble();
            System.out.print(
                "Enter number of years:      ");
            years = sc.nextInt();
        }
        catch (InputMismatchException e)
        {
            sc.next();    // discard the invalid number
            System.out.println(
                "Error! Invalid number. Try again.\n");
            continue;    // jump to the top of the loop
        }
```

Java Language Overview: Data Validation

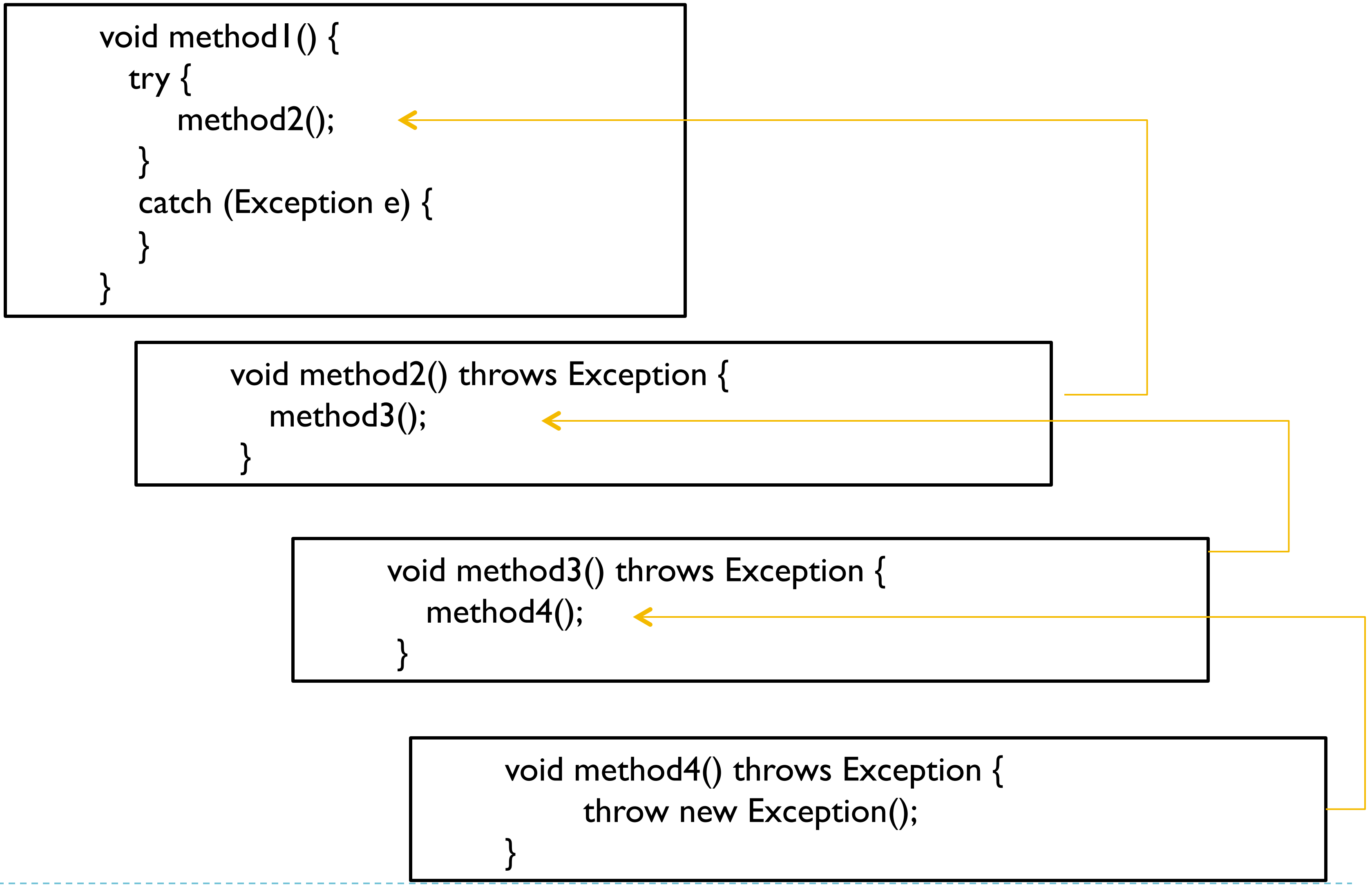
- Exception handling example: Future Value Application (cont.)

```
        private static double calculateFutureValue(
        double monthlyInvestment, double monthlyInterestRate,
        int months)
        {
            double futureValue = 0;
            for (int i = 1; i <= months; i++)
                futureValue =
                    (futureValue + monthlyInvestment) *
                    (1 + monthlyInterestRate);
            return futureValue;
        }
    }
```

Java Language Overview: Data Validation

- There are five keywords used in Java for exception:
  - **try**: block for offending code
  - **catch**: block to catch exception
  - **throw**: to throw exception manually. Java throws system generated exception automatically
  - **throws**: use to describe a method which throws an exception out
  - **finally**: block executed after try with or without catch

Java Language Overview: Data Validation



```
void method1() {
   try {
        method2();
    }
    catch (Exception e) {
    }
}
```

```
void method2() throws Exception {
    method3();
}
```

```
void method3() throws Exception {
    method4();
}
```

```
void method4() throws Exception {
        throw new Exception();
}
```

Java Language Overview: Data Validation

- You can throw only those exceptions which you have declared in the '**throws**' clause of method declaration or the subclass of that exception.
- If you can't find a suitable exception class in the library, you can create your own exception
- Derive your exception class from any of Exception classes

```
// Write your own
class MyOutOfRangeException extends Exception {
   MyOutOfRangeException(String message){
      super(message); //call the base class constructor
   }
}
```

- More on extends when we do inheritance

Java Language Overview: Data Validation

- Using your own MyOutOfRangeException

```
public static int getInput(Scanner in) throws
            NullPointerException,
            MyOutOfRangeException             {

     int userChoice = 0;

     if (in == null) {
         throw new NullPointerException("Null Scanner");
     }
     System.out.println(
             "Please enter a value between 1 to 5");
     userChoice = Integer.parseInt(in.next());
     if (userChoice < 1 || userChoice > 5) {
         throw new MyOutOfRangeException
             ("Please enter value between 1 to 5");
     }
     return userChoice;
}
```

Java Language Overview: Data Validation

- Using exception properly. Order of catch matters

```
userChoice = 0; //initialize it
try {
   userChoice = getInput(inputReader);
}// order of catch is important

catch (NullPointerException e) {
   System.out.println("Message:"+e.getMessage());

}

catch (MyOutOfRangeException e) {
   System.out.println("Message: " + e.getMessage());
}

catch (Exception e) {
   System.out.println("Catch all Exception");
}

finally { //always exectued with our without catch
   if (userChoice==0) //still has initial value
       System.out.println("Please try again ...");
   else
      System.out.println("Your choice is:"+ userChoice);
```

Java Language Overview: Data Validation

- Your code should avoid the exception situation in the first place – defensive programming
- Use following Scanner methods to validate the data:
  - `hasNext()`
  - `hasNextInt()`
  - `hasNextDouble()`
  - `nextLine()`

Java Language Overview: Data Validation

- Code that prevents an InputMismatchException

```
double subtotal = 0.0;
System.out.print("Enter subtotal:   ");
if (sc.hasNextDouble())
{
    subtotal = sc.nextDouble();
}
else
{
    sc.nextLine();      // discard the entire line
    System.out.println(
        "Error! Invalid number. Try again.\n");
    continue;           // jump to the top of the loop
}
```

- Output of above code

```
Enter subtotal:   $100
Error! Invalid number. Try again.

Enter subtotal:
```

Java Language Overview: Data Validation

- Code that prevents NullPointerEception

```
if (customerType != null)
{
    if (customerType.equals("R"))
         discountPercent = .4;
}
```

Java Language Overview: Data Validation

- Code that gets a valid double value within a specified range

```
Scanner sc = new Scanner(System.in);
double subtotal = 0.0;
boolean isValid = false;
while (isValid == false)
{
    // get a valid double value
    System.out.print("Enter subtotal:   ");
    if (sc.hasNextDouble())
    {
        subtotal = sc.nextDouble();
        isValid = true;
    }
    else
    {
        System.out.println(
            "Error! Invalid number. Try again.");
    }
    sc.nextLine();
            // discard any other data entered on the line
```

Java Language Overview: Data Validation

- Code that gets a valid double value within a specified range (cont.)

```
        // check the range of the double value
        if (isValid == true && subtotal <= 0)
        {
            System.out.println(
                "Error! Number must be greater than 0.");
            isValid = false;
        }
        else if (isValid == true && subtotal >= 10000)
        {
            System.out.println(
                "Error! Number must be less than 10000.");
            isValid = false;
        }
    }
```

Java Language Overview: Data Validation

- A method that gets a valid numeric format

```
public static double getDouble(Scanner sc, String prompt)
{
    double d = 0.0;
    boolean isValid = false;
    while (isValid == false)
    {
        System.out.print(prompt);
        if (sc.hasNextDouble())
        {
            d = sc.nextDouble();
            isValid = true;
        }
        else
        {
            System.out.println(
                "Error! Invalid number. Try again.");
        }
        sc.nextLine();       // discard any other data
    }
    return d;
}
```

Java Language Overview: Data Validation

- A method that gets a valid numeric range

```
public static double getDoubleWithinRange(Scanner sc,
                String prompt, double min, double max)
{
    double d = 0.0;
    boolean isValid = false;
    while (isValid == false){
        d = getDouble(sc, prompt);
       if (d <= min){
            System.out.println(
                "Error! Number must be greater than " +
                min + ".");
        }
        else if (d >= max){
            System.out.println(
                "Error! Number must be less than " +
                max + ".");
        }
        else
            isValid = true;
    }
    return d;
}
```

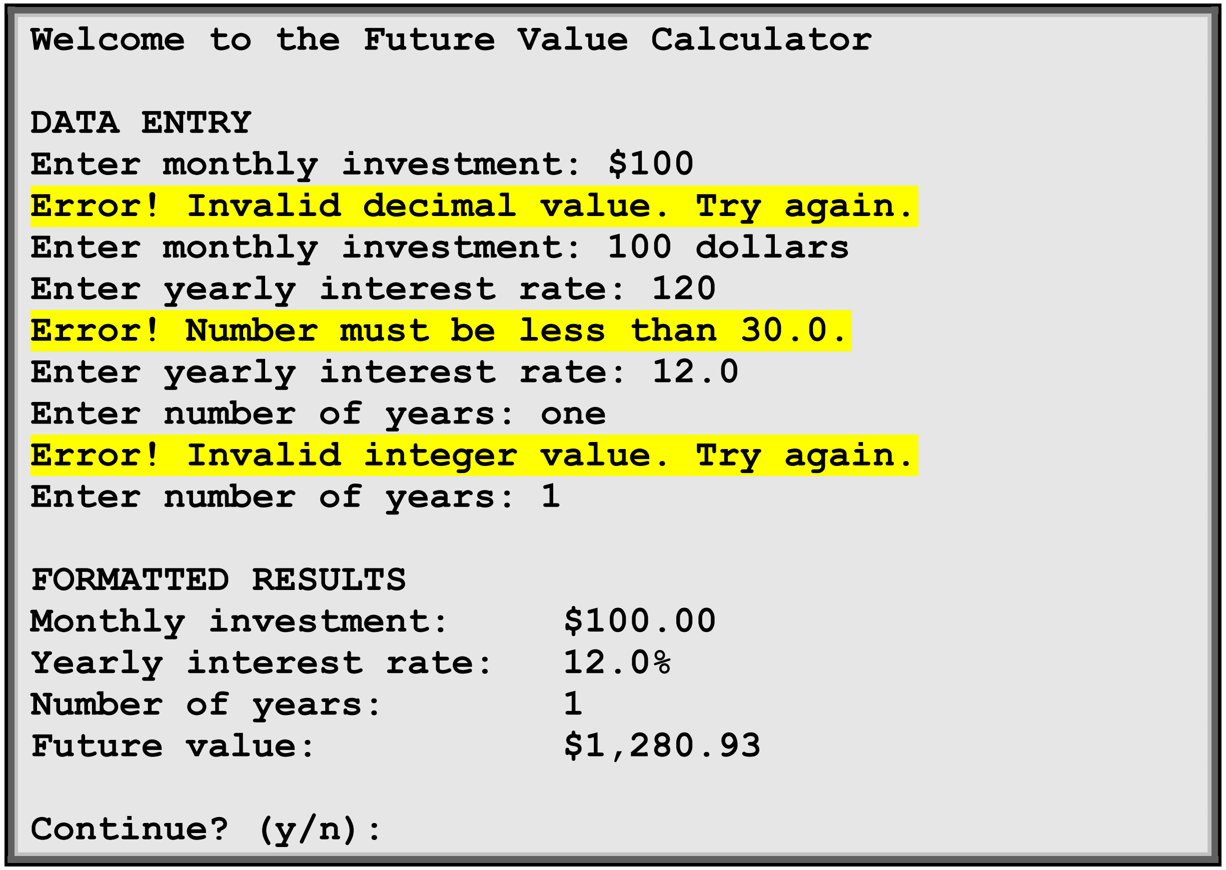
Java Language Overview: Data Validation

- Code that uses these methods to return two valid double values:

```
Scanner sc = new Scanner(System.in);
double subtotal1 = getDouble(sc, "Enter subtotal: ");
double subtotal2 = getDoubleWithinRange(
    sc, "Enter subtotal: ", 0, 10000);
```

Java Language Overview: Data Validation

- Output of the Future Value application



```
Welcome to the Future Value Calculator

DATA ENTRY
Enter monthly investment: $100
Error! Invalid decimal value. Try again.
Enter monthly investment: 100 dollars
Enter yearly interest rate: 120
Error! Number must be less than 30.0.
Enter yearly interest rate: 12.0
Enter number of years: one
Error! Invalid integer value. Try again.
Enter number of years: 1

FORMATTED RESULTS
Monthly investment:      $100.00
Yearly interest rate:    12.0%
Number of years:         1
Future value:            $1,280.93

Continue? (y/n):
```

Java Language Overview: Data Validation

- The code for the Future Value application with data validation

```
import java.util.*;
import java.text.*;

public class FutureValueValidationApp{
    public static void main(String[] args) {
        System.out.println(
            "Welcome to the Future Value Calculator\n");

        Scanner sc = new Scanner(System.in);
        String choice = "y";
        while (choice.equalsIgnoreCase("y")) {
            System.out.println("DATA ENTRY");
            double monthlyInvestment =
                getDoubleWithinRange(sc,
                "Enter monthly investment: ", 0, 1000);
            double interestRate = getDoubleWithinRange(sc,
                "Enter yearly interest rate: ", 0, 30);
```

Java Language Overview: Data Validation

- The code for the Future Value application with data validation (cont.)

```
int years = getIntWithinRange(sc,
    "Enter number of years: ", 0, 100);

double monthlyInterestRate =
    interestRate/12/100;
int months = years * 12;
double futureValue = calculateFutureValue(
    monthlyInvestment, monthlyInterestRate,
    months);

NumberFormat currency =
    NumberFormat.getCurrencyInstance();
NumberFormat percent =
    NumberFormat.getPercentInstance();
percent.setMinimumFractionDigits(1);
```

Java Language Overview: Data Validation

- The code for the Future Value application with data validation (cont.)

```
                    String results =
                      "Monthly investment:\t"
                    + currency.format(monthlyInvestment)+ "\n"
                    + "Yearly interest rate:\t"
                    + percent.format(interestRate/100) + "\n"
                    + "Number of years:\t" +  years + "\n"
                    + "Future value:\t\t"
                    + currency.format(futureValue) + "\n";

                  System.out.println();
                  System.out.println("FORMATTED RESULTS");
                  System.out.println(results);

                  System.out.print("Continue? (y/n): ");
                  choice = sc.next();
                  sc.nextLine();        // discard any other data
                  System.out.println();
              }
          }
```

Java Language Overview: Data Validation

- The code for the Future Value application with data validation (cont.)

```
public static int getIntWithinRange(
         Scanner sc, String prompt, int min, int max) {
    int i = 0;
    boolean isValid = false;
    while (isValid == false) {
        i = getInt(sc, prompt);
        if (i <= min)
            System.out.println(
                "Error! Number must be greater than " +
                min + ".");
        else if (i >= max)
            System.out.println(
                "Error! Number must be less than " +
                max + ".");
        else
            isValid = true;
    }
    return i;
}
```

Java Language Overview: Data Validation

- The code for the Future Value application with data validation (cont.)

```
public static int getInt(Scanner sc, String prompt) {
       int i = 0;
       boolean isValid = false;
       while (isValid == false)
       {
           System.out.print(prompt);
           if (sc.hasNextInt()) {
               i = sc.nextInt();
               isValid = true;
           }
           else {
               System.out.println(
                   "Error! Invalid integer value. " +
                   "Try again.");
           }
           sc.nextLine();
           // discard any other data entered on the line
       }
       return i;
   }
```

Java Language Overview: Data Validation

- The code for the Future Value application with data validation (cont.)

```
public static double calculateFutureValue(
    double monthlyInvestment, double monthlyInterestRate,
    int months) {
        double futureValue = 0;
        for (int i = 1; i <= months; i++) {
            futureValue =
                (futureValue + monthlyInvestment) *
                (1 + monthlyInterestRate);
        }
        return futureValue;
    }
}
```

Java API: String

- String is a good API class to learn and solidify OOP concept
- The String Class: creates immutable object

```
java.lang.String;
```

Common constructors of the String class

- `String()`
- `String(arrayName)`
- `String(arrayName, intOffset, intLength)`

Java API: String

Two ways to create an empty string

```
String name = "";
String name = new String();
```

Two ways to create a string from another string

```
String title = "Murach's Beginning Java";
String title = bookTitle;
```

Two ways to create a string from an array of characters

```
char cityArray[] = {'D','a','l','l','a','s'};
String cityString1 = new String(cityArray);
String cityString2 = new String(cityArray, 0, 3);
```

Two ways to create a string from an array of bytes

```
byte cityArray[] = {68, 97, 108, 108, 97, 115};
String cityString1 = new String(cityArray);
String cityString2 = new String(cityArray, 0, 3);
```

Java API: String

Methods for manipulating strings

- **`length()`**
- **`indexOf(`**`String`**`)`**
- **`indexOf(`**`String, startIndex`**`)`**
- **`lastIndexOf(`**`String`**`)`**
- **`lastIndexOf(`**`String, startIndex`**`)`**
- **`trim()`**
- **`substring(`**`startIndex`**`)`**
- **`substring(`**`startIndex, endIndex`**`)`**
- **`replace(`**`oldChar, newChar`**`)`**
- **`split(`**`delimiter`**`)`**
- **`charAt(`**`index`**`)`**

Java API: String

Methods for comparing strings

- **`equals`**`(String)`
- **`equalsIgnoreCase`**`(String)`
- **`startsWith`**`(String)`
- **`startsWith`**`(String, startIndex)`
- **`endsWith`**`(String)`
- **`isEmpty`**`()`
- **`compareTo`**`(String)`**:** Returns 0 if both strings 'equals', +ve if this string is greater than another or -ve if smaller (lexicographically)
- **`compareToIgnoreCase`**`(String)`

Java API: String

Code that parses a first name from a name string

```
String fullName = " Pamela Caldwell ";
fullName = fullName.trim();
int indexOfSpace = fullName.indexOf(" ");
String firstName = fullName.substring(0,
indexOfSpace);
```

Code that parses a string containing a tab-delimited address

```
String address = "805 Main
                 Street\tDallas\tTX\t12345";
address = address.trim();
String[] addressParts = address.split("\t");

String street = addressParts[0];
String city = addressParts[1];
String state = addressParts[2];
String zip = addressParts[3];
```

Java API: String

Code that adds dashes to a phone number

```
String phoneNumber1 = "9775551212";
String phoneNumber2 = phoneNumber1.substring(0, 3);
phoneNumber2 += "-";
phoneNumber2 += phoneNumber1.substring(3, 6);
phoneNumber2 += "-";
phoneNumber2 += phoneNumber1.substring(6);
```

Code that removes dashes from a phone number

```
String phoneNumber3 = "977-555-1212";
String phoneNumber4 = "";
for(int i = 0; i < phoneNumber3.length(); i++)
{
   if (phoneNumber3.charAt(i) != '-')
      phoneNumber4 += phoneNumber3.charAt(i);
}
```

Java API: String

Code that uses the isEmpty method

```
String customerNumber = "";
~~//if (customerNumber.equals(""))~~     // old way
~~//if (customerNumber.length() == 0)~~ // old way
if (customerNumber.isEmpty())        // Java 1.6 and later
    System.out.println(
        "customerNumber contains an empty string.");
```

Code that compares strings

```
String lastName1 = "Smith";
String lastName2 = "Lee";
int sortResult =
    lastName1.compareToIgnoreCase(lastName2);
if (sortResult < 0)
    System.out.println(lastName1 + " comes first.");
else if (sortResult == 0) //counter intuitive
    System.out.println("The names are the same.");
else
    System.out.println(lastName2 + " comes first.");
```

Java API: String

- compareTo(): two strings differ – if chars differ at certain index, or length differ or both:
  - If k is smallest index where they differ, then it returns:
    - this.charAt(k)-anotherString.charAt(k)
  - Otherwise it returns
    - this.length()-anotherString.length()
- What could you do with this information?

Java API: String

- What is so special about String?
  - **Shorthand initializer**: Since strings are used frequently, Java provides a shorthand initializer for creating a string as well, unlike other classes

    String message = ”Hello world and students of Java”;
  - **A String object is immutable:** Once created content does not change. What happens in this scenario?

    String s1 = "Hello";

    s1 = "Java";

Java API: String

- What is so special about String?
  - **Strings are interned:** JVM uses a unique instance for string literals (constants) with the same character sequence. This instance is called *interned*. For example, the following statements:

    String s1 = "Hello";

    String s2 = "Hello";

    String s3 = new String("Hello");

    In this case:

    s1 == s2 is true

    s1 == s3 is false

    Because a new String object is created when you use new operator instead of using interned string

Java API: String

- The StringBuffer class:
  - A thread-safe, mutable sequence of characters
  - It has methods to manipulate the strings it holds, instead of giving a copy of
- The newer StringBuilder class:
  - Is also a mutable sequence of characters
  - Runs faster and is more efficient
  - Has same sets of methods as StringBuffer class
  - Is not thread safe

Java API: String

- The StringBuilder Class : creates mutable strings

```
java.lang.StringBuilder;
```

Constructors of the StringBuilder class

```
• StringBuilder()           //empty string of 16 chars
• StringBuilder(intLength)//intLength chars
• StringBuilder(String)    //String size + 16 chars
```

Java API: String

Methods of the StringBuilder class

- **`capacity()`**
- **`length()`**
- **`setLength(`**`intNumOfChars`**`)`**
- **`append(`**`value`**`)`**
- **`insert(`**`index, value`**`)`**
- **`replace(`**`startIndex, endIndex, String`**`)`**
- **`delete(`**`startIndex, endIndex`**`)`**
- **`deleteCharAt(`**`index`**`)`**
- **`setCharAt(`**`index, character`**`)`**
- **`charAt(`**`index`**`)`**
- **`substring(`**`index`**`)`**
- **`substring(`**`startIndex, endIndex`**`)`**
- **`toString()`**

Java API: String

Code that creates a phone number

```
StringBuilder phoneNumber = new StringBuilder();
phoneNumber.append("977");
phoneNumber.append("555");
phoneNumber.append("1212");
```

Code that adds dashes to a phone number

```
phoneNumber.insert(3, "-");
phoneNumber.insert(7, "-");
```

Code that removes dashes from a phone number

```
for(int i = 0; i < phoneNumber.length(); i++)
{
   if (phoneNumber.charAt(i) == '-')
      phoneNumber.deleteCharAt(i--);
}
```

Java API: String

Code that parses a phone number

```
StringBuilder phoneNumber =
    new StringBuilder("977-555-1212");
String areaCode = phoneNumber.substring(0,3);
String prefix = phoneNumber.substring(4,7);
String suffix = phoneNumber.substring(8);
```

Code that shows how capacity automatically increases

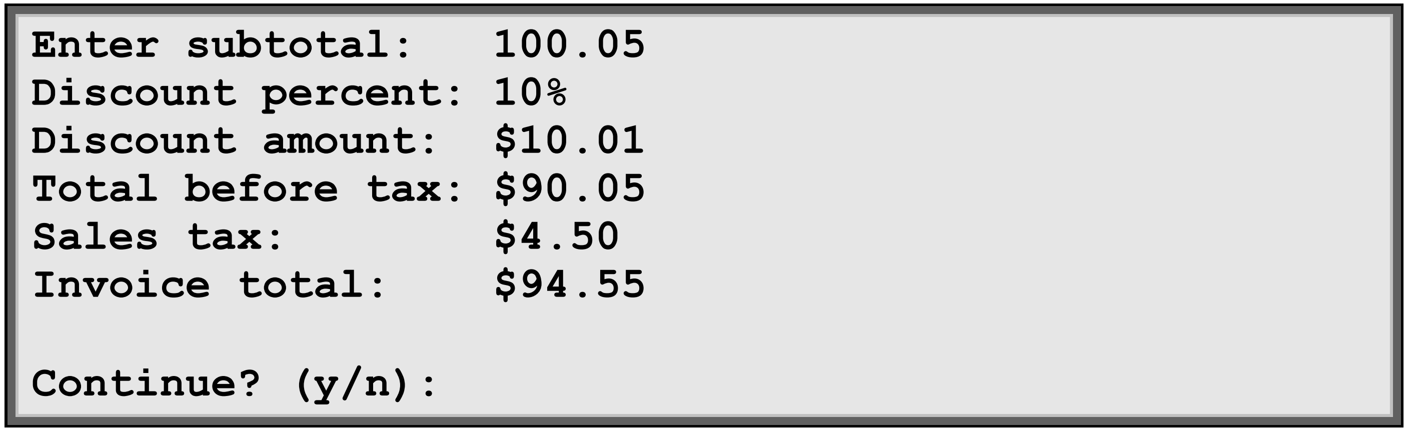
```
StringBuilder name = new StringBuilder(8);
int capacity1 = name.capacity();   // capacity1 is 8
name.append("Raymond R. Thomas");
int length = name.length();        // length is 17
int capacity2 = name.capacity();   // capacity2 is 18
                                   // (2 * capacity1 + 2)
```

Java API: String

- Which String class to use?
  - String:
    - Simple to use
    - Efficiently used by Java because of interning
    - Immutable, so, a new object gets created during manipulation
    - Don't use where a manipulation (append, concatenate etc.) is in long loops
  - StringBuilder:
    - You can change the object itself
    - Not inturned
    - Efficient in manipulation (much faster inside a long loop)
  - StringBuffer:
    - Like StringBuilder, but synchronized
    - Use it where you want thread safety
    - Slower than StringBuilder in manipulation

Java Language Overview: BigDecimal

- Example: This is the application we ran last time. Do you see any issue with the result?:

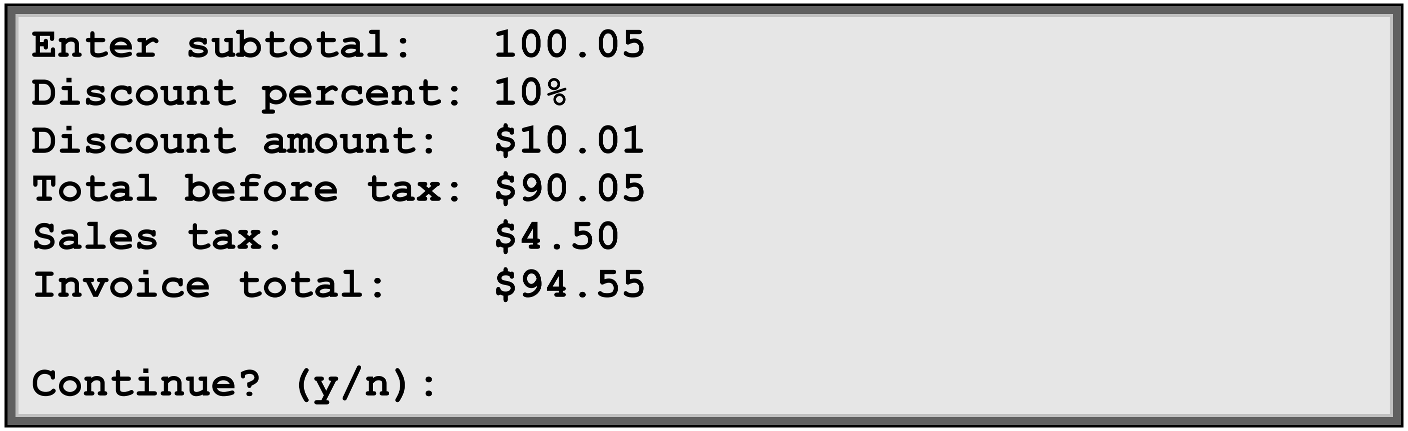


```
Enter subtotal:    100.05
Discount percent: 10%
Discount amount:   $10.01
Total before tax: $90.05
Sales tax:         $4.50
Invoice total:     $94.55

Continue? (y/n):
```

Java Language Overview

- The result don't add up. Discount of $10.01 do not add up to Total before tax of $90.05 for a 'Enter subtotal' of 100.05. What is going on?



```
Enter subtotal:    100.05
Discount percent: 10%
Discount amount:   $10.01
Total before tax: $90.05
Sales tax:         $4.50
Invoice total:     $94.55

Continue? (y/n):
```

- It is called rounding issue – age old issue with floating point numbers.
- What is the solution? Let us do some debugging first

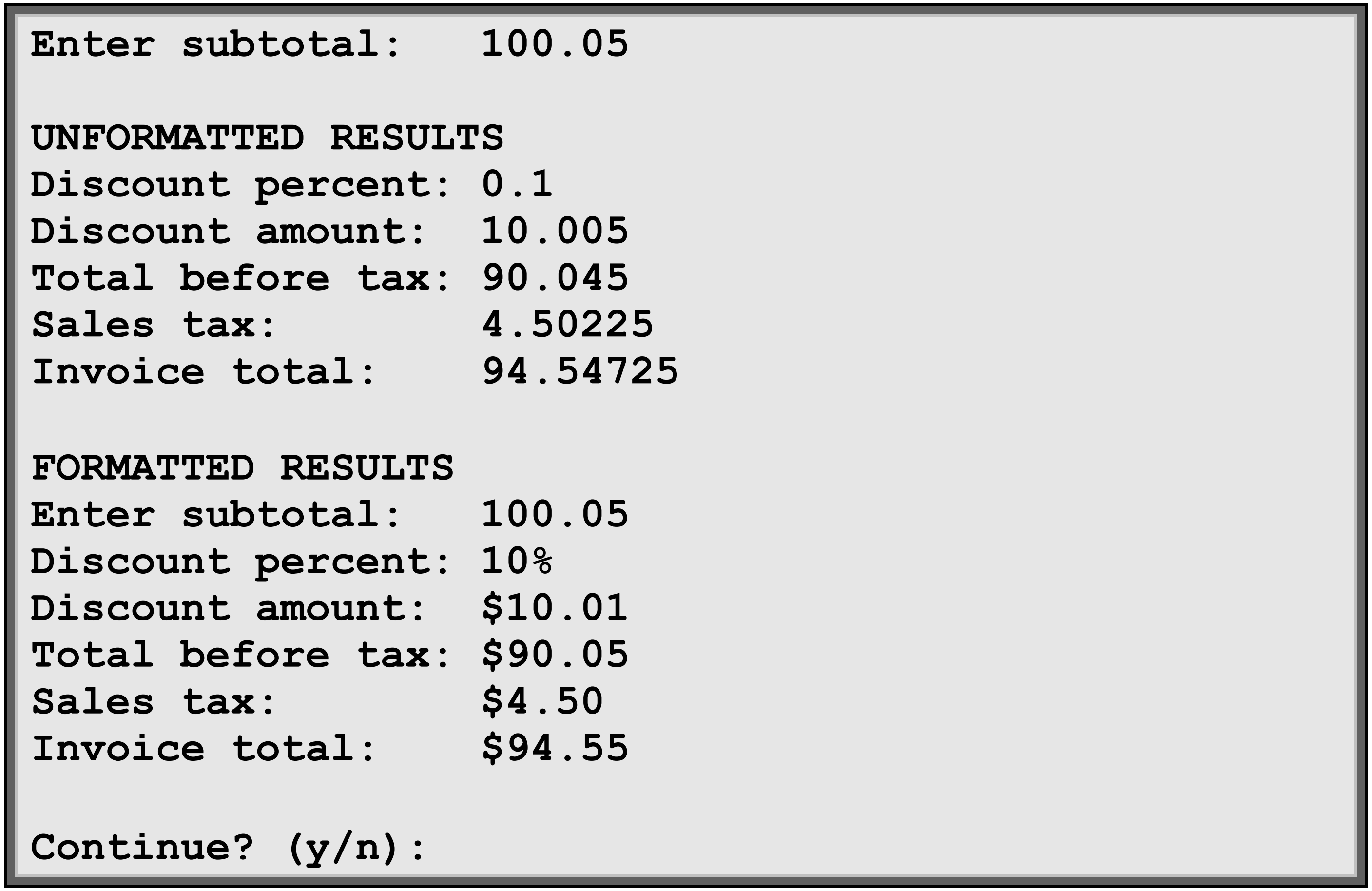
Java Language Overview

- Use debugging statements: You can use it before and after calculations – use for original data and calculated data:

```
String debugMessage = "\nUNFORMATTED RESULTS\n"
                    + "Discount percent: "
                    + discountPercent + "\n"
                    + "Discount amount:  "
                    + discountAmount + "\n"
                    + "Total before tax: "
                    + totalBeforeTax + "\n"
                    + "Sales tax:        "
                    + salesTax + "\n"
                    + "Invoice total:    " + total + "\n"
                    + "\nFORMATTED RESULTS";
System.out.println(debugMessage);
```

Java Language Overview

- Output with debugging information:



```
Enter subtotal:   100.05

UNFORMATTED RESULTS
Discount percent: 0.1
Discount amount:  10.005
Total before tax: 90.045
Sales tax:        4.50225
Invoice total:    94.54725

FORMATTED RESULTS
Enter subtotal:   100.05
Discount percent: 10%
Discount amount:  $10.01
Total before tax: $90.05
Sales tax:        $4.50
Invoice total:    $94.55

Continue? (y/n):
```

Java Language Overview

- Java provides different ways to deal with rounding issues
- BigDecimal class solves rounding issue and more:
  - Decimal numbers representation, and
  - Deal with numbers with more than 16 significant digits

The BigDecimal class

```
java.math.BigDecimal
```

Constructors of the BigDecimal class

```
•   BigDecimal(int)
•   BigDecimal(double)
•   BigDecimal(long)
•   BigDecimal(String)   :   Better to use this because of
                             the limitations of floating
                             point numbers
```

Java Language Overview

- Some of the important methods of BigDecimal class:
  - **`add`**`(value)` : `returns the Big Decimal result of adding a BigDecimal value after adding to this BigDecimal object`
  - **`compareTo`**`(value)`
  - **`divide`**`(value, scale, rounding-mode)`
  - **`multiply`**`(value)`
  - **`setScale`**`(scale, rounding-mode)`
  - **`subtract`**`(value)`
  - **`toString()`**

Java Language Overview

The RoundingMode enumeration

`java.math.RoundingMode`

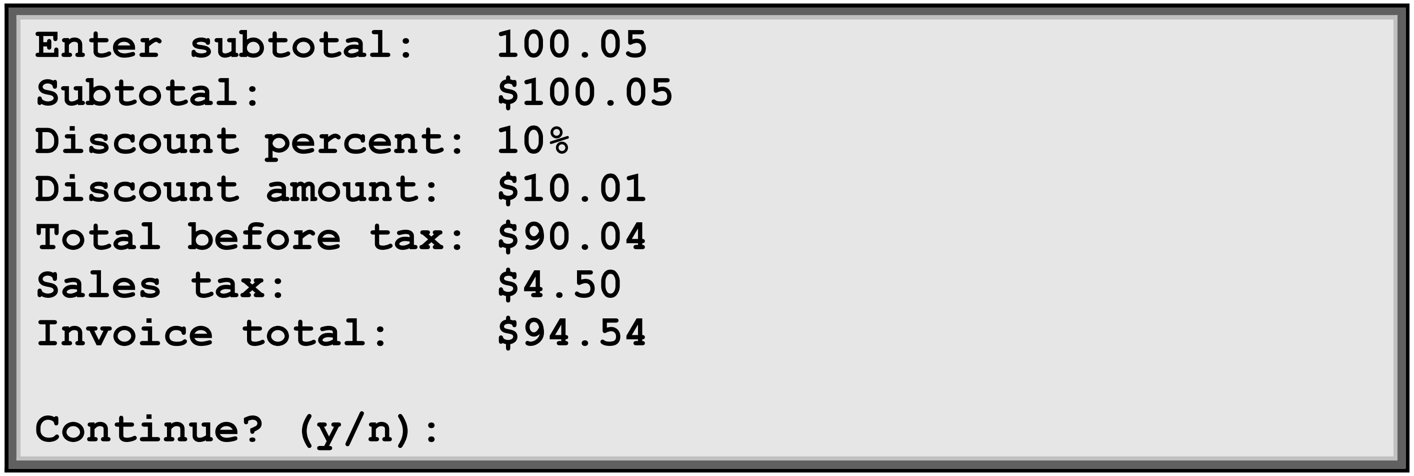
Two of the values in the RoundingMode enumeration

- **`HALF_UP:`** `Rounding mode to round towards the "nearest neighbor" unless both neighbors are equidistant, in which case, round up.`

- **`HALF_EVEN:`** `Rounding mode to round towards "nearest neighbor" unless both neighbors are equidistant, in which case round towards the even neighbor.`

Java Language Overview

- Output of Invoice application with the BigDecimal class



```
Enter subtotal:   100.05
Subtotal:         $100.05
Discount percent: 10%
Discount amount:  $10.01
Total before tax: $90.04
Sales tax:        $4.50
Invoice total:    $94.54

Continue? (y/n):
```

Java Language Overview

- Example: Invoice application with the BigDecimal class

The import statement that's required for BigDecimal arithmetic

```
import java.math.*;  // imports all classes and
                     // enumerations in java.math
```

```
// convert subtotal and discount percent to BigDecimal
BigDecimal decimalSubtotal =
    new BigDecimal(Double.toString(subtotal));
decimalSubtotal =
    decimalSubtotal.setScale(2, RoundingMode.HALF_UP);
BigDecimal decimalDiscountPercent =
    new BigDecimal(Double.toString(discountPercent));

// calculate discount amount
BigDecimal discountAmount =
    decimalSubtotal.multiply(decimalDiscountPercent);
discountAmount = discountAmount.setScale(
    2, RoundingMode.HALF_UP);
```

Java Language Overview

- Example: Invoice application with the BigDecimal class (cont.)

```
// calculate total before tax, sales tax, and total
BigDecimal totalBeforeTax =
    decimalSubtotal.subtract(discountAmount);
BigDecimal salesTaxPercent =
    new BigDecimal(SALES_TAX_PCT);
BigDecimal salesTax =
    salesTaxPercent.multiply(totalBeforeTax);
salesTax = salesTax.setScale(2, RoundingMode.HALF_UP);
BigDecimal total = totalBeforeTax.add(salesTax);
```

You can even create another BigDecimal object from an existing BigDecimal object (can't use BigDecimal object directly**):**

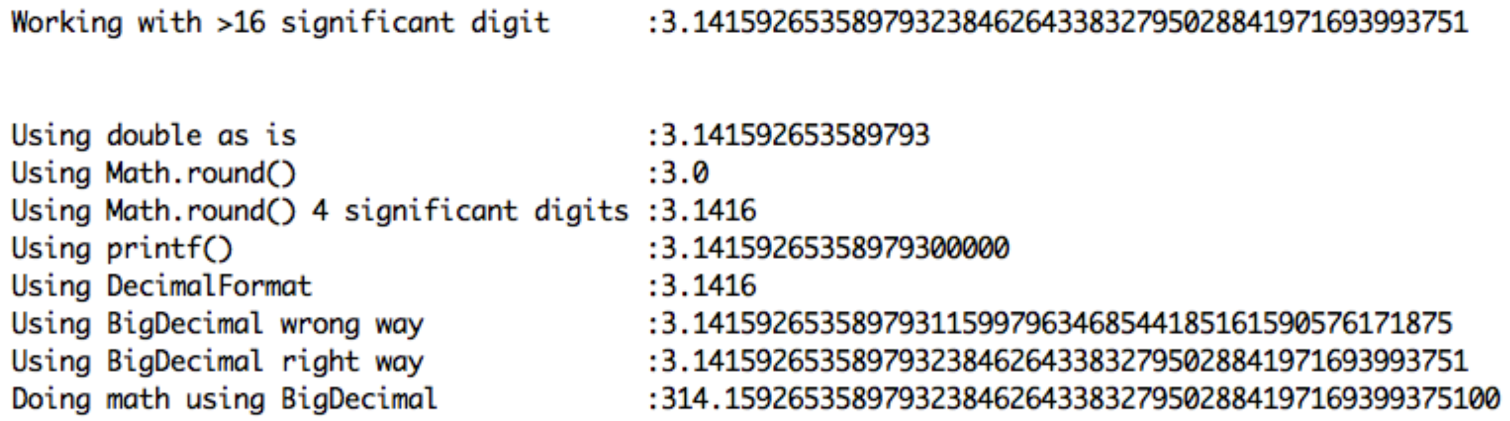
```
BigDecimal total2 = new BigDecimal(total.toString());
```

Java Language Overview

- Other options available to do rounding in Java:
  - Use Math.round()
  - Use printf format specifiers
  - Use formatting API, DecimalFormat, CurrencyFormat etc
- However, if you need to work on more than 16 significant digits, you need to use:
  - BigDecimal class

Java Language Overview

- Example output working with >16 significant digits



```
Working with >16 significant digit      :3.141592653589793238462643383279502884197169399375 1


Using double as is                      :3.141592653589793
Using Math.round()                      :3.0
Using Math.round() 4 significant digits :3.1416
Using printf()                          :3.1415926535897930000 0
Using DecimalFormat                     :3.1416
Using BigDecimal wrong way              :3.141592653589793115997963468544185161590576171875
Using BigDecimal right way              :3.141592653589793238462643383279502884197169399375 1
Doing math using BigDecimal             :314.159265358979323846264338327950288419716939937510 0
```

- Code walk through WorkingWithBIGNumbers.java
- BigDecimal runs a lot slower compare with primitive types or even DecimalFormat.
- Use only where significant digit is required to be >16

Further Reading

- Murach’s Java Programming:
  - Chapter 1 – 6
- Oracle Tutorials:
  - http://docs.oracle.com/javase/tutorial/essential/exceptions/

What Will You Do This Week & Next?

- Go through lecture#2 material and demo codes
- Complete test your understanding quiz
  - Test your understanding quizzes are not graded
- Complete HW#2
  - It is available in the assignment section. Submit your work and you will be awarded with "Model Solution" right away.
  - Homework are worth 5% of your grade
- Read ahead for lecture#3

Next Lecture

- OOP Concepts
- OOP Programming

Summary

- Rounding issues
  - BigDecimal
- Control statements
  - Relational/Logical operators
  - if/else, nested ifs, switch
  - Loops
- Static method
- Data validation
  - Syntax, Logical, and Run-time error
- String
  - String, StringBuffer, and StringBuilder