# LESSON 2
# FUNDAMENTAL PROGRAMMING STRUCTURES

***Transcendental Consciousness is the simplest form of awareness***

# WHOLENESS OF THE LESSON

**Java is an object-oriented programming language that supports both primitive and object data types. These data types make it possible to store data in memory and modify it or perform computations on it to produce useful output. Execution of a program is an example of the "flow of knowledge"—the intelligence that has been coded into the program has a chance to be expressed when the program executes.**

***Maharishi's Science of Consciousness locates three components to any kind of knowledge: the knower, the object of knowledge, and the process of knowing. These can be found in the structure of a Java program: the "knower" aspect of the program is the intelligence underlying the creation of Java objects—a Java class. The data that a program works on, which is stored in program variables of either primitive or object type, is the "object of knowledge." And the Java methods, which act on the data, are the "process of knowing."***

# OUTLINE OF TOPICS

**Comments in Java**

**Data Types:**

- The Primitive Types
- Other data types at the basis of object creation

**Operators In Java**

- Arithmetic Operators
- Increment and Decrement Operators
- Relational And Boolean Operators
- Bitwise Operators

**Java Strings**

**Control Flow:**

- Conditional Logic
- While Loops
- For loops
- The switch Statement

**Arrays**

# COMMENTS IN JAVA

**commenting out a line with //**

**commenting out a block with /* … */**

**commenting using javadoc format /** … */**

**some javadoc keywords @author, @version, @since, @param @return**

**javadoc in Eclipse view by clicking Window->Show View ->javadoc**

***Style:* Every significant method you write should be documented with comments, javadoc style.**

# DATA TYPES: THE PRIMITIVE TYPES

**Strongly typed (Every variable must have a declared data type)**

**Eight primitive data types: `int`, `short`, `long`, `byte`, `float`, `double`, `char`, `boolean`**

**Storage requirements for each type are platform independent (unlike C, C++)**

**boolean has just 2 values: *true* and *false*. (Unlike C, not the same as 1 and 0.)**

# DATA TYPES: THE PRIMITIVE TYPES

| Type | Storage Requirement | Range (Inclusive) |
|---|---|---|
| int | 4 bytes | $-2^{31}$ to $2^{31} - 1$ |
| short | 2 bytes | $-2^{15}$ to $2^{15} - 1$ |
| long | 8 bytes | $-2^{63}$ to $2^{63} - 1$ |
| byte | 1 byte | $-2^{7}$ to $2^{7} – 1$ (-128 to 127) |
| float | 4 bytes | 7 significant (dec) digits |
| double | 8 bytes | 15 significant (dec) digits |

# THE CHAR TYPE

**Java uses the Unicode character set, a superset of ASCII uses sixteen bits (2 bytes) per character, allowing for 65,536 unique characters.**

**It is an international character set, containing symbols and characters from many languages**

**code chart can be found at:**
**http://www.unicode.org/charts/**

**http://www.ssec.wisc.edu/~tomw/java/unicode.html**

**To view a unicode code point as a character, pass in the *code units* to System.out.println. A code unit is specified using the escape character '\u' followed by 4 hex digits.**

```
System.out.println('\u0041');  //output is A
System.out.println("\u03C0"); //output is ?
```

Demo code : Unicode.java

# THE CHAR TYPE

**To represent a character literal in Java, for commonly used characters, simply place the character between single quotes: 'A' represents the letter A.**

**Characters can also be represented using a *unicode* representation – this is useful for characters that cannot be typed directly from a keyboard.**

***Examples*:**

- the ordinary letter 'A' is represented in this notation by '`\u0041`'
- the Chinese character 终 by '`\u7ec8`' (Zhōng)

**Occasionally, a character can be represented only if two of Java's unicode characters are concatenated together – these are called *supplementary characters,* and typically show up only in very specialized applications.**

***Example*:**

**the symbol for the set ℤ of integers in mathematics is represented by the pair "`\ud835\udd6b`"**

**`code chart can be found at:`**
**`http://www.ssec.wisc.edu/~tomw/java/unicode.html`**

# THE CHAR TYPE (CONTINUED)

**To compute the unicode value of a basic Java character, cast it to an int (and convert to hex notation)**

```
char c = 'K';

int AsVal = (int) c;  // return 75

String hexVal = Integer.toHexString(AsVal);  // return 4b
```

**To render a character in output, pass in the Java unicode to `System.out.println`.**

```
System.out.println('\u7ec8');  //output is 终

System.out.println("\ud835\udd6b"); //output is 𝕫
```

# ESCAPE SEQUENCE CHARACTER

**\u is an example of an *escape character.* Other common escape characters are used to represent special characters:**

**\b - backspace**

**\t - tab**

**\n - newline**

**\r – carriage return**

**\" – double quote**

**\' – single quote**

**\\ - backslash**

**System.out.println("After Seeing, he said \"hello\"");**

**//output: After Seeing, he said "hello"**

**Ex : `EscapeSequence.java`**

# IDENTIFIER

**A Java identifier can be of any length. However, it must obey the following three rules:**

**1.Every character in an identifier is a letter (A to Z, a to z), or a digit (0 to 9), or the underscore character (_), or the dollar sign ($).**

**2.The first character of an identifier must be a letter or the underscore character or the dollar sign.**

**3.An identifier cannot be a reserved word.**

# VARIABLES IN JAVA

**Variables in Java store values, like strings, characters, numbers, and other data.**

**A variable in Java always has a type; a variable is *declared* by displaying the type, followed by the variable name.**

**Examples of declaring variables:**

```
double salary;
int amount;
boolean found;
```

# VARIABLE

***Variable Initialization.***

```
int sum;
sum = 0;
```

**OR**

```
int  sum = 0;
```

***Coding Style***

***variable names begin with lower case letter.***

***variable names composed of multiple words written so that each new word (except the first) begins with a capital letter, but all other letters are lower case. For example*** `newInterestRate`.

***underscores should not be used typically in variable names***

**for *constants*, capitals and underscores are used**

***For example*** `final int HOURS_IN_A_DAY = 24;`

# ACCEPTED NAMING CONVENTION

**Package names in lowercase**

**Class names in CamelCase**

**Interface names in CamelCase**

**Methods names in mixedCase**

**Variable names in mixedCase**

**Constant names in UPPERCASE**

# READING CONSOLE INPUT

**Console input provides the simplest means for a user to interact with a Java program**

**j2se5.0 introduced a new mechanism for retrieving console input: Scanner.**

**System.in and Readers (jdk1.1)  (behaves like Scanner approach)**

**JOptionPane (jdk1.2) (creates a GUI window for input)**

```
String input = JOptionPane.showInputDialog("Type your name");
```

# EXAMPLE

```
Scanner in = new Scanner(System.in);
System.out.print("Type your name: ");
String name = in.nextLine();
System.out.println("you wrote: "+name);
System.out.print("Type your age: ");
int age = in.nextInt();
System.out.println("your age: "+age);
//output
Type your name: Jim Stevens
you wrote: Jim Stevens
Type your age: 36
your age: 36
   next() – Word Input, nextDouble(), nextFloat(),
   hasNext(), hasNextInt() etc
Demo : ScannerInput.java, ScannerHas.java
```

# EXAMPLE

```
String input = null;
BufferedReader in = new BufferedReader(new
 InputStreamReader(System.in));
System.out.print("Type your name: ");
input = in.readLine();
in.close();
```

BufferedReading.java

# MAIN POINT 1

**Variables in Java are *declared* and *initialized* to provide room in RAM for the data that is to be stored. *Pure consciousness manifests as individuals in space.***

# QUIZ - 1

You are reviewing some old java programs in your company's IT department and you read the following assignment statement in some abandoned code:

```
int a = (b = 5);
```

is the statement legal (i.e. Would it compile)? If so, what value is assigned to the variable a? Explain.

Answer :

1. Compilation Error, b cannot resolved to be a variable. So we need to declare b as int.
2. If we declare b, then no compilation error. The code assign 5 to a.

# OPERATORS IN JAVA: ARITHMETIC OPERATORS

- **Standard binary operations represented in Java by +, -, *, /. Also the modulus operator %.**

  **Note: In Java, to compute –5/2 (integer division) and -5 % 2, remove the minus sign, compute, and then insert the minus sign again:**

  **-5/2 = - ( 5/2 ) = -2**

  **-5 % 2 = - ( 5 % 2 ) = -1**

- ***Warning*! This is an old mistake that is found in most procedural languages – the computation differs from the usual mathematical definition of modulus – in math, the modulus is always a nonnegative number.**

- **Java 8 corrects this with Math.floorMod:**

  **Math.floorMod(-5, 2) = -5 (mod 2) = 1**

- **Division by `0`: for `int`, an exception is thrown; for floating point numbers, the value is `NaN`**

- **Shorthand Assignment Arithmetic Operators : +=, *=, /=, -=, %=**

# OPERATORS IN JAVA
# ARITHMETIC OPERATORS

**System.*out.println(17/12);*** ***// 1***

**System.*out.println(-17/-12);*** ***// 1***

**System.*out.println(-17/12);*** ***// -1***

**System.*out.println(17/-12);*** ***// -1***

**The sign of the result is left hand side operand.(%)**

**System.*out.println(17%12);*** ***// 5***

**System.*out.println(-17%-12);*** ***// -5***

**System.*out.println(-17%12);*** ***// -5***

**System.*out.println(5/0);*** ***// Run time error – divide by zero***

**System.*out.println(5.5f/0.0f);*** **// Infinity**

**System.*out.println(5.5f/-0.0f);*** **// -Infinity**

# OPERATORS IN JAVA: INCREMENT AND DECREMENT OPERATORS

```
int k = 1;
k++;  //new value of k is 2 (postfix  form)
++k;  //new value of k is 3 (prefix form)
```

**Difference arises only when used in expressions.**

```
int k = 0;
int m = 3 * k++; //m equals 0, k equals 1
```

```
int k = 0;
int n = 3 * ++k; //n equals 3, k equals 1
```

**Commonly used in for loops. Usually better to avoid using this in expressions, for the sake of readability.**

**Similar is the behavior of the decrement operator, --**

# WHAT IS THE OUTPUT?

```
int x = 4;
int y = ++x;
```

**What is x =  ;  y =  ;**

```
int x = 4;
int y = x++;
```

**What is x =  ;  y =  ;**

# OPERATORS IN JAVA: RELATIONAL AND LOGICAL OPERATORS

**Relational: == (equals), != (not equals), < (less than), <= (less than or equal to), >, >= (greater than, greater than or equal to)**

**Logical: &&, ||, !**

| && | T | F | | \|\| | T | F | | ! | T | F |
|---|---|---|---|---|---|---|---|---|---|---|
| T | T | F | | T | T | T | | | F | T |
| F | F | F | | F | T | F | | | | |

# BITWISE OPERATORS

**To write programs at the machine-level, often you need to deal with binary numbers directly and perform operations at the bit-level.**

**Java provides the bitwise operators and shift operators.**

**The bit operators apply only to integer types (byte, short, int, and long).**

**& (and), | (or), ^ (xor), ~ (not), << (left shift), >> (right shift)**

**Ref : For more information read Pages 134-137. Beginning Java 8 Fundamendals by Kishori Sharan from Apress. Read this thru Sakai Resourses→Textbooks.**

# OPERATORS IN JAVA: TERNARY OPERATOR

***condition ? expression1 : expression2***

**evaluates to *expression1* if condition is true, *expression2* otherwise**

```
customerStatus = (income > 100000) ? PLATINUM :SILVER;
if (income > 100000)
        customerStatus = PLATINUM;
else
        customerStatus = SILVER;
String res = (x<y && y<z)? “Sorted” : “Unsorted”;
```

**Quiz : 2**

**Consider the following if…then logic:**

**if(age > 65) socialSecurityStatus = "eligible";**

**else socialSecurityStatus = "ineligible";**

**1. Rewrite this (pseudo) code using the Java ternary operator.**

**2. Find greatest among three numbers using Ternary operator.**

# MATHEMATICAL CONSTANTS AND FUNCTIONS

**Special math functions and constants are available in Java by using the syntax**

**Math.<*constant*> and Math.<*function*>**

**Math.PI (the number pi – approximately 3.14159)**

**Math.pow(a,x) (the number raised to the power x)**

**Math.random() (for a random number) – (Number from the range of 0 to 1)**

**Examples:**

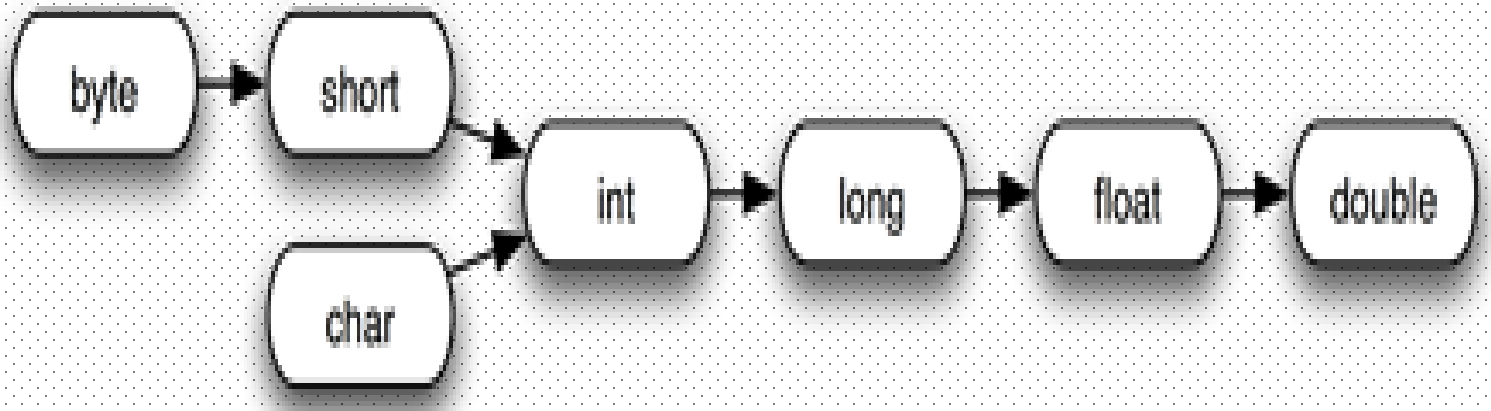
```
//produces a randomly generated int
Random random = new Random();
int num = random.nextInt();
//produces a randomly 1 possible values, ranging from 0 to 10.
int num = random.nextInt(11);
```

**You can see the 'Math' class in Oracle's documentation at**

**http://docs.oracle.com/javase/8/docs/api/**

# Type conversion

Typically: a double combined with another type results in a double; an int combined with a smaller type (byte, short, char) results in an int.




(A complete list of rules is given on your text book.)

# IMPLICIT CONVERSION

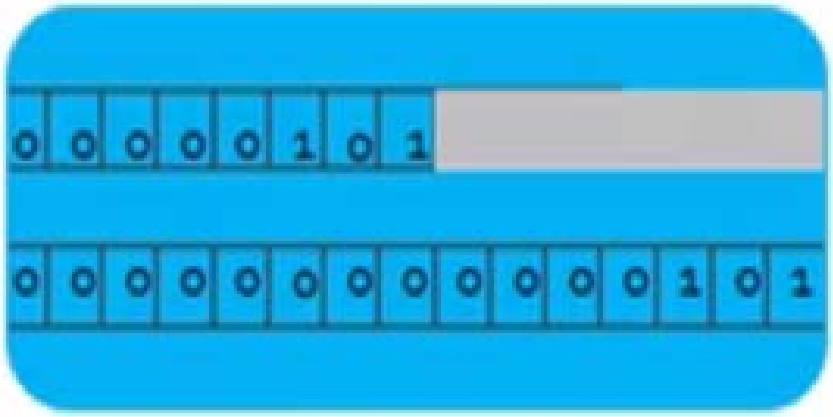
The implicit conversion **WIDENS** the memory storage for the value of x from 8 to 16 bits. There is no possibility of data loss.

```
byte x = 3;
```

A memory storage location of size 8 bits is created for x

```
short y = x;
```

Implicitly converts x from byte to short




Memory

The implicit conversion will be performed automatically.

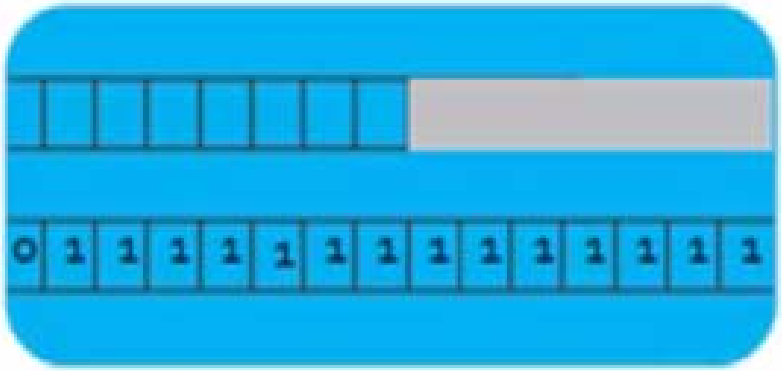
# EXPLICIT CONVERSION

Changing the type of y from short to byte reduces (**NARROWS**) the available space in memory. There will be data loss.

```
byte x = y;
```

A memory storage location of size 8 bits is created for x

```
short y = 32767;
```




**Memory**

The conversion will **NOT** be performed automatically. The conversion must be performed explicitly confirming the potential for data loss.

***Automatic promotion of integral types.*** **When a binary operation (like +, *, or any shift operator) is applied to values of type `byte` or `short`, the types are promoted to `int` before the computation is carried out.**

**<u>Example</u>: The following produces a compiler error. Why?**

```
byte x = 5;
byte y = 7;
byte z = x + y;
```

# MATHEMATICAL CONSTANTS AND FUNCTIONS

**Explicit conversions can be “forced” by casting.**

```
double x = 9.997;
int nx = (int) x;  //nx has value 9
```

**“Rounding” is usually preferable to casting and is done by using the round function of the Math class:**

```
double x = 9.997;
int nx = (int) Math.round(x);
```

**Note: round returns a long. So cast is necessary. nx becomes 10.**

# TELL THE OUTPUT

```
System.out.println((int)1.7); // 1
System.out.println((double)1 / 2); // 0.5
System.out.println(1 / 2); // 0
char ch = (char)65.25; //65
System.out.println(ch); // A
int i = (int)'A';
System.out.println(i); // 65
```

# OPERATOR PRECEDENCE

| Operator precedence | |
|---|---|
| Operators | Associativity |
| [] . () (method call) | left to right |
| ! ~ ++ +(unary) –(unary) ()(cast) new | right to left |
| * / % | left to right |
| + - | left to right |
| << >> >>>> | left to right |
| < <= > >= instanceof | left to right |
| == != | left to right |
| & | left to right |
| ^ | left to right |
| \| | left to right |
| && | left to right |
| \|\| | left to right |
| ?: (ternary operator) | right to left |
| = += -= *= /= %= &= \|= ^= <<= >>= >>>= | right to left |

See page 64 of Core Java, Vol 1, 10th ed. for a list of all the rules

```
int x = 3+2*5;
int y = (3+2)*5;
```

# MAIN POINT 2

**Variables of primitive type can be combined to form expressions through the use of *operators.* The Java syntax requires one to observe rules for forming expressions – precedence rules, type conversion rules, and others. *Likewise, Pure consciousness also has laws that govern its self-combining. Combining of the three fundamental aspects of consciousness in all possible ways results in the manifest creation.***

# JAVA STRING

- **A `String` is a sequence of characters.**
- **“String” is not a built-in data type (unlike `int` and `float`)**
- **It is represented by an object of the java.lang.String class**
- **A String can be initialized using a String literal.**

  **Example:**

  **String name = “Maharishi";**

  **String emtpy = "";**

- **Java Strings are *immutable.* This means the contents of a String object cannot be modified after it has been created.**

# STRING METHODS

**charAt() and length()**

**Thinking of a Java String as a sequence of characters, the charAt method extracts the character at a specified position in this sequence.**

```
"Hello".charAt(1)  //value is 'e'
```

**The length() method returns the number of characters in a String.**

```
"Hello".length() \\ Value is 5
```

**However, the value of**

```
"𝕫_𝕫".length()3
```

**http://docs.oracle.com/javase/8/docs/api/**

# STRING METHODS

## substring

```
String name = “Maharishi";
String nickname = name.substring(0,3);
 //  “Mah“ – From  index 0 to  last index (3-1)

String whole = name.substring(0,name.length());  //" Maharishi"
String empty = name.substring(0,0);  // ""
System.out.println(name.substring(4)); // rishi – return from index 4 to end
```

## indexof

```
String name = "Robert";
int posOfT = name.indexOf('t'); //5
int posOfSubstr = name.indexOf("bert"); //2
```

# STRING METHODS

## + (concatenation) – creates a new String

```
String name = “Maharishi";
String space = " ";
String lastName = “University”;
String fullname = name+space+lastName;
                  // “Maharishi University"
```

- **startsWith**

```
String name = "Robert";
boolean result = name.startsWith("Rob");//true
boolean result2 = name.startsWith("R"); //true
boolean result3 = name.startsWith("bert"); //false
```

**Example : StringMethods.java**

# STRING METHODS

## equals and ==

**String name = "Robert";**

**boolean equal = name.equals("Robert"); //true**

**boolean refEqual = (name == "Robert");  //true**

**== operator used to compare two references are same. Mainly used to compare primitives.**

**equals() used to compare two objects contents.**

**Example : EqualsDemo.java**

# STRING OBJECT VS STRING LITERAL

**Both are an object of java.lang.String class.**

**new operator stores a string objects in a heap memory.**

**Literals are stored in a String pool(part of heap memory) to minimize the memory usage and for better performance.**

**Eg:**

**String str1 = “Hello";**

**String str2 = “Hello";**

**String str3 = new String(“Hello“);**

**String str4 = new String(“Hello“);**

**String str5 = str4;**

**What is the output?**

**1. str1==str2 ? _____ 2. str2==str3 ? _____ ( T F )**

**3. str3==str4 ? _____ 4. str4==str5? _____ (F T )**

# STRING FUNCTIONS: COMPARETO

**The natural ordering on Strings is *alphabetical order.* In that ordering, for example, "Bob" comes before "Charles". The `compareTo` method on `Strings` specifies this ordering on `Strings`:**

```
int compareTo(String t)
```

**`s.compareTo(t)` returns**

- a positive integer if s is "greater than" t
- a negative integer is s is "less than" t
- zero, if s and t are equal as Strings

**Examples**

```
public static void main(String[] args) {
    System.out.println("a".compareTo("b"));
    System.out.println("b".compareTo("a"));
    System.out.println("a".compareTo("a"));
}
```

**//output:**

```
-1
1
0
```

- The `compareTo` method is used to sort `Strings` (see the section on Arrays)

# QUIZ

**Suppose a String s is initialized by**

**String s = "a friendly face";**

**a. Compute the value of the expression  s.charAt(2);**

**Answer : f**

**b. What is the value of  s.length()?**

**Answer : 15**

**c. What is the value of**

**s.substring(2,8)?  friend**

**s.substring(4)? iendly face**

# PACKAGES AND IMPORT STATEMENT

**In Java, classes are grouped into packages. Most commonly used classes are in the package java.lang and this package is available to all programs.**

**The Scanner class is in the package java.util. In order to use the Scanner class, you need to import either the Scanner class or the package java.util.**

**This is accomplished by placing an import statement in the beginning of the source program.**

```
import java.util.*;                 // imports java.util package
                or
import java.util.Scanner;           // imports Scanner class
```

# FORMATTED CONSOLE OUTPUT

- **j2se5.0 introduced C-like formatting features with System.out.printf and String.format**
- **Use System.out.printf to print formatted output directly to the console**
- **Use String.format, with the same formatting options, to store formatted String in memory, perhaps to be sent to the console or a file (for example) at a later time**
- **Can be combined with Date and Time formatting**
- **A complete list of conversion characters (like s, d) can be found on p. 83, Core Java, 10th edition.**

# FORMATTED CONSOLE OUTPUT USING PRINTF()

```
System.out.printf("You owe me $%f \n", 195.50f);
System.out.printf("You owe me $%.2f \n", 195.50f);
System.out.printf("You owe me $%7.2f \n", 195.50f);
```

```
You owe me $195.500000
You owe me $195.50
You owe me $ 195.50
```

```
String name = "Bob";
int age = 30;
System.out.printf("Happy birthday %s. I can't believe you're %d.", name, age);
```

```
Happy birthday Bob. I can't believe you're 30.
```

# FORMATTED CONSOLE OUTPUT USING FORMAT()

**String oweMe = String.format("You owe me %.2f dollars", 196);**

**String oweMe2 = String.format("You owe me %d dollars", 196);**

**System.out.println(oweMe);**

**System.out.println(oweMe2);**

**You owe me 196.00 dollars**

**You owe me 196 dollars**

**`FormatOutput.java`**

# PRACTICE EXAMPLES USING LECTURE SLIDE AND TEXT BOOK