

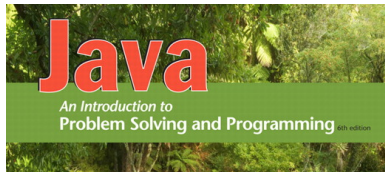
# SENG1110/SENG6110

## Object Oriented Programming



### Lecture 2

#### Java basics - 2



## First computer lab

- This week, you will have your first computer lab.
- You will receive instructions in the beginning of your lab from your demonstrator(s).
- You will find some videos in each lab to help you understand some key concepts.
- Each lab you will have a minimal task that will be necessary to complete. These tasks will be 5% of your final mark.
- **Please, do all the exercises!!!**

## Outline

- Previously...
  - A computer: hardware and software
  - Programming X software engineering
  - Programming languages (Java)
- Now...
  - Variables, expressions and operators
  - Input and output
  - The Class String
  - Documentation and Style
  - Examples

2/26/17  
Dr Regina Berretta



## Variables

4

- *Variables* store data such as numbers and letters.
  - Think of them as places to store data.
  - They are implemented as memory locations.
- The data stored by a variable is called its *value*.
  - The value is stored in the memory location.
- Its value can be changed.

## Variables

If you have  
6 eggs per basket and  
10 baskets, then  
the total number of eggs is 60

Sample  
Screen  
Output

## Variables

6

```
public class EggBasket
{
    public static void main (String [] args)
    {
        int numberOfBaskets, eggsPerBasket, totalEggs;
        numberOfBaskets = 10; eggsPerBasket = 6;
        totalEggs = numberOfBaskets * eggsPerBasket;
        System.out.println ("If you have");
        System.out.println (eggsPerBasket + " eggs per basket and");
        System.out.println (numberOfBaskets + " baskets, then");
        System.out.println ("the total number of eggs is " +totalEggs);
    }
}
```

## Variables and Values

7

- Variables
  - `numberOfBaskets`
  - `eggsPerBasket`
  - `totalEggs`
- Assigning values
  - `eggsPerBasket = 6;`
  - `eggsPerBasket = eggsPerBasket - 2;`

## Naming and Declaring Variables

8

- Choose names that are helpful such as `count` or `speed`, but not `c` or `s`.
- When you *declare* a variable, you provide its name and type.
  - `int numberOfBaskets, eggsPerBasket;`
- A variable's *type* determines what kinds of values it can hold (`int`, `double`, `char`, etc.).
- A variable must be declared before it is used.

## Syntax and Examples

9

- Syntax

```
type variable_1, variable_2, ...;  
(variable_1 is a generic variable called a syntactic variable)
```

- Examples

```
int styleChoice, numberOfChecks;  
double balance, interestRate;  
char jointOrIndividual;
```

## Data Types

10

- A *primitive type* is used for simple, non decomposable values such as an individual number or individual character.
  - `int`, `double`, and `char` are primitive types.
- A *class type* is used for a class of objects and has both data and methods.
  - "`Java is fun`" is a value of class type `String`

## Primitive Types

- Figure 2.1 Primitive Types

Type Name	Kind of Value	Memory Used	Range of Values
<code>byte</code>	Integer	1 byte	–128 to 127
<code>short</code>	Integer	2 bytes	–32,768 to 32,767
<code>int</code>	Integer	4 bytes	–2,147,483,648 to 2,147,483,647
<code>long</code>	Integer	8 bytes	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
<code>float</code>	Floating-point	4 bytes	$\pm 3.40282347 \times 10^{+38}$ to $\pm 1.40239846 \times 10^{-45}$
<code>double</code>	Floating-point	8 bytes	$\pm 1.79769313486231570 \times 10^{+308}$ to $\pm 4.94065645841246544 \times 10^{-324}$
<code>char</code>	Single character (Unicode)	2 bytes	All Unicode values from 0 to 65,535
<code>boolean</code>		1 bit	True or false

## Java Identifiers

- Identifiers may not contain any spaces, dots (`.`), asterisks (`*`), or other characters:  
`7-11 netscape.com util.*` (not allowed)
- Identifiers can be arbitrarily long.
- Since Java is *case sensitive*, `stuff`, `Stuff`, and `STUFF` are different identifiers.

## Keywords or Reserved Words

---

- Words such as `if` are called *keywords* or *reserved words* and have special, predefined meanings.
  - Cannot be used as identifiers.
  - See Appendix 1 for a complete list of Java keywords.
- Example keywords: `int`, `public`, `class`

## Where to Declare Variables

---

- Declare a variable
  - Just before it is used or
  - At the beginning of the section of your program that is enclosed in `{}`.

```
public static void main(String[] args)
{
    /* declare variables here */
    . . .
}
```

## Naming Conventions

---

- Class types begin with an uppercase letter (e.g. `String`).
- Primitive types begin with a lowercase letter (e.g. `int`).
- Variables of both class and primitive types begin with a lowercase letters (e.g. `myName`, `myBalance`).
- Multiword names are "punctuated" using uppercase letters.

## Primitive Types

---

- Four integer types (`byte`, `short`, `int`, and `long`)
  - `int` is most common
- Two floating-point types (`float` and `double`)
  - `double` is more common
- One character type (`char`)
- One boolean type (`boolean`)

## Examples of Primitive Values

---

- Integer types

`0 -1 365 12000`

- Floating-point types

`0.99 -22.8 3.14159 5.0`

- Character type

`'a' 'A' '#' ' '`

- Boolean type

`true false`

## Assignment Examples

---

`amount = 3.99;`

`firstInitial = 'W';`

`score = numberOfCards + handicap;`

`eggsPerBasket = eggsPerBasket - 2;`

## Assignment Statements

---

- An assignment statement is used to assign a value to a variable.

`answer = 42;`

- The "equal sign" is called the *assignment operator*.
- We say, "The variable named `answer` is assigned a value of 42," or more simply, "`answer` is assigned 42."

## Initializing Variables

---

- A variable that has been declared, but not yet given a value is said to be *uninitialized*.
- Uninitialized primitive variables may have a default value.
- It's good practice not to rely on a default value.

## Initializing Variables

---

- To protect against an uninitialized variable (and to keep the compiler happy), assign a value at the time the variable is declared.

- Examples:

```
int count = 0;
char grade = 'A';
```

## Assignment Evaluation

---

- The expression on the right-hand side of the assignment operator (=) is evaluated first.
- The result is used to set the value of the variable on the left-hand side of the assignment operator.

```
score = numberOfCards + handicap;
eggsPerBasket = eggsPerBasket - 2;
```

## Simple Input

---

- Sometimes the data needed for a computation are obtained from the user at run time.

- Keyboard input requires

```
import java.util.Scanner
```

at the beginning of the file.

## Simple Input

---

- Data can be entered from the keyboard using `Scanner keyboard = new Scanner(System.in);` followed, for example, by

```
eggsPerBasket = keyboard.nextInt();
```

which reads one `int` value from the keyboard and assigns it to `eggsPerBasket`.

## Simple Screen Output

```
System.out.println("The count is " + count);
```

- Outputs the string literal `"the count is "`
- Followed by the current value of the variable `count`.

```
import java.util.Scanner;
public class EggBasket2
{
    public static void main (String [] args)
    {
        int numberOfBaskets, eggsPerBasket, totalEggs;
        Scanner keyboard = new Scanner (System.in);
        System.out.println ("Enter the number of eggs in each basket:");
        eggsPerBasket = keyboard.nextInt ();
        System.out.println ("Enter the number of baskets:");
        numberOfBaskets = keyboard.nextInt ();
        totalEggs = numberOfBaskets * eggsPerBasket;
        System.out.println ("If you have");
        System.out.println (eggsPerBasket + " eggs per basket and");
        System.out.println (numberOfBaskets + " baskets, then");
        System.out.println ("the total number of eggs is " + totalEggs);
        System.out.println ("Now we take two eggs out of each basket.");
        eggsPerBasket = eggsPerBasket - 2;
        totalEggs = numberOfBaskets * eggsPerBasket;
        System.out.println ("You now have");
        System.out.println (eggsPerBasket + " eggs per basket and");
        System.out.println (numberOfBaskets + " baskets.");
        System.out.println ("The new total number of eggs is " + totalEggs);
    }
}
```

## Simple Input/output

```
Enter the number of eggs in each basket:
6
Enter the number of baskets:
10
If you have
6 eggs per basket and
10 baskets, then
the total number of eggs is 60
Now we take two eggs out of each basket.
You now have
4 eggs per basket and
10 baskets.
The new total number of eggs is 40
```

Sample  
screen  
output

## Constants

- Literal expressions such as `2`, `3.7`, or `'y'` are called *constants*.
- Integer constants can be preceded by a `+` or `-` sign, but cannot contain commas.
- Floating-point constants can be written
  - With digits after a decimal point or
  - Using *e notation*.

## e Notation

---

- e notation is also called *scientific notation* or *floating-point notation*.
- Examples
  - 865000000.0 can be written as 8.65e8
  - 0.000483 can be written as 4.83e-4
- The number in front of the **e** does not need to contain a decimal point.

## Imprecision in Floating-Point Numbers

---

- Floating-point numbers often are only approximations since they are stored with a finite number of bits.
- Hence 1.0/3.0 is slightly less than 1/3.
- 1.0/3.0 + 1.0/3.0 + 1.0/3.0 is less than 1.

## Named Constants

---

- Java provides mechanism to ...
    - Define a variable
    - Initialize it
    - Fix the value so it cannot be changed
- ```
final Type Variable = Constant;
```
- Example
- ```
final double PI = 3.14159;
```

## Assignment Compatibilities

---

- Java is said to be *strongly typed*.
  - You can't, for example, assign a floating point value to a variable declared to store an integer.
- Sometimes conversions between numbers are possible.

```
doubleVariable = 7;
```

is possible even if **doubleVariable** is of type **double**, for example.



## Assignment Compatibilities

---

- A value of one type can be assigned to a variable of any type further to the right

`byte --> short --> int --> long  
--> float --> double`

- But not to a variable of any type further to the left.
- You can assign a value of type `char` to a variable of type `int`.

## Type Casting

---

- A *type cast* temporarily changes the value of a variable from the declared type to some other type.
- For example,  
`double distance;  
distance = 9.0;  
int points;  
points = (int)distance;`
- Illegal without `(int)`

## Type Casting

---

- The value of `(int)distance` is `9`,
- The value of `distance`, both before and after the cast, is `9.0`.
- Any nonzero value to the right of the decimal point is *truncated* rather than *rounded*.

## Arithmetic Operators

---

- Arithmetic expressions can be formed using the `+`, `-`, `*`, and `/` operators together with variables or numbers referred to as *operands*.
  - When both operands are of the same type, the result is of that type.
  - When one of the operands is a floating-point type and the other is an integer, the result is a floating point type.

## Arithmetic Operations

---

- Example

If `hoursWorked` is an `int` to which the value `40` has been assigned, and `payRate` is a `double` to which `8.25` has been assigned

`hoursWorked * payRate`

is a `double` with a value of `500.0`.

## Arithmetic Operations

---

- Expressions with two or more operators can be viewed as a series of steps, each involving only two operands.
  - The result of one step produces one of the operands to be used in the next step.
- example  
`balance + (balance * rate)`

## The Division Operator

---

- The division operator (`/`) behaves as expected if one of the operands is a floating-point type.
- When both operands are integer types, the result is truncated, not rounded.
  - Hence, `99/100` has a value of `0`.

## The `mod` Operator

---

- The `mod` (`%`) operator is used with operators of integer type to obtain the remainder after integer division.
- 14 divided by 4 is 3 *with a remainder of 2*.
  - Hence, `14 % 4` is equal to `2`.
- The mod operator has many uses, including
  - determining if an integer is odd or even
  - determining if one integer is evenly divisible by another integer.

## Parentheses and Precedence

- Parentheses can communicate the order in which arithmetic operations are performed
- examples:  
`(cost + tax) * discount`  
`cost + (tax * discount)`
- Without parentheses, an expressions is evaluated according to the *rules of precedence*.

## Sample Expressions

- Figure 2.3 Some Arithmetic Expressions in Java

Ordinary Math	Java (Preferred Form)	Java (Parenthesized)
$rate^2 + delta$	<code>rate * rate + delta</code>	<code>(rate * rate) + delta</code>
$2(salary + bonus)$	<code>2 * (salary + bonus)</code>	<code>2 * (salary + bonus)</code>
$\frac{1}{time + 3mass}$	<code>1 / (time + 3 * mass)</code>	<code>1 / (time + (3 * mass))</code>
$\frac{a - 7}{t + 9v}$	<code>(a - 7) / (t + 9 * v)</code>	<code>(a - 7) / (t + (9 * v))</code>

## Specialized Assignment Operators

- Assignment operators can be combined with arithmetic operators (including `-`, `*`, `/`, and `%`, discussed later).  
`amount = amount + 5;`  
can be written as  
`amount += 5;`  
yielding the same results.

## Increment and Decrement Operators

- Used to increase (or decrease) the value of a variable by 1
- Easy to use, important to recognize
- The increment operator  
`count++` or `++count`
- The decrement operator  
`count--` or `--count`

## Increment and Decrement Operators

---

- equivalent operations

```
count++;  
++count;  
count = count + 1;
```

```
count--;  
--count;  
count = count - 1;
```

## Increment and Decrement Operators in Expressions

---

- after executing

```
int m = 4;  
int result = 3 * (++m)  
result has a value of 15 and m has a value of 5
```

- after executing

```
int m = 4;  
int result = 3 * (m++)  
result has a value of 12 and m has a value of 5
```

## The Class `String`

---

- We've used constants of type `String` already.

```
"Enter a whole number from 1 to 99."
```

- A value of type `String` is a
  - Sequence of characters
  - Treated as a single item.

## String Constants and Variables

---

- Declaring

```
String greeting;  
greeting = "Hello!";
```

or

```
String greeting = "Hello!";
```

or

```
String greeting = new String("Hello!");
```

- Printing

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

## Concatenation of Strings

---

- Two strings are *concatenated* using the `+` operator.

```
String greeting = "Hello";
String sentence;
sentence = greeting + " officer";
System.out.println(sentence);
```
- Any number of strings can be concatenated using the `+` operator.

## Concatenating Strings and Integers

---

```
String solution;
solution = "The answer is " + 42;
System.out.println (solution);
```



The answer is 42

## String Methods

---

- An object of the `String` class stores data consisting of a sequence of characters.
- Objects have methods as well as data
- The `length()` method returns the number of characters in a particular `String` object.

```
String greeting = "Hello";
int n = greeting.length();
```

## The Method `length()`

---

- The method `length()` returns an `int`.
- You can use a call to method `length()` anywhere an `int` can be used.

```
int count = command.length();
System.out.println("Length is " +
    command.length());
count = command.length() + 3;
```

# String Indices

- Figure 2.4

Indices	0	1	2	3	4	5	6	7	8	9	10	11
	J	a	v	a		i	s		f	u	n	.

- Positions start with 0, not 1.
  - The 'J' in "Java is fun." is in position 0
- A position is referred to as an *index*.
  - The 'f' in "Java is fun." is at index 8.

## String Methods

<b>charAt(<i>Index</i>)</b>
Returns the character at <i>Index</i> in this string. Index numbers begin at 0.
<b>compareTo(<i>A_String</i>)</b>
Compares this string with <i>A_String</i> to see which string comes first in the lexicographic ordering. (Lexicographic ordering is the same as alphabetical ordering when both strings are either all uppercase letters or all lowercase letters.) Returns a negative integer if this string is first, returns zero if the two strings are equal, and returns a positive integer if <i>A_String</i> is first.
<b>concat(<i>A_String</i>)</b>
Returns a new string having the same characters as this string concatenated with the characters in <i>A_String</i> . You can use the <code>⋈</code> operator instead of <b>concat</b> .
<b>equals(<i>Other_String</i>)</b>
Returns true if this string and <i>Other_String</i> are equal. Otherwise, returns false.

Figure 2.5a

## String Methods

<b>equalsIgnoreCase(<i>Other_String</i>)</b>
Behaves like the method <b>equals</b> , but considers uppercase and lowercase versions of a letter to be the same.
<b>indexOf(<i>A_String</i>)</b>
Returns the index of the first occurrence of the substring <i>A_String</i> within this string. Returns -1 if <i>A_String</i> is not found. Index numbers begin at 0.
<b>lastIndexOf(<i>A_String</i>)</b>
Returns the index of the last occurrence of the substring <i>A_String</i> within this string. Returns -1 if <i>A_String</i> is not found. Index numbers begin at 0.

Figure 2.5b

## String Methods

<b>length()</b>
Returns the length of this string.
<b>toLowerCase()</b>
Returns a new string having the same characters as this string, but with any uppercase letters converted to lowercase.
<b>toUpperCase()</b>
Returns a new string having the same characters as this string, but with any lowercase letters converted to uppercase.

Figure 2.5c

## String Processing

- No methods allow you to change the value of a **String** object.
- But you can change the value of a **String** variable.

```
Text processing is hard!  
012345678901234567890123  
The word "hard" starts at index 19  
The changed string is:  
TEXT PROCESSING IS EASY!
```

Sample  
Screen  
Output

## Escape Characters

```
\ " Double quote.  
\ ' Single quote.  
\ \ Backslash.  
\ n New line. Go to the beginning of the next line.  
\ r Carriage return. Go to the beginning of the current line.  
\ t Tab. Add whitespace up to the next tab stop.
```

- Figure 2.6
- Each escape sequence is a single character even though it is written with two symbols.

## Escape Characters

- How would you print  
    **"Java" refers to a language.** ?
- The compiler needs to be told that the quotation marks (") do not signal the start or end of a string, but instead are to be printed.

```
System.out.println(  
    "\"Java\" refers to a language.");
```

## Examples

```
System.out.println("abc\\def");
```

```
abc\def
```

```
System.out.println("new\nline");
```

```
new  
line
```

```
char singleQuote = '\'';  
System.out.println(singleQuote);
```

```
'
```

```

public class StringDemo
{
    public static void main (String [] args)
    {
        String sentence = "Text processing is hard!";
        int position = sentence.indexOf ("hard");
        System.out.println (sentence);
        System.out.println ("012345678901234567890123");
        System.out.println ("The word \"hard\" starts at index " + position);
        sentence = sentence.substring (0, position) + "easy!";
        sentence = sentence.toUpperCase ();
        System.out.println ("The changed string is:");
        System.out.println (sentence);
    }
}

```

## Screen Output

- We've seen several examples of screen output already.
- `System.out` is an object that is part of Java.
- `println()` is one of the methods available to the `System.out` object.

## Screen Output

- The concatenation operator (+) is useful when everything does not fit on one line.  
`System.out.println("Lucky number = " + 13  
+ "Secret number = " + number);`
- Do not break the line except immediately before or after the concatenation operator (+).

## Screen Output

- Alternatively, use `print()`  
`System.out.print("One, two,");`  
`System.out.print(" buckle my shoe.");`  
`System.out.println(" Three, four,");`  
`System.out.println(" shut the door.");`  
 ending with a `println()`.



## Keyboard Input

- Java has reasonable facilities for handling keyboard input.
- These facilities are provided by the **Scanner** class in the **java.util** package.
  - A *package* is a library of classes.

## Using the Scanner Class

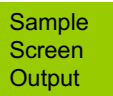
- Near the beginning of your program, insert  

```
import java.util.Scanner;
```
- Create an object of the **Scanner** class  

```
Scanner keyboard = new Scanner (System.in)
```
- Read data (an **int** or a **double**, for example)  

```
int n1 = keyboard.nextInt();  
double d1 = keyboard.nextDouble();
```

## Keyboard Input Demonstration



```
Enter two whole numbers  
separated by one or more spaces:  
42 43  
You entered 42 and 43  
Next enter two numbers.  
A decimal point is OK.  
9.99 21  
You entered 9.99 and 21.0  
Next enter two words:  
plastic spoons  
You entered "plastic" and "spoons"  
Next enter a line of text:  
May the hair on your toes grow long and curly.  
You entered "May the hair on your toes grow long and curly."
```

```
import java.util.Scanner;  
public class ScannerDemo  
{  
    public static void main (String [] args)  
    {  
        Scanner keyboard = new Scanner (System.in);  
        System.out.println ("Enter two whole numbers\nseparated by one or more spaces:");  
        int n1, n2;  
        n1 = keyboard.nextInt (); n2 = keyboard.nextInt ();  
        System.out.println ("You entered " + n1 + " and " + n2);  
        System.out.println ("Next enter two numbers.\nA decimal point is OK.");  
        double d1, d2;  
        d1 = keyboard.nextDouble (); d2 = keyboard.nextDouble ();  
        System.out.println ("You entered " + d1 + " and " + d2);  
        System.out.println ("Next enter two words:");  
        String s1, s2;  
        s1 = keyboard.next (); s2 = keyboard.next ();  
        System.out.println ("You entered \"\" + s1 + \"\" and \"\" + s2 + \"\"");  
        s1 = keyboard.nextLine (); //To get rid of '\n'  
        System.out.println ("Next enter a line of text:");  
        s1 = keyboard.nextLine ();  
        System.out.println ("You entered: \"\" + s1 + \"\"");  
    }  
}
```

## Some **Scanner** Class Methods

- Figure 2.7a

```
Scanner_Object_Name.next()
Returns the String value consisting of the next keyboard characters up to, but not including, the first delimiter character. The default delimiters are whitespace characters.

Scanner_Object_Name.nextLine()
Reads the rest of the current keyboard input line and returns the characters read as a value of type String. Note that the line terminator '\n' is read and discarded; it is not included in the string returned.

Scanner_Object_Name.nextInt()
Returns the next keyboard input as a value of type int.

Scanner_Object_Name.nextDouble()
Returns the next keyboard input as a value of type double.

Scanner_Object_Name.nextFloat()
Returns the next keyboard input as a value of type float.
```

2/26/17  
Dr Regina Berretta



## nextLine () Method Caution

- The **nextLine ()** method reads
  - The remainder of the current line,
  - Even if it is empty.

2/26/17  
Dr Regina Berretta



## Some **Scanner** Class Methods

- Figure 2.7b

```
Scanner_Object_Name.nextLong()
Returns the next keyboard input as a value of type long.

Scanner_Object_Name.nextByte()
Returns the next keyboard input as a value of type byte.

Scanner_Object_Name.nextShort()
Returns the next keyboard input as a value of type short.

Scanner_Object_Name.nextBoolean()
Returns the next keyboard input as a value of type boolean. The values of true and false are entered as the words true and false. Any combination of uppercase and lowercase letters is allowed in spelling true and false.

Scanner_Object_Name.useDelimiter(Delimiter_Word);
Makes the string Delimiter_Word the only delimiter used to separate input. Only the exact word will be a delimiter. In particular, blanks, line breaks, and other whitespace will no longer be delimiters unless they are a part of Delimiter_Word.

This is a simple case of the use of the useDelimiter method. There are many ways to set the delimiters to various combinations of characters and words, but we will not go into them in this book.
```

2/26/17  
Dr Regina Berretta



## nextLine () Method Caution

- Example – given following declaration.

```
int n;
String s1, s2;
n = keyboard.nextInt();
s1 = keyboard.nextLine();
s2 = keyboard.nextLine();
```

- Assume input shown

**n** is set to **42**  
but **s1** is set to the empty string.

**42**  
and don't you  
forget it.

2/26/17  
Dr Regina Berretta



## Documentation and Style

---

- Most programs are modified over time to respond to new requirements.
- Programs which are easy to read and understand are easy to modify.
- Even if it will be used only once, you have to read it in order to debug it .

2/26/17  
Dr Regina Berretta



## Meaningful Variable Names

---

- A variable's name should suggest its use.
- Observe conventions in choosing names for variables.
  - Use only letters and digits.
  - "Punctuate" using uppercase letters at word boundaries (e.g. `taxRate`).
  - Start variables with lowercase letters.
  - Start class names with uppercase letters.

2/26/17  
Dr Regina Berretta



## Comments

---

- The best programs are self-documenting.
  - Clean style
  - Well-chosen names
- Comments are written into a program as needed explain the program.
  - They are useful to the programmer, but they are ignored by the compiler.

2/26/17  
Dr Regina Berretta



## Comments

---

- A comment can begin with `//`.
- Everything after these symbols and to the end of the line is treated as a comment and is ignored by the compiler.

```
double radius; //in centimeters
```

2/26/17  
Dr Regina Berretta



## Comments

- A comment can begin with `/*` and end with `*/`
- Everything between these symbols is treated as a comment and is ignored by the compiler.  

```
/**  
This program should only  
be used on alternate Thursdays,  
except during leap years, when it should  
only be used on alternate Tuesdays.  
*/
```

## When to Use Comments

- Begin each program file with an explanatory comment
  - What the program does
  - The name of the author
  - Contact information for the author
  - Date of the last modification.
- Provide only those comments which the expected reader of the program file will need in order to understand it.

## Comments Example

```
Enter the radius of a circle in inches:  
2.5  
A circle of radius 2.5 inches  
has an area of 19.6349375 square inches.
```

Sample  
Screen  
Output

```
import java.util.Scanner;  
  
/** Program to compute area of a circle.  
Author: Jane Q. Programmer.  
E-mail Address: janeq@somemachine.etc.etc.  
Programming Assignment 2.  
Last Changed: October 7, 2008. */  
  
public class CircleCalculation  
{  
    public static void main (String [] args)  
    {  
        double radius; //in inches  
        double area; //in square inches  
        Scanner keyboard = new Scanner (System.in);  
        System.out.println ("Enter the radius of a circle in inches:");  
        radius = keyboard.nextDouble ();  
        area = 3.14159 * radius * radius;  
        System.out.println ("A circle of radius " + radius + " inches");  
        System.out.println ("has an area of " + area + " square inches.");  
    }  
}
```

## Indentation

---

- Indentation should communicate nesting clearly.
- A good choice is four spaces (~1 tab space) for each level of indentation.
- Indentation should be consistent.
- Indentation should be used for second and subsequent lines of statements which do not fit on a single line.

## Indentation

---

- Indentation does not change the behavior of the program.
- Proper indentation helps communicate to the human reader the nested structures of the program

## Using Named Constants

---

- To avoid confusion, always name constants (and variables).  

```
area = PI * radius * radius;
```

is clearer than  

```
area = 3.14159 * radius * radius;
```
- Place constants near the beginning of the program.

## Named Constants

---

- Once the value of a constant is set (or changed by an editor), it can be used (or reflected) throughout the program.

```
public static final double INTEREST_RATE = 6.65;
```

- If a literal (such as 6.65) is used instead, every occurrence must be changed, with the risk that another literal with the same value might be changed unintentionally.

## Declaring Constants

- Syntax

```
public static final Variable_Type = Constant;
```

- Examples

```
public static final double PI = 3.14159;  
public static final String MOTTO = "The customer is  
    always right.";
```

- By convention, uppercase letters are used for constants.

2/26/17  
Dr Regina Berretta



## Named Constants

```
import java.util.Scanner;  
  
/** Program to compute area of a circle.  
Author: Jane Q. Programmer.  
E-mail Address: janeq@somemachine.etc.etc.  
Programming Assignment 2.  
Last Changed: October 7, 2008. */
```

```
public class CircleCalculation  
{  
    public static final double PI = 3.14159;  
    public static void main (String [] args)  
    {  
        double radius; //in inches  
        double area; //in square inches  
        Scanner keyboard = new Scanner (System.in);  
        System.out.println ("Enter the radius of a circle in inches:");  
        radius = keyboard.nextDouble ();  
        area = PI * radius * radius;  
        System.out.println ("A circle of radius " + radius + " inches");  
        System.out.println ("has an area of " + area + " square inches.");  
    }  
}
```

2/26/17  
Dr Regina Berretta



## Example - salary

- Work out in a code to calculate the salary in the end of a month. You know that:
  - The salary rate is A\$10/hour during normal hours and A\$15/hour for extra hours.

Enter the radius of a circle in inches:

2.5

A circle of radius 2.5 inches  
has an area of 19.6349375 square inches.

Sample  
Screen  
Output

2/26/17  
Dr Regina Berretta



## Salary pseudocode

---

```
input normalhours
input extrahours
salary = 10*normalhours +15*extrahours
output salary
```



## The Structure of Simple Java Programs

---

```
import java.util.*;
public class <name of class>
{
    public static void main (String[] args)
    {
        Scanner console = new Scanner(System.in);
        <declare the model's variables>
        <what needs to be done when the program is run>
    }
}
```



## The Structure of Simple Java Programs

---

```
import java.util.*;
public class Salary
{
    public static void main (String[] args)
    {
        Scanner console = new Scanner(System.in);
        double normal,extra,total;
        System.out.print("Please Enter number of normal hours: ");
        normal = console.nextDouble();
        System.out.print("Please Enter number of extra hours: ");
        extra = console.nextDouble();
        total = 10*normal+15*extra;
        System.out.print("total salary is "+total);
    }
}
```



## Example - SMS Message Costing

---

92

- Our model of a set of messages
  - Input number of Messages (it will be a whole number)
  - Unit Cost of each Message (say 22 cents per message)
  - Setup cost = 10 dollars
  - Output total cost



## The Structure of Simple Java Programs

93

```
import java.util.*;

public class <name of class>
{
    public static void main (String[] args)
    {
        Scanner console = new Scanner(System.in);
        <declare the model's variables>
        <what needs to be done when the program is run>
    }
}
```

Try to write the code for the SMS cost example

## Your task

94

- Read
  - Chapter 2 of the text book
- Exercises
  - Implement/compile/run the examples from lecture slides
  - In the book
  - Computer lab

