ELEMENTARY PROGRAMMING

Writing a Simple Program

- computing the area of a circle
- Algorithm:
  - 1. Read in the circle's radius.
  - 2. Compute the area using the following formula:
    - *area* = *radius* * *radius* * p
  - 3. Display the result

Step 1:

- Java program begins with a class definition
- keyword **class** is followed by the class name

- **public class** ComputeArea {
- // Details to be given later
- }

Step 2:

- Java program must have a **main** method where program execution begins
- **public class** ComputeArea {
- **public static void** main(String[] args) {
- // Step 1: Read in radius
- // Step 2: Compute area
- // Step 3: Display the area
- }
- }

- Reading the radius.
- Storing the radius in the program
- Two variables with their data types are required
- *Primitive data types* or *fundamental types??*

- **public class** ComputeArea {
- **public static void** main(String[] args) {
- **double** radius;
- **double** area;
- // Step 1: Read in radius
- // Step 2: Compute area
- // Step 3: Display the area
- }
- }

- public class ComputeArea {
- public static void main(String[] args) {
- double radius; // Declare radius
- double area; // Declare area
- //Assign a radius
- radius = **20**; // radius is now 20
- // Compute area
- area = radius * radius * **3.14159**;
- // Display results
- System.out.println("**The area for the circle of radius** " + radius + " **is** " + area);
- }
- }

- The plus sign (+) has two meanings: one for addition and the other for concatenating (combining) strings.
- A string cannot cross lines in the source code
  - System.out.println("**Introduction to Java Programming,**

    by Y. Daniel Liang"); //error
  - System.out.println("**Introduction to Java Programming, "** +

    "by Y. Daniel Liang");

```
public class Test {
public void main(string[] args) {
double i = 50.0;
double k = i + 50.0;
double j = k + 1;
System.out.println("j is " + j + " and
k is " + k);
}
}
```

Reading Input from the Console

- **Scanner** class for console input
- **System.out** to refer to the standard output device and **System.in** to the standard input device
- Scanner input = **new** Scanner(System.in);
- **double** radius = input.nextDouble();
- **nextDouble()** method to read a **double** value

```
import java.util.Scanner; // Scanner is in the java.util package
public class ComputeAreaWithConsoleInput {
public static void main(String[] args) {
// Create a Scanner object
Scanner input = new Scanner(System.in);
// Prompt the user to enter a radius
System.out.print("Enter a number for radius: ");
double radius = input.nextDouble();
// Compute area
double area = radius * radius * 3.14159;
// Display results
System.out.println("The area for the circle of radius " + radius + " is " +
area);
 }
 }
```

Exercise

- WAP program to read three numbers and displays their average.

Identifiers

- Identifiers are the names that identify the elements such as classes, methods, and variables in a program
- Rules for identifiers:
  - An identifier consists of letters, digits, underscores (_), and dollar signs ($)
  - An identifier must start with a letter, an underscore (_), or a dollar sign ($). It cannot start with a digit
  - An identifier cannot be a reserved word.
  - An identifier cannot be true, false, or null.
  - An identifier can be of any length.

Variables

- Variables are used to represent values that may be changed in the program
- syntax for declaring a variable
  - datatype variableName;
  - datatype variable1, variable2, ..., variablen;
- Variable initialization
  - int count = 1;
  - int i = 1, j = 2;
- A variable must be declared before it can be assigned a value.

```
public class Test {
public static void main(String[] args) {
int i = k + 2;
System.out.println(i);
}
}
```

Assignment Statements and Assignment Expressions

- An assignment statement designates a value for a variable
- syntax for assignment statements
  - variable = expression;
- An expression represents a computation involving values, variables, and operators
- To assign a value to a variable, you must place the variable name to the left of the assignment operator.
- assignment statement is also known as an assignment expression
  - System.out.println(x = 1); equivalent to
  - x = 1;

    System.out.println(x);

Contd…

- i = j = k = 1; equivalent to
  - k = 1;
  - j = k;
  - i = j;
- In an assignment statement, the data type of the variable on the left must be compatible with the data type of the value on the right.

- public class Test {
- public static void main(String[] args) {
- int i = j = k = 2;
- System.out.println(i + " " + j + " " + k);
- }
- }

Named Constants

- A named constant is an identifier that represents a permanent value
- syntax for declaring a constant
  - **final** datatype CONSTANTNAME = value;
- constant must be declared and initialized in the same statement
- word **final** is a Java keyword for declaring a constant
- benefits of using constants:
  - don't have to repeatedly type the same value if it is used multiple times
  - if you have to change the constant value, you need to change it only in a single location in the source code
  - descriptive name for a constant makes the program easy to read

Naming Conventions

- Make sure that you choose descriptive names with straightforward meanings for the variables, constants, classes, and methods
- Use lowercase for variables and methods
  - the variables **radius** and **area** and the method **print**
- Capitalize the first letter of each word in a class name
  - the class names ComputeArea and System
- Capitalize every letter in a constant, and use underscores between words
  - constants PI and MAX_VALUE
- makes programs easy to read

Numeric Data Types and Operations

- Java has six numeric types for integers and floating-point numbers with operators +, -, *, /, and %.
- Java uses four types for integers: byte, short, int, and long
- two types for floating-point numbers: **float** and **double**

Name	*Range*	*Storage Size*	
`byte`	-2^7 to $2^7 - 1$ (−128 to 127)	8-bit signed	`byte` type
`short`	-2^{15} to $2^{15} - 1$ (−32768 to 32767)	16-bit signed	`short` type
`int`	-2^{31} to $2^{31} - 1$ (−2147483648 to 2147483647)	32-bit signed	`int` type
`long`	-2^{63} to $2^{63} - 1$	64-bit signed	`long` type
	(i.e., −9223372036854775808 to 9223372036854775807)		
`float`	Negative range: −3.4028235E + 38 to −1.4E − 45	32-bit IEEE 754	`float` type
	Positive range: 1.4E − 45 to 3.4028235E + 38		
`double`	Negative range: −1.7976931348623157E + 308 to −4.9E − 324	64-bit IEEE 754	`double` type
	Positive range: 4.9E − 324 to 1.7976931348623157E + 308		

Reading Numbers from the Keyboard

Method	*Description*
`nextByte()`	reads an integer of the `byte` type.
`nextShort()`	reads an integer of the `short` type.
`nextInt()`	reads an integer of the `int` type.
`nextLong()`	reads an integer of the `long` type.
`nextFloat()`	reads a number of the `float` type.
`nextDouble()`	reads a number of the `double` type.

Numeric Operators

Name	*Meaning*	*Example*	*Result*
+	Addition	34 + 1	35
-	Subtraction	34.0 – 0.1	33.9
*	Multiplication	300 * 30	9000
/	Division	1.0 / 2.0	0.5
%	Remainder	20 % 3	2

Exponent Operations

- Math.pow(a, b) method can be used to compute a^b
  - System.out.println(Math.pow(2, 3)); // Displays 8.0
  - System.out.println(Math.pow(4, 0.5)); // Displays 2.0
  - System.out.println(Math.pow(2.5, 2)); // Displays 6.25
  - System.out.println(Math.pow(2.5, -2)); // Displays 0.16

Integer Literals

- An integer literal can be assigned to an integer variable
  - System.out.println(0B1111); // Displays 15
  - System.out.println(07777); // Displays 4095
  - System.out.println(0XFFFF); // Displays 65535
- to denote a binary integer literal, use a leading ***0b*** or ***0B***
- to denote an octal integer literal, use a leading ***0***
- to denote a hexadecimal integer literal, use a leading ***0x*** or ***0X***

Floating-Point Literals

- By default, a floating-point literal is treated as a **double** type value
- make a number a **float** by appending the letter **f** or **F**
  - **100.2f** or **100.2F**
- Make a number a **double** by appending the letter **d** or **D**
  - **100.2d** or **100.2D**
- **double** type values are more accurate than the **float** type values
- System.out.println("1.0 / 3.0 is " + 1.0 / 3.0);
  - displays 1.0 / 3.0 is 0.3333333333333333 (16 digits)
- System.out.println("1.0F / 3.0F is " + 1.0F / 3.0F);
  - displays 1.0F / 3.0F is 0.33333334 (8 digits)

Scientific Notation

- Floating-point literals can be written in scientific notation in the form of $a * 10^b$
  - 123.456 is $1.23456 * 10^2$
  - 0.0123456 is $1.23456 * 10^{-2}$
- $1.23456 * 10^2$ is written as 1.23456E2 or 1.23456E+2
- $1.23456 * 10^{-2}$ as **1.23456E-2**
- Java allows you to use underscores between two digits in a number literal
  - **long** ssn = **232_45_4519**;
  - **long** creditCardNumber = **2324_4545_4519_3415L**;

Evaluating Expressions and Operator Precedence

- parentheses are evaluated
- Multiplication, division, and remainder operators are applied first (left to right)
- Addition and subtraction operators are applied last (left to right)
  - 3 + 4 * 4 + 5 * (4 + 3) - 1
- WAP that converts a Fahrenheit degree to Celsius using the formula celsius = (5/9)(fahrenheit - 32)

Augmented Assignment Operators

- *operators +, -, *, /, and % can be combined with the assignment operator to form augmented operators*
  - count = count + 1; can be written as count += 1;

Operator	*Name*	*Example*	*Equivalent*
+=	Addition assignment	`i += 8`	`i = i + 8`
-=	Subtraction assignment	`i -= 8`	`i = i - 8`
*=	Multiplication assignment	`i *= 8`	`i = i * 8`
/=	Division assignment	`i /= 8`	`i = i / 8`
%=	Remainder assignment	`i %= 8`	`i = i % 8`

Contd…

- The augmented assignment operator is performed last after all the other operators in the expression are evaluated
- x += **2**; // Statement

  System.out.println(x += **2**); // Expression
- **double** a = **6.5**;

  a += a + **1**;

  System.out.println(a);

  a = **6**;

  a /= **2**;

  System.out.println(a);

Increment and Decrement Operators

- *increment operator (++) and decrement operator (- -) are for incrementing and decrementing a variable by 1.*
- Postfix increment and postfix decrement
- Prefix increment and prefix decrement

Operator	*Name*	*Description*	*Example (assume i = 1)*
`++var`	preincrement	Increment `var` by `1`, and use the new `var` value in the statement	`int j = ++i;` `// j is 2, i is 2`
`var++`	postincrement	Increment `var` by `1`, but use the original `var` value in the statement	`int j = i++;` `// j is 1, i is 2`
`--var`	predecrement	Decrement `var` by `1`, and use the new `var` value in the statement	`int j = --i;` `// j is 0, i is 0`
`var--`	postdecrement	Decrement `var` by `1`, and use the original `var` value in the statement	`int j = i--;` `// j is 1, i is 0`

- ```
  int i = 10;
  int newNum = 10 * i++;
  System.out.print("i is " + i + ", newNum is " + newNum);
  ```
- ```
  int i = 10;
  int newNum = 10 * (++i);
  System.out.print("i is " + i + ", newNum is " + newNum);
  ```
- ```
  double x = 1.0;
  double y = 5.0;
  double z = x-- + (++y);
  ```
  ```

Numeric Type Conversions

- Floating-point numbers can be converted into integers using explicit casting.
- You can always assign a value to a numeric variable whose type supports a larger range of values
- You cannot assign a value to a variable of a type with a smaller range unless you use *type casting*
- Java will automatically widen a type, but you must narrow a type explicitly
- System.out.println((**int**)1.7);
- System.out.println((**double**)1 / 2);
- System.out.println(1 / 2);

- Write a program that reads an integer and adds all the digits in the integer. For example, if an integer is 932, the sum of all its digits is 14.
- Write a program that prompts the user to enter two points (x1, y1) and (x2, y2) and displays their distance between them. The formula for computing the distance is $\sqrt{(x2 - x1)2 + (y2 - y1)2}$. Note that you can use Math.pow(a, 0.5) to compute $\sqrt{a}$.