CS 1233 Object Oriented and Design

Created by Eddy C. Borera, PhD

Reminders

- Due this week:
  - Hello World
  - Sponge Bob

Java Primitive Data Types

There are 8 primitive data types within Java:

- int
- double
- short
- char
- boolean
- long
- float
- byte

String is not a primitive type.

Data Types

Type	description	literal values
int	integers	-2, 0, 1, 456
char	character	'a' , 'Z'
double	floating-point numbers	3.1415
boolean	truth values	*true* , *false*
String	sequence of characters	"abcdEf 124 $2 +"

Variables

Definition:

```
Variable represents a memory location used to store data
```

Example:

```
int age; // declaring an integer variable
```

```
double weight; // declaring a floating-point variable
```

```
String str; // declaring a string variable
```

Assignments

An assignment statement like `age = 20;` stores the right-side item's current value into the variable on the left side.

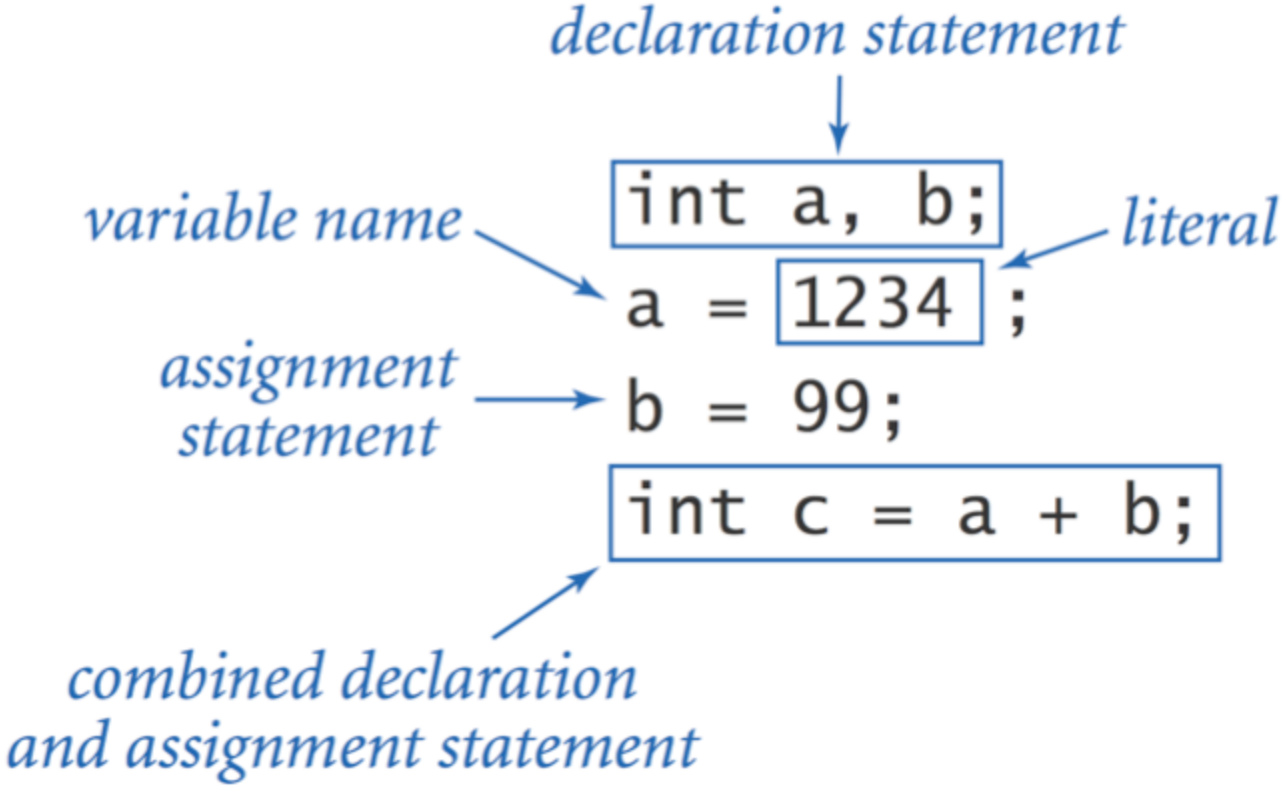
```
variableName = expression;
```

Example

```
public class GradeTotal {
  public class static void main(String[] args){
    double CS1233 = 92.5;
    double BY1003 = 90.0;
    double BI4213 = 98.0;

    double total = CS1233 + BY1003 + BI4213;
    double average = total / 3.0;
  }
}
```

Variable Declaration



Variable Declaration

Trace. Table of variable values after each statement.

	a	b	t
`int a, b;`	*undefined*	*undefined*	
`a = 1234;`	1234	*undefined*	
`b = 99;`	1234	99	
`int t = a;`	1234	99	1234
`a = b;`	99	99	1234
`b = t;`	99	1234	1234

Identifiers

```
A name created by a programmer for an item like
a variable or method is called an identifier.
```

- An identifier must be a sequence of letters (a-z, A-Z, _, $) and digits (0-9) and must start with a letter.
- Note that "_", called an underscore, and "$", called a dollar sign or currency symbol, are considered to be letters.
- A good practice followed by many Java programmers is to not use _ or $ in programmer-created identifiers.

Identifiers (1)

Valid Identifiers:

- c
- cat
- num
- Num
- _value
- num5

Identifiers are case sensitive, meaning upper and lower case letters differ. So numCats and NumCats are different.

Invalid Identifiers (1)

Invalid Identifiers:

- 12Num
- my var
- num!
- class
- main

Inavalid Identifiers (2)

- A reserved word is a word that is part of the language, like class, main, public, int, short, or double.
- A reserved word is also known as a keyword. A programmer cannot use a reserved word as an identifier.

Which are valid identifiers?

A. numCars

B. num_cars

C. num_Cars1

D. _numCars

E. __numCars

F. num cars

G. tall

H. short

Integers

- ..., -3, -2, -1, 0, 1, 2, 3, ...
- 32-bit data type
- integer values are between -2^{31} and $+2^{32} - 1$
- operators: + , -, * , %, /

operator	operations	example
add	+	3 + 2
substract	-	10 - 2
multiply	*	6 * 5
divide	/	10 / 3
remainder	%	10 % 3

Example (Integers)

```
public class IntOps {
   public static void main(String[] args) {
      int a = Integer.parseInt(args[0]);
      int b = Integer.parseInt(args[1]);
      int sum  = a + b;
      int prod = a * b;
      int quot = a / b;
      int rem  = a % b;
      System.out.println(a + " + " + b + " = " + sum);
      System.out.println(a + " * " + b + " = " + prod);
      System.out.println(a + " / " + b + " = " + quot);
      System.out.println(a + " % " + b + " = " + rem);
   }
}
```

```
$> javac IntOps.java
$> java IntOps 1234 99
1234 + 99 = 1333
1234 * 99 = 122166
1234 / 99 = 12
1234 % 99 = 46
```

Floating-Point Numbers

Double data type:

- real numbers
- 64-bit data type
- operators: + , -, * , %, /

operator	operations	example	Value
add	+	1.13 + 0.31	1.44
substract	-	2.14 - 0.02	2.12
multiply	*	2.0 * 2.5	5.0
divide	/	1.0 / 0.0	Infinity
remainder	%	10 % 3.14	0.58

Solve quadratic equation $x^2 + bx + c = 0$.

$$\text{roots} = \frac{-b \pm \sqrt{b^2 - 4c}}{2}$$

```
public class Quadratic {
   public static void main(String[] args) {

      // parse coefficients from command-line
      double b = Double.parseDouble(args[0]);
      double c = Double.parseDouble(args[1]);

      // calculate roots
      double discriminant = b*b - 4.0*c;
      double d = Math.sqrt(discriminant);
      double root1 = (-b + d) / 2.0;
      double root2 = (-b - d) / 2.0;

      // print them out
      System.out.println(root1);
      System.out.println(root2);
   }
}
```

Character

- A variable of char type can store a single character.
  Example, the letter m or the symbol %.
- A character literal is surrounded with single quotes, as in 'm' or '%'.

```
public class CharArrow {
   public static void main (String [] args) {
      char arrowBody = '-';
      char arrowHead = '>';

      System.out.println(arrowHead);
      System.out.println("" + arrowBody + arrowHead);

      arrowBody = 'o';

      System.out.println("" + arrowBody + arrowHead);
   }
}
```

Escape Sequences

A character preceded by a backslash () is an escape sequence and has special meaning to the compiler.

Escape	Description
`\t`	tab
`\n`	newline
`\b`	backspace
`\'`	single quote
`\"`	double quote
`\\`	backslash

String Data Type

- a string is a sequence of characters.
- a string literal uses double quotes

```
public class Ruler {
   public static void main(String[] args) {
      String ruler1 = "1";
      String ruler2 = ruler1 + " 2 " + ruler1;
      String ruler3 = ruler2 + " 3 " + ruler2;
      System.out.println(ruler3);
   }
}
```

Output:

```
1 2 1 3 1 2 1
```

Boolean Data Type

- Boolean refers to a quantity that has only two possible values, true or false.

a	!a
true	false
false	true

a	b	a && b	a || b
false	false	false	false
false	true	false	true
true	false	false	true
true	true	true	true

Data conversion

Type conversion. Convert value from one data type to another.

expression	*expression type*	*expression value*
`"1234" + 99`	`String`	`"123499"`
`Integer.parseInt("123")`	`int`	`123`
`(int) 2.71828`	`int`	`2`
`Math.round(2.71828)`	`long`	`3`
`(int) Math.round(2.71828)`	`int`	`3`
`(int) Math.round(3.14159)`	`int`	`3`
`11 * 0.3`	`double`	`3.3`
`(int) 11 * 0.3`	`double`	`3.3`
`11 * (int) 0.3`	`int`	`0`
`(int) (11 * 0.3)`	`int`	`3`

Examples

```
public class RandomInt {
   public static void main(String[] args) {
      int N = Integer.parseInt(args[0]);
      double r = Math.random();
      int n = (int) (r * N);

      System.out.println("random integer is " + n);
   }
}
```

```
$> java RandomInt 6
random integer is 3
$> java RandomInt 10000
random integer is 3184
```

Constant

- A good practice is to minimize the use of literal numbers in code.
- One reason is to improve code readability.
  - `newPrice = origPrice - 5`
  - vs
  - `newPrice = origPrice - priceDiscount`
- An initialized variable whose value cannot change is called a constant variable (a.k.a final variable).

Constant (1)

```
import java.util.Scanner;

public class Circle {
   public static void main (String[] args) {
      Scanner scnr = new Scanner(System.in);
      final double PI_VAL = 3.14159;

      System.out.println("Enter the radius:");
      double radius = scnr.nextDouble();

      double area = PI_VAL * radius * radius;
      double circ = 2 * PI_VAL * radius;

      System.out.println("Area: " + area);
      System.out.println("Circumference: " + circ);
   }
}
```

```java
import java.util.Scanner;

public class MortgageSystem {
   public static void main (String[] args) {
      final double INTERST_RATE = 0.065; // 6.5%
      final String BANK_NAME = "Bank of America";
      ...
   }
}
```

Debugging a Code

- Debugging is the process of determining and fixing the cause of a problem in a computer program.
  Debugging Process:

Let's Debug this code

```
import java.util.Scanner;

public class CircumferenceToArea {
   public static void main (String [] args) {
      Scanner scnr = new Scanner(System.in);
      double circleRadius        = 0.0;
      double circleCircumference = 0.0;
      double circleArea          = 0.0;
      final double PI_VAL        = 3.14159265;

      System.out.print("Enter circumference: ");
      circleCircumference = scnr.nextDouble();

      circleRadius = circleCircumference / 2 * PI_VAL;
      circleArea = PI_VAL * circleRadius * circleRadius;

      System.out.println("Circle area is: " + circleArea);
   }
}
```

```
$> Enter circumference: 10
Circle area is: 775.1569143502577
```

Let's Debug the code

- If circumference is 10, then radius is `10 / 2 * PI_VAL`, so about 1.6.
- The area is then `PI_VAL * 1.6 * 1.6`, or about 8, but the program outputs about 775.

Predict problem is bad output

```
import java.util.Scanner;
public class CircumferenceToArea {
   public static void main (String [] args) {
      Scanner scnr = new Scanner(System.in);
      double circleRadius        = 0.0;
      double circleCircumference = 0.0;
      double circleArea          = 0.0;
      final double PI_VAL        = 3.14159265;

      System.out.print("Enter circumference: ");
      circleCircumference = scnr.nextDouble();

      circleRadius = circleCircumference / 2 * PI_VAL;
      circleArea = PI_VAL * circleRadius * circleRadius;

      circleArea = 999; // FIXME delete
      System.out.println("Circle area is: " + circleArea);
   }
}
```

```
$> Enter circumference: 0
Circle area is: 999
```

Predict computation is bad

```
import java.util.Scanner;
public class CircumferenceToArea {
   public static void main(String[] args) {
      Scanner scnr = new Scanner(System.in);
      double circleRadius        = 0.0;
      double circleCircumference = 0.0;
      double circleArea          = 0.0;
      final double PI_VAL        = 3.14159265;

      System.out.print("Enter circumference: ");
      circleCircumference = scnr.nextDouble();
      circleRadius = circleCircumference / 2 * PI_VAL;

      circleRadius = 0.5; // FIXME delete
      circleArea = PI_VAL * circleRadius * circleRadius;

      System.out.println("Circle area is: " + circleArea);
   }
}
```

```
$> Enter circumference: 0
Circle area is: 0.7853981625
```

Predict problem is bad radius computation.

```
import java.util.Scanner;

public class CircumferenceToArea {
   public static void main(String[] args) {
      Scanner scnr = new Scanner(System.in);
      double circleRadius        = 0.0;
      double circleCircumference = 0.0;
      double circleArea          = 0.0;
      final double PI_VAL        = 3.14159265;

      System.out.print("Enter circumference: ");
      circleCircumference = scnr.nextDouble();

      circleRadius = circleCircumference / 2 * PI_VAL;
      System.out.println("Radius: " + circleRadius); // FIXME

   }
}
```

Debugging process

- Manually set a variable to a value.
- Insert print statements to observe variable values.
- Comment out unused code.
- Visually inspect the code (not every test requires modifying/running the code).

Debug this code (In class)

```
import java.util.Scanner;

public class CubeVolume {
   public static void main (String [] args) {
      Scanner scnr = new Scanner(System.in);
      int sideLength = 0;
      int cubeVolume = 0;

      System.out.println("Enter cube's side length: ");
      sideLength = scnr.nextInt();

      cubeVolume = sideLength * sideLength * sideLength;

      System.out.println("Cube's volume is: " + cubeVolume);

      return;
   }
}
```

```
Enter cube's side length: 10000
Cube's volume is: -727379968
```

Output Formatting

```
System.out.printf(" You know %d people.\n", totalPpl);
```

Format String has two parts:

- format string
  - format of the text that will be printed
- Format specifiers
  - define the type of values being printed in place of the format specifier

Output Formatting

Format specifier	Data Type(s)	Notes
%c	char	Prints a single Unicode character
%d	int, long, short	Prints a decimal integer value.
%o	int, long, short	Prints an octal integer value.
%h	int, char, long, short	Prints a hexadecimal integer value.
%f	float, double	Prints a floating-point value.
%e	float, double	Prints a floating-point value in scientific notation.
%s	String	Prints the characters in a String variable or literal.
%%		Prints the '%' character.
%n		Prints the platform-specific new-line character.

Formatting Specifiers

- The format specifiers within the format string of printf() can include format sub-specifiers.
- The formatting sub-specifiers are included between the % and format specifier characters.

```
%(flags)(width)(.precision)specifier
```

Example

```
System.out.printf("%.3f", myFloat)
```

Formatting Specifiers

```
%(flags)(width)(.precision)specifier
```

width: Specifies the minimum number of characters to be printed

```
double myFLoat = 12.34124
System.out.printf("Value: %7.2f", myFloat);
Value:   12.34
```

Formatting Specifiers

```
%(flags)(width)(.precision)specifier
```

precision: Specifies the number of digits to print following the decimal point.

```
double myFLoat = 12.34

System.out.printf("%.4f", myFloat);
12.3400

System.out.printf("%3.4e", myFloat);
1.2340e+01
```