Lecture 2 - Java Fundamentals

CS260 – Android App Development

Paul Cao

Review

- Android System overview (hello world project)
- Primitive data types in Java
  - int
  - double/float
  - char
  - boolean
- Console output (System.out.print)

Plan for today

- Operators
- final for constants
- Scanner for reading in keyboard input
- String class

Arithmetic Operators

- Java has five (5) arithmetic operators.

Operator	Meaning	Type	Example
+	Addition	Binary	`total = cost + tax;`
-	Subtraction	Binary	`cost = total - tax;`
*	Multiplication	Binary	`tax = cost * rate;`
/	Division	Binary	`salePrice = original / 2;`
%	Modulus	Binary	`remainder = value % 5;`

Integer Division

- Division can be tricky.

  In a Java program, what is the value of 1/2?

- Integer division will truncate any decimal remainder.

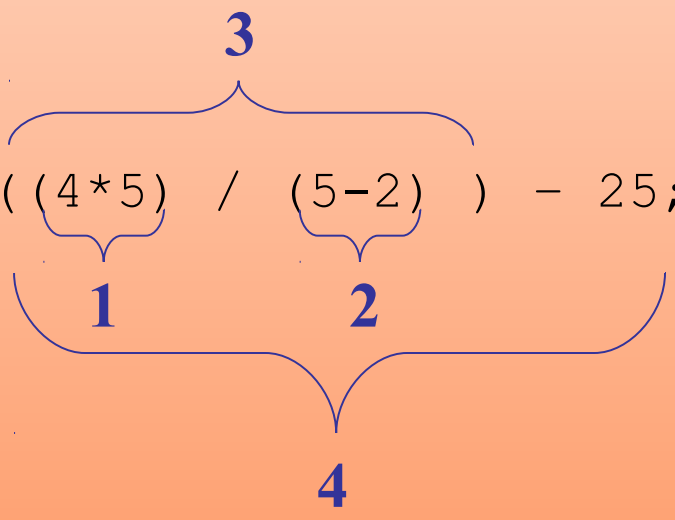
Operator Precedence

- Mathematical expressions can be very complex.
- There is a set order in which arithmetic operations will be carried out.

	Operator	Associativity	Example	Result
Higher Priority	– (unary negation)	Right to left	`x = -4 + 3;`	-1
	`* / %`	Left to right	`x = -4 + 4 % 3 * 13 + 2;`	11
Lower Priority	`+ -`	Left to right	`x = 6 + 3 - 4 + 6 * 3;`	23

Grouping with Parenthesis

- When parenthesis are used in an expression, the inner most parenthesis are processed first.
- If two sets of parenthesis are at the same level, they are processed left to right.




```
x = ((4*5) / (5-2) ) - 25;   // result = -19
```

Combined Assignment Operators

- Java has some combined assignment operators.
- These operators allow the programmer to perform an arithmetic operation and assignment with a single operator.
- In essence, these operators' job is to change the variable's value by ….
- Although not required, these operators are popular since they shorten simple equations.

Combined Assignment Operators

Operator	Example	Equivalent	Value of variable after operation
+=	`x += 5;`	`x = x + 5;`	The old value of **x** plus 5.
-=	`y -= 2;`	`y = y - 2;`	The old value of **y** minus 2
***=**	`z *= 10;`	`z = z * 10;`	The old value of **z** times 10
/=	`a /= b;`	`a = a / b;`	The old value of **a** divided by **b**.
%=	`c %= 3;`	`c = c % 3;`	The remainder of the division of the old value of **c** divided by 3.

Type conversion

- We sometimes massage different types of variable and data together. → may create problems called type mismatch.
  - ```
    int x=121;
    double y=300.25;
    y=x+y;//type mismatch!
    ```
    ```

Type conversion

- Solution: cast into the same type!
  - ```
    int x=121;
    double y=300.25;
    y=(double)x+y;
    //now it is double add with double
    ```
- Are the following statements the same?

  ```
  y=(double)x+y;
  y=(double)(x+y);
  ```
  -
  ```

Creating Constants

- Constants allow the programmer to use a name rather than a value throughout the program.
- Constants also give a singular point for changing those values when needed.
- Constants are declared using the keyword `final`.
- Constants need not be initialized when declared; however, they must be initialized before they are used or a compiler error will be generated.

Creating Constants

- Once initialized with a value, constants cannot be changed programmatically.
- By convention, constants are all upper case and words are separated by the underscore character.

```
final int CAL_SALES_TAX = 0.725;
```

- Make sure you use your constants!

The `Scanner` Class

- To read input from the keyboard we can use the `Scanner` class.
- The `Scanner` class is defined in `java.util`, so we will use the following statement at the top of our programs:

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

The `Scanner` Class

- Scanner objects work with `System.in`
- To create a `Scanner` object:

```
Scanner keyboard = new Scanner (System.in);
```

- `Scanner` class methods can be found by using the ctrl+space in Eclipse.

Scanner object methods

Methods=	Examples
nextByte	byte x; Scanner keyboard=new Scanner(System.in); System.out.print("Enter a byte value: "); x=keyboard.nextByte();
nextInt	Very similar as above
nextDouble	
nextFloat	
nextLong	
nextShort	
next	String s; Scanner keyboard=new Scanner(System.in); System.out.print("Enter your first name"); s=keyboard.next();

Some details

- Be careful with the nextLine method.
- How about read in the full name?
  - Read in first name, then last name.
- How to read in a character instead of a word?
  - Use charAt(0) method of string.

Exercise

- Can you write a program that asks the user the length and width of a room (decimals) and the name of the kid to whom the room belongs to, then output the area of the room in the following way (suppose the name of the kid is xxxxxx)

- Hey xxxxxx, your room is …... square feet! Enjoy!

Primitive vs. Reference Variables

- Primitive variables actually contain the value that they have been assigned.

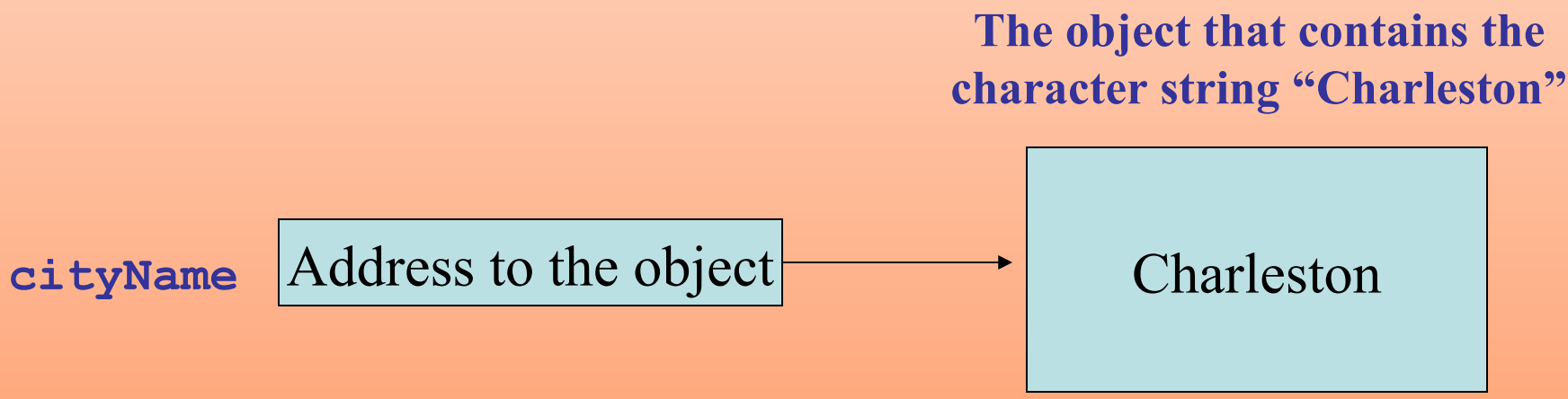
```
number = 25;
```

- The value 25 will be stored in the memory location associated with the variable `number`.
- Objects are not stored in variables, however. Objects are *referenced* by variables.

Primitive vs. Reference Variables

- When a variable references an object, it contains the memory address of the object's location.
- Then it is said that the variable *references* the object.

```
String cityName = "Charleston";
```



Think about the difference between your name and yourself

`String` Objects

- A variable can be assigned a String literal.

```
String value = "Hello";
```

- `Strings` are the only objects that can be created in this way.
- A variable can be created using the *new* keyword.

```
String value = new String("Hello");
```

- This is the method that all other objects must use when they are created.

See example: StringDemo.java

The `String` Methods

- Since `String` is a class, objects that are instances of it have methods.
- One of those methods is the `length` method.

```
stringSize = name.length();
```

- This statement runs the `length` method on the object pointed to by the `name` variable.

See example: StringLength.java

`String` Methods

- The `String` class contains many methods that help with the manipulation of `String` objects.
- `String` objects are *immutable*, meaning that they cannot be changed.
- Many of the methods of a `String` object can create new versions of the object.
- Pay attention to `charAt`. Index begins at 0.

See example: StringMethods.java

Scope

- *Scope* refers to the part of a program that has access to a variable's contents.
- Variables declared inside a method (like the main method) are called *local variables*.
- Local variables' scope begins at the declaration of the variable and ends at the end of the method in which it was declared.

Commenting Code

- Java provides three methods for commenting code.

Comment Style	Description
`//`	Single line comment. Anything after the // on the line will be ignored by the compiler.
`/* ... */`	Block comment. Everything beginning with /* and ending with the first */ will be ignored by the compiler. This comment type cannot be nested.
`/** ... */`	Javadoc comment. This is a special version of the previous block comment that allows comments to be documented by the javadoc utility program. Everything beginning with the /** and ending with the first */ will be ignored by the compiler. This comment type cannot be nested.

Commenting Code

- Javadoc comments can be built into HTML documentation.
- See example: Comment.java
- To create the documentation:
  - Run the `javadoc` program with the source file as an argument
  - Ex: **`javadoc Comment.java`**
- The `javadoc` program will create `index.html` and several other documentation files in the same directory as the input file.

- Also Ctrl+Shift+F is useful to format your code automatically

Commenting Code

- Example index.html:

All Classes
Comment3

Package **Class** Tree Deprecated Index Help

PREV CLASS NEXT CLASS — FRAMES NO FRAMES
SUMMARY: NESTED | FIELD | CONSTR | METHOD — DETAIL: FIELD | CONSTR | METHOD

Class Comment3

```
java.lang.Object
  └ Comment3
```

public class **Comment3**
extends java.lang.Object

This class contains a program that calculates company payroll/

Constructor Summary

Comment3()

Method Summary

static void	**main**(java.lang.String[] args) The main() method is the program's starting point.

Exercise

Write a program that asks user for their first name. Then asks user to enter a position in their first name. Your program should output the letter at that position. You can assume the user will give you a correct (not too big) position.

```
What is your first name: paul
Which position do you want to see: 2
At position 2, the letter is u
```

Dialog Boxes

- A *dialog box* is a small graphical window that displays a message to the user or requests input.
- A variety of dialog boxes can be displayed using the `JOptionPane` class.
- Two of the dialog boxes are:
  - Message Dialog - a dialog box that displays a message.
  - Input Dialog - a dialog box that prompts the user for input.

The `JOptionPane` Class

- The `JOptionPane` class is not automatically available to your Java programs.
- The following statement must be before the program's class header:

```
import javax.swing.JOptionPane;
```

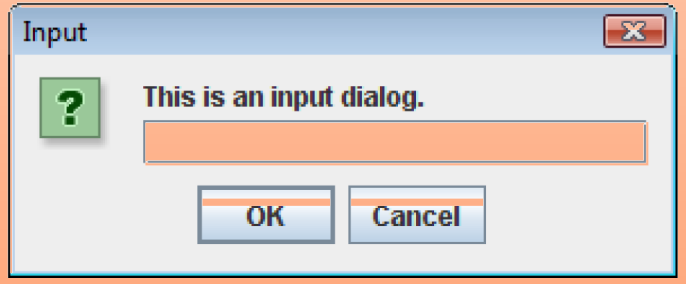
- This statement tells the compiler where to find the `JOptionPane` class.

The `JOptionPane` Class

The `JOptionPane` class provides methods to display each type of dialog box.

Message dialog

Input dialog

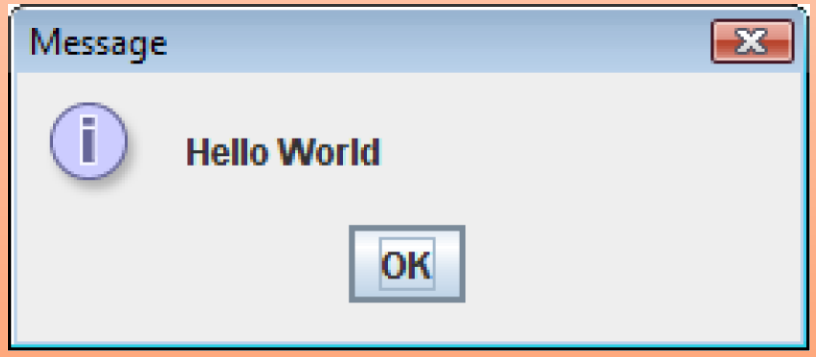


Message Dialogs

- `JOptionPane.showMessageDialog` method is used to display a message dialog.

```
JOptionPane.showMessageDialog(null, "Hello World");
```

- The first argument will be discussed in Chapter 7.
- The second argument is the message that is to be displayed.



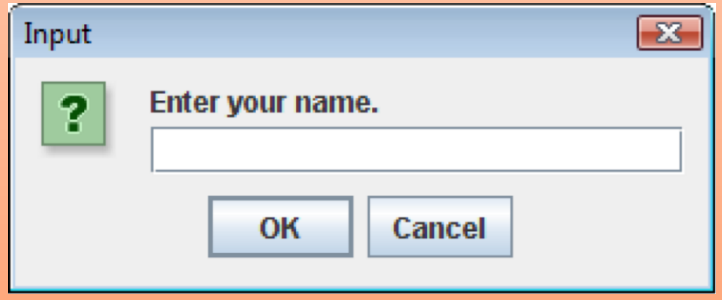
Input Dialogs

- An input dialog is a quick and simple way to ask the user to enter data.
- The dialog displays a text field, an Ok button and a Cancel button.
- If Ok is pressed, the dialog returns the user's input.
- If Cancel is pressed, the dialog returns null.

Input Dialogs

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

- The argument passed to the method is the message to display.
- If the user clicks on the OK button, `name` references the string entered by the user.
- If the user clicks on the Cancel button, `name` references `null`.



The `System.exit` Method

- A program that uses `JOptionPane` does not automatically stop executing when the end of the main method is reached.
- Java generates a *thread*, which is a process running in the computer, when a `JOptionPane` is created.
- If the `System.exit` method is not called, this thread continues to execute.

The `System.exit` Method

- The `System.exit` method requires an integer argument.

```
System.exit(0);
```

- This argument is an *exit code* that is passed back to the operating system.
- This code is usually ignored, however, it can be used outside the program:
  - to indicate whether the program ended successfully or as the result of a failure.
  - The value 0 traditionally indicates that the program ended successfully.
- Example: Greeting.java

Converting a String to a Number

- The `JOptionPane`'s `showInputDialog` method always returns the user's input as a `String`
- A `String` containing a number, such as "127.89, can be converted to a numeric data type.

The Parse Methods

- Each of the numeric wrapper classes, (covered in Chapter 10) has a method that converts a string to a number.
  - The `Integer` class has a method that converts a string to an `int`,
  - The `Double` class has a method that converts a string to a `double`, and
  - etc.
- These methods are known as *parse methods* because their names begin with the word "parse."

The Parse Methods

```
// Store 1 in bVar.
byte bVar = Byte.parseByte("1");

// Store 2599 in iVar.
int iVar = Integer.parseInt("2599");

// Store 10 in sVar.
short sVar = Short.parseShort("10");

// Store 15908 in lVar.
long lVar = Long.parseLong("15908");

// Store 12.3 in fVar.
float fVar = Float.parseFloat("12.3");

// Store 7945.6 in dVar.
double dVar = Double.parseDouble("7945.6");
```

Reading an Integer with an Input Dialog

```
int number;
String str;
str = JOptionPane.showInputDialog(
   "Enter a number.");
number = Integer.parseInt(str);
```

Reading a `double` with an Input Dialog

```
double price;
String str;
str = JOptionPane.showInputDialog(
    "Enter the retail price.");
price = Double.parseDouble(str);
```

See example: LetterGrade.java

The `DecimalFormat` Class

- When printing out `double` and `float` values, the full fractional value will be printed.
- The `DecimalFormat` class can be used to format these values.
- In order to use the `DecimalFormat` class, the following `import` statement must be used at the top of the program:

```
import java.text.DecimalFormat;
```

How to use DecimalFormat class

- Special format
  - # and 0 both means digit
    - # means if there is a digit display it. Or else don't do anything.
    - 0 means if there is a digit display it. Or else, add a 0 at that place.
  - . means decimal dot
- What does the following pattern do?
  - #0.00
  - ##.00
  - 00.00
- Example: Formatter.java

Flow control

- The flow control structures in Java are exactly the same as in C++ or other majors languages.
  - If statements
  - While loop and for loop
  - Switch statement

Exercise

Write a for loop to calculate the total of the following series of numbers. Show the result with 6 places after the dot.

$$\frac{1}{30} + \frac{2}{29} + \frac{3}{28} + \ldots + \frac{30}{1}$$

Exercise

Write a program that calculates the amount a person would earn over a period of time if his or her salary is one penny the first day, two pennies the second day, and continues to double each day. The program should display a table showing the salary for each day, and then show the total pay at the end of the period. The output should be displayed in a dollar amount, not the number of pennies.

Input validation: Do not accept a number less than 1 for the number of days worked.