# LEARN PROGRAMMING IN JAVA™

Learning OO Java™ as a Beginner – Control Flow

# COURSE AGENDA

- What is an Object
- The Basics
- Operators
- Control Flow
- Implementation Hiding
- Reuse
- Interfaces
- Collections
- Functional Programming
- Stream
- Exceptions
- Generics
- Arrays

# INSTRUCTOR – HUGO SCAVINO

- 30 Years of IT Experience
- Using Java since the beta
- Taught Java and OOP in USA, UK, and France to Fortune 500
- Senior Software Architect with
  - DTCC
  - Penske
  - HSBC
  - Government Agencies

https://www.linkedin.com/in/hugoscavino/

# TOPIC DESCRIPTION

Control Flow: The order in which individual statements, instructions, or function calls are executed or evaluated. It determines how a program makes decisions and performs operations based on certain conditions. We will investigate conditional statements, loops, and branching statements.

- Conditional Statements
- Loops (repetition)
- Branching (boo!)

# REQUIRED SOFTWARE

- JDK™
  - JDK 8 or Newer (Open Source, Amazon, or Oracle)
  - The beginner course focuses on core language constructs; feel free to use 11, 17, 21, etc.

- IDE
  - IntelliJ Community Edition  (Free) or Enterprise (Paid)
  - While you can use any IDE with Java, the course and labs are explicitly made with IntelliJ in mind

# RECOMMENDED TEXTS

- Java 5 Book
  - Thinking in Java – 4th Edition - Free – PDF - GitHub
  - Thinking in Java – 4th Edition – Hard Cover - Amazon

- Recommended Java 8 Book
  - Bruce Eckel on Java 8
    - Contains references to newer Java syntax
- #1 Best Seller in Beginner's Guide to Java Programming
  - Head First Java: A Brain-Friendly Guide 3rd Edition

# ONLINE VIDEO TUTORIAL

- Learning Programming In Java
  - YouTube Tutorials
- Lab Exercises
  - http://www.learnprogramminginjava.com/

# IF ELSE

- The most basic and likely popular construct
- If the condition resolves to `true` then follow the first flow
- Else follow the other flow
- Possible to chain the if-else
  - Chaining is frowned upon
  - Consider a `switch` statement in these conditions
- There is an unnecessary war on the style of if-else
  - Ignore the debate and use my way ; I am right

# IF ELSE - EXAMPLE

```java
int today = Calendar.getInstance().get(Calendar.DAY_OF_WEEK);
boolean isMonday = (today == Calendar.MONDAY);
boolean isTheWeekend = (today == Calendar.SATURDAY) || (today == Calendar.SUNDAY);

if (isMonday) {
    System.out.println("You have a case of the Mondaze");
} else if (isTheWeekend) {
    System.out.println("Happy Weekend");
} else {
    System.out.println("You in a weekday");
}
```

# PRECEDENCE EXAMPLE

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

    int x = 100;
    int y = 200;
    int z = 300;

    // What would expect from adding the three values?
    // 600?
    System.out.println(x + y + z);

    // Now What?
    // Is it 602 or is the result or 502?
    System.out.println(x + y + 4/2 + z);

    // Now What?
    // Is it 602 or is the result or 502?
    System.out.println(x + (y + 4)/2 + z);
```

Don't waste your time memorizing more than the basic rules; Multiplication and Division take precedence over addition and subtraction.

Instead, use parenthesis every time!

Don't let the compiler decide the logic for you. Be explicit.

# ITERATION WHILE

- While a `boolean` condition is '`true`', proceed
- The code will run forever if the condition does not change.
- The condition is guaranteed to be evaluated at least once
- However, the code may never run if the condition is never met .

# WHILE EXAMPLE

```java
// value from 0 through 59
int currentMinute = LocalDateTime.now().getMinute();

final int initialMinute = currentMinute;
// Will loop for at most 59 seconds
while (initialMinute == currentMinute) {
    // update the currentMinute, the while at the end of the loop re-evaluates
    // condition
    currentMinute = LocalDateTime.now().getMinute();
    // sleep for a second so we do not clog the console
    sleep( millis: 1000);
    int currentSecond = LocalDateTime.now().getSecond();
    System.out.println(currentSecond);
}
```

# ITERATION DO-WHILE

- Perform the statements at least ONCE
- While the `boolean` condition is '`true`', loop again
- The code will run forever if the condition does not change.
- The condition is guaranteed to be evaluated at least once
- The least popular iteration, not used often

```
do
  statement
while(Boolean-expression);
```

# ITERATION FOR

- Used extensively
- The construct will
  - Initialize
  - Perform a Boolean Expression
  - Step ( or increment, decrement)
  - Loop
- Any of the above construct elements can be empty
  - This is rare

# FOR LOOP EXAMPLE

Initialize the `dayOfWeekInt` to the integer value for Sunday
Until the value is equal to SATURDAY, add one to `dayOfWeekInt`

```java
// Start at Sunday and loop through the days of the week
// convert the integer to a day of the week
for (int dayOfWeekInt = Calendar.SUNDAY; dayOfWeekInt <= Calendar.SATURDAY; dayOfWeekInt++){
    String dayOfWeekStr = toDayName(dayOfWeekInt);
    System.out.println("For Each Day of Week [Sunday first day of the week] : " + dayOfWeekStr);
}
```

# ITERATION FOR-EACH

- Used even more extensively with Arrays
- The more efficient method
- The counter is not required

```
int[] DaysOfWeekArray = {Calendar.SUNDAY, Calendar.MONDAY, Calendar.TUESDAY, Calendar.WEDNESDAY,
        Calendar.THURSDAY, Calendar.FRIDAY, Calendar.SATURDAY};

for (int dayOfWeekInt : DaysOfWeekArray) {
    String dayOfWeekStr = toDayName(dayOfWeekInt);
    System.out.println("For Each Day of Week [Sunday first day of the week] : " + dayOfWeekStr);
}
```

** Assume there is a utility function to convert an `int` to a String representing the day of week name*

# BREAK AND CONTINUE

Use with caution. The Break keyword is a code-smell and an indicator that you may want to refactor your code so as not to need to break.

Further, the continue is rarely used, and another concern if the keyword is used.

```
// Questionable use of break
for (int dayOfWeekInt : DaysOfWeekArray) {
    String dayOfWeekStr = toDayName(dayOfWeekInt);

    if (dayOfWeekStr.equals("Wednesday")) {
        break;
    }
    System.out.println("For Each Day of Week [Sunday
}
```

# SWITCH STATEMENT

- Personal favorite
- There is a new enhanced switch
- Obviates the need for complicated if-then-else logic
- Can execute several lines of code per condition
- Proper use of the keyword `break`
- Efficient and easy -to- read
- The switch can break on a string
  - Previous versions of Java only allowed `int`

# SWITCH EXAMPLE

```java
static String toDayName(int dayOfWeek) {
    return switch (dayOfWeek) {
        case Calendar.SUNDAY -> "Sunday";
        case Calendar.MONDAY -> "Monday";
        case Calendar.TUESDAY -> "Tuesday";
        case Calendar.WEDNESDAY -> "Wednesday";
        case Calendar.THURSDAY -> "Thursday";
        case Calendar.FRIDAY -> "Friday";
        case Calendar.SATURDAY -> "Saturday";
        default -> "Unknown";
    };
}
```

- Switch can use an `int`, `char`, or a `String` as the pivot point.
- Only one condition will be executed.
- Once a condition is met, the other possibilities are not executed
- If no condition is met, the `` `default` `` will be executed.
  - The `default` case does not need to change the state.

```java
/**
 * Legacy Switch
 * @param oneLetter one letter as a String
 * @return Boolean is a vowel or not
 */
static boolean isVowel(String oneLetter) {
    boolean isVowel;
    switch (oneLetter) {
        case "a":
        case "e":
        case "i":
        case "o":
        case "u":
            isVowel = true;
            break;  // caution forgetting this line
                    // is a wicked bug
        case "y":
            System.out.println("sometimes");
            isVowel = true;
            break;
        default:
            isVowel = false;
            break;
    };
```

# LEGACY SWITCH EXAMPLE

- Switch can use an `int`, `char`, or a `String` as the pivot point.
- Multiple cases can be stacked before the one `break`.
- The `default` is optional
- We could have initialized the `isVowel` to `false`.

- What would happen if we forgot the break after the “u” case?

# TOPIC SUMMARY

Control flow: The order in which individual statements, instructions, or function calls are executed or evaluated.

- Conditional Statements
  - If-else
  - returns
- Loops (repetition)
  - While
    - Do While
  - For
  - For-Each
- Branching (boo!)
  - Switch
  - Legacy Switch

# ONLINE VIDEOS

- Learning Programming In Java
  - YouTube Tutorials
- Lab Exercises
  - http://www.learnprogramminginjava.com/