Java Basics - Expressions

- **Literals**
  - A literal is a constant value also called a *self-defining term*
  - Possibilities:
    - **Object:** `null`, the only object literal available
    - **Boolean:** `true` or `false`
    - **Integer:** e.g., `127`, `−13`, `42`, or `0` create 32-bit integers
      - For 64-bit `long` append `L` or `l`, e.g., `17L`
    - **Floating Point:** `3.14592` or `0.0` or `2.1E16` for 64-bit doubles
      - For 32-bit `float` append `F` or `f`, e.g., `2.56F` or `0.5e−12f`
    - **Character:** e.g., `'A', 'Z', 'w', '$', '%' for 16 bit Unicode
      - control: `'\n', '\b', '\f', '\t', '\r'`
      - escape: `'\', '\\', '"'`
    - **Strings:** e.g., "*How are things?*" or "" (null string) or `null`
      - Use mostly same control and escape characters as `char`
Java Basics - Expressions

Operators
- Arithmetic
  - +, -, *, /, % (remainder or mod)
- Increment/Decrement
  - e.g., k++, k-- , ++k, --k
- Logical (results in boolean value)
  - <, <=, ==, !=, >=, >
  - Used only for numbers except == and !=
  - For boolean only: !, &&, ||
- String Concatenation
  - "I’m " + 19 + " years old and live in " + city
- Assignment
  - variable = expression
  - variable op= expression
  - (shorthand for: variable = variable op expression)
Java Basics - Expressions

- **Operator Precedence**
  - Determines order of operation
  - See table in text
  - For arithmetic, matches grammar school learning
    - Multiplication and division before addition and subtraction
    - What is the value of $4.0 + 5.0 / 9.0 * 27.0$?
    - (What is the value for the integer version?)
  - Parentheses override precedence rules (and don’t do harm when not needed)
  - For equal precedence (e.g., * and /) work strictly left to right except for assignment and prefix operations which work right to left
  - Precedence rules same as for C and C++
Java Basics - Expressions

- **Casting**
  - Allows us to change the type of the value of an expression
  - (Type change must be reasonable and supported.)
  - Simple example:
    ```java
double x = 5.5, y = 2.9999;
int k = (int) x;
int m = (int) y;
double z = (double) k;  // what is in x, y, z, k, m ?
```
- **Implicit Casting**
  - When an `int` expression is assigned to a `double`, casting is automatic (no information is lost).
    - (Double cast at end of previous example not needed)
  - When `double` is on one side of an operator and `int` at other, `int` is automatically cast to a `double` before op is used.
    ```java
    5 / 9 * (68 - 32)  vs.  5.0 / 9 * (68 - 32)
    ```
Java Basics - Expressions

- Autoboxing/Unboxing
  - Since Java 5.0, there is automatic casting between primitive types and their related Object types (also called wrapper classes).
  - Simple examples:
    ```java
    Double d = 2.9;
    used to require:
    Double d = new Double(2.9);
    and
    double x = d;
    used to require
    double x = d.doubleValue();
    ```
Java Basics – Control of Flow

❖ If Statement

- if (boolean \_exp) {
  what\_to\_do\_if\_true
}
- if (boolean \_exp) {
  what\_to\_do\_if\_true
}
  else {
    what\_to\_do\_if\_false
  }
- if (1\(^{st}\) boolean \_exp) {
  what\_to\_do\_if\_1\(^{st}\) true
}
  else if (2\(^{nd}\) boolean \_exp){
    what\_to\_do\_if\_2\(^{nd}\) true
  }
  else {
    what\_to\_do\_if\_all\_false
  }
Java Basics – Control Flow

- **Switch Statement**

```java
switch (int_type_exp) {
    case CONST1:
        action_for_CONST1;
        break;
    case CONST1:
        action_for_CONST1;
        break;
    case CONST2:
        action_for_CONST2;
        break;
    case CONST3:
        action_for_CONST3;
        break;
    ...
    default:
        action_for_no_match;
        break;
}
```
Switch Statement Example

```java
switch (stars) {
    case 4:
        message = "truly exceptional";
        break;
    case 3:
        message = "quite good";
        break;
    case 2:
        message = "fair";
        break;
    case 1:
    case 0:
        message = "forget it";
        break;
    default:
        message = "no info found";
        break;
}
```
Java Basics – Loops

- **While Loops**
  - Syntax
    ```java
    initialize
    while (boolean_exp) {
        work_to_be_done
        update
    }
    ```
  - Example
    ```java
    int counter = 10;
    while (counter > 0) {
        System.out.println(counter);
        counter--;
    }
    System.out.println("Blast Off");
    ```
  - What is the output?
  - What if we exchange order of two statements in loop?
Java Basics – Loops

- For Loops
  - Syntax
    ```java
    for (initialization; boolean_exp; update) {
        work_to_be_done
    }
    ```
  - Example
    ```java
    for (int counter = 10; counter > 0; counter--) {
        System.out.println(counter);
    }
    System.out.println("Blast Off!");
    ```
  - What is the output?
  - When is update performed?
  - What is value of `counter` after loop?
Java Basics – Loops

- Do-While Loops
  - Syntax
    ```java
    initialize
    do
    {
      work_to_be_done
      update
    } while (boolean_exp);
    ```
    NOTE REQUIRED SEMICOLON!!!

- Example
  ```java
  int counter = 10;
  do {
    System.out.println(counter);
    counter--;
  } while (counter > 0);
  System.out.println("Blast Off!");
  ```
Java Basics – Loops

Which Kind of Loop Do I Use?

- **While Loop**
  - Don’t know how often it’s going be
  - Update can be anywhere in the loop body

- **For Loop**
  - Know how often in advance
  - All information controlling loop together, in front

- **Do-While Loop**
  - Least popular
  - Often used with data input

What is the minimum number of times each of these loop?

- while?
- for?
- do-while?
Java Basics – Control Flow

- **Returning from a Method**
  - Executing a `return` statements means you exit from the method. Subsequent statements are ignored!
  - **void** Methods
    - Implicit `return` at end of body
      - Can make it explicit
    - Can have other `return` statements as logic dictates
  - Functions (non-void Methods)
    - Require `return` as last statement (with argument of correct type)
    - Can have other `return` statements as logic dictates
Java Basics – Control Flow

❖ Break Statement
  □ Use to exit from loop or switch
    ○ One level only!
    ○ With nested loops, only leave loop immediately surrounding break

❖ Continue Statement
  □ Use to go to the end of a loop, ignoring remaining statements
    ○ Loop continues with next iteration (if needed)
    ○ One level only!
    ○ With nested loops, only got to end of loop immediately surrounding continue