Problem 3

- Given a real number \( c \) and some error tolerance \( \epsilon \), estimate \( t \), the square root of \( c \).

Problem 2

- Given \( n \), calculate \( 2^n \).
  - What if you wanted to print all from \( 2^n \) to \( 2^{n+1} \)?
  - What if you wanted to return the value?

Problem 1

- Consider a problem in which \( m \) pancakes need to be cooked in the shortest time possible using only one pan that holds \( n \) pancakes at once.
  - Each pancake must be cooked for five minutes on each side.
  - How many minutes are needed to cook the pancakes?

Java Basics - Expressions

- Literals
  - A literal is a constant value also called a self-defining term.
  - Possibilities:
    - Object: \( \text{null} \), the only object literal available
    - Boolean: \( \text{true} \) or \( \text{false} \)
    - Integer: e.g., \( 127, -13, 42 \), or \( 0 \) create 32-bit integers
      - For 64-bit \( \text{long} \) append \( \text{L} \) or \( \text{L} \), e.g., \( 17\text{L} \)
    - Floating Point: \( 3.14592 \) or \( 0.0 \) or \( 2.1e16 \) for 64-bit doubles
      - For 32-bit \( \text{float} \) append \( \text{F} \) or \( \text{f} \), e.g., \( 2.56\text{F} \) or \( 0.5e-12\text{f} \)
    - Character: e.g., \( \text{A}, \text{Z}, \text{W}, \text{$}, \% \) for 16 bit Unicode
      - control: \( \text{\n}, \text{\b}, \text{\f}, \text{\t}, \text{\r} \)
      - escape: \( \text{\'} , \text{\"} , \text{\'}\text{'}\)
    - Strings: e.g., "How are things?" or "" (null string) or \( \text{null} \)
      - Use mostly same control and escape characters as \text{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 !=
  - **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 of 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:
    - 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.
      - 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:
    - Double d = 2.9;
      - used to require:
        - Double d = new Double(2.9);
        - 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 (1st Boolean Exp) {
  what_to_do_if_1st_true
} else if (2nd Boolean Exp) {
  what_to_do_if_2nd_true
} else {
  what_to_do_if_all_false
}

Java Basics – Loops

**While Loops**

- Syntax
  initialize
  while (boolean_exp) {
    work_to_be_done
    update
  }
- Example
  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
  for (initialization; boolean_exp; update) {
    work_to_be_done
  }
- Example
  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 – Control Flow

**Switch Statement Example**

- 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;
}

CompSci 100E 2.9

CompSci 100E 2.10

CompSci 100E 2.11

CompSci 100E 2.12
Java Basics – Loops

- **Do-While Loops**
  - Syntax
    - `initialize`
    - `do`
      - `work_to_be_done`
      - `update`
    - `while` (boolean_exp);
    - `return` as last statement (with argument of correct type)
  - 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? o **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
      - **One level only!**
        - With nested loops, only got to end of loop immediately surrounding `continue`