CS 200 - Programming I: Primitives and Expressions

Marc Renault

Department of Computer Sciences
University of Wisconsin – Madison

Fall 2019
TopHat Sec 3 (1:20 PM) Join Code: 682357
TopHat Sec 4 (3:30 PM) Join Code: 296444
Primitives
**Two-state machine**

**Binary**

**Data Units**

**Bit**  The smallest unit of information. 0 or 1, false or true, off or on, low or high, ...
Two-state machine

Binary

Data Units

**Bit**  The smallest unit of information. 0 or 1, false or true, off or on, low or high, ...

**Byte**  8-bits.
Two-state machine

Binary

Data Units

**Bit**  The smallest unit of information. 0 or 1, false or true, off or on, low or high, ...

**Nibble**  4-bits.

**Byte**  8-bits.
**Two-state machine**

**Binary**

### Data Units

- **Bit**  The smallest unit of information. 0 or 1, false or true, off or on, low or high, ...
- **Nibble**  4-bits.
- **Byte**  8-bits.
- **Word**  A group of bytes, depends on the systems: 16-bit systems is 2 byte words, 32-bit is 4 byte words, ...
### Two-state machine

#### Binary

**Data Units**

<table>
<thead>
<tr>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bit</strong></td>
<td>The smallest unit of information. 0 or 1, false or true, off or on, low or high, ...</td>
</tr>
<tr>
<td><strong>Nibble</strong></td>
<td>4-bits.</td>
</tr>
<tr>
<td><strong>Byte</strong></td>
<td>8-bits.</td>
</tr>
<tr>
<td><strong>Word</strong></td>
<td>A group of bytes, depends on the systems: 16-bit systems is 2 byte words, 32-bit is 4 byte words, ...</td>
</tr>
<tr>
<td><strong>KB vs kB</strong></td>
<td>1024 bytes vs 1000 bytes.</td>
</tr>
</tbody>
</table>
Two-state machine

Binary

Data Units

**Bit**  The smallest unit of information. 0 or 1, false or true, off or on, low or high, ...

**Nibble**  4-bits.

**Byte**  8-bits.

**Word**  A group of bytes, depends on the systems: 16-bit systems is 2 byte words, 32-bit is is 4 byte words, ...

**KB vs kB**  1024 bytes vs 1000 bytes.

**MiB vs MB**  $1024^2$ bytes vs $1000^2$ bytes.
Two-state machine

Binary

Data Units

**Bit**  The smallest unit of information. 0 or 1, false or true, off or on, low or high, ...

**Nibble**  4-bits.

**Byte**  8-bits.

**Word**  A group of bytes, depends on the systems: 16-bit systems is 2 byte words, 32-bit is 4 byte words, ...

**KB vs kB**  1024 bytes vs 1000 bytes.

**MiB vs MB**  $1024^2$ bytes vs $1000^2$ bytes.

**GiB vs GB**  $1024^3$ bytes vs $1000^3$ bytes.
**Variable Declaration**

```
type variableName = expression;
```

**Type**

- Java primitives: `int`, `byte`, `char`, `short`, `long`, `float` and `double`. 
Variable Declaration

:type variableName = expression;

Type

- Java primitives: `int`, `byte`, `char`, `short`, `long`, `float` and `double`.
- Restricts the variable to:
Variable Declaration

```
type variableName = expression;
```

**Type**

- Java primitives: `int`, `byte`, `char`, `short`, `long`, `float` and `double`.
- Restricts the variable to:
  - A set of values.
**Variable Declaration**

```java
type variableName = expression;
```

**Type**

- **Java primitives:** `int`, `byte`, `char`, `short`, `long`, `float` and `double`.
- Restricts the variable to:
  - A set of values.
  - A set of operations.
Variable Declaration

\[ \text{type variableName = expression;} \]

Type

- Java primitives: int, byte, char, short, long, float and double.
- Restricts the variable to:
  - A set of values.
  - A set of operations.
- Java is a strongly typed language.
**Variable Declaration**

```java
type variableName = expression;
```

**Type**

- **Java primitives:**
  - `int`, `byte`, `char`, `short`, `long`, `float` and `double`.
- Restricts the variable to:
  - A set of values.
  - A set of operations.
- Java is a *strongly typed language*.
- All variables are declared with a specific type and remain that type.
A rose by any other name...

Naming Rules

- Starts with a letter, $, or underscore (_).
A rose by any other name...

Naming Rules

- Starts with a letter, $, or underscore (_).
- Subsequent characters can be letters, $, (_) or digits.
A rose by any other name...

Naming Rules

- Starts with a letter, $, or underscore (_).
- Subsequent characters can be letters, $, (_) or digits.
- Cannot be a *keyword* (Ex: int, case, while)
A rose by any other name...

Naming Rules

- Starts with a letter, $, or underscore (_).
- Subsequent characters can be letters, $, (_) or digits.
- Cannot be a keyword (Ex: int, case, while)
- Can be any length.
A rose by any other name...

Naming Rules

- Starts with a letter, $, or underscore (_).
- Subsequent characters can be letters, $, _ or digits.
- Cannot be a keyword (Ex: int, case, while)
- Can be any length.
- Case-sensitive: foobar ≠ fooBar
A rose by any other name...

Naming Rules

- Starts with a letter, $, or underscore (_).
- Subsequent characters can be letters, $, (_) or digits.
- Cannot be a keyword (Ex: int, case, while)
- Can be any length.
- Case-sensitive: foobar ≠ fooBar

CS 200 Style and Good Practice

- Use lowerCamelCase: thisIsLowerCamelCase
- Don’t use $ or _
- Length: not too long, but clearly identify the role.
A rose by any other name...

Naming Rules

- Starts with a letter, $, or underscore (_).
- Subsequent characters can be letters, $, (_) or digits.
- Cannot be a *keyword* (Ex: `int`, `case`, `while`)
- Can be any length.
- Case-sensitive: `foobar` \(\neq\) `fooBar`

CS 200 Style and Good Practice

- Style guide:
  
A rose by any other name...

Naming Rules

- Starts with a letter, $, or underscore (_).
- Subsequent characters can be letters, $, (_ or digits).
- Cannot be a keyword (Ex: int, case, while)
- Can be any length.
- Case-sensitive: foobar $\neq$ fooBar

CS 200 Style and Good Practice

- Use lowerCamelCase: thisIsLowerCamelCase
A rose by any other name...

Naming Rules

- Starts with a letter, $, or underscore (_).
- Subsequent characters can be letters, $, (_) or digits.
- Cannot be a keyword (Ex: int, case, while)
- Can be any length.
- Case-sensitive: foobar ≠ fooBar

CS 200 Style and Good Practice

- Use lowerCamelCase: thisIsLowerCamelCase
- Don’t use $ or _
A rose by any other name...

Naming Rules

- Starts with a letter, $, or underscore (_).
- Subsequent characters can be letters, $, (_) or digits.
- Cannot be a keyword (Ex: int, case, while)
- Can be any length.
- Case-sensitive: foobar ≠ fooBar

CS 200 Style and Good Practice

- Use lowerCamelCase: thisIsLowerCamelCase
- Don’t use $ or _
- Length: not too long, but clearly identify the role.
**Type: int**

Declaration:

```java
int myInt = 2;
```

**Values**

- Integer in 4 bytes (32 bits)
Type: int

Declaration:

\[
\text{int myInt} = 2;
\]

Values

- Integer in 4 bytes (32 bits)
- \(-2, 147, 483, 648\) to \(2, 147, 483, 647\) inclusive
**Type: int**

Declaration:

```cpp
int myInt = 2;
```

**Values**

- Integer in 4 bytes (32 bits)
- $-2,147,483,648$ to $2,147,483,647$ inclusive

**Other integer types**

<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>1 byte</td>
<td>$-128$ to $127$</td>
</tr>
<tr>
<td>short</td>
<td>2 bytes</td>
<td>$-32,768$ to $32,767$</td>
</tr>
<tr>
<td>long</td>
<td>8 bytes</td>
<td>$-9,223,372,036,854,775,808$ to $9, ... , 807$</td>
</tr>
</tbody>
</table>
**Type: DOUBLE**

Declaration:

```java
double myDb1 = 1.618;
```

**Values**

- Real (floating point) number in 8 bytes (64 bits)
**Type: DOUBLE**

Declaration:

```java
double myDbl = 1.618;
```

**Values**

- Real (floating point) number in 8 bytes (64 bits)
- About 15 digits of precision
Type: DOUBLE

Declaration:

double myDbI = 1.618;

Values

- Real (floating point) number in 8 bytes (64 bits)
- About 15 digits of precision
- Range is approx. $\pm 1.7 \times 10^{308}$
**Type: double**

Declaration:
```java
double myDbl = 1.618;
```

**Values**
- Real (floating point) number in 8 bytes (64 bits)
- About 15 digits of precision
- Range is approx. $\pm 1.7 \times 10^{308}$

**Another real type: float**
**Type: Double**

Declaration:

```java
double myDbl = 1.618;
```

**Values**

- Real (floating point) number in 8 bytes (64 bits)
- About 15 digits of precision
- Range is approx. $\pm 1.7 \times 10^{308}$

**Another real type: Float**

- Real (floating point) number in 4 bytes (32 bits)
**Type: double**

Declaration:

```java
double myDbl = 1.618;
```

**Values**
- Real (floating point) number in 8 bytes (64 bits)
- About 15 digits of precision
- Range is approx. $\pm 1.7 \times 10^{308}$

**Another real type: float**
- Real (floating point) number in 4 bytes (32 bits)
- About 6 digits of precision
**Type: double**

Declaration:
```
double myDbl = 1.618;
```

**Values**
- Real (floating point) number in 8 bytes (64 bits)
- About 15 digits of precision
- Range is approx. $\pm 1.7 \times 10^{308}$

**Another real type: float**
- Real (floating point) number in 4 bytes (32 bits)
- About 6 digits of precision
- Range is approx. $\pm 3.4 \times 10^{38}$
TopHat Question 1

What is not a double literal?

a. 0.5
b. .5
c. 5
d. 5.5e5
**Constant Variable**

```java
final type VARIABLE_NAME = expression;
```

- **Final Modifier**
  - Cannot change the value after initialization.
**CONSTANT VARIABLE**

```java
final type VARIABLE_NAME = expression;
```

**final Modifier**
- Cannot change the value after initialization.

**CS 200 Style and Good Practice**
- Use constants to keep any hard-coded data in one place.
- Names are in ALL CAPS with _ between words. E.g.: SOME_CONST_NAME
Expressions
Assignments

someVar = expression

Assignment Operator: =

- Right-to-left associative:

E.g: a = b = c = d = 5 + 5;

TopHat Question 2: What is the value of a?
ASSIGNMENTS

someVar = expression

Assignment Operator: =

- Right-to-left associative:
  - Evaluate expression.
**Assignments**

\[ \text{someVar} = \text{expression} \]

**Assignment Operator:** =

- Right-to-left associative:
  1. Evaluate \( \text{expression} \).
  2. Assign the value of \( \text{expression} \) to \( \text{someVar} \).
Assignments

\[ \text{someVar} = \text{expression} \]

**Assignment Operator: \( = \)**

- Right-to-left associative:
  1. Evaluate \( \text{expression} \).
  2. Assign the value of \( \text{expression} \) to \( \text{someVar} \).

- E.g:
  \[
  a = b = c = d = 5 + 5;
  \]
Assignments

someVar = expression

Assignment Operator: =

- Right-to-left associative:
  1. Evaluate expression.
  2. Assign the value of expression to someVar

- E.g:
  a = b = c = d = 5 + 5;

TopHat Question 2: What is the value of a?
SWAPPING VARIABLES

Consider the following code snippet:

```java
int a = 2;
int b = 3;
```
Consider the following code snippet:

```java
int a = 2;
int b = 3;
```

TopHat Question 3: How can we swap the values?
Operators

Arithmetic Operators

- Unary:

  - Negation:
  - Addition:
  - Subtraction:
  - Multiplication:
  - Division:
  - Remainder (modulo):

- Compound:
  - Add and assign:
    - E.g. $a += b$ is shorthand for $a = a + b$
  - Other variants:
    - $-=$, $*=$, $/=$, $%=$
Operators

Arithmetic Operators

• Unary:
  • -: Negation
Operators

Arithmetic Operators

- **Unary:**
  - `-:` Negation
- **Binary:**
OPERATORS

Arithmetic Operators

- **Unary:**
  - `-`: Negation

- **Binary:**
  - `+`: Addition
Operators

Arithmetic Operators

- **Unary:**
  - `-`: Negation

- **Binary:**
  - `+`: Addition
  - `-`: Subtraction

E.g. `a += b` is shorthand for `a = a + b`.

Other variants: `-=`, `*=`, `/=`, `%=`
## Operators

### Arithmetic Operators

- **Unary:**
  - `-`: Negation

- **Binary:**
  - `+`: Addition
  - `-`: Subtraction
  - `*`: Multiplication
  - `/`: Division
  - `%`: Remainder (modulo)

- Other variants: 
  - `-=`
  - `*=`
  - `/=`
  - `%=`
Operators

Arithmetic Operators

- Unary:
  - -: Negation

- Binary:
  - +: Addition
  - -: Subtraction
  - *: Multiplication
  - /: Division

E.g. `a += b` is shorthand for `a = a + b`
OPERATORS

Arithmetic Operators

- **Unary:**
  - -: Negation

- **Binary:**
  - +: Addition
  - -: Subtraction
  - *: Multiplication
  - /: Division
  - %: Remainder (modulo)
Operators

Arithmetic Operators

- **Unary:**
  - -: Negation

- **Binary:**
  - +: Addition
  - -: Subtraction
  - *: Multiplication
  - /: Division
  - %: Remainder (modulo)

- **Compound:**
Operators

Arithmetic Operators

- **Unary:**
  - -: Negation

- **Binary:**
  - +: Addition
  - -: Subtraction
  - *: Multiplication
  - /: Division
  - %: Remainder (modulo)

- **Compound:**
  - +=: Add and assign
    E.g. a += b is shorthand for a = a + b
Operators

Arithmetic Operators

- **Unary:**
  - `-`: Negation

- **Binary:**
  - `+`: Addition
  - `-`: Subtraction
  - `*`: Multiplication
  - `/`: Division
  - `%`: Remainder (modulo)

- **Compound:**
  - `+=`: Add and assign
    - E.g. `a += b` is shorthand for `a = a + b`
  - Other variants: `-=` , `*=` , `/=` , `%=`
## Operator Precedence

**Operator Precedence Table:**

<table>
<thead>
<tr>
<th>Level</th>
<th>Operator</th>
<th>Description</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>higher</td>
<td>(  )</td>
<td>grouping with parentheses</td>
<td>left to right</td>
</tr>
<tr>
<td></td>
<td>[ ]</td>
<td>array index, method call, member access</td>
<td>left to right</td>
</tr>
<tr>
<td></td>
<td>( )</td>
<td>(dot operator)</td>
<td>left to right</td>
</tr>
<tr>
<td></td>
<td>++ --</td>
<td>post-increment, post-decrement</td>
<td>left to right</td>
</tr>
<tr>
<td></td>
<td>++ -- + -- !</td>
<td>pre-increment, unary plus/minus, logical negation</td>
<td>right to left</td>
</tr>
<tr>
<td></td>
<td>(type) new</td>
<td>casting and creating object</td>
<td>right to left</td>
</tr>
<tr>
<td></td>
<td>* / %</td>
<td>multiplication, division, modulus</td>
<td>left to right</td>
</tr>
<tr>
<td></td>
<td>+ - +</td>
<td>addition, subtraction, concatenation</td>
<td>left to right</td>
</tr>
<tr>
<td></td>
<td>&lt; &lt;= &gt; &gt;=</td>
<td>relational and Java’s instanceof operator</td>
<td>left to right</td>
</tr>
<tr>
<td></td>
<td>instanceof</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>== !=</td>
<td>equality</td>
<td>left to right</td>
</tr>
<tr>
<td></td>
<td>&amp;&amp;</td>
<td>conditional AND (short-circuits)</td>
<td>left to right</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>? :</td>
<td>ternary conditional</td>
<td>right to left</td>
</tr>
<tr>
<td>lower</td>
<td>= += -= *= /= %=</td>
<td>assignment</td>
<td></td>
</tr>
</tbody>
</table>
### Operator Precedence

<table>
<thead>
<tr>
<th>Level</th>
<th>Operator</th>
<th>Description</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>higher</td>
<td>( &lt;expression&gt; ) [ ] ( ) . ++ -- ++ -- + + ! (type) new * / % + - + &lt; &lt;= &gt; &gt;= instanceof == != &amp;&amp;</td>
<td>grouping with parentheses array index, method call, member access (dot operator) post-increment, post-decrement pre-increment, unary plus/minus, logical negation casting and creating object multiplication, division, modulus addition, subtraction, concatenation relational and Java’s instanceof operator equality conditional AND (short-circuits) conditional OR (short-circuits) ternary conditional assignment</td>
<td>left to right left to right left to right right to left left to right left to right left to right right to left right to left</td>
</tr>
<tr>
<td>lower</td>
<td>= += -= *= /= %=</td>
<td>left to right left to right left to right</td>
<td></td>
</tr>
</tbody>
</table>

---

**CS 200 Style and Good Practice**

- Use parentheses to explicitly indicate the precedence.
**Type Conversion**

char

\[\downarrow\]

byte $\rightarrow$ short $\rightarrow$ int $\rightarrow$ long $\rightarrow$ float $\rightarrow$ double

Narrowest $\rightarrow$ Widest
**Type Conversion**

```
  char
  ↓
byte → short → int → long → float → double
```

Implicit Conversion

- Arithmetic operators: Implicit cast to the widest type.
- Assignment: Implicit cast RHS to LHS type if wider. E.g.
  ```java
double dbl = 2;
  ```
Type Conversion

char
↓
byte → short → int → long → float → double

Implicit Conversion

- Arithmetic operators: Implicit cast to the widest type.
- Assignment: Implicit cast RHS to LHS type if wider. E.g.
  double dbl = 2;

Explicit Conversion: Casting

(type)
E.g.
int i = (int)2.5;
Integer Division

Truncation with \(/\)

- When both operands are integers (\texttt{int}, \texttt{short}, etc), the result will also be an integer.

\[ \text{int} \ a = 3 / 2; \quad \text{// Sets } a \text{ to 1} \]
\[ \text{double} \ b = (\text{double})(3 / 2); \quad \text{// Still 1} \]
\[ \text{double} \ c = (\text{double})3 / 2; \quad \text{// Now 1.5} \]
Integer Division

Truncation with `/`

- When both operands are integers (int, short, etc), the result will also be an integer.
- Java will *truncate* the result: Chop off the decimal.
## Integer Division

### Truncation with `/`

- When both operands are integers (int, short, etc), the result will also be an integer.
- Java will *truncate* the result: Chop off the decimal.
- E.g.:
  ```java
  int a = 3 / 2; //Sets a to 1
  double b = (double)(3 / 2); // Still 1
  double c = (double)3 / 2; // Now 1.5
  ```
**Simple Memory Model**

Memory

0xffffffff

0xff000000
Simple Memory Model

```java
int i = 6;
```
Simple Memory Model

Key Points:
- Variables are allocated slots in memory with a memory address.
- The value of the variables are stored in the assigned memory slots.

```cpp
int i = 6;
int j = 6;
```
**Simple Memory Model**

```java
int i = 6;
int j = 6;
double d = 2.5;
```
**Simple Memory Model**

```plaintext
int i = 6;
int j = 6;
double d = 2.5;
short s = 25;
```
### Simple Memory Model

Variables are allocated slots in memory with a memory address.

The value of the variables are stored in the assigned memory slots.

```
int i = 6;
int j = 6;
double d = 2.5;
short s = 25;
```
Methods
Methods and Functions

What is a Method?

- Evolved from mathematical functions, e.g. \( f(x) = 2x + 1 \)
Methods and Functions

What is a Method?
- Evolved from mathematical functions, e.g. \( f(x) = 2x + 1 \)
- A parameterized block of code that performs a sequence of instructions and returns a single value.
Methods and Functions

What is a Method?

- Evolved from mathematical functions, e.g. \( f(x) = 2x + 1 \)
- A parameterized block of code the performs a sequence of instructions and returns a single value.
- Can have many parameters, but only a **single** return value.
Methods and Functions

What is a Method?
- Evolved from mathematical functions, e.g. \( f(x) = 2x + 1 \)
- A parameterized block of code that performs a sequence of instructions and returns a single value.
- Can have many parameters, but only a **single** return value.

Why use Methods?
- Abstraction: modular design.
- End of redundant code: tempted to copy-and-paste code; make a method.
- Good coding practice: easier to read.
Structure

modifiers returnType methodName(pType1 pName1, ...) {
  body
}

Method Terminology

Modifiers E.g. public, static, final.

Header Modifier, return type, method name, and the types and names of the parameters.

Signature Method name and the types of the parameters.

Body Block of code containing the method’s instructions.

Definition The header and the body.
Structure

E.g.
public static double avgTwoInts(int p1, int p2) {
    return (p1 + p2)/2.0;
}

Method Terminology

Modifiers  E.g. public, static, final.

Header  Modifier, return type, method name, and the types and names of the parameters.

Signature  Method name and the types of the parameters.

Body  Block of code containing the method’s instructions.

Definition  The header and the body.
public static double avgTwoInts(int p1, int p2) {
    return (p1 + p2)/2.0;
}

Calling a Method

avgTwoInts(5, 7)

- methodName(arg1, arg2, ...)
- The call of the method is evaluated to the return value of the method, using the arguments provided.
PARAMETERS

```java
public static double avgTwoInts(int p1, int p2) {
    return (p1 + p2)/2.0;
}
```

Method Parameters

- **Scope**: considered as variables within the body of the function.
- **Pass by value**: changing the value in the method doesn’t change the passed value.
- **Initialized at the moment of the method call.**
TopHat Question 4

What is the output?

```java
class TopHat4 {
    public static int someMethod(int a) {
        a = 5;
        return a;
    }

    public static void main(String[] args) {
        int a = 4;
        someMethod(a);
        System.out.print(a);
    }
}
```
Return Value

Method Value (return)

- The method is evaluated with the passed parameters.
- `void` methods do not return a value.
- All other methods return a **single** value.
Return Value

Method Value (return)

- The method is evaluated with the passed parameters.
- `void` methods do not return a value.
- All other methods return a single value.

Method Side Effects

- Anything in a method that changes the state of the running program is considered a side effect, such as:
**Return Value**

**Method Value (return)**
- The method is evaluated with the passed parameters.
- `void` methods do not return a value.
- All other methods return a **single** value.

**Method Side Effects**
- Anything in a method that changes the state of the running program is considered a side effect, such as:
  - Printing to the screen.
  - Reading input from the user.
  - Changing *global* variables.
TopHat Question 5

What is the value of the following method when called by `myMethod(10)`:

```java
public static int myMethod(int i) {
    System.out.println("Value is " + i);
    return 6;
}
```

a. 6  
b. 10  
c. Value is 10  
d. void
Method Comments
CS 200 and Good Practice

/**
 * Calculates the average of two integer values.
 *
 * @param p1 First integer value.
 * @param p2 Second integer value.
 * @return Average of p1 and p2.
 */

public static double avgTwoInts(int p1, int p2) {
    return (p1 + p2)/2.0;
}
Method Comments
CS 200 and Good Practice

/**
 * Calculates the average of two integer values.
 *
 * @param p1 First integer value.
 * @param p2 Second integer value.
 * @return Average of p1 and p2.
 */

public static double avgTwoInts(int p1, int p2) {
    return (p1 + p2)/2.0;
}

Javadoc Comments

1. An explanation of the method.
2. @param An explanation for each parameter.
3. @return An explanation of the value returned.
Method Exercise

Take the program circle.java that we wrote and refactor it to use methods. Consider the code, what might be some reasonable methods to add?
Using Libraries
Libraries

Code Libraries

- A collection of classes and methods.
- Java comes with a set of standard libraries such as `java.util.Scanner` and `java.lang.Math`.
- 3rd Party Libraries also exist.
Math Class

Standard Java class (optional for Math):
import java.lang.Math;

Basic Numeric Operations
- Collection of static methods and constants.
- Methods: pow, abs, ceil, etc...
- Members: E, PI
- API: https://docs.oracle.com/javase/8/docs/api/java/lang/Math.html
FURTHER READING

*COMP SCI 200: Programming I*
zyBook code:
WISCCOMPSCI200Fall2019

- Chapter 2. Primitives and Expressions
REFERENCES
Image Sources I

https://brand.wisc.edu/web/logos/

http://www.zybooks.com/