CS180 Recitation 3
Lecture: Overflow

```java
byte b;
b = 127;
b += 1;
System.out.println("b is" + b); b is -128
```

```java
byte b;
b = 128;  //will not compile!
```

b went out of bounds and wrapped around. **Overflow.**

**Example**

```
  1 0 0
+ 1 1 0
  1 0 1 0
```
Lecture: Underflow

Similarly **underflow** can occur.

byte b;

b = -128;

b -= 1;

Every type has their own range. Variable types should be chosen wisely.
Data Types

* Primitive Data Types

**Numeric Data Types**

Integer Numbers
- byte (8-bits), short (2-bytes)
- int (4-bytes), long (8-bytes)

Real Numbers
- float (4-bytes), double (8-bytes)

**Character Data Types**

- char (2-bytes)

**Logic**

- boolean - Requires 1-bit information (true or false) but its size is not precisely defined.

* Reference Data Types

- Object (varied-sized bytes)
- String
- Why many data types?
  
  • Small numbers require less bytes for representation.
  • Precision – Some numbers need to be represented precisely for computation.

-Type safety in Java

  • Java is strongly typed. The type of every variable must be declared before its use.
  • This allows Java to perform many safety and sanity checks during compilation and avoids many program errors.
Variables

The format of variable declaration is

\[ \text{data\_type} \ \text{variable\_name}; \]

Variable Declaration:

- Variable declaration sets aside memory to hold the variable and attaches a name to that space.
- No need to use \textit{new} operator.

Examples:

- \textit{double} volume;
  
  - Memory is allocated to store double values.
- \textit{int} num;
  
  - Memory is allocated to store integer values.
Variable assignment and Initialization

* float x = 10.05f; //x is initialized to 10.05.
* float x;
  x = 10.05f;
* float x = 10.0f; //float variable x is initialized to 10.0
  float y = 20.0f; //float variable y is initialized to 20.0
  x = y; //variable assignment. x has value 20.0

Different from Object assignment

- Student student1;
- student1 = new Student();

The reference to the first object is stored in student1.

- Student student2;
- student2 = student1;

The reference stored in student1 is copied to student2.
Variables, Literals, Constants : Things to remember

- Java does not allow two variables to have the same name within the same method (More details later).
  - int num = 5;
  - int num = 10;

- Changing from one type to another is called casting.
  - double value = 10.008d;
  - int num = (int) value;

- Here, the values, 5 and 10 are called literals.
  - The type of 10 is int.
  - The type of 10.0 is double.

- Constants
  Precede variable declaration with 'final' keyword.
  - final double PI = 3.14f;
One’s Complement Representation

Positive number uses positional representation.

Negative number formed by inverting all bits of positive value.

Example: 4-bit representation

- 0 0 1 0 represents 2
- 1 1 0 1 represents -2
Two’s Complement Representation

Positive number uses positional representation.

Negative number formed by subtracting 1 from positive value and inverting all bits of result.

Example: 4-bit representation

- 0 0 1 0 represents 2
- 1 1 1 0 represents -2
- High-order bit is set if number is negative
Integer representation

Positive Numbers – Binary Equivalent

• \texttt{short} \ x = 8; \ // \ short – 2 bytes
• \ 8 \ = \ 00000000 \ 00001000
• \texttt{short} \ y = 10;
• \ 10 \ = \ 00000000 \ 00001010
• \texttt{short} \ z = x + y; \ // \ Binary \ Addition

Negative Numbers – Two's complement Binary Equivalent

• \ short \ x = -8;
• \ (-8) \ will \ be \ stored \ in \ two's \ complement \ representation
Converting a negative number to two's complement representation

Step 1: Convert the positive number to binary equivalent.
Step 2: Flip the 0's to 1's and 1's to 0's. (One's complement)
Step 3: Add 1 to the one's complement number obtained in step 2.

Example:

short x = -8;
1. $8 = 00000000 \ 00001000$
2. $11111111 \ 11110111$
3. $11111111 \ 11101111 + 1 = 11111111 \ 11111000$
IEEE floating point representation

IEEE floating point numbers are represented using 32 bits (single precision) or 64 bits (double precision).

Three Components

Sign bit - Single bit. 0 represents negative number and 1 represents positive number.

Exponent – The exponent field needs to represent both positive and negative exponents.

Mantissa - Significant digits of the number.  \( a \times b^e \)
   \( a \) – mantissa, \( e \) – exponent, \( b \) - base

Floating point numbers are typically stored in normalized form. This basically puts the radix point(that which separates the integer part and the fractional part) after the first non-zero digit. In normalized form, five is represented as  \( 5.0 \times 100 \).
Single Precision
Sign bit - 1 bit
Exponent – 8 bits
Mantissa – 23 bits

Double Precision
Sign bit - 1 bit
Exponent – 11 bits
Mantissa – 52 bits

Errors in Floating point Arithmetic
  _ Many rational numbers can only be approximately represented in memory.
  _ The arithmetic on these numbers yield approximate answers.
  _ These errors get propagated in floating point calculations.
Errors in floating point arithmetic: continued

Example:

- Converting 0.3 to binary yields 0.01001 which is approximately 0.28125.

- In this case, five bits are not enough to represent 0.3 fully. We have an error of
  \[ 0.3 - 0.28125 = 0.01875 \]

- 1.3 cannot be represented exactly using a 32-bit value.
  \[ 1.3 \times 3.0 \text{ will be } 3.899999999 \text{ instead of } 3.9 \]
Arithmetic operators for Numeric data

Addition
    int a = 10;
    double x = a + 35.98  // 'a' is promoted to double

Subtraction
    double a = 5.48;
    int b = 10;
    b = b – (int) a;       // Needs explicit casting

Multiplication
    short a = 2;
    int b = 4;
    double c = a * b;
Division

*Integer Division*

- $20/10 = 2$
- $20/11 = 1$

*Real Division*

When either or both numbers are float or double, then the result is a float or double respectively.

- $10.0/5.0 = 2.0$
- $5.0/2 = 2.5$

Double $y = 1999.0/1000$

double $z = 1999/1000$

What is the value of $y$ and $z$?
Modulo

Returns the remainder of a division, usually involves only integers.

Examples:
11 % 5 = 1
23 % 5 = 3
8 % 2 = 0

Note: In both division and modulus operations, the second number should not be 0.
Arithmetic Operators precedence

In decreasing order of precedence,

Subexpression  ( )  Starting with innermost.

Unary Operators  -, +       Left to right evaluation.

Multiplicative operators  *, /, %       Left to right evaluation.

Additive operators  +, -       Left to right evaluation.

Example:

• int a = 31, b = 16, c = 1, d = 2;
• int k = b + c * d – a / b / d;
• int l = (-b + c) * a / d;
Math Class

Math class is very powerful. It provides all kinds of useful methods such as:

- `abs(a)`: absolute value of a
- `max(a, b)`: the larger of a and b
- `min(a, b)`: the smaller of a and b
- `pow(a, b)`: a raised to power b
- `round(a)`: a rounded to the nearest whole number
- `sqrt(a)`: square root of a
- `ceil(a)`: smallest whole number no less than a
- `floor(a)`: largest whole number no bigger than a
- `sin(a)`: trigonometric sine of a
- `cos(a)`: trigonometric cosine of a
- `exp(a)`: natural number e(2.718…) raised to power a
public class MathDemo{
    public static void main(String[] args){
        //E and round()
        System.out.println("e = " + Math.round(Math.E*100)/100f);
        //PI
        System.out.println("pi = " + Math.round(Math.PI*100)/100f);
        //abs()
        System.out.println("Absolute number = " + Math.abs(Math.PI));
        //ceil()
        System.out.println("Integer greater than or equal to = " + Math.ceil(Math.PI));
        //exp()
        System.out.println("Exponent number powered by the argument = " + Math.exp(0));
        //min()
        System.out.println("Minimum Number = " + Math.min(10,10.3));
        //pow()
        System.out.println("Power = " + Math.pow(10,3));
    }
}