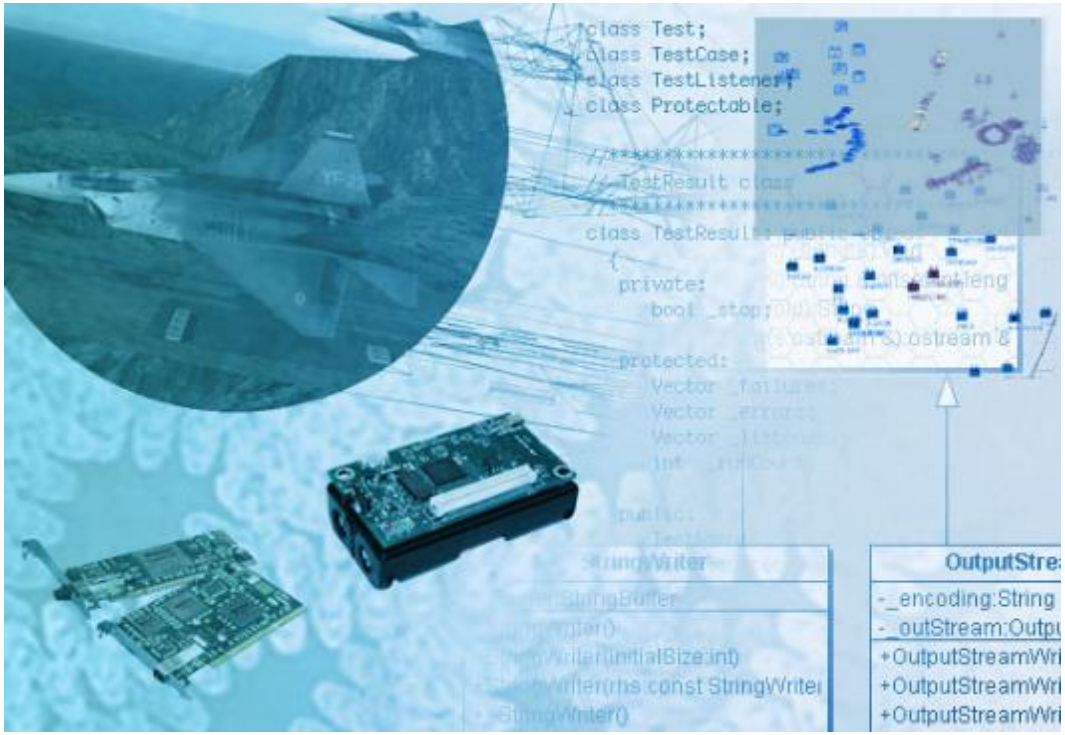
CSYE 6200

CONCEPTS OF OBJECT-ORIENTED DESIGN
SESSION 2

MARK G. MUNSON



ADMINISTRATION

- **What do you need**
  - Computer (Windows, Mac, or Linux)
  - Books – see the syllabus
- **Expectations**
  - Full attendance – only two classes may be missed
    - Course auditing is not permitted
    - Attendance will be tracked
- **Class TA**
  - Rishabh Sood – sood.r@northeastern.edu

THE LECTURE

- Installing Java
- Recap
- Java Expressions and Operators
- Program Control
- Looping
- Arrays
- Methods
- Assignment #1

JAVA EXPRESSIONS & OPERATORS

EXPRESSIONS

- Variables and literal values may be used in expressions
- Java supports standard operators for mathematics

Addition (+)	int x = 5 + 7;
Subtraction (-)	int y = x – 3;
Multiplication (*)	int z = x * y;
Division (/)	int a = z / 4;

ADDITIONAL OPERATORS

- **The remainder of an integer division may be computed using the modulo ("mod") operator**

  remainder = value % divisor

- **Examples:**
  - 15 % 3 = 0
  - 10 % 3 = 1
  - 12 % 15 = 12

ASSIGNMENT OPERATOR

The assignment operator has the form:

var* = *expression

Internally evaluated as var.operator=(*expression*)

Assignment may be chained:

```
int x, y, z;
x = y = z = 7;
```

Same as:

```
x = (y = (z = 7)))
```

OPERATOR PRECEDENCE

- **Each expression is evaluated using operator precedence**
  - ( ) Highest
  - * / %
  - + - Lowest
- **Operators with highest precedence are evaluated first, the lowest are evaluated last.**
- **Operators with equal precedence are evaluated from left-to-right.**
- **Example: x = 5 + (4 * 5) / 2**
  - x = 15

OPERATOR CONVENIENCE

Programs commonly have expressions like:

x = x + 1; or y = y / 8;

As a convenience, expressions of the form:

variable* = *variable* *oper value

may be expressed using a simplified notation:

variable* *oper= value

Expression	Convenience
x = x + 2	x += 2
x = x - 2	x -= 2
x = x * 2	x *= 2
x = x / 2	x /= 2

INCREMENT AND DECREMENT

It's also common in programs to increment and decrement variables:

```
variable = variable + 1; // increment by one
```

or

```
variable = variable - 1; // decrement by one
```

This happens so often, there is a special operator, so we can write the following instead:

```
variable++; // increment by one
```

and

```
variable--; // decrement by one
```

When the the C language was extended to add support for objects, the new language was called C++.

INCREMENT PREFIX/POSTFIX

- Increment/Decrement may be expressed in Prefix or Postfix form:

  `++val` or `--val` (Prefix form)

  `val++` or `val--` (Postfix form)

- All operators return a value that me be used in conditional evaluations

```
int cnt = 10;
if (cnt++ > 10) { }
```

resolves to `if (11 > 10) { }`

10

or

```
return(cnt++)
```

JAVA PRIMITIVE TYPES

Type	Meaning
boolean	Logical True/False values
byte	8-bit integer
char	Character
float	Single-precision floating point
double	Double-precision floating point
short	Half-precision integer
int	Integer
long	Double-precision integer

INTEGER NUMBERS

Type	Range	bits	Memory (bytes)
byte	-128 to 127	8	
short	-32,768 to 32,767	16	
int	-2,147,483,648 to 2,147,483,647	32	
long	+/- 9,223,372,036,854,775,807	64	

- **The default number type in Java is the int**
- **All Java values are signed [+/-]**
  (C/C++ allows for unsigned values)
- **Long values are typically used to hold the system clock time**

REAL NUMBERS

Type	Form	bits	Memory (bytes)
float	IEEE 754 single precision	32	
double	IEEE 754 double precision	64	

- **Used for most floating point calculations**
- **Single precision float values are used to support older interfaces (like OpenGL graphics)**
- **Most math routines in Java use the double type, so this is the default choice**
- **Currency calculations should not use this type, but instead use the BigDecimal class**

CHARACTER

- **Characters in most other languages (like C) are encoded as 8-bit ASCII (American Standard Code for Information Interchange)**
  - ASCII provides A-Z, 0-9, control characters, plus special and lower case characters
  - Could work on a 7 bit computer
- **Java switched to 16 bit UniCode characters**
  - Provided support for multiple language fonts
  - Supports bidirectional display
- **Example assignment:**
  - `char c = 'g';`

BOOLEAN

- Represents a value of True or False
- Evaluates numerically in C as:

  True = 1

  False = 0

- Unlike C/C++, Java <u>does not permit</u> boolean expressions to be used in calculations

  (i.e. `int x = 5 + (y > 5) * 3;`)

RELATIONAL OPERATORS

- **Relational operators evaluate a test condition to produce a boolean true or false result**

Comparators:

==	equals
!=	not equals
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to

LOGICAL OPERATORS

Logical operators evaluate two values to produce a result that can be evaluated as true or a false

&	**AND** ***(result of 0x000F & 0x1234 is 4)***
|	**OR** ***(result of 0x000F | 0x1234 is 0x123F)***
^	**XOR (exclusive OR)** **Used in non-destructive graphics**
!	**NOT**
&&	**Logical AND** ***(faster and always returns True or False)***
||	**Logical OR** ***(faster and always returns True or False)***

LITERALS

- **Primitive types may be assigned values from literals**
  - Default types are integers and doubles

```
int count = 5;
double wallLength = 23.3;
```

- **Float values may be specified using an ‘F’ or ‘f’ suffix**

```
float redVal = 0.45f;
```

- **Long values may be specified using an ‘L” or ‘l’ suffix**

```
long timeDelta = 1045L;
```

- **Hex and Octal values**

```
int mask = 0x1A2F; // Hexidecimal(base 16)
int nine = 011; // 8 + 1 (base 8)
byte threeXfour = 0b1100; // 8 + 4 (base 2)
```

CASTING

- Primitive value types may be converted from one form to another through the use of casting "( )"
- Examples:

```
int count = 5; long timeCnt = 27L;
double dValue = (double) count;
int timeDelta = (int) timeCnt;
float greenVal = (float) dValue;
```

  Some rounding/clipping may result

- Casting may be combined with precedence

```
int minSize = (int) (xLen / yLen);
float greenVal = (float) (dVal * 2.34187);
```

INTEGER

PROGRAM CONTROL

PROGRAM CONTROL

- **Concepts**
  - Flow Control
    - `if`
    - `if-else`
    - `switch`
  - Looping
    - `for`
    - `while`
    - `do-while`
  - Loop exiting
    - `break`
    - `continue`

IF

- **The if Statement**
  - Allows code to selectively execute parts of a program

```
if (condition) statement;
or
if (condition) { statements }
```

- **Examples**

```
if (c < 10) System.out.println("c is less than ten");
if (c == 5) {
     System.out.println("c is equal to five");
     c = c+1;
     System.out.println("c plus one is : " + c);
}
```

IF-ELSE

- **The if-else Statement**
  - Allows code to selectively execute two paths of a program

```
if (condition) statement;
else statement;
```

or

```
if (condition) { statements }
else { statements }
```

- **Examples**

```
if (c < 10) System.out.println("c is less than ten");
else System.out.println("c is >= to ten");

if (c == 5) {
      System.out.println("c is equal to five");
      c++;
}
else {
     d = c * 2;
     c = c + 2;
     System.out.println("c is : " + c);
}
```

IF-ELSE CONT.

If statements may be nested

```
if (c > 5) {
     System.out.println("c is greater than five");
     if (cSize == 6) {
        System.out.println("cSize is equal to six");
     }
     c++;
}
```

If-else statements may be laddered

```
if (c == 1) System.out.println("Type 1");
else if (c == 2) System.out.println("Type 2");
else if (c == 3) System.out.println("Type 3");
else System.out.println("Type Unknown");
```

Laddering is so common, that a special flow control, called 'switch' was added to make it faster and simpler.

SWITCH

- `Switch` is a multi-way branch, that allows you to select from multiple alternatives

```
switch(expression) {
        case constant1:
                statement;
                break;
        case constant2:
                statement;
                break;
        default:
        case constant3:
                statement;
                break;
}
```

CONSTANTS

- Not all variables can vary. Some are constant.

```
double area = 3.1415 * (r * r);
double weight = 9.8 * m;
```

- The inclusion of undocumented literal values is discouraged. A preferred approach is to use variables that are defined to be constant.

```
private static final double GRAVITY_ACCEL = 9.8;
```

- By convention, a constant value is named with UPPER_CASE_WITH_UNDERSCORES
- In the world of 'C', the `const` keyword is used instead of `final`.

SWITCH

Block.java UNREGISTERED

Block.java

```java
    public final static int FORM_BILLBOARD = 2;
    public final static int FORM_CUBETOP = 3;
    public final static int FORM_FOLIAGE = 4;

   /**
     * For a supplied block form value, return a String representation
     * @param blockForm the block form to select
     * @return a string description of the block form
     */
    static public String getBlockFormStr(int blockForm) {
        switch (blockForm) {
        default:
        case FORM_CUBE: return ("FORM_CUBE");
        case FORM_SLOPED: return ("FORM_SLOPED");
        case FORM_BILLBOARD: return ("FORM_BILLBOARD");
        case FORM_CUBETOP: return ("FORM_CUBETOP");
        case FORM_FOLIAGE: return ("FORM_FOLIAGE");

        }
}
```

Line 7, Column 46 Tab Size: 4 Java

LOOPING

FOR

- **The for Loop**
  - Repeatedly execute a sequence of code

```
for (initialization; condition; iteration) statement;
   or
for (initialization; condition; iteration ){statements}
```

  - Example

```
for (int count = 0; count < 10; count++) {
   System.out.println("This is count " + count);
}
```

WHILE

- **The while Loop**
  - Repeatedly execute a sequence of code

```
while (condition) statement;
or
while (condition){statements}
```

  - Example

```
int counter = 0;
boolean done = false;
while (!done) {
   System.out.println("This is loop pass " + counter);
   counter++;
   if (counter == 10) done = true;
}
```

DO-WHILE

- **The do-while Loop**
  - Repeatedly execute a sequence of code

```
do {statements}
while (condition);
```

  - Evaluation occurs at the end, not at the beginning
  - {statements} are always executed at least once
  - Example

```
int counter = 0;
boolean done = false;
do {
   System.out.println("This is loop pass " + counter);
   counter++;
   if (counter == 10) done = true;
   }
while (!done);
```

BREAK

- Sometimes you want to exit a loop early. The `break` keyword forces an exit from the current loop, and can jump to a named block of code.

```
break label;
```

- Example:

```
for (int i = 0; i < 10; i++) {
    System.out.println("Counter i = " + i);
    int iSqr = i * i;
    if (iSqr == 9) break;
    System.out.println("iSqr = " + iSqr);
}
System.out.println("Done");
```

BREAK (CONT.)

SwitchDemo.java

```java
class SwitchDemo {
    public static void main(String args[]) throws java.io.IOException {
        char inChar = ' ';
        System.out.println("Enter choice (a,b,c), or \'q\' to quit, then press <enter>,");
        do {
            inChar = (char) System.in.read();
            switch (inChar) {
                case 'a':
                    System.out.println("You selected option a");
                    break;
                case 'b':
                    System.out.println("You selected option b");
                case 'c':
                    System.out.println("You selected option c");
                    break;
                case 'q':
                case '\n':
                    // do nothing
                    break;
                default:
                    System.out.println("Invalid selection"); // Show help?
                    break;
            }
        }
        while (inChar != 'q'); // Exit on quit
        System.out.println("Quiting...");
    }
}
```

CONTINUE

- To force an early completion of a loop pass, use the `continue` statement
- Example:

```
int sum = 0;
for (int i = 0; i <= 100; i++) {
    sum += i;
    if ((i%10) != 0) continue;
    System.out.println("i = " + i);
}
System.out.println("Done");
```

BREAK (10 MIN)

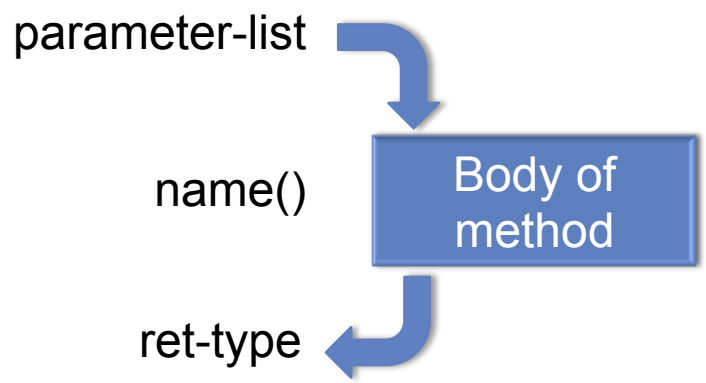
METHODS

METHODS

- **To simplify your code, complex activities may be broken up into smaller tasks. Each task can be performed by a stand-alone piece of software called a method.**
- **Methods have the general form:**

```
ret-type name(parameter-list) {
// body of method
}
```

- **Parameters are passed to a method, and a return value may be returned.**



METHOD EXAMPLES

MethodDemo.java UNREGISTERED

Vehicle.java | MethodDemo.java

```java
class MethodDemo {

    static long calcTimeDelta(long startTime, long endTime) {
        long delta = endTime - startTime;
        return delta;
    }

    static double calcTimeDeltaSecs(long startTime, long endTime) {
        long delta = endTime - startTime;
        double secs = ((double) delta / 1000.0); // mSec to Seconds
        return secs;
    }
```

ne 30, Column 2 Tab Size: 4 Java

METHOD EXAMPLE

```java
/*
Demonstration of method calls
Filename MethodDemo.java
*/

class MethodDemo {

    static long calcTimeDelta(long startTime, long endTime) {
        long delta = endTime - startTime;
        return delta;
    }

    static double calcTimeDeltaSecs(long startTime, long endTime) {
        long delta = endTime - startTime;
        double secs = ((double) delta / 1000.0); // mSec to Seconds
        return secs;
    }

    public static void main(String args[]) {
       long start = System.currentTimeMillis();
       for (int count = 0; count < 100; count++)
          System.out.println("This is count: " + count);
       long end = System.currentTimeMillis();

       System.out.println("Start time: " + start);
       System.out.println("End time: " + end);
       System.out.println("Elapsed Time: " + calcTimeDelta(start, end));
       System.out.println("Elapsed Time: " + calcTimeDeltaSecs(start, end) + " seconds
    }
}
```

Line 28, Column 8 Tab Size: 4 Java

ASSIGNMENT #1

ASSIGNMENT 1 – PASSWORD

Assignment 1: Due Feb. 8th, 6:00 pm EST

- **Write a java program that accepts password entries**
- **Use a loop to accept keyboard command input**
- **Use a switch statement to provide for four input options (commands):**
  1. Input a password, and then print a sum of the ASCII values for each character
  2. Input a password, and convert each character to a number and print its value
  3. Print a history of the last three password attempts and the corresponding sum value from (1.)
  4. Create a special password based on your first name plus two characters from your last name. If the user types this password in step (2.) then print "Special password accepted" to the console screen
  5. Quit – Allow the user to exit from the program
- **Please comment your code, and think about making your code simple and easy to write**
- **Submit your source code to Canvas. Please fill in your name and NUID number.**

NEXT WEEK / ASSIGNMENT

- **JABG: Read Ch. 4 and 5**
- **Assignment #1: Due , 6:00 pm (prior to class)**
  - Write in your name and NUID number.
  - **Submit your source code to Canvas.**