CS415
INTRODUCTION TO COMPUTER SCIENCE
FALL 2017

10 EXPRESSIONS
CHAPTER 6 AND 10

LAST TIME

- Polymorphism in Java
  - Inheritance
  - Interfaces
- Using polymorphism

PREVIEW

- Random Numbers
- Numbers and operators:
  - Integers
  - Floating point
- Constants
- Class variables and methods
- The Math class

RANDOM NUMBERS

- Sometimes in a program we might want to have some "random" behavior.
  - flip a coin, roll the dice, deal a card.
- Java provides methods to generate "pseudo-random" number sequences.
- A pseudo-random sequence is a sequence that "appears random" even though it is generated by deterministic process.

RANDOM NUMBER SEEDS

- A pseudo-random generator begins with an initial value called a seed.
- The sequence of numbers generated with a given seed is always the same.
- This is useful for testing.
- To get different sequences generated you start with different seeds.
- Often the time on the system clock is used as a seed (*after* testing is done).

JAVA.UTIL.RANDOM

- First create a (pseudo)random generator:

```
import java.util.Random;

Random gen1 = new Random( 12345 ); // seed for testing
Random gen2 = new Random( ); // seed based on system clock
```

- You can now generate (pseudo)random values

```
double d = gen1.nextDouble( );       //    0.0 <= x   < 1.0

int i = gen1.nextInt( 6 );           //    0   <= i   < 6

int die = gen1.nextInt( 6 ) + 1;     //    1   <= die <= 6

float f = gen1.nextFloat( ) * 100;   //    0.0 <= f   < 100.0

boolean win = gen1.nextBoolean( );   //   win == true or win == false
```


PRIMITIVE TYPES

- Java implements several simple types of values as primitive types.
  - Integers: 0, -5, 125, ...
  - Floating point numbers: 1.234, 0.0, ...
  - Booleans: true, false
  - Characters: 'a', 'B', '$', ...

JAVA INTEGER TYPES

Type	Size (Bytes)	Minimum Value	Maximum value
byte	1	-128	127
short	2	-32,768	32,767
int	4	-2,147,483,648	2,147,483,647
long	8	-9.2x10^18	9.2x10^18

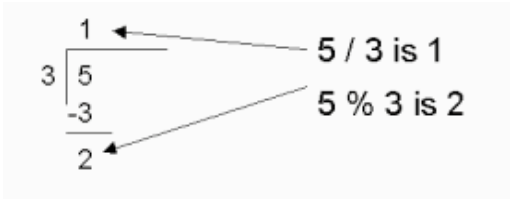
ARITHMETIC OPERATIONS

- Java provides the following arithmetic operators:
  - Binary operators (require two arguments) `*, /, %, +, -`
  - Unary operator (requires one argument) `-`

DIVISION AND REMAINDER

- The result of division (/) with integers is the integer quotient; there is no fractional part
  - the value of `5 / 3` is `1`
- The % operator returns the division remainder.

```
5 * 3
5 / 3
3 / 5
5 % 3
3 % 5
```



INTEGER EXPRESSIONS

- An expression is a sequence of values, operators and parentheses that can be reduced to a single value.

```
int x = 4;

3
x
x + 2
2 * - ( x + 1 )
( 2 * x ) % ( 3 + 2 )
2 + 3 * 4
```


OPERATOR PRECEDENCE

- The order of evaluation of adjacent unlike operators is determined by precedence.
- Java uses the following precedence, from highest to lowest:

```
-   (unary)
/, % , *
+, -
```

- What is the value of: `1 + 3 * -2` ?

OPERATOR ASSOCIATIVITY

- The order of evaluation of adjacent operators with the same precedence is determined by associativity.
- Java evaluates like integer binary operators left to right.
- What is the value of `2 / 2 / 2` ?

PARENTHESES

- Parentheses can be used to override precedence or associativity:
  `( 1 + 2 ) * 3`
- Parentheses can also be used for clarity (or in case you don't know the precedence) :
  `1 + ( 2 * 3 )`

INTEGER VARIABLES AND ASSIGNMENT

```
int total;
total = 3 * 2 + 1;
```

total

7

- In the assignment statement:
  - First, the expression is evaluated to a value.
  - Then, a copy of the value is stored in the variable.
- In general the assignment statement is of the form:

  `<variable> = <expression>;`
- Where the type of the expression is compatible with the type of the variable.

AUGMENTING A VALUE

- Sometimes we want to update the value already stored in a variable; this is called *augmenting* a value.

```
public class Car
{
  private int totalMiles;

  public Car( )
  {
     totalMiles = 0;
  }
  public void addMiles( int miles )
  {
     totalMiles = totalMiles + miles;
  }
}
```

AUGMENTED ASSIGNMENT OPERATIONS

- Java provides augmented assignment operations.
- In general the augmented assignment

```
x = x <op> b;
```

- Can be written

```
x <op>= b;
```

- For example:

```
x = x + 5;
```
can be written
```
x += 5;
```

```
x = x * 3;
```
can be written
```
x *= 3;
```


INCREMENT AND DECREMENT OPERATORS

```
int x;
```

- Java provides an increment operator that adds one to a numeric variable and a decrement operator that subtracts one from a numeric variable.
- `x++` and `++x` each add one to the variable x.
- `x--` and `--x` each subtract one from the variable x.

PREFIX AND POSTFIX INCREMENT

- The value of the postfix increment operator is the value of the variable before it is incremented.

```
x = 0;
y = x++;  // now y = 0 and x = 1
```

- The value of the prefix increment operator is the value of the variable after it is incremented.

```
x = 0;
y = ++x;  // now y = 1 and x = 1
```


PREFIX AND POSTFIX DECREMENT

- The decrement operator also has a prefix and postfix version.
- The value of the postfix decrement operator is the value of the variable before it is decremented.

```
x = 4;
y = x--;  // what is the value of x and y ?
```

- The value of the prefix decrement operator is the value of the variable after it is decremented.

```
x = 4;
y = --x;  //what is the value of x and y?
```

FLOATING POINT NUMBERS

- Integer types are appropriate for modeling numbers with no fractional part: counts, pixels, ...
- Sometimes we need to model numbers with fractional parts: weights, temperatures, ...
- Java provides two primitive types to represent such floating point numbers.
- The difference is the number of bytes of memory allocated to store the numbers.
- The more memory allocated, the greater the range and decimal accuracy of the numbers that can be stored.

JAVA FLOATING POINT TYPES

Type	Size (Bytes)	Largest (farthest from 0)	Smallest (closest to 0)	Precision (decimal digits)
float	4	+/- 3.4 E+38	+/- 1.4 E-45	7
double	8	+/- 1.8 E+308	+/1 4.9 E-324	15

FLOATING POINT LITERALS

- Type double literal values contain a decimal point:
  `12.3, 0.0, 0.000001`, ...
- Type float literal values are written with an appended f:
  `12.3f, 0.0f`
- It is critical to remember that type matters:
  `0.0` is not an int and `0` is not a double.
- The following are "bad type in assignment" errors:

```
int x = 0.0;
float f = 0.0;
```


FLOATING POINT OPERATORS

- Java's floating point operators are same as integer
  - Binary: `*, /, %, +`
  - Unary: `-`
- The division operator for floating point numbers is "real" division.
- The precedence is the same as it is for the integer operators.

TYPE COERCION

- There are times when it would be useful to convert a value of one type to another type.
- This is referred to as type *coercion* (or *casting*).
- A *widening* coercion converts a value to a "larger" type (no information is lost), e.g.

  int to *long* or *int* to *double*
- A *narrowing* coercion converts a value to a "smaller" type (information may be lost), e.g.
  long to *int* or *double* to *int*

IMPLICIT TYPE COERCION

- Under certain situations Java will perform **widening** coercions silently.
- Assignment coercion

```
double x = 0;
```

  - The type *int* 0 is converted to 0.0 of type *double*
- Arithmetic coercion

```
1 / 2.0
```

  - The type *int* 1 is converted to 1.0 of type *double* then floating point division is used, result: 0.5

EXPLICIT TYPE COERCION

- **Narrowing** coercions are **not** done implicitly

```
int x = 0.0;  // compiler error!
```

- But they can be done explicitly:

```
int x = (int) 1.9;
```

- The type *double* 1.9 is truncated to 1 of type *int*
- In general

  (<type name>)

  is the unary coercion operator.

TYPE COERCION WITH CLASS OR INTERFACE TYPES

- Type coercion also applies to class and interface types.
- Conversion of a subclass type to a superclass type (or an implementing type to it's interface type), are done implicitly.

```
RectangularShape s = new Ellipse(  );
```

- Conversion of a superclass type to a subclass type, are not done implicitly but can be done explicitly.

```
RectangularShape s = new Ellipse(  );
Ellipse t = s;                  // compiler error
Ellipse r = (Ellipse) s;        // Compiler allows this, but generates
                                // code to test if s is an Ellipse.
                                // if it isn't, a runtime error results
```

PRECEDENCE SO FAR

Operators in the same box in the table have the same precedence

Operator	Meaning
++ -- - (<type>)	increment decrement unary - type cast
* / %	multiplication division remainder
+ -	addition subtraction
=	assignment

CONSTANTS

- Some magic numbers may change during the execution of your program:
  `private int myXLocation;`
- Others may be expected not to change (all cars have the same width and it does not change):
  `private int bodyWidth;`
- The values that do not change during execution are called constants.

```
public class Car
{
   private int myXlocation;
   private int bodyWidth = 70;
   ...

   public Car()
   {
         myXLocation = 100;
         ....
   }
    ....
}
```


FINAL VARIABLES

- Java provides a way to implement constants as variables that cannot be changed:
  `private final int MY_WIDTH = 70;`
- The keyword final means that this variable cannot be changed.
- The variable must be initialized to a value in the declaration (it's now or never!)
- Style dictates that we use all capitals for constant names and separate words with underscores.

INSTANCE VS. CLASS VARIABLE

- Each instance of Car has a copy of the same MY_WIDTH constant; it might be convenient to have one constant that is shared by all the cars.
- Such a variable is called a class variable. It belongs to the *class* itself, not to the individual objects (the *instances* of the class)

```
public class Car
{
   private int myXlocation;
   private final int MY_WIDTH = 70;
   ...

   public Car()
   {
         myXLocation = 100;
         ....
   }
    ....
}
```

CLASS VARIABLES

```
public class Car
{
   private int myXlocation;
   private static final int MY_WIDTH = 70;
   ...
```

- Java implements class variables with the keyword static.
  `private static final int MY_WIDTH = 70;`
- All instances of Car have their own copy of the instance variable myXLocation but share the class constant MY_WIDTH

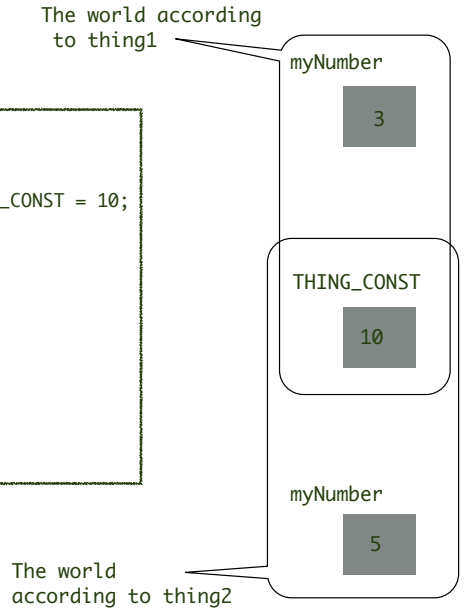
MEMORY

```
public class Thing
{
      private int myNumber;
      private static final int THING_CONST = 10;

      public Thing( int aNumber)
      {
            myNumber = aNUmber;
      }
}

....

Thing thing1 = new Thing(3);
Thing thing2 = new Thing(5);
```



PROGRAM CONSTANTS

```
public class PizzaConstants
{
   public static final double RADIUS = 8.2;
   public static final int  COOKING_TIME = 27;
}
```

- Suppose an entire program composed of interacting classes needs to share some constants.
- Declare the public constants in a Class.
- They can now be used anywhere in the program:
  `x = PizzaConstants.COOKING_TIME + 5;`
- This is a very different use of Class, no objects are created. It is simply a mechanism to hold and organize constants.

AHHA!

- What is `Color.GREEN` ?
- It is a constant named `GREEN` in the class *Color* that has been initialized to a green *Color* instance.

```
public class Color
{
      public static final Color GREEN = new Color( 0, 255, 0 );

      ....
```

INSTANCE VS. CLASS METHODS

- The methods that we have seen so far are meant to model object behavior.
- They are messages sent to objects, instances of a class.
- Just as we can have class variables, we can have class methods that exist independent of instances.
- Java uses the keyword static to declare class methods.

CLASS METHODS

```
public class Area
{
   public static final PI = 3.14;
   public static double circleArea( double radius )
   {
       return radius * radius * PI;
   }
}
```

- Declare the class method using the keyword static.
- It can now be used in the program:

```
x = Area.circleArea(12.5);
```

  - This code sends a message to the class *Area*, rather than to an instance of *Area*
  - *(*In fact, *Area* is an instance of the class, *Class.)*
- There are no objects of type *Area*. The *Area* class is simply a mechanism to hold and organize static methods and constants.

THE JAVA MATH CLASS

- Java provides a standard library class Math.
- It provide the class constants PI and E and many class functions (see the API)

```
area = Math.pow(radius,2.0) * Math.PI;
```


REVIEW

- Numbers and operators:
  - Integers
  - Floating point
- Constants
- Class variables and methods
- The Math class
- Random Numbers

NEXT TIME

- Conditions: boolean expressions
  - Relational operators
  - Boolean operators
- Conditionals: Modeling Decisions
  - if/then
  - if/then/else